

Learn Java While Escaping from Prison Theory and Solved Exercises

Hugo Ribé

About the Author

Before We Dive In, Let Me Introduce Myself. My name is Hugo, and I'll be your Java teacher. A few years ago, I started learning programming on my own—just as a hobby—but I soon realized it was something I truly enjoyed. I believe what best defines me is a passion for self-learning. I'm always striving to acquire new knowledge and collect educational materials wherever I can find them.

Throughout my career, I've had the opportunity to work for two of the world's leading tech multinationals. With no prior experience—and almost without realizing it—I found myself as part of a technical team where data analysis and SQL skills were essential. Shortly after, the department evolved, and I learned to train and configure artificial intelligence models. Within a few months, I was promoted to project manager, and began holding meetings and collaborating with some of the most talented software developers I've ever met.

Immersed in that stimulating environment, I soon became curious about programming. Although Python was more commonly used in my workplace, I decided to go with Java. At first, I didn't even think I'd reach a decent level. However, with persistence and motivation, I gradually reached my goals and ended up mastering this language that has given me so many enjoyable hours in front of the screen.

Of course, this new skill has opened doors for me within the corporate world where I work. Knowing how to code is always a valuable asset professionally, even if you're not currently planning a change or a promotion. You never know what the future holds, and it's always better to be a flexible person with as many skills to offer as possible.

At work, my colleagues have often told me I have a great talent for teaching. That's why I was usually the one in charge of onboarding new team members and guiding them through their first months. Thanks to this feedback, the idea of taking teaching more seriously began to take shape, and later on, the idea of writing this book was born.

Clarifications About This Book and Learning Java

Learning to program is not as hard as you might think. In my case, I remember that even before I started reading the theory, I believed it would be impossible for me to learn programming—at least at an advanced level. Mostly because I thought it would require an extremely high level of commitment and hundreds of hours of effort. To be honest, I was convinced I wouldn't understand a thing and that after a couple of weeks I'd give up on my goal.

I won't lie to you—it's true that learning requires time, both reading theory and doing exercises. But you don't need any prior experience or exceptional computer skills. The only thing you really need is motivation and the right learning materials.

I found several resources online, but none of them fully convinced me. Maybe I was just unlucky. The courses were too expensive, the tutorials incomplete, the forums full of answers to overly specific questions, random videos—and in short, a ton of material that, although often high quality, just didn't work for me.

What I really needed was something structured, so I could learn step by step and feel a sense of progress that would keep me motivated to continue. That's when I thought about buying books. In general, these resources are expensive, and I didn't want to spend a lot of money on something I wasn't even sure I'd enjoy. Luckily, I found several second-hand copies. The problem was that, not knowing what I needed, I ended up with material that was way too advanced or focused on areas of Java like interfaces or databases. I even got a beginner's book where I couldn't understand the very first sentence. Now that I know how to code, I've gone back and reread some of those books, and I can see that the structure and order of topics are terrible.

Still, I didn't give up and kept gathering information from wherever I could. It wasn't an easy path—sometimes I was sure I had learned something, only to discover later that I had misunderstood it. But with persistence, almost anything is achievable. Eventually, I was solving complex exercises and had built a collection of notes that finally explained everything step by step in a structured way.

I'm aware that many people have been, are, and will be in the same situation. So, the idea behind this book is to help all those who, for whatever reason, decide to learn Java programming. From the start, my goal was to create a resource that experienced programmers would look at and say, "I wish I had this book when I was starting out." And I can honestly say—I wish I had.

Before we move on, I'd like to make a few things clear. I don't believe that memorizing concepts is the best way to learn anything—especially not programming. It's true that the upcoming chapters will cover a lot of theoretical concepts that are essential, and yes, you need to understand them—but you don't need to waste time trying to memorize them. As you keep practicing and revisiting the theory, the information will naturally start to stick in your mind, almost effortlessly.

Continuing with this idea, you should understand that what you're trying to learn is a language. What I mean is that, just like in Spanish, there are many ways to express the same idea. The same goes for Java. If you want to write a program that organizes your weekly schedule, there are countless ways to code it.

This chapter includes a bunch of solved exercises. The idea is not to memorize them. I've solved them one way, but maybe you'll come up with a better, more efficient, or more organized solution. Once you've got some experience with programming, you'll realize that the hardest part isn't the syntax—it's figuring out how to use it to solve the problem you're facing.

What I enjoy about programming is that each exercise can be seen as a challenge—or even something fun. Some people do sudoku, others word searches—you can solve Java exercises. It might sound silly now, but soon you'll discover the satisfaction of cracking a problem that's been driving you crazy for hours, days, or even weeks.

Here's a personal tip: don't push yourself to learn everything all at once. When I started out, I tried to spend dozens of hours a week on it, and my brain ended up completely saturated. I was making progress, yes, but there were still many key points I couldn't fully grasp, which was frustrating. Due to work, I had to take a month-long break from Java, and when I came back, I was pleasantly surprised. With a fresh mind, most of the things I hadn't understood became totally clear, and all that mental blockage was gone. So, I recommend taking it step by step.

This book includes a storyline, theoretical content, and practical exercises. I believe the story makes it more enjoyable and different from other books. It also serves as motivation to keep reading. Feel free to skip those pages if you're not interested. On the other hand, if you're already familiar with the theory, you can jump straight into the exercises. I've tried to make this book versatile enough to meet the needs of all kinds of readers. The goal was to compile everything needed into one volume so that anyone—regardless of their prior knowledge—can end up programming with confidence.

Of course, the story is completely fictional and not based on any real people or scenarios. With that clarified, I now wish you the best of luck in your learning journey.

Book Index

Chapter 1: A Carefree Life in Salamanca / Introduction and Data Types

- Introduction and presentation of data types
- Definition of OOP and its fundamental concepts
- Declaration and assignment of variables

Chapter 2: Chani's Disappearance / Displaying Data on the Console

- Importance of data output
- Using `System.out.println` and `System.out.print`
- String concatenation
- Changing font color
- Comments in code
- Rounding numbers, Part I

Chapter 3: Rich's Friend / Keyboard Input

- Importance of data input
- Using the `Scanner` class
- Concept of packages in Java
- Declaration and usage of `Scanner`
- Difference between assignment and declaration of variables

Chapter 4: Killing Time at the Police Station / Working with Conditionals

- Importance of conditionals
- Basic structure of `if`
- `else` and `else if` statements
- Nested conditionals
- `switch-case`
- Comparison operators
- Logical operators

Chapter 5: Meeting Bud. Learning to Use Loops

- Definition and importance of loops
- for loop
- while loop
- do-while loop
- break keyword
- continue keyword
- String manipulation: charAt() and length()

Chapter 6: Helping Bud with His Tasks / Random Numbers

- Importance of random numbers
- Generating random numbers with Math.random
- Rounding numbers, Part II – Using Math.round()

Chapter 7: Chani's Cousin Visits / Arrays

- Definition and importance of arrays
- Declaration and creation of arrays
- Assigning values to arrays
- Traversing arrays with loops

Chapter 8: Bud's Job / 2D Arrays

- Definition and importance of 2D arrays
- Declaration, creation, and assignment of 2D arrays
- Accessing elements in a 2D array
- Initializing 2D arrays
- Traversing 2D arrays with nested loops
- Summing elements in a matrix
- Searching for specific elements in a matrix

Chapter 9: Creating a Distraction Device

- Using our knowledge to move forward in the story

Chapter 10: The Path to Vicente's Office / Working with Date and Time

- LocalTime, LocalDate, and LocalDateTime classes
- Getting the current date, time, and date-time
- Creating specific dates and times
- Adding and subtracting days, hours
- Comparing dates
- Formatting dates
- Calculating the difference between dates

Chapter 11: Hacking the Security System

- Using our knowledge to move forward in the story

Chapter 12: Planning the Return to Prison / Try & Catch

- Importance of try and catch blocks
- Common exception types
- Basic structure of try-catch
- Multiple catch blocks
- The finally block

Chapter 13: Gathering Evidence Against Phil

- Using our knowledge to move forward in the story

Chapter 14: Escaping Through the Front Door / Reading and Writing Text Files

- Importance of reading and writing files
- Importing classes to read files
- Reading text files
- Importing classes to write files
- Writing to text files

Chapter 15: Outsmarting the Timer System

- Using our knowledge to move forward in the story

Chapter 16: Final Preparations / for-each

- Importance of for-each loops
- Advantages and limitations of for-each
- Syntax of the for-each loop

Chapter 17: Decoding Communication Systems

- Using our knowledge to move forward in the story

Chapter 18: Intercepting the Guards' Conversations

- Using our knowledge to move forward in the story

Chapter 19: Functions and OOP

- Difference between methods and functions
- Access modifiers: public, protected, private
- Methods that return values vs void
- Creating and calling methods in the Main class
- Creating and calling methods in other classes
- Difference between class and instance
- Difference between static and dynamic methods
- Creating objects and using constructors
- Defining getter and setter methods

Chapter 20: The Final Phase – We Build a Door Unlocking Tool

- Using our knowledge to move forward in the story

Chapter 21: The Ending

- Ending and farewell

Chapter 1 introduction.

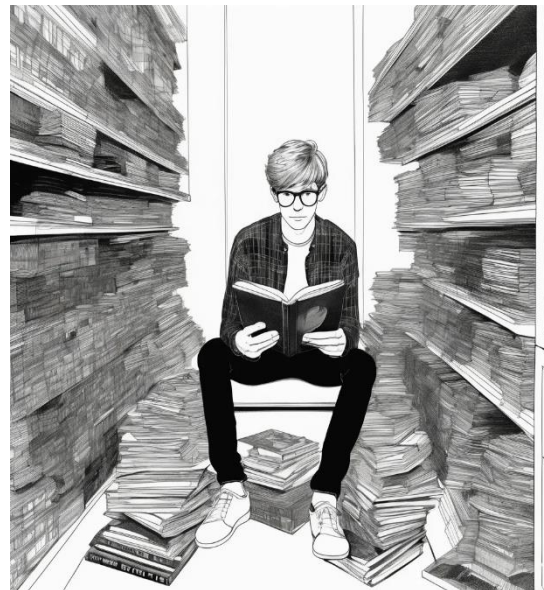
A Carefree Life in Salamanca / Introduction and Data Types

All my life, I've felt untouchable. I never had any major worries and always had the support of friends and family. I wasn't aware that, at any moment, my luck could change drastically. Most people think they have everything under control, but there are things that are simply beyond us. And it's precisely those things that can catch you off guard at any time. That's exactly what happened to me.

I've heard you want to learn how to program in Java. Don't worry — I'm going to teach you, and I'll make it simple. Honestly, programming isn't hard — at least not as hard as it seems. But before we begin, let me introduce myself.

My real name is Pelayo, although everyone calls me Peli. I'm a first-year university student. I live in Salamanca, and I'd like to work in computer science someday. For now, things aren't going quite as I had planned. The truth is, we're already in the second half of the semester, and I've barely shown up to class. Many of the professors don't even know who I am.

To be honest, I've spent most of my time having fun and meeting new people. I know I probably should start studying as soon as possible, but that's just not my priority right now.



This weekend, I'm going to throw a party in my apartment and I want it to be perfect. I don't like to brag, but I think that, little by little, gatherings at my place are becoming really popular. Actually, I don't live alone; so, it's not exactly my apartment. I share the house with two other university students, Rich and Chani. The truth is, they are better students than me, although you can't really say they're very responsible either. We didn't know each other before, but in a short time, we've become good friends.



Rich is an English exchange student, although he speaks perfect Spanish. He is always very well dressed and spends many hours in front of the mirror. That doesn't mean he is arrogant. On the contrary, he is a very humble and charismatic person.

He gets along well with everyone and is always the center of attention. He likes that. Over the past few months, we have spent a lot of time together, and he has become my best friend.

Actually, he was the first person I met here, or at least the first one I started to become friends with. In my first class, I had to sit next to him. I don't remember why we started talking, but from the very first minute, it felt like I had known him all my life.

That same day, he told me he didn't want to live in the student dormitory. I still didn't have a place to live, so I suggested we share a house. He accepted immediately. The problem was that the two of us couldn't afford to pay the rent for an apartment. We needed a third person. We posted an ad on the university's online bulletin board, and an Asian young man contacted us within a few hours. His name was Chani, and he became our new friend.

Chani is also an exchange student, specifically from South Korea. He is making an effort to learn our language, but it's not easy for him. Sometimes, I think it would be impossible for me to learn Korean.

He's quite introverted, but that doesn't stop him from attending all the parties. He spends almost the entire day locked in his room, either sleeping or playing on his computer. He only leaves his room to cook and go to the bathroom. He seems like a vampire. As soon as night falls, he decides it's time to get active and hang out with us.



We're an unconventional group of friends, but it works well for us. Living together is really good. We never argue and we respect each other's space a lot. A couple of times a week, we order pizza and have dinner together in our large living room. During the week, we hardly ever have guests over, but we don't feel lonely because we have each other. I couldn't have been luckier choosing my roommates.

We love the common area of our house. It's very big and spacious. It has huge windows and access to the terrace. We got some very cheap sofas and even a small foosball table that visitors enjoy playing. I had never lived in a better house than this one, and the price is much cheaper than it seems.



I spend almost my entire day lying here, often in Rich's company. He knows I'm barely attending university classes and worries that I won't pass any of my exams. So, he has offered to teach me how to program in Java. He knows quite a bit about it and explains things in a simple and clear way.

But well, I won't go on any longer. Now that the introductions are done, I think it's a good time to review Rich's lessons. So, for the rest of this chapter, I'll focus on revising and reinforcing some concepts. It will be very basic, just a small introduction to the world of Java and its most commonly used data types.

Introduction. Definition of OOP and presentation of data types.

The first thing that is always mentioned when teaching Java is that it is a type of object-oriented programming. It doesn't matter where you get the information from or who is explaining it to you, you will always come across phrases like these:

"Object-oriented programming (OOP) is a programming paradigm based on the concept of 'objects.' These objects are entities that combine data and functions (called methods) related to them into a single unit. The main idea behind object-oriented programming is to model the real world in software, which allows for a more accurate and efficient representation of real-world elements and processes in our programs."

If you didn't understand anything, don't worry. If you did, congratulations—you are one step ahead of most beginners. When you program in Java, as in many other languages, the code is not written in a linear way. That would be impractical. You have to keep in mind that certain complex programs or applications have tens or even hundreds of thousands of lines of code.

It is much more practical to divide that code into small groups that interact with each other. This will help us, for example, to identify and fix errors in the code much faster, to reuse already written code and avoid duplication, to easily divide tasks among several people, to maintain code effectively and in an organized way, to implement new features in the application, and ultimately, to make our lives easier.

Pay attention now, because I'm going to pose a possible exam question: tell me the four fundamental concepts on which object-oriented programming is based.

OOP is based on four fundamental concepts:

Classes: *A class is like a blueprint or template that defines the common characteristics and behaviors of a particular type of object. It defines the data that an object can hold (called attributes) and the operations it can perform (called methods). For example, if we think of a "car" object, the corresponding class might define attributes such as color, model, and speed, and methods like accelerate and brake.*

Objects: *Objects are concrete instances of a class. Each object has its own copy of the data defined in the class but shares the methods defined within it. For example, if we have a class "Car," a specific object could be a red Toyota Corolla model car with a current speed of 60 km/h.*

Encapsulation: *Encapsulation is the concept of hiding the internal details of an object and exposing only the necessary functionality through well-defined interfaces. This is achieved by defining the class attributes as private and providing public methods (getters and setters) to access and modify these attributes in a controlled manner. Encapsulation helps keep the code modular, facilitates maintenance, and reduces the risk of errors.*

Inheritance: *Inheritance is a mechanism that allows a class (called a subclass or derived class) to inherit the attributes and methods of another class (called a superclass or base class). This allows for code reuse and the creation of class hierarchies, where more specific classes can add or modify inherited behavior. For example, a "Vehicle" class can be a superclass of the "Car" and "Truck" classes, which inherit its basic characteristics but may have additional specific behaviors.*

You don't need to memorize these terms. We will work with them in the upcoming topics, and little by little, they will become clearer in your mind. However, it's good for you to get familiar with them and know they exist. For now, just keep in mind the idea that Java is an object-oriented language.

Programming is a somewhat unintuitive activity that uses many unfamiliar terms. Gradually, everything will start to make sense, but remember that you need to be consistent and not give up.

Data Types and Variables:

I don't know if you like cooking, but I'm sure you've tried to prepare a recipe at home at some point. If we compare programming to cooking, we could say that data types are the ingredients, the raw materials we use to create our program.

Variables, on the other hand, are the places where you store those ingredients to use later in your recipe, like jars or bowls that hold flour, sugar, etc.

- Data Types

In Java, data types are classified into two main groups: primitive data types and reference data types.

1) **Primitive data types:** These are the basic types built into the language. There are eight in total:

- **byte:** An 8-bit integer type that stores values from -128 to 127.
- **short:** A 16-bit integer that can handle values from -32,768 to 32,767.
- **int:** A 32-bit integer covering values from -2^{31} to $2^{31} - 1$.
- **long:** A 64-bit integer that represents values from -2^{63} to $2^{63} - 1$.
- **float:** A 32-bit floating-point number, with a precision of approximately 6-7 decimal digits.
- **double:** A 64-bit floating-point number, with a precision of around 15 decimal digits.
- **char:** Represents a single 16-bit Unicode character.
- **boolean:** Can only have two possible values: true or false.

2) **Reference data types:** Instead of directly containing data, these types store references to objects in memory. Some of the most common ones include:

- **String:** A sequence of characters used to represent text. Unlike many languages, in Java it is not a primitive type but a reference type.
- **Arrays:** Structures that store multiple elements of the same type contiguously in memory.

- Variables

In Java, a **variable is a container that stores data** which can change during the execution of a program. I repeat: the data inside a variable can change as the program runs. Variables have a name used to refer to them and a data type that specifies what kind of values they can hold.

For example: `int variable = 7;`

In the previous variable, we can see that the data type is `int` and its name is "variable." Also, we see that right now its value is 7. That value may vary during the program's execution, but for now, it is 7.

When we declare a variable, we must do so depending on the type of data we need. Variables are constrained by the type of data they hold. A numeric variable cannot store text data. Therefore, we can refer to them with the same classification we give to data types. This causes that, many times, both terms are used interchangeably, which can be confusing in the end.

It's easier to explain with an example. Imagine we need to create a variable called "name" (name). That variable will hold a text data type (`String`), specifically the name Pedro. So here it is:

```
String name = "Pedro";
```

That variable is now a "String" type variable since it holds text data.

There are quite a few data types, and it's good that you know them all. However, for now, and for the exercises we'll be doing, you'll only need to know a few. As I mentioned in the previous paragraph, data types are linked to variables. Therefore, to avoid confusion, from now on we will only use the term variable.

Now, let's see a slightly more detailed definition of the variables you need to know:

Integer variables (int): Integer variables are used to store whole numbers. For example, we can use an integer variable to represent a person's age or the quantity of products in inventory. In Java, we declare an integer variable using the keyword `int`, followed by the name of the variable and, optionally, its initial value.

For example:

```
int age = 25;
```

Text variable (String): It is used to store sequences of characters, i.e., text. To define a String variable, we simply use the keyword String, followed by the name of the variable and, optionally, assign it a value. Here's an example of how to define a String variable in Java:

```
String myName = "Juan";
```

Floating-point variables (float and double): These variables are used to store decimal numbers. The difference between float and double lies in the precision of the decimal number they can store. In Java, we use float for single-precision numbers and double for double-precision numbers. For example:

```
float height = 1.75f;  
double pi = 3.14159;
```

Boolean variables (boolean): A boolean variable can only take one of two values: true or false. These variables are useful for representing logical conditions. For example:

```
boolean isOfLegalAge = true;
```

Character variables (char): Character variables are used to store a single character. For example, we can use a character variable to represent a letter or symbol. In Java, we declare a character variable using the keyword char. For example:

```
char initial = 'A';
```

The advantages of specifying the data type that a variable can hold are the following:

1. **Make our code simpler and easier to understand:** If we didn't specify the type of data a variable can hold, the code would be nearly impossible to understand—for you and for other programmers. Knowing what type of data, a variable expects helps us understand how it will be used in the program.

2. **Error detection:** If you try to assign a value of the wrong type to a variable, you'll be warned about the error. This helps catch mistakes even before the program runs, making programming easier and preventing runtime errors.
3. **Better performance:** In some cases, using typed variables can improve the program's performance, since the code can be optimized by knowing the data types being used.
4. **Easier code maintenance:** When you revisit or modify your code in the future, having specified data types can help you remember how certain variables are supposed to be used. If someone else reviews the code, this becomes even more helpful. This makes maintaining and updating the code easier as the project evolves.

Difference between variable assignment and declaration.

Now that we already know the different data types, we need to understand how to implement them in our code. We've already seen some examples in the previous paragraphs:

```
String myName = "John";  
int age = 25;  
double pi = 3.14159;  
boolean isAdult = true;  
char initial = 'A';
```

As you can see, we always start by writing the data type we need—for example, String, int, double, boolean, or char. Then comes the name we've chosen for the variable, followed by an equal's sign. After that, we assign the value, and we always end the statement with a semicolon.

Note that for the String data type, the assigned text must always be enclosed in double quotation marks: "John". In the case of a char, the character we want to store in our variable must be enclosed in single quotation marks: 'A'.

However, it's not always necessary to assign a value to our variable. We can simply declare it and use it later. For example, we start an exercise that asks us to create a String variable called name and another int variable called enrollment. As you can see, it's not necessary to assign them a value right away.

```
String name;  
int enrollment;
```

It may seem like this doesn't make sense right now. Why not assign a value as soon as you create a variable? It's basically for reasons of order and structure. Once you get more comfortable with the exercises, you'll see that everything has a reason.

You might be wondering how to assign a value to a variable that has already been declared. Let's continue with the two examples we saw a couple of paragraphs above (name and enrollment). We simply do it like this:

```
name = "John";  
enrollment = 55;
```

The only thing we must keep in mind is the type of data we want to use (String and int in the example). Then, we simply follow the usual structure we've already seen: variable name, equals sign, and the value we want to assign.

Naming convention for variables in Java

In Java, variables follow certain naming convention rules. Here are the main ones explained:

Start with a lowercase letter: The name of a variable must begin with a lowercase letter. For example: age, height, width, etc.

Use camelCase: This means that if the variable name consists of more than one word, the first letter of each word after the first is capitalized, with no spaces or hyphens. For example: currentAge, fullHeight, totalWidth, etc.

Meaningful and descriptive: Try to use names that indicate what is being represented. For example, instead of `x` or `y`, you could use `age` or `accountBalance`.

Avoid special characters: Only letters, numbers, and the underscore `_` are allowed. Other characters such as spaces, hyphens (`-`), periods (`.`), and so on should not be used in variable names.

Don't use Java keywords: Avoid using reserved words in Java (such as `int`, `float`, `while`, etc.) as variable names—doing so may cause errors in your code.

Don't use accents or special characters: Java does not allow the use of accents or special characters in variable names.

By following these rules, your programs will be easier to read and understand for both you and others who read your code.

Summary of What You've Learned

In this first chapter, you've begun your journey into the world of Java programming. Congratulations! Not everyone dares to take on this challenge. Now we can say that you have a general idea of how Java is structured and what the term "object-oriented programming" (OOP) means.

You also know the four fundamental pillars of OOP: classes, objects, encapsulation, and inheritance. If any of these concepts still aren't clear to you, don't worry—it's completely normal. I don't think many people fully understand everything on the first try. Realistically, we haven't gone too deep into the topic yet, and there are still many questions left to answer. But from this point on, learning new concepts will be much easier for you.

By now, you're probably more comfortable talking about data types and variables. These are more concrete concepts and easier to grasp. Once we start with the practical part, they will be essential in every one of our exercises. We know that there are primitive and reference data types. Additionally, we've emphasized the importance of specifying data types in Java. Doing so improves clarity, helps detect errors, and makes the code easier to maintain.

We've also taken the time to learn the difference between assigning a value to a variable and simply declaring one.

Even though it's not a top priority in your learning right now, you've had a glimpse at the naming conventions for variables in Java. Soon you'll be working with variables and creating your own code, where you'll have complete freedom to name your variables. My advice is to try to follow these conventions so you get used to applying good practices.

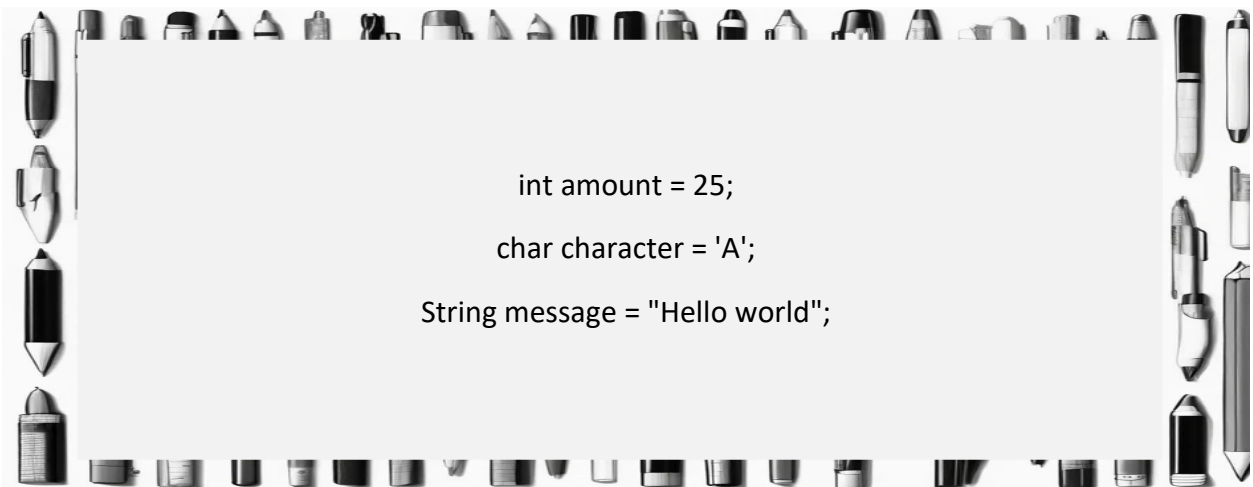
In summary, you've taken your first step into the world of programming. There's still a long way to go. I understand that theory can sometimes feel a bit boring, but it's absolutely necessary. This introduction will help you build a solid foundation for understanding the more advanced concepts that we'll cover in the following chapters.

I'm sure you're already excited to get your hands dirty and start writing your own programs. To be honest, that's my favorite part too. So don't worry—soon we'll start solving exercises and applying the theoretical knowledge we've been building up.

Practice Exercises

Exercise 1. Declare three variables of different types: one variable of type `int`, another of type `String`, and a third of type `char`. Directly assign the value 25 to the `int` variable, "Hello world" to the `String` variable, and the character 'A' to the `char` variable.

Solution:



```
int amount = 25;
char character = 'A';
String message = "Hello world";
```



Use descriptive names: Choose variable names that are clear and meaningful. A good variable name makes the code easier to understand.

Exercise 2. Declare three variables of different types: one variable of type `int`, another of type `String`, and a third of type `char`. Then, in different lines of code, assign the following values to each: the value 25 to the `int` variable, the value "Hello world" to the `String` variable, and the character 'A' to the `char` variable.

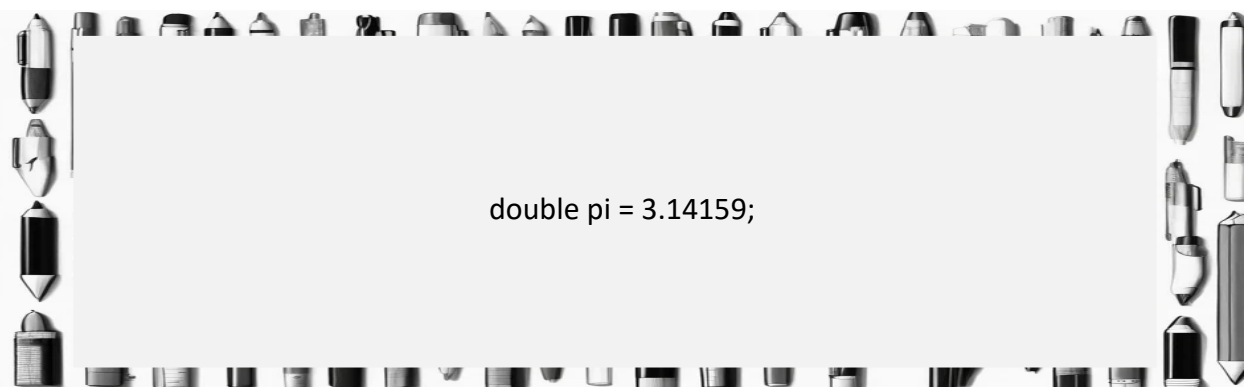
Solution:



```
int amount;  
char character;  
String message;  
amount = 25;  
character = 'A';  
message = "Hello world";
```

Exercise 3. Declare and assign a value to a double variable that represents the number π (pi).

Solution:




```
double pi = 3.14159;
```

Exercise 4. In a mysterious world of adventures, you are exploring an ancient temple full of traps. You must create a Java program that declares a boolean variable named `activated` and assigns it the value `true`, indicating that you have activated the temple's security mechanism.

Solution:

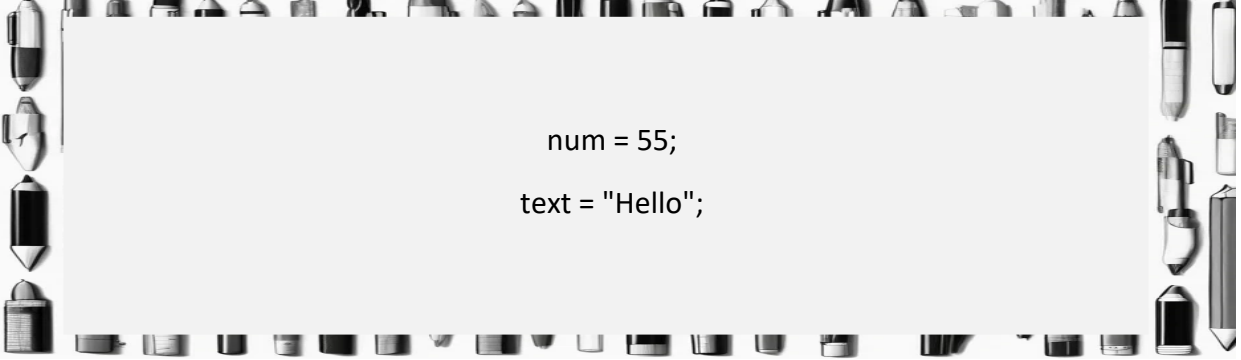


 **Initialize your variables:** In your first programs, initialize your variables when you declare them. This prevents errors and unexpected behavior in your program.

Exercise 5. In the following lines of code, two variables are declared: `num` and `text`. Assign them the values 55 and "Hello", respectively.

```
public class Main {  
    public static void main(String[] args) {  
        int num;  
        String text;  
    }  
}
```

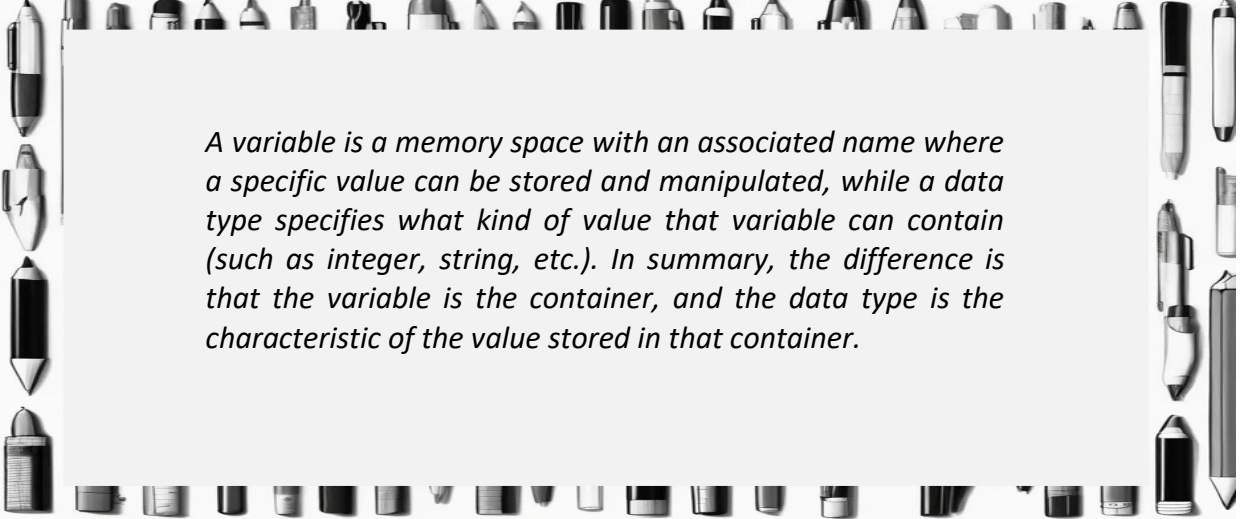
Solution:



```
num = 55;  
text = "Hello";
```

Exercise 6. Tell me the difference between a variable and a data type.

Solution:

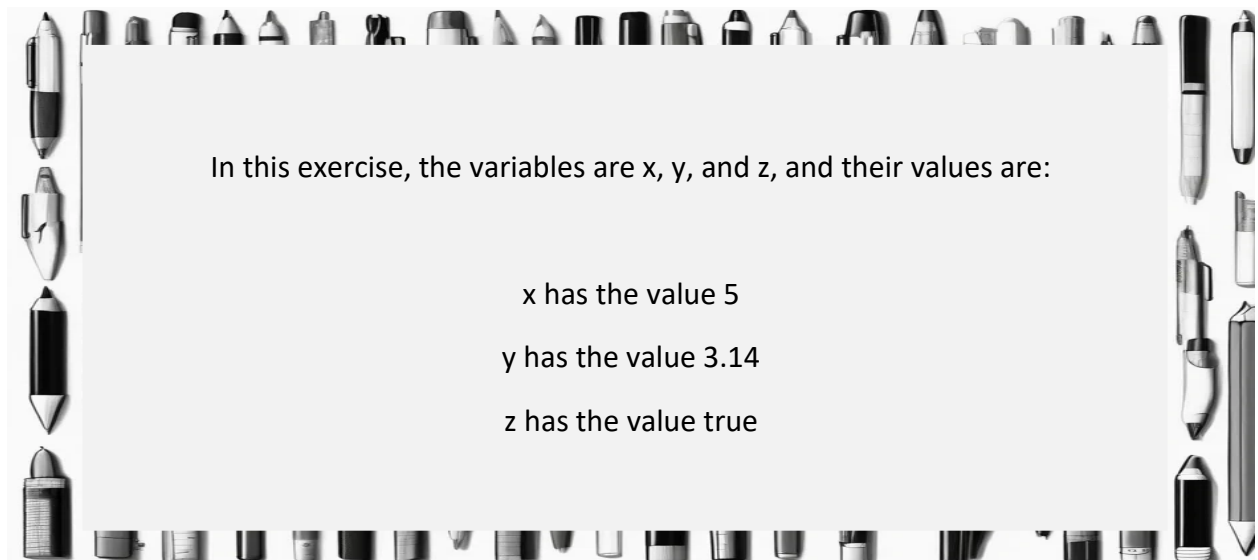


A variable is a memory space with an associated name where a specific value can be stored and manipulated, while a data type specifies what kind of value that variable can contain (such as integer, string, etc.). In summary, the difference is that the variable is the container, and the data type is the characteristic of the value stored in that container.

Exercise 7. Look at this code and identify the variables and their respective values.

```
public class Main {  
    public static void main(String[] args) {  
        int x = 5;  
        double y = 3.14;  
        boolean z = true;  
  
        System.out.println("The value of x is: " + x);  
        System.out.println("The value of y is: " + y);  
        System.out.println("The value of z is: " + z);  
    }  
}
```

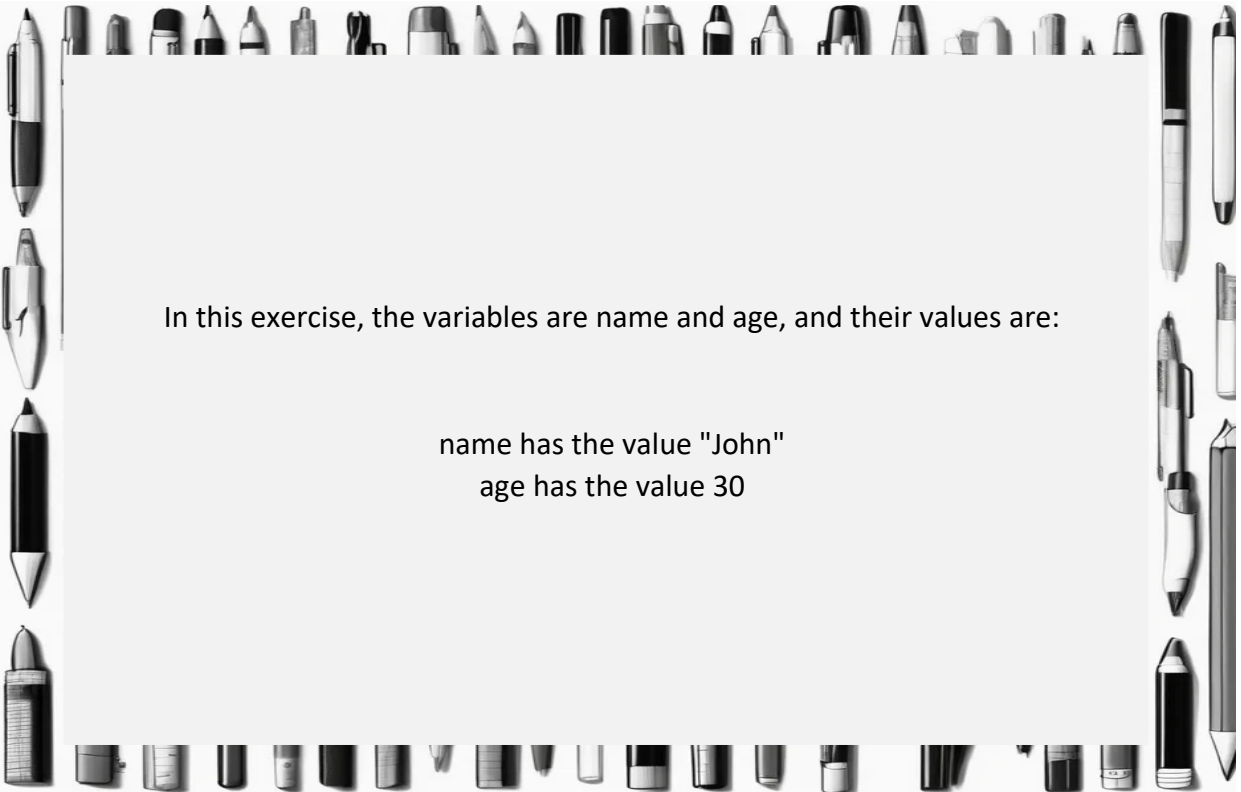
Solution:



Exercise 8. Look at this code and identify the variables and their respective values.

Solution:

```
public class Main {  
    public static void main(String[] args) {  
        String name = "John";  
        int age = 30;  
  
        System.out.println("The name is: " + name);  
        System.out.println("The age is: " + age);  
    }  
}
```



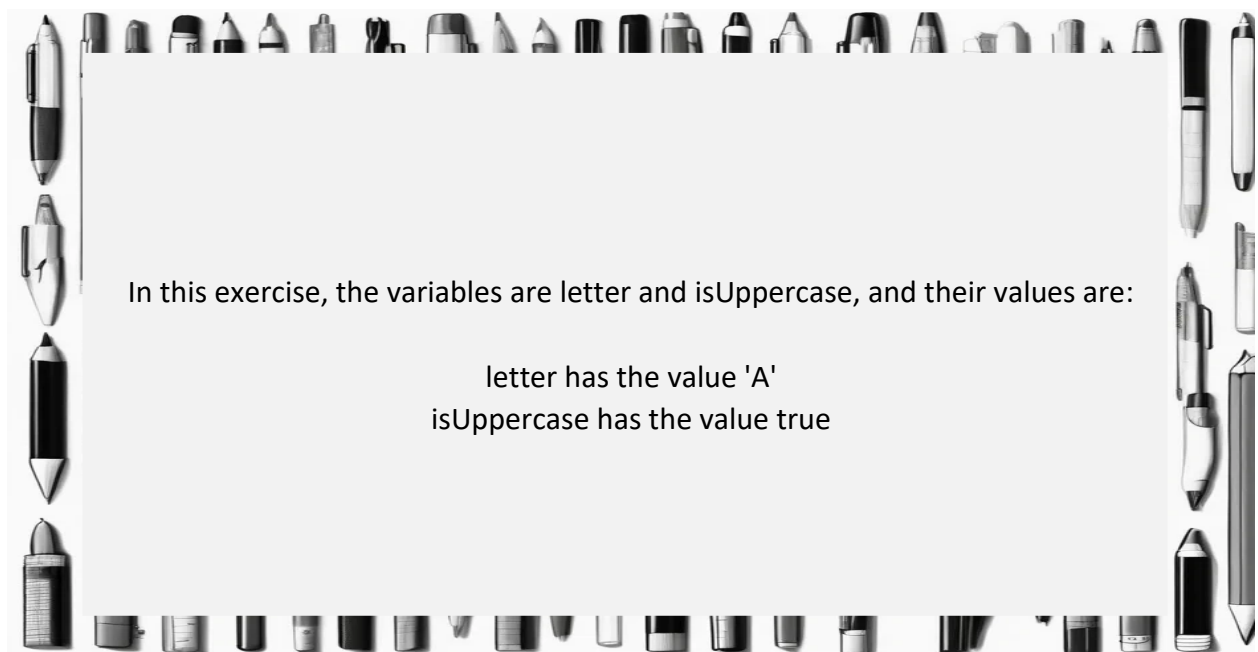
In this exercise, the variables are name and age, and their values are:

name has the value "John"
age has the value 30

Exercise 9. Look at this code and identify the variables and their respective values.

```
public class Main {  
    public static void main(String[] args) {  
        char letter = 'A';  
        boolean isUppercase = true;  
  
        System.out.println("The letter is: " + letter);  
        System.out.println("Is it uppercase?: " + isUppercase);  
    }  
}
```

Solution:



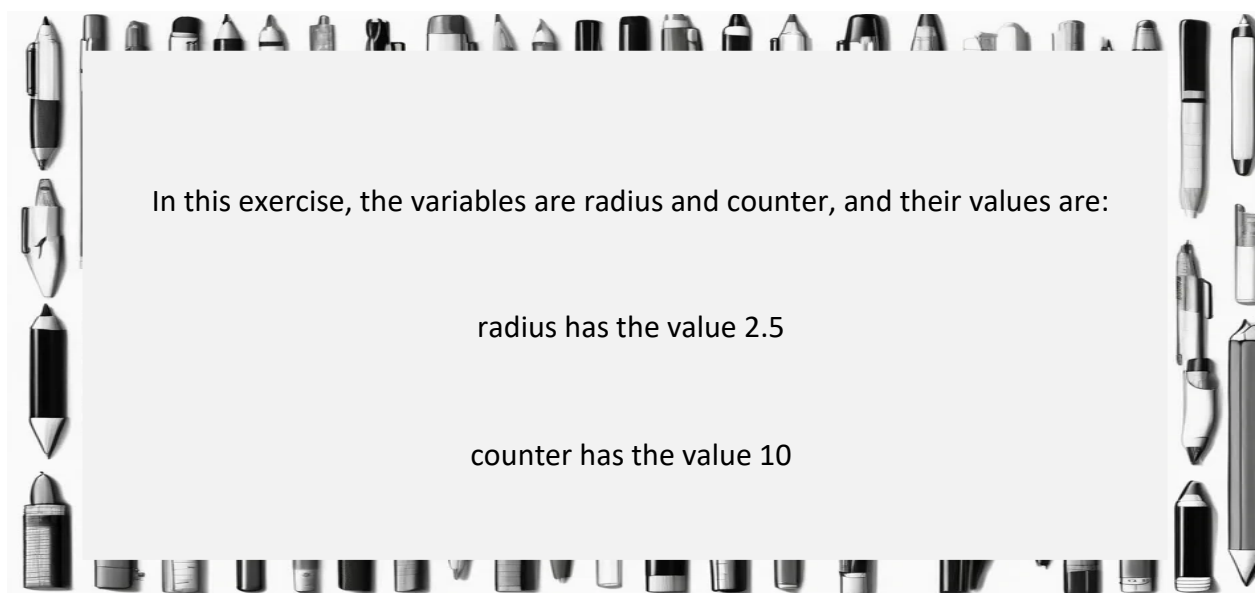


Follow the naming convention: Use the camelCase convention to name variables in Java (for example, myVariable). This improves readability and aligns with common practices in Java.

Exercise 10. Look at this code and identify the variables and their respective values.

```
public class Exercise10 {  
    public static void main(String[] args) {  
        double radius = 2.5;  
        int counter = 10;  
  
        System.out.println("The radius is: " + radius);  
        System.out.println("The counter is: " + counter);  
    }  
}
```

Solution:



Chapter 2.

The disappearance of Chani / Display data in console

Do you remember when I said I was going to have a party at my house? Well, it was a huge success. I'm not sure how many people were there, but more and more people kept showing up. I didn't know half of them, but that didn't worry me. It was all about having a good time.

Before bringing guests home, we like to be cautious, so we have a routine. Hours before the attendees start arriving, we store all our valuables in a closet located in Chani's room. The closet is secured with a padlock, and only Chani has the key. This way, we feel safer in case things get out of control. The next day, we wake Chani up, she opens the closet, and we retrieve our precious belongings. Then we eat something and spend almost the entire afternoon lying on the couch watching some TV series. It's a big part of our student routine.



The problem is that this time everything was different. The morning after the party, everything was chaos. It was much messier than usual, and there were drink spills everywhere. Not to mention the incredible number of bottles and glasses scattered throughout all the rooms.

I woke up around two in the afternoon, quite hungry. I went to the kitchen and opened the fridge. I discovered that, unfortunately, there was nothing to eat, and that greatly affected my mood. Then I started to consider my options. Going out to buy food was completely out of the question. I was very tired and didn't feel like moving. Ordering food seemed like the most sensible option, but I wasn't exactly sure what I wanted.

While deciding, I lay down on the couch and texted my roommates. I didn't want to eat alone, and besides, I thought they would probably be hungry when they woke up too. After a few minutes, Rich came out of his room. He had read the message and wanted pizza. Chani didn't respond, but we ordered for him anyway.



The pizzas finally arrived around three-thirty in the afternoon. Chani showed no signs of life, so we didn't wake him up. We ate our pizza and chatted for a while. Although we had nothing to do, we didn't feel like starting to clean up.

We didn't feel like sleeping more either. We were starting to get a bit bored and wanted our laptops. The only problem was that they were in Chani's closet. We didn't wait much longer before deciding to wake him up. We knocked on his door several times but got no response. After a few minutes, we decided to go in.

As we crossed the doorway, we looked at each other in surprise. Chani wasn't there, but the strangest thing was that his room was completely tidy. It was very odd because it contrasted with the rest of the house, which looked like a pigsty. We were very confused, so we tried calling him, but got no answer.

We didn't give it much more thought. We assumed he was fine; the only annoyance was that we didn't have our computers and didn't know what to do. We sat on the couch and, after a few minutes, Rich started talking. He asked me about my Java learning. I told him I hadn't made any progress since the last lesson, so he offered to continue with the classes right then and there.

Display data in our console

Before moving forward, you need to understand the stages involved in developing a Java program or application. From when you start writing the code until the completion of the project.

- 1) **Writing the source code:** The process begins when the programmer writes the code in Java language. This is done using a text editor or an Integrated Development Environment (IDE) such as IntelliJ IDEA or Eclipse. The source code follows a specific structure and 'grammar'.
- 2) **Saving the source file:** When the code is ready, it is saved in a file with the '.java' extension. This file includes all the necessary instructions for the program to function correctly.
- 3) **Code compilation:** The next step is to compile the source code using a Java compiler. This takes the '.java' file and converts it into code called 'bytecode'. This file is executable on any system that has the Java Virtual Machine (JVM) installed.
- 4) **Generation of .class files:** After compilation, one or more files with the '.class' extension are generated, containing the bytecode. These files will be the ones executed by the JVM.
- 5) **Program execution:** Finally, with the generated bytecode files, the program can be executed using the JVM. The JVM interprets the bytecode and carries out the operations defined in the code that was manually written by the programmer."

At this point, you don't need to memorize these steps. Our IDE will take care of performing all these steps except the first one - writing the code. But it's interesting to know the complete process. Additionally, this has helped us introduce the following two terms:

IDE, or Integrated Development Environment: IDE stands for Integrated Development Environment. This is a software tool that contains everything needed for software application development. It can be described as a 'command center' for programmers, containing various useful tools and features in one place.

Java Compiler: This is a tool that translates human-readable source code (the code we have written) into a machine-understandable format. This allows Java code to be portable and run on different platforms without requiring modifications. Without the compiler, we couldn't transform our ideas into functional applications.

Displaying text and variables in the console

Data Output Importance

In the general programming process, data output is absolutely necessary as it is the main way in which the program can communicate results and its updated status to us. Without visual output, we wouldn't be able to know what's happening, whether the program is following our instructions exactly, or if something isn't working as it should. Most of the time, programmers work based on trial and error, which is why constantly seeing the results is vital. For a novice programmer, it's even more important since seeing the result of the lines we write will help us learn and understand what happens when we write our syntax.

Common Console Output Uses

1. User Interaction

The console allows interaction with the user by requesting information and displaying results. This interaction is fundamental in basic programs and is an excellent way to teach input and output concepts.

2. Displaying Calculation Results

Presenting the results of mathematical operations, algorithms, or logical processes is one of the basic functions of any program.

3. Feedback in Long Processes

During the execution of time-consuming processes, providing feedback in the console helps the user understand that the program is still running.

4. Code Debugging

Printing variable values and states helps programmers understand how the program is working and identify possible errors.

5. Demonstrations and Educational Examples

Examples that print results to the console are essential for teaching basic programming concepts in a visible and tangible way.

Using `System.out.println` and `System.out.print`

We're going to see how to use a line of code that allows us to display data in our IDE's console. For this, we can use two commands: `println` and `print`.

If you already have some programming experience or have looked for educational material elsewhere, you've probably encountered the most famous programming exercise of all time. It consists of displaying the phrase 'hello world' in our console. The answer is:

```
System.out.println("Hello world");
```

It always follows the same structure, `System.out.println()`;

In this specific case, we use double quotes since it's a text line. We can also display variables that we have previously declared and assigned a value to. Let's look at an example. Below you will see an `int` variable called `total` and we've already assigned it a value, in this case 9. We can print it to the screen as follows:

```
int total = 9;  
System.out.println(total);
```

As mentioned earlier, we can use both `print` and `println`, but there is a significant difference between them:

- **`System.out.println`** adds a new line after printing what you specify. It's like adding a line break after displaying something.
- **`System.out.print`** simply prints what you specify without adding a line break at the end. It's like writing in a continuous line.

String Concatenation

We can display variable operations and even combine them with text. Imagine that we have declared two `int` variables 'a' and 'b'. We have assigned them the values 5 and 7 respectively.

```
int a = 5;  
int b = 7;
```

Then we ask to display the following text: 'The result of multiplying a and b is ='. Finally, the result of that multiplication.

```
System.out.println("The result of multiplying a and b is = " + a * b);
```

As you can see, after showing the text we have used the `+` symbol to join it. ('text ' + result).

We could have done it in the following way, which might be easier to understand:

```
System.out.println("The result of multiplying a and b is = " + (a * b));
```

Right after the `=` symbol, we left a space so that, when displayed in the console, there's a gap. Although we could have done it the following way.

```
System.out.println("The result of multiplying a and b is =" + " " + (a * b));
```

I have used + ' ' + to create a space.

When combining text and numeric data it's not very useful, but if we want to display two numeric variables and need them to be separated, we'll use this trick.

Let's look at another example, but this time with additions.

If we want to add variables a and b from the previous exercise, we'll simply do it this way:

```
System.out.println(a+b);
```

But we might just need both of them to be displayed separately in the console, one after the other. In that case, we'll do it like this:

```
System.out.println(a+ ' ' +b);
```

Of course, we can also use subtraction and division symbols (- and /) depending on what we need.

Changing the Font Color

It's possible that at some point we may need the color appearing in our console to be different from the default white. We can achieve this very easily. We simply need to insert a color code into our line from those shown in the following table:

```
BLACK => "\033[30m",  
RED   => "\033[31m",  
GREEN => "\033[32m",  
YELLOW => "\033[33m",  
BLUE  => "\033[34m",  
PURPLE => "\033[35m",  
CYAN  => "\033[36m",  
WHITE => "\033[37m",
```

We'll do it this way:

```
System.out.println("\033[31m" + "The result of multiplying a and b is =" + " " + (a * b));
```

We insert the code right before the part that we need to change color. From that point until the end of the line, it will be displayed in the new color. If we need two or more colors, we'll insert the corresponding codes right before the parts that should be shown in that specific shade.

```
System.out.println("\033[31m" + "The result of multiplying a and b is =" + " " + "\033[34m" + (a * b));
```

Comments in our code that don't print

The double forward slashes `/**` are used in Java to indicate a single-line comment. This means that any text after the double slashes in that line will be ignored by the Java compiler and won't affect the program's execution.

```
// This is a single-line comment
```

```
int age = 25; // This line declares a variable called age and assigns it the value 25
```

In this case, comments are used to make annotations in the code that help understand what each part of the program does. They are also useful for temporarily disabling a line of code without having to completely delete it.

Limiting data output to two decimal places

For now, you won't need to limit the number of decimal places in any exercise. In fact, **if you're new to Java, you should skip this section.** The reason I'm including it here is because some people might find it useful and don't want their operations to show a long string of decimal places.

However, even if it's not useful to you right now, remember that you have this information here and can come back to it when you need it. On the other hand, if you feel confident enough, you can try to limit the decimal places in your exercises.

Methods for limiting decimals

- In Java, you can use the `DecimalFormat` class to easily limit the number of decimal places to two. Here's an example:

```
import java.text.DecimalFormat;
```

```
public class Main {  
    public static void main(String[] args) {  
        double num = 3.14159;  
        DecimalFormat df = new DecimalFormat("#.00");  
  
        System.out.println("Number with two decimals: " + df.format(num));  
        // Output: 3.14  
    }  
}
```


Another way to limit decimals to two places in Java is by using the `printf()` function or `String.format()`.

- Example with `printf()`:

```
public class Main {  
    public static void main(String[] args) {  
        double num = 3.14159;  
        System.out.printf("Number with two decimals: %.2f", num); // Output: 3.14  
    }  
}
```

- Example with `String.format()`:

```
public class Main {  
    public static void main(String[] args) {  
        double num = 3.14159;  
        String result= String.format("%.2f", num);  
        System.out.println("Number with two decimals: " + result); // Output: 3.14  
    }  
}
```

When we cover the topic of random numbers, I'll teach you another different method, which for me is the simplest and the one I always use.

Summary of What You've Learned

In Java, to display information in the console, we use the `System.out.println` and `System.out.print` commands. Both display data on the screen, but there's an important difference: `System.out.println` automatically adds a line break after printing, while `System.out.print` doesn't, keeping the next output on the same line.

For example, if we want to display text like 'Hello world' in the console, we write:

```
System.out.println("Hello world");
```

If we want to display a variable, like `total`, which has a value of 9, we write:

```
int total = 9;
```

```
System.out.println(total);
```

We can also perform operations and combine text with variables. For example, if we have two variables `a` and `b` with values 5 and 7 respectively, and we want to show the result of their multiplication along with text, we do it like this:

```
int a = 5;
```

```
int b = 7;
```

```
System.out.println("The result of multiplying a and b is = " + (a * b));
```

Additionally, we can change the text color in the console by adding color codes before the text we want to change. For example, to display text in red, we write:

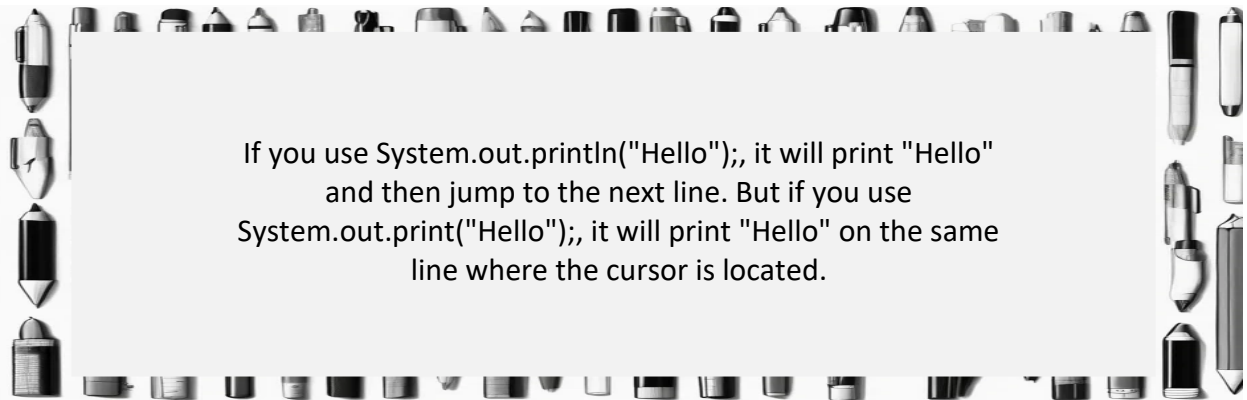
```
System.out.println("\033[31m" + "This text will be red");
```


Finally, to make annotations in our code that won't print to the console and are only meant for us or other programmers reading the code, we use double forward slashes `/**` to indicate a single-line comment.

Practical exercises

Exercise 1. Tell me the difference between `System.out.println` and `System.out.print`.

Solution:



 Use **`System.out.println`** to print messages with a line break at the end. It's useful for displaying information and separating different parts of the output.

Exercise 2. Display a pyramid of asterisks in your IDE's console. Something similar to this:

```
*  
***  
*****
```

Solution:

```
System.out.println(" *");
System.out.println(" ***");
System.out.println(" *****");
```

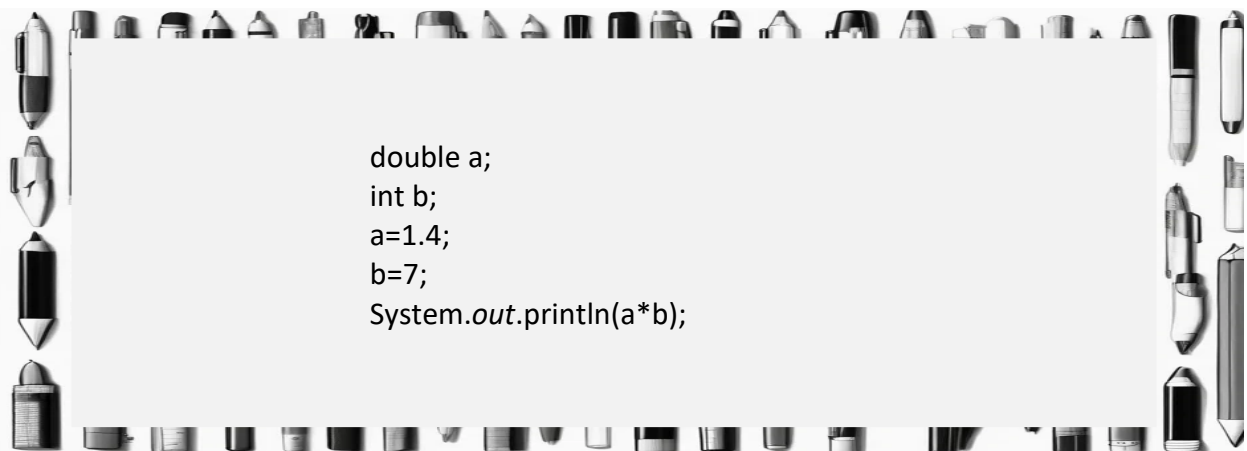
Exercise 3. Create a three-day calendar with three subjects and display it on screen.

Solution:

```
System.out.println("  Monday   |   Tuesday   |   Wednesday  ");
System.out.println("_____");
System.out.println("    Math    |   Language   |   Economics   ");
System.out.println("_____");
System.out.println(" Programming |   Philosophy   |   Geography   ");
System.out.println("_____");
System.out.println("      P.E.   |   Language    |   Chemistry   ");
System.out.println("_____");
```

Exercise 4. Declare two variables: one of type double called a and another of type int called b. Assign them the values 1.4 and 7, respectively. Then, display the multiplication of both on the screen.

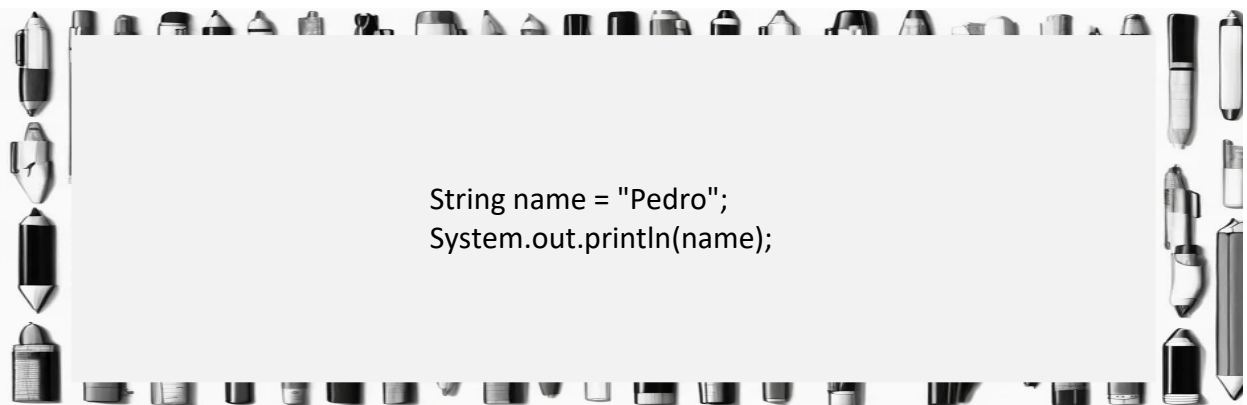
Solution:



```
double a;  
int b;  
a=1.4;  
b=7;  
System.out.println(a*b);
```

Exercise 5. Create a String variable and display it on screen.

Solution:



```
String name = "Pedro";  
System.out.println(name);
```

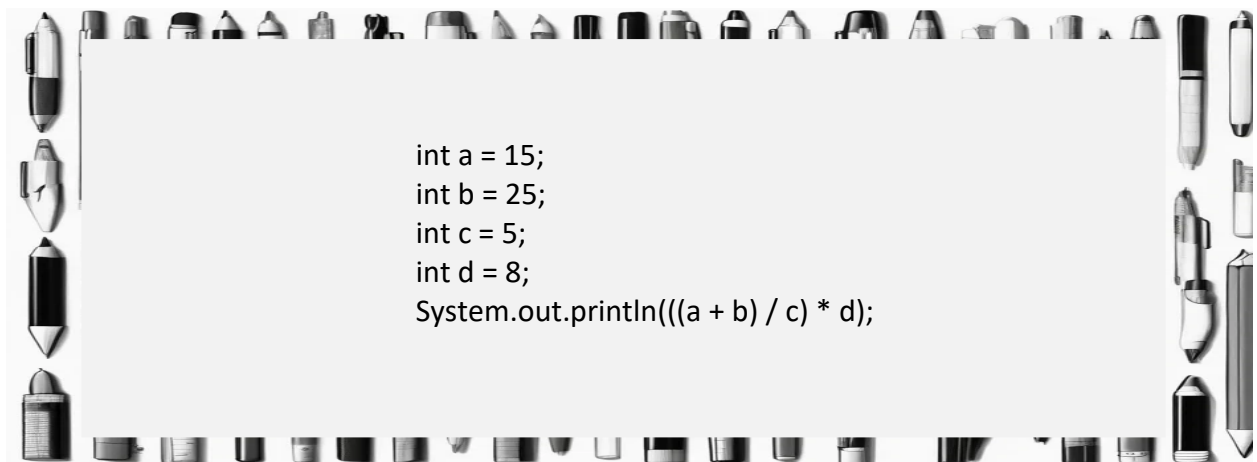
Exercise 6. Create a char variable and display it on screen.

Solution:



Exercise 7. Create 4 variables of type int and name them a, b, c, and d. Assign them the values 15, 25, 5, and 8, respectively. Perform an operation that adds a + b, divides that result by c, and then multiplies everything by d.

Solution:



Exercise 8. Create 4 variables of type `int` and name them `a`, `b`, `c`, and `d`. Assign them the values 14, 24, 5, and 7, respectively. Perform an operation that adds `a + b`, divides that result by `c`, and then multiplies everything by `d`.

Solution:

If you notice in this case, the results are not reliable because dividing the sum of `a` and `b` by `c` gives decimals. The problem is that the `int` data type cannot hold decimals. This causes the decimal part to be lost. So in the end, the result is not exact. Before performing the operations, we must convert the `int` variables to `double`. We will do it as follows:

```
int a = 14;  
int b = 24;  
int c = 5;  
int d = 7;  
double aa=a;  
double bb=b;  
double cc=c;  
double dd=d;
```

Now we can perform the operations:
`System.out.println(((aa+bb)/cc)*dd);`



Use **`System.out.print`** when you need to print without adding a line break. Ideal for building output lines dynamically.

Exercise 9. Create a basic currency converter that converts euros to dollars. We will assume that one euro equals 1.10 dollars. To do this, you should create a double variable and assign it the exact value of euros you need to convert. We want the final value to be displayed in the console.

Solution:

Method 1.

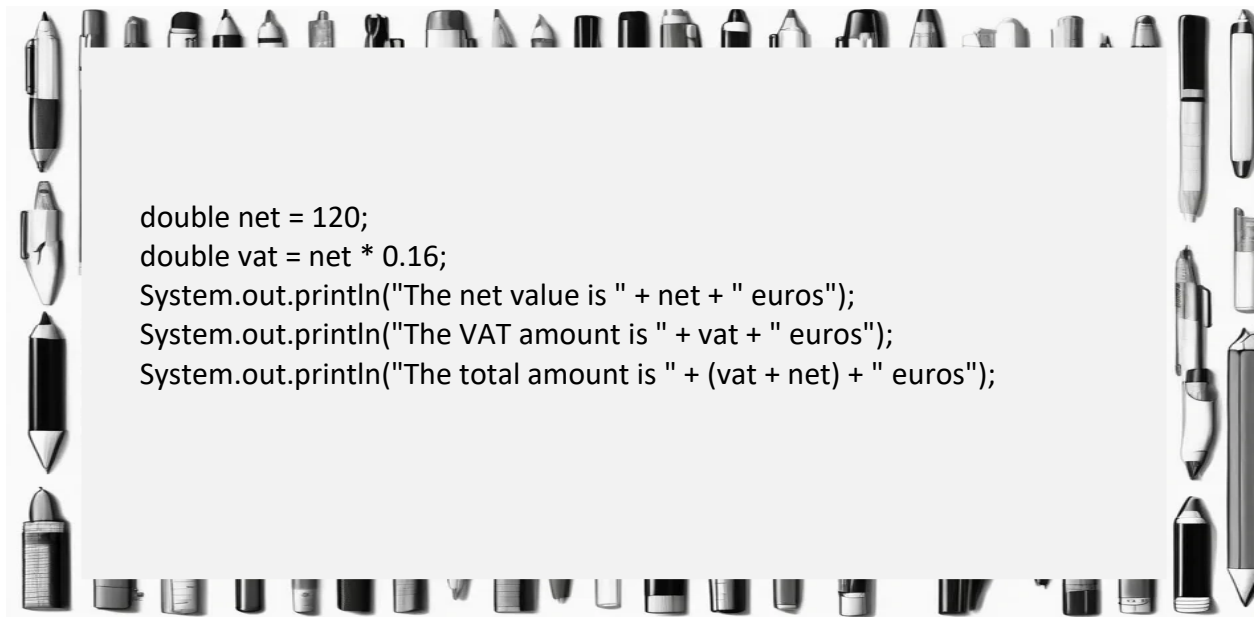
```
double euros= 70;  
System.out.println(euros *1.1);
```

Method 2.

```
double euros= 70;  
double dollards = euros * 1.1;  
System.out.println(dollards);
```


Exercise 10. Create a program that calculates the VAT of an invoice and applies it to the net amount. The program should display both the net value and the VAT separately, and then show the total invoice amount. The net value should be assigned to a double variable. The VAT is 16%.

Solution:



```
double net = 120;  
double vat = net * 0.16;  
System.out.println("The net value is " + net + " euros");  
System.out.println("The VAT amount is " + vat + " euros");  
System.out.println("The total amount is " + (vat + net) + " euros");
```



Concatenate variables and text with the + operator. It facilitates the construction of informative and detailed messages.

Chapter 3.

Rich's Friend / Keyboard Data Input

It's been two days since the party and it's time to clean up. I can't say it's one of my favorite activities, but this time there's no choice. I can't stop thinking about how lucky Chani is to get out of this unpleasant task. Many of the people who came to the party were his friends, and it doesn't seem fair to clean up what they messed up. Although, thinking about it more carefully, we didn't know the reason why he had suddenly left. It's very possible that it was for something important, and the fact that he's not answering his phone worries me quite a bit.

On the other hand, I'm angry. Among other belongings, my laptop is trapped inside his closet. A question keeps circling in my mind: why, before leaving, did he take the time to clean his room? If you need to leave urgently, you just take what you need and go.



Rich seems calm, barely saying anything. Every now and then he smiles and appears to have his mind elsewhere. He's hardly spoken to me all day. But I'm not surprised at all. He's sometimes like this.

He gets lost in his thoughts and spends hours without talking to anyone. Although the truth is, in that state, he's quite productive. He has an amazing ability to concentrate. I think that's why he's a better student than me.

Right now, I prefer not to bother him and let him focus on cleaning. The sooner we finish, the better. I'll leave him to tidy up the kitchen while I mop the rest of the house. After this, everything should be clean.

Finally, it seems this torture is over. After several hours of suffering, I can say we're done. My plan is to lie on the couch for the rest of the day. Rich, on the other hand, seems to have plans with someone. I guess he'll be leaving the house soon. He's in the bathroom getting ready, so he could be there for hours. He loves looking at himself in the mirror and always leaves the house perfectly groomed.

I decide to get comfortable, but soon I have to get up. Someone has rung the doorbell. During the week we don't usually receive visitors, so it seems a bit strange to me. They're asking for Rich, so I open the building door. I stay at the apartment door, to avoid having to get up again. Besides, I'm very curious to see who's coming.

It's a friend of Rich's. I've talked to him a couple of times, but honestly, I don't remember his name. He seems to know the house, although I don't remember him ever being here before. After greeting me, he went straight to Rich's room. I'm a bit surprised, but I guess Rich had already made plans with him.



It seems I can finally relax, and since I have absolutely nothing to do, I've grabbed a Java book and I'm going to try to learn a bit more.

Entering data from keyboard

The topic is quite simple, but before we get into it, we still need to learn some more theoretical concepts. All this information might seem somewhat overwhelming at first, but little by little everything will make sense. As personal advice, I'll tell you that it's important to combine theory with practice. The exercises are pleasant and I would even dare to say entertaining. The important thing is to keep moving forward and know a little more each day. Now it's time to understand in general terms what a class is.

Introduction to the Class Concept

When we work in our IDE, we write our code within these lines:

```
public class Main {  
    public static void main(String[] args)
```

Our code goes here.

```
    }  
}
```

Let me explain what those first two lines mean; you've probably been wondering for a while why they're there and what they're for. In later topics, we'll delve deeper and analyze the concept of class in detail. For now, it's important that you understand that these lines are necessary for the code to identify that we're working with a class, and this will be fundamental in object-oriented programming.

public class Main: In Java, a program is organized into 'classes'. A class is like a blueprint or a recipe that tells the computer how to do something. When you write `public class Main`, you're saying that you're creating a new class called 'Main'. The `public` means that this class is accessible from any part of the program.

When you start a project in Java, you need at least one class to act as a starting point, and it's usually called 'Main' (although it can have any name). This Main class is where the program execution begins.

public static void main(String[] args): It's the main method that the system executes to start a program.

- **public:** This means that the main method is accessible from any part of the program. It's like an open door that allows other methods to call it.
- **static:** This indicates that the main method belongs to the class itself, rather than to a specific instance of the class. In simple terms, you can call `main` without needing to create an object of that class.
- **void:** This means that the main method doesn't return any value. In other words, it doesn't produce any result when executed.
- **main:** This is the name of the method. It's the main entry point for the program.
- **String[] args:** This is a parameter that the main method receives. It's an array of text strings that can contain arguments passed to the program when it runs. For example, if you run your program from the command line and write something after the program name, those 'somethings' are stored in this array. For example, if you run `java myProgram Hello World`, then 'Hello' and 'World' would be stored in `args`.

All of this might seem a bit strange to understand, but soon everything will start to make sense. In the following topics, we'll cover these concepts in more depth. For now, let's focus on learning how to solve problems and understanding other basic concepts.

Program Execution Order

To better understand this section, you need to understand the order in which our Java code lines are executed. The program executes from top to bottom. For example, if we create a variable on line 18 and try to assign it a value on line 17 (one line before), the program will return an error. On line 17, the variable hasn't been created yet, so the program doesn't know it exists. The only valid process is to first create a variable and then assign it a value.

The same happens when inputting data from the keyboard. It doesn't matter if we have five thousand lines written; if on line 22 we've given the instruction that keyboard input is needed, the program won't read the following lines until we input something.

With this small clarification made, we can now dive deep into this topic. First, we need to know what a Scanner is and then how we apply it to our programs and applications.

Importance of Data Input in Interactive Applications

Data input is fundamental in any interactive application, as it allows users to provide the necessary information for the program to function. Here are some key reasons why data input is so important:

1. User-Program Interaction

Data input allows direct interaction between the user and the program. Without this interaction, the software couldn't adapt to the user's needs and preferences.

For example, in a banking application, users can enter their account number and PIN to access their financial information.

2. Customization and Adaptation

Programs can be customized based on user inputs, providing a more relevant and personalized experience.

3. **Information Gathering**

Applications collect data through user input to perform calculations, store information, and make informed decisions.

4. **Control and Navigation**

Data input allows users to navigate through different options and functionalities.

5. **Validation and Security**

Through data input, applications can implement validation and security mechanisms, ensuring that only authorized users access certain functions.

6. **Feedback Collection and Improvements**

Applications can include mechanisms for users to provide feedback, which is crucial for continuous software improvement."

Using the Scanner Class

In Java, a 'scanner' commonly refers to the Scanner class from the `java.util` package. This class is used to read user input from the console or from other input streams, such as files. It allows reading different types of data, such as integers, floating-point numbers, strings, etc.

To use scanner in Java, we must write the following line above our main class:

```
import java.util.Scanner;
```

It would look like this:

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        // program code

    }

}
```

Java Package Concept

In Java, a 'package' is simply a way to group and organize code. Think of it as a folder where you store related files. Packages help us avoid confusion with class names and allow our code to be better organized.

A package can contain various things, such as classes, interfaces, and subpackages. For example, if you're creating a program for an online store, you might have a package called 'com.onlinestore' where you store all the classes related to your store.

To indicate that a class belongs to a package, the package keyword is used at the beginning of the class file. For example:

```
package com.onlinestore;

public class Product {

    // Product class code

}
```

In this example, the Product class belongs to the com.onlinestore package. This means that other classes in the same package can access it directly without needing to import it, while classes outside this package will need to import it to use it. Once we have imported the Scanner package, we can continue writing our program code inside our class.

Creating and Using a Scanner

To input data from the keyboard, the first thing we need to do is create a Scanner, which is always done in the following way:

```
Scanner exercise = new Scanner(System.in);
```

In this case, we have named our Scanner 'exercise', but obviously you can name it whatever you prefer.

Once created, we need to **call that scanner**, and we should do it exactly when we need keyboard input. The program won't continue executing until a value is entered from the keyboard. **The line of code we'll use to call that Scanner will depend on the type of data we're going to input.**

Reading Different Data Types

With Scanner, you can capture text strings, integers, floating-point numbers, boolean values, and more, all with specific methods that ensure accurate and efficient reading. However, depending on the data type, we'll need to use a different line of code to 'call' the Scanner we previously created.

Check this examples:

```
int a = exercise.nextInt();  
double a = exercise.nextDouble();  
String a = exercise.nextLine();  
boolean a = exercise.nextBoolean();  
char a = exercise.next().charAt(0);
```

Be careful as they all have the same structure except in the case of char type variables.

Practical example:

Let's look at an example, as it's the easiest way to understand it. We need to create a program where two integer numbers are input. We'll call them a and b. The program should ask us to input their values one by one. Then, it will multiply both values and show us the result in the console.

In the following code, we'll see how we import the Scanner package, then create a Scanner called exercise, and finally call the Scanner at the exact moment we need the data to be input.

```
import java.util.Scanner;
```

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
//Create the Scanner
```

```
    Scanner exercise = new Scanner(System.in);
```

```
// Ask for the value of number a
```

```
    System.out.println("Enter the value of number a");
```

```
    int a = exercise.nextInt();
```

```
// Ask for the value of number b
```

```
    System.out.println("Enter the value of number b");
```

```
    int b = exercise.nextInt();
```

```
// Perform the operation with both variables and display it on screen
```

```
    System.out.println("The multiplication result is " + (a*b));
```

```
scanner.close();
```

```
    }
```

```
}
```

Closing the Scanner

Closing the Scanner object is a crucial practice in Java to release resources associated with data input. When you finish using a Scanner, especially if it's reading from `System.in`, you should close it using the `close()` method. This not only helps free up memory but also prevents potential security issues and resource leaks. Proper Scanner closure ensures that the program handles system resources efficiently and responsibly. For example, at the end of your program, you can use `scanner.close();` to properly close the Scanner object.

Summary of what you have learned

In Java, a class is like a building design: it defines the structure and organizes the parts of the program. The Main class is where everything begins, that is, the program's entry point. When we write `public class Main`, we're saying this class is accessible from any part of the code.

Within the Main class, the `public static void main(String[] args)` method is fundamental because it's where the program first starts executing. Here's an explanation of each part of this method:

`public`: This means the method can be used from anywhere in the program.

`static`: Means this method belongs directly to the class and not to an object created from the class.

`void`: Indicates that the method won't return any value.

`main`: Is the method name that Java always looks for when starting a program.

`String[] args`: Is an array that can contain arguments provided when the program runs.

The main concept explained in this topic is keyboard data input. For this, we use the `Scanner` class. This class facilitates reading data like numbers and text from the keyboard.

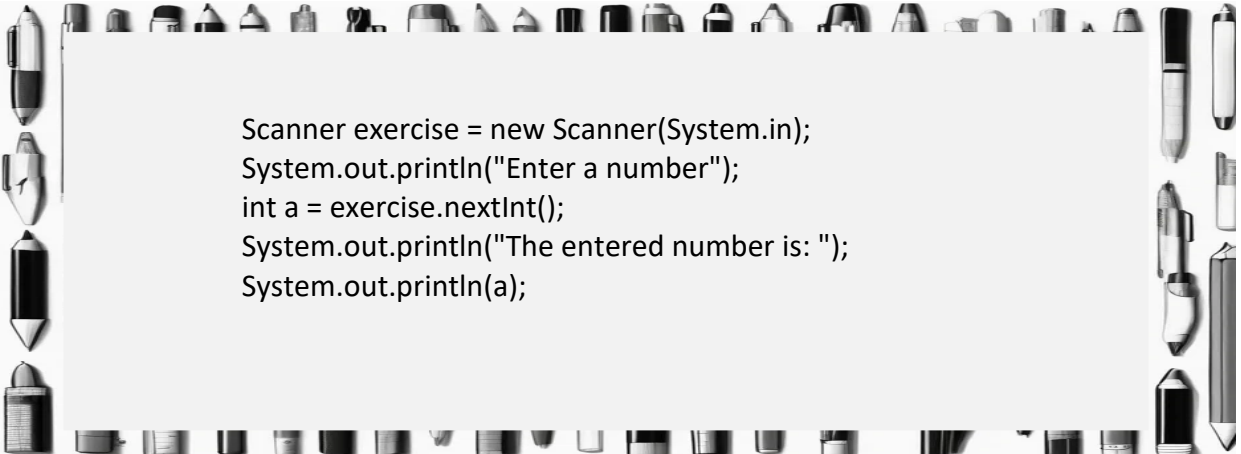
To use `Scanner`, we first need to 'import' the class. Then, we create the `Scanner` itself, and call it at the exact part of the program where we need it. Depending on the type of data we're handling, the code we use to call the `Scanner` will be different. Finally, remember that it's important to close the `Scanner` when finished, to release the resources it was using.

In summary, you now understand how a class works in Java and how to handle data input with `Scanner`. And if it's still not completely clear to you, I now propose some exercises that will definitely clear up your doubts.

Practical exercises

Exercise 1. Create a program that allows us to input an integer number and then display it on screen. Remember to import the Scanner class. `import java.util.Scanner;`

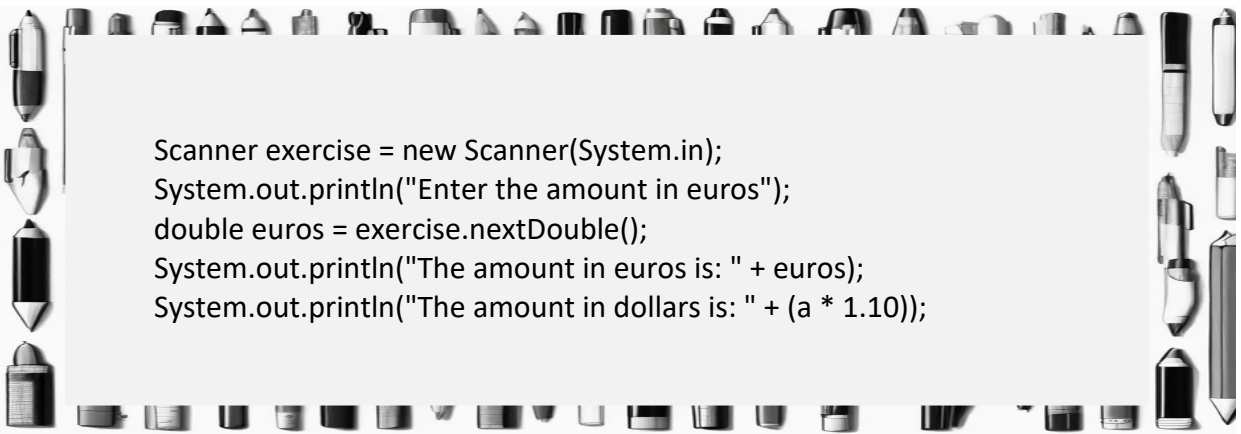
Solution:



```
Scanner exercise = new Scanner(System.in);
System.out.println("Enter a number");
int a = exercise.nextInt();
System.out.println("The entered number is: ");
System.out.println(a);
```

Exercise 2. Create a currency converter program where we input an amount in euros and it returns the result in dollars. We'll assume that 1 euro equals 1.10 dollars.

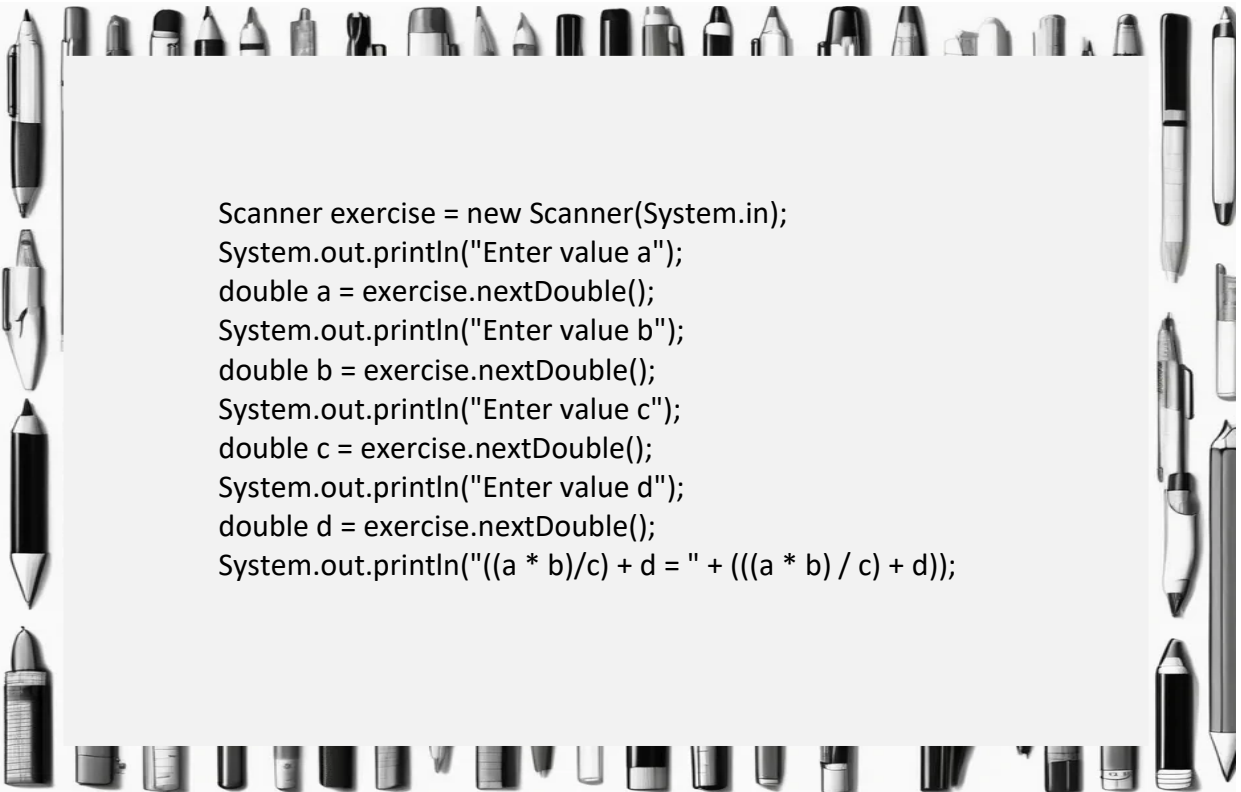
Solution:



```
Scanner exercise = new Scanner(System.in);
System.out.println("Enter the amount in euros");
double euros = exercise.nextDouble();
System.out.println("The amount in euros is: " + euros);
System.out.println("The amount in dollars is: " + (euros * 1.10));
```

Exercise 3. Create four double variables and name them a, b, c, and d. The program should ask to input the data for each variable from the keyboard. Then, the program will multiply a by b, divide the result by c, and add d to all of it. The result should be displayed in the console.

Solution:

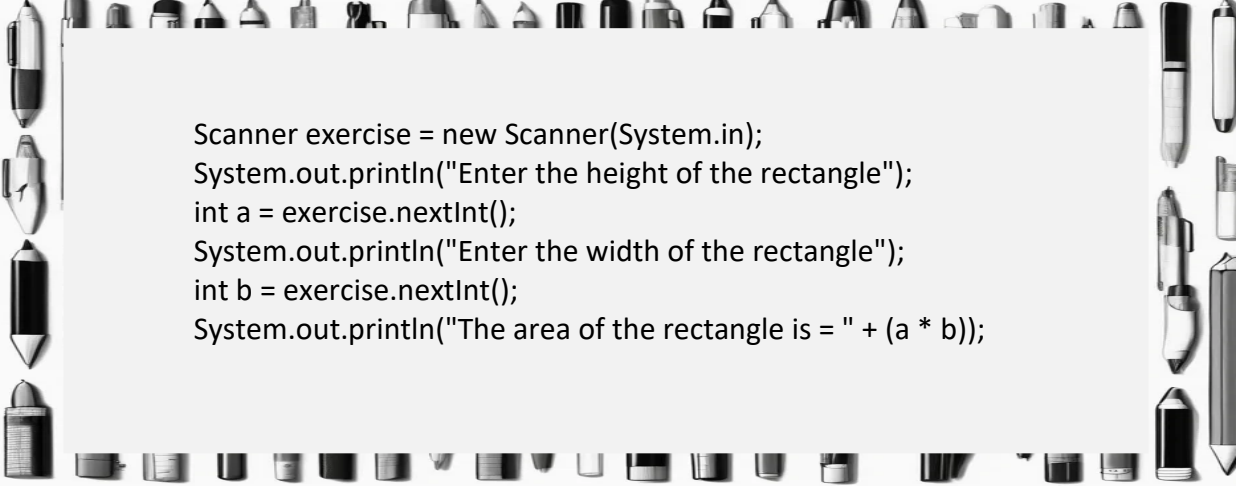


```
Scanner exercise = new Scanner(System.in);
System.out.println("Enter value a");
double a = exercise.nextDouble();
System.out.println("Enter value b");
double b = exercise.nextDouble();
System.out.println("Enter value c");
double c = exercise.nextDouble();
System.out.println("Enter value d");
double d = exercise.nextDouble();
System.out.println("((a * b)/c) + d = " + (((a * b) / c) + d));
```

💡 Scanner provides specific methods for reading different data types, such as `nextInt()` for integers, `nextDouble()` for floating-point numbers, and `nextLine()` for strings. Use the appropriate method according to the type of data you expect to receive.

Exercise 4. Create a program that calculates the area of a rectangle. For this, it should ask for the height and width measurements. Remember that the area of a rectangle equals the multiplication of its sides.

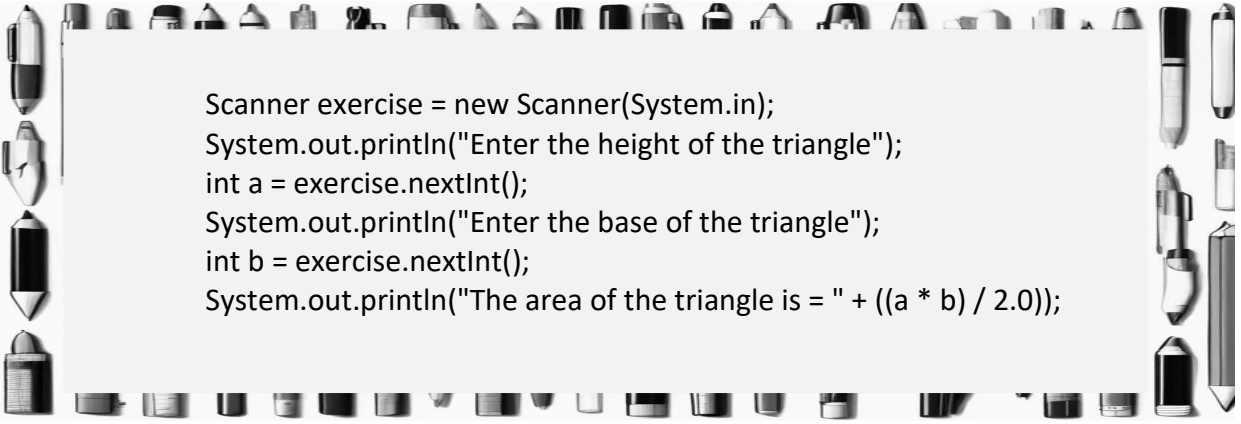
Solution:



```
Scanner exercise = new Scanner(System.in);
System.out.println("Enter the height of the rectangle");
int a = exercise.nextInt();
System.out.println("Enter the width of the rectangle");
int b = exercise.nextInt();
System.out.println("The area of the rectangle is = " + (a * b));
```

Exercise 5. Create the same program as in the previous exercise, but in this case, calculate the area of a triangle. Remember that the area of a triangle is the base times the height, all divided by two.

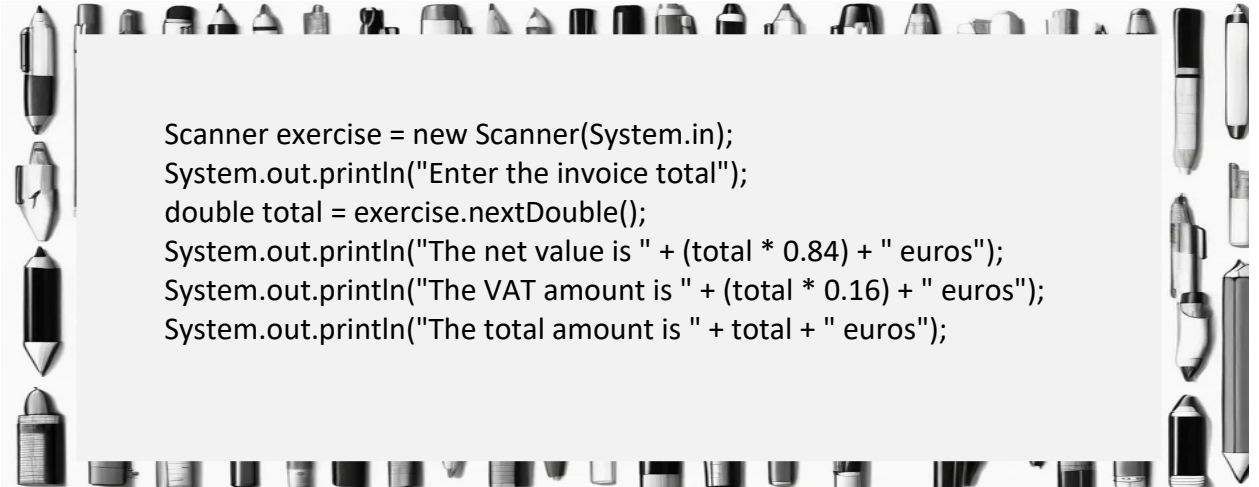
Solution:



```
Scanner exercise = new Scanner(System.in);
System.out.println("Enter the height of the triangle");
int a = exercise.nextInt();
System.out.println("Enter the base of the triangle");
int b = exercise.nextInt();
System.out.println("The area of the triangle is = " + ((a * b) / 2.0));
```

Exercise 6. Create a program where you input the total amount of an invoice and it tells you how much VAT has been paid. The tax rate is 16%.

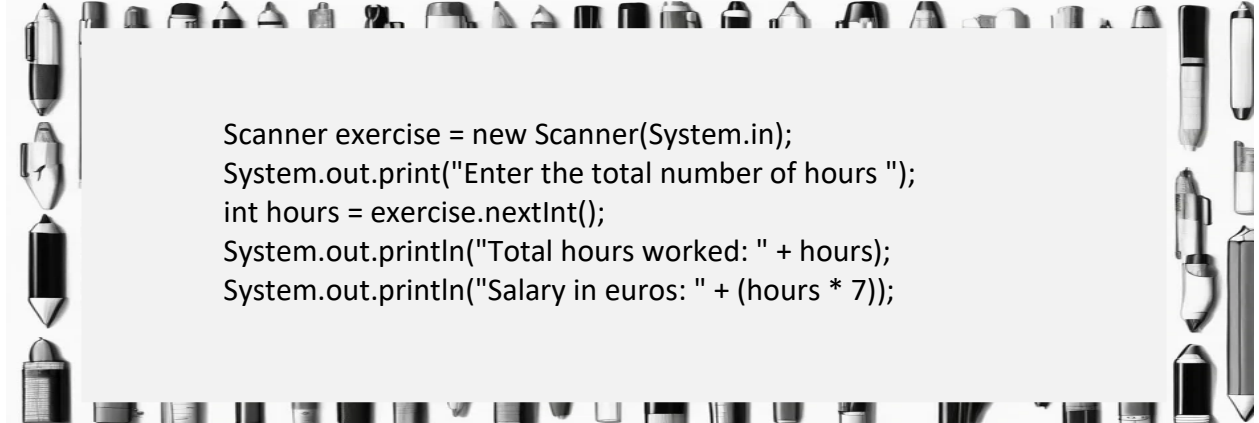
Solution:



```
Scanner exercise = new Scanner(System.in);
System.out.println("Enter the invoice total");
double total = exercise.nextDouble();
System.out.println("The net value is " + (total * 0.84) + " euros");
System.out.println("The VAT amount is " + (total * 0.16) + " euros");
System.out.println("The total amount is " + total + " euros");
```

Exercise 7. Create a program that calculates the weekly salary of a worker based on the hours worked. Up to 40 hours, they earn 7 euros/hour. The program should display the total number of hours, as well as the total salary.

Solution:



```
Scanner exercise = new Scanner(System.in);
System.out.print("Enter the total number of hours ");
int hours = exercise.nextInt();
System.out.println("Total hours worked: " + hours);
System.out.println("Salary in euros: " + (hours * 7));
```


Exercise 8. Celsius to Fahrenheit temperature converter. The formula is: $(\text{Celsius} * 9 / 5) + 32$.

Solution:

```
Scanner exercise = new Scanner(System.in);
System.out.println("Welcome to the temperature converter!");
System.out.println("Please, enter the temperature in Celsius degrees:");
double celsius = exercise.nextDouble();
System.out.println("The temperature in Fahrenheit is: " + ((celsius * 9 / 5) + 32));
```

Exercise 9: Create a program that asks us to enter a symbol. We will store it in a char variable. Then, the program should display the symbol in the console.

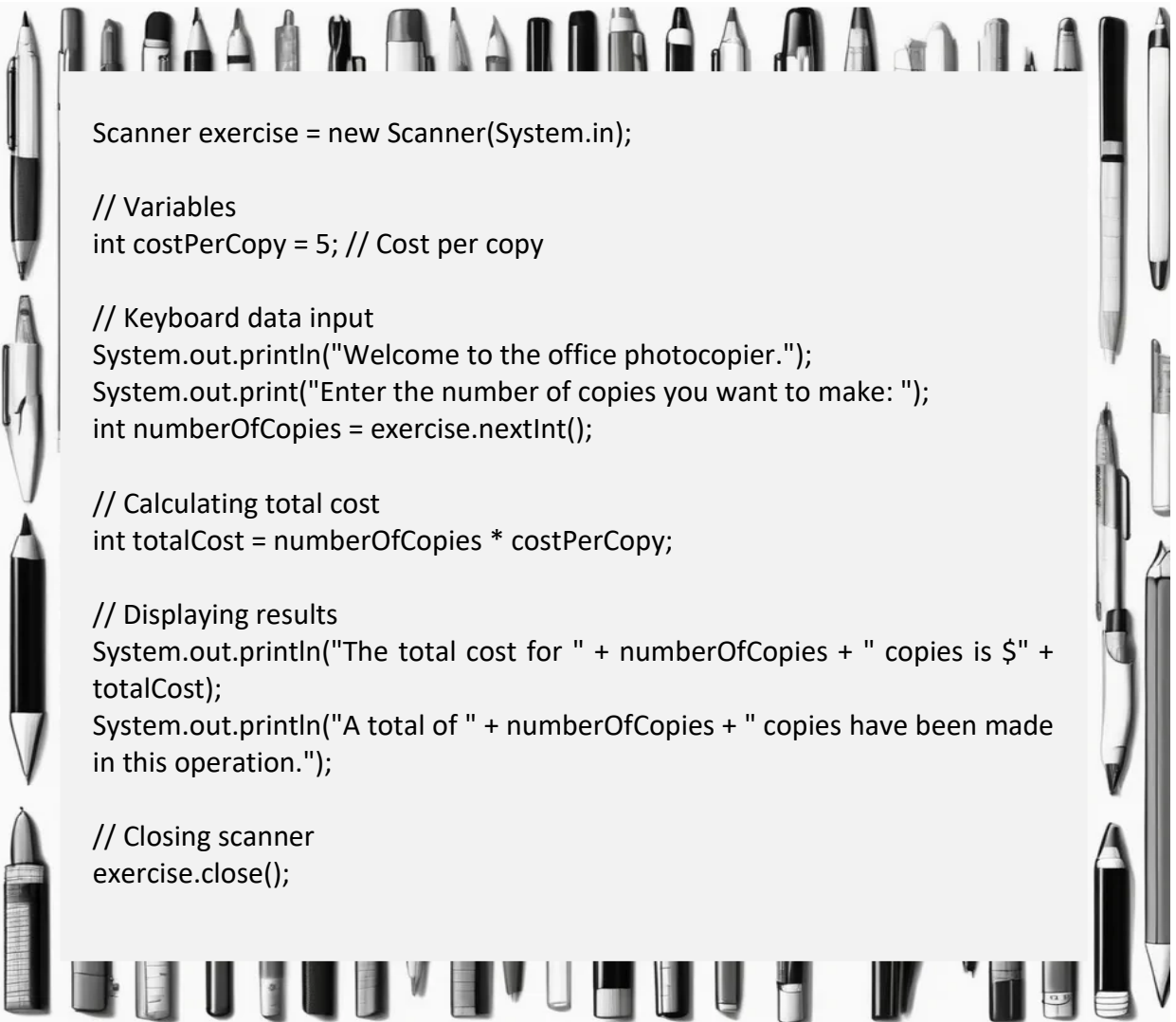
Solution:

```
Scanner exercise = new Scanner(System.in);
System.out.print("Enter a symbol ");
char a = exercise.next().charAt(0);
System.out.println("The entered symbol is: " + a);
```

💡 It's important to close the Scanner object with the `close()` method when finishing its use to properly release system resources. This prevents possible memory leaks and ensures efficient resource management.

Exercise 10. Create a program that simulates the usage control of a photocopier in an office. The program will allow employees to enter the number of copies they want to make and will calculate the total cost. It will also keep track of the total copies made during the day.

Solution:



```
Scanner exercise = new Scanner(System.in);

// Variables
int costPerCopy = 5; // Cost per copy

// Keyboard data input
System.out.println("Welcome to the office photocopier.");
System.out.print("Enter the number of copies you want to make: ");
int numberOfCopies = exercise.nextInt();

// Calculating total cost
int totalCost = numberOfCopies * costPerCopy;

// Displaying results
System.out.println("The total cost for " + numberOfCopies + " copies is $" +
totalCost);
System.out.println("A total of " + numberOfCopies + " copies have been made
in this operation.");

// Closing scanner
exercise.close();
```

Chapter 4.

Hanging out at the police station / Working with conditionals

I don't know how it happened, but I'm at the police station. Specifically, I'm locked in a room with Rich and his friend, who, by the way, is named Fernando. The story is quite simple to tell, although it doesn't make any sense to me. I don't know exactly what's happening, or why I've been arrested.

Shortly after Fernando arrived at the house, the police burst in without warning. Everything happened very quickly. Several armed men entered shouting and ordering us to lie on the floor. They gathered the three of us in the living room and sat us on the floor. Meanwhile, they started searching the entire house. I don't know what they were looking for, but it was clear they didn't want to leave anything unchecked. The search seemed very thorough.

A few minutes after the search began, I saw a police officer come out with our computers. I couldn't help but think that they had managed to open Chani's cabinet, while we had spent two days unable to access our things. It didn't seem fair to me.

I also wondered who was going to clean up all that mess. We had just cleaned the whole house and now everything was even worse than after the party. But what kept going through my mind most often was Fernando's bad luck. He had never come before, and 20 minutes after arriving, the police show up. At no point did I think his visit had anything to do with this arrest. It was simply very bad luck.



Except for the initial shock, I haven't felt worried at any point. I'm not afraid, since I haven't done anything wrong. They can check my computer and my phone if they want. They won't find anything that could be considered a crime.



Fernando is completely pale and hasn't said a word since they brought us to this room. Rich, on the other hand, is very calm. He's always calm. He says it must have been a mistake. In our building, there are many rented apartments and lots of people just passing through. Most likely they entered the wrong apartment. I think the same.

The worst part is not knowing what's happening or how long we'll have to wait. They confiscated our phones, so we don't have much to do. Rich's suggestion is that we spend some time improving my Java knowledge. I don't really feel like it, but that will keep our minds occupied.

It seems the next topic is conditionals. According to Rich, this is going to help us greatly improve our programs. He says it's a fun and easy topic to learn, so it has greatly increased my morale. Let's get started on it as soon as possible.

Working with Conditionals

The Importance of Conditionals in Java

Conditionals allow programs to make decisions based on different situations. Without them, programs would follow a series of steps strictly, without being able to adapt to changes or different types of data. Conditionals make code more flexible, which is essential for creating interactive and functional applications.

Thanks to conditionals, programmers can make their programs respond to user actions, environmental changes, or real-time data. This is very important for developing useful applications in the real world, as they allow programs to handle different situations according to the variables that the user inputs.

For example, in a banking application, conditionals are necessary to verify transactions and validate user-entered information. They also help maintain system security. In video games, conditionals control how the game reacts to player actions, making the experience more dynamic and interactive.

Basic Structure of an if Statement

An "if" is a way to tell our program to do something only if a specific condition is met. Think of this like making decisions in everyday life. For example, "If it's raining, take an umbrella." In programming, it's written in a similar way:

```
if (condition) {  
    // Code that executes if the condition is true  
}
```

1. **if:** This keyword indicates the beginning of a condition.
2. **Condition:** Inside the parentheses, we write the condition we want to check. For example, "if a number is greater than 10."
3. **Code between braces { }:** Inside the braces, we write the code that we want to execute if the condition is true.

Let's have a look at a very simple example. Imagine you're creating a program and you want it to say "It's a big number" if a number is greater than 10. This is how you should write the code:

```
int number = 15; // We have defined a number  
if (number > 10) {  
    System.out.println("It's a big number");  
}
```

- As you can see, the first thing we've done is define a number and give it the value 15.
- Then we want it to understand that we want it to perform an action, but only if the number is greater than 10. `if (number > 10)`: Here we're saying "if the number is greater than 10".
- Finally, we create a line using `System.out.println` that will print to the screen if the condition is true.

Therefore, if the number is greater than 10 (in this case, 15 is greater than 10), the computer will print "It's a big number"; if it's less, it simply won't do anything at all.

else and else if Statements

When we use `if`, we can tell our program to do something if a condition is true. But what happens if that condition isn't met? That's where `else` and `else if` statements come in.

- "Else"

The `else` statement is used to specify a block of code that will execute if the `if` condition is not true. It's like saying: "If the condition isn't met, do this instead."

```
int number = 15;  
if (number > 10) {  
    System.out.println("It's a big number");  
} else {  
    System.out.println("It's a small number");  
}
```

Above we see the previous example but slightly modified. If the number is greater than 10, it will show "It's a big number". Exactly the same as in the previous program, but before if a number was less than 10 it didn't do anything, now it will show the following text: "It's a small number".

- "Else if"

The else if statement is used to check another condition if the first if condition is not met. It's like saying: "If the first condition isn't met, try this other condition"

```
int num = 15;

if (numero > 10) {
    System.out.println("It's a big number");
} else if (numero > 5) {
    System.out.println("It's a medium number");
} else {
    System.out.println("It's a small number");
}
```

We repeat the same example again. Only now there aren't two conditionals but three. The example speaks for itself. The first time we use an else if since we still want to propose one more condition. The second time, a simple else, since it's the last condition we need to propose.

Nesting Conditionals

Explaining what nesting conditionals means is simple; understanding it might be a bit more difficult, but here we go. Nesting conditionals means placing an if statement (or else if or else) inside another if statement. This allows us to create more complex decisions by checking multiple conditions.

To avoid getting completely confused, it's best to explain it directly with an example. Pay attention, as the code starts to become a bit more complex.

Let's create a program that tells us:

- If the number is greater than 10.
- If the number is greater than 5 but less than or equal to 10.
- If the number is greater than 2 but less than or equal to 5.
- If the number is less than or equal to 2.

// For this example, let's assume the number we're considering is 3.

// But it would work the same with any number.

```
int num = 3;
if (num > 10) {
    System.out.println("The number is bigger tan 10");
} else {
    if (num > 5) {
        System.out.println("The number is greater than 5 but less than or equal to 10");
    } else {
        if (num > 2) {
            System.out.println("The number is greater than 2 but less than or equal to 5");
        } else {
            System.out.println("The number is less than or equal to 2");
        }
    }
}
```


That's the code, now let's analyze it by parts.

1) First if condition:

- We check if number is greater than 10.
- If true, we print "The number is greater than 10".
- If number is not greater than 10, we enter this else block.

2) Second if condition (nested inside the first else):

- We check if number is greater than 5.
- If true, we print "The number is greater than 5 but less than or equal to 10".
- If number is not greater than 5, we enter this else block.

3) Third if condition (nested inside the second else):

- We check if number is greater than 2.
- If true, we print "The number is greater than 2 but less than or equal to 5".
- If number is not greater than 2, we print "The number is less than or equal to 2".

Making a side note and realizing that in this case, the theory is much harder than the practice, I would like to give you some advice. Keep your code as simple and clean as possible. Although I understand that, in some cases, depending on the program's complexity, it may be impossible or at least very difficult.

If we have to deal with situations where we need to use many nested conditionals, it's best to use else if. Look at this example of how the code becomes much clearer:

```
int num = 3;
```

```
if (num > 10) {
```

```
    System.out.println("The number is greater than 10");
```

```
} else if (num > 5) {
```

```
    System.out.println("The number is greater than 5 but less than or equal to 10");
```

```
} else if (num > 2) {  
    System.out.println("The number is greater than 2 but less than or equal to 5");  
} else {  
    System.out.println("The number is less than or equal to 2");  
}
```

Even explaining it is simpler:

- 1) First if Condition:
 - We check if number is greater than 10.
- 2) First else if Condition:
 - If number is not greater than 10, we check if it's greater than 5.
- 3) Second else if Condition:
 - If number is not greater than 5, we check if it's greater than 2.
- 4) else Condition:
 - If none of the previous conditions are true, we execute this block.

By using else if, we can make our code more readable and easier to understand. This is very useful for making decisions based on multiple criteria.

Switch case

The switch-case in Java is a conditional structure used when we have several possible options and want to execute different blocks of code depending on the value of a variable. It's a simpler version to write than a bunch of chained if-else statements.

In a switch, the variable we're evaluating is compared to several values called cases. If it finds a match, it executes the code corresponding to that case. If it doesn't find any matches, we can use a default, which is like a "plan B": the code in default will execute if none of the other options are met.

Let's look at a simple example. We'll create a program that tells the user what day of the week it is based on a number from 1 to 7 (1-Monday, 2-Tuesday, etc.). We'll use switch-case to decide which day corresponds to each number.

```
public class Main {  
  
    public static void main(String[] args) {  
  
        int day = 3; // Let's suppose the user enters the number 3  
  
        switch (day) {  
  
            case 1:  
  
                System.out.println("Monday");  
  
                break;  
  
            case 2:  
  
                System.out.println("Tuesday");  
  
                break;  

```

```
case 3:

    System.out.println("Wednesday");

    break;

case 4:

    System.out.println("Thursday");

    break;

case 5:

    System.out.println("Friday");

    break;

case 6:

    System.out.println("Saturday");

    break;

case 7:

    System.out.println("Sunday");

    break;

default:

    System.out.println("Invalid number");

    break;
}
}
```

Comparison Operators

Operators won't just be useful with conditionals; from now on, you'll use them in practically all the programs and exercises you do. Therefore, pay attention and keep this table in sight. I assure you that you'll need it soon, and until you memorize it, you'll have no choice but to come back and take a look.

`a == b`: Checks if a is equal to b.

`a != b`: Checks if a is not equal to b.

`a > b`: Checks if a is greater than b.

`a >= b`: Checks if a is greater than or equal to b.

`a < b`: Checks if a is less than b.

`a <= b`: Checks if a is less than or equal to b.

Logical Operators

The same goes for logical operators: you'll see them everywhere and they are fundamental if you want to program in Java. However, I consider them perhaps somewhat more difficult to understand than comparison operators. That's why, after showing you the table, I'll show you a small example.

- `&&` (Logical AND)

`a && b`: Checks if both conditions a and b are true.

- `||` (Logical OR)

`a || b`: Checks if at least one of the conditions a or b is true.

- `!` (Logical NOT)

`!a`: Inverts the logical value of a.

```
public class Main {  
    public static void main(String[] args) {  
  
        int a = 6;  
  
        // Logic OR  
        // We'll use || to see if a is odd or greater than 8.  
  
        if (a>8 || a%2==0) {  
            System.out.println("a is even or greater than 8");  
        } else System.out.println("a doesn't meet either of the two conditions ");  
  
        // Logic AND  
        // We'll use && to see if a is even and also greater than 8.  
  
        int b = 10;  
        if (b>8 && b%2==0) {  
  
            System.out.println("b is even and greater than 8");  
        } else System.out.println("b doesn't meet one of the two conditions");  
  
        // Logic NOT  
        // We use this operator to invert the logical value  
        // For variables a and b we looked for even numbers. For this one we want odd.
```

```
int c = 7;

if (c%2!=0) {
    System.out.println("c is odd ");
} else System.out.println("c is even ");

}
}
```

Let's see another example :

```
int x = 5;
int y = 10;

// Logical AND with comparison operators
if (x < 10 && y > 5) {
    System.out.println("x is less than 10 and y is greater than 5");
}

// Logical OR with comparison operators
if (x < 10 || y < 5) {
    System.out.println("At least one of the conditions is true");
}
```

```
// Logical NOT with comparison operators
```

```
if (!(x > 10)) {
```

```
    System.out.println("x is not greater than 10");
```

```
}
```

- `x < 10 && y > 5`: Checks if x is less than 10 and y is greater than 5. Both conditions are true, so the message prints.
- `x < 10 || y < 5`: Checks if x is less than 10 or y is less than 5. The first condition is true, so the message prints.
- `!(x > 10)`: Checks if x is not greater than 10. The original condition `x > 10` is false, so `!(x > 10)` is true and the message prints.

Summary of what you've learned

Conditionals in Java are absolutely necessary for programs to make decisions based on different scenarios. Without conditionals, programs would follow rigid instructions and couldn't adapt to changes or diverse data. Conditionals add flexibility and dynamism, allowing the creation of interactive and functional applications that respond to user actions, environmental changes, or real-time data.

The basic structure of a conditional in Java starts with the keyword "if", followed by a condition to be checked. If the condition is true, the specified code is executed. If the condition is not met, "else" and "else if" statements can be used to handle other possible situations, providing alternative code blocks as needed. Nesting conditionals allows for more complex decisions by checking multiple conditions within others.

Another slightly less intuitive structure is the switch-case. We could say it's a simpler version to write than a bunch of chained if-else statements. The main advantage of this structure is its simplicity when there are multiple possible options.

Additionally, we've analyzed comparison and logical operators, as we couldn't work with conditionals without them. Comparison operators include `==`, `!=`, `>`, `>=`, `<`, and `<=`.

Logical operators, such as `&&` (AND), `||` (OR), and `!` (NOT), allow for combining and manipulating conditions, making the code more dynamic and efficient. Using these operators, we can perform complex checks and make decisions based on multiple criteria.

Although at this point the theory might seem a bit overwhelming, with the exercises we've solved, you shouldn't have any problem handling these operators. Once you complete the other exercises proposed for this topic, you'll see that you won't have any doubts left.

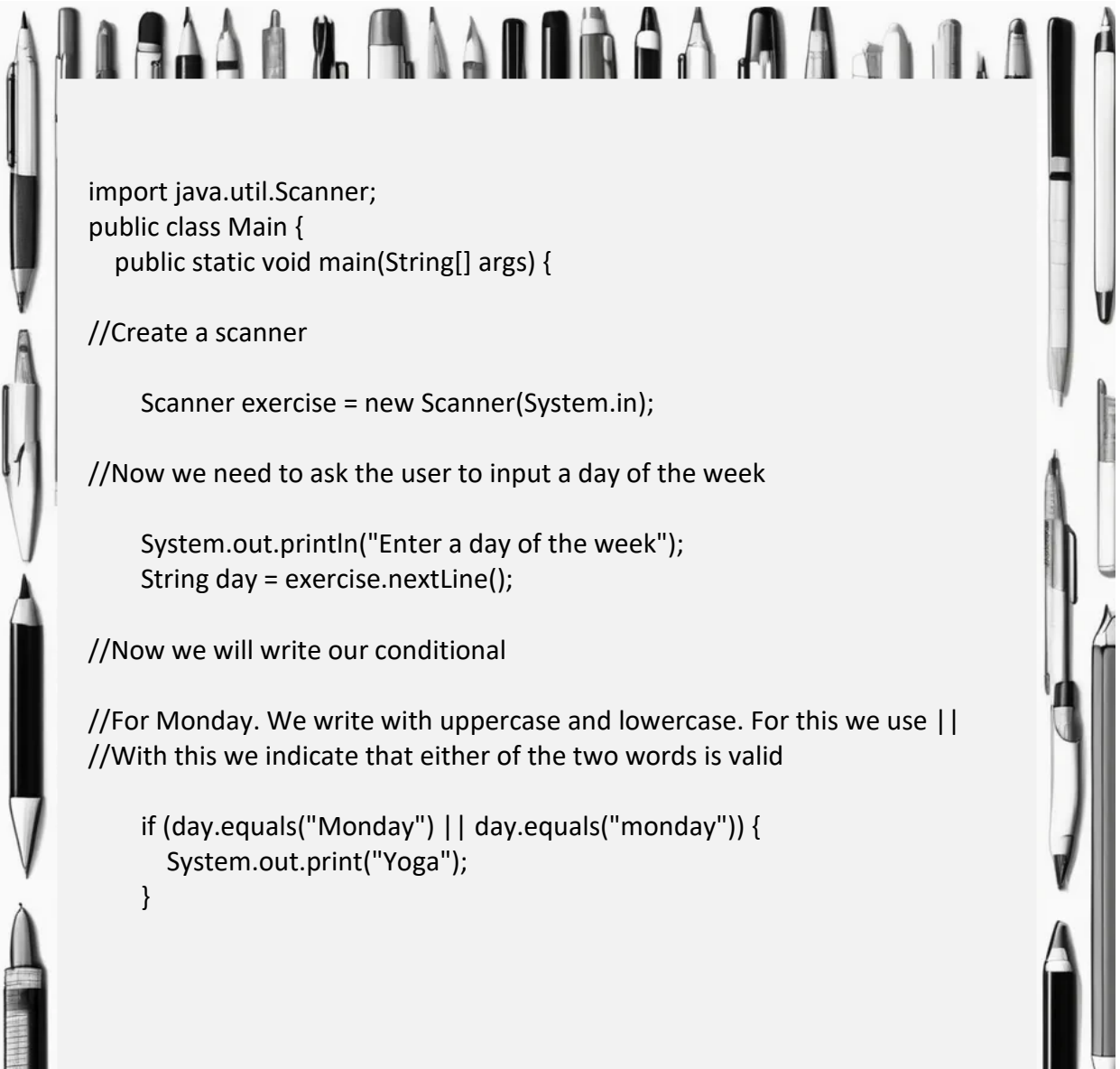
To make your life easier, I'm leaving you again with the list of all the operators. Keep it in mind, as it will be very useful to you:

Operator	Description
+	Adds two values
-	Subtracts two values
*	Multiplies two values
/	Divides the first value by the second
%	Returns the remainder of the integer division
++	Increments the value of the variable by 1
--	Decrements the value of the variable by 1
+=	Adds the specified value to the variable
-=	Subtracts the specified value from the variable
*=	Multiplies the variable by the specified value
/=	Divides the variable by the specified value
==	Checks if two values are equal
!=	Checks if two values are not equal
>	Checks if the first value is greater than the second
<	Checks if the first value is less than the second
>=	Checks if the first value is greater or equal to the second
<=	Checks if the first value is less than or equal to the second
&&	Logical AND operator
	Logical OR operator
!	Logical NOT operator
?:	Ternary (conditional) operator
instanceof	Checks if an object is an instance of a class

Practical exercises

Exercise 1. Type a day of the week and, depending on which one it is, we will show the activity that we need to do. The activities are: Yoga, Biking, Gym, Painting, and Reading.

Solution:



```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        //Create a scanner

        Scanner exercise = new Scanner(System.in);

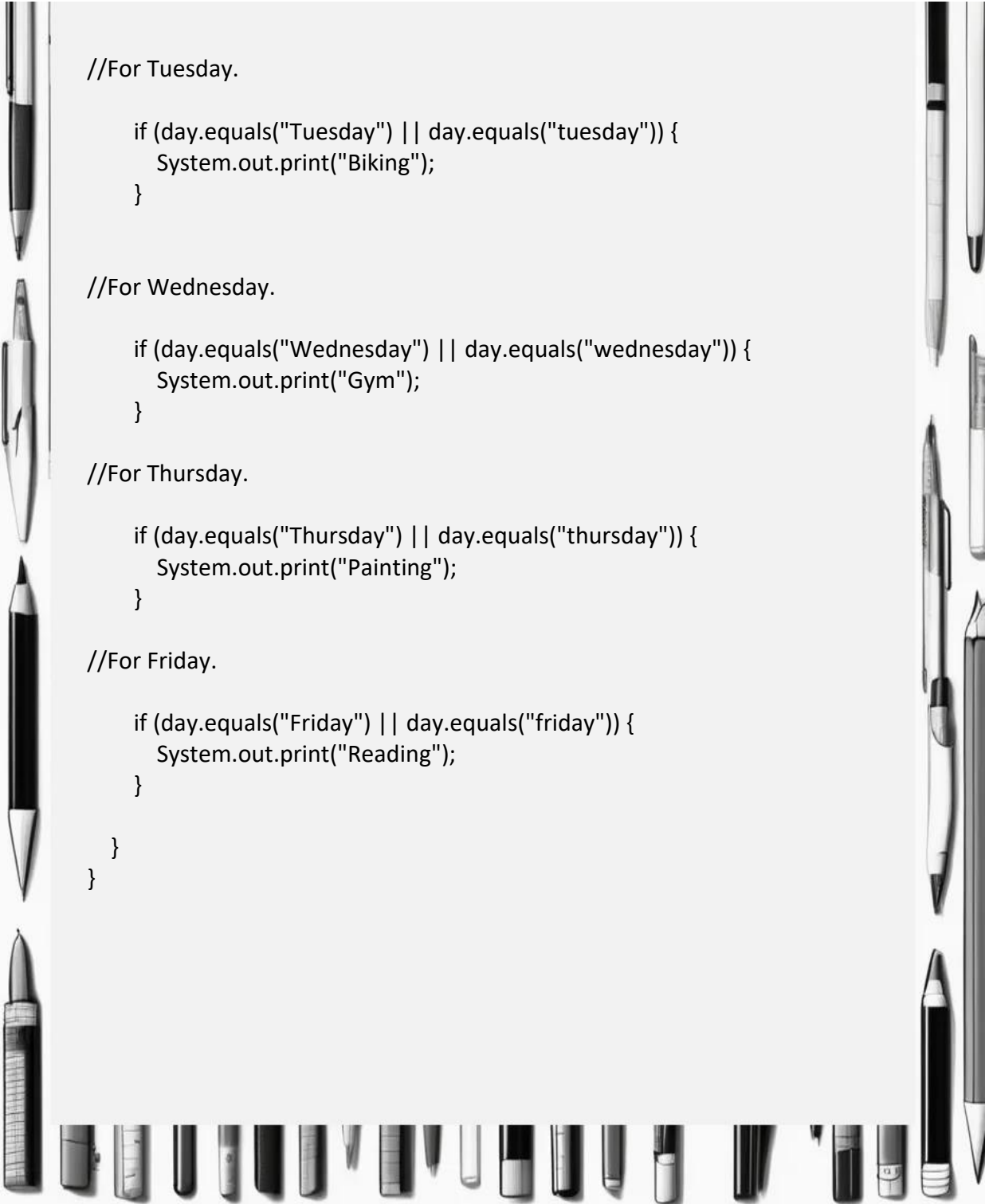
        //Now we need to ask the user to input a day of the week

        System.out.println("Enter a day of the week");
        String day = exercise.nextLine();

        //Now we will write our conditional

        //For Monday. We write with uppercase and lowercase. For this we use ||
        //With this we indicate that either of the two words is valid

        if (day.equals("Monday") || day.equals("monday")) {
            System.out.print("Yoga");
        }
    }
}
```



```
//For Tuesday.
```

```
    if (day.equals("Tuesday") || day.equals("tuesday")) {  
        System.out.print("Biking");  
    }
```

```
//For Wednesday.
```

```
    if (day.equals("Wednesday") || day.equals("wednesday")) {  
        System.out.print("Gym");  
    }
```

```
//For Thursday.
```

```
    if (day.equals("Thursday") || day.equals("thursday")) {  
        System.out.print("Painting");  
    }
```

```
//For Friday.
```

```
    if (day.equals("Friday") || day.equals("friday")) {  
        System.out.print("Reading");  
    }  
}
```

Exercise 2. Exactly the same as the previous one, but in case we enter an incorrect day, the program will tell us if we have entered incorrect data.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner(System.in);
        System.out.println("Enter a day of the week");

        String day = exercise.nextLine();

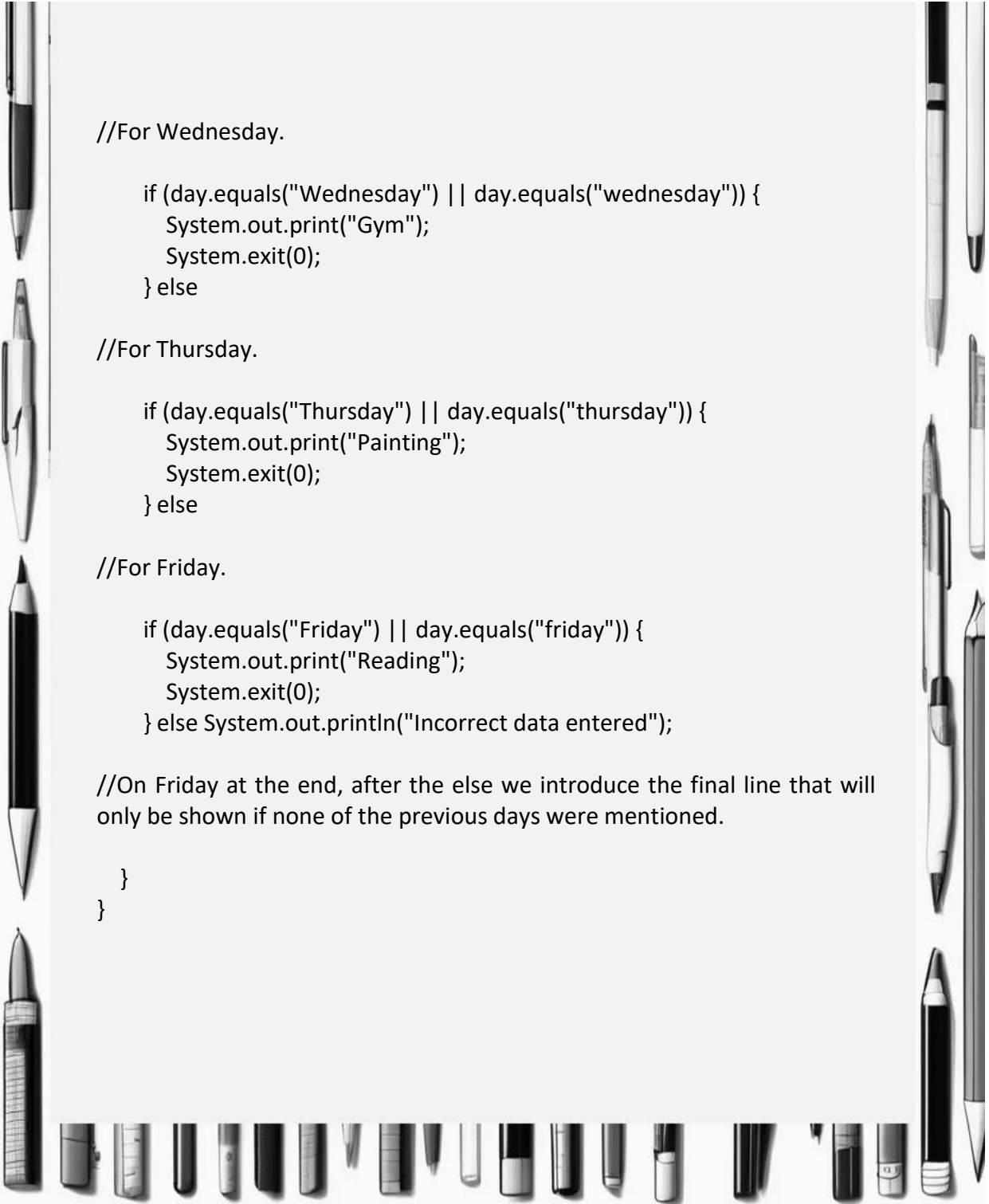
        //We are going to use else and system.exit in the way you'll see
        next:

        if (day.equals("Monday") || day.equals("monday")) {
            System.out.print("Yoga");
            System.exit(0);
        } else

        //This way the program will close if it matches the day of the week
        we entered. Otherwise, it will use the else to go to the next
        conditional.

        //For Tuesday.

        if (day.equals("Tuesday") || day.equals("tuesday")) {
            System.out.print("Biking");
            System.exit(0);
        } else
```



```
//For Wednesday.
```

```
    if (day.equals("Wednesday") || day.equals("wednesday")) {  
        System.out.print("Gym");  
        System.exit(0);  
    } else
```

```
//For Thursday.
```

```
    if (day.equals("Thursday") || day.equals("thursday")) {  
        System.out.print("Painting");  
        System.exit(0);  
    } else
```

```
//For Friday.
```

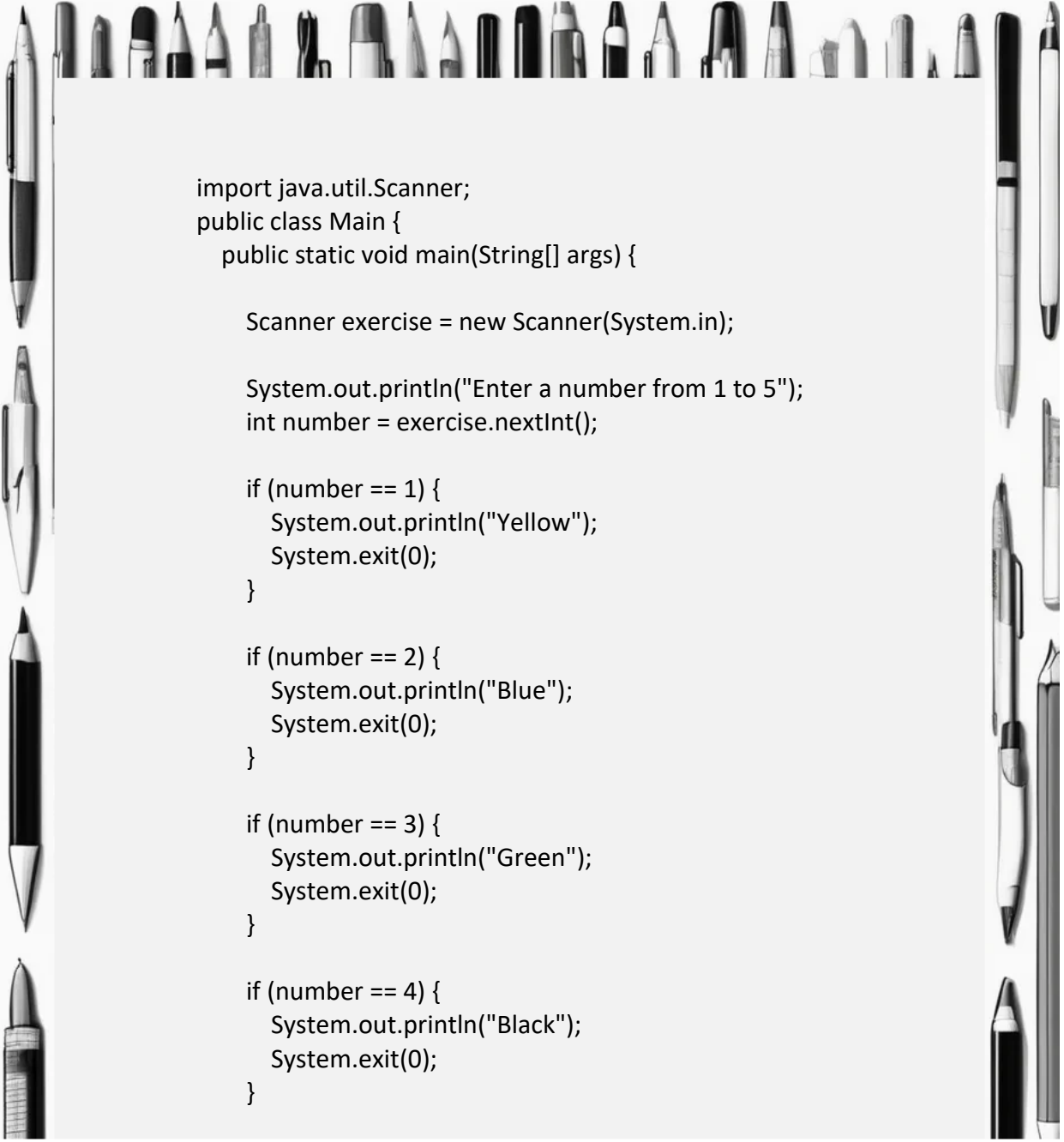
```
    if (day.equals("Friday") || day.equals("friday")) {  
        System.out.print("Reading");  
        System.exit(0);  
    } else System.out.println("Incorrect data entered");
```

//On Friday at the end, after the else we introduce the final line that will only be shown if none of the previous days were mentioned.

```
    }  
}
```

Exercise 3. Enter a number from 1 to 5 using the keyboard and, depending on the number, a text will appear on the program screen in a color: yellow, blue, green, black or white.

Solution:



```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner(System.in);

        System.out.println("Enter a number from 1 to 5");
        int number = exercise.nextInt();

        if (number == 1) {
            System.out.println("Yellow");
            System.exit(0);
        }

        if (number == 2) {
            System.out.println("Blue");
            System.exit(0);
        }

        if (number == 3) {
            System.out.println("Green");
            System.exit(0);
        }

        if (number == 4) {
            System.out.println("Black");
            System.exit(0);
        }
    }
}
```


```
if (number == 5) {  
    System.out.println("White");  
    System.exit(0);  
} else System.out.println("The entered data is incorrect");  
  
}  
}
```

💡 It's important that your conditionals are easy to read and understand. Avoid nesting too many if, else if, and else statements, as this can make the code become confusing. Instead, consider splitting the logic into separate functions if it becomes too complex.

💡 Take advantage of logical operators (&&, ||, !) and comparison operators (==, !=, >, <, >=, <=) to combine conditions and make your if statements more efficient. Make sure you understand how they work to avoid logical errors in your code.

Exercise 4. Enter a number using the keyboard that will refer to the temperature of a swimming pool. If it's less than 20, the program will say it's cold; for any other temperature, it will say it's hot. It can have decimal places.

Solution:



```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner(System.in);

        System.out.println("Enter the pool temperature");

        double temperature = exercise.nextDouble();

        if (temperature < 20) {

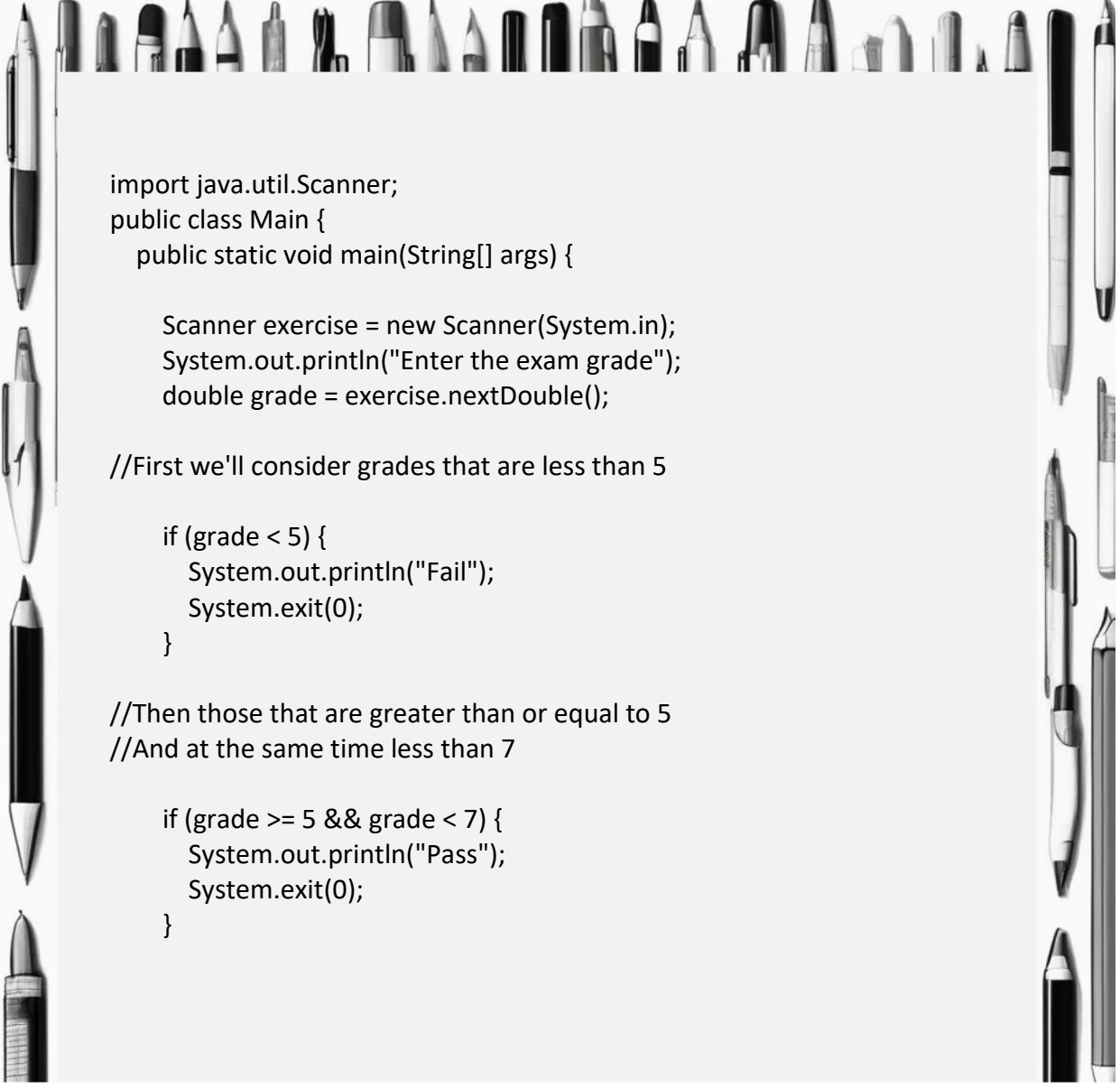
            System.out.println("Cold");

        } else System.out.println("Hot");

    }
}
```

Exercise 5. We are going to enter an exam grade using the keyboard. The program will tell us if the student has failed, passed, received a B, or received an A. If it's less than 5, it's a fail. Between 5 and 6.99 is a pass. Between 7 and 8.99 is a B, and greater than 9 is an A.

Solution:



```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

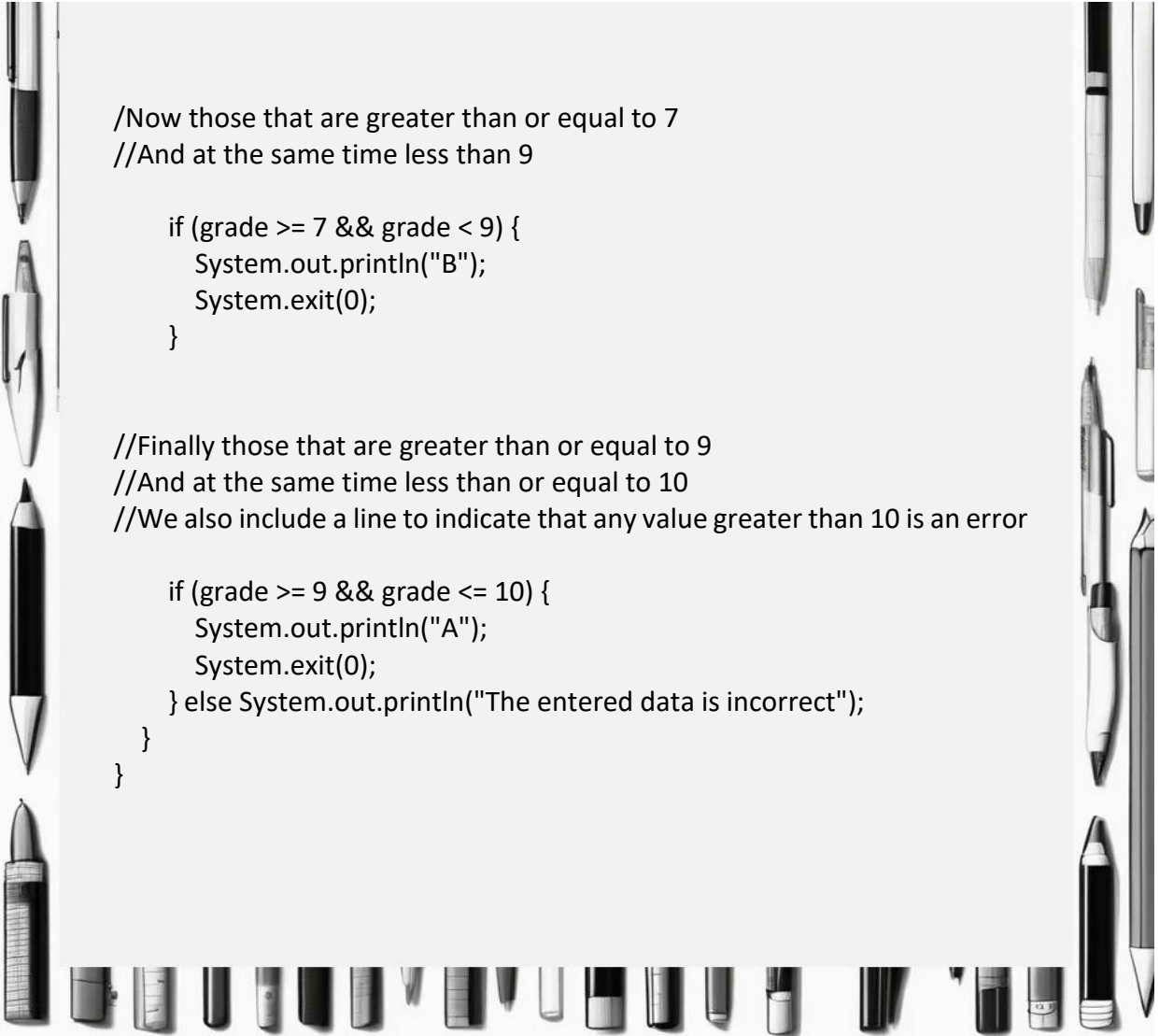
        Scanner exercise = new Scanner(System.in);
        System.out.println("Enter the exam grade");
        double grade = exercise.nextDouble();

        //First we'll consider grades that are less than 5

        if (grade < 5) {
            System.out.println("Fail");
            System.exit(0);
        }

        //Then those that are greater than or equal to 5
        //And at the same time less than 7

        if (grade >= 5 && grade < 7) {
            System.out.println("Pass");
            System.exit(0);
        }
    }
}
```



```
//Now those that are greater than or equal to 7  
//And at the same time less than 9
```

```
    if (grade >= 7 && grade < 9) {  
        System.out.println("B");  
        System.exit(0);  
    }
```

```
//Finally those that are greater than or equal to 9  
//And at the same time less than or equal to 10  
//We also include a line to indicate that any value greater than 10 is an error
```

```
    if (grade >= 9 && grade <= 10) {  
        System.out.println("A");  
        System.exit(0);  
    } else System.out.println("The entered data is incorrect");  
}
```



Review your code to eliminate redundant conditions. For example, if a condition has already been checked and handled, it's not necessary to check it again. This simplifies the code and improves performance.

Exercise 6. We are going to create a program that calculates a worker's salary. The first 160 hours are paid at 7.8 euros. Hours exceeding 160 are considered overtime and are paid at 12.2 euros.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner(System.in);

        //We create the salary variable. In this case it's necessary to initialize it.
        //Otherwise we couldn't use it in the last line of this program.
        //You can't ask the program to show a variable if it doesn't have a value.
        //Therefore we set it to zero, but it will change later depending on the data
        entered.

        double salary = 0;

        System.out.println("Enter the number of hours worked");

        int hours = exercise.nextInt();

        //First we make a calculation assuming there was no overtime.

        if (hours <= 160) {
            salary = (hours * 7.8);
        }
```

```
// In this case there was overtime
// Therefore we know there are always more than 160. So we will always pay
160 hours at 7.8 euros.
// To these 160 hours we now add the extra ones. To know how many extra
hours we worked, we subtract 160 from the total hours.
// Example. If the worker did 180 hours.  $180 - 160 = 20$ . Twenty are extra.
// Finally we simply add the normal hours plus the extra hours.
```

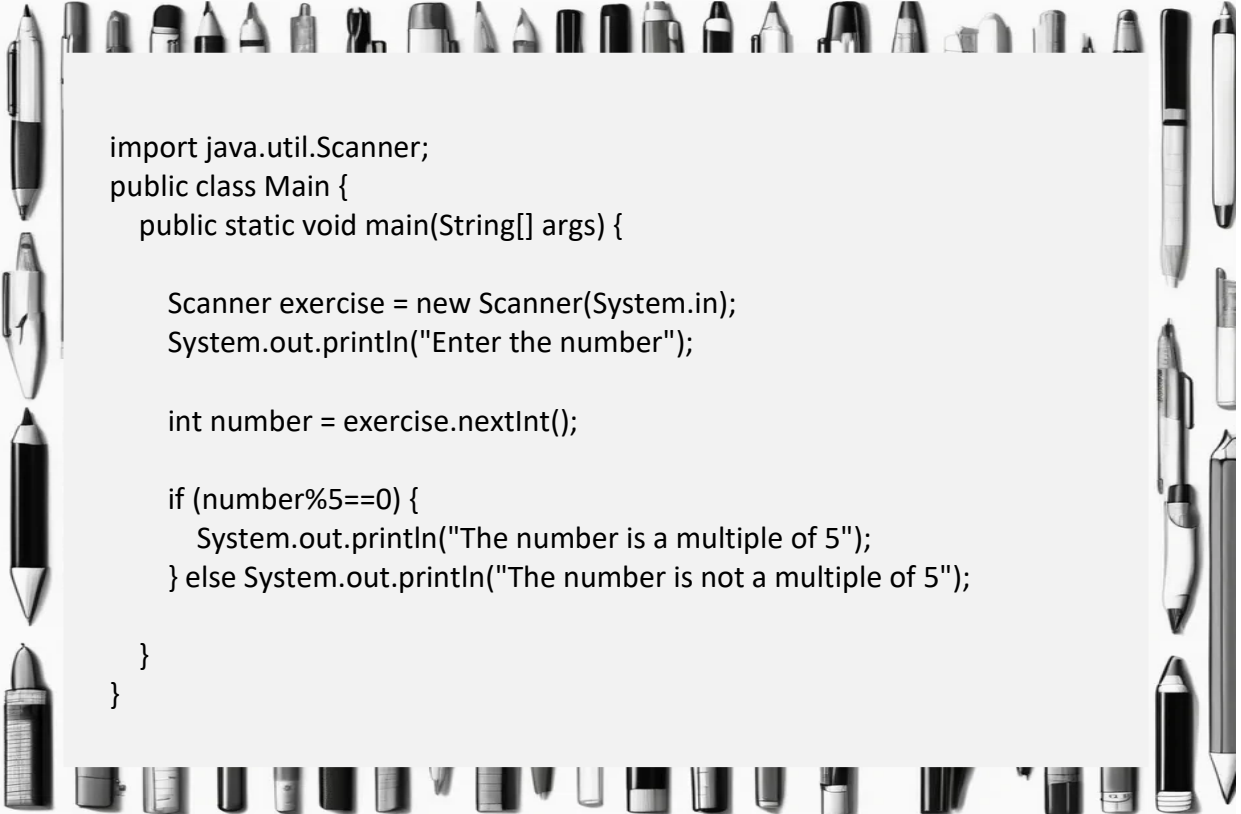
```
    if (hours > 160) {
        salary = (160 * 7.8) + ((hours-160) * 12.2);
    }

    System.out.println("The total salary is " + salary + " euros");
}
}
```

💡 Place the conditions that are more likely to be met at the beginning of your if blocks. This can improve the program's performance by reducing the number of checks needed in common cases.

Exercise 7. Enter a number using the keyboard and the program should tell us if it is a multiple of 5.

Solution:



```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

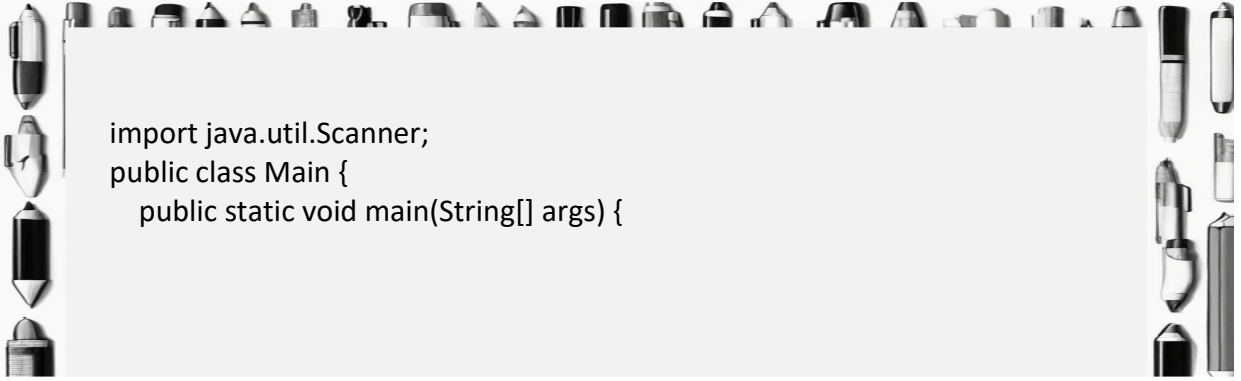
        Scanner exercise = new Scanner(System.in);
        System.out.println("Enter the number");

        int number = exercise.nextInt();

        if (number%5==0) {
            System.out.println("The number is a multiple of 5");
        } else System.out.println("The number is not a multiple of 5");
    }
}
```

Exercise 8. We enter a number using the keyboard and ask it to tell us if it is even or odd.

Solution:



```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
```

```
Scanner exercise = new Scanner(System.in);
System.out.println("Enter the number");

int number = exercise.nextInt();

if (number%2==0) {
    System.out.println("The number is even");
} else System.out.println("The number is odd");
}
```

Exercise 9. Create a program where you enter a three-digit number using the keyboard and the program will tell you if it's a palindrome number or not.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        //Create a scanner

        Scanner exercise = new Scanner(System.in);

        //Now we need to ask the user to enter the number

        System.out.println("Enter a number");
        int num = exercise.nextInt();
```

```
if (num<=99) {
    System.out.println("The number must have three digits");
}
if (num>999) {
    System.out.println("The number must have three digits");
}

//Calculating the hundreds is easy. We divide the number by 100.
//For example 356/100 = 3.56. Being an int variable it only shows the 3,
no decimals.

int hundreds = num/100;

//For the tens we'll do the same but first we must eliminate the hundreds.
//Example 678. If we could remove the 6, we'd have 78.
//If we could divide 78 by 10 we'd get 7. The hundred we need.
//So we multiply the hundreds by 100. In this case 6*100 = 600.
//We subtract 600 from our number; 678-600 = 78. 78/10 = 7.8 but being
an int = 7

int tens = ((num - (hundreds * 100))/10);


//We do the same to get the units.
//(Hundreds * 100) + (tens * 10) and we subtract it from our number.
//600 * 100 + 7*10 = 670. 678-670 = 8. The unit we need.

int units = (num - (hundreds * 100) - tens * 10);

if (units==hundreds) {
    System.out.println("The number is a palindrome");
} else System.out.println("The number is not a palindrome");
}
}
```


Exercise 10. Create a program that tells how many digits a number entered via keyboard has.

Solution:



```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner(System.in);

        System.out.println("Enter a number");
        int num = exercise.nextInt();

        if (num>9999) {
            System.out.println("The number is too big");
        }
        if (num<10) {
            System.out.println("The number has one digit");
        }
        if (num>=10&&num<100) {
            System.out.println("The number has two digits");
        }
        if (num>=100&&num<1000) {
            System.out.println("The number has three digits");
        }
        if (num>=1000&&num<10000) {
            System.out.println("The number has four digits");
        }
    }
}
```

Chapter 5.

Meeting Bud. Learning to use loops

I can't stop thinking about how quickly life can change. Fernando and Rich have been quite lucky, but I haven't been so fortunate. It seems the police found something on my computer, although they haven't specified anything yet. My lawyer says I'm facing several years in prison, but I don't even know what I'm being accused of.

While I await my trial, my new home is a small shared cell. The bed is very uncomfortable, everything smells terrible, and worst of all, there's no privacy to use the bathroom. It's going to be difficult to get used to. In real life, your head is full of stimuli from everywhere. It stays busy, you think little, and you don't get bored. Here there's absolutely nothing to entertain yourself with. I really miss my phone. I can barely communicate with my friends and family, only with the closest ones. I'd like to tell everyone that I'm innocent and that I haven't done anything wrong. I don't know what people will think of me, but surely nothing good.



When you have too much free time, you can't stop thinking about things. Obviously, I think about what happened. You don't have to be very smart to realize that Chani is the guilty one. He mysteriously disappeared a few days before the police showed up. During the party, our computers were inside his cabinet, and therefore, he could have accessed any of them. Besides, it's very strange that nobody can locate him anywhere and that his phone is disconnected.

I wonder why he would do it. I mean, why frame me or Rich? We were friends. We always had a good time and never had any kind of problem. I guess he needed someone to blame and, in the end, it fell on me. I'd like to know where he is and if he made any money from his crime. I want to understand why he did it and what kind of trouble he's involved in. I'm aware that I might never know, but still, I have the feeling that someday I will.

It's obvious that I'm quite worried, as I don't know how long I'll have to be here. Every minute is torture and I wouldn't like to have to spend much more time here. Deep down, I trust that everything will be cleared up. I have nothing to hide and, additionally, I have Rich's favorable testimony. He can help me prove that Chani is the real culprit.



My family has gotten a good lawyer, who is preparing the case and will soon inform me of the strategy to follow and what sentence I could face. Overall, I feel quite supported and that allows me to keep going.

The visits brighten my days and break the monotony. Although it affects me to see their worried faces, it's comforting to know that I have people behind me fighting from the outside. I wouldn't want to be completely alone. In here I'm defenseless and it's impossible for me to get evidence of my innocence. Now all I can do is trust that other people will get me out of this situation.

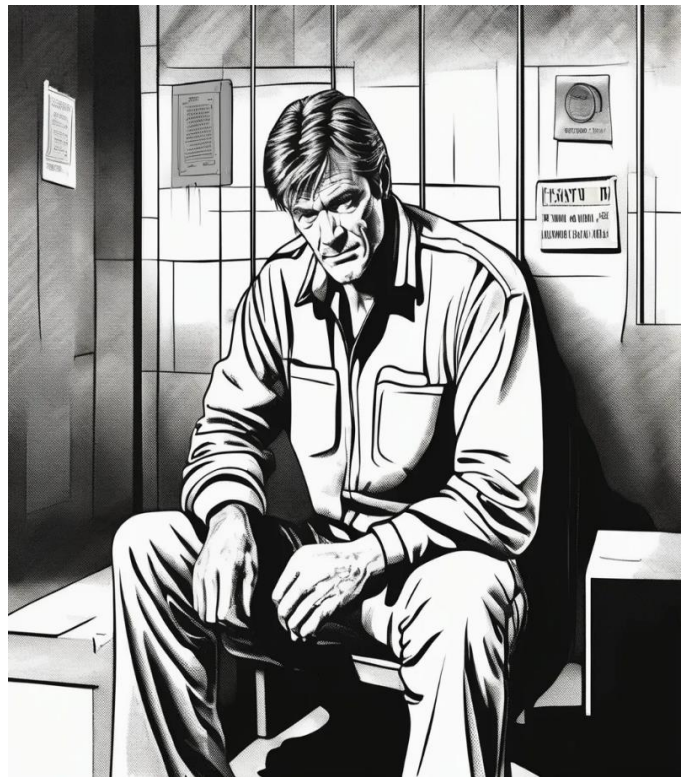
Now that you know my mood is good, perhaps you're wondering what life in prison is like. I don't just mean how it feels to be inside the cell; I've already explained that. I'm talking about relationships with other inmates, the food, the yard, physical security...

I'm not going to lie. The first day I was very scared. From the moment I entered prison, I felt a very strange sensation in my body. I could barely breathe. I felt a strong pressure in my chest that barely allowed me to speak. In movies they always say you have to pretend to be tough, but I couldn't pretend. I just walked looking at the ground until I reached my cell.

Fortunately, that feeling didn't last long. Specifically, it vanished when I met my cellmate, a man from Valencia called "Bud." Well, I think it's obvious that's not his real name. He introduced himself that way and I never asked anything more.

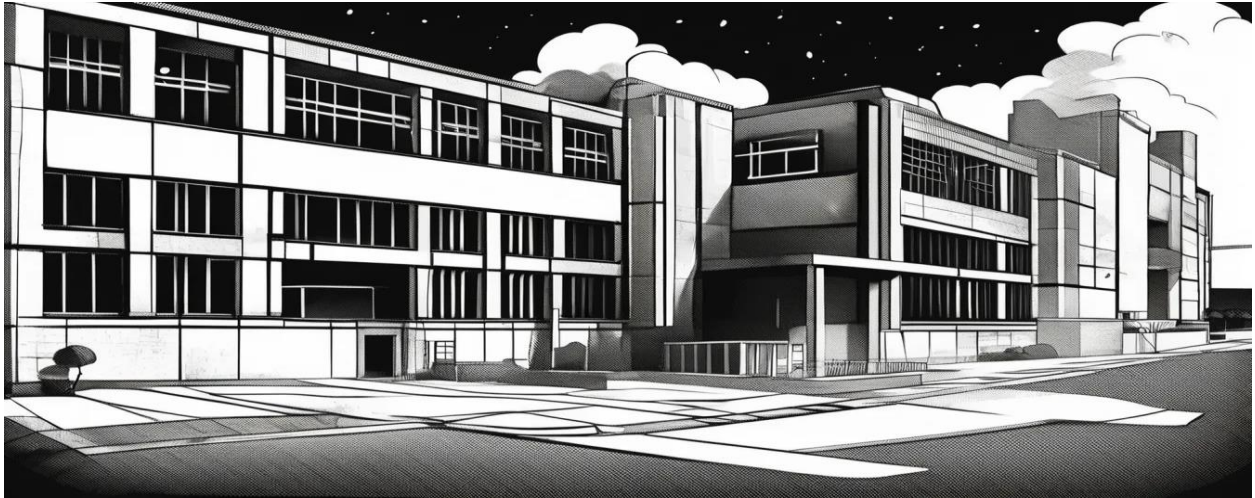
Bud has been in the same cell for the last 8 years and is highly respected by everyone there. He knows many people, both inmates and prison officers. So being his friend can offer me practically total immunity. As long as I don't look for trouble, people will leave me alone.

Quickly, Bud became interested in my story. He wanted to know how I had ended up there. I explained the situation and he was understanding. Later, he informed me that he was there because he tried to rob a jewelry store. He didn't want to give more details and would never give them to me.



We continued chatting and I ended up mentioning that I was studying computer engineering. He asked if I knew how to program in Java and I told him I had some knowledge. Then, he explained that, to reduce his sentence, he helps the prison officers with security software. This allows him to have a laptop from time to time.

I thought I was very lucky. My stay in the cell would be much more pleasant if I could have a computer and internet access. Besides being able to contact my family and friends, I could even learn Java. Being friends with the prison guards can have great advantages.



I think Bud read my mind because he quickly told me that he could help me improve my programming skills. After giving me a small improvised test, he could more or less calculate my level. He asked me if I knew what loops were. The truth is I had no idea, so I told him no.

That's how I got a private Java tutor in prison. I felt very lucky for it and didn't want to disappoint my new mentor. So I took it very seriously. The lesson on loops had begun and I'm going to tell you everything that Bud explained to me.

Loops

Definition, purpose and importance of loops in programming

A loop in programming is a way to make your program repeatedly execute a block of code while a certain condition is met. Loops are a fundamental tool not only in Java but in any programming language. This is because they allow us to automate repetitive tasks efficiently.

In Java, there are mainly three types of loops: for, while, and do-while. Each of these loops has its own syntax and specific application, but they all share the same essential purpose: to repeatedly execute a set of instructions.

I will explain the three types of loops, but I advise you to focus only on the first two, for and while. I believe that if you're taking your first steps in Java, at first it's essential to master a few basic things instead of trying to cover everything. However, it's important that you're aware that there are three. If you really like programming and it becomes part of your life, at some point you'll come across do-while loops and I don't want them to catch you by surprise.

Introduction to loops

- **for:** Used when you know in advance the number of iterations that need to be performed.
- **while:** Used when you don't know for certain how many times a block of code should be repeated and the termination condition depends on some variable that is evaluated in each iteration.
- **do-while:** Similar to while, but guarantees that the code block is executed at least once, since the condition is evaluated after the first execution.

After reading this brief definition, you probably still don't fully understand what we're talking about, although you're getting an idea. Don't worry though, because when we look at practical examples, everything will become clear. However, before that, it's necessary to list the main reasons why loops are important.

Automation of Repetitive Tasks: Loops allow us to automatically execute repetitive tasks; this way, we don't need to write the same code multiple times. This will be especially useful for operations like processing list elements, reading files, or generating number sequences.

Efficiency and Performance: If we use loops appropriately, we can significantly improve the efficiency and performance of our programs. We'll avoid redundancy and thus reduce the amount of code needed to perform repetitive tasks. This will optimize the use of system resources.

Facilitates Code Maintenance: Code that uses loops is generally easier to understand, maintain, and modify. If the logic of a repetitive operation needs to be changed, you only need to modify the code block within the loop instead of changing multiple instances of the same code.

Handling Collections and Data: Loops are particularly useful for handling data collections, such as arrays and lists. They allow you to traverse the elements of these collections and perform operations on each of them systematically.

Loop for

A for loop in Java is a way to repeat a block of code a specific number of times. It's useful when we know how many times we want the code to execute.

A for loop has three main parts:

- Initialization: Sets up a variable to start.
- Condition: While this condition is true, the loop will continue executing.
- Update: Changes the variable after each loop repetition.

Here's the basic structure of a for loop:

```
for (initialization; condition; update) {  
    // Code to repeat  
}
```

Example 1: Display numbers from 1 to 5.

```
for (int i = 1; i <= 5; i++) {  
    System.out.println(i);  
}
```

Let's analyze this exercise carefully. The first thing we need to do is declare a variable. As you can see in the example, we've named it "i". It would work the same regardless of the name, but you'll always see it written like this. Therefore, I recommend you do the same.

The second thing we see is that the variable i must be less than or equal to five.

Lastly, i++ means that after each repetition, i increases by 1.

If we had wanted to do a countdown, to show the numbers from 5 to 1, we would do it as follows:

```
for (int i = 5; i >= 0; i--) {  
    System.out.println(i);  
}
```

Again, we start by declaring the variable "i" and, additionally, we assign it the value of the first data we need, in this case, a 5. Then we say that the values we want to show must be equal to or greater than 0. Finally, we use i-- so that i decreases one by one.

Loops aren't just for showing lists of numbers. Little by little you'll see that they can be used for many things. Let's look at another example:

Let's create a program that shows "Hello" three times in a row:

```
for (int i = 0; i < 3; i++) {  
    System.out.println("Hello");  
}
```

We need a loop that repeats exactly three times. Perfect, since we know exactly the number of times, we use a for. We follow the same structure as before; in this case, we say that i starts at 0 and we need i to repeat three times, so:

The first i will be = 0

The second i will be = 1

The third i will be = 2

Therefore, we need the condition to be that i is less than or equal to 2. We could have done it in two ways and it would be exactly the same:

- i < 3
- i <= 2
-

In this case, we care little whether our variable starts at 0, 1, or any other number, for example, 33. We simply need the loop to repeat three times. It could have been like this:

```
for (int i = 33; i <= 35; i++) {  
    System.out.println("Hello  
");  
}
```

Now that we have a loop that repeats three times, we simply use a line of code to print each time.

Things will get a bit more complicated when we look at the exercises, but I hope you're understanding the basics. The for loop is usually quite simple to learn, since you just have to follow specific steps. You create a variable with the first value, define the second, and then end with `i++` or `i--`, depending on each scenario. The while loop, on the other hand, is not so structured and, although at first it might seem even simpler than the for loop, when you face complex exercises is when you realize that applying while is not as intuitive as you thought. Even so, you'll see how you'll learn it quickly without problems.

Loop while

A while loop follows this basic structure:

```
while (condition) {  
    // Code to repeat  
}
```

Seems very simple, right? Well, the truth is that it is. Maybe I scared you a bit before by telling you it's more difficult than the for loop. Actually, when we explain the basic exercises, everything seems very simple, but later, when the statements become more complicated, that's when you'll have to think things through to find the answer you're looking for. But let's not get ahead of ourselves, let's look at an example:

Let's use a while loop to print numbers from 1 to 5.

```
int i = 1; // Initialization  
while (i <= 5) { // Condition  
    System.out.println(i);  
    i++; // Update  
}
```

We create and assign a value to an int type variable outside our loop. For now, and before starting the loop, the variable still has a value of 1. But variables can change value as the program progresses. Remember that when executing, the program reads the lines of code from top to bottom.

When the loop executes for the first time, i equals one, and it will show the line of code `System.out.println(i);` in this case "1".

But what happens right after? We find an `i++`, so now i equals 2. Since we said the loop doesn't close until i is equal to or greater than 6, it will show `System.out.println(i);` again and in this case we'll see "2".

What happens in the next iteration? i equals 3... the cycle repeats again. This continues until reaching 6, and at that moment the loop closes and won't show it.

Shall we raise the level a bit?

I'll use an exercise similar to the previous one, as it will be familiar to you. In this case, we'll create a loop from 1 to 10. But we're going to put an if inside. We'll show a text that says: "The value is greater than 6". We need it to show i and then, next to it, the text I mentioned, for i = 7, 8, 9, and 10.

```
public class Main {  
    public static void main(String[] args) {  
        int i = 1;  
        while (i <= 10) {  
            if (i<=6) {  
                System.out.println(i);  
            } else System.out.println(i + " The value is greater than 6");  
            i++;  
        }  
    }  
}
```

It's simple. Once we had the loop done, we needed one thing to happen for the first 6 numbers and something different for the next 4. The first six only show the value of *i*, while the next 4 show *i* and also the message. You can try to repeat the exercise on your own using a for loop instead of the while I used. It's even easier.

do while loops

A do-while loop is a structure that repeats our instructions while a condition is met. What makes it different from other loops is that it always executes the code at least once, even if the condition is false from the beginning.

- First, the code inside the loop is executed.
- Then, the condition is evaluated.
- If the condition is true, the code is executed again. If it's false, the loop ends.

This type of loop is useful when we want to make sure the code block is executed at least once, regardless of the condition. Here's an example:

```
int number = 1;
```

```
do {
```

```
    System.out.println(number);    // Prints the current number
```

```
    number++;                      // Increases the number by 1
```

```
} while (number <= 5);            // Repeats while the number is less than or equal to 5
```

- First, it prints the number 1.
- Then it verifies the condition `number <= 5`. Since it's still true, the loop continues.
- The process repeats until the number reaches 6, at which point the condition becomes false and the loop ends.

Break Concept

When the program encounters a break, it exits the loop and continues with the next line of code outside the loop. The use of break is important mainly for two reasons.

- The first is that it allows us to exit a break prematurely. Something that you'll see will be necessary to do in many situations. Without looking too far, in several of the exercises at the end of the topic.
- The second is that it improves program efficiency by saving resources. If you know you've already found what you were looking for, there's no need to go through the loop until the end.

Furthermore, break will help us simplify the program's logic; it makes the code cleaner, simpler, and easier to understand. Finally, it will be an elegant solution for handling exceptional cases. In some cases, within a loop you might encounter exceptional situations such as errors or invalid conditions that require immediately closing the loop. You won't find anything simpler or faster than a break.

Example of using break

Imagine that you create an infinite loop using a while loop. We could do it in the following way:

```
int a = 1;
while (a >= 0) {
    a++;
    if (a > 10) {
        System.out.println(a);
    }
}
```

According to that loop, as long as a is greater than 0, it will keep repeating. Our program would be infinitely showing number after number, one by one on our screen. However, we want it to show only the number after 10 and then for the loop to close.

```
int a = 1;
while (a >= 0) {
    a++;
    if (a > 10) {
        System.out.println("First number greater than 10 is = " + a);
        break; // Exits the loop when the first number greater than 10 is found
    }
}
```

By now you will have realized that this exercise could have been done in another way. Not only that, but it can be done in many different ways. The code is just a tool that the programmer has to solve problems. It's at our disposal and can be used as best suits us in each situation.

Concept of continue

A continue is used to skip to the next iteration of the loop, omitting the code in between within the loop for the current iteration. When our program encounters a continue instruction, the loop doesn't stop, but instead jumps to the next iteration. This is useful when you want to skip certain iterations under specific conditions without stopping the loop completely.

Example of continue

Imagine you have a loop that counts from 1 to 10, but you want to skip the number 5. You can use continue to achieve this.

```
public class Main {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 10; i++) {  
            if (i == 5) {  
                continue; // Skips the rest of the loop when i is equal to 5  
            }  
            System.out.println(i);  
        }  
    }  
}
```

Both concepts, break and continue, are useful tools for controlling the execution flow within loops in Java, allowing you to manipulate the iteration according to your specific needs.

String Manipulation

Although string manipulation could perfectly well have its own topic, to avoid making things excessively long I've decided to show you how this concept works in this topic. Additionally, from now on you'll see two specific methods in the exercises and, so they don't catch you by surprise, I'm going to explain what they are and what they're used for. This will notably enrich your knowledge of the syntax.

charAt() and length() Methods

- The charAt() Method

The charAt() method in Java is used to get the character at a specific position within a text string. It takes an index as an argument and returns the character at that position within the string.

Syntax:

```
char charAt(int index)
```

index: Is the index of the character you want to obtain. The first character of the string has an index of 0, the second has an index of 1, and so on.

Example:

```
String text = "Hello World";  
char character = text.charAt(0);  
System.out.println("The character at position 0 is: " + character); // Output: 'H'
```

In this case, charAt(0) returns the first character of the string "Hello World", which is 'H'. If you use another index, you'll get the character at that position, for example charAt(5) would return 'W'.

- The length() Method

The length() method in Java is used to know how many characters a text string has. It returns an integer that represents the length of the string.

Syntax:

```
int length()
```

Example:

```
String greeting = "Hello";
```

```
int length = greeting.length();
```

```
System.out.println("The length of the string is: " + length); // Output: 4
```

It's not complicated at all; in the following exercises you'll encounter these methods several times. As soon as you start working with them, you'll see that they are very simple to use.

Summary of what you've learned

A loop allows a block of code to be executed repeatedly while a specific condition is met. This technique is essential for efficiently automating repetitive tasks. In Java, the main types of loops are for, while, and do-while, each with its particular structure and use.

The for loop is used when the number of times the code should be executed is known in advance. In contrast, the while loop is used when it's not known how many times the code block will be repeated, as the termination condition depends on a variable evaluated in each iteration. Meanwhile, the do-while loop is similar to while, but guarantees that the code block is executed at least once, evaluating the condition after the first execution.

Loops are necessary for several reasons:

- First, they automate repetitive tasks, allowing code to be executed multiple times without having to write it repeatedly.
- Second, they improve efficiency and performance by reducing redundancy and optimizing resource use. They also facilitate code maintenance, as it's easier to modify a block of code within a loop than multiple instances of the same code.
- Finally, they are especially useful for handling data collections, such as arrays and lists, allowing systematic operations on each element.

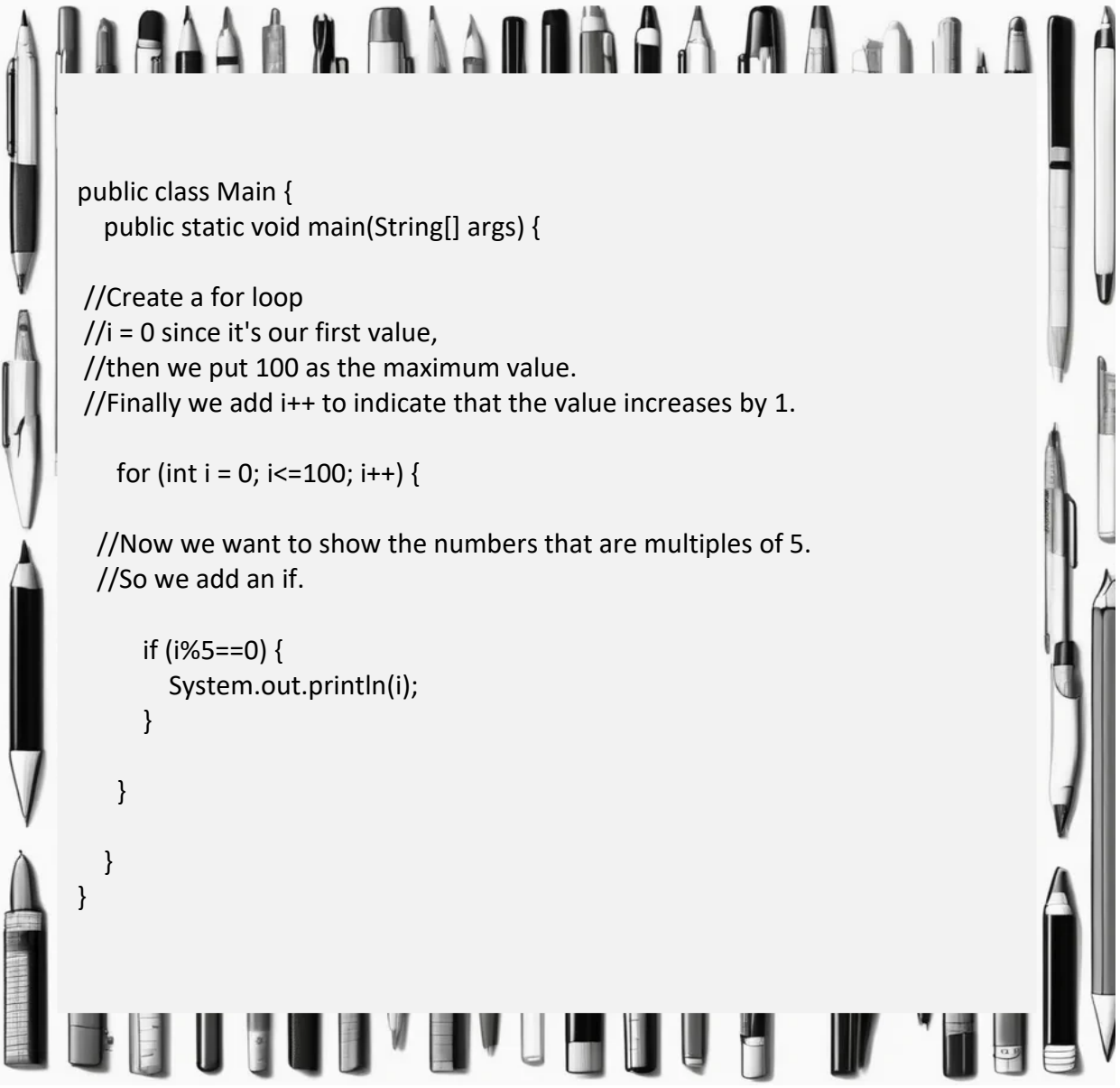
Additionally, in this topic we've seen two very useful commands when working with conditionals.

- The concept of "break" is used to exit a loop before completing all iterations. This is useful for improving Program efficiency and simplifying code logic, making it cleaner and easier to understand.
- On the other hand, "continue" is used to skip to the next loop iteration, omitting the intermediate code for the current iteration. Both concepts are valuable tools for controlling execution flow within loops in Java and manipulating iteration according to the Program's specific needs.

Practical exercises

Exercise 1. Create a program that shows multiples of 5 between the numbers 0 and 100 using a for loop.

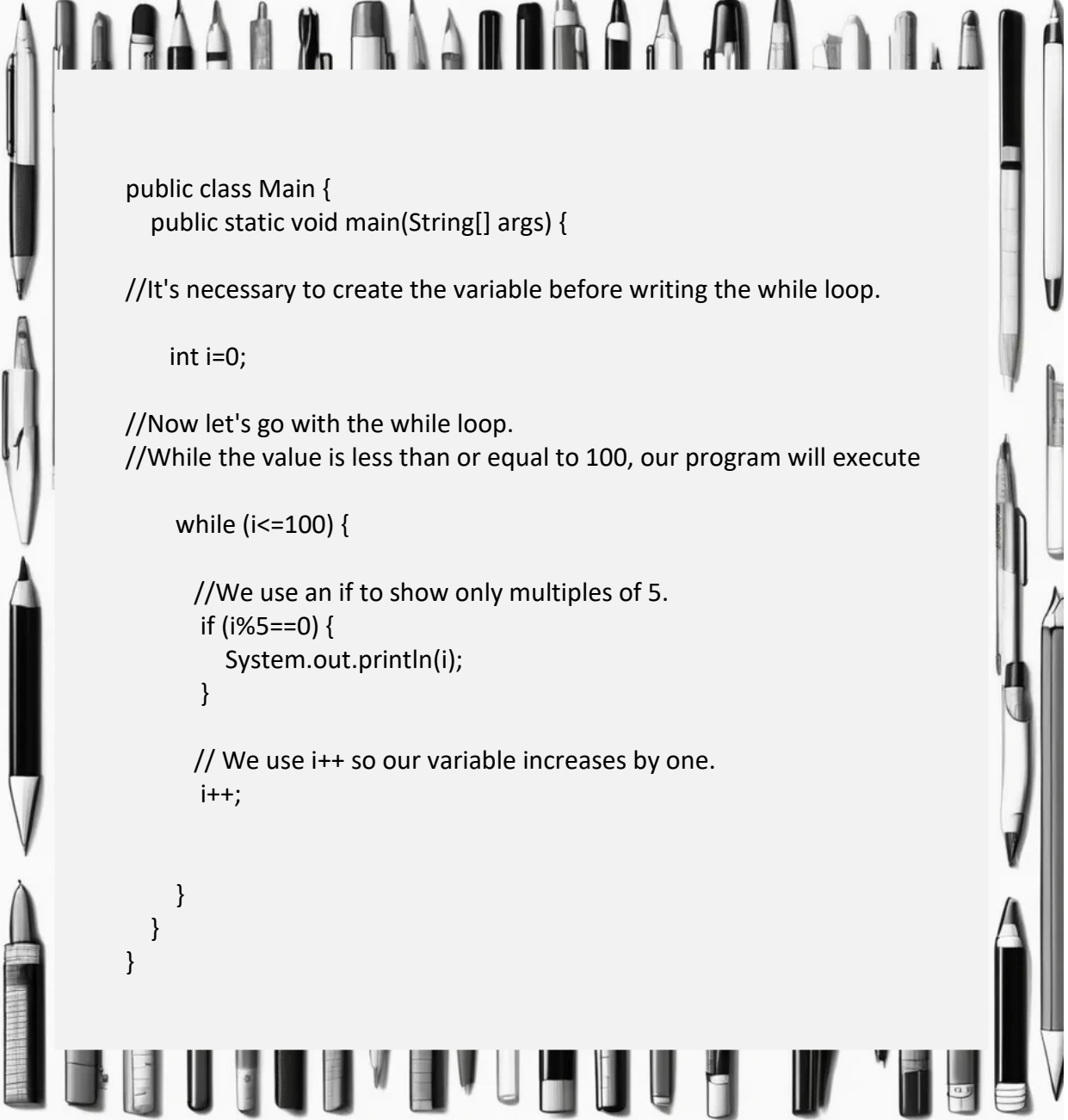
Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        //Create a for loop  
        //i = 0 since it's our first value,  
        //then we put 100 as the maximum value.  
        //Finally we add i++ to indicate that the value increases by 1.  
  
        for (int i = 0; i<=100; i++) {  
  
            //Now we want to show the numbers that are multiples of 5.  
            //So we add an if.  
  
            if (i%5==0) {  
                System.out.println(i);  
            }  
        }  
    }  
}
```

Exercise 2. Create a program that shows multiples of 5 between the numbers 0 and 100 using a while loop.

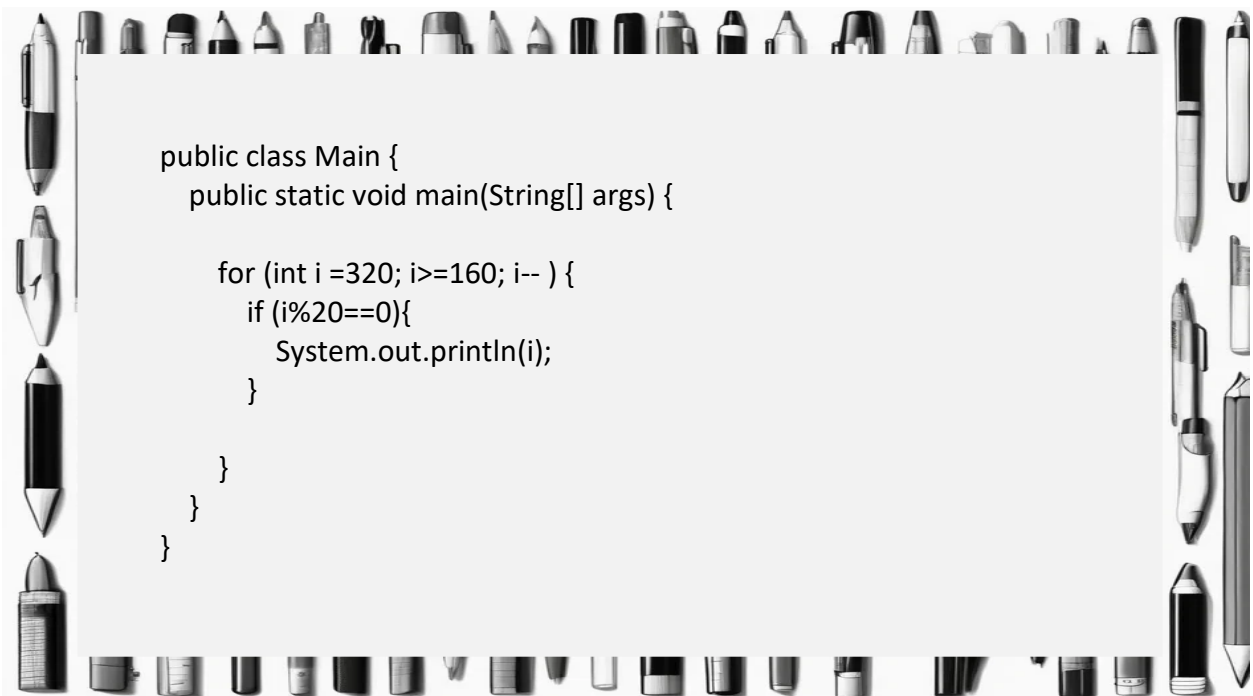
Solution:




```
public class Main {  
    public static void main(String[] args) {  
  
        //It's necessary to create the variable before writing the while loop.  
  
        int i=0;  
  
        //Now let's go with the while loop.  
        //While the value is less than or equal to 100, our program will execute  
  
        while (i<=100) {  
  
            //We use an if to show only multiples of 5.  
            if (i%5==0) {  
                System.out.println(i);  
            }  
  
            // We use i++ so our variable increases by one.  
            i++;  
  
        }  
    }  
}
```

Exercise 3. Show numbers between 320 and 160 counting down by twenty using a for loop.

Solution:

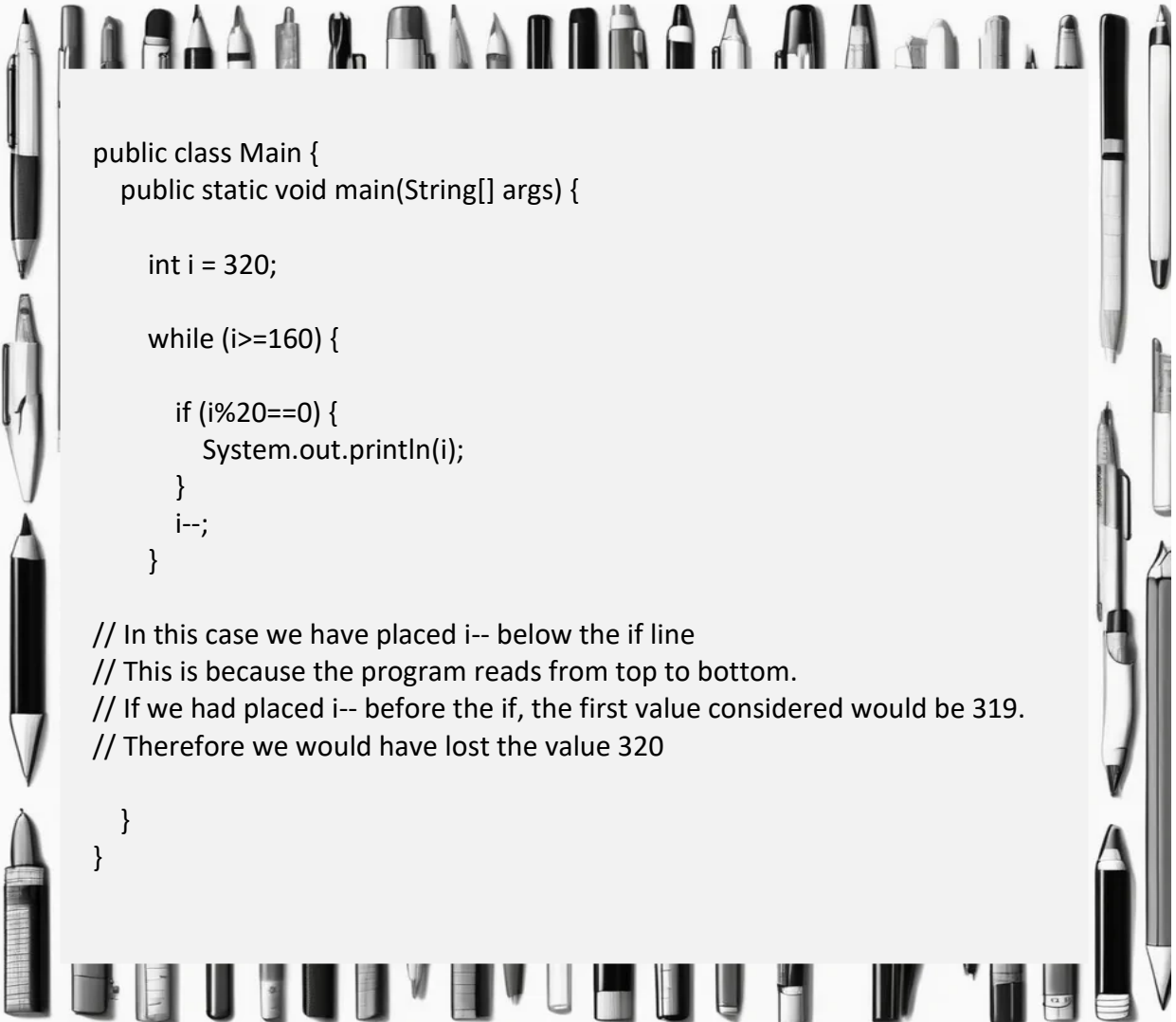


```
public class Main {  
    public static void main(String[] args) {  
  
        for (int i =320; i>=160; i-- ) {  
            if (i%20==0){  
                System.out.println(i);  
            }  
        }  
    }  
}
```

 **Initialize the control variable:** The control variable in a Java loop is a variable that is used to count or keep track of the number of times the loop has executed. It helps decide when the loop should stop.

Exercise 4. Show numbers between 320 and 160 counting down by twenty using a while loop.

Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        int i = 320;  
  
        while (i>=160) {  
            if (i%20==0) {  
                System.out.println(i);  
            }  
            i--;  
        }  
  
        // In this case we have placed i-- below the if line  
        // This is because the program reads from top to bottom.  
        // If we had placed i-- before the if, the first value considered would be 319.  
        // Therefore we would have lost the value 320  
  
    }  
}
```



Update the control variable: Modify the control variable in each iteration so that the loop can advance and eventually end.

Exercise 5. Create a program that asks for a password to open a suitcase. The password is 7678. If it's correct, it opens. But if after 5 attempts it's not achieved, it locks.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        // First we create our password.

        int password = 7678;

        //We create a scanner since we'll have to input data via keyboard.
        Scanner exercise = new Scanner(System.in);

        //We use a for to have 5 attempts.
        //We add i++ so the attempts go one by one.

        for (int i = 0; i<5; i++ ) {

            System.out.println("Enter a password");

            //We input data via keyboard

            int attempt = exercise.nextInt();

            //Now we use our conditional.

            if (attempt == password) {
                System.out.println("Suitcase opened");
            }
        }
    }
}
```

```
//It's necessary to put a System.exit if we guess the password correctly
//This way the program will close.
```

```
    System.exit(0);
```

```
    } else System.out.println("Incorrect data");
  }
```

```
//Outside the loop, after the five attempts, we add the following line:
    System.out.println("You ran out of attempts");
```

```
  }
}
```

Exercise 6. Create a program that shows the multiplication table for 8.

Solution using for:

```
public class Main {
    public static void main(String[] args) {
        for (int i=1; i<=10; i++ ) {
            System.out.println(i + " x " + 8 + " = " + (i*8));
        }
    }
}
```


Solution using while:

```
public class Main {  
    public static void main(String[] args) {  
        int i = 0;  
        while (i<10) {  
            i++;  
            System.out.println(i + " x " + 8 + " = " + (i*8));  
        }  
    }  
}
```

Exercise 7. Create a program that calculates the arithmetic mean of numbers entered via keyboard until a negative number is entered. (The negative number doesn't count for the average).

Solution:

```
import java.util.Scanner;  
public class Main {  
    public static void main(String[] args) {  
  
        Scanner exercise = new Scanner(System.in);  
  
        //The arithmetic mean is the division of the total sum by the number of  
        //values.  
        //So we're going to create two variables.  
  
        double totalSum = 0;  
        double totalData = 0;
```

//We need to create a variable that holds the value each time we enter a number.

```
int number = 0;
```

//We create the average variable.

```
double average = 0;
```

//We assign zero value to all variables since that's what they're worth for now.

//We create the while only for when our entered number is greater than 0.

```
while (number>=0) {
```

```
    System.out.println("Enter a number");  
    number = exercise.nextInt();
```

// This while considers all entered numbers.

// But the last one has negative value and we don't want to use it.

// So we put in an if to only count positive values.

```
    if (number>=0) {  
        totalData++;  
        totalSum = totalSum + number;  
    }  
}
```

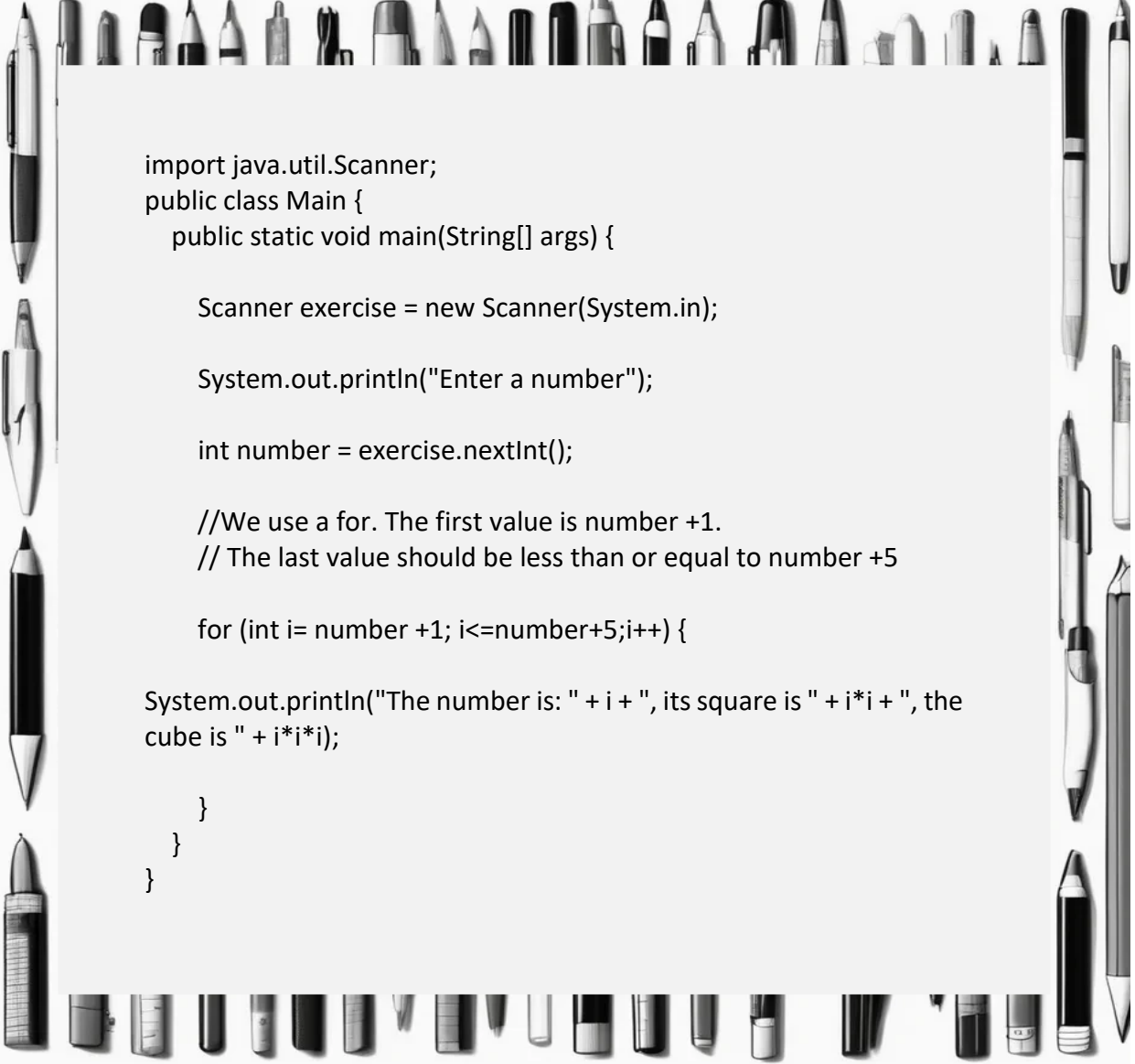
```
average = totalSum/totalData;
```

```
System.out.println("Total sum is = " + totalSum);  
System.out.println("The number of digits entered is " + totalData);  
System.out.println("The average is: = " + average);
```

```
    }  
}
```

Exercise 8. Create a program that shows the square and cube of the 5 numbers following a number entered via keyboard.

Solution:



```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner(System.in);

        System.out.println("Enter a number");

        int number = exercise.nextInt();

        //We use a for. The first value is number +1.
        // The last value should be less than or equal to number +5

        for (int i= number +1; i<=number+5;i++) {

            System.out.println("The number is: " + i + ", its square is " + i*i + ", the
            cube is " + i*i*i);

        }
    }
}
```

Exercise 9. Create a program where we enter 10 numbers via keyboard and it tells us how many are even and how many are odd.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner (System.in);

        // We create two variables to count the number of even and odd
        numbers.

        int evens = 0;
        int odds = 0;

        // We create a for loop that allows us to enter ten data points

        for (int i= 0; i<10;i++) {

            System.out.println("Enter a number");
            int data = exercise.nextInt();

            // Now using an if, we make it add 1 to either evens or odds
            // It can be done in two ways. Adding ++ or doing: variable = variable +1

            if (data%2==0) {
                evens = evens +1;
            } else odds++;

        }
    }
}
```

```
//Now we simply show the total sum of both even and odd numbers.
```

```
System.out.println("Total even numbers: " + evens);
```

```
System.out.println("Total odd numbers: " + odds);
```

```
}
```

```
}
```

Exercise 10. We make exercise 5 a bit more complicated. Create a program that asks for a password to open a suitcase. The password is 7678. If it's correct, it opens. But if after 5 attempts it's not achieved, it locks. If you enter a number with more or less than 4 digits, the program will tell you that the password must have 4 digits and won't count it as an attempt.

Solution:

```
import java.util.Scanner;
```

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        int password = ****;
```

```
        int attemptsUsed = 0;
```

```
        Scanner exercise = new Scanner(System.in);
```

```
while (attemptsUsed<5) {  
  
    System.out.println("Enter a password");  
  
    int attempt = exercise.nextInt();  
  
    if (attempt > 999 && attempt < 100000 ) {  
        attemptsUsed ++;  
    }  
  
    if (attempt <= 999 || attempt >= 100000 ) {  
        System.out.println("The password must have four digits");  
    }else  
  
    //Now we use our conditional.  
  
    if (attempt == password) {  
        System.out.println("Suitcase opened");  
  
        //It's necessary to put a System.exit if we guess the password correctly  
        //This way the program will close.  
  
        System.exit(0);  
  
    } else System.out.println("Incorrect password");  
    }  
  
    //Outside the loop, after the five attempts, we add the following line:  
  
    System.out.println("You ran out of attempts");  
  
    }  
}
```

Exercise 11. Create a program where a base and an exponent are entered, and that shows the result. Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner (System.in);

        System.out.println("Enter the base");
        int base = exercise.nextInt();

        System.out.println("Enter the exponent");
        int exponent = exercise.nextInt();

        //We also create a result variable that for now has the same value as the base.

        int result = base;

        // When we raise something to the square we multiply base by base, so one
        // multiplication.
        // When raising it to the cube we multiply base by base by base. Two
        // multiplications.
        // When raising it to 4, it's three operations. So always one less than the
        // exponent.
        // Therefore this will be our for.

        for (int i = 0; i<exponent -1;i++) {
            result = result*base;
        }

        System.out.println("The result is " + result);

    }
}
```

Exercise 12. Create a program where we enter a number via keyboard and it tells us whether it is prime or not.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner (System.in);
        System.out.println("Enter a number");
        int num = exercise.nextInt();

        //A prime number is one that can only be divided by 1 and itself.
        //Therefore we're going to make a loop with an int i, that goes one by one.
        //From 1 to the number itself.
        //We make our number divide by i each time
        //We create an int counter.
        //Each time our number is divisible by i, the counter will add 1.
        //If the counter is 2 it means it's divisible by 1 and itself. Prime.
        //If the counter is greater than two then it won't be prime.

        int counter = 0;

        for (int i = 1; i<=num; i++) {

            if(num%i==0) {
                counter++;
            }
        }

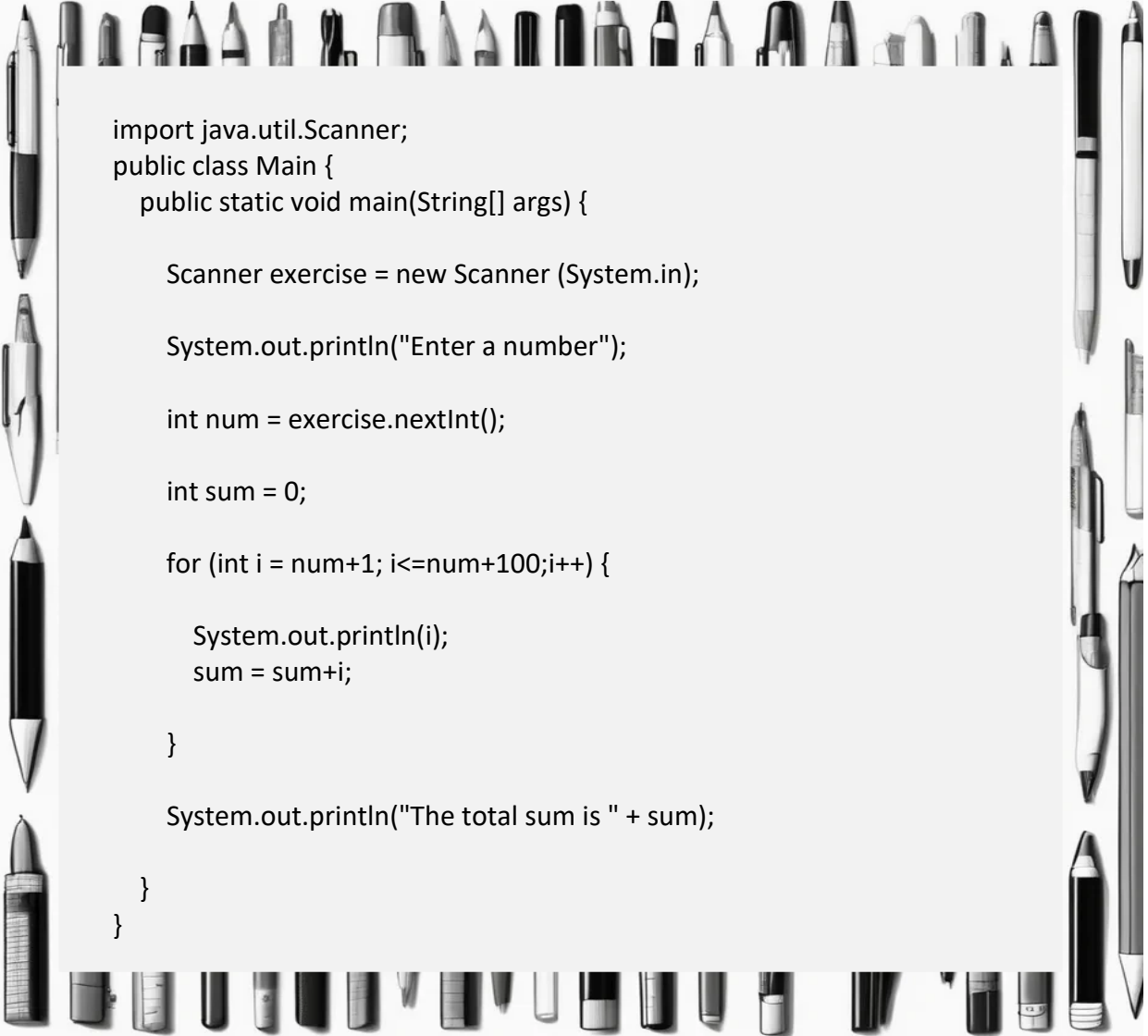
        if (counter <=2) {
            System.out.println ("It's prime");
        } else System.out.println ("It's not prime");

    }}

```


Exercise 13. Create a program that adds the next 100 numbers to a number entered via keyboard.

Solution:



```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner (System.in);

        System.out.println("Enter a number");

        int num = exercise.nextInt();

        int sum = 0;

        for (int i = num+1; i<=num+100;i++) {

            System.out.println(i);
            sum = sum+i;

        }

        System.out.println("The total sum is " + sum);


    }
}
```



Avoid infinite loops: Verify that the loop condition will change at some point so that the loop doesn't run indefinitely.

Exercise 14. Create a program that asks for two numbers. Then, show the numbers between them, starting with the first one and advancing by 7.

Solution using for:



```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner (System.in);

        System.out.println("Enter number a");

        int numA = exercise.nextInt();

        System.out.println("Enter number b");

        int numB = exercise.nextInt();

        for (int i = numA; i<=numB;i+=7) {

            System.out.println(i);

        }
    }
}
```

Solution using while

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner (System.in);

        System.out.println("Enter number a");

        int numA = exercise.nextInt();

        System.out.println("Enter number b");

        int numB = exercise.nextInt();

        int currentValue=numA;

        while (currentValue<=numB) {

            System.out.println(currentValue);
            currentValue += 7;

        }
    }
}
```



Use break and continue in moderation: Use break to exit the loop early and continue to skip to the next iteration, but do it carefully to keep your code clear and understandable.

Exercise 15. Enter 10 numbers via keyboard and calculate the average of odd numbers and the largest of the even numbers.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner (System.in);

        int data;
        double sumOdds = 0;
        double numberOfOdds = 0;
        int largestEven = 0;

        for (int i=1; i <=10;i++) {

            System.out.println("Enter number " + i);
            data = exercise.nextInt();

            if (data%2!=0) {
                numberOfOdds ++;
                sumOdds = sumOdds + data;
            } else
```

```
        if (data > largestEven) {
            largestEven = data;
        }
    }

    System.out.println("The largest even number is " + largestEven);
    System.out.println("The average of odd numbers is " +
        (sumOdds/numberOfOdds));
    }
}
```

Exercise 16. Program that asks for numbers until the sum is 1000. Then, show that sum on the screen.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner (System.in);

        int sum = 0;

        while (sum < 1000) {
            System.out.println("Enter a number");
            int data = exercise.nextInt();
            sum = sum + data;
        }
    }
}
```

```
System.out.print("The final amount is " + sum);  
  
    }  
}
```

Exercise 17. We create a program where we enter a number via keyboard. Then, the program calculates all multiples of 3 up to that number, adds them up, and displays the result on the screen.

Solution:

```
import java.util.Scanner;  
public class Main {  
    public static void main(String[] args) {  
  
        Scanner exercise = new Scanner (System.in);  
  
        System.out.println("Enter a number");  
  
        int num = exercise.nextInt();  
        int sum = 0;  
  
        for (int i=1;i<=num;i++){  
  
            if (i%3==0) {  
                System.out.println(i);  
                sum = sum+i;  
            }  
        }  
    }  
}
```

```
        System.out.println("The total sum is " + sum);
    }
}
```

Exercise 18. Create a program that shows all numbers from 1 to 1000 that are not multiples of the number entered via keyboard.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner (System.in);

        System.out.println("Enter the multiple");


        int multiple = exercise.nextInt();

        for (int i=1;i<=1000;i++) {

            if (i%multiple!=0) {
                System.out.println(i + " is not a multiple of " + multiple );
            }
        }
    }
}
```

Exercise 19. Write a program that asks the user for a text string and a specific character, then counts and shows how many times that character appears in the string.

Solution:



```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner (System.in);

        System.out.print("Enter a text string: ");
        String sentence = exercise.nextLine();

        System.out.print("Enter the character to count: ");
        char letter = exercise.next().charAt(0);

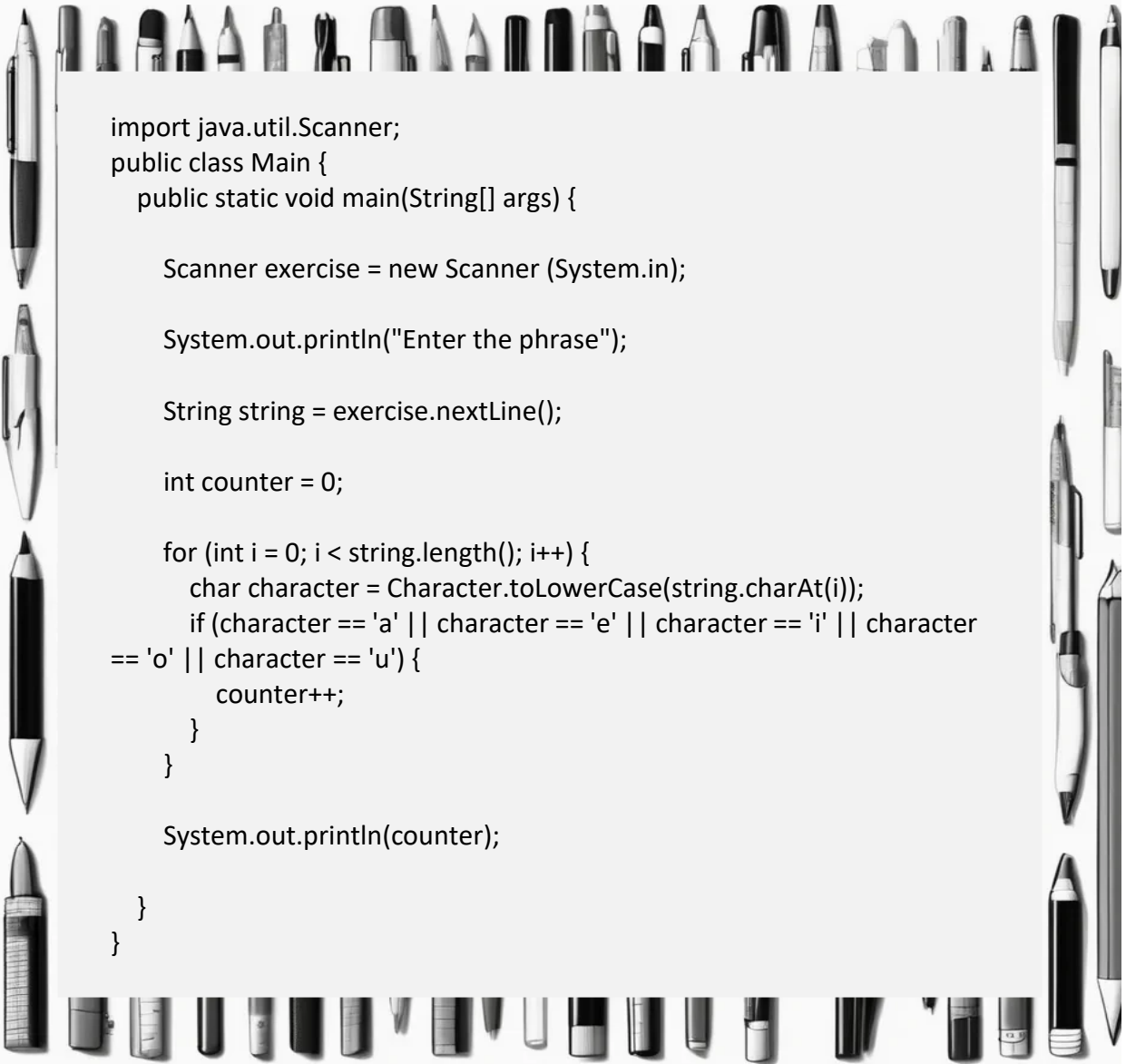
        int counter = 0;

        for (int i = 0; i < sentence.length(); i++) {

            if (sentence.charAt(i) == letter) {
                counter++;
            }
        }
        System.out.print("The number of times repeated is " + counter);
    }
}
```


Exercise 20. Create a program that counts the vowels in a phrase entered via keyboard.

Solution:



```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner (System.in);

        System.out.println("Enter the phrase");

        String string = exercise.nextLine();

        int counter = 0;

        for (int i = 0; i < string.length(); i++) {
            char character = Character.toLowerCase(string.charAt(i));
            if (character == 'a' || character == 'e' || character == 'i' || character
            == 'o' || character == 'u') {
                counter++;
            }
        }

        System.out.println(counter);

    }
}
```



Define a clear exit condition: Establish a condition that allows the loop to end at an appropriate time to avoid infinite loops.

Chapter 6.

Helping Bud. Random Numbers.

I have news from my lawyer and it's not good at all. My computer was used to access a criminal website that provided services to hack companies' security systems. But that's not all, this software is currently being used to extort several multinational companies.

The police believe that I devised the plan along with an accomplice who is still at large. I guess they think it's Chani. I hope they catch him so this whole matter can be cleared up. Meanwhile, defending myself against the accusations is very difficult. My computer was involved and few people believe it was used without my permission.

Rich's statements are the only ones that can help me. He has come to visit me a couple of times and told me that he has spoken several times with the police to explain to them how my laptop ended up in Chani's hands. He even told them that in our house we didn't lock the doors and we left our electronic devices anywhere.



The truth is I'm not very optimistic. I imagine Chani must have fled far away. If they don't arrest him, I'm going to spend quite a few years here. Again, I think about how life changes in an instant. I haven't done absolutely anything and now I have to pay for a crime I have nothing to do with. The worst part is that I can't do anything. I can only wait to see what happens.

My social circle has been drastically reduced; before, I used to fill the house with friends on weekends. Now, I practically only talk to Bud and once a day with the visitors I receive from the outside. My family is very worried and I know they're trying to help me from the outside. I don't feel alone, but sometimes I miss having more company.

At least I have Bud; we spend almost all day talking. Most of the time, about nonsense, nothing serious. It's better to keep a clear head and try to disconnect a bit. Sometimes he asks about my situation and follows the developments, but the truth is that not even I know what's happening.

Bud is lucky; he can spend much of the day with the computer. He can even leave the cell on many occasions to help prison employees with various computer tasks. Perhaps, if I learn, I could also lend a hand. It could be a good opportunity to learn things, entertain myself, and, in the process, see if they can reduce my sentence.



Bud loves to teach and constantly asks me surprise questions. He told me that if I apply myself, he might give me a small job or at least let me help him. We have a deal. He's going to teach me how to work with random numbers in Java and then give me a little test. If I pass it, he'll let me help him.

Working with Random Numbers

Importance of Random Numbers in Java

Random numbers are constantly used in all types of programs. They can serve us, among other things, to simulate random events in games and simulations, generate data for software testing, and optimize algorithms through random decisions.

Furthermore, they are vital in computer security for creating keys and in data analysis for modeling statistical distributions. In summary, random numbers in Java are versatile tools that improve the robustness, security, and performance of computer systems in various applications.

Generating Random Numbers with Math.random

First of all, you should know that there isn't just one way to generate random numbers. I'm going to show you what, in my view, is the simplest method. In Java, random number generation can be done using the Math class and its static random() method. The number we obtain is greater than or equal to 0.0 and less than 1.0. This means it can generate numbers like 0.0, but never reaches 1.0 (that is, 0.999999999... is the maximum possible value).

As you can see, we have some limitations, but nothing we can't solve with a bit of imagination and creativity.

Now, let's see how to use Math.random() in practice to generate random numbers in Java:

- **Generate a random number between 0.0 (inclusive) and 1.0 (exclusive)**

```
double number = Math.random();  
System.out.println("Random number generated: " + number);
```

- **Generate a random number between two figures. Using decimals.**

For this, we use the following structure:

```
double number = Math.random() * (max - min) + min;
```

For example, if we want to create a random number between 10.0 (Included) and 100.0 (Excluded), we'll do it like this:

```
double number = Math.random() * (100 - 10) + 10;  
System.out.println("Random number between 10.0 and 100.0: " + number);
```

- **Generate a random number between two figures without decimals.**

```
int number = (int) (Math.random() * (max - min + 1) + min);
```

There are two changes in this line of code. The first thing we see is the use of (int). Thanks to this, our syntax goes from referring to a decimal variable to an integer one. Notice that the rest of the line is also in parentheses. It starts just before Math.random() and ends at the end. This basically eliminates all decimals. For example, if our random number is 88.87, it will now be 88.

If we continue with the example and want numbers that go from 10 to 100, before we could get figures that ranged from 10.00 to 99.999999... Therefore, if we remove the decimals, what we have is a range between 10 and 99. The problem is that it doesn't include 100. This leads us to the next change, which is the +1. By adding this element, our problem is solved, since the range now extends to 100.999999..., which when converted to a natural number gives us 100. This way, we now have the range of random numbers we needed, 10-100.

Rounding the Number of Decimals Using the Math.round() Method

In the second topic, I already explained two ways to round the number of decimals and thus avoid a long tail of numbers. As I promised you at that time, here I bring you another way to achieve it. In my opinion, the simplest one, although I considered it important to add the other two in this book. As you've already seen in the title of this section, we're going to use the Math.round() method, the same one we've been using throughout this topic to generate random numbers. I'll explain it to you directly with examples.

Rounding to two decimals:

```
public class Main {  
    public static void main(String[] args) {  
        double number = 3.14159;  
        double result = Math.round(number * 100.0) / 100.0;  
        System.out.println("Number with two decimals: " + result); // Output: 3.14  
    }  
}
```

Rounding to three decimals:

```
public class Main {  
    public static void main(String[] args) {  
        double num = Math.random() * 10;  
        double result = Math.round(num * 1000.0) / 1000.0;  
        System.out.println("Random number rounded to three decimals: " + result);  
    }  
}
```

Summary of what you've learned

Random numbers are essential in programming in general. They are used, among other things, to simulate random events in games and simulations, generate data for software testing, or optimize algorithms. Additionally, they are crucial in computer security, serving to create cryptographic keys and in data analysis to model statistical distributions. These uses improve the security and performance of computer systems.

In Java, there are several ways to generate random numbers, but we are going to focus on one in particular. We will create them using the `Math` class and its `random()` method. This method produces numbers between 0.0 (inclusive) and 1.0 (exclusive). With this numerical range we are somewhat limited, but this is not a problem since with ingenuity we can access other values.

To generate a random number between two decimal numbers, the formula `Math.random() * (max - min) + min` is used.

To obtain a random integer in a specific range, the result is converted to an integer and the range is adjusted by adding 1, thus guaranteeing the inclusion of the upper value in the range.

Practical exercises

Exercise 1. Create a program that calculates the roll of twenty dice and the sum of the results.

Solution:

```
public class Main {  
    public static void main(String[] args) {  
  
        int sum = 0;  
  
        //First we'll create a loop for the 20 dice.  
        //This way we won't have to copy and paste the same line 20 times.  
  
        for (int i=0; i<20;i++) {  
  
            //We use Math.random to generate random numbers.  
            //In this case we want the maximum to be six. That's why the multiplication.  
            //When multiplying by 6 we get numbers between 0 and 5.99. So we add 1.  
            //Now we have numbers between 1 and 6.99 but we must transform them  
            //into integers.  
            //This is done by converting the whole expression to integer with (int). As  
            //seen below.  
  
            int a =(int)(Math.random()*6 + 1);  
            sum = sum+a;  
        }  
  
        System.out.println("The sum of the 20 dice is " + sum);  
    }  
}
```


Exercise 2. Java program that randomly shows 100 cards from a Spanish deck.

Solution:

```
public class Main {  
    public static void main(String[] args) {  
  
        String suit = "";  
        String figures = "";  
  
        //We create a for loop to show 100 cards  
  
        for (int i = 0; i <= 100; i++) {  
  
            //We create a random number from 1 to 4 to generate the suits.  
  
            int a = (int) (Math.random() * 4 + 1);  
  
            //Depending on which value comes up, we assign a suit  
  
            if (a == 1) {  
                suit = " coins ";  
            }  
            if (a == 2) {  
                suit = " swords ";  
            }  
            if (a == 3) {  
                suit = " cups ";  
            }  
            if (a == 4) {  
                suit = " clubs ";  
            }  
        }  
    }  
}
```

```
//Now we create random numbers between 1 and 10.  
//From 1 to 7 will correspond to normal cards  
// 8 to jack, 9 to knight, and 10 to king.
```

```
int b = (int) (Math.random() * 10 + 1);
```

```
//We assign value to the figures variable as follows.
```

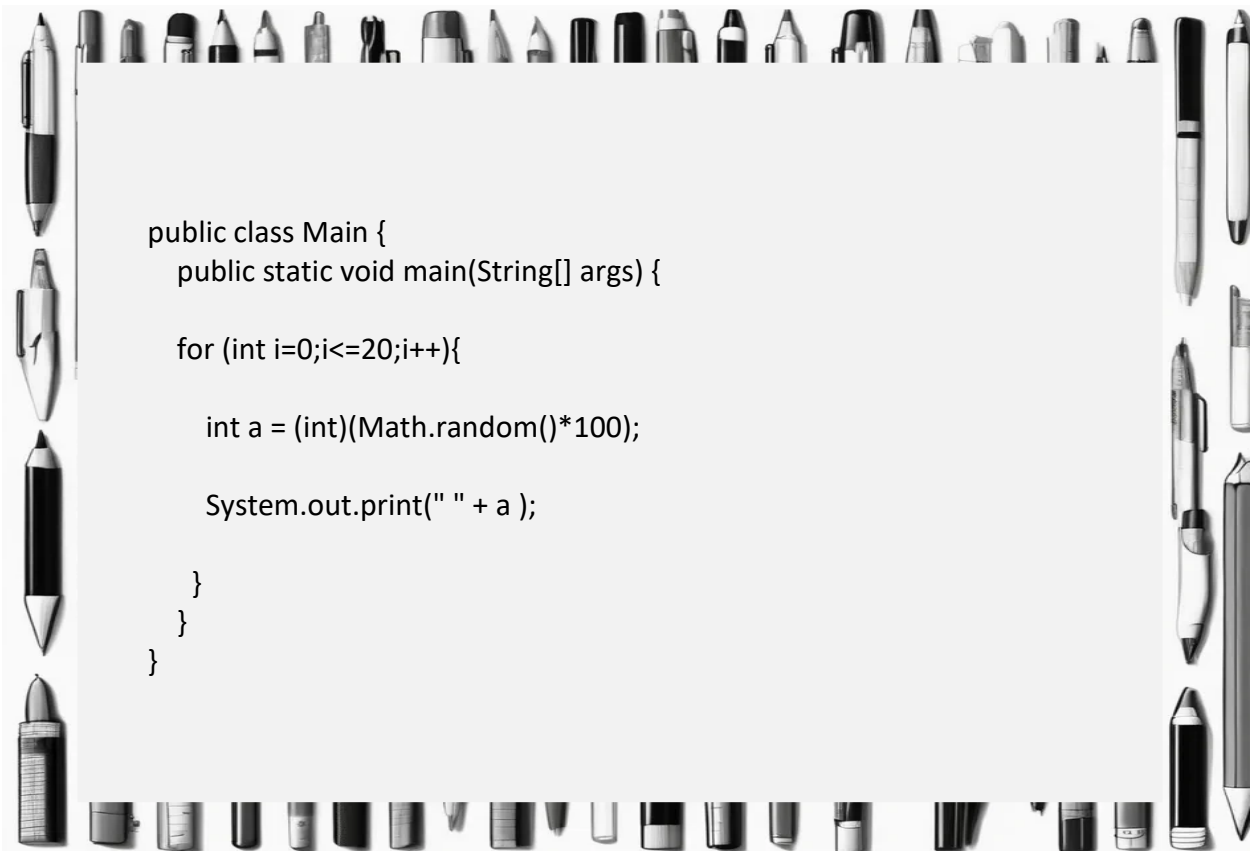
```
if (b==8) {  
    figures = "Jack";  
}  
if (b==9) {  
    figures = "Knight";  
}  
if (b==10) {  
    figures = "King";  
}
```

```
//We show the results on the screen using conditionals.
```


```
if (b==1) {  
    System.out.println("Ace of" + suit);  
}  
if (b>=2&&b<=7) {  
    System.out.println(b + " of" + suit);  
}  
if (b>=8&&b<=10) {  
    System.out.println(figures + " of" + suit);  
}  
}  
}
```

Exercise 3. Java program that shows 20 random numbers between 0 and 100 on the same line.

Solution:

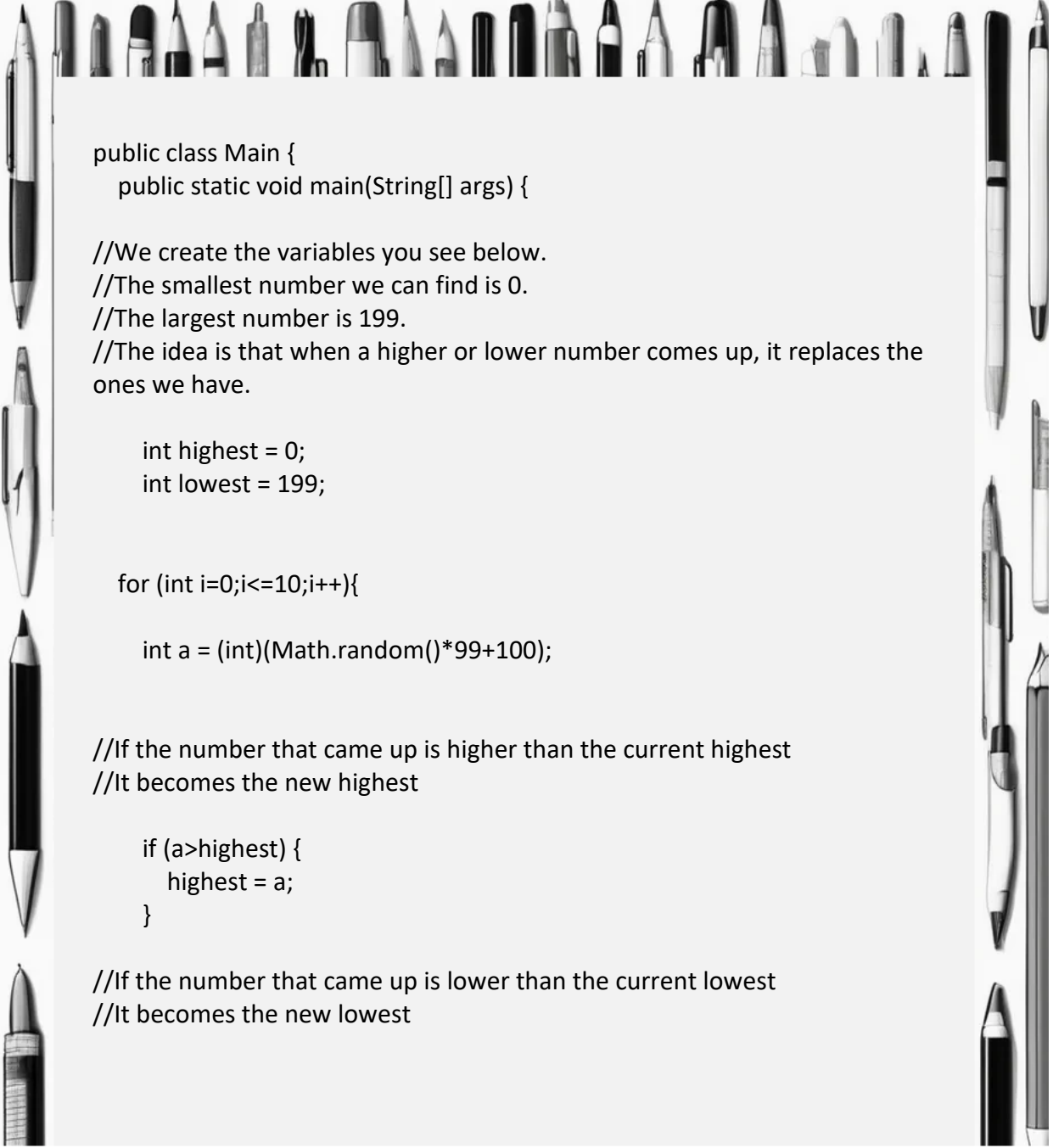


```
public class Main {  
    public static void main(String[] args) {  
  
        for (int i=0;i<=20;i++){  
  
            int a = (int)(Math.random()*100);  
  
            System.out.print(" " + a );  
  
        }  
    }  
}
```

 **Understanding the Range:** To generate a random number between two specific values, use the formula $\text{Math.random()} * (\text{max} - \text{min}) + \text{min}$. Make sure you understand that $\text{Math.random}()$ generates a number between 0.0 and 1.0, so multiplying by $(\text{max} - \text{min})$ and then adding min correctly adjusts the range.

Exercise 4. Java program that shows 10 random numbers between 100 and 199 and then displays which was the highest and the lowest.

Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        //We create the variables you see below.  
        //The smallest number we can find is 0.  
        //The largest number is 199.  
        //The idea is that when a higher or lower number comes up, it replaces the  
        ones we have.  
  
        int highest = 0;  
        int lowest = 199;  
  
        for (int i=0;i<=10;i++){  
  
            int a = (int)(Math.random()*99+100);  
  
            //If the number that came up is higher than the current highest  
            //It becomes the new highest  
  
            if (a>highest) {  
                highest = a;  
            }  
  
            //If the number that came up is lower than the current lowest  
            //It becomes the new lowest
```

```
        if (a<lowest) {  
            lowest = a;  
        }  
    }  
  
    System.out.println("The highest number is: " + highest);  
    System.out.println("The lowest number is: " + lowest);  
}  
}
```

Exercise 5. Create a program that generates random numbers until 24 comes up. Then, it will count all the numbers that have come up.

Solution:

```
public class Main {  
    public static void main(String[] args) {  
  
        int a = 0;  
        int count = 0;  
  
        while (a!=24) {  
            a=(int)(Math.random()*100);  
            count++;  
        }  
  
        System.out.println("The total count of numbers is: " + count);  
    }  
}
```

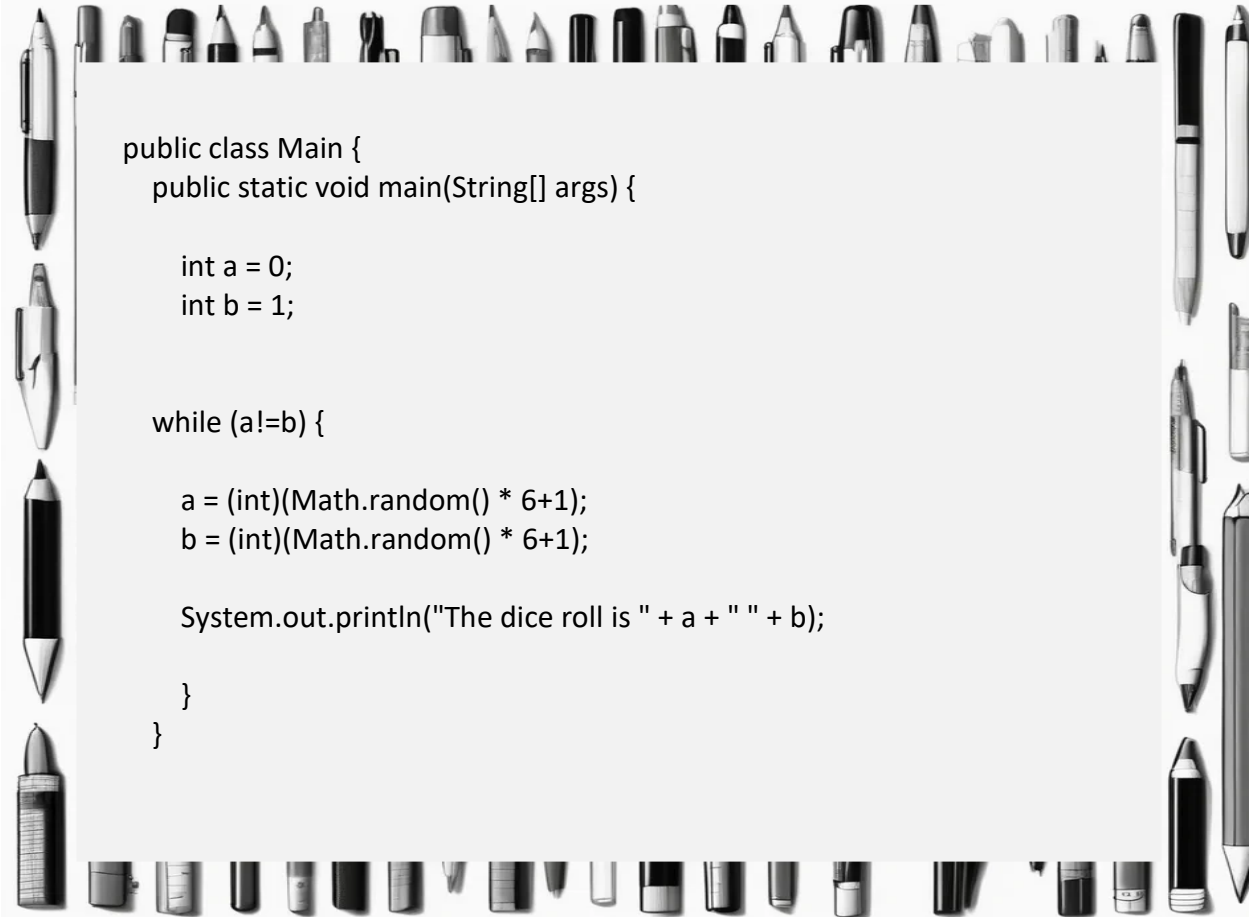
Exercise 6. Create a program that randomly generates 20 grades and classifies them into sufficient, pass, good, and excellent.

Solution:


```
public class Main {  
    public static void main(String[] args) {  
  
        String grade = "";  
  
        for (int i=0;i<=20;i++) {  
  
            double a =(Math.random() * 10);  
            String result = String.format("%.2f", a);  
  
            if (a<5) {  
                grade = "Fail";  
            }  
            if (a>=5&&a<7) {  
                grade = "Pass";  
            }  
            if (a>=7&&a<9) {  
                grade = "Good";  
            }  
            if (a>=9) {  
                grade = "Excellent";  
            }  
  
            System.out.println(result + " ----> " + grade);  
  
        }  
    }  
}
```

Exercise 7. Create a program that shows the roll of two dice until both show the same result.

Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        int a = 0;  
        int b = 1;  
  
        while (a!=b) {  
  
            a = (int)(Math.random() * 6+1);  
            b = (int)(Math.random() * 6+1);  
  
            System.out.println("The dice roll is " + a + " " + b);  
  
        }  
    }  
}
```

 **Using Integers:** To get integer numbers in a specific range, convert to integer with (int). Additionally, adjust the range by adding 1 to the formula: (int)(Math.random() * (max - min + 1) + min). This includes both the minimum and maximum values in the range of possible results.

Exercise 8. Java program in which we have five attempts to guess a randomly created number.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner exercise = new Scanner(System.in);
        int attempts = 5;

        //We create a four-digit password. Between 1000 and 9999.

        int password = (int)(Math.random()*9000+1000);

        System.out.println("The password is " + password);

        while (attempts >0) {
            attempts--;
            System.out.println("Enter a password ");
            int guess = exercise.nextInt();

            if (guess==password) {
                System.out.println("You've guessed correctly");
                break;
            }


            if (attempts ==0) {
                System.out.println("No more attempts");
            }
        }
    }
}
```


Exercise 9. We create a program that simulates a soccer pool of 9 matches. Three columns will be shown where 1, X or 2 will appear marked.

Solution:

```
public class Main {
    public static void main(String[] args) {

        System.out.println("The result of the soccer pool is the following:");
        System.out.println(" ");
        for (int i =1;i<=9;i++) {
            int result = (int)(Math.random()*3+1);
            if (result==1) {
                System.out.println("Match " + i + " = " + " 1 |   |");
            }
            if (result==2) {
                System.out.println("Match " + i + " = " + "   | X |");
            }
            if (result==3) {
                System.out.println("Match " + i + " = " + "   | X | 2");
            }
        }
    }
}
```

 **Avoid Undesired Values:** If you need to exclude certain values from the range, use conditionals to regenerate the random number in case an undesired value is obtained. For example, if you generate a random number and need to exclude 100, you can use a while loop to repeat the generation until a valid value is obtained.

Exercise 10. Java exercise on random numbers, but in this case using a char variable. Create a program that randomly displays the following symbols five times: &^%\$.*

Solution:

```
public class Main {  
    public static void main(String[] args) {  
  
        char a = ' ';  
        for (int i=1;i<=5;i++) {  
  
            int b = (int)(Math.random()*5+1);  
  
            if (b==1) {  
                a = '&';  
            }  
            if (b==2) {  
                a = '*';  
            }  
            if (b==3) {  
                a = '^';  
            }  
            if (b==4) {  
                a = '%';  
            }  
  
            if (b==5) {  
                a = '$';  
            }  
  
            System.out.println(a);  
        }  
    }  
}
```

Chapter 7.

The visit of Rich's cousin. Arrays

Today I received an unexpected visit. I'm still a bit dazed. Around 12 noon, a guard came to get me and took me to the visiting room. I thought it would be my mother or perhaps the lawyer, but it was Chani's cousin. I had never seen him before, so I didn't know it was him until he introduced himself.



I remained silent for several seconds. I could feel myself getting angry little by little, until I started cursing and stumbling over my words, as I wanted to express many things at the same time. At one point, one of the guards came over and asked me to calm down. The truth is, it was a good release. After that, I felt much calmer and willing to hear what he had to say.

Besides being angry, I was very curious. All the answers I had been yearning for during weeks were right in front of me. I wanted to know exactly what kind of trouble Chani was involved in, why he had done it, and above all, why he had implicated me. Perhaps it had all been a mistake. Maybe he was remorseful and wanted to exonerate me. Countless questions were running through my head, and I didn't know which one to ask first.

I was just about to say something when he interrupted me. He told me he knew what I was thinking, that he knew I believed Chani was guilty, but that wasn't the case. Although all the evidence pointed to him, he was actually innocent.

It seemed pretty hard to believe, so I asked him how he could be so sure. Then he told me that Chani had sent him to tell me the true story. I became furious again. I told him that if he knew where Chani was, he should tell or he would be guilty of harboring a fugitive. Once again, the guard came to call me out.

Once I calmed down, the cousin proceeded to tell me the story. According to him, Chani had nothing to do with it. He had no choice but to flee when he discovered the real culprits' plan: Rich and Fernando.



It appears that the day before the party, Chani arrived home and went straight to his room. Fernando and Rich were in the living room and didn't notice his presence. Without fear of being discovered, they began to discuss their plan. At that moment, they were obtaining the illegal software they would later use to extort companies.

Chani received a call and was discovered. The three of them sat down in the living room, and it was then that he realized they were using my computer. Finding themselves cornered, they had no choice but to offer him a part in the crime. They explained the plan in detail, including the part where, if something went wrong, they would blame me.

Despite their insistence, Chani refused. That's when they threatened him. They came up with a way to incriminate him if he didn't cooperate. They advised him to think about it for a few hours and said they wanted a definitive answer the next day.

During the night, they got that answer. But it wasn't what they expected. So, things became even more tense, and they physically threatened him. They forced him to leave as soon as possible and also coerced him into not saying anything.

The story made sense, but I didn't believe it. Rich is my friend. He would never do something like that to me. After a few seconds of silence, he looked at me intently and asked if I believed him. I told him I didn't, and then he pulled out his phone. He showed me an audio recording from the night of the party in which Rich could be heard telling him to leave that same night and never come back.

Chani tried to find evidence against them, but there was a lot of noise at the party, and Rich is very astute. From that conversation, only the part where he tells him to leave the city that very night can be distinguished.



It seems Chani has a plan to unmask Rich, but he might need my help. I don't know how I can help while locked up in here. Maybe during one of his visits, I could get a confession or at least some information. The worst part about being in isolation isn't just the lack of information, but also that I'm not very useful.

After the brief conversation, I returned to my cell. I'm still processing the information. Rich has come to visit me several times and has always seemed very concerned. I didn't expect this, but it makes sense. Now I just hope he comes to see me again, so I can clear things up face to face.

To try to think about something else, I've asked Bud to continue with the Java lessons. He seems especially motivated. He told me that once I learn about arrays, he'll be able to explain exactly how he helps the prison staff with the jail's software. For now, I don't want to tell him anything about my personal situation. I know it would worry him. So, for the time being, I put on a brave face and try to learn something.

Arrays

In Java, arrays are a basic way to store **multiple values** of the same type in a **single variable**. Unlike normal variables that can only contain one value, an array can store multiple values, called elements, which are organized in a specific order and accessed using numbers called indices.

An array is created by specifying what type of data it will hold and how many elements it will store. For example, an int array of integers with a size of 5 can store five whole numbers. Arrays in Java start counting from 0, so the first element is at position 0.

Importance of Arrays in Programming

- They allow data to be organized in a structured manner, facilitating access and manipulation.
- With arrays, we can efficiently store large amounts of related data and access them through indices, instead of managing many independent variables.
- Array elements can be quickly accessed using their index, which is crucial for applications that need fast and frequent data reading or modification operations.
- Furthermore, arrays allow common operations such as iteration, searching, sorting, and data manipulation to be performed easily. Many algorithms and data structures are based on arrays due to their simplicity and efficiency, such as sorting and searching algorithms.
- In real-world applications, arrays are widely used, from game development to data processing and scientific applications, where they are essential for managing large datasets and performing complex numerical calculations.

Array Declaration

The declaration of an array in Java is done by specifying the data type of the elements it will contain, followed by square brackets [] and the array name.

Step 1: Declaring the array

```
type[] arrayName;
```

Example:

```
int[] numbers; // Declaration of an integer array
```

```
String[] names; // Declaration of a string array
```

```
double[] temperatures; // Declaration of a double array
```

Step 2: Creating the array

```
arrayName = new type[size];
```

Example:

```
numbers = new int[5]; // Creates an integer array with 5 elements
```

```
names = new String[3]; // Creates a string array with 3 elements
```

```
temperatures = new double[7]; // Creates a double array with 7 elements
```

However, you'll rarely use this structure. It's much simpler, faster, and more effective if we do it all in one line. Besides, it will be much easier for you to remember.

Declaration and Creation in a Single Line

```
type[] arrayName = new type[size];
```

Examples:

```
int[] numbers = new int[5]; // Declaration and creation of an integer array with 5 elements
```

```
String[] names = new String[3]; // Declaration and creation of a string array with 3 elements
```

```
double[] temperatures = new double[7]; // Declaration and creation of a double array with 7 elements
```

Assigning Values to Arrays

```
int[] numbers = new int[5];
```

```
    numbers[0] = 10;
```

```
    numbers[1] = 20;
```

```
    numbers[2] = 30;
```

```
    numbers[3] = 40;
```

```
    numbers[4] = 50;
```

Now each element in our array has a value. For example, if we want to display the fifth position on the screen, we'll do it in the following way:

```
System.out.println(numbers[4]);
```



```
String[] names = new String[3];
```

```
names[0] = "Lucía";
```

```
names[1] = "Pedro";
```

```
names[2] = "Luís";
```

No further theoretical explanation is necessary. This topic is quite simple to understand just by looking at the exercises. At least for now. I've only shown examples of natural numbers (int type variables) and text (String type variables). But it's clear that, for example, we can use decimal numbers if we use a double or characters with a char. For the latter, instead of using double quotes " ", we would use single quotes ' '.

Regardless of the variable type, the same structure is always followed.

Assigning Values to Arrays in a Single Line

Sometimes you'll encounter brackets in arrays. They are also used to assign values. This is the structure used for this purpose:

```
- type[] arrayName = {value1, value2, value3, ...};
```

Of course, let's see it with an example. We'll start with an int array with 5 elements. We'll assign values and display the fifth element. Remember that arrays always start at position 0. So our fifth element is at position 4.

```
int[] numbers = {10, 20, 30, 40, 50};
```

```
System.out.println(numbers[4]);
```

Now let's continue with our text example. We have three names and we'll declare and assign values to an array in a single line.

```
- String[] names = {"Lucía", "Pedro", "Luis"};
```

As you can see, at first glance, this way of creating arrays and assigning values is much simpler. The code looks more organized and elegant. However, it's not all advantages. Not having the position of each element indicated can sometimes make us get lost or a bit confused. As I mentioned before, the array always starts at position zero and goes up to the maximum number of elements we have. Having the position next to each value is very convenient, and with a quick glance, we can visualize the entire structure of our array in our head.

Traversing Arrays Using Loops in Java

Traversing an array means accessing each of its elements one by one. Loops are essential tools for this task, as they allow us to iterate over array elements in a simple and efficient way. The simplest way to do this task is to use a for loop.

Example: Traversing an Integer Array using a For Loop

```
public class Main {  
    public static void main(String[] args) {  
        int[] numbers = {1, 2, 3, 4, 5};  
        // We know the array size is 5  
        for (int i = 0; i < 5; i++) {  
            System.out.println(numbers[i]);  
        }  
    }  
}
```

Example: Traversing a String Array

```
public class Main {  
    public static void main(String[] args) {  
        String[] colors = {"Red", "Green", "Blue", "Yellow"};  
        // We know the array size is 4  
        for (int i = 0; i < 4; i++) {  
            System.out.println(colors[i]);  
        }  
    }  
}
```

Example: Traversing a Character Array (char)

```
public class Main {  
    public static void main(String[] args) {  
        char[] letters = {'A', 'B', 'C'};  
        // We know the array size is 3  
        for (int i = 0; i < 3; i++) {  
            System.out.println(letters[i]);  
        }  
    }  
}
```

Notice that in all examples, the loop's first position is 0 since arrays always start from 0.

Summary of What We've Learned

In Java, an array is a structure that allows storing multiple values of the same type in a single variable. Unlike a normal variable that can only contain one value, an array can store multiple values, called elements. These elements are organized in a specific order and are accessed through numerical indices, starting from 0.

Array Declaration and Creation

- Declaration: Involves specifying the data type that the array will contain and giving it a name.
- Creation: Involves allocating memory for the array, defining how many elements it can store.
- Declaration and Creation in a Single Line: To simplify and make the code more efficient, an array can be declared and created in a single line.

Assigning Values to an Array

Values are assigned to array elements using their index. This allows accessing and modifying each array element directly through its index.

Assigning Values in a Single Line

It's possible to assign values to an array directly when declaring and creating it, resulting in cleaner and more readable code.

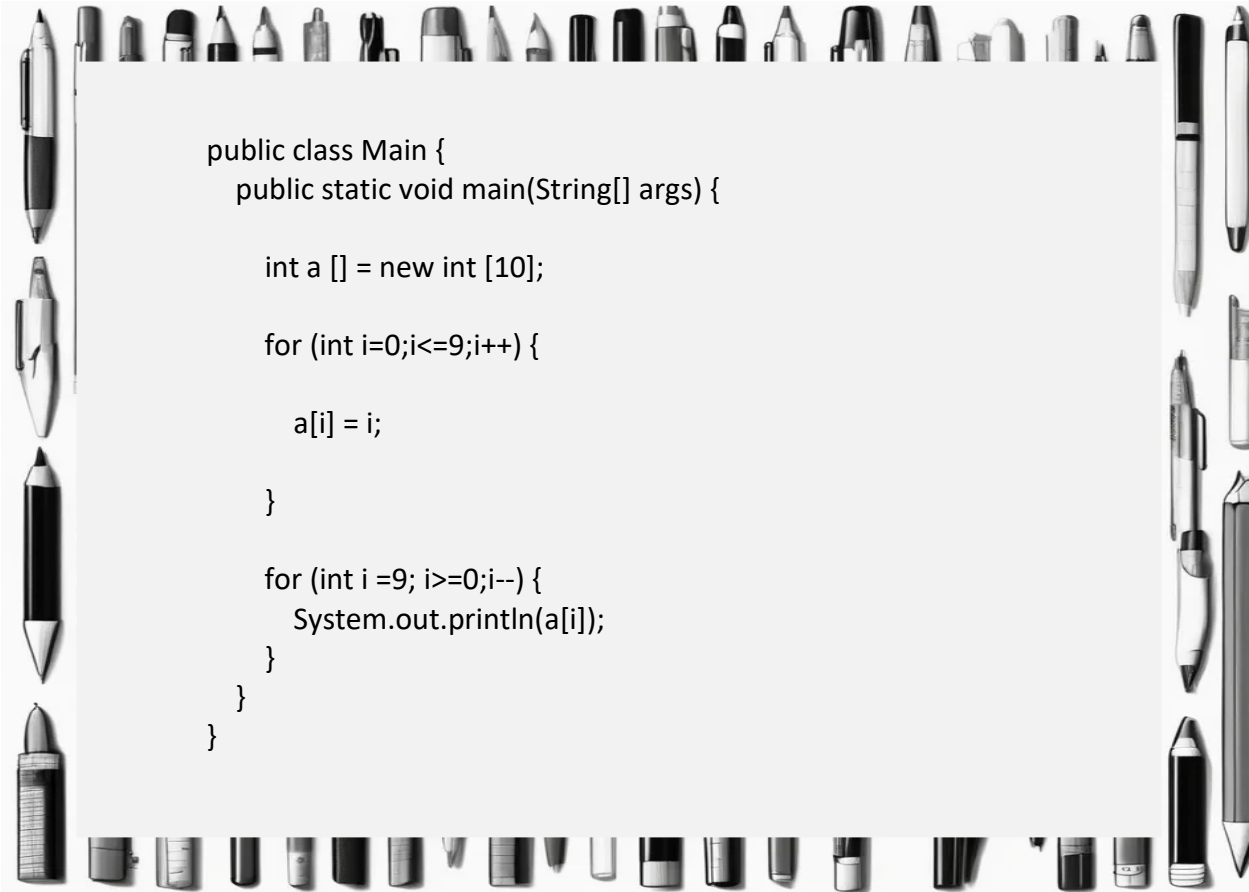
Traversing Arrays Using Loops

To access each element of an array, loops such as the for loop are used. This type of loop is efficient for iterating over each array element and performing various operations with them.


Practical exercises.

Exercise 1. Create an array containing the numbers from 0 to 9. Then, display them on the screen in reverse order.

Solution:

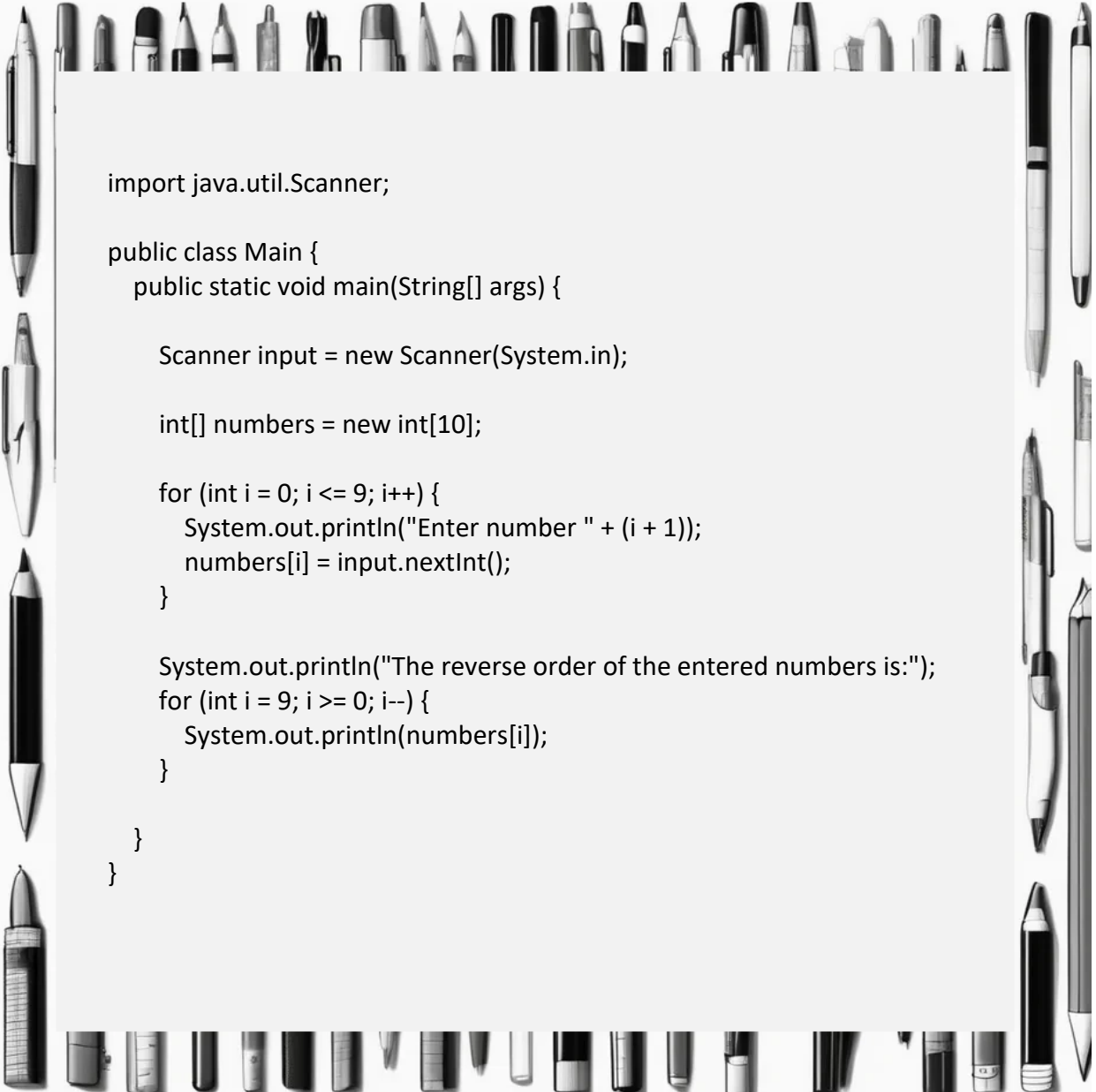


```
public class Main {  
    public static void main(String[] args) {  
  
        int a [] = new int [10];  
  
        for (int i=0;i<=9;i++) {  
  
            a[i] = i;  
  
        }  
  
        for (int i =9; i>=0;i--) {  
            System.out.println(a[i]);  
        }  
    }  
}
```

 **Initialize Arrays Properly:** Before using an array, make sure to initialize it correctly with the appropriate size and necessary values. This prevents index out of range errors and ensures that the array has the data you need.

Exercise 2. Enter 10 numbers from the keyboard and then display them in reverse order, using an array.

Solution:



```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

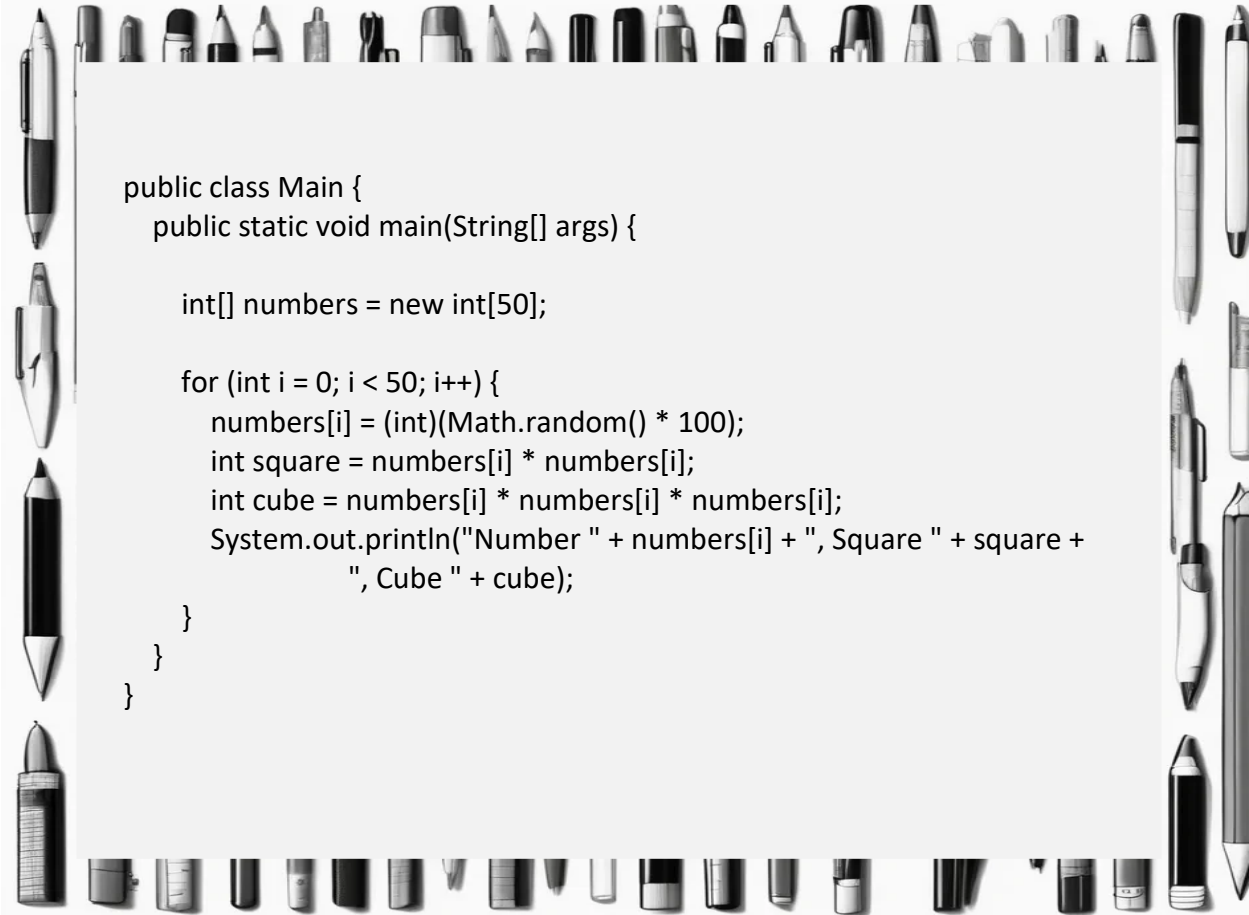
        int[] numbers = new int[10];

        for (int i = 0; i <= 9; i++) {
            System.out.println("Enter number " + (i + 1));
            numbers[i] = input.nextInt();
        }

        System.out.println("The reverse order of the entered numbers is:");
        for (int i = 9; i >= 0; i--) {
            System.out.println(numbers[i]);
        }
    }
}
```

Exercise 3. Generate 50 random numbers between 0 and 100, and then display their squares and cubes, using an array.

Solution:



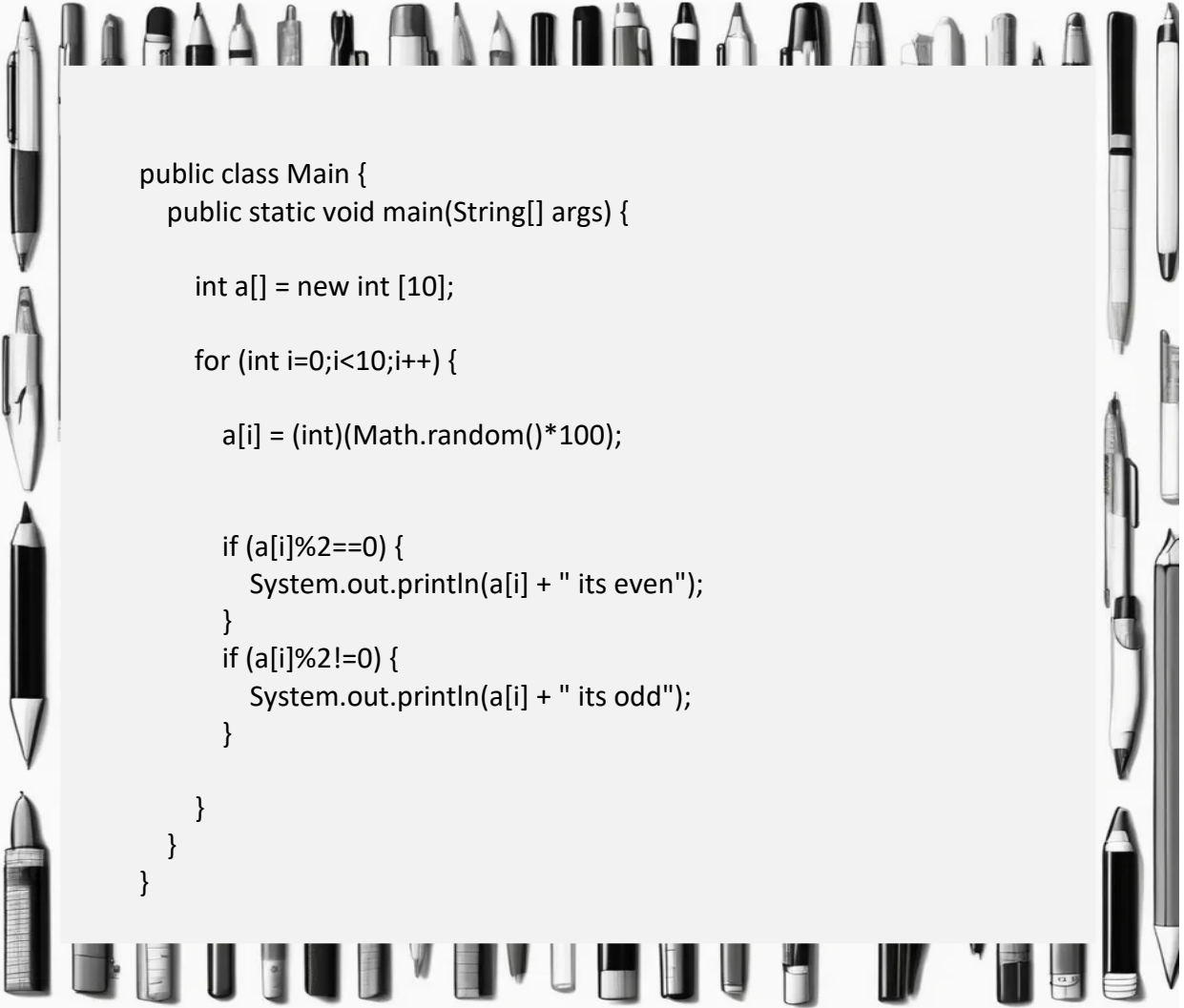
```
public class Main {  
    public static void main(String[] args) {  
  
        int[] numbers = new int[50];  
  
        for (int i = 0; i < 50; i++) {  
            numbers[i] = (int)(Math.random() * 100);  
            int square = numbers[i] * numbers[i];  
            int cube = numbers[i] * numbers[i] * numbers[i];  
            System.out.println("Number " + numbers[i] + ", Square " + square +  
                               ", Cube " + cube);  
        }  
    }  
}
```




Always verify that the indices used are within the array's bounds.
Accessing indices outside the allowed range can cause runtime errors.

Exercise 4. Create a program that displays 10 random numbers from 0 to 100. Next to each number, show whether it is even or odd.

Solution:




```
public class Main {  
    public static void main(String[] args) {  
  
        int a[] = new int [10];  
  
        for (int i=0;i<10;i++) {  
  
            a[i] = (int)(Math.random()*100);  
  
            if (a[i]%2==0) {  
                System.out.println(a[i] + " its even");  
            }  
            if (a[i]%2!=0) {  
                System.out.println(a[i] + " its odd");  
            }  
        }  
    }  
}
```

 **Use Loops to Traverse Arrays:** Use for loops to efficiently traverse array elements. Loops facilitate the manipulation and access of elements, allowing operations such as additions, searches, and updates to be performed quickly.

Exercise 5. Create a program that displays 10 numbers entered from the keyboard. Next to each number, show whether it is even or odd.

Solution:



```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

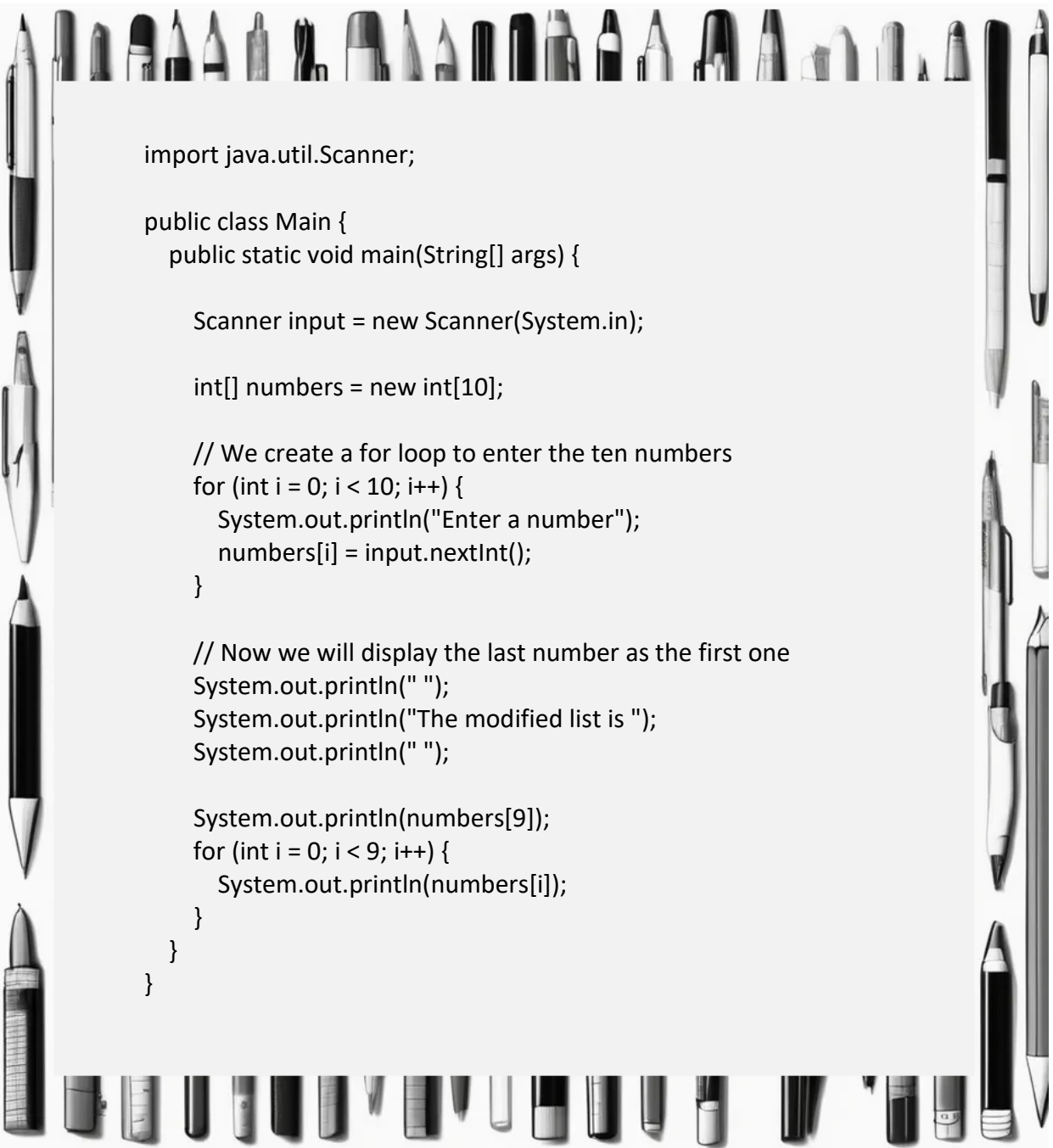
        int[] numbers = new int[10];

        for (int i = 0; i <= 9; i++) {
            System.out.println("Enter a number");
            numbers[i] = input.nextInt();
        }

        for (int i = 0; i <= 9; i++) {
            if (numbers[i] % 2 == 0) {
                System.out.println(numbers[i] + " is even");
            }
            if (numbers[i] % 2 != 0) {
                System.out.println(numbers[i] + " is odd");
            }
        }
    }
}
```

Exercise 6. Exercise where we enter 10 numbers from the keyboard and move the last number to the first position.

Solution:



```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        int[] numbers = new int[10];

        // We create a for loop to enter the ten numbers
        for (int i = 0; i < 10; i++) {
            System.out.println("Enter a number");
            numbers[i] = input.nextInt();
        }

        // Now we will display the last number as the first one
        System.out.println(" ");
        System.out.println("The modified list is ");
        System.out.println(" ");

        System.out.println(numbers[9]);
        for (int i = 0; i < 9; i++) {
            System.out.println(numbers[i]);
        }
    }
}
```

Exercise 7. Program that creates 50 random numbers between 0 and 100. We create a searcher that looks for a specific number and replaces it with another number that we will also specify.

Solution:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        System.out.println("Enter the number you want to replace");
        int original = input.nextInt();

        System.out.println("Enter the number you want to replace it with");
        int replacement = input.nextInt();

        int[] numbers = new int[50];


        // We create a for loop to generate 50 random numbers
        for (int i = 0; i < 50; i++) {
            numbers[i] = (int)(Math.random() * 100);
            if (numbers[i] == original) {
                numbers[i] = replacement;
            }
        }

        System.out.println("Here's the modified list of numbers");

        for (int i = 0; i < 50; i++) {
            System.out.println(numbers[i]);
        }
    }
}
```

Exercise 8. Create a Java Program that classifies 10 numbers greater than 0, entered via keyboard, into even or odd, using an array.

Solution:



```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {


        Scanner input = new Scanner(System.in);

        // Create an array to store the 10 numbers
        int[] numbers = new int[10];
        // Create an array to store even numbers
        int[] evenNumbers = new int[10];
        // Create an array to store odd numbers
        int[] oddNumbers = new int[10];

        // Counters for even and odd numbers
        int evenCount = 0;
        int oddCount = 0;

        // Create a loop to enter 10 numbers
        // They will be stored in our numbers array
        // Depending on whether they are even or odd,
        // they will also be stored in evenNumbers/oddNumbers

        for (int i = 0; i < 10; i++) {
            System.out.println("Enter a number");
            numbers[i] = input.nextInt();
        }
    }
}
```



```
        if (numbers[i] % 2 == 0) {
            evenNumbers[evenCount++] = numbers[i];
        } else {
            oddNumbers[oddCount++] = numbers[i];
        }
    }

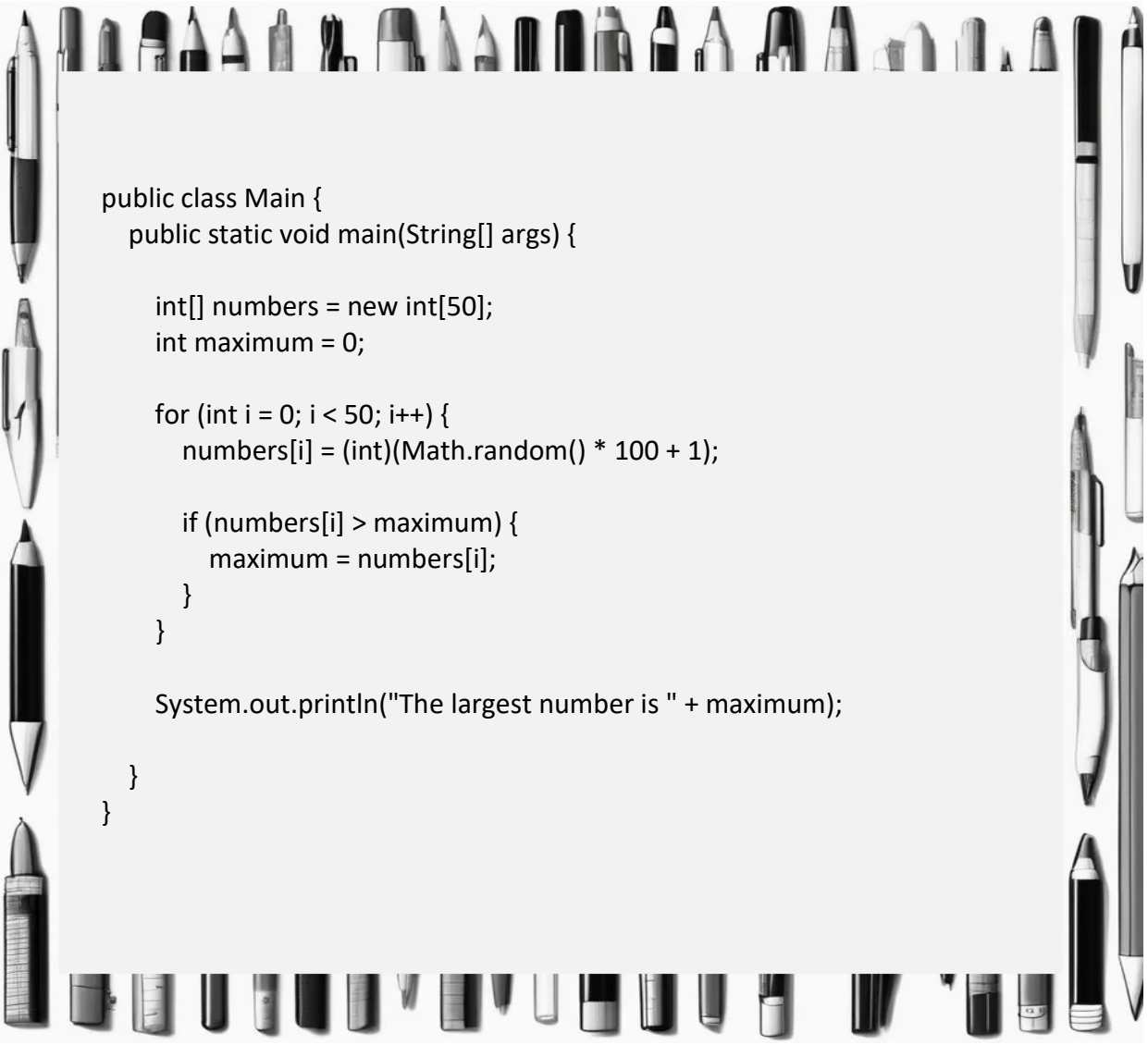
    System.out.println(" ");
    System.out.println("Even numbers");

    // Display even numbers
    // In positions where there's no data for the array,
    // the default value is 0.
    // That's why we use an if statement to only show numbers greater than 0

    for (int i = 0; i < evenCount; i++) {
        if (evenNumbers[i] > 0) {
            System.out.print(evenNumbers[i] + " ");
        }
    }
    System.out.println(" ");
    System.out.println(" ");
    System.out.println("Odd numbers");
    for (int i = 0; i < oddCount; i++) {
        if (oddNumbers[i] > 0) {
            System.out.print(oddNumbers[i] + " ");
        }
    }
}
```

Exercise 9. Java program that displays 50 numbers from 1 to 100. Then, shows which is the maximum value, using an array.

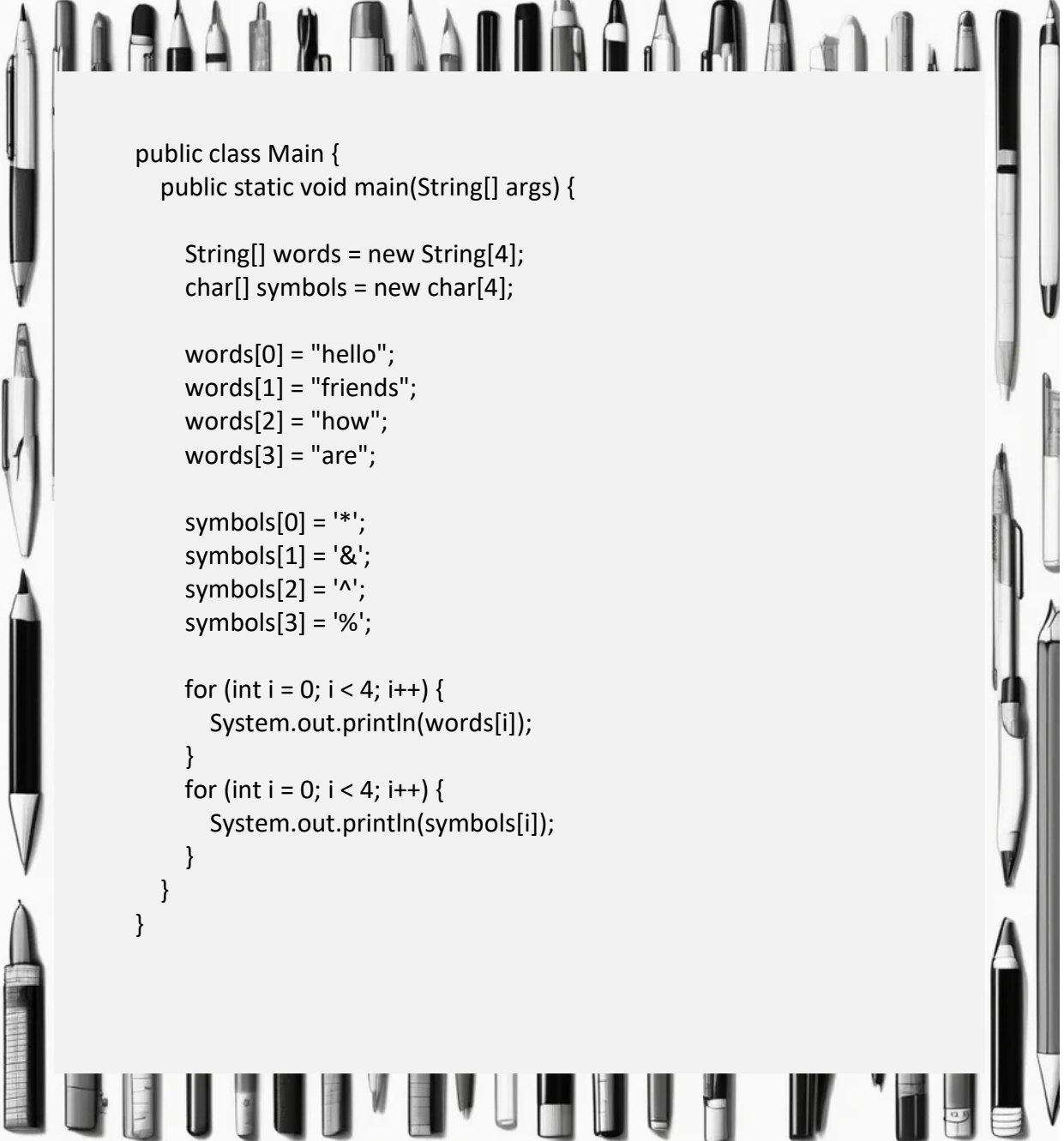
Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        int[] numbers = new int[50];  
        int maximum = 0;  
  
        for (int i = 0; i < 50; i++) {  
            numbers[i] = (int)(Math.random() * 100 + 1);  
  
            if (numbers[i] > maximum) {  
                maximum = numbers[i];  
            }  
        }  
  
        System.out.println("The largest number is " + maximum);  
    }  
}
```

Exercise 10. Array exercises with char and String type variables. Create an array containing these words: "Hello", "friends", "how", "are". Then, create another array containing the following symbols: ^, &, *, %. Finally, display everything on screen.

Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        String[] words = new String[4];  
        char[] symbols = new char[4];  
  
        words[0] = "hello";  
        words[1] = "friends";  
        words[2] = "how";  
        words[3] = "are";  
  
        symbols[0] = '*';  
        symbols[1] = '&';  
        symbols[2] = '^';  
        symbols[3] = '%';  
  
        for (int i = 0; i < 4; i++) {  
            System.out.println(words[i]);  
        }  
        for (int i = 0; i < 4; i++) {  
            System.out.println(symbols[i]);  
        }  
    }  
}
```

Chapter 8.

El trabajo de bud. Matrices 2D

The days feel endless. I'm anxiously waiting for Rich's visit, but it's taking longer than usual. It's been ten days since I had the conversation with Chani's cousin, and he still hasn't come to see me. Bud keeps me entertained; he recently introduced me to his friend Pedro. He also helps the prison staff with security software. He's a very introverted person. He hardly talks to anyone and usually spends most of the day on the computer. I think, given his personality, it's lucky for him that they let him spend so much time glued to the computer.

On the other hand, Bud is more sociable. He enjoys talking with anyone, but even so, having an activity that keeps you busy is the best thing that can happen to you while you're locked up in here.

If I want to be his new apprentice, I need to know exactly what they do. Right now, I have no idea. Bud thinks I'm ready to perform small tasks and that, little by little, he'll give me more responsibilities.



Without warning, I'm informed that I have a visitor. I hope it's Rich, but it's not. Again, it's Chani's cousin. This time he tells me they have a plan to prove my innocence. They want to set a trap for Rich. The problem is that, to carry it out, I need to be outside the prison. The simplest way would be to fake some kind of ailment, like a dental problem, so they would transfer me to an external doctor.

The problem would be evading the police officers who would supposedly escort me there. If I managed to do that, I suppose the next step of the plan would be to meet with Rich and try to get a confession out of him.



I'm not at all sure about this. If I leave and escape from the officers escorting me, I'll get into really big trouble. They would convict me of another crime, and besides, they wouldn't let me participate in any activities inside the prison. I would have to forget about helping Bud and Pedro with the security system. But, on the other hand, if I do nothing, I'll most likely have to spend a long time here.

I think it's crazy, but if I have to get out of prison, I need to ask advice from someone who has lived here for several years. So I think the best option is to talk to Bud about it. Maybe he has ideas on how to get out for a few hours without getting into trouble.

I thought he would tell me I was crazy and to forget about the idea. But far from it, he said he could help me. However, first he wants me to understand what his job in prison is. For this, he says I have to learn about 2D arrays. He's going to explain how they work and then give me 10 examples. If I can manage to handle them, he'll help me get out of prison.

Importance of Two-Dimensional Arrays in Java

Two-dimensional arrays in Java allow us to represent data in table format, distributed in rows and columns. Also known as 2D arrays, these matrices can be considered an evolution of the arrays we saw in the previous topic. Arrays work in only one dimension. Matrices allow storing information in table form. This arrangement makes them useful in many applications.

When using 2D arrays, it's possible to structure data in a more organized way. Additionally, they facilitate access to elements through indices, which speeds up tasks such as data manipulation and searching.

Two-dimensional arrays are also essential for implementing algorithms that require a table structure, such as some search and sorting algorithms. Thanks to their versatility, they are used in various fields, from video game creation and graphics to scientific simulations and data analysis.

Declaration and Creation of 2D Arrays

To declare a 2D array, you must specify the data type it will store and its name. Basically like this:

```
int[][] myMatrix;
```

After declaring the matrix, you need to create it, that is, allocate memory for it. For this, you must specify how many rows and columns it will have. The syntax is:

```
matrixName = new dataType[numRows][numColumns];
```

For example, to create an integer matrix with 3 rows and 4 columns:

```
myMatrix = new int[3][4];
```

Just like with arrays, I've started by explaining step by step how to declare, then create, and finally assign values. I think this is the most graphic way to understand how Java syntax works. Although, in practice, the simplest thing is to do everything at once in a single line. We save code and everything becomes much more organized. Let's see how it's done.

Declaration, Creation, and Value Assignment in a Single Line

```
int[][] myMatrix = new int[3][4];
```

I think the example explains itself. In the first part of the line, before the equal sign, we declare the matrix and name it as we see fit. After the equal sign, we create the matrix using new plus the data type and, in brackets, we assign the number of rows and columns we need.

Accessing Elements of a 2D Matrix

To access a specific element in a 2D matrix, you must use two indices: one for the row and another for the column. The basic syntax is:

```
matrixName[row][column]
```

Imagine you have a matrix called "myMatrix" of size 3x3 and you want to access the element in the second row and third column. You might think to do it like this:

```
int value = myMatrix[2][3];
```

But you'd be wrong. Remember that, just like with arrays, we don't start from 1, but from 0. Therefore, to access the value in the second row and third column, we would do it like this:

```
int value = myMatrix[1][2];
```

We can see it with another example, given the following matrix:

```
int[][] myMatrix = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9}  
};
```

```
int value = myMatrix[1][1]; // 5
```

If we want to change the value 6 to 14, we would do it as follows:

```
myMatrix[1][2] = 14;
```

Initialization of 2D Matrices

- You can initialize a matrix directly when you declare it, specifying all its elements in advance:

```
int[][] myMatrix = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9}  
};
```

- You can also declare a matrix first and assign values to its elements afterwards:

```
int[][] myMatrix = new int[3][3];  
myMatrix[0][0] = 1;  
myMatrix[0][1] = 2;  
myMatrix[0][2] = 3;  
myMatrix[1][0] = 4;  
myMatrix[1][1] = 5;  
myMatrix[1][2] = 6;  
myMatrix[2][0] = 7;  
myMatrix[2][1] = 8;  
myMatrix[2][2] = 9;
```

Traversing 2D Matrices Using Loops

To traverse arrays, we used one loop. In this case, one won't be enough since we have elements arranged in rows and columns. Therefore, we'll need to use two nested loops.

This would be the structure we would follow:

```
for (int i = 0; i < myMatrix.length; i++) { // Traverses the rows  
    for (int j = 0; j < myMatrix[i].length; j++) { // Traverses the columns  
        System.out.println("Element at [" + i + "][" + j + "]: " + myMatrix[i][j]);  
    }  
}
```

If we start analyzing the code, in the first line we find `myMatrix.length`. That's basically the number of rows our matrix has. We can leave it written like that or simply write the number of rows.

In the next line, we find the second loop. Generally, when we create a for loop, we call the variable "i"; for a second loop, the letter "j" is used. Basically, what it does is, for each row in our matrix, traverse each column.

The last line simply serves to print the data on the screen and ensure that the loop is being traversed correctly and that we've done everything right.

Sum of Elements in a Matrix

Now that we know how to traverse matrices, let's look at a couple of very basic operations we can perform using loops. The first one is the sum of elements. The structure is as follows:

```
int sum = 0;
for (int i = 0; i < myMatrix.length; i++) {
    for (int j = 0; j < myMatrix[i].length; j++) {
        sum += myMatrix[i][j];
    }
}
System.out.println("The sum of all elements is: " + sum);
```

Searching for Specific Elements

```
int[][] myMatrix = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9}  
};  
  
int searchFor = 5;  
boolean found = false;  
  
for (int i = 0; i < myMatrix.length; i++) {  
    for (int j = 0; j < myMatrix[i].length; j++) {  
        if (myMatrix[i][j] == searchFor) {  
            System.out.println("Element " + searchFor + " found at [" + i + "][" + j + "]);  
            found = true;  
            break;  
        }  
    }  
    if (found) break;  
}  
  
if (!found) {  
    System.out.println("Element " + searchFor + " not found.");  
}
```

In the first part of this code, we find the same matrix we were using in previous exercises. It's a matrix that's declared and has values already assigned. After that, we have two variables. The first is an int type and is the element we want to find. The second is a boolean type, by default it will be false, but when the program finds the element, we need to make it change to true. We could perfectly write our program without using this last variable, but there isn't always such a good opportunity to use a boolean in our examples.

Up to here, nothing we didn't know. Now we traverse the loop using nesting, just as we did in the previous examples. If you notice, it's exactly the same code and the same structure. The only thing that changes is that we've added an if, and as you can see, its operation is very simple. If it finds the element we're looking for, it will tell us exactly what position it's in and will close the loop thanks to the break we added. If it doesn't find it, it will continue searching

We use the boolean found to make the program end completely if we've already found the desired element or to display a final message saying that the element could not be found.

It's worth noting that when **dealing with boolean variables and working with an if**, we can write the conditional in the following way:

```
if (found == true) {  
  
}
```

Although, the more correct way to do it is:

```
if (found) {  
  
}
```


The same applies if it's false, we can do it this way:

```
if (found == false) {  
  
}
```

But it's recommended to do it like this:

```
if (!found) {  
  
}
```

This is due to three reasons:

- **Clarity and Readability:**

It's clearer and more concise to write `if (boolean)` than `if (boolean == true)`.

The expression `if (boolean)` is easier to read and understand because it's directly verifying the boolean condition.

- **Avoiding Common Errors:**

Using `if (boolean == true)` introduces the possibility of typographical errors, such as accidentally writing `if (boolean = true)`.

If we write the equal sign with a single equals, it will cause serious errors in the program and it won't function correctly. Often, detecting this error is quite difficult, so it's better to avoid the possibility of it happening.

- **Programming Convention:**

In Java and many other programming languages, it's a convention and good practice to simply use the boolean variable in a control structure, rather than explicitly comparing it with true.

Summary of What We've Learned

Two-dimensional matrices or 2D arrays are structures capable of organizing data in tables composed of rows and columns. This form of storage is particularly useful for programmers in numerous scenarios. This is because they allow information to be managed in an orderly and logical manner. In a two-dimensional matrix, elements can be easily accessed using two indices: one for rows and another for columns.

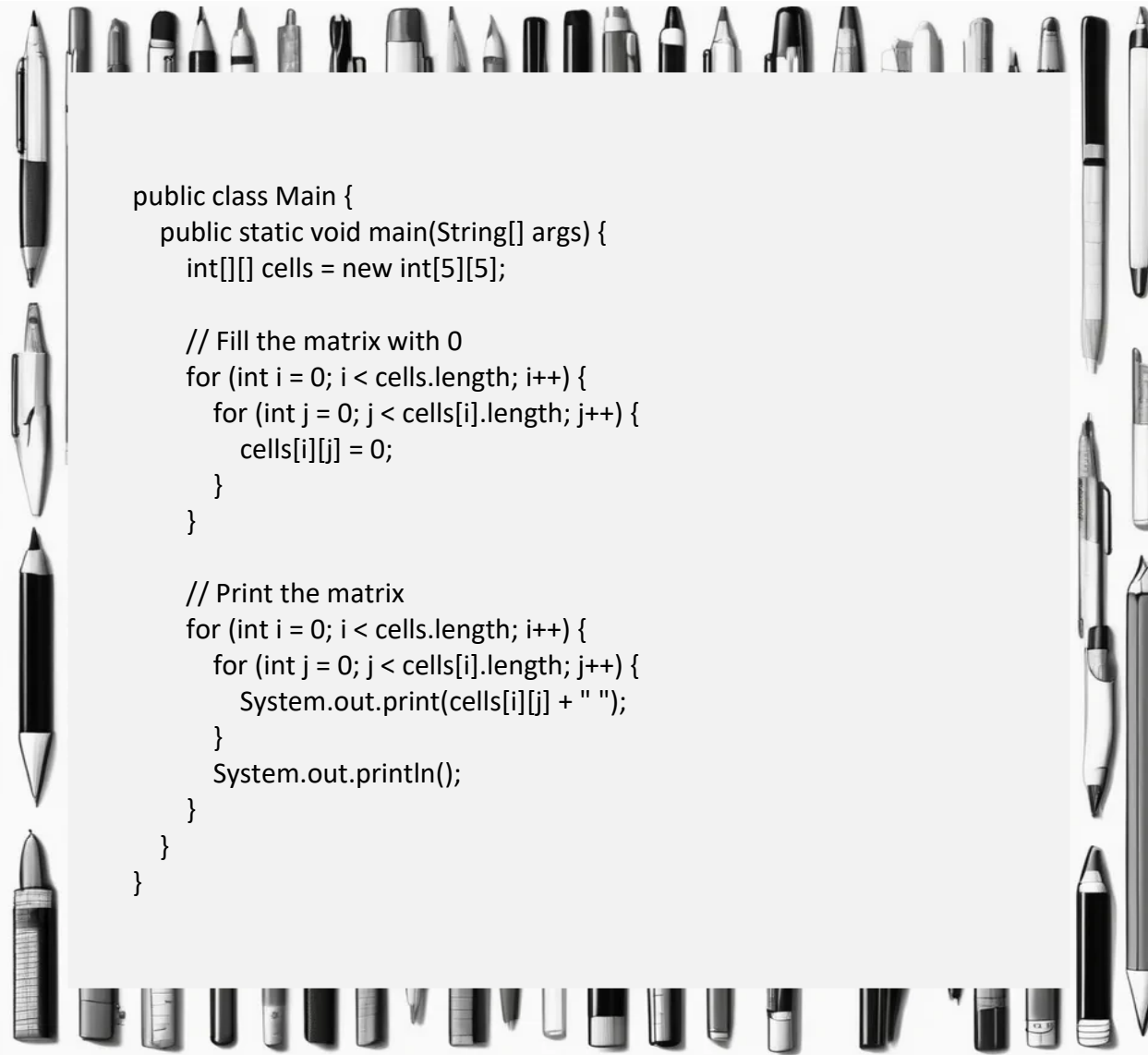
Many search and sorting algorithms use 2D arrays, as they allow working with data sets structured in two dimensions. For example, they can be used to represent game boards, graphs, and even simple databases, where ease of access and manipulation of data is essential.

But it doesn't stop there; two-dimensional matrices are characterized by their flexibility and ease of implementation, which makes them ideal for a wide range of applications. So, in summary, 2D arrays constitute a powerful and versatile tool that, in some cases, simplifies programming and provides solutions for complex problems.

Practical exercises

Exercise 1. Create a 5x5 2D matrix that represents the cells of a prison. Fill all positions with the value 0, indicating that all cells are empty.

Solution:



```
public class Main {  
    public static void main(String[] args) {  
        int[][] cells = new int[5][5];  
  
        // Fill the matrix with 0  
        for (int i = 0; i < cells.length; i++) {  
            for (int j = 0; j < cells[i].length; j++) {  
                cells[i][j] = 0;  
            }  
        }  
  
        // Print the matrix  
        for (int i = 0; i < cells.length; i++) {  
            for (int j = 0; j < cells[i].length; j++) {  
                System.out.print(cells[i][j] + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```

Exercise 2. Let's assume that cells [0][1], [2][2], and [4][3] are occupied by prisoners. Update the matrix from the previous exercise to reflect this using the value 1 for occupied cells.

Solution:

```
public class Main {
    public static void main(String[] args) {
        int[][] cells = new int[5][5];

        // Fill the matrix with 0
        for (int i = 0; i < cells.length; i++) {
            for (int j = 0; j < cells[i].length; j++) {
                cells[i][j] = 0;
            }
        }

        // Mark the occupied cells
        cells[0][1] = 1;
        cells[2][2] = 1;
        cells[4][3] = 1;

        // Print the matrix
        for (int i = 0; i < cells.length; i++) {
            for (int j = 0; j < cells[i].length; j++) {
                System.out.print(cells[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

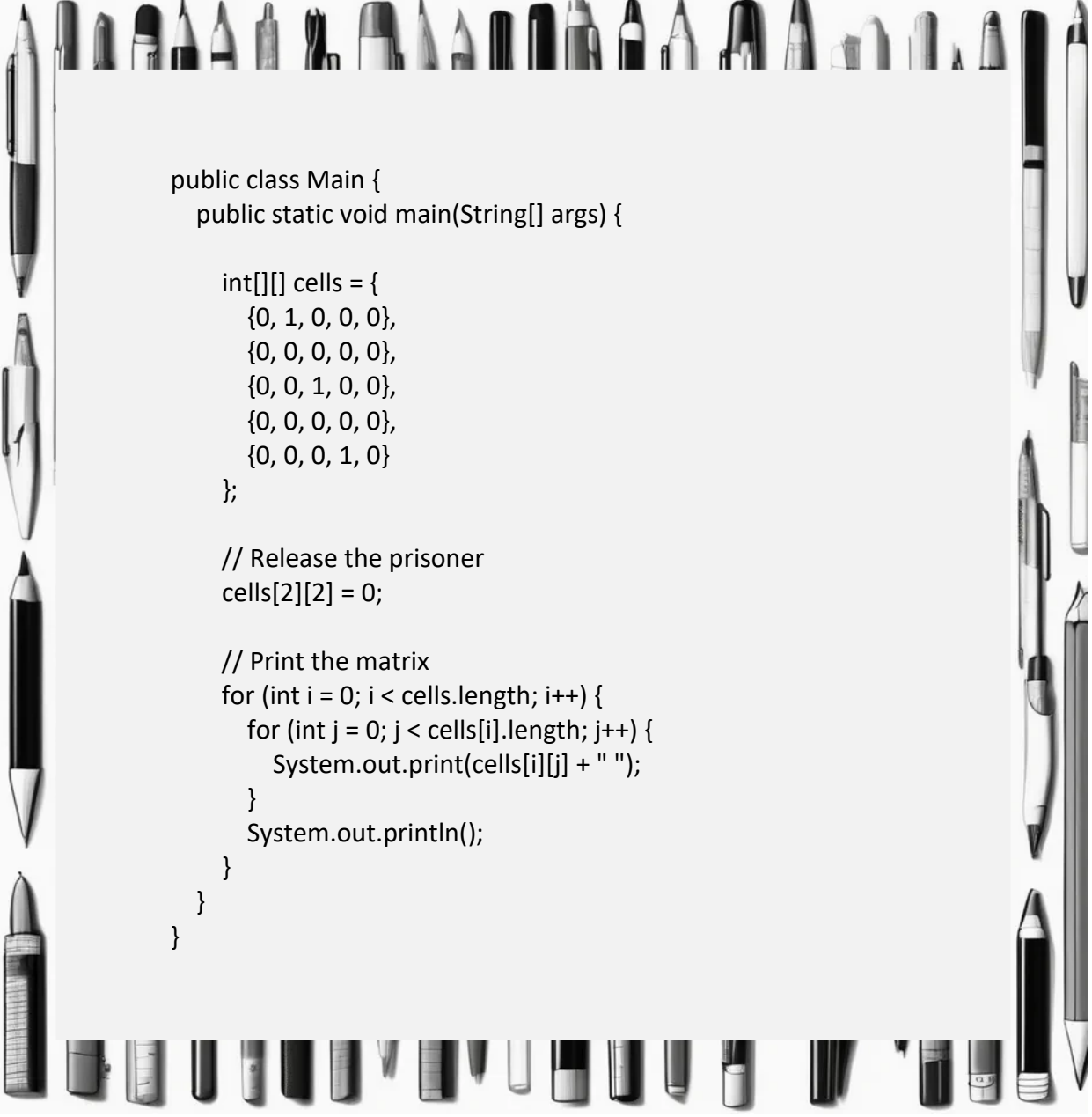
Exercise 3. Count how many cells are occupied in the matrix created in exercise 2.

Solution:

```
public class Main {  
    public static void main(String[] args) {  
  
        int[][] cells = {  
            {0, 1, 0, 0, 0},  
            {0, 0, 0, 0, 0},  
            {0, 0, 1, 0, 0},  
            {0, 0, 0, 0, 0},  
            {0, 0, 0, 1, 0}  
        };  
  
        int counter = 0;  
  
        // Count occupied cells  
        for (int i = 0; i < cells.length; i++) {  
            for (int j = 0; j < cells[i].length; j++) {  
                if (cells[i][j] == 1) {  
                    counter++;  
                }  
            }  
        }  
  
        System.out.println("Occupied cells: " + counter);  
    }  
}
```

Exercise 4. Release the prisoner from cell [2][2] in the matrix from exercise 2 and update the matrix.

Solution:



```
public class Main {
    public static void main(String[] args) {

        int[][] cells = {
            {0, 1, 0, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 1, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 1, 0}
        };

        // Release the prisoner
        cells[2][2] = 0;

        // Print the matrix
        for (int i = 0; i < cells.length; i++) {
            for (int j = 0; j < cells[i].length; j++) {
                System.out.print(cells[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Exercise 5. Find all empty cells in the matrix from exercise 2 and display their positions.

Solution:

```
public class Main {
    public static void main(String[] args) {

        int[][] cells = {
            {0, 1, 0, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 1, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 1, 0}
        };

        System.out.println("Empty cells:");

        // Find empty cells
        for (int i = 0; i < cells.length; i++) {
            for (int j = 0; j < cells[i].length; j++) {
                if (cells[i][j] == 0) {
                    System.out.println "[" + i + "]" + j + " ";
                }
            }
        }
    }
}
```


Exercise 6. Transfer a prisoner from cell [4][3] to cell [1][4] in the matrix from exercise 2.

Solution:

```
public class Main {  
    public static void main(String[] args) {  
  
        int[][] cells = {  
            {0, 1, 0, 0, 0},  
            {0, 0, 0, 0, 0},  
            {0, 0, 1, 0, 0},  
            {0, 0, 0, 0, 0},  
            {0, 0, 0, 1, 0}  
        };  
  
        // Transfer the prisoner  
        cells[4][3] = 0;  
        cells[1][4] = 1;  
  
        // Print the matrix  
        for (int i = 0; i < cells.length; i++) {  
            for (int j = 0; j < cells[i].length; j++) {  
                System.out.print(cells[i][j] + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```


Exercise 7. Count how many occupied neighboring cells the cell [2][2] has in the matrix from exercise 2.


Solution:

```
public class Main {  
    public static void main(String[] args) {  
  
        int[][] cells = {  
            {0, 1, 0, 0, 0},  
            {0, 0, 0, 0, 0},  
            {0, 0, 1, 0, 0},  
            {0, 0, 0, 0, 0},  
            {0, 0, 0, 1, 0}  
        };  
  
        int[][] directions = {  
            {-1, -1}, {-1, 0}, {-1, 1},  
            {0, -1},      {0, 1},  
            {1, -1}, {1, 0}, {1, 1}  
        };  
  
        int x = 2, y = 2;  
        int counter = 0;  
  
        // Count occupied neighboring cells  
        for (int[] dir : directions) {  
            int newX = x + dir[0];  
            int newY = y + dir[1];
```

```
        if (newX >= 0 && newX < cells.length && newY >= 0 && newY <
cells[0].length) {
            if (cells[newX][newY] == 1) {
                counter++;
            }
        }
    }

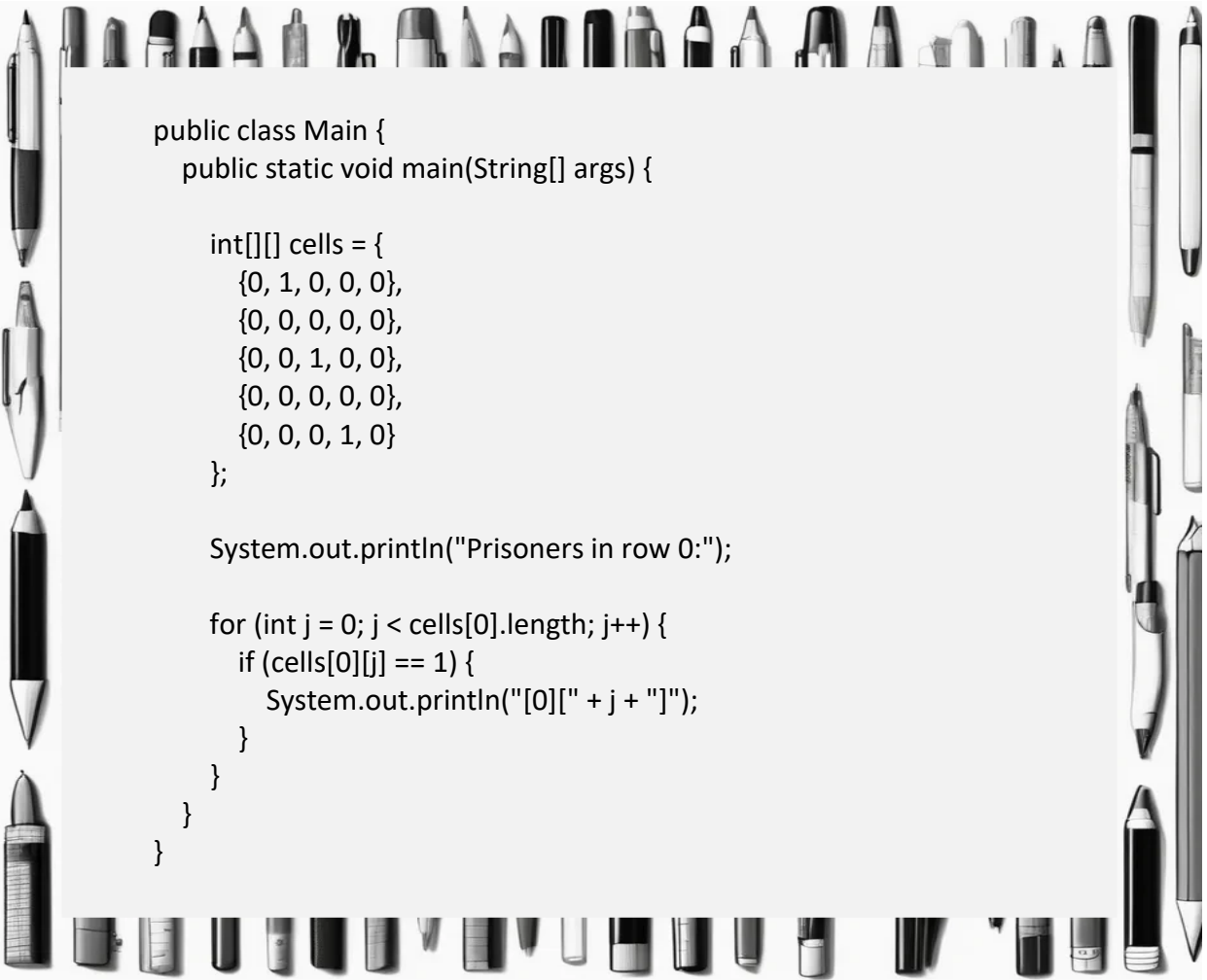
    System.out.println("Occupied neighboring cells of [2][2]: " +
counter);
}
```

 **Initialization and Declaration:** Always initialize your matrix correctly after declaring it. You can do it in a single line to keep the code clean and easy to read.

 **Use of Nested Loops:** Use nested loops to traverse and manipulate elements in the matrix. The outer loop iterates through rows and the inner loop iterates through columns. This is essential for performing operations on each element of the matrix.

Exercise 8. Find all prisoners in row 0 of the matrix from exercise 2 and display their positions.

Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        int[][] cells = {  
            {0, 1, 0, 0, 0},  
            {0, 0, 0, 0, 0},  
            {0, 0, 1, 0, 0},  
            {0, 0, 0, 0, 0},  
            {0, 0, 0, 1, 0}  
        };  
  
        System.out.println("Prisoners in row 0:");  
  
        for (int j = 0; j < cells[0].length; j++) {  
            if (cells[0][j] == 1) {  
                System.out.println("[0][" + j + "]");  
            }  
        }  
    }  
}
```



Check Boundaries: Make sure you don't exceed the matrix limits.
Always verify that your indices are within the valid range.

9. Rotate the matrix from exercise 2 90 degrees clockwise.

Solution:

```
public class Main {
    public static void main(String[] args) {

        int[][] cells = {
            {0, 1, 0, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 1, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 1, 0}
        };

        int[][] rotated = new int[cells[0].length][cells.length];

        for (int i = 0; i < cells.length; i++) {
            for (int j = 0; j < cells[i].length; j++) {
                rotated[j][cells.length - 1 - i] = cells[i][j];
            }
        }

        // Print the rotated matrix
        for (int i = 0; i < rotated.length; i++) {
            for (int j = 0; j < rotated[i].length; j++) {
                System.out.print(rotated[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Exercise 10. Check if there is an escape pattern in the matrix from exercise 2, defined as three consecutive occupied cells in any row.

Solution:

```
public class Main {
    public static void main(String[] args) {

        int[][] cells = {
            {0, 1, 0, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 1, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 1, 0}
        };

        boolean escapeFound = false;

        for (int i = 0; i < cells.length && !escapeFound; i++) {
            int consecutive = 0;
            for (int j = 0; j < cells[i].length; j++) {
                if (cells[i][j] == 1) {
                    consecutive++;
                    if (consecutive == 3) {
                        escapeFound = true;
                        break;
                    }
                } else {
                    consecutive = 0;
                }
            }
        }

        if (escapeFound) {
            System.out.println("Escape pattern found!");
        } else {
            System.out.println("No escape pattern found.");
        }
    }
}
```

Chapter 9.

Create a Distraction Device. With Loops, Conditionals, and Scanner

I hadn't realized it, but I've been behind bars for almost a month and a half now. It might not be a long time, but I've already forged a great friendship with Bud. I told him about my situation, and he's willing to help me. Today, finally, he's going to tell me the plan that will allow me to get out of here. He hasn't given me any hints, so I have no idea how we're going to do it.

Bud is quite happy with my Java level. I've been a good student and I'm learning very quickly. The problem is that if I escape from prison, I can forget about continuing my progress. My future is very uncertain right now. I can only make daily plans; planning things from one week to the next is a luxury.



My fate is uncertain. Just today, I received a message from one of the guards: I've been assigned to work in the laundry. I have to report there at three o'clock, and I don't know what my duties will be. Whatever it is, it's surely better than sitting in the cell doing nothing.

I've been alone in the cell all day. It's nice to have some privacy occasionally. Solitude sometimes helps you clear your thoughts. Time has passed very quickly and, without realizing it, I'm already on my way to the laundry. I wonder how many people work there. I might be able to make new friends.

As soon as I arrived, I realized my plan to socialize was completely ruined. Those who were waiting for me were Bud and Pedro. That's when they told me they had a plan to break out of prison. They needed one more person and, thanks to my programming knowledge, I could join.

There was a small problem. They wanted to escape, but I didn't. I didn't want to spend the rest of my life on the run. I just need to get out for a while to unmask Rich. Bud knew this, so he offered me a solution. The two of them would escape, but I would return to prison after two hours. It should be enough time to get a confession from Rich, return to my cell, and have no one find out about my excursion.

It took me less than two minutes to decide. Of course, I accepted. I think these two have everything well planned out, so it seems the risk is minimal. What I didn't know was that the plan would start the next day. Bud hadn't been teaching me Java by chance. They needed a third person who knew how to program to carry out the plan.

The first part of the plan is simple. Well, at least my contribution. In the prison library, there are several computers that can access the entire security system. From there, Pedro will access the prison's alarm function and make it go off for a few minutes. Thanks to this distraction, Bud will be able to access the warden's office, where he'll get something that will help us with the next step.

It's important to access from one of the library computers, since if it's done from Pedro's or Bud's computer, they would be discovered within minutes. Both laptops are closely monitored, and every action is logged.



There's a small problem, and that is that Pedro won't have much time to hack the system. The truth is that there's a lot to program and only a few minutes to do it without being discovered. That's where I come in. I need to create five Java programs, hidden in the library PC, which Pedro will later use to work his magic.

I have exactly one hour of time in the library. But usually the computers are in high demand, so I might only be able to use them for half an hour. It doesn't matter, they've explained exactly what I need to do, so I have plenty of time.

Tomorrow is the big day. I'm a bit nervous, but tonight I'll try to rest. It's not easy to sleep under these circumstances, but I'll try to think that, if everything goes well, I'll prove my innocence and finally be able to get out of here.

Inside the Prison Library:

I arrived at the library among the first ones. My goal was to make sure I wouldn't be left without a computer. I've been lucky, as today there aren't many people wanting to use them. I have at least half an hour. It's not much time and there's quite a bit of work to do. As I mentioned before, my part is to create and hide several Java programs that Pedro will later use to enter the central system and create a distraction device.

Program 1

First, we'll need to create a Program that discovers the central system's password. The password consists of 6 digits, of which we believe we know 5. Therefore, we'll only need to check 10 combinations.

The problem is that if the data is checked too quickly, the system might detect it. So, after the ninth attempt, we need to wait at least 30 seconds. For this, we'll make the Program stop after 9 attempts, and to check the last one, we'll need to type "continue" on the keyboard.

Here's the code I used. Review it line by line and you'll see how you can understand its operation without problems:

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        // Pass is the password we don't know. It's hidden in the Program.
        // For now, we generate a variable for it and give it value 0.
        // But in the actual prison code lines it's a real password.

        int pass = 0;

        // We know all the password numbers except the last one.

        int incompletePass = 58987;

        // We declare the variable for password guessing attempts.

        int completePass;

        // We create a Scanner since we'll need to input text.

        Scanner scan = new Scanner(System.in);

        // We create a variable to continue
        // If reaching the ninth attempt Pedro should wait 30 seconds
        // Then he should type "continue"

        String continue = "continue";

        // Our Program will have 10 attempts.
        // So we create a "for" loop for it.

        for (int i = 0; i < 10; i++) {

            // This way we generate the last number that is added
            // to our incomplete password.
```

```

completePass = incompletePass*10 + i;

i++;

// When our attempt matches the real password
// A text will be displayed

if (pass == completePass) {
    System.out.println("This is the password");
    break;
}

// We need to warn Pedro if the Program has made 8 attempts
// Then he decides if he wants to try the last one

if (i == 9) {
    System.out.println("This is the ninth attempt, you must wait 30
seconds");
    System.out.println("Type continue to proceed");

    String pause = scan.nextLine();

    if (!continueStr.equals(pause)) {
        System.out.println("Error");
        System.exit(0);
    }
}

System.out.println("Password successfully obtained: " + pass);

}
}

```

Program 2

With the password, we can enter the central system; from there, we can access the alarm. This is located in a circuit where energy is generally saved and is turned off all the time. Therefore, first of all, we need to make sure that electricity is working. After this condition is met, Pedro must mark the manual fire alarm option as true. This way, he can later trigger it. Let's prepare the code properly for him.

Let's create a Program that looks for the boolean variable `alarmActivated` and marks it as true. But only if the boolean variable `electricityOn` is activated. If not, the Program will have to activate it.

Solution:

```
public class Main {  
    public static void main(String[] args) {  
  
        // Declaration of variables  
        boolean electricityOn = false;  
        boolean alarmActivated = false;  
  
        // Check the status of electricityOn  
        if (electricityOn) {  
  
            // If electricityOn is activated, we activate the alarm  
            alarmActivated = true;  
            System.out.println("Electricity is on. The alarm has been  
activated.");  
  
        } else {
```

```
// If electricityOn is not activated, we activate it and then activate the alarm
{
    electricityOn = true;
    alarmActivated = true;
    System.out.println("Electricity was not on. Electricity and alarm
have been activated.");
}

// Show final status of variables
System.out.println("Final status - Electricity: " + electricityOn + ",
Alarm: " + alarmActivated);

}
}
```

Program 3.

The alarm cannot be triggered at any time. It must be done when all guards are in their watchtowers. This way, it will take at least 60 seconds until they reach the central room to deactivate the alarm. That's the time we need the distraction to last. In the hypothetical case that only one guard is in the central room, they could turn off the alarm in a few seconds and the plan would be ruined.

The prison system marks each watchtower with a 1 if someone is inside and with a 0 if it's empty. Therefore, we must create a Program that establishes a 2D matrix representing the prison's 6 watchtowers. Then, we create a boolean variable that will be true if all watchtowers are occupied. If at least one is empty, it will be false.

Solution:

```

public class Prison {

    public static void main(String[] args) {

        // Create a 2D matrix of 6 rows and 1 column to represent the 6
        watchtowers
        int[][] watchtowers = {
            {1}, // Watchtower 1
            {1}, // Watchtower 2
            {1}, // Watchtower 3
            {1}, // Watchtower 4
            {1}, // Watchtower 5
            {1} // Watchtower 6
        };

        // Boolean variable that indicates if all watchtowers are occupied
        boolean allOccupied = true;

        // Check the status of watchtowers
        for (int i = 0; i < watchtowers.length; i++) {
            if (watchtowers[i][0] == 0) {
                allOccupied = false;
                break;
            }
        }

        // Print the result
        if (allOccupied) {
            System.out.println("All watchtowers are occupied.");
        } else {
            System.out.println("There are empty watchtowers.");
        }
    }
}

```

Program 4.

In addition to the 60 seconds of sound distraction, we'll need the complex's lights to turn off afterward. So the plan is that, once the guards deactivate the alarm, there will be an instant blackout. Our job is to create a simple Program that changes the state of a boolean variable called `turnOffLight` from `false` to `true`.

Solution:

```
public class StateChange {  
    public static void main(String[] args) {  
  
        boolean turnOffLight = false; // Initial state of turnOffLight variable  
        System.out.println("Initial state of turnOffLight: " + turnOffLight);  
  
        // Change turnOffLight state from false to true  
        turnOffLight = true;  
        System.out.println("New state of turnOffLight: " + turnOffLight);  
    }  
}
```

Program 5.

Those are all the Programs that Pedro will need to provide a good distraction for Bud. But he also needs to cover his tracks. The alarm can't go off while he's sitting in front of a computer. It would be too obvious that he's responsible. We need to create a very simple Program where the number of seconds needed before the alarm starts ringing is entered via keyboard. This way, Pedro can leave the library without anyone suspecting it was him. Depending on the situation, he'll decide how many seconds he needs.

The only condition is that the entered number must be greater than 600. That equals 10 minutes, which will give Pedro enough time to reach his cell and avoid any suspicion.

```
import java.util.Scanner;
public class Prison {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Ask the user to enter a number
        System.out.print("Enter a number greater than 600: ");
        int number = scanner.nextInt();

        // Verify if the number is greater than 600
        if (number > 600) {
            System.out.println("The number " + number + " is correct.");
        } else {
            System.out.println("The number is not correct. It must be greater than
600.");
        }

        // Close the scanner
        scanner.close();
    }
}
```

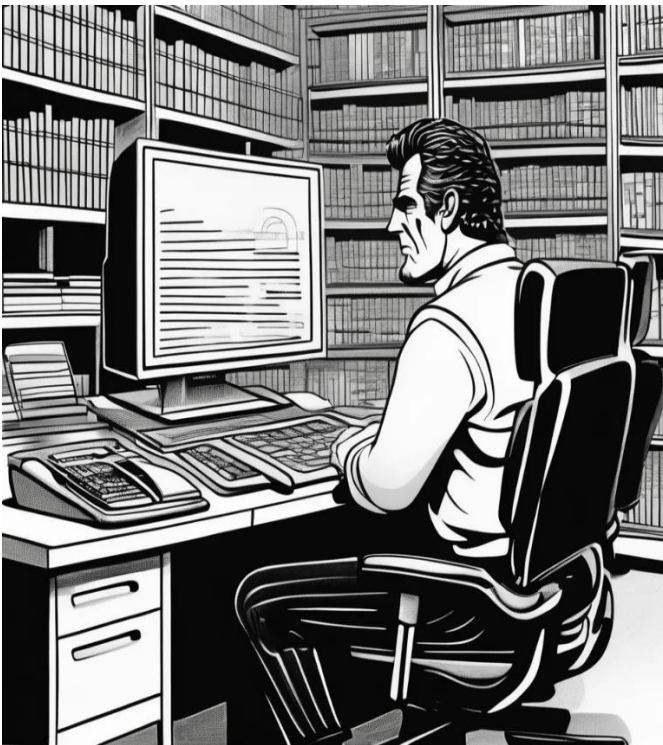

Chapter 10.

Working with Date and Time

My part is done. After creating the code for the Programs that Pedro needs, I've hidden them in a system folder. It didn't even take me half an hour to do it. I don't need the computer anymore, so I'll go read for a while.

I'm quite relieved because, now that this part of the plan works out, it doesn't depend on me. I just have to wait and see what happens. Even the risk of getting caught is minimal. They could only discover me if Pedro or Bud betrayed me, but I trust them.

I've been informed that Chani's cousin is coming to see me again this afternoon, so I'll tell him that I'll be getting out soon. My excursion will be short, just a few hours, but I don't know exactly which day yet. Therefore, we'll need to stay in touch.



Tomorrow afternoon, Pedro will come to the library, and, in less than ten minutes, he'll have to use my Programs to create the distraction device. Before the alarm starts ringing, he'll have a few minutes to leave the library and return to his cell. This way, no one will suspect him.

Once the alarm goes off, the prison officers will dedicate all their effort to turning it off and checking that the security system is still active. Bud will use that time to manage to open the cell and cross the first corridor, where there shouldn't be any guard. Although he'll have to be careful with the cameras.

In the second corridor, there's a guard who never leaves his post but patrols in circles. So, knowing his surveillance pattern, Bud will have to dodge him to avoid being seen. Remember that when they manage to deactivate the alarm, the lights will go out. That will give Bud even more time. In theory, it will be enough time for him to enter the office unseen.

The riskiest part of the entire plan will be leaving the office. Bud will have to monitor the guards' surveillance patterns to exit at exactly the right moment. He'll also need to take into account the camera patterns, exactly the same as he did to reach the office.

To carry out this complicated plan, Bud will have to use some Java concepts that I haven't learned yet, like working with date and time. I won't need to know how to do that, but I'd like to learn it anyway.

I've asked Bud to teach me, and he's delighted. Maybe in the end he's not as calm as he seems and perhaps, he wants to keep his mind occupied. Whatever the case, I appreciate it. I like learning new things and, truthfully, it's also good for me to keep my mind busy.

Theory for Working with Dates and Times in Java

The theoretical explanation in this topic will be a bit different from the previous ones. Much more summarized. The first reason is that I don't think it's necessary to mention the importance of date and time in any application or Program. We all have a mobile phone or computer and know that many functions of our devices don't even work without correct time data.

The second reason is that, at this point, you're already able to understand and write a large part of the code used in Java. You understand, among other things, what variables are, the main structures, and the order of operation of Programs. Therefore, this part will be a piece of cake for you.

The third reason is that the syntax of this topic is not complex at all. It can't even be called a structure. We'll simply use one line of code or another depending exactly on the type of date or time we need to obtain.

The LocalTime, LocalDate, and LocalDateTime Classes

- LocalTime represents only the time of day, without including any information about the date or time zone.
- LocalDate represents only a date, without including any information about time or time zone.
- LocalDateTime combines the information from LocalDate (date) and LocalTime (time), representing both date and time, but without including time zone information.

We import them as follows:

- `import java.time.LocalTime;`
- `import java.time.LocalDate;`
- `import java.time.LocalDateTime;`

Now, I'm going to show you, point by point, the code you'll need to use depending on each specific situation.

- **Get the current date**

To get the current date in Java, use the `LocalDate` class. This class represents a date without time, in ISO format (yyyy-mm-dd).

```
LocalDate currentDate = LocalDate.now();
```

- **Get the current time**

To get the current time, use the `LocalTime` class, which represents a time without date in hh:mm format.

```
LocalTime currentTime = LocalTime.now();
```

- **Get the current date and time**

The `LocalDateTime` class combines `LocalDate` and `LocalTime` to represent a date and time in the same object.

```
LocalDateTime currentDateTime = LocalDateTime.now();
```

- **Create a specific date**

You can create a specific date using the `of` method of the `LocalDate` class, which requires year, month, and day as parameters.

```
LocalDate date = LocalDate.of(2024, 5, 29);
```

- **Create a specific time**

Similarly, `LocalTime` also has an `of` method to create a specific time, using hours, minutes, and seconds as parameters.

```
LocalTime time = LocalTime.of(14, 30, 45);
```

- **Add days to a date**

You can add days to a date using the `plusDays` method of the `LocalDate` class.

```
LocalDate newDate = currentDate.plusDays(10); // Ten days are added in this example
```

- **Subtract days from a date**

```
LocalDate newDate = currentDate.minusDays(10); // Ten days are subtracted
```

- **Add hours to a time**

```
LocalTime newTime = currentTime.plusHours(3); // 3 hours are added in this example
```

- **Subtract hours from a time**

To subtract hours from a time, use the `minusHours` method of the `LocalTime` class.

```
LocalTime nuevaHora = horaActual.minusHours(3); // 3 son las horas restadas en este ejemplo
```

- **Compare two dates**

To compare two dates, `LocalDate` provides methods such as `isAfter`, `isBefore`, and `isEqual`.

```
boolean isAfter = date1.isAfter(date2);
```

```
boolean isBefore = date1.isBefore(date2);
```

```
boolean isEqual = date1.isEqual(date2);
```

- **Format a date**

// To format a date in a specific format, use the `DateTimeFormatter` class.

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
```

```
String formattedDate = date.format(formatter);
```

- **Get the difference between two dates**

To calculate the difference between two dates, use the `Period` class.

```
Period difference = Period.between(startDate, endDate);
```

Summary of What We've Learned

In the Java Programming language, if you need to get the current date, you can use the `LocalDate` class, while to get the current time, the `LocalTime` class is used. When you need to handle both date and time together, the `LocalDateTime` class is the appropriate option. To set a specific date, you can use the `of` method of `LocalDate`. Similarly, to define a specific time, the `of` method of `LocalTime` is used.

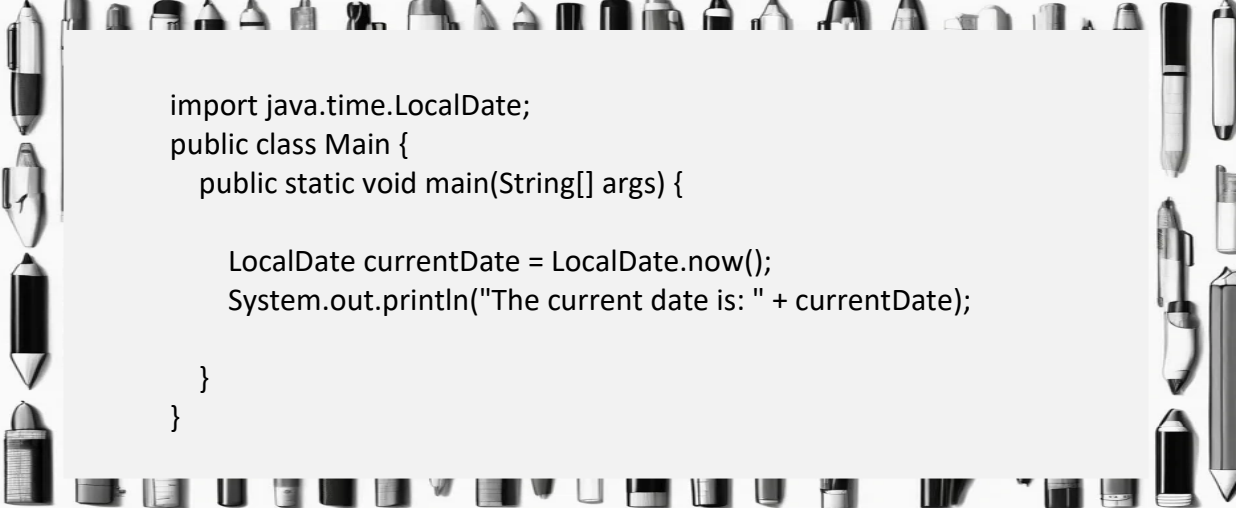
The `plusDays` and `minusDays` methods of `LocalDate` allow you to add or subtract days from a date, respectively. Likewise, with the `plusHours` and `minusHours` methods of `LocalTime`, you can add or subtract hours from a specific time.

The `LocalDate` class provides methods such as `isAfter`, `isBefore`, and `isEqual` to compare two dates. To format a date in a specific style, the `DateTimeFormatter` class is used. Finally, to determine the difference between two dates, the `Period` class is employed.

Practical exercises

Exercise 1. Get the current date

Solution:



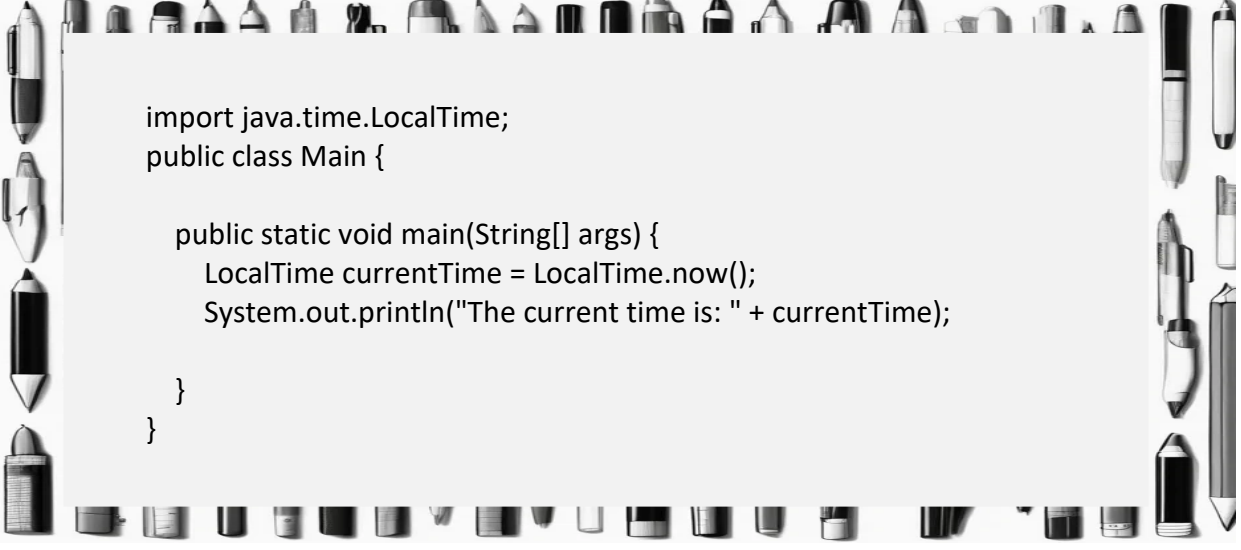
```
import java.time.LocalDate;
public class Main {
    public static void main(String[] args) {

        LocalDate currentDate = LocalDate.now();
        System.out.println("The current date is: " + currentDate);

    }
}
```

Exercise 2. Get the current time

Solution:



```
import java.time.LocalTime;
public class Main {

    public static void main(String[] args) {
        LocalTime currentTime = LocalTime.now();
        System.out.println("The current time is: " + currentTime);

    }
}
```


Exercise 3. Get the current date and time

Solution:

```
import java.time.LocalDateTime;
public class Main {

    public static void main(String[] args) {
        LocalDateTime currentDateTime = LocalDateTime.now();
        System.out.println("The current date and time is: " +
            currentDateTime);
    }
}
```

Exercise 4. Create a specific date

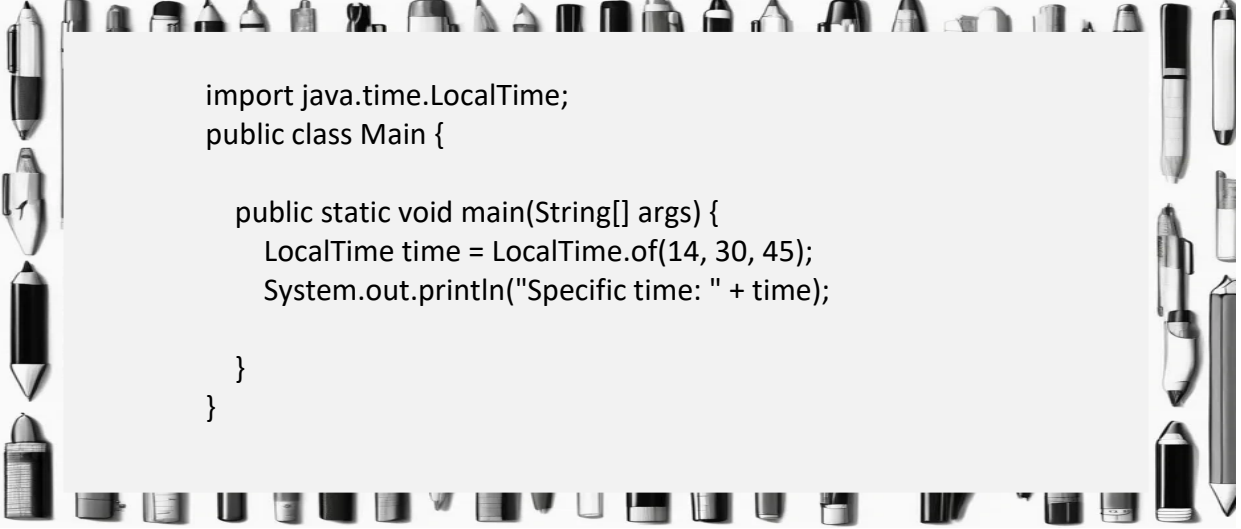
Solution:

```
import java.time.LocalDate;
public class Main {

    public static void main(String[] args) {
        LocalDate date = LocalDate.of(2024, 5, 29);
        System.out.println("Specific date: " + date);
    }
}
```

Exercise 5. Create a specific time

Solution:

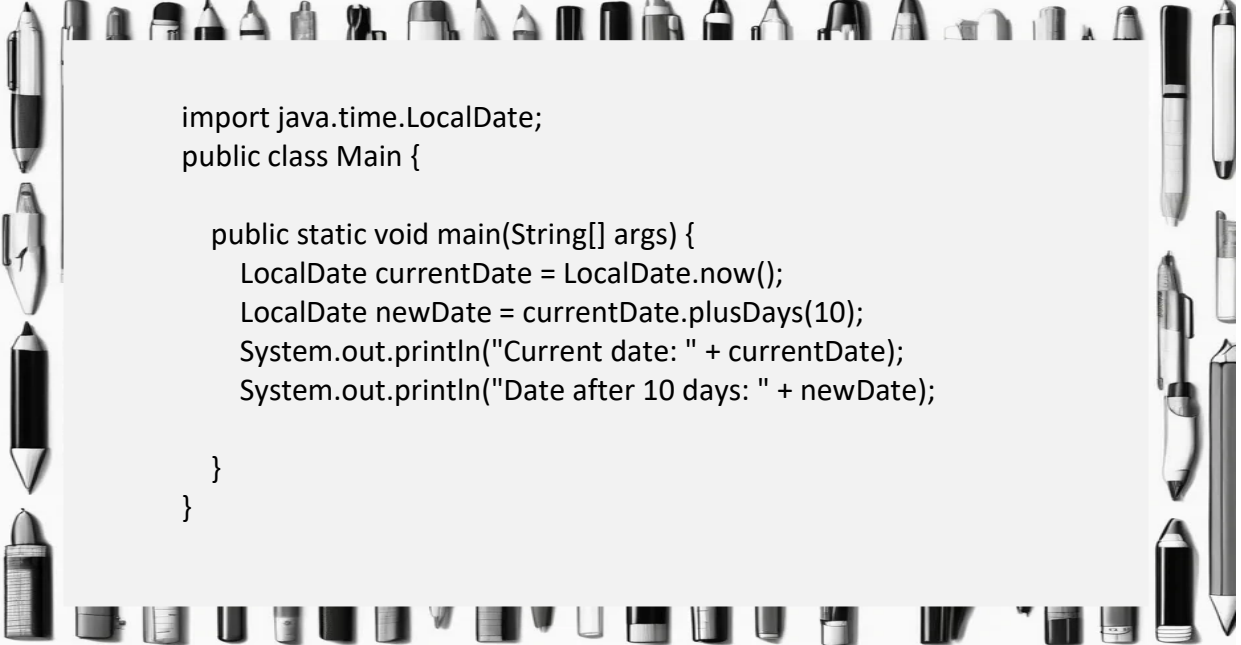


```
import java.time.LocalDateTime;
public class Main {

    public static void main(String[] args) {
        LocalDateTime time = LocalDateTime.of(14, 30, 45);
        System.out.println("Specific time: " + time);
    }
}
```

Exercise 6. Add days to a date

Solution:

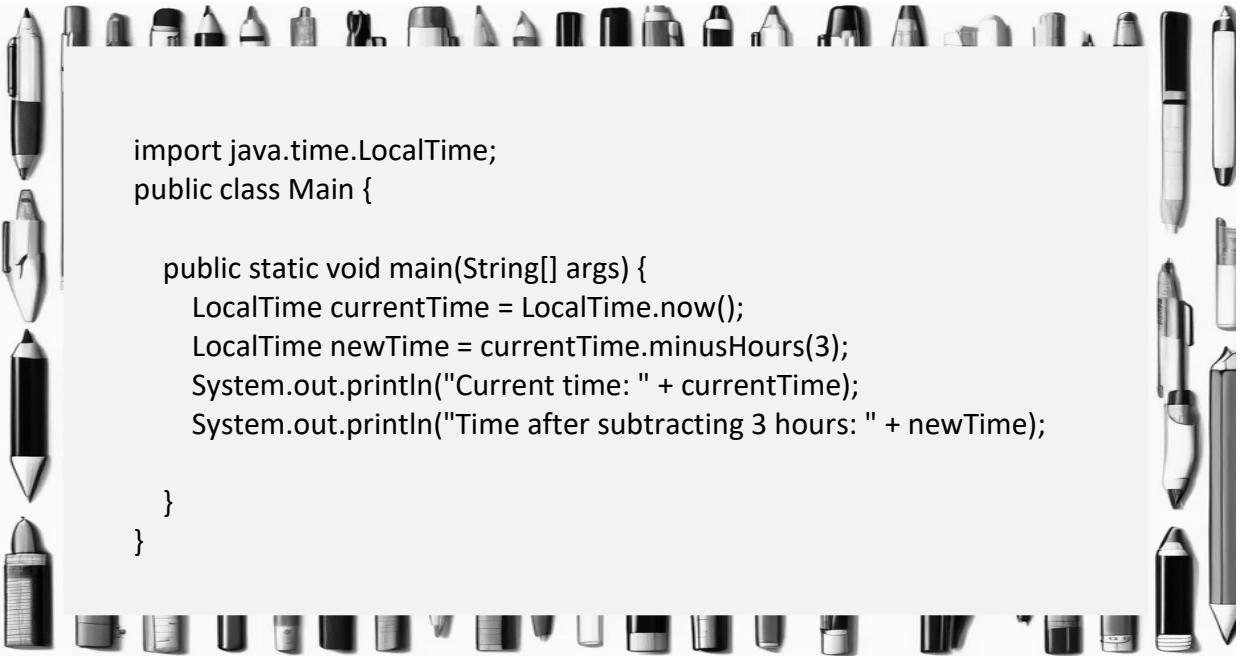


```
import java.time.LocalDate;
public class Main {

    public static void main(String[] args) {
        LocalDate currentDate = LocalDate.now();
        LocalDate newDate = currentDate.plusDays(10);
        System.out.println("Current date: " + currentDate);
        System.out.println("Date after 10 days: " + newDate);
    }
}
```

Exercise 7. Subtract hours from a time

Solution:

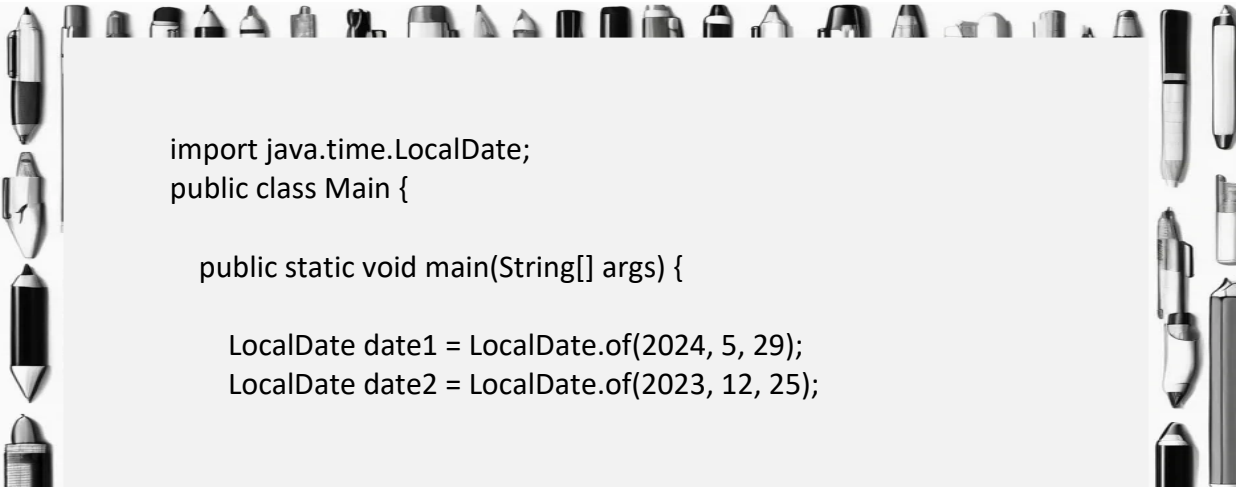


```
import java.time.LocalTime;
public class Main {

    public static void main(String[] args) {
        LocalTime currentTime = LocalTime.now();
        LocalTime newTime = currentTime.minusHours(3);
        System.out.println("Current time: " + currentTime);
        System.out.println("Time after subtracting 3 hours: " + newTime);
    }
}
```

Exercise 8. Compare two dates

Solution:



```
import java.time.LocalDate;
public class Main {

    public static void main(String[] args) {

        LocalDate date1 = LocalDate.of(2024, 5, 29);
        LocalDate date2 = LocalDate.of(2023, 12, 25);
    }
}
```

```
        if (date1.isAfter(date2)) {
            System.out.println(date1 + " is after " + date2);
        } else if (date1.isBefore(date2)) {
            System.out.println(date1 + " is before " + date2);
        } else {
            System.out.println(date1 + " is equal to " + date2);
        }
    }
}
```

Exercise 9. Format a date

Solution:

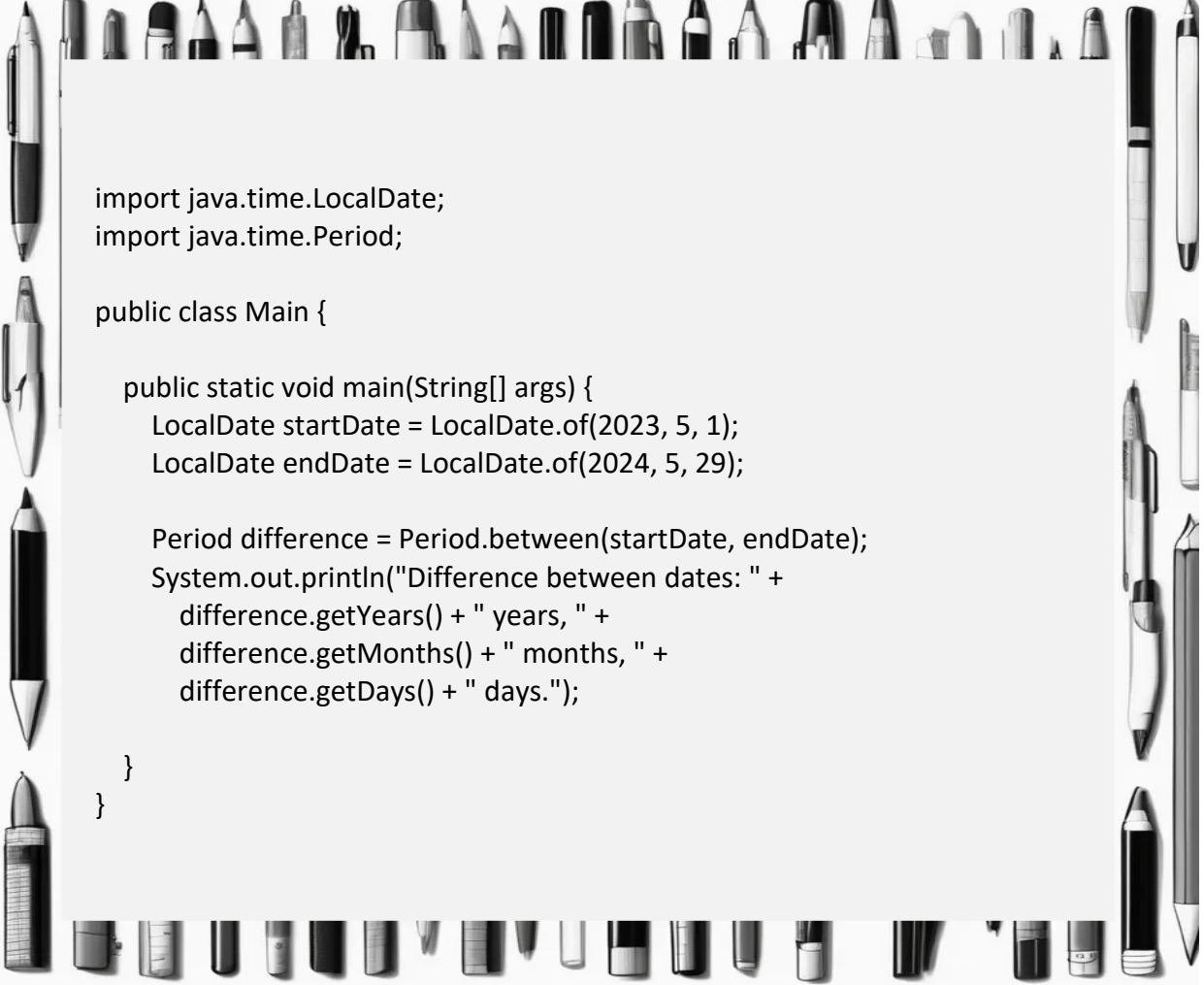
```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public class Main {
    public static void main(String[] args) {

        LocalDate date = LocalDate.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
        String formattedDate = date.format(formatter);
        System.out.println("Formatted date: " + formattedDate);
    }
}
```

Exercise 10. Get the difference between two dates

Solution:



```
import java.time.LocalDate;
import java.time.Period;

public class Main {

    public static void main(String[] args) {
        LocalDate startDate = LocalDate.of(2023, 5, 1);
        LocalDate endDate = LocalDate.of(2024, 5, 29);

        Period difference = Period.between(startDate, endDate);
        System.out.println("Difference between dates: " +
            difference.getYears() + " years, " +
            difference.getMonths() + " months, " +
            difference.getDays() + " days.");
    }
}
```

Chapter 11.

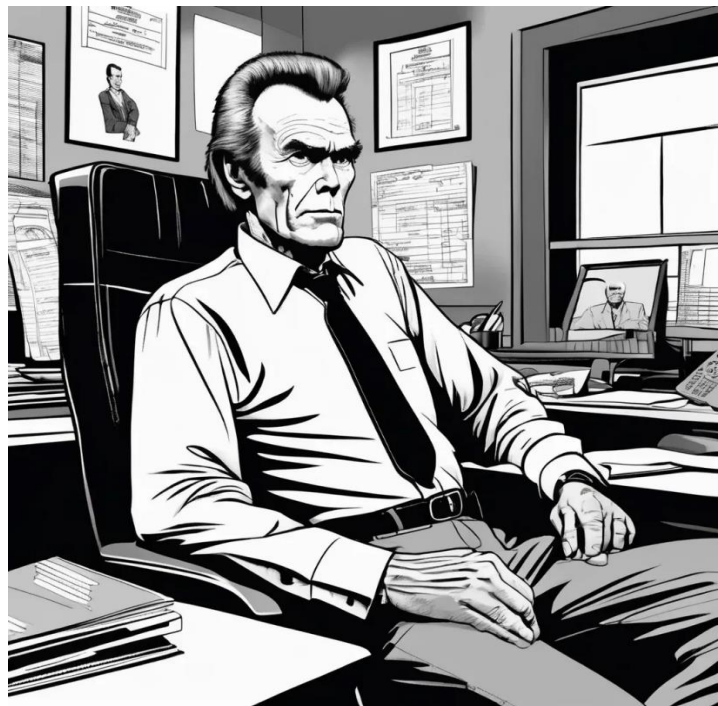
Back to Hacking the Security System

The day has arrived, the escape has officially begun and, from today, there's no turning back. Pedro must already be in the library, so if everything happens as we expect, the alarm will sound shortly. Bud is already prepared with his computer; without it, he couldn't reach the warden's office. We don't say anything; Bud stares at the screen intently, deeply focused.

They don't let him have the laptop in his cell every day, but today he made sure it would be there. He managed to make the kitchen's ventilation system stop working and, in theory, he's trying to repair it.

Breaking into the warden's office has a single objective: to access the warden's notebook, where he writes everything down manually. He's an elderly gentleman who doesn't like computers much. His name is Vicente. The information we're trying to get is very valuable, as it will help us with the next phases of the plan.

Finally, the moment arrived: the alarm went off. It was much louder than I had imagined. I saw Bud start typing at full speed. The raid on Vicente's office had begun.



To make the wait more pleasant, we're going to put ourselves in Bud's shoes and program all the code necessary to carry out this part of the plan. If we want to reach the end, we need to complete five steps and also have some luck.

Exercise 1. Bud needs to unlock the main door of his cell. To do this, he must write a Program that verifies a security code before inserting it. If the wrong code is entered, the door could lock. The correct code is stored in an array and must be compared with a user input.

Instructions:

- Create an array containing the security code, for example: {1, 4, 5, 7, 9}.
- Ask the user to enter a five-digit code.
- Compare the entered code with the code stored in the array.
- If the code is correct, display a success message. If not, display an error message.

Solution:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        int[] securityCode = {1, 4, 5, 7, 9};

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the 5-digit security code:");

        int[] enteredCode = new int[5];

        for (int i = 0; i < 5; i++) {
            enteredCode[i] = scanner.nextInt();
        }
    }
}
```

```

        boolean isCorrect = true;

        for (int i = 0; i < 5; i++) {
            if (enteredCode[i] != securityCode[i]) {
                isCorrect = false;
                break;
            }
        }

        if (isCorrect) {
            System.out.println("Correct code. The door is unlocked.");
        } else {
            System.out.println("Incorrect code. Try again.");
        }
    }
}

```

Exercise 2. Avoiding the Security Camera. Bud needs to move through the first corridor without being detected by a security camera that turns on and off at regular intervals. He must write a Program that determines if he can move at a given moment.

Actually, Bud's Program is a bit more elaborate than what we're going to do, since he doesn't know in which seconds the cameras turn on. He gets that information directly thanks to his connection to the central system, and his Program works with that data.

Instructions:

- Use a for loop to simulate the seconds in a minute (0 to 59).
- The camera is on during the first 10 seconds of each minute and off during the next 50 seconds.
- Ask the user to enter a second when they plan to move.
- Determine if the camera will be on or off at that second and display a corresponding message.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the second when you plan to move (0-59):");
        int second = scanner.nextInt();

        if (second >= 0 && second < 10) {
            System.out.println("The camera is on. Wait a bit longer.");
        } else if (second >= 10 && second < 60) {
            System.out.println("The camera is off. You can move!");
        } else {
            System.out.println("Invalid input. Try again.");
        }
    }
}
```

Exercise 3. There's another small problem: the office is located in an area separate from the main pavilion. This means that the fire alarm won't sound here. Therefore, the guard will remain there, right in front of the door we need to access. Bud will have to avoid him. The idea is simple yet very effective: the plan consists of increasing the corridor's temperature until the guard can't stand it anymore and is forced to go to the central room to lower the degrees. This will give Bud enough time to enter and exit the office unseen. The system functions that don't require a high level of security, like thermostat regulation, aren't controlled, so Bud will be able to modify the temperature without anyone detecting it.

Create a Program that meets the following requirements:

- Enter the initial corridor temperature
- Enter the number of degrees you want to increase
- Add the desired increase to the initial temperature
- Display the new corridor temperature after the adjustment

Solution:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        // Create a scanner to read user input
        Scanner scanner = new Scanner(System.in);

        // Ask user for initial temperature
        System.out.print("Enter the initial corridor temperature (in Celsius):");
        double initialTemperature = scanner.nextDouble();

        // Ask user for desired temperature increase
```

```
System.out.print("Enter the number of degrees to increase: ");
double increase = scanner.nextDouble();

// Calculate the new temperature
double newTemperature = initialTemperature + increase;

// Display the result
System.out.println("The new corridor temperature is: " +
newTemperature + "°C");

// Close the scanner
scanner.close();

}
}
```

Exercise 4. Password to Enter the Office. Bud needs to generate a security code based on the current time (minutes and seconds). This code changes every minute and must be entered correctly to unlock a door.

Instructions:

- Use the `LocalTime` class to get the current time.
- Generate a security code by adding the current minutes and seconds.
- Show the generated code to the user.
- Ask the user to enter the displayed code.
- Verify if the entered code is correct and display an appropriate message.

Solution:

```
import java.time.LocalTime;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        // Get current time
        LocalTime now = LocalTime.now();
        int minutes = now.getMinute();
        int seconds = now.getSecond();

        // Generate security code by adding minutes and seconds
        int generatedCode = minutes + seconds;

        // Display the generated code
        System.out.println("The generated security code is: " +
            generatedCode);

        // Ask user to enter the displayed code
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the security code to validate:");
        int enteredCode = scanner.nextInt();

        // Verify if entered code is correct
        if (enteredCode == generatedCode) {
            System.out.println("Correct code. The door is open.");
        } else {
            System.out.println("Incorrect code. Try again.");
        }
    }
}
```

Exercise 5. Once inside, Bud will need to grab the notebook and get out as quickly as possible. Although probably, by the time he finds it, the guard will have already returned to his post. To leave unseen, the guard needs to be moved again. In the same room where the guard lowered the corridor temperature, there's a coffee machine. It's broken and, every time it starts its self-cleaning process, it makes a terrible noise. It needs to be turned off manually. So the idea is to activate this self-cleaning and force the guard to go turn it off.

Fortunately for Bud, this Program is already created, since the coffee maker comes programmed from the factory; even so, we are going to create one. Since we have barely used switch-case conditionals, I think it's a good time for you to review them.

Instructions:

- Implement a simple menu that allows the user to select between making coffee or cleaning the coffee maker.
- When the user chooses the option to clean the coffee maker, display messages indicating that cleaning has started and finished.
- Allow the user to repeat the process as many times as desired until selecting the option to exit the Program.

Solution:

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);

        // Variable to control if the Program keeps running
        boolean continue = true;
```

```

// Main Program loop
while (continue) {
    // Display menu
    System.out.println("\n--- Coffee Maker Menu ---");
    System.out.println("1. Make coffee");
    System.out.println("2. Activate manual cleaning function");
    System.out.println("3. Exit");
    System.out.print("Choose an option: ");

    // Read user's selected option
    int option = scanner.nextInt();

    // Process selected option
    switch (option) {
        case 1:
            System.out.println("Making coffee... Enjoy your drink!");
            break;
        case 2:
            System.out.println("Starting manual cleaning function...");
            // Cleaning process simulation
            System.out.println("Coffee maker is cleaning...");
            System.out.println("Cleaning finished! Coffee maker is ready to
use.");
            break;
        case 3:
            System.out.println("Exiting Program. Goodbye!");
            continue = false; // Exit loop
            break;
        default:
            System.out.println("Invalid option. Please choose an option from the
menu.");
            break;
    }
}

// Close scanner
scanner.close();
}

```

Chapter 12.

Try & catch

Bud has obtained the keys he needed from Vicente's office. They're going to be essential for executing the plan. Leaving the prison won't be easy, but returning to it without anyone noticing will be even more complicated.

It appears that one of the staff members working in the prison hospital has been stealing some medications to sell among the inmates. The idea is to blackmail him to help us get me back in. It shouldn't be too difficult; he's sure to cooperate without problems.

Once outside, when it's time to return, I'll hide in the trunk of his car and we'll simply drive into the employee parking lot. From there, accessing the library unseen is very simple. I'll have to hide until the daily library visit time and then blend in with the group.



The guard in question is called Phil and he won't help us without reason. In fact, if he learns about our escape plans, he would report us immediately.

We need evidence of his wrongdoing; once we have it, he'll have no choice but to cooperate. Bud will take care of that. He has many resources and it seems he has been closely following Phil's activities. For now, I don't have to do anything, although I've already been warned that my contribution will be crucial at the end of the escape.

We already have an approximate date for the final phase of our plan. In two weeks there will be some holidays, and security measures tend to be reduced. Not by much, but at least that gives us some advantage. With this information, I can now coordinate with Chani's cousin and tell him more or less which day I'm going to get out.

Once this range of days is specified, I hope Chani will take care of contacting Rich and arranging a meeting. I don't know how we're going to get the confession; I guess the idea is to record audio with the phone, but since I'm practically in isolation, I still don't have more details.

Yesterday I had a long conversation with Bud, and he told me what he plans to do when our paths separate. Pedro and he have been planning the escape for a long time. It's an opportunity they can't let slip away. I don't know exactly how old they are, but I estimate they're around 50. They still have a considerable amount of their sentence left and don't want to spend the rest of their lives locked up.

They have contacts outside and will leave the country the same day they get out. The plan is to be very far away once the guards realize they've escaped. I hope everything goes well.



We need to get to work and obtain that evidence that incriminates Phil. For this, we need to reach the prison hospital and, for that, I'm going to need to use some Java concepts that I haven't learned yet. So, without me saying anything, Bud has offered to give me a new lesson. He's a great teacher, so I can't refuse.

Introduction and Importance of try and catch Blocks in Java

In Java, try and catch code blocks are used to handle exceptions. Exceptions are situations that can occur while a Program is running and could interrupt the normal flow of Program execution. Therefore, to avoid this, we must take into account these possible events and control them in advance.

These blocks are absolutely necessary as they are an organized and controlled way of handling errors and exceptions during Program execution. Thanks to try and catch, we will avoid possible errors in our Program that we didn't even account for.

Most Common Exceptions

Division by Zero: By wrapping the division inside a try block, you can catch the `ArithmeticException` and display an appropriate error message.

String to Number Conversion: Use try-catch to catch `NumberFormatException` and handle invalid inputs.

Array Index Access: Protect array element access with try-catch to catch `ArrayIndexOutOfBoundsException`.

Safe Arithmetic Operations: Use `Math.subtractExact` inside a try block to catch `ArithmeticException` in case of overflow.

String Concatenation: Wrap the concatenation loop in a try block to catch `OutOfMemoryError` and handle the error.

Structure of try-catch blocks

```
- try Block
  try {
    // Code that might generate an exception
  }
```

- **catch Block**

It's placed directly after the try block and has this structure:

```
catch (ExceptionType e) {  
    // Code to handle the exception  
}
```

- **Basic example**

```
public class Main {  
    public static void main(String[] args) {  
        try {  
            int result = 10 / 0; // This throws ArithmeticException  
            System.out.println("The result is: " + result);  
        } catch (ArithmeticException e) {  
            System.out.println("Error: Cannot divide by zero.");  
        }  
    }  
}
```

Multiple catch Blocks

You can have multiple catch blocks to handle different types of exceptions. Each catch handles a specific exception.

```
try {  
    // Code that can throw different exceptions  
} catch (ArithmeticException e) {  
    // Handle ArithmeticException  
} catch (NullPointerException e) {  
    // Handle NullPointerException  
}
```

The finally Block (Optional): You can add a finally block after the catch blocks. The finally block always executes, regardless of whether an exception was thrown or not, and is useful for releasing resources such as closing files or database connections.

```
try {  
    // Code that might throw exceptions  
} catch (Exception e) {  
    // Handle any exception  
} finally {  
    // Code that always executes, with or without exception  
}
```

Summary of What We've Learned

As I mentioned at the beginning of the topic, in Java, try and catch blocks are fundamental for managing unexpected situations during Program execution, preventing interruptions in its normal development.

In simple exercises, it might not be necessary to use this structure, but when Programs become more complex, it's necessary to consider all possible events that may occur during the execution of our code. These blocks allow us to handle, among other things, errors such as division by zero, incorrect string conversions, out-of-range array accesses, and arithmetic operations that could exceed limits.

At this point, the try-catch structure shouldn't be very complicated for you. Inside the try block, we find the code that could generate an exception. If this occurs, that's when the catch block comes into action and handles the error. You can have multiple catch blocks for different types of exceptions and optionally use the finally block to execute code that must always run, such as resource release.

Practical exercises

Exercise 1. Division by zero. Write a Program that asks the user for two numbers and tries to divide the first by the second. Handle the `ArithmeticException` in case of division by zero.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the first number:");
        int num1 = scanner.nextInt();

        System.out.println("Enter the second number:");
        int num2 = scanner.nextInt();

        try {

            int result = num1 / num2;
            System.out.println("The result is: " + result);

        } catch (ArithmeticException e) {


            System.out.println("Error: Cannot divide by zero.");

        }

        scanner.close();
    }
}
```

Exercise 2. Handling `NumberFormatException`: Write a Program that asks the user for an integer number and tries to convert it. Handle the `NumberFormatException` in case the format is not valid. Once the exercise is completed, try entering a decimal number to see what happens.

Solution:



```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter an integer number:");
        String input = scanner.nextLine();

        try {

            int number = Integer.parseInt(input);
            System.out.println("The entered number is: " + number);

        } catch (NumberFormatException e) {

            System.out.println("Error: Invalid number format.");

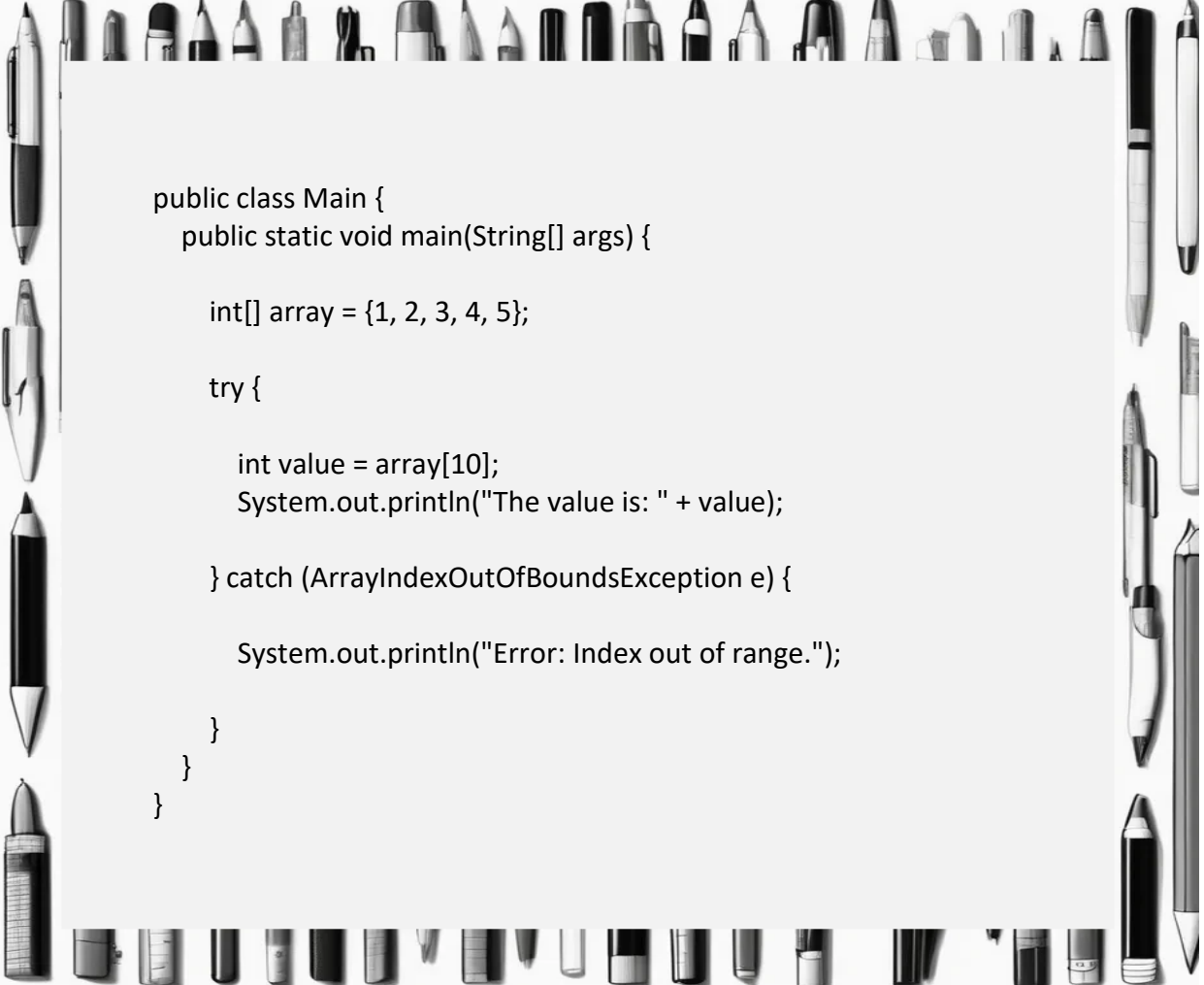
        }

        scanner.close();

    }
}
```

Exercise 3. Out of range index in an array. Write a Program that creates an array of 5 elements and tries to access a position out of range. Handle the `ArrayIndexOutOfBoundsException`.

Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        int[] array = {1, 2, 3, 4, 5};  
  
        try {  
  
            int value = array[10];  
            System.out.println("The value is: " + value);  
  
        } catch (ArrayIndexOutOfBoundsException e) {  
  
            System.out.println("Error: Index out of range.");  
  
        }  
    }  
}
```

Exercise 4. Invalid arithmetic operation. Write a Program that asks the user for two numbers and subtracts them. Handle the `ArithmeticException` if the operation is not valid for any reason (e.g., subtraction of extremely large values).

Solution:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the first number:");
        long num1 = scanner.nextLong();

        System.out.println("Enter the second number:");
        long num2 = scanner.nextLong();

        try {

            long result = Math.subtractExact(num1, num2);
            System.out.println("The result is: " + result);

        } catch (ArithmeticException e) {

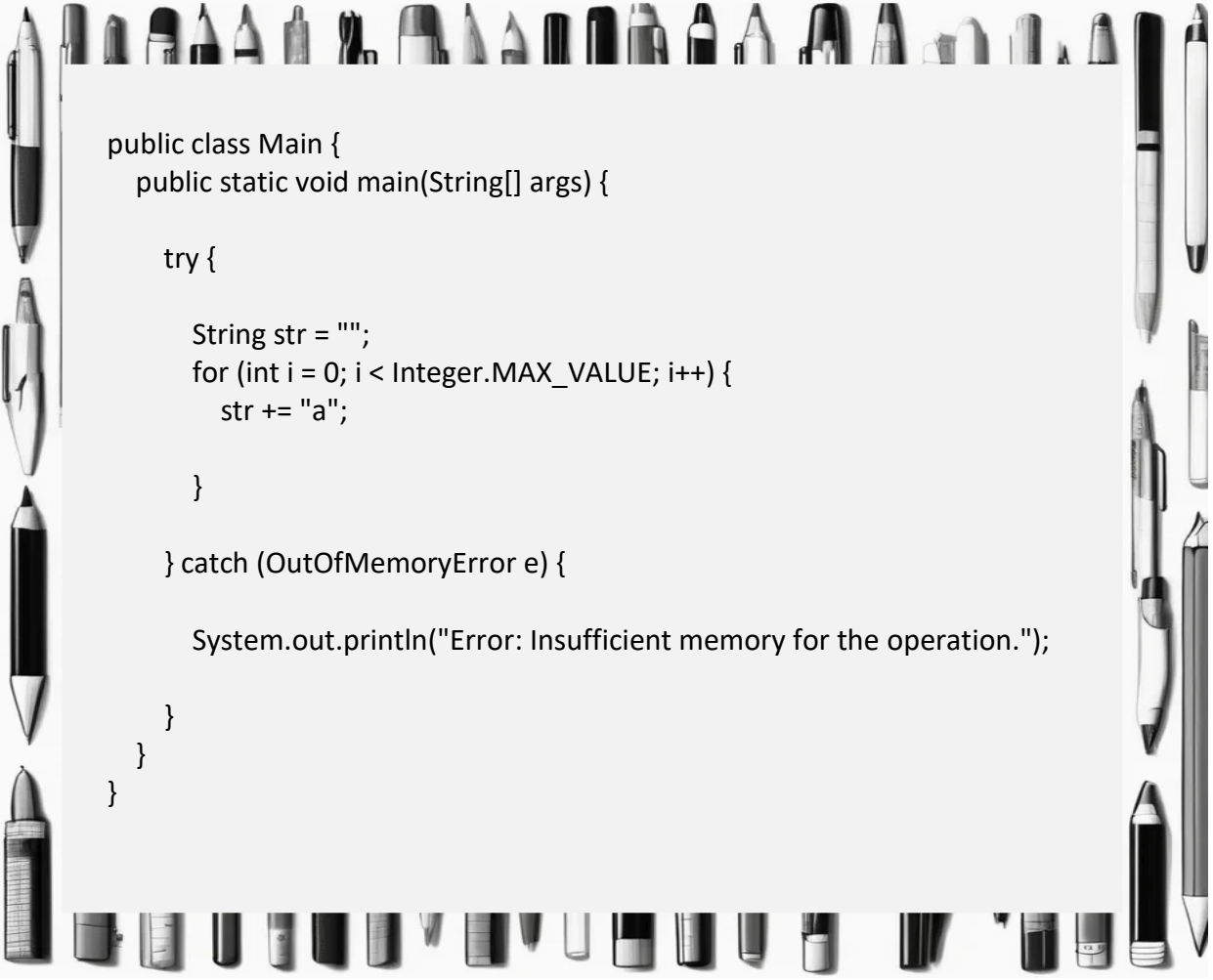
            System.out.println("Error: Invalid arithmetic operation.");

        }

        scanner.close();
    }
}
```


Exercise 5. String concatenation. Write a Program that tries to concatenate a very large number of strings, causing an `OutOfMemoryError`. Handle the exception.

Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        try {  
  
            String str = "";  
            for (int i = 0; i < Integer.MAX_VALUE; i++) {  
                str += "a";  
            }  
  
        } catch (OutOfMemoryError e) {  
  
            System.out.println("Error: Insufficient memory for the operation.");  
        }  
    }  
}
```

Chapter 13.

Gathering evidence against Phil

After the incident with the alarm and subsequent blackout, prison officials are on alert. They are beginning to suspect something, and extraordinary precautionary measures are being taken. Most of them don't affect us, but to access the prison hospital, the first thing Bud needs to do is deactivate the GPS location of his laptop. After what happened with the fire alarm, many security measures are being taken, and everyone is being monitored. It's very likely that prison officials already suspect Bud and Pedro. From now on, they need to be completely invisible.

Exercise 1. Changing GPS signal: Bud and Pedro must change the GPS signal of their computers so guards cannot track their real location. We will write a Program to help us with this task.

Instructions:

- 1) Create an array containing the real coordinates of each computer.
- 2) Allow the user to enter fake coordinates.
- 3) Verify if the entered coordinates are within an acceptable range (for example, within a 1 km radius of the real coordinates).
- 4) If the coordinates are acceptable, display a success message. If not, display an error message.

Solution:

```

import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        double[] realCoordinates = {40.7128, -74.0060}; // (example)

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the fake coordinates one by one (latitude
and longitude:");

        double fakeLatitude = scanner.nextDouble();

        double fakeLongitude = scanner.nextDouble();

        double distance = Math.sqrt(Math.pow(fakeLatitude -
realCoordinates[0], 2) + Math.pow(fakeLongitude - realCoordinates[1],
2));

        if (distance <= 0.01) { // Approximately 1 km radius
            System.out.println("Fake coordinates accepted. The guards won't
be able to track you.");

        } else {

            System.out.println("Fake coordinates out of range. Please try
again.");

        }
    }
}

```

Just after entering the hospital, there's a room with medium security level that has five cameras. Additionally, officers are always checked when entering to prevent them from bringing objects from outside. The problem is that they are not checked when leaving, which allows Phil to steal materials. In that room, Bud must avoid the cameras.

The first thing to do is to check that all cameras are turned on, since, if they are, Bud will be able to intercept and turn them off. If any camera is off beforehand, it cannot be intercepted. Just because our Program indicates that a camera is off doesn't necessarily mean it really is; it might be using another type of power source to function. That would be too big a risk. Therefore, the objective is to verify that all cameras are active and then turn them off.

Exercise 2. Intercepting Video Signals. Bud needs to intercept the video signals from the security cameras. Write a Program that simulates the interception of video signals and displays a message each time a signal is intercepted.

Instructions:

1. Declare 5 boolean variables. (In the real prison Program, we won't know if they are declared as true or false. We'll just declare them with random values to see if the Program works)
2. Use a for loop to simulate the interception of video signals from multiple cameras (for example, 5 cameras). Make it check the status of all 5 cameras.
3. Display a message indicating that each camera's signal has been intercepted. True means it's on and therefore the Program intercepts it. False means it's off and the Program won't be able to intercept it.

Solution:

```

public class Main {
    public static void main(String[] args) {

        // Boolean variable declaration
        // As mentioned in the statement, in the prison system we won't
        // know the status of the cameras. That's why we create this
        // Program.
        // We simply add values to see if the Program works.

        boolean camera1 = true;
        boolean camera2 = false;
        boolean camera3 = true;
        boolean camera4 = false;
        boolean camera5 = true;

        // For loop to check the status of each camera

        for (int i = 1; i <= 5; i++) {

            // We create a boolean status. We need to assign a value to avoid
            // an error.
            // It doesn't really matter if it's true or false. It will change
            // depending on each camera.

            boolean status = false;

            if (i == 1) {

                status = camera1;

            } else if (i == 2) {

                status = camera2;

            } else if (i == 3) {

                status = camera3;

```

```

    } else if (i == 4) {
        status = camera4;
    } else if (i == 5) {
        status = camera5;
    }

    if (status) {
        System.out.println("Camera " + i + " status: Intercepted");
    } else {
        System.out.println("Camera " + i + " status: Not intercepted");
    }
}
}
}

```

In the next room, the security level is high. They no longer use cameras, but motion sensors, specifically five of them. To reach the computer room, it's necessary to cross this room.

Exercise 3: Deactivating Motion Sensors. Now Bud must deactivate the motion sensors to move without being detected. Write a Program that allows deactivating the sensors one by one.

Instructions:

1. Use a boolean array to represent the status of the sensors (activated or deactivated).
2. Use a loop to allow the user to deactivate each sensor.
3. Show the status of the sensors after each change.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        boolean[] sensors = {true, true, true, true, true}; // All sensors are
        initially activated

        Scanner scanner = new Scanner(System.in);

        for (int i = 0; i < sensors.length; i++) {

            System.out.println("Do you want to deactivate sensor " + (i + 1) + "?"
            (true/false));
            sensors[i] = !scanner.nextBoolean();
        }

        System.out.println("Sensors status:");
        for (int i = 0; i < sensors.length; i++) {

            System.out.println("Sensor " + (i + 1) + ": " + (sensors[i] ? "Activated"
            : "Deactivated"));
        }
    }
}
```

Protecting the computer room, there's a door that requires a numeric access code. If an incorrect or non-numeric password is entered, the Program will detect it and an alert will be activated. We need to create a Program using try and catch that warns in case a non-numeric or incorrect code is entered. This is a precaution in case the code stolen from Vicente doesn't work. If that happened, the plan would be compromised and we couldn't escape, but at least we wouldn't be discovered trying to flee.

Exercise 4. Exception Handling when Hacking the Security System. Bud tries to access the security system, but can make mistakes during the process. Write a Program that simulates hacking and handles possible exceptions (such as entering a non-numeric value).

Instructions:

1. Ask the user to enter a numeric code.
2. Use try and catch to handle possible exceptions if the user enters a non-numeric value.
3. If the value is correct, display a success message. If not, display an error message.

Solution:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
```



```
try {
    System.out.println("Enter the hacking code (integer number):");
    int code = Integer.parseInt(scanner.nextLine());
    System.out.println("Correct code. Hack successful.");
} catch (NumberFormatException e) {

    System.out.println("Error: You entered a non-numeric value.
Please try again.");

}
}
```

Once inside the last room, we need to access the computer. This last password works in a peculiar way. It uses a new code each time to gain access. This makes it practically impossible to hack. To access the system, we'll need a key obtained from Vicente's office that is then randomly combined with two other numbers. Before entering it, we need to check exactly what those two numbers are.

Exercise 5. Create a Program that finds the complete 6-digit key from a 4-digit base key. The base key is combined with a variable two-digit number to form a complete key.

- Create three int variables, one for the incomplete password we have, another for the final password, and a temporary one that we'll use within our for loop to check all combinations from 00 to 99.
- Use a for loop to iterate from 0 to 99, representing all possible combinations of the last two digits.

- In each iteration of the loop, combine the base key with the current loop number multiplied by 100, adding the loop value. This generates a possible complete key (checkedKey).
- Compare the key generated in each iteration (checkedKey) with the known complete key. If a match is found, print the complete key found and end the loop.

In the Program that Bud uses to find the password, obviously he wouldn't have the final password. But we have included it in this exercise to verify that it works properly.

Solution:

```
public class Main {  
  
    public static void main(String[] args) {  
        // Known base key  
        int baseKey = 8787;  
  
        // Complete key to be found  
        int completeKey = 878793;  
  
        // This key will increase in our loop one by one  
        int checkedKey;  
  
        // Check combinations of the last two digits  
        boolean found = false;  
  
        for (int i = 0; i < 100; i++) {  
  
            checkedKey = (baseKey * 100) + i;
```

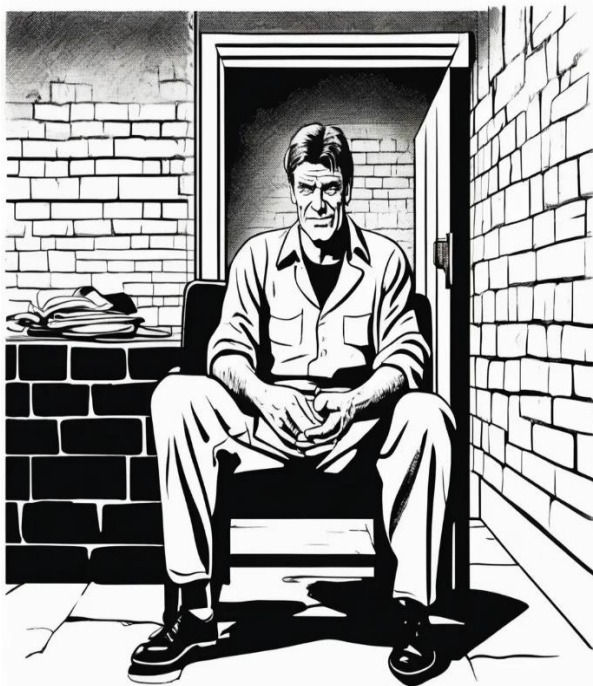
```
        System.out.println("checkedKey: " + checkedKey);
        if (checkedKey == completeKey) {
            System.out.println("This is the complete key: " + checkedKey);
            break;
        }
    }
}
```

Chapter 14.

Reading and writing text files

Now that we have the record of medicines stolen by Phil, it's time to blackmail him. Pedro will meet with him today. We have no doubt that he will cooperate. If we presented the evidence, he would not only be fired, but he would also be involved in a legal process where he would possibly be sentenced to some years in prison.

The plan is working perfectly. No one suspects anything, and we've already overcome more than half the way. Before the final step, and finally getting out, we still need two things: to have a record of the security measures that await us in the last stretch before the exit and to access the guards' communication.



I haven't told you yet because it's crazy, but the idea is to leave through the main door. It seems like suicide, but no one will suspect that someone would try to escape through there. As you can imagine, it's the most guarded part of the prison and we need to know exactly what we're up against.

To analyze the guards' activity and tasks, we have some records of the guards' activities. These are in text format and we need to create Java Programs to help us understand the information they contain and even modify it if necessary.

As usual, I have no idea what's happening outside. I hope Chani is taking care of everything and that she's in communication with Rich. I hope that when I get out, I don't find out that Rich has fled and that all this effort has been in vain.

On the other hand, I'm still in contact with my family, but I've lost some contact with my close friends. The truth is that right now I don't want to have visits. I'm focused on the escape plan. I don't want to think about what people might be saying about me. That could affect me. I'm going to prove my innocence and show everyone who doubts me that they're wrong.



It seems that my private tutor is ready to teach me how to read and edit files using Java. The truth is that it's not a topic that particularly excites me, but it's necessary to continue with this plan. Bud has already warned me that it's possibly the most complex part of the entire curriculum, so I need to prepare myself mentally. Although with such a good teacher, I'm sure it won't be difficult to understand it.

Importance of Reading and Writing Files in Java

As you're taking your first steps in Java, reading and writing files in Java is probably not a priority for you. Most Programs don't need this functionality. However, it wouldn't be smart to ignore that it's a fundamental tool for many applications. When reading a file, an application can process external data and work with it. Just as we've been giving instructions manually to our Programs so far, now they can follow these same guidelines from a text file.

Similarly, writing to files allows Programs to save results, create logs, or store important data that can later be used or analyzed.

Another important advantage is that Java offers a wide variety of classes and methods within its library to handle files safely and easily. This facilitates the manipulation of large volumes of data, whether in text or binary format. Additionally, file handling in Java is essential for developing applications that require data persistence, such as information management systems or databases.

Reading Text Files

In my opinion, when learning Java, the important thing is not to memorize concepts like a parrot, but to understand how Programming works and comprehend the logic behind it. If you do it this way, it will be much easier for you; the feeling of progress will be much more satisfying and, moreover, you'll know how to Program for the rest of your life. It's true that if you spend a long time without touching code, you might not remember almost anything, but by dedicating a few hours, that knowledge quickly returns to your head.

After so many hours Programming, you'll already have quite advanced knowledge of Java syntax. What seemed impossible to learn at the beginning now makes sense and even seems simple. Not everyone gets this far, so congratulations.

For this specific topic, and much to my regret, we'll have to learn some things by memory. For our Programs to read and write text, we have to import a series of classes.

Importing the Necessary Classes for Reading Files

```
import java.io.File;  
import java.io.FileReader;  
import java.io.BufferedReader;  
import java.io.IOException;
```

Creating a File Object that Represents the Text File

Obviously, if you have a file that you want to work with, you don't need to create it, but you do need to know the path where the file is located on your computer.

```
File file = new File("path/to/file.txt");
```

This is an example of a real file path that I use for example exercises:

```
File file = new File("C:\\Users\\ribeh\\IdeaProjects\\Project 1\\activity.txt");
```

Reading the File Using FileReader and BufferedReader

I'd like to tell you that the Program can already read the data from our file and even print it on the screen, but that's not the case. There's still one more step.

We need to use try and catch, but since we already know how they work, there won't be any problem.

```
try (FileReader fr = new FileReader("activity.");
    BufferedReader br = new BufferedReader(fr)) {

    String line;
    while ((line = br.readLine()) != null) {
        System.out.println(line); // Prints each line of the file
    }

} catch (IOException e) {
    e.printStackTrace();
}
```

We call the `FileReader` "fr" and the `BufferedReader` "br", but you can use whatever name suits you best. Then we create a while loop that goes through the lines of our file and keeps printing as long as there is data to print. The last part is the catch block in case there's any exception.

Writing Text Files

Again, we'll need to import the necessary classes. In this case, they are these:

```
import java.io.File;
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.IOException;
```


Of course, we'll need to have the path to our file. I've already explained how the path should be:

```
File file = new File("path/to/file.txt");
```

This is an example of a real file path that I use for example exercises:

```
File file = new File("C:\\Users\\ribeh\\IdeaProjects\\Project 1\\activity.txt");
```

Writing to the File Using FileWriter and BufferedWriter

Write to the file using FileWriter and BufferedWriter

```
String activity = "activity.txt"; // Define the file name
```

```
try (FileWriter fw = new FileWriter(activity);
```

```
    BufferedWriter bw = new BufferedWriter(fw)) {
```

```
    bw.write("Hello, this is a new text file.");
```

```
    bw.newLine(); // Adds a new line
```

```
    bw.write("Here we can add more content.");
```

```
} catch (IOException e) {
```

```
    e.printStackTrace();
```

```
}
```

We've used "Fw" to refer to our FileWriter and "Bw" for the BufferedWriter, but, as in the previous case, we could give them another name if it suits us better. In this case, we don't need to create a loop inside the try block because we don't need to go through all the lines. We only need to add one or several lines. Finally, logically, we close with the catch block in case a possible exception occurs.

Summary of What We've Learned

To read text files in Java, the first thing we need to do is import the `File`, `FileReader`, `BufferedReader`, and `IOException` classes. To read a file, you must choose that document and copy its path. Then, you use `FileReader` and `BufferedReader` to read the file line by line and display its contents. It's essential to handle possible exceptions with `try` and `catch` blocks so the Program can manage any errors that arise. Fortunately, we learned how to do this in the previous topic.


Let's look at the other side of the coin: writing to those text files. To start, you also need to import some classes, which in this case are: `File`, `FileWriter`, `BufferedWriter`, and `IOException`. Just like before, we'll need the path to our file. Then, we'll use `FileWriter` and `BufferedWriter` to write content to the file, adding new lines when necessary. It's crucial to handle exceptions in this process to ensure that any problems during writing are handled appropriately.

These steps allow you to effectively read and write text files in Java, managing errors and exceptions in a controlled manner.

Practical exercises

Exercise 1. Write a Program that saves the phrase 'Hello World' in a file called output.txt.

Solution:

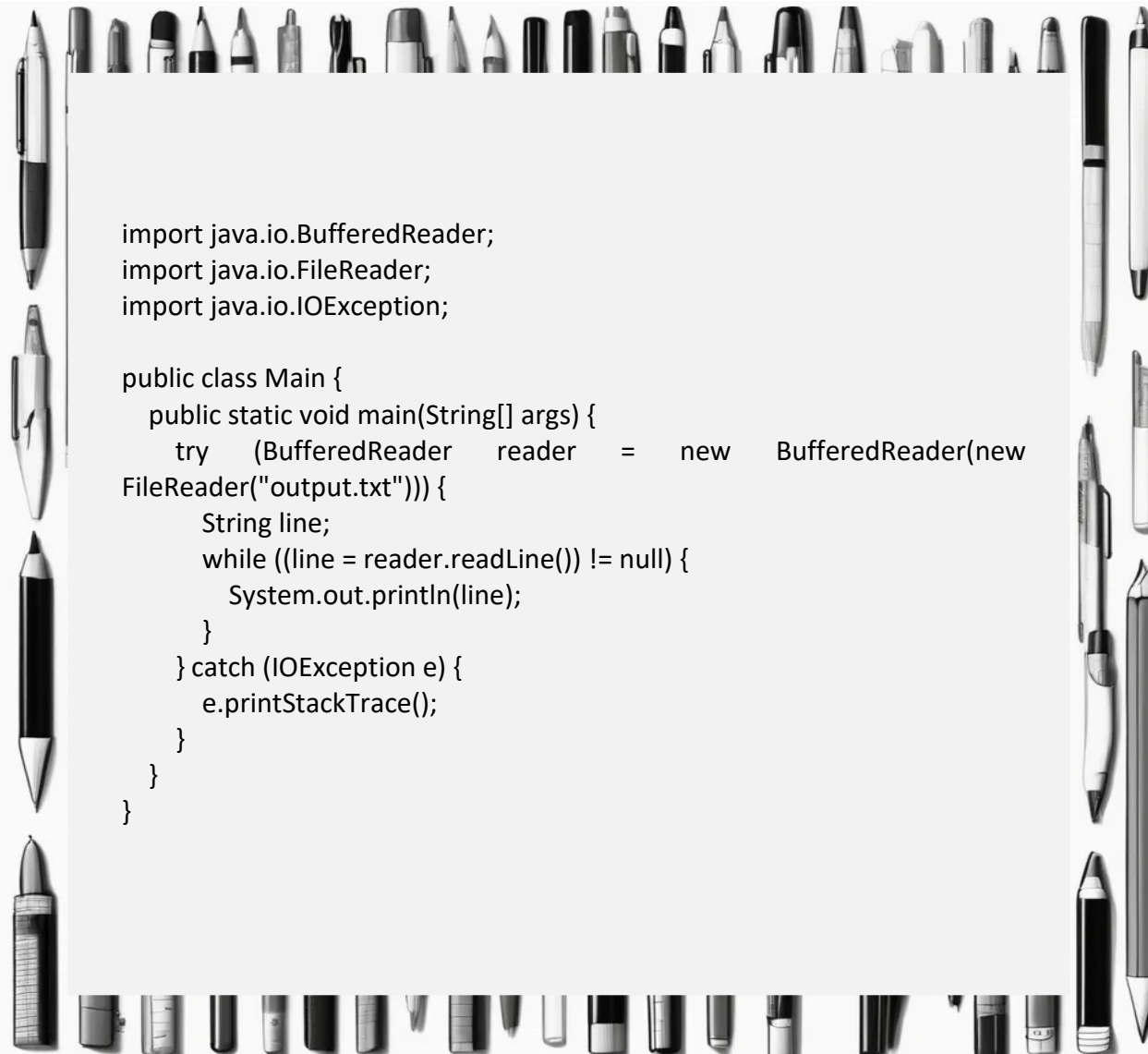


```
import java.io.FileWriter;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        try (FileWriter writer = new FileWriter("output.txt")) {
            writer.write("Hello World");
            System.out.println("Text written to output.txt");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Exercise 2. Write a Program that reads the content of the file called output.txt and displays it in the console.

Solution:

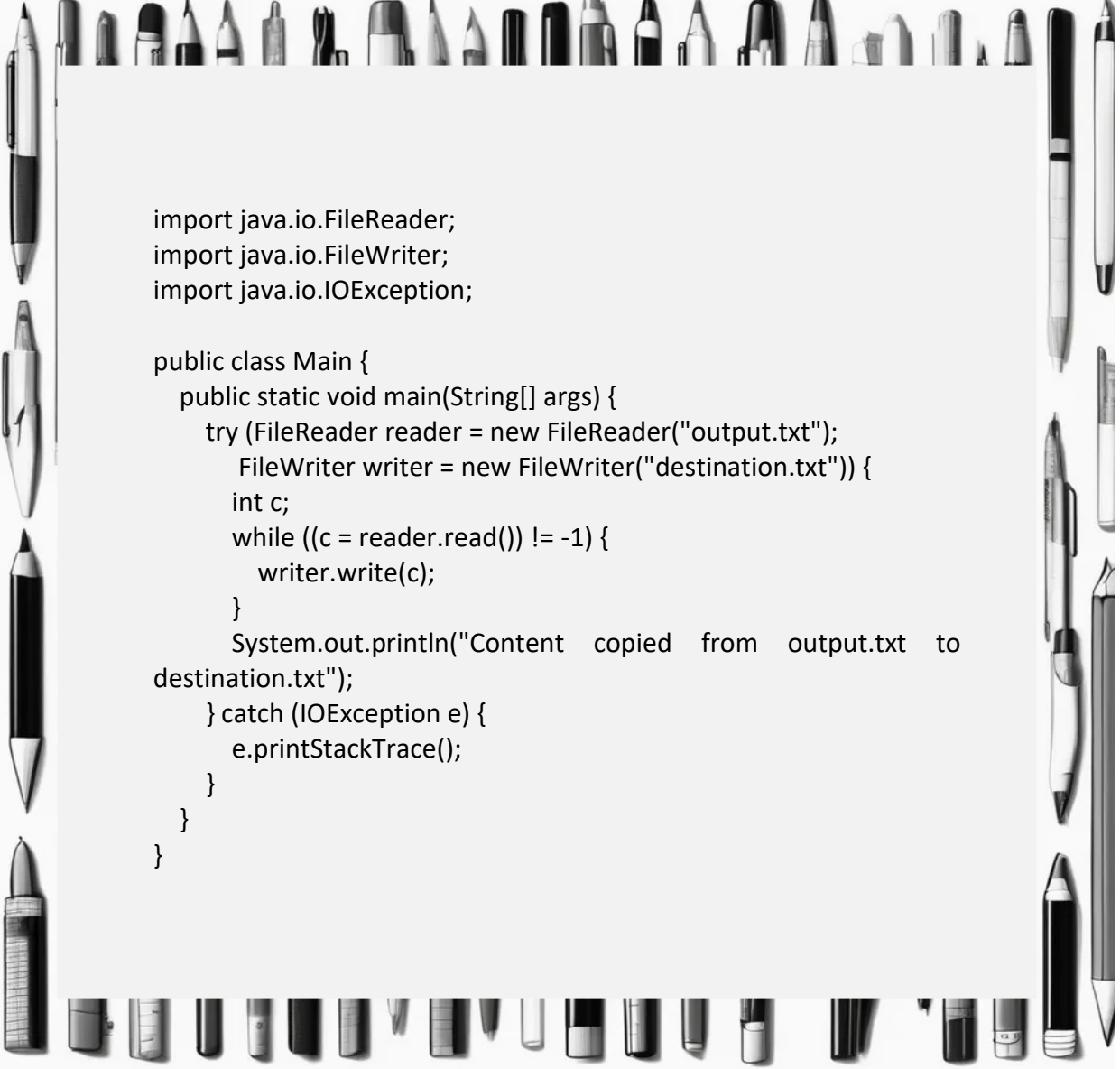


```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        try (BufferedReader reader = new BufferedReader(new
        FileReader("output.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Exercise 3. Write a Program that copies the content from output.txt to destination.txt

Solution:



```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        try (FileReader reader = new FileReader("output.txt");
            FileWriter writer = new FileWriter("destination.txt")) {
            int c;
            while ((c = reader.read()) != -1) {
                writer.write(c);
            }
            System.out.println("Content copied from output.txt to
destination.txt");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Exercise 4. Write a Program that counts the number of words in the file called destination.txt

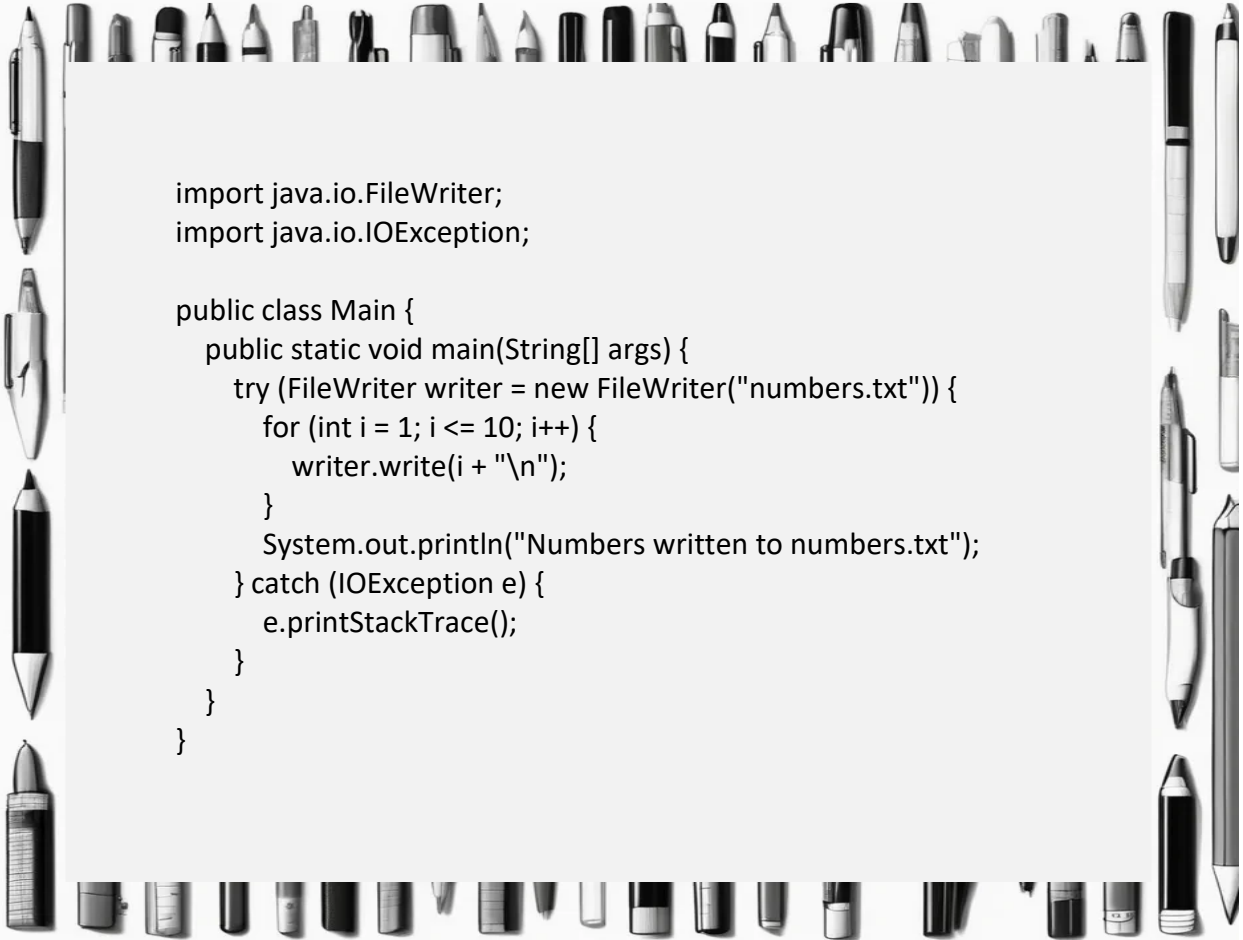
Solution:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        int wordCount = 0;
        try (BufferedReader reader = new BufferedReader(new
FileReader("destination.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] words = line.split("\\s+");
                wordCount += words.length;
            }
            System.out.println("Number of words in destination.txt: " +
wordCount);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Exercise 5. Write a Program that saves the numbers from 1 to 10 in a file called numbers.txt, with each number on a new line.

Solution:

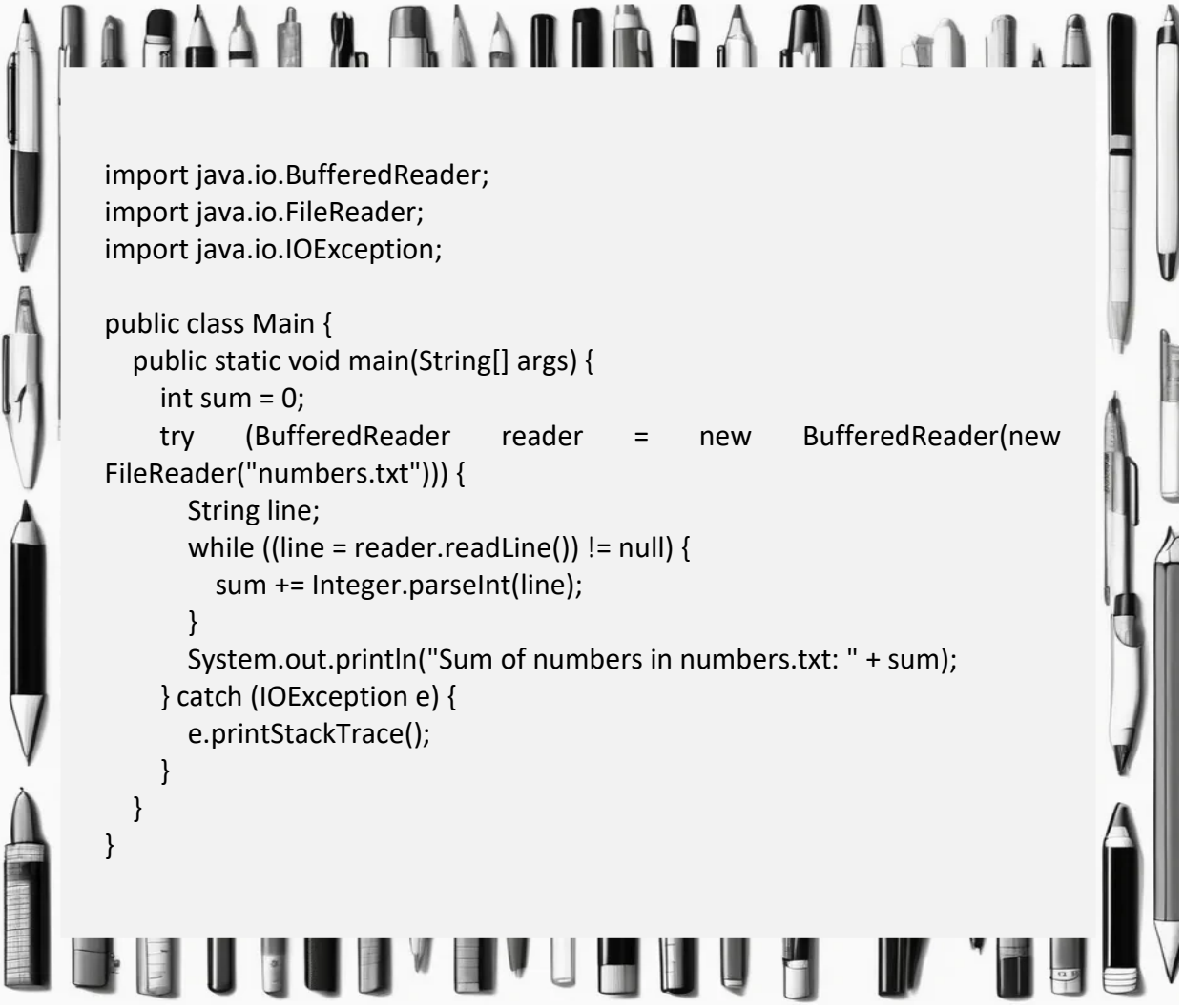


```
import java.io.FileWriter;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        try (FileWriter writer = new FileWriter("numbers.txt")) {
            for (int i = 1; i <= 10; i++) {
                writer.write(i + "\n");
            }
            System.out.println("Numbers written to numbers.txt");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Exercise 6. Write a Program that reads integer numbers from a file called numbers.txt and calculates the sum of these numbers.

Solution:

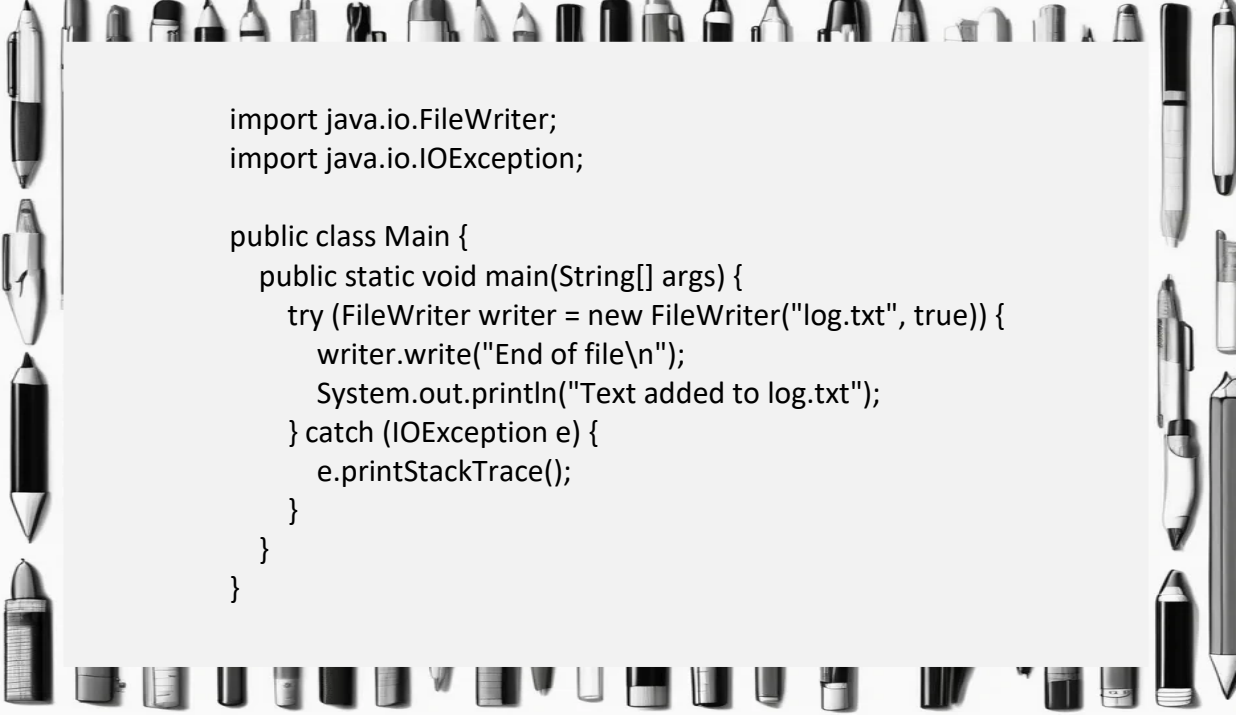


```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        int sum = 0;
        try (BufferedReader reader = new BufferedReader(new
        FileReader("numbers.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                sum += Integer.parseInt(line);
            }
            System.out.println("Sum of numbers in numbers.txt: " + sum);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```


Exercise 7. Write a Program that adds the phrase 'End of file' to the end of a file called log.txt.

Solution:

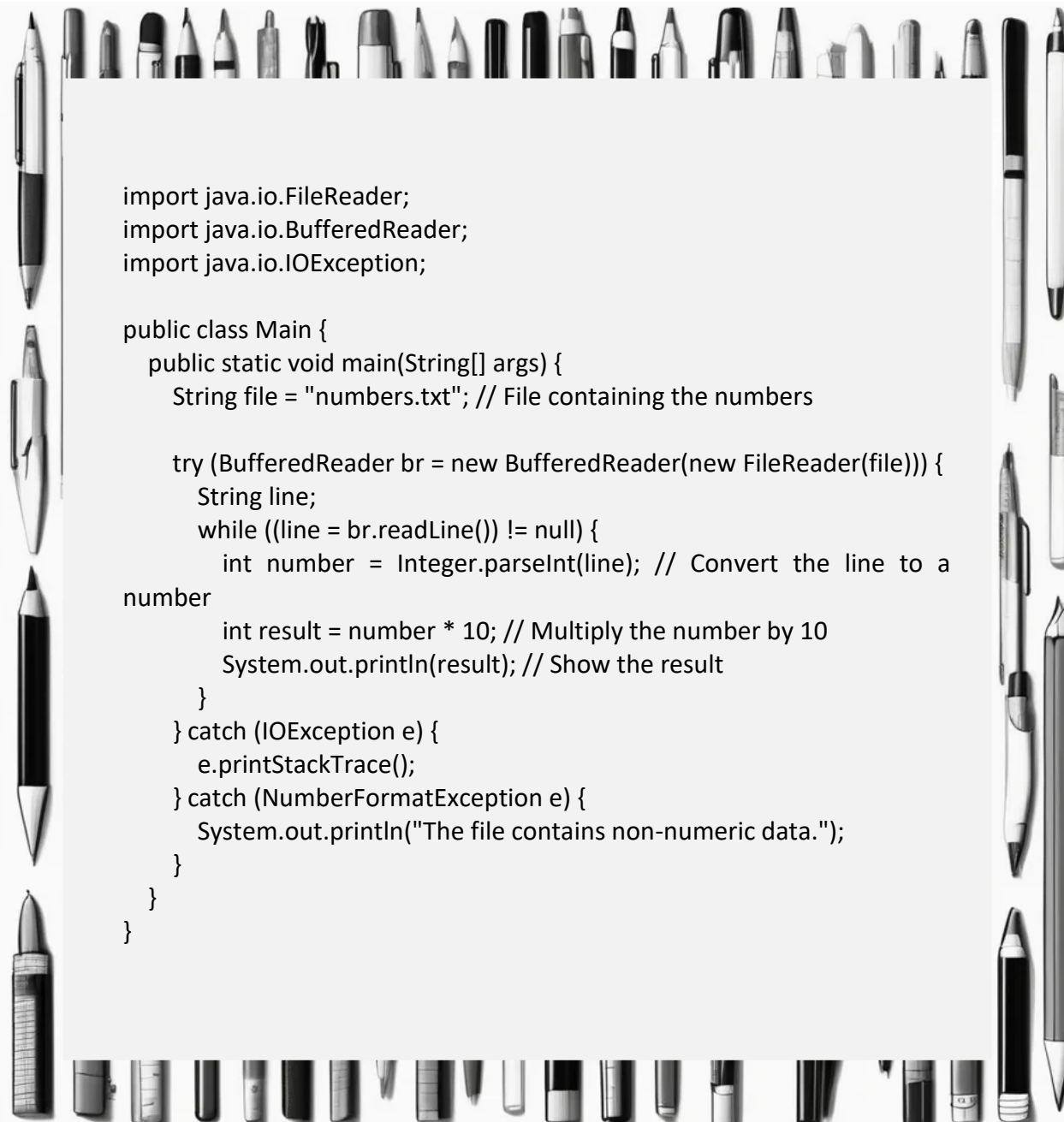


```
import java.io.FileWriter;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        try (FileWriter writer = new FileWriter("log.txt", true)) {
            writer.write("End of file\n");
            System.out.println("Text added to log.txt");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Exercise 8. Write a Program that reads the content of the numbers.txt file and displays each value multiplied by 10 in the console.

Solution:



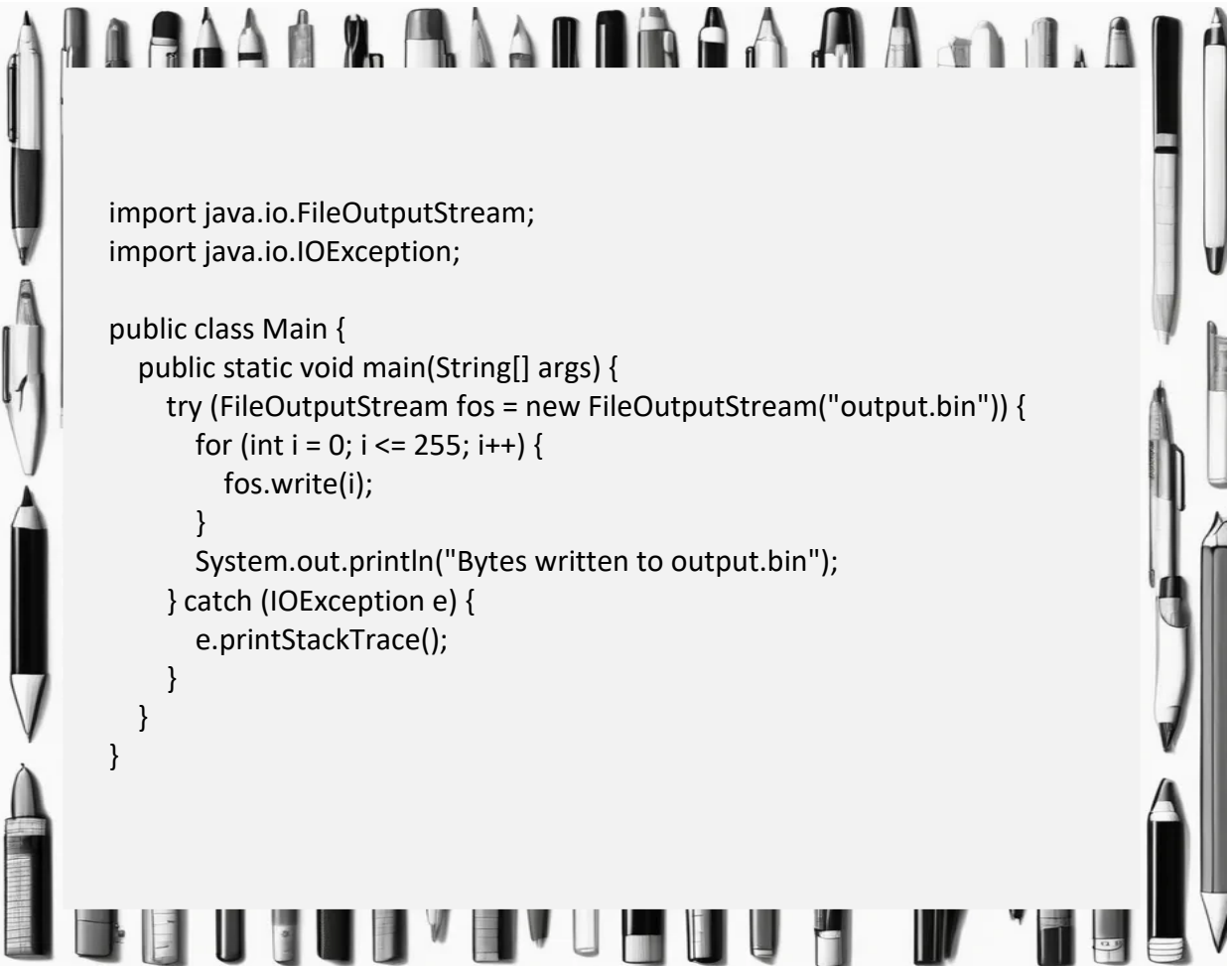
```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        String file = "numbers.txt"; // File containing the numbers

        try (BufferedReader br = new BufferedReader(new FileReader(file))) {
            String line;
            while ((line = br.readLine()) != null) {
                int number = Integer.parseInt(line); // Convert the line to a
number
                int result = number * 10; // Multiply the number by 10
                System.out.println(result); // Show the result
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (NumberFormatException e) {
            System.out.println("The file contains non-numeric data.");
        }
    }
}
```

Exercise 9. Write a Program that writes bytes from 0 to 255 in a file called output.bin.

Solution:

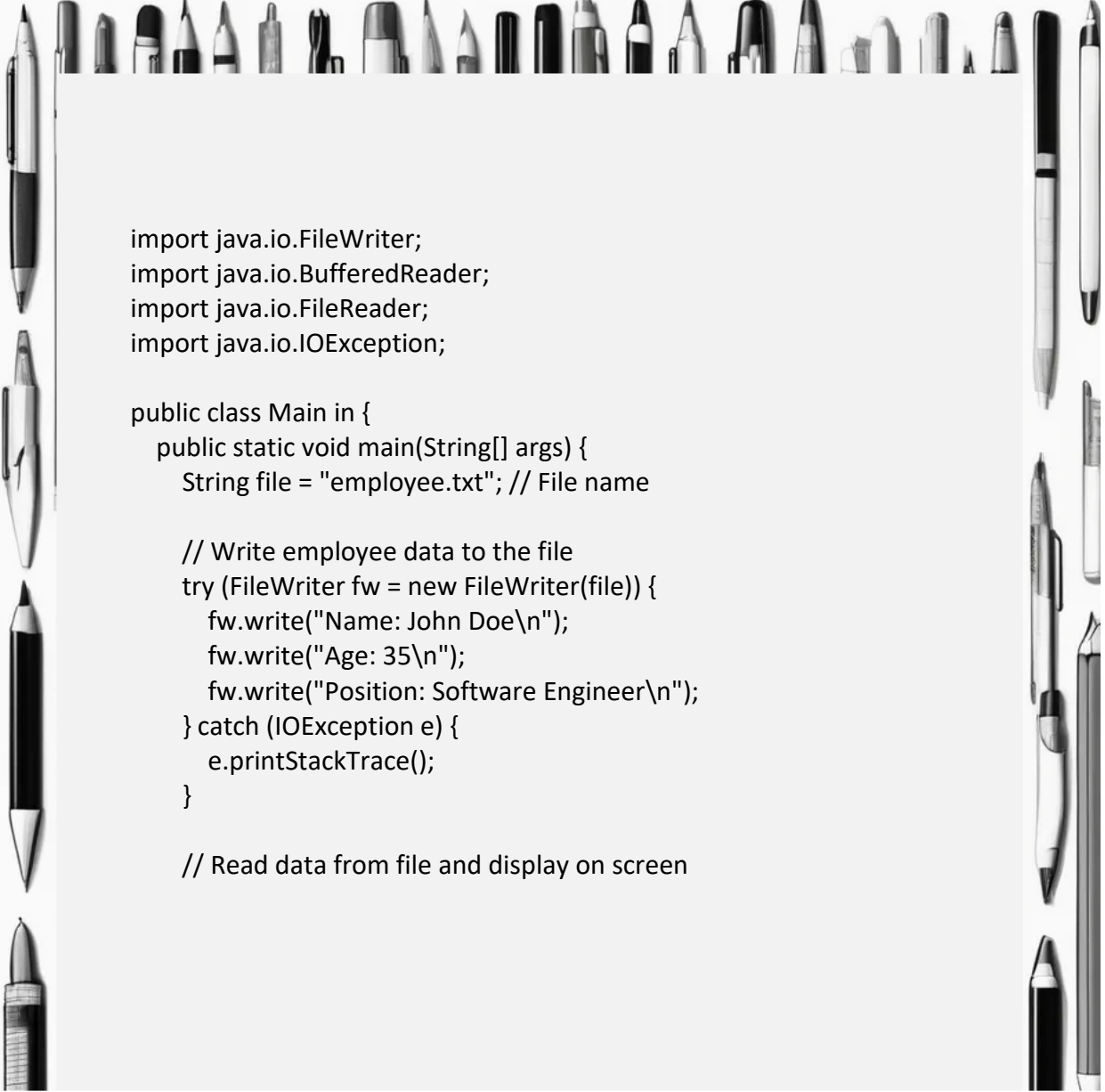


```
import java.io.FileOutputStream;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        try (FileOutputStream fos = new FileOutputStream("output.bin")) {
            for (int i = 0; i <= 255; i++) {
                fos.write(i);
            }
            System.out.println("Bytes written to output.bin");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Exercise 10. Write a Program that writes an employee's data (Name, Age, and Position) to a file. Then the Program will read this data and display it on the screen.

Solution:



```
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Main in {
    public static void main(String[] args) {
        String file = "employee.txt"; // File name

        // Write employee data to the file
        try (FileWriter fw = new FileWriter(file)) {
            fw.write("Name: John Doe\n");
            fw.write("Age: 35\n");
            fw.write("Position: Software Engineer\n");
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Read data from file and display on screen
    }
}
```

```
try (BufferedReader br = new BufferedReader(new
FileReader(file))) {
    String line;
    while ((line = br.readLine()) != null) {
        System.out.println(line); // Display each line on screen
    }
} catch (IOException e) {
    e.printStackTrace();
}
}
```

Chapter 15.

Bypassing the Time Counter System

The hallway leading to the prison exit is the most heavily guarded. You might think it's because of fear that prisoners might escape through there, but that's not the case. What happens is that everyone enters and exits through the same place. Both visitors and officers and prisoners always follow the same path.

Every time someone enters the prison, they are searched and the images are recorded. Additionally, you must pass through three registration points, divided into various rooms. At each stop, there is a guard who doesn't leave their post until relief arrives.

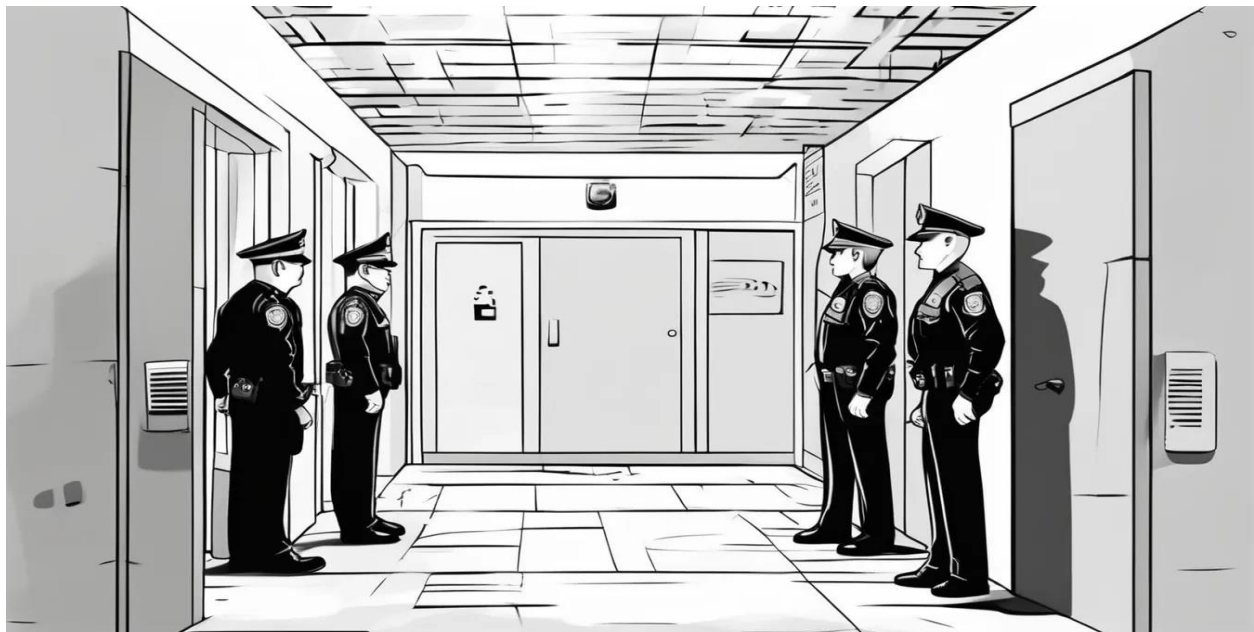


At the end of the hallway, there are two doors. One leads to the cells and the other to a waiting room where, in turn, we find other doors. From there you can access other parts of the prison, but they don't interest us since we don't need to set foot in them.

Knowing that there are at least five guards guarding the exit at all times, it seems unwise to try to use that route. However, not everything is as it seems, since during the night there are no visits and, after the shift change at 22:00, no one else enters. That means they don't need to have five people guarding the entrance.

Bud believes that, at most, there will be two people. It's even possible that at some point they might need to perform some task during their shift and there might be a short period of time when we could be alone.

The plan is to identify, measure, and analyze the jailers' activities to know exactly what time we should attempt the escape. To do this, the first step will be to analyze when doors are opened and closed. That way, we can identify when there is movement. We'll create a Program that calculates the inactivity time of the door leading to the waiting room. We can visually control the door leading to the cells, and we don't need to worry about the street door, since no one uses it during the night.



We have a small advantage: with one of the keys obtained from Vicente's office, Pedro has managed to access the seven door activity logs from last week, one for each day. With this, we can already get an idea of which day is the quietest and the hours when the guards enter and exit. The first step will be to determine which day we choose. For this, we need to create a Program that reads the periods of door activity and adds them up. We will choose the escape day that has the least total activity time.

The file divides the day into minutes, and each line tells us how many seconds the door was active in that minute. To give you an idea, it has this format:

```
0
0
0
0
60
60
12
0
0
```

Exercise 1. Recording Activity Time. We need to record the doors' activity time to choose a day for our escape. For this, we need to write a Program that reads the inactivity time from a file and records the total inactivity time in another file.

Instructions:

1. We use the activity.txt file obtained by Pedro, with activity times in seconds (one per line).
2. Read the activity times from the file.
3. Calculate the total inactivity time.
4. Write the total inactivity time to a file total_activity.txt

Solution:


```
import java.io.*;
import java.util.*;
public class Main {
    public static void main(String[] args) {

        String inputFile = "activity.txt";

        String outputFile = "total_activity.txt";

        try (BufferedReader br = new BufferedReader(new FileReader(inputFile));
            BufferedWriter bw = new BufferedWriter(new FileWriter(outputFile)))
        {

            String line;
            int totalActivity = 0;

            while ((line = br.readLine()) != null) {
                totalActivity += Integer.parseInt(line);
            }

            bw.write("Total activity time: " + totalActivity + " seconds");

        } catch (IOException e) {

            e.printStackTrace();

        }
    }
}
```

Monday is the day with the least activity, so we already know when we should attempt the escape. According to the record, from minute 1322 to 1438, that is, from 22:02 until 23:58, the door is inactive. Then, it becomes active again at approximately 2:00 and subsequently every hour until 6:00, when the main door opens and fills with police. On each occasion, the doors are active and open for less than two minutes.

Once we know at what times the doors open, we need to determine if they were opened to let someone in or perhaps for someone to exit. The officers must perform a series of tasks during their guard duty. Pedro has gained access to the record of these night tasks and the times that guards mark them as completed. We need to create a small Program that records these activities and returns them in a text format. We know that their main activities are: regular patrolling, inspection and maintenance, and cleaning and disinfection.

The log file looks like this:

```
"iRKJi2KpMW", "ixpFHX0v8s", "rMxNXbeUxC", "bjO9IUFGc", "LpTB2ffO8w", "S4WwZffjqB",  
"HuuSb3RH6I", "O8OmH1ApJx", "LpFKbFT7hU", "TZ9BmcroA7", "dyLWJCvbU6",  
"qNIWgK2wzD", "s6vEOgmfsY", "ccDtnysyec", "waITKPMWbY", "HH9N0Ufih4", "cGlet9IE4y",  
"MCwCJ6LNQP", "OfecGXWhqY", "SOd3Tco6ml", "r8QwQDbHyp", "4V6bJhJ20x",  
"nzkiWhmxVj", "OtT2nDTToVa", "67NEMfhs2x", "W3HXrDI7SJ", "KgGoiFJfMN",  
"2WK7UZVmgX", "3KNx0fDu54", "Rn20v6baaz", "t8jChEgFSz", "489dXWEFi6", "oNsSEfNwaP",  
"V4zx2r6XR8", "2dPXcKbZnQ", "WpsxQvEbVw", "7f6floF3VU", "W1joPNEiK1",  
"pXpGUMRsZq", "1M1jDBpiGu", "xjZBVieX1Q", "y3dVBuhTrJ", "s0YB6CVSFV", "vZQgDge2HJ",  
"EsWhUsnhLa", "hevhsUt0ux", "bYwU4Qy6GI", "sCW32VXSfD", "mwCdwQ809e",  
"zyRwiYkfDK", "ko4o26MERp", "Lk1rspAJK", "d5Y9IPuMk5", "yKj992kdn", "Q1XsuVqUGr",  
"fHMDmXiAcp", "dql0nmQJcm", "qRqSHARwSA", "CgV8ne6fRG", "30DYrnKWoa",  
"r1QrdR0es2", "HXTV8XvYoP", "GQAWwyhMSL", "4yMmh7e5NW", "yT2kIJvDqN",  
"AblqpbKj5h", "haAriF42gR", "O6MciwEwQ4", "hW8Pf9IkX7", "kUaN7Cnv96", "tARoxTCuiP",  
"aZEZynwwpj", "cqzR5Y3Ipi", "M69Qr3BWJj", "ySTegpfDxw", "f47AKb3Bsn", "XOsmCGvcSo",  
"wA7iC3Fq1k", "NqTC2OpWGr", "lvvkjHR0Q2", "JyT5lu4QPo", "2whhr0ixDg", "xcYX1ELvDi",  
"zTS8EJE0hh", "7gYXpaaPIs", "UiZ8tMlvfl", "uuSBzI6IfZ", "BdcQ3ci06N", "VNrQV6uZvU",  
"gZILASO2gT", "iOIRRFxhc", "jkSGZFtlpw", "2gkQKh6J8L", "jnTyMxcG8I", "X07Mj6M0Rj",  
"SqjfH3cyLO", "GfKiA3DZ4w", "RBIquXxt0k", "udPC3Q0Pxo", "CM44vkuwyT", "20n5Uwv6s8",  
"5ST85ixStb", "uliTvpFukE", "ErQPLyDuRw", "XX5eaAXr1a", "nLenQQgqoP", "ThjBwzns36",  
"8RghH8Vneq", "FWFaHu0c2a", "HqGai14Ao6", "nFEi87Bafj", "rotkzVIY3k", "Tah5BZU49N",  
"YILUBnOTTh", "kyhPp7TWZp", "q1kiVAMsHv", "OvKuo3mvSX", "htDbhBg7vs",  
"O8ZBRQB5co", "TIUDfrzdUi", "7yIIIvtzSI", "HkAkB7qO3c", "j4DvSZL3Ry", "ONmRHuWdGc",  
"a7LmXNTfap", "MIHEWHZhrz", "ADKHiZeBAS", "0KQ6aXlsZY", "OKMfZsps6e", "vJpOxdfzEd",
```

```
"BKs9UTKTQJ", "DBNrFCM5d7", "hse8XLeIZK", "3jz3bLreNc", "ZZSHQ35FpT", "jOmaC6iMOW",
"iXvhHmq2pn", "gqnFV11and", "DDfLkv3uBg", "UQjsb1ixF0", "08zSNrhvxs", "LSWyFeFyYv",
"Regular patrol", "4Bpd1idsum", "vCgKs8GFTI", "uhUlpi1VLd", "AQn63M3ANF", "Y39NIKxlGJ",
"YcpNNmT3GX", "C8Di70GqS7", "Ybg4P3BOij", "6Tb4opfNeC", "aTmzLU9opz",
"QSHg6ZRmyH", "IVUz2kHAOn", "Inspection and maintenance", "wUxaxSm9lC",
"DVTb4JEZQ9", "sjahqvnVsk", "TfvPongU7w", "gQ6g0rJ7ax", "bejbKLD6ZR", "SMj8MjDEWY",
"fanVKMvsmZ", "Regular patrol", "gmqX2oBxvm", "Mb9B4SHLvM", "64rh8hh6Li",
"SjalZaHsOA", "yxDLOhr4Od", "qfN1GbZNFA", "C4q5ridJ7I", "aQLm7SJeEo", "LNDWA1UHzG",
"QpYvFFSY6S", "I7ySXRZkQH", "WwtkFmM7pP", "gl794m6lyl", "6fvGhjprUx", "SVINByrKBN",
"DNEmWPvI51", "Mm6NxxNkdF", "gyLw8TP8kH", "vcqjXfWTYz", "bpuMR5EVFB",
"sMMp5HpxJg", "9hU32sjepp", "RsYnv80Hx2", "SilPvzI3gD", "L7pspbXgSw", "RvAtxqkYF2",
"9gOPHDbw31", "uUDPIKRjU3", "NkPKW7zkZK", "dBOiqar26k", "GXPsNzuULq",
"KwwpvpPqQ2", "Tif0qfs2eq", "3xTc242fPW", "c5Jz3DVRv", "zrK0VW0AQH", "zPJt2IGxKU",
"C3ssWKqU5s", "Xsfl0o5asg", "QiR4pDd4N0", "q4iY9xGn3V", "msQISGxfMJ", "yRyD1ec8fb",
"dBfcrywOUD", "O9cMzNXF5t", "INuV5TSkbo", "YEgbviK7AV", "HKrrLQBtt2", "ZU0hqtXuNa"
```

Note that if no activity is entered for a certain time, a random code is generated.

Exercise 2. Generate an Activity Log. Bud needs to generate an activity log to analyze the behavior of the jailers. Write a Program that records the different system activities in a file.

Instructions:

1. Create an array with different activities ("Regular patrol", "Inspection and maintenance", "Cleaning and disinfection").

Write these activities to a log_activities.txt file, one per line.

Solution:

```

import java.io.*;
public class Main {
    public static void main(String[] args) {

        String[] activities = {"iRKJi2KpMW", "ixpFHX0v8s", "rMxNXbeUxC", "bjO9IUsgFc",
"LpTB2ffO8w", "S4WwZffjqB", "HuuSb3RH6l", "O8OmH1ApJx", "LpFKbFT7hU",
"TZ9BmcroA7", "dyLWJCvbU6", "qNIWgK2wzD", "s6vEOgmfsY", "ccDtnysyec",
"waITKpMwBY", "HH9N0Ufih4", "cGlet9IE4y", "MCwCJ6LNQP", "OfecGXWhqY",
"SOd3Tco6ml", "r8QwQDbHyp", "4V6bJhJ20x", "nzkiWhmxVj", "OtT2nDToVa",
"67NEMfhs2x", "W3HXrDI7SJ", "KgGoiFJfMN", "2WK7UZVmgX", "3KNx0fDu54",
"Rn20v6baaz", "t8jChEgFSz", "489dXWEFi6", "oNSsEfNwaP", "V4zx2r6XR8",
"2dPXcKbZNQ", "WpsxQvEbVw", "7f6floF3VU", "W1joPNEiK1", "pXpGUMRsZq",
"1M1jDBpiGu", "xjZBVieX1Q", "y3dVBuhTrJ", "s0YB6CVSFV", "vZQgDge2HJ",
"EsWhUsnLa", "hevhsUt0ux", "bYwU4Qy6GI", "sCW32VXSfD", "mwCdwQ809e",
"zyRwiYkfDK", "ko4o26MERp", "Lk1rspAJQK", "d5Y9IPuMk5", "yKj992kdnn",
"Q1XsuVqUGr", "fHMDmXiAcp", "dql0nmQJcm", "qRqSHARwSA", "CgV8ne6fRG",
"30DYrnKWoa", "r1QrdR0es2", "HXTV8XvYoP", "GQAWwyhMSL", "4yMmh7e5NW",
"yT2kIjvDqN", "AblqpbKj5h", "haAriF42gR", "O6MciwEwQ4", "hW8Pf9IkX7",
"kUaN7Cnv96", "tAROxtCUiP", "aZEYnwwpj", "cqzR5Y3Ipi", "M69Qr3BWJj",
"ySTegpfDxw", "f47AKb3Bsn", "XOsmCGvcSo", "wA7iC3Fq1k", "NqTC2OpWGr",
"lvvkjHR0Q2", "JyT5lu4QPo", "2whhr0ixDg", "xcYX1ELvDi", "zTS8EJE0hh", "7gYXpaaPls",
"UiZ8tMlvfi", "uuSBZl6lFz", "BdcQ3ci06N", "VNrQV6uZvU", "gZILASO2gT", "iOIRRKfxhc",
"jkSGZftlpw", "2gkQKh6J8L", "jnTyMxcG8l", "X07Mj6M0Rj", "SqjfH3cyLO",
"GfKiA3DZ4w", "RBIquXXt0k", "udPC3Q0Pxo", "CM44vkuwyT", "20n5Uwv6s8",
"5ST85ixStb", "uliTvpFuke", "ErQPLyDuRw", "XX5eaAXr1a", "nLenQQgqoP",
"ThjBwzns36", "8RghH8Vneq", "FWFaHu0c2a", "HqGai14Ao6", "nFEi87Bafj",
"rotkzVIY3k", "Tah5BZU49N", "YILUBnOTTh", "kyhPp7TWZp", "q1kiVAMsHv",
"OvKuo3mvSX", "htDbhBg7vs", "O8ZBRQB5cO", "TIUDfrzdUi", "7yIltvZSI",
"HkAkB7qO3c", "j4DvSZL3Ry", "ONmRHuWdGc", "a7LmXNTfap", "MIHEWHZhrz",
"ADKHiZeBAS", "OKQ6aXlsZY", "OKMfZsps6e", "vJpOxdfzEd", "BKs9UTKTQJ",
"DBNrFCM5d7", "hse8XLelZK", "3jz3bLreNc", "ZZSHQ35FpT", "jOmaC6iMOW",
"iXvhHmq2pn", "gqnfV11and", "DDfLkv3uBg", "UQjsb1ixF0", "08zSNrhvxs",
"LSWyFeFyYv", "Regular patrol", "4Bpd1idsum", "vCgKs8GFTI", "uhUIpi1VLd",
"AQn63M3ANF", "Y39NIKxlgJ", "YcpNNmT3GX", "C8Di70GqS7", "Ybg4P3BOij",
"6Tb4opfNeC", "aTmzLU9opz", "QSHg6ZRmyH", "IVUz2kHAOn", "Inspection and
maintenance", "wUxaxSm9IC", "DVTb4JEZQ9", "sjahqvnVsk", "TfvPongU7w",
"gQ6g0rJ7ax", "bejbKLd6ZR", "SMj8MjDEWY", "fanVKMvsmZ", "Regular patrol",

```

```

"gmqX2oBxvm", "Mb9B4SHLvM", "64rh8hh6Li", "SjalZaHsOA", "yxDLOhr4Od",
"qfN1GbZNFA", "C4q5ridJ7I", "aQLm7SJeEo", "LNDWA1UHzG", "QpYvFfSY6S",
"l7ySXRZkQH", "WwtkFmM7pP", "gl794m6lyl", "6fvgHjprUx", "SVINByrKBN",
"DNEmWPvI51", "Mm6NxxNkdF", "gyLw8TP8kH", "vcqjXfWTYz", "bpuMR5EVFB",
"sMMP5HpxJg", "9hU32sjepp", "RsYnv80Hx2", "SilPvzI3gD", "L7pspbXgSw",
"RvAtxqkYF2", "9gOPHDbw31", "uUDPIKRjU3", "NkPKW7zkZK", "dBOiqar26k",
"GXPsnZuULq", "KwwpvvPqQ2", "Tif0qfs2eq", "3xTc242fPW", "c5Jz3DVrV",
"zrK0VW0AQH", "zPJt2IGxKU", "C3ssWKqU5s", "Xsfl0o5asg", "QiR4pDd4N0",
"q4iY9xGn3V", "msQISGxfMJ", "yRyD1ec8fb", "dBfcrywOUD", "O9cMzNXF5t",
"lNuV5TSkbO", "YEgbviK7AV", "HKrrLQBtt2", "ZU0hqtXuNa"};

```

```

String outputFile = "activities_log.txt";

try (BufferedWriter bw = new BufferedWriter(new FileWriter(outputFile))) {

    for (String activity : activities) {
        bw.write(activity);
        bw.newLine();
    }

} catch (IOException e) {

    e.printStackTrace();

}
}

```

Not all tasks are performed every day. Our plan is to escape on a Monday, so we'll create a Program that shows us how many times each task is performed per day. We need to find one that isn't performed at all.

Each task is always carried out at the same time, regardless of the day. For example, inspection and maintenance are always done from 2:00 to 3:00. Therefore, if on Monday this activity wasn't performed at all, it would mean that during that period, the guards would have nothing to do and possibly be resting. We know at what time, in theory, each of the tasks is performed. Now we just need to verify if there's any task that isn't performed on Monday.

Exercise 3. We need to analyze the activity log to identify repetitive patterns during the night. If any of them do not repeat during the night we plan to escape, it means that the hallway might be empty.

Instructions:

1. Read the log_activities.txt file.
2. Count how many times each activity occurs.
3. Write the results to a summary_activities.txt file.

Solution:

```
import java.io.*;
import java.util.*;

public class Main {
    public static void main(String[] args) {

        String inputFile = "activities_log.txt";
        String outputFile = "activities_summary.txt";

        Map<String, Integer> activityCounter = new HashMap<>();

        try (BufferedReader br = new BufferedReader(new
        FileReader(inputFile));
            BufferedWriter bw = new BufferedWriter(new
        FileWriter(outputFile))) {
```

```

        String line;
        while ((line = br.readLine()) != null) {
            activityCounter.put(line, activityCounter.getDefault(line, 0)
+ 1);
        }

        for (Map.Entry<String, Integer> entry :
activityCounter.entrySet()) {
            bw.write(entry.getKey() + ": " + entry.getValue());
            bw.newLine();
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

We have reviewed the log and the results are better than we expected. According to the analysis, the 'Cleaning and disinfection' activity is not performed on Mondays. This task is carried out between 3 and 4 in the morning. Now we know that during that hour the hallway remains inactive, without any type of activity.

With so much time available, we won't have to worry about anything. It will be a piece of cake. However, we must be completely sure that there will be no surprises or unforeseen events.

Bud and Pedro should be satisfied, but they are too serious. Given this good news, I feel optimistic. We are just one step away from escaping. However, I think Bud is suspicious. Something is escaping us, and we will need external help to discover it.

Fortunately, we know a guard who can't refuse to collaborate. Phil has informed us that two more activities have been recently implemented: 'Camera monitoring' and 'Communication with control post'. We need to update the log to verify if, during the time period we thought was free, the guards are actually in the hallway performing either of these two tasks.

Exercise 4. Update Activity Log. We need to update the activity log in real time to reflect changes in the system. Write a Program that adds new activities to the activity log file.

The new log we have found is as follows:

```
String[] activities = {"iRKJi2KpMW", "ixpFHx0v8s", "rMxNXbeUxC", "bjO9IUFGc",  
"LpTB2ffO8w", "S4WwZffqB", "HuuSb3RH6I", "O8OmH1ApJx", "LpFKbFT7hU", "TZ9BmcroA7",  
"dyLWJCvbU6", "qNIWgK2wzD", "s6vEOgmfsY", "ccDtnysyec", "waITKpMwBY",  
"HH9N0Ufih4", "cGlet9IE4y", "MCwCJ6LNQP", "OfecGXWhqY", "SOd3Tco6ml",  
"r8QwQDbHyp", "4V6bJhJ20x", "nzkiWhmxVj", "OtT2nDToVa", "67NEMfhs2x", "W3HXrDI7SJ",  
"KgGoiFJfMN", "2WK7UZVmgX", "3KNx0fDu54", "Rn20v6baaz", "t8jChEgFSz", "489dXWEFi6",  
"oNSsEfNwaP", "V4zx2r6XR8", "2dPXcKbZnQ", "WpsxQvEbVw", "7f6floF3VU", "W1joPNEiK1",  
"pXpGUMRsZq", "1M1jDBpiGu", "xjZBVieX1Q", "y3dVBuhTrJ", "s0YB6CVSFV", "vZQgDge2HJ",  
"EsWhUsnhLa", "hevhsUt0ux", "bYwU4Qy6GI", "sCW32VXSfD", "mwCdwQ809e",  
"zyRwiYkfDk", "ko4o26MErP", "Lk1rspAJQK", "d5Y9IPuMk5", "yKj992kdnn", "Q1XsuVqUGr",  
"fHMDmXiAcp", "dql0nmQJcm", "qRqSHARwSA", "CgV8ne6fRG", "30DYrnKWoa",  
"r1QrdR0es2", "HXTV8XvYoP", "GQAWwyhMSL", "4yMmh7e5NW", "yT2kIjvDqN",  
"AblqpbKj5h", "haAriF42gR", "O6MciwEwQ4", "hW8Pf9IkX7", "kUaN7Cnv96", "tAROxtCuiP",  
"aZEZynwwpj", "cqzR5Y3Ipi", "M69Qr3BWJj", "ySTegpfDxw", "f47AKb3Bsn", "XOsmCGvcSo",  
"wA7iC3Fq1k", "NqTC2OpWGr", "lvvkjHR0Q2", "JyT5lu4QPo", "2whhr0ixDg", "xcYX1ELvDi",  
"zTS8EJE0hh", "7gYXpaaPls", "UiZ8tMlvfl", "uuSBz16lfZ", "BdcQ3ci06N", "VNrQV6uZvU",  
"gZILASO2gT", "iOIRRFxhc", "jkSGZFtlpw", "2gkQKh6J8L", "jnTyMxcG8I", "X07Mj6M0Rj",  
"SqjfH3cyLO", "GfKiA3DZ4w", "RBIquXxt0k", "udPC3Q0PXo", "CM44vkuwyT", "20n5Uwv6s8",  
"5ST85ixStb", "uliTvpFukE", "ErQPLyDuRw", "XX5eaAXr1a", "nLenQQgqoP", "ThjBwzns36",  
"8RghH8Vneq", "FWFaHu0c2a", "HqGai14Ao6", "nFEi87Bafj", "rotkzVIY3k", "Tah5BZU49N",  
"YILUBnOTTh", "kyhPp7TWZp", "q1kiVAMsHv", "OvKuo3mvSX", "htDbhBg7vs",  
"O8ZBRQB5co", "TIUDfrzdUi", "7yIltvZSI", "HkAkB7qO3c", "j4DvSZL3Ry", "ONmRHuWdGc",  
"a7LmXNTfap", "MIHEWHZhrz", "ADKHiZeBAS", "OKQ6aXlsZY", "OKMfZsps6e", "vJpOxdfzEd",  
"BKs9UTKTQJ", "DBNrFCM5d7", "hse8XLeIzK", "3jz3bLreNc", "ZZSHQ35FpT", "jOmaC6iMOW",  
"iXvhHmq2pn", "gqnfV11and", "DDfLkv3uBg", "UQjsb1ixF0", "08zSNrhvxs", "LSWyFeFyYv",  
"Regular patrol", "4Bpd1idsum", "vCgKs8GFTI", "uhUlp1VLd", "AQn63M3ANF", "Y39NlKxlgJ",  
"YcpNNmT3GX", "C8Di70GqS7", "Ybg4P3BOij", "6Tb4opfNeC", "aTmzLU9opz",  
"QSHg6ZRmyH", "IVUz2kHAOn", "Inspection and maintenance", "wUxaxSm9IC",  
"DVTb4JEZQ9", "sjahqvnVsk", "TfvPongU7w", "gQ6g0rJ7ax", "bejbKLd6ZR", "SMj8MjDEWY",
```



```
"fanVKMvsmZ", "Regular patrol", "gmqX2oBxvm", "Mb9B4SHLvM", "64rh8hh6Li",
"SjalZaHsOA", "yxDLOhr4Od", "qfN1GbZNFA", "C4q5ridJ7I", "aQLm7SJeEo", "LNDWA1UHzG",
"QpYvFfSY6S", "I7ySXRZkQH", "WwtkFmM7pP", "gl794m6lyl", "6fvGHjprUx", "SVINByrKBN",
"DNEmWPvI51", "Mm6NxxNkdF", "gyLw8TP8kH", "vcqjXfWYz", "bpuMR5EVFB",
"sMMP5HpxJg", "9hU32sjepp", "RsYnv80Hx2", "SilPvzl3gD", "L7pspbXgSw", "RvAtxqkYF2",
"9gOPHDbw31", "uUDPIKRjU3", "NkPKW7zkZK", "dBOiqar26k", "GXPsNzuULq",
"KwwpvvPqQ2", "Tif0qfs2eq", "3xTc242fPW", "c5jJz3DVrV", "zrK0VW0AQH", "zPJt2IGxKU",
"C3ssWKqU5s", "Xsfl0o5asg", "QiR4pDd4N0", "q4iY9xGn3V", "msQISGxfMJ", "yRyD1ec8fb",
"dBfcrywOUD", "O9cMzNXF5t", "INuV5TSkbo", "YEgbviK7AV", "HKrrLQBtt2", "ZU0hqtXuNa"};
```

Instructions:

Use an array to simulate new activities being added to the system: "Camera monitoring", "Communication with control post"

1. Use an array to simulate new activities being added to the system, "Camera monitoring", "Communication with control post".
2. Add these activities to the end of the activities_log.txt file.
- 3.

Solution:

```
import java.io.*;

public class Main {
    public static void main(String[] args) {

        String[] nuevasActividades = {"6UX5w1vjQ", "KAB1GksspV", "ziuAVNowpG",
"uG0BEclHuq", "OxydgyaK7", "IkSjUcqTbg", "7ZzjJYTCID", "7oGiZxDCIP",
"9BRePDOokd", "SLR5Qql1gm", "dzHGx84mQH", "tTP235qyws", "Fk7qbJXHUN",
"b82VQ4j1R5", "B2I4iZ096c", "yl6ntrP2cz", "y4rC1OQMjg", "Wljc2dUrde",
"OYukZtwS1L", "qkwGHmjPao", "HycTVJeLB4", "ZaUckOVqbT", "qODG1FAhVE",
"erLTxdRe61", "qZ1NgPNFt9", "bUifZB8w4s", "Camera monitoring", "zCrzN1ZDUg",
```

```

String[] newActivities = {"gzCzIGm23u", "cDFm0hkrFw", "IDIOVBjizw",
    "jOxs8ry2fx", "GuscO6H64v", "DJrhWpPTqr", "zM3MWg5rSj", "DMYzD7vkHF",
    "BNghDoM55o", "yr89K38y7l", "H2qEcM6eIW", "iw6OVfFJDh", "yj16xbky6M",
    "IFcJ6bQs54", "CNSJhoARCw", "jOxs8ry2fx", "INZj1DoLqA", "GSEGuaChrR",
    "jrE68aipWX", "h8doJ65vcr", "2EoBD02CKL", "C1av3xn5eG", "R8w7lp6uKQ",
    "sr1E5Nf6cz", "6C5Eff5czg", "WHVPxvyciZ", "KIO6H5Vduk", "x6NIpTkJBK",
    "3YvHL9eQQ0", "qYYtIPWjpv", "UZKlgaZ4TD", "msiexEsxj9", "RzoCEriGyZ",
    "vLuY3gwLJO", "d45CH4k7cs", "rGdyVlst2Q", "3XRVIfo9mc", "eWUICZ8UOo",
    "Camera monitoring", "ic7SdX8di8", "C4qAoBxA3D", "Communication with
    control post", "VBxKrQngfS", "fICz9PFnDX", "dk2ncCGc8x", "QG1MBYgT9i",
    "HAJkKvCVwT", "gYoMT0TLOu", "vaNsJoXLnd", "TpBnt4leav", "TduiTd43zP",
    "Qm2zhvHZXB", "n7BOCacWYA", "VY4N9FrFPB", "TACDi2dNQY",
    "VW4PHE8ZDb", "SDYMTOCNJc", "Jnt9v9dgKy", "pY0yYILavE", "G9ERpaXMNI",
    "qDkqFSu4Sr", "O3VyiOh6E", "5qB9sTCmKH", "g5MyR0IXeu", "egzksA1lg2",
    "h6uLmm8tbp", "Fx7ay5mNkD", "neDWml2spW", "lwKPB704pi", "7WqwiNEBrv",
    "9ed11mNwh7", "kf3qNbMBkS", "MYFOZ3hHC8", "t4I0RK0M8O", "tkvxwGzdjg",
    "8YsZ8xVw0m", "EdFd9zOWF5", "Te6y3Oa5XP", "9sg2XIQxbQ", "kJhouzOXkS",
    "128laUqDWt", "nJDSEaneBg", "2rGSBCuzTd", "a5GbJeDTDh", "WVvQGSEkjl",
    "Q26giaMbeM", "ArdXbDImAh", "LI3nyJ5PRS", "WjZL4th5V5", "fTtNnPheOS",
    "73rwDzESwL", "8eXQ7OW9gv", "SMgcsFvILx", "wRKnDvc55v", "UWC1QxcVZc"};
    String outputFile = "activities_log.txt";

    try (BufferedWriter bw = new BufferedWriter(new FileWriter(outputFile,
true))) {

        for (String activity : newActivities) {
            bw.write(activity);
            bw.newLine();
        }

    } catch (IOException e) {

        e.printStackTrace();

    }

}

```

Now we will use again the Program created in exercise three to check if any of the new activities are performed on Mondays.

The news is not good. It seems that, at the time we thought the hallway was empty, there are actually two guards performing monitoring tasks and communication with the control post.

Our hope of escaping was set on Monday. We haven't checked the other days, since, according to door activity, there is even more guard movement. We only have one Solution left: we must figure out how to make "Camera monitoring" and "Communication with control post" not happen next Monday.

Pedro's idea is to change the system's task list and eliminate these two new activities. The problem is that they are already Programmed and can only be modified if the director or deputy director decide to rotate their order. It's not something they do every week, far from it; in fact, sometimes it takes years for them to do it.

However, if the system enters maintenance mode, files can be modified, and we can eliminate the tasks we don't want to be performed on Monday. A good reason for the system to enter maintenance is to misalign the door activity log; besides, it's something very easy to accomplish.

Remember that this file records minute by minute how many seconds the door was active. What we'll do is assign one more second of activity to each minute. In minutes where the door is active for 60 seconds, it will show 61 seconds. This will generate an error, as the system will detect that a minute can only have a maximum of 60 seconds of activity.

Exercise 5. Manipulate Door Activity Times. Write a Program that reads activity times and assigns them from a file and modifies them. Instructions:

1. Create a record_times.txt file with registration times in milliseconds (one per line).
2. Read the registration times from the file.
3. Increase each time by a fixed amount (1 second).
4. Write the modified times to a modified_times.txt file.

Solution:

```
import java.io.*;
public class Main {

    public static void main(String[] args) {

        String inputFile = "activity.txt ";
        String outputFile = "modified_times.txt";
        int increment = 1;

        try (BufferedReader br = new BufferedReader(new FileReader(inputFile));
            BufferedWriter bw = new BufferedWriter(new FileWriter(outputFile))) {

            String line;
            while ((line = br.readLine()) != null) {

                long originalTime = Long.parseLong(line);
                long modifiedTime = originalTime + increment;
                bw.write(Long.toString(modifiedTime));
                bw.newLine();

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}
```

As expected, the Program quickly detected the error and proceeded to identify and repair it. It simply restarted the system, and the records returned to normal.

We achieved our goal: during that maintenance and error-finding period, a time window opened that allowed modification of all files. Pedro took advantage of that moment to change the list of Programmed tasks for Monday to the list we needed, the one that leaves the hallway empty for an hour.

Chapter 16.

Final Preparations / For-each Loops

There's nothing left, we escape next Monday. Today is Tuesday, so we have enough time to prepare everything. Tomorrow Chani's cousin will come and I'll give him all the information. On the other hand, I need to explain to him exactly where we'll meet. I also need to tell Phil, since he'll be the one picking me up to bring me back to prison without anyone noticing.

I've had plenty of time to choose the location and, without a doubt, I think it's the perfect place. Once we escape, we'll go to a house owned by Pedro's friend. That's where we'll separate, since everyone has their own plan. Near that house there's a small public library that almost nobody uses. It's possible to reserve rooms, so Chani can even hide some microphones in advance to record the conversation.



There's a small parking lot nearby where there are always free spaces. That's where I'll meet Phil. It's located in a somewhat marginal neighborhood, so it's very unlikely there are cameras. Even so, no one will be looking for me, since everyone will assume I'm still in prison. Nevertheless, I like to be cautious, something I've learned from Bud and Pedro.

It's important that my voice doesn't appear in the recording that will incriminate Rich. Otherwise, the authorities would find out that I left prison and I would be in trouble. Chani could delete the parts where I appear, but an edited audio would lose credibility. Therefore, once I enter the library room, I must remain silent. For it not to seem strange, it's vital that I greet Rich outside the library. I'll be able to have a completely natural conversation and then, once inside the room, let the two of them talk.

I have everything well tied up, but I hope there aren't many unforeseen events. The truth is that I'm worried about not being able to control what happens outside the prison. Here it's simpler and, besides, I have help. Outside I'll be alone; I'll have Chani and Phil's support, but it's not the same. I don't trust them as much as I trust my fellow inmates.



Tonight I'll continue with my Java classes. I'm sure I'll get a good grade on my next university exam. It seems we're going to cover a type of loop that we didn't see when we addressed the topic earlier. The truth is that I quite liked the loop exercises, so I imagine this topic will also be entertaining and easy to understand.

Advantages of for-each Loops

Several topics ago, we covered the concept of loops and worked in depth with for and while loops. Now we're going to focus on another type, which I intentionally left aside at the time. Even focusing on just two, it's a somewhat dense concept and a complicated topic. But your Programming level is much higher now so you can handle this and more.

For-each loops are a fundamental tool due to their simplicity and efficiency when iterating over collections and arrays. The main advantages of using this type of loops are:

- **Simplicity and better readability:** The syntax is clearer and more concise than traditional loops (for or while). This makes the code easier to understand.
- **Error reduction:** By eliminating the need to handle indices manually, it reduces the risk of errors such as index overflow (accessing a position outside the array range).
- **Convenience:** It's especially useful for iterating over collections of objects, like lists and sets, where indices are not as relevant.

But not everything is advantageous, otherwise we would only use this type of loops, so let's look at the disadvantages and situations where it's better not to use them.

- **Limitation when working with indices:** You can't directly access the current element's index. If you need the index, you must use a traditional for loop.
- **Can't modify the collection:** If you need to add or remove elements from the collection while iterating, a for-each loop is not suitable.
- **Forward iteration only:** The for-each loop only allows forward iteration, so you can't traverse the collection in reverse.
- **Limitations in multidimensional Arrays:** This is basically because in these cases it's somewhat more complicated and less intuitive to use for-each compared to traditional loops.

For-each Loop Syntax

The basic syntax of the for-each loop in Java is:

```
for (ElementType temporaryVariable : array) {  
    // Code that uses temporaryVariable  
}
```

The syntax is simple: first we see the data type and then the temporary variable. Basically, this will take a different value each time the loop makes an iteration. Finally, we put the array that will be traversed. Let's see it with an example with real data:

```
public class Main {  
    public static void main(String[] args) {  
        int[] numbers = {1, 2, 3, 4, 5};  
  
        for (int number : numbers) {  
            System.out.println(number);  
        }  
    }  
}
```

In this example:

- `int` is the type of elements in the "numbers" array.
- `number` is the temporary variable that takes the value of each element in "numbers".
- `numbers` is the array being iterated.

Now let's see an example with a text array:

```
public class Main {  
    public static void main(String[] args) {  
        String[] fruits = {"Apple", "Orange", "Banana"};  
  
        for (String fruit : fruits) {  
            System.out.println(fruit);  
        }  
    }  
}
```

In this example:

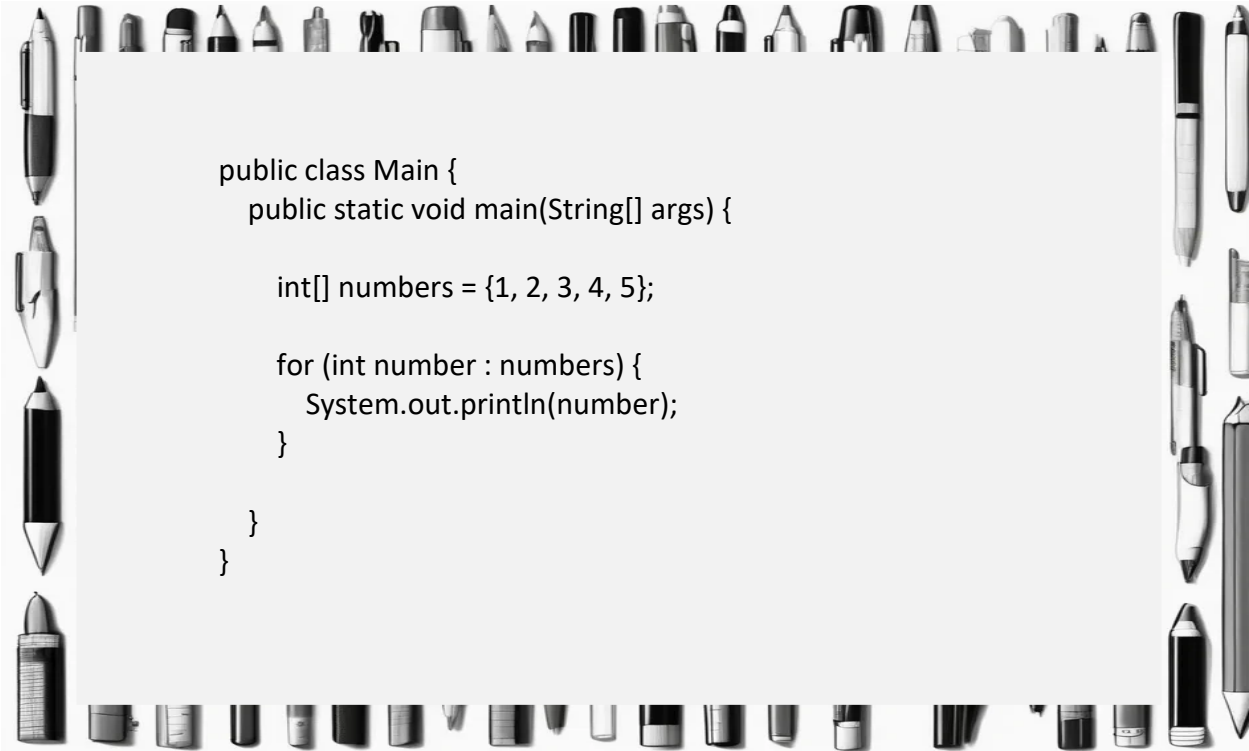
- String is the type of elements in the fruits array.
- fruit is the temporary variable that takes the value of each element in fruits.
- fruits is the array being iterated.

Since this topic has been very short and a continuation of the loops topic, we won't make a summary of it. Doing so would simply be repeating the same thing again. As you have observed, the syntax is very simple. Although to make it stick in your head, and gain confidence, I recommend that you do all the exercises I propose below.

Practical exercises

Exercise 1. Traverse an array of integers and print each number on a new line.

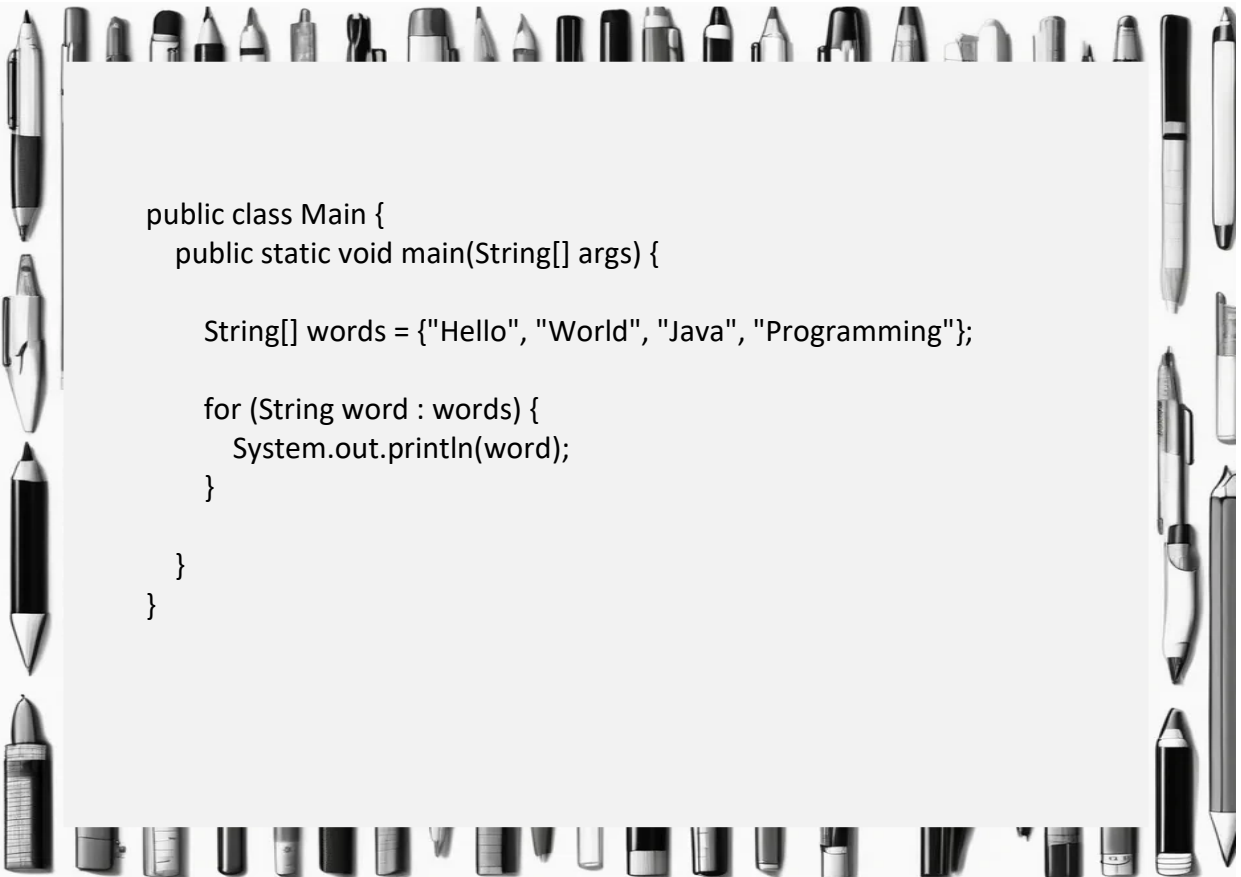
Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        int[] numbers = {1, 2, 3, 4, 5};  
  
        for (int number : numbers) {  
            System.out.println(number);  
        }  
    }  
}
```

Exercise 2. Traverse a String array and print each string on a new line.

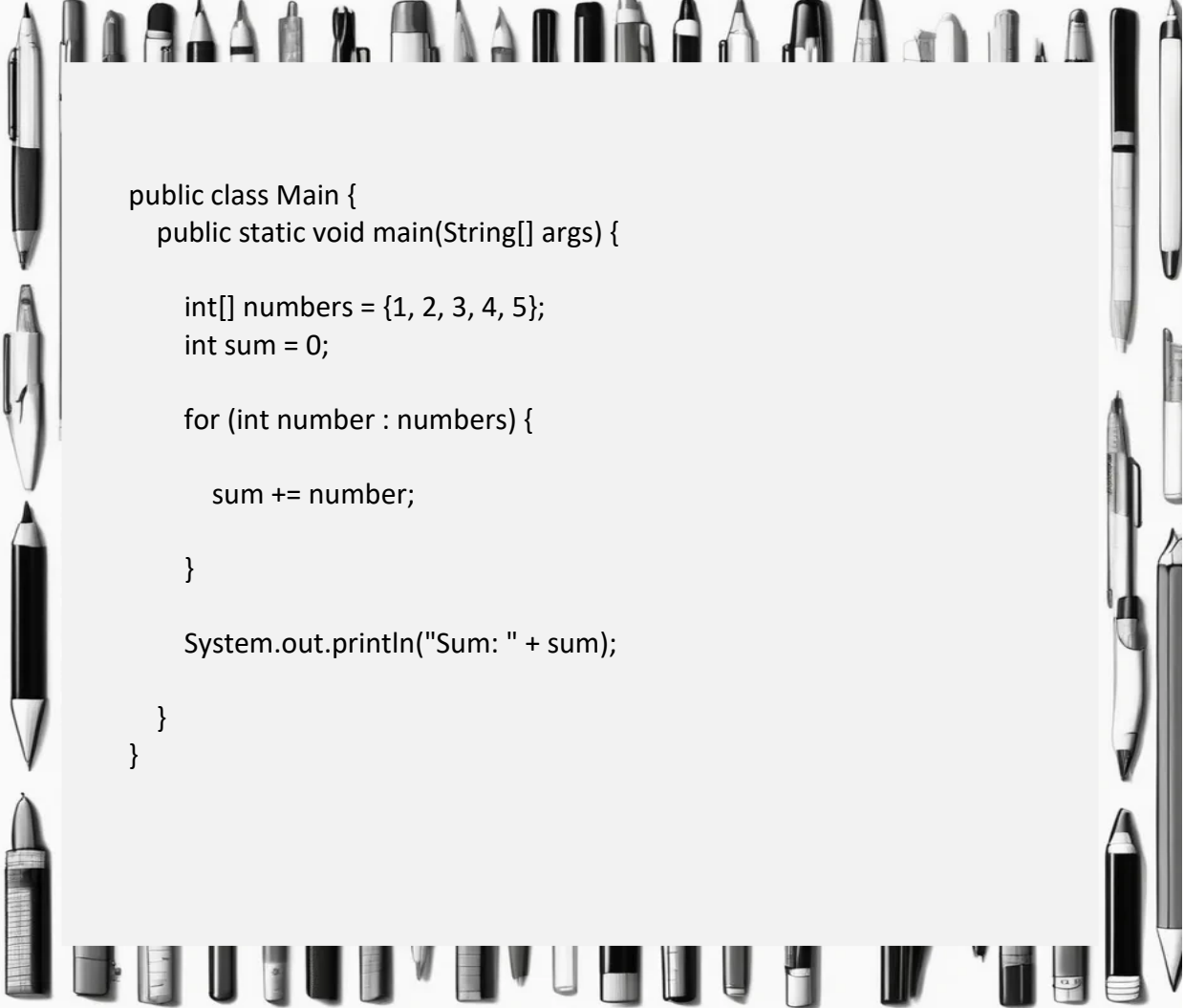
Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        String[] words = {"Hello", "World", "Java", "Programming"};  
  
        for (String word : words) {  
            System.out.println(word);  
        }  
    }  
}
```

Exercise 3. Add all elements of an integer array and print the result.

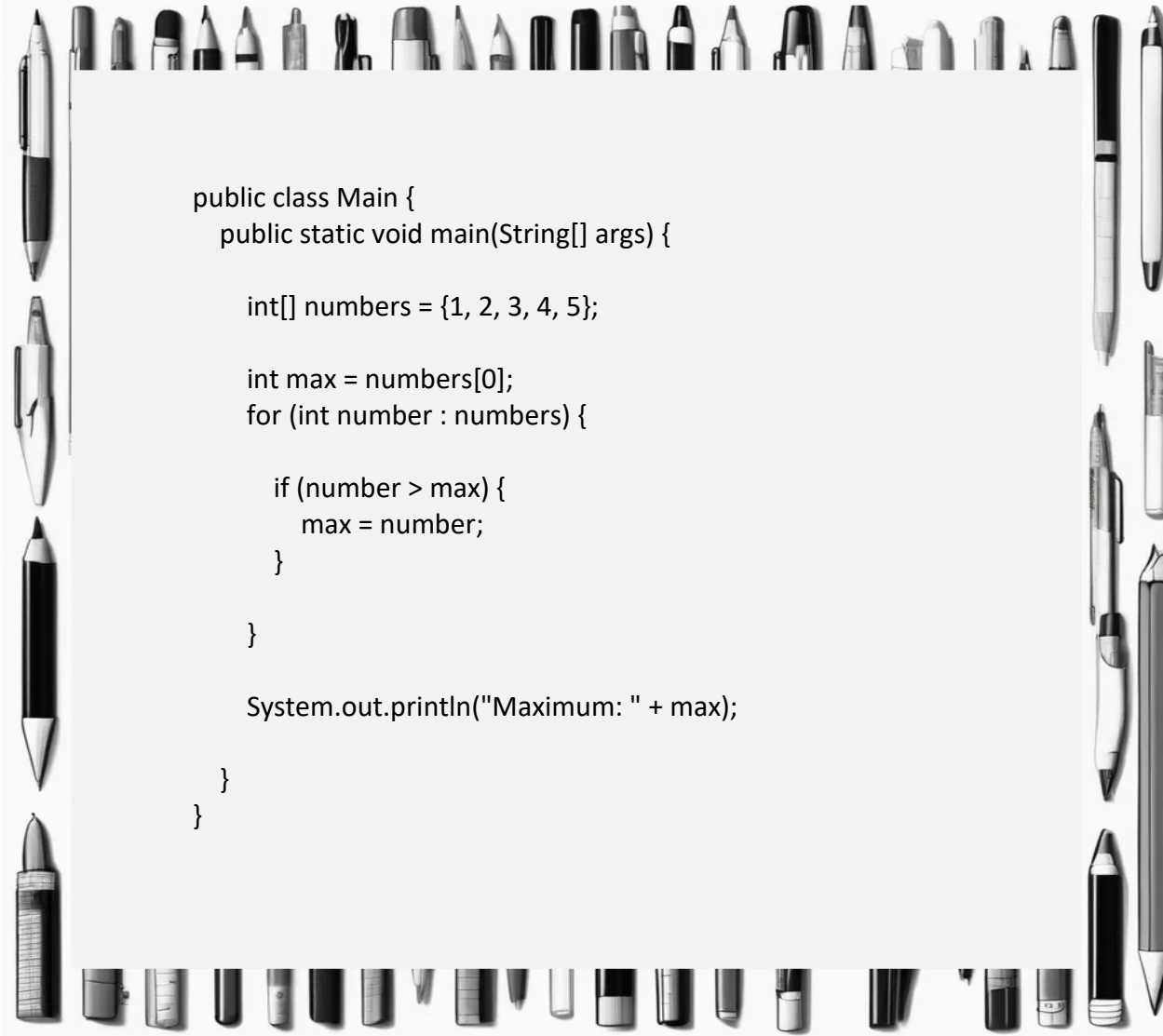
Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        int[] numbers = {1, 2, 3, 4, 5};  
        int sum = 0;  
  
        for (int number : numbers) {  
  
            sum += number;  
  
        }  
  
        System.out.println("Sum: " + sum);  
    }  
}
```

Exercise 4. Find the maximum value in an integer array and print the result.

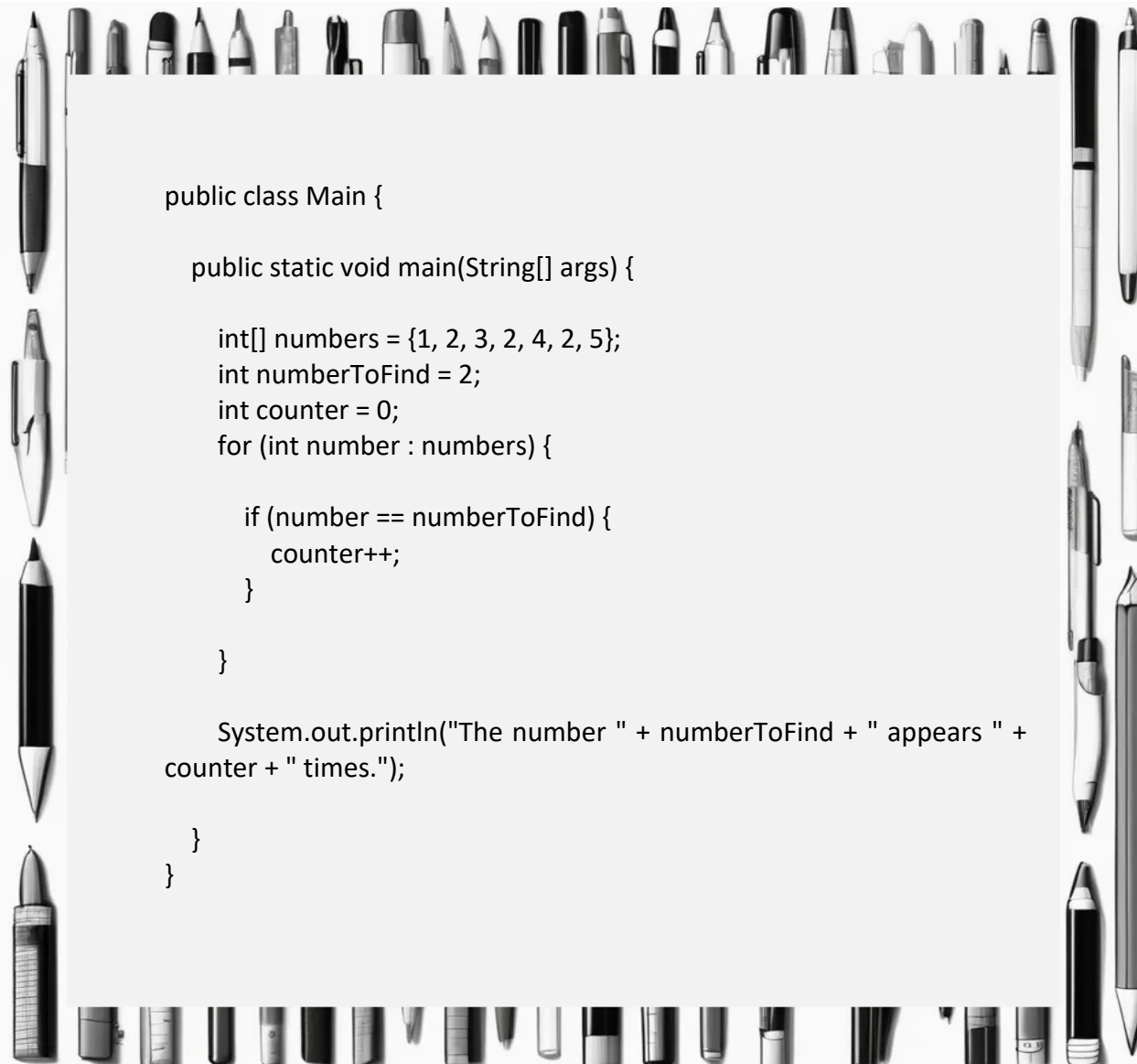
Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        int[] numbers = {1, 2, 3, 4, 5};  
  
        int max = numbers[0];  
        for (int number : numbers) {  
  
            if (number > max) {  
                max = number;  
            }  
  
        }  
  
        System.out.println("Maximum: " + max);  
  
    }  
}
```

Exercise 5. Count how many times a specific number appears in an integer array.

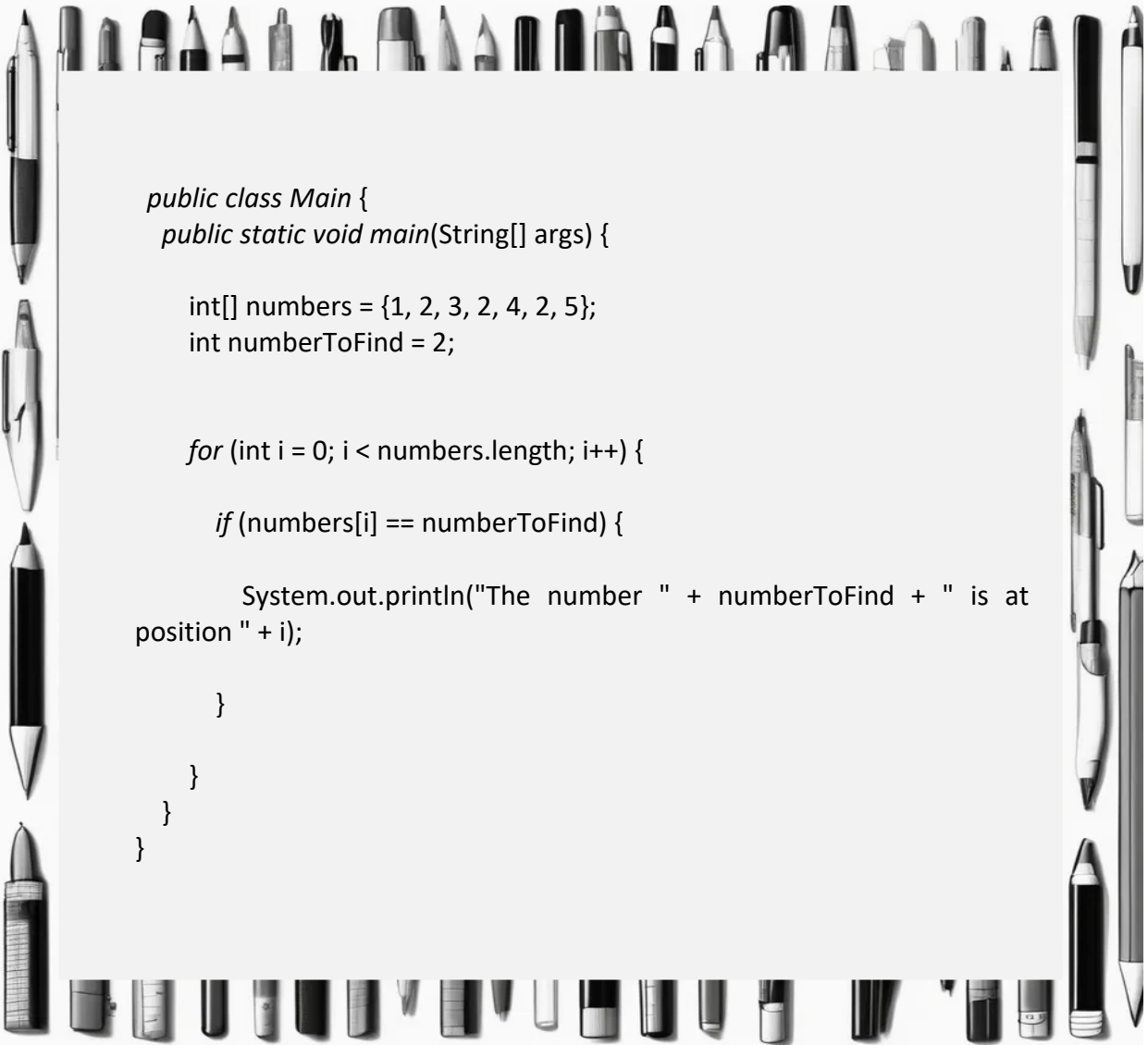
Solution:



```
public class Main {  
  
    public static void main(String[] args) {  
  
        int[] numbers = {1, 2, 3, 2, 4, 2, 5};  
        int numberToFind = 2;  
        int counter = 0;  
        for (int number : numbers) {  
  
            if (number == numberToFind) {  
                counter++;  
            }  
  
        }  
  
        System.out.println("The number " + numberToFind + " appears " +  
            counter + " times.");  
  
    }  
}
```

Exercise 6. Print the positions of a specific number in an integer array.

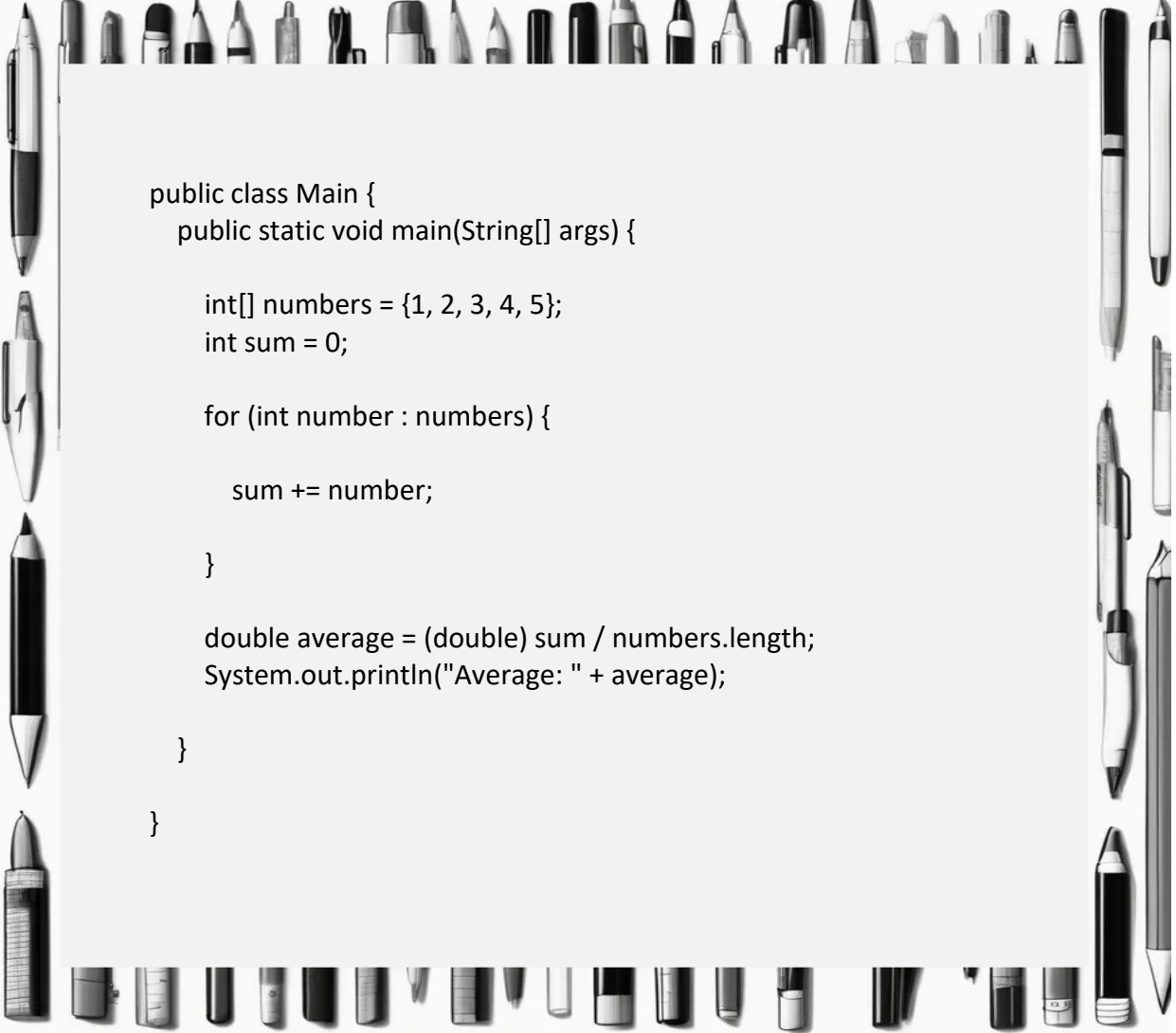
Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        int[] numbers = {1, 2, 3, 2, 4, 2, 5};  
        int numberToFind = 2;  
  
        for (int i = 0; i < numbers.length; i++) {  
  
            if (numbers[i] == numberToFind) {  
  
                System.out.println("The number " + numberToFind + " is at  
position " + i);  
  
            }  
  
        }  
  
    }  
}
```


Exercise 7: Create an array of integers and calculate the average of its elements.

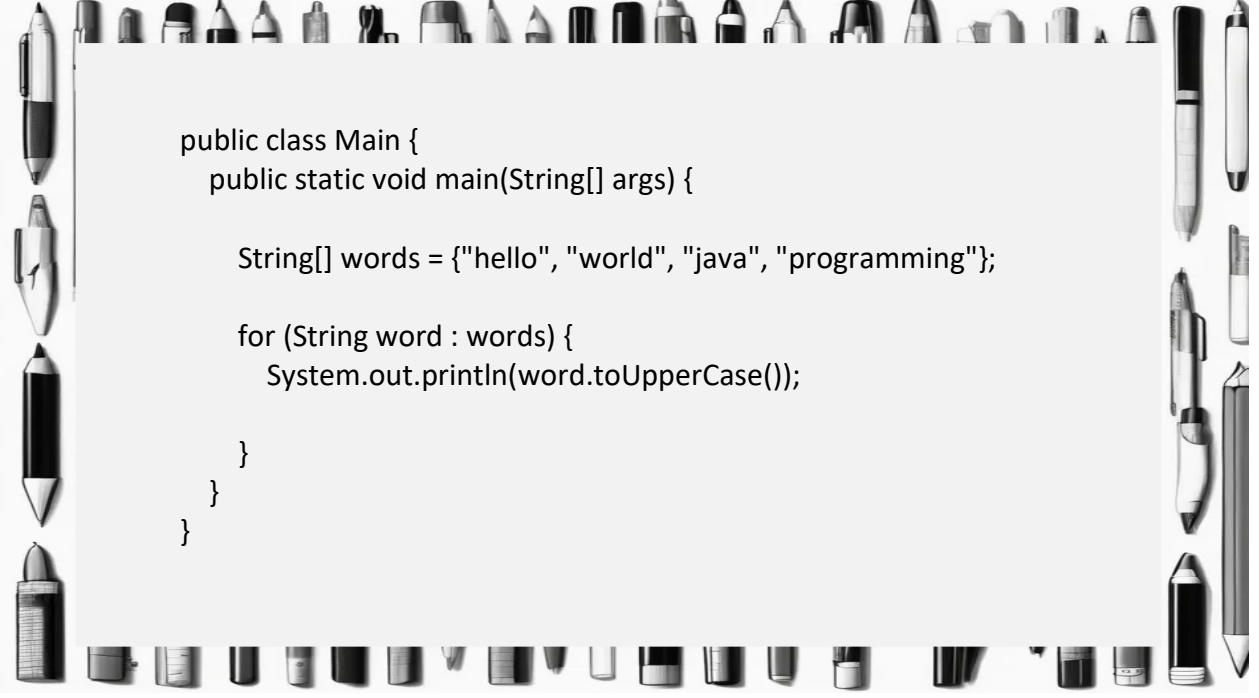
Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        int[] numbers = {1, 2, 3, 4, 5};  
        int sum = 0;  
  
        for (int number : numbers) {  
  
            sum += number;  
  
        }  
  
        double average = (double) sum / numbers.length;  
        System.out.println("Average: " + average);  
  
    }  
}
```

Exercise 8. Traverse a string array and convert each string to uppercase, then print them.

Solution:



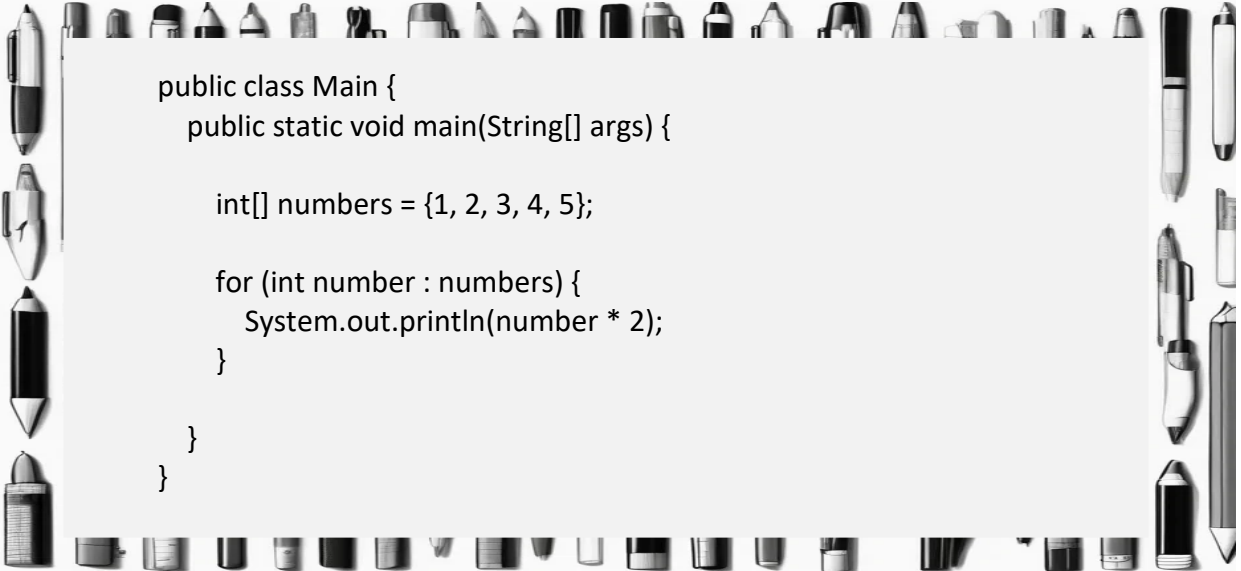
```
public class Main {  
    public static void main(String[] args) {  
  
        String[] words = {"hello", "world", "java", "programming"};  
  
        for (String word : words) {  
            System.out.println(word.toUpperCase());  
        }  
    }  
}
```

As we have seen, to convert text to uppercase in Java, we use the `toUpperCase()` method from the `String` class. This method takes a text string and returns a new string with all letters converted to uppercase. Here I explain how to do it in a simple way. Let's look at another simple example:

```
String text = "hello world";  
  
String textInUppercase = text.toUpperCase();  
  
System.out.println(textInUppercase);
```

Exercise 9. Traverse an array of integers and multiply each element by 2, printing the results.

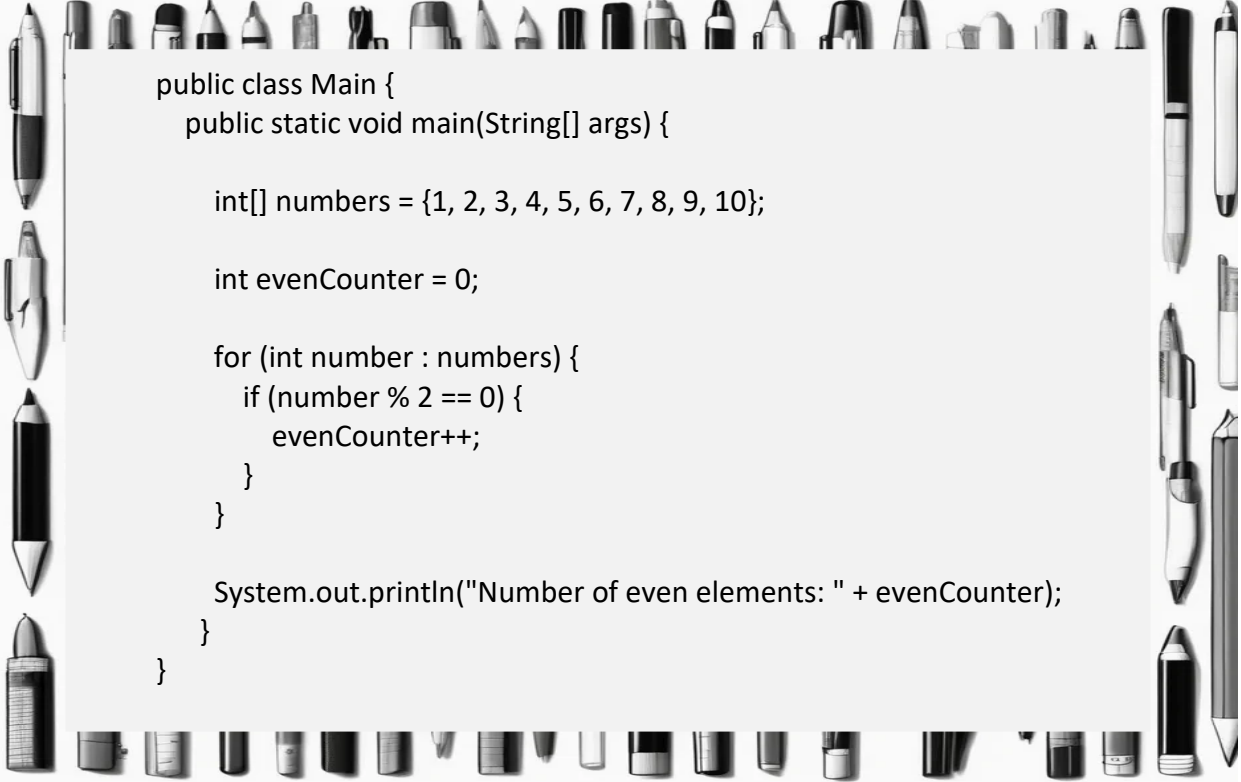
Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        int[] numbers = {1, 2, 3, 4, 5};  
  
        for (int number : numbers) {  
            System.out.println(number * 2);  
        }  
    }  
}
```

Exercise 10. Traverse a String array and print only even-numbered elements.

Solution:



```
public class Main {  
    public static void main(String[] args) {  
  
        int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
  
        int evenCounter = 0;  
  
        for (int number : numbers) {  
            if (number % 2 == 0) {  
                evenCounter++;  
            }  
        }  
  
        System.out.println("Number of even elements: " + evenCounter);  
    }  
}
```

Chapter 17.

Decoding Communication Systems

My escape companions have given me homework. I think what they're trying to do is help me keep my mind occupied so I can be a little calmer. Bud is going to access the guards' communication system and has asked me to help him.

If anyone finds out that we've escaped before I can return to prison, the plan will have been a complete failure, at least for me. Everything is well calculated so that doesn't happen, but if we run into any jailer, we must prevent them from raising the alarm. For this, we'll create a firewall.



Inside the prison, the situation will be a little different. If a guard discovers us, we'll have no choice but to subdue and tie them up. As long as they can't call for reinforcements, the plan will continue.

Bud and Pedro are going to carry the computer with them and, among other things, they'll use it to monitor and block communications between prison guards. Of course, not all communications, just the ones we're interested in.

So, taking advantage of having just learned the for-each loop, I'm going to make the five Programs they've asked me for using that concept. Of course, before starting the plan, Bud and Pedro will review them to make sure everything is correct.

For the first exercise, I'll have to intercept some of the messages, specifically those that mention one of the three sectors we're going to pass through: sector 6, 12, and 14

Exercise 1. Intercept Messages.

- Initialize an array with three elements:

"Message 12: sector 12"

"Message 6: sector 6"

"Message 14: sector 14"

- Use a for-each loop to print the current message to the console preceded by the string "Intercepted message: ".
-

Solution:

```
public class Main {  
    public static void main(String[] args) {  
        String[] messages = {  
  
            "Message 12: sector 12",  
            "Message 6: sector 6",  
            "Message 14: sector 14"  
  
        };  
    }  
}
```

```
        for (String message : messages) {  
            System.out.println("Intercepted message: " + message);  
        }  
    }  
}
```

In case any message from another sector slips through, we must do something to filter them and avoid false positives. Remember that only messages referring to sectors 6, 12, and 14 are important to us

Exercise 2. Filter messages.

- Inicializa un array con catorce elementos:

```
"Message 1: Alert in sector 1",  
"Message 2: Alert in sector 2",  
"Message 3: Alert in sector 3",  
"Message 4: Alert in sector 4",  
"Message 5: Alert in sector 5",  
"Message 6: Alert in sector 6",  
"Message 7: Alert in sector 7",  
"Message 8: Alert in sector 8",  
"Message 9: Alert in sector 9",  
"Message 10: Alert in sector 10",  
"Message 11: Alert in sector 11",  
"Message 12: Alert in sector 12",  
"Message 13: Alert in sector 13",  
"Message 14: Alert in sector 14"
```

- Use a for-each loop to filter and display important messages:

Inside the for-each loop, verify if the current message contains the number "6", "12", or "14".

If the condition is met, print the current message to the console preceded by the string "Important message: ".

```
public class Main {  
    public static void main(String[] args) {  
        String[] messages = {  
  
            "Message 1: Alert in sector 1",  
            "Message 2: Alert in sector 2",  
            "Message 3: Alert in sector 3",  
            "Message 4: Alert in sector 4",  
            "Message 5: Alert in sector 5",  
            "Message 6: Alert in sector 6",  
            "Message 7: Alert in sector 7",  
            "Message 8: Alert in sector 8",  
            "Message 9: Alert in sector 9",  
            "Message 10: Alert in sector 10",  
            "Message 11: Alert in sector 11",  
            "Message 12: Alert in sector 12",  
            "Message 13: Alert in sector 13",  
            "Message 14: Alert in sector 14"  
  
        };  
    }  
}
```

```
for (String message : messages) {  
    if (message.contains("6") || message.contains("12") ||  
        message.contains("14")) {  
        System.out.println("Important message: " + message);  
    }  
}  
}
```

Once we've identified and filtered the messages that concern us, it's time to block them. This way we'll prevent them from spreading and alerting all the prison guards..

Exercise 3. Block Communications

Initialize an array with three elements:

"Message 12: sector 12"

"Message 6: sector 6"

"Message 14: sector 14"

- Use a for-each loop to display blocked messages. Inside the for-each loop, print the current message to the console preceded by the string "Blocking message:".

Solution:

```
public class Main {  
    public static void main(String[] args) {  
        String[] messages = {  
            "Message 12: Alert in sector 12",  
            "Message 6: Alert in sector 6",  
            "Message 14: Alert in sector 14"  
        };  
  
        for (String message : messages) {  
            System.out.println("Blocking message: " + message);  
        }  
    }  
}
```

In addition to blocking them, we need to redirect them to our main Program. Bud and Pedro use two different computers, and both will need to be alert to know if any guard has raised the alarm.

Exercise 4. Redirect communications.

- Initialize the array with three elements:
- "Message 12: sector 12"
- "Message 6: sector 6"
- "Message 14: sector 14"
- Use a for-each loop to iterate over and display redirected messages:

Inside the for-each loop, print the current message to the console preceded by the string "Redirecting message: " and followed by the string " to external system".

Solution:

```
public class Main {  
    public static void main(String[] args) {  
        String[] messages = {  
  
            "Message 12: sector 12",  
            "Message 6: sector 6",  
            "Message 14: sector 14"  
        };  
  
        for (String message : messages) {  
            System.out.println("Redirecting message: " + message + " to  
external system");  
        }  
    }  
}
```

In addition to communications between guards, there is another security measure that concerns us: camera monitoring. The prison uses artificial intelligence software that analyzes images and generates a message every minute. Three of these messages are harmless, but we must create software that detects the one that incriminates us. After that, Bud will complete the software so that these communications are blocked, just as we did with the previous exercises. Below, I show you the four messages generated by the AI software. As you can imagine, the message that says 'Alert' is the one we must identify and neutralize.

Exercise 5. Camera Monitoring

- Initialize the array with four elements:

"Image 1: Empty hallway"

"Image 2: Alert"

"Image 3: Open door"

"Image 4: All clear"

- Use a for-each loop to iterate over each element of the images array. Detect Suspicious Activity.

Inside the for-each loop, verify if the current image contains the words "Open door" or "Guard".

If the condition is met, print the message to the console preceded by the string "Suspicious activity detected: ".

If the condition is not met, print the message to the console preceded by the string "Normal monitoring: ".

Solution:

```
public class Main {  
    public static void main(String[] args) {  
        String[] images = {  
  
            "Image 1: Empty hallway",  
            "Image 2: Alert",  
            "Image 3: Open door",  
            "Image 4: All clear"  
  
        };  
    }  
};
```

```
for (String image : images) {  
    if (image.contains("Open door") || image.contains("Guard")) {  
        System.out.println("Suspicious activity detected: " + image);  
    } else {  
        System.out.println("Normal monitoring: " + image);  
    }  
}  
}
```

I've done my part. With these Programs, Bud and Pedro have everything they need. From now on, it's my turn to rest and mentally prepare for the final phase of the escape. So far, everything is going perfectly and there haven't been any setbacks. I hope the good streak continues.

Chapter 18.

Intercepting Guards' Conversations

During the last phase of our plan, Rich and Pedro will stay a bit behind, and I'll go ahead to take care of opening the door. I have very clear Instructions, so I don't think there will be any problem.

Obviously, Bud and Pedro have much more Programming skill. In theory, they should be in charge of opening the door, but they will take care of another equally important function: controlling the guards' communications and monitoring the cameras.

During the escape, Pedro will need to access one of the control posts in the main hallway. Remember that these posts are only occupied during the day. At night there's no need for anyone to be watching, so the idea is to use the computer there to monitor all the prison cameras. This way, we'll know if anyone is approaching.



To be even more certain that no one discovers us, Bud will be with me and will be in charge of two things. Controlling the internal communication of the guards and, at the same time, communicating with Pedro. This way, if Pedro sees something suspicious, he can warn Bud, and Bud can warn me, so we can return to our cells or hide.

Things aren't always simple, and in this case everything gets complicated. The problem is that all incoming and outgoing communications from Bud's computer are monitored. It's not a normal computer; remember that Bud is a prisoner; he has many restricted options and is not allowed to communicate with people from outside.

So, to communicate among ourselves, we've had no choice but to create a coded message system. The idea is to replace each letter with symbols. This way, the messages will be totally impossible to read if you don't know how to decipher them. To avoid even more suspicion, we'll only send messages if it's absolutely necessary. Even if the guards become suspicious, by the time they realize it we'll have finished.



Once we get out, we'll have approximately three hours until room inspection. Therefore, I must hurry. It will take me about 30 minutes to reach Pedro's friend's house. From there, I'll need another 15 minutes to reach the library. After the meeting with Chani and Rich, I'll need another 15 minutes to reach the parking lot where I'll meet Phil. Then, it will take me between 15 and 30 minutes to return to the prison and, possibly, another 15 to 30 additional minutes to reach my cell, as I might have to hide and wait until there are no guards.

This adds up to a total of between 1 hour and 45 minutes and 2 hours. Considering possible setbacks, I calculate a margin of 15 additional minutes, making a total of 2 hours and 15 minutes. Therefore, the meeting should last approximately half an hour. Chani's cousin already knows my itinerary and the time I have, so I assume he will have everything planned according to that. It seems that everything is going as planned, so I'm quite calm. Now let's review together that the Programs for encoding and decoding messages work.

Exercise 1. Create a Program to encode text entered from keyboard, replacing all lowercase letters with specific defined symbols. It will work in a loop that will process multiple messages.

Solution:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("Enter text to encode (or type 'exit' to end):");
            String text = scanner.nextLine();
            if (text.equalsIgnoreCase("exit")) {
                break;
            }

            String encodedText = text.replace('a', '!')
                                    .replace('b', '@')
                                    .replace('c', '#')
                                    .replace('d', '$')
                                    .replace('e', '%')
                                    .replace('f', '^')
                                    .replace('g', '&')
                                    .replace('h', '*');
```

```

        replace('i', '(')
        .replace('j', ')')
        .replace('k', '-')
        .replace('l', '_')
        .replace('m', '=')
        .replace('n', '+')
        .replace('o', '[')
        .replace('p', ']')
        .replace('q', '{')
        .replace('r', '}')
        .replace('s', ';')
        .replace('t', ':')
        .replace('u', '<')
        .replace('v', '>')
        .replace('w', ',')
        .replace('x', '.')
        .replace('y', '?')
        .replace('z', '/');

        System.out.println("Encoded text: " + encodedText);
    }
}

```

Exercise 2. Create a Program to decode previously encoded text, replacing the symbols with the original lowercase letters. It will work in a loop to process multiple messages.

Solution:


```

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("Enter text to decode (or type 'exit' to end):");
            String encodedText = scanner.nextLine();
            if (encodedText.equalsIgnoreCase("exit")) {
                break;
            }
            String decodedText = encodedText.replace('!', 'a')
                .replace('@', 'b')
                .replace('#', 'c')
                .replace('$', 'd')
                .replace('%', 'e')
                .replace('^', 'f')
                .replace('&', 'g')
                .replace('*', 'h')
                .replace('(', 'i')
                .replace(')', 'j')
                .replace('-', 'k')
                .replace('_', 'l')
                .replace('=', 'm')
                .replace('+', 'n')
                .replace('[', 'o')
                .replace(']', 'p')
                .replace('{', 'q')
                .replace('}', 'r')
                .replace(';', 's')
                .replace(':', 't')
                .replace('<', 'u')
                .replace('>', 'v')
                .replace(',', 'w')
                .replace('.', 'x')
                .replace('?', 'y')
                .replace('/', 'z');

            System.out.println("Decoded text: " + decodedText);
        }
    }
}

```

Chapter 19.

Functions and OOP

Before moving on to the final phase of our plan, you need to understand a couple more things. I'm referring to functions and object-oriented programming exercises. At this point, you already master the syntax, but you program in a linear way. We've only made programs within the Main class. In real programs, it's impossible to find this linearity. Programs reuse code and it's stored in different classes. Then, from the Main class, they are called and executed depending on the order in which we need them.

We're going to learn to reuse code and work with different classes; for this, we'll start with functions and methods. In Java, the terminology for "methods" and "functions" can be a bit confusing, especially if you come from other programming languages. In Java terms:

Method: It's a function that is defined within a class. In Java, all functions must be inside a class, so all functions in Java are methods. The term "method" is more common in Java.

Function: It's a more general term and is used in many programming languages to refer to a block of code that performs a specific task. In Java, it refers to the same thing as a method, but is used less frequently.

Difference between public, protected or private

public: A method or variable that is declared as public can be accessed from any other class. It doesn't matter which package it's in. We use it when we want the method or variable to be globally available.

When a method or variable is declared as **private**, then it will only be accessible within the same class in which it's defined. It's used to protect data from external access.

Finally, a **protected** method or variable is accessible within the same package and by subclasses, even if they're in different packages.

Difference between methods that return a value and "void" methods

- **Methods that return a value:** These methods specify a return type (for example, int, String, boolean, etc.) in their declaration. They must use the return keyword to return a value of the specified type. They are useful when you need to process data and return a result.
- **Void methods:** These methods use the void keyword to indicate that they won't return any value.

Reusing Static Methods within the Main class

- We'll start with a simple example. The idea is to create some lines of code that add two to a number. I'll write the code and then analyze it bit by bit.

```
public class Main {  
  
    public static void addTwo(int number) {  
        int result = number + 2;  
        System.out.println("The result of adding 2 to " + number + " is: " + result);  
    }  
  
    public static void main(String[] args) {  
  
        addTwo(5);  
  
    }  
}
```

Let's go with the explanation. Our method is highlighted in black. But, first of all, we see the line that refers to the class we're working in. As I mentioned in the first paragraphs of the topic, all methods must go inside a class; otherwise, the Program will return an error message. In this case, the Main class.

But, on the other hand, it goes above the main method of the class:

```
public static void main(String[] args) {  
}
```

Therefore, this class will have two methods, but only the code inside the main one is executed. That's why, if we want to use a secondary method, we must call it from the main one. In this example, we've created a secondary method, but there could be as many as necessary.

Inside the main method, `public static void main(String[] args)`, we see how the `addTwo` method is being called:

```
addTwo(5);
```

We place the number we want to add two to between parentheses. As we've created the method, it's mandatory that we enter a value in the parenthesis, since we've specified it in the code by writing `(int number)`. But we won't always need to indicate a value; that leads us to the next example.

- Now we're going to create the same example, but we want that, when we call the method, it asks us via keyboard for the number we want to enter and then shows us what the sum of that number plus two is.

```
import java.util.Scanner;

public class Main {

    public static void addTwo() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        int result = number + 2;
        System.out.println("The result of adding 2 to " + number + " is: " + result);
    }

    public static void main(String[] args) {

        addTwo();

    }
}
```

Just like before, we write our method inside the Main class and before the main method. In this case, we've created a code that asks us to enter a number via keyboard and the Program will add 2 to this value and, additionally, display the result on the screen.

- Now let's imagine that, from time to time, we need to input a rather long text in our code. We could write it every time, but the most elegant and orderly way is to create a method to reuse that text. We could do it as I show you below.

```
public class Main {  
  
    public static void text() {  
  
        System.out.println("Sometimes a text can be too long to be using it over and over again");  
        System.out.println("It's better to create a method and then call it whenever we need that  
text to be displayed.");  
        System.out.println("Your Program will be much cleaner and more organized");  
  
    }  
  
    public static void main(String[] args) {  
  
        text();  
  
    }  
}
```

In the previous code, we've seen a method that basically consists of three lines of text. At the moment and anywhere we need those lines to appear, we simply have to call our method and thus we'll avoid having large texts mixed with the syntax.

- As a final example (For now) let's look at a method where we ask for two numbers and it returns their multiplication.

```
import java.util.Scanner;

public class Main {

    public static int multiply(int a, int b) {

        int c = a*b;

        System.out.println("The product of a * b = " + c);

        return c;

    }

    public static void main(String[] args) {

        multiply(4,8);

    }

}
```

With these 4 examples, we've seen 3 void functions and the last one that returns a value. That's why when we create it, instead of static void we use static int. Also, if you've noticed in the three void examples, in the first one we put a variable in parentheses: public static void addTwo(int number). In the other two we didn't. This is because in the first one we wanted to write the number manually ourselves. However, in the second one, the Program didn't need any data since it was going to start executing and ask us to enter data via keyboard. As for the text exercise, we didn't have to specify any data either, the method always shows the same text.

Reusing methods created in classes other than the Main class

The first thing we're going to do is create a class that we'll call Exercises. Initially it will look like this before we start writing our text:

```
public class Exercises {  
  
    // Here we will write our methods.  
  
}
```

Our Main class will be like this:

```
public class Main {  
  
    public static void main(String[] args) {  
  
        // Here we will call them.  
  
    }  
}
```

Now that we have it created, we'll go example by example writing the methods inside it and then calling them from within the Main method of the Main class.

- In the first example, we're going to create a method in the Exercises class that tells us whether a number is even or not. We'll manually enter the number when we call the method from the Main class.

This is the code for our method:

```
public class Exercises {  
  
    public static boolean isEven(int number) {  
  
        if (number % 2 == 0) {  
            System.out.println("The number is even");  
        } else System.out.println("The number is odd");  
  
        return number % 2 == 0; // Returns true if the number is even, false if it's odd  
    }  
  
}
```

So far there's not much to get lost, the exercise is simple. What you should pay attention to is which class it's in and that within it we've created a public static boolean method. Now we must call it from the Main class.

To do this, we always follow a very simple structure. Inside our Main class and inside the main method, we write the name of our class, then a dot, and all of it followed by the name of the method and parentheses. If during the method declaration we didn't use any variables in the parentheses, when we call the method from the main method, the parentheses will be empty. If we used any variables during the declaration, we'll put the data, or data, that we need in the parentheses. In this case, it would look like this:

```
Exercises.isEven(6);
```

The complete code of the Main method is as follows:

```
public class Main {  
  
    public static void main(String[] args) {  
  
        Exercises.isEven(6);  
  
    }  
  
}
```

- Now we're going to do the same exercise, but in this case we have to enter the number via keyboard, then the Program will tell us if it's even or not.

In the Exercises class is where we do all the work. This is how it would look:

```
import java.util.Scanner;  
  
public class Exercises {  
  
    public static void isEven() {  
  
        Scanner exercise = new Scanner(System.in);
```

```
System.out.println("Enter a number");  
int number = exercise.nextInt();  
  
if (number % 2 == 0) {  
    System.out.println("The number is even");  
} else System.out.println("The number is odd");  
  
}  
  
}
```

We see that we import the Scanner in this class and not in the Main class. Otherwise, it's a very simple code.

To call the method from the Main class we'll do it as follows:

```
public class Main {  
  
    public static void main(String[] args) {  
  
        Exercises.isEven();  
  
    }  
}
```

- To finish, we use another void method of String type. The same one we used at the beginning of the topic.

The Exercises class will be as follows:

```
public class Exercises {  
    public static void text() {  
  
        System.out.println("Sometimes a text can be too long to be using it over and over again");  
        System.out.println("It's better to create a method and then call it whenever we need that  
text to be displayed.");  
        System.out.println("Your Program will be much cleaner and more organized");  
  
    }  
}
```

And this will be the Main class:

```
public class Main {  
    public static void main(String[] args) {  
  
        Exercises.text();  
  
    }  
}
```

Difference between class and instance

I want to explain what this "static" that appears all the time in our methods is, but for that you must first understand the difference between class and instance. A class, as we had mentioned, is a mold or template that defines the properties (attributes) and behaviors (methods) common to all objects that will be created from it.

An instance is an object created from a class that has specific values for the attributes defined in the class.

Difference between static and dynamic methods

Static methods (All the ones we've been using so far) are directly associated with the class, not with specific objects of that class. This means they can be invoked without needing to create an instance (object) of the class.

For example:

```
public class Calculator {  
    public static int add(int a, int b) {  
        return a + b;  
    }  
}
```

// To use the add method, you don't need to create an object. We could call it from the Main class like this:

```
int result = Calculator.add(5, 10);
```

Dynamic methods belong to specific instances of a class. Each object created from a class has its own copy of dynamic methods.

```
public class Car {  
    public void accelerate() {  
        System.out.println("The car is accelerating.");  
    }  
}  
  
// To use the accelerate method, you need to create an instance (object)  
Car myCar = new Car();  
myCar.accelerate();
```

The concept of object is still something unknown to you, but we're going to remedy that. I know that after all this time programming linearly, the use of methods has changed your schemes a bit and now, on top of that, we have instances. But the truth is that it's much easier than it seems. In what remains of the topic, we're going to dive right into it and soon you'll have no doubts about it.

Working with instances or objects

What is an Object?

An object is a concrete entity that we can use in our programs. Imagine you have a blueprint for building a house. This blueprint would be the class, which describes how the house will be, but the real house you build from the blueprint is the object. In programming, an object is an instance of a class and contains attributes (characteristics) and methods (actions).

Classes vs Objects

- Class: It's like a blueprint or recipe. It describes what characteristics and behaviors the objects created from it will have.
- Object: It's a concrete instance created from the class. It's the real "thing" that we can use and manipulate in our program.

What is a Constructor in Java?

A constructor is a special type of method that is used to initialize objects. It's the first method that's called when we create an object of a class. Its purpose is to give initial values to the object's attributes. It's created inside the class to which that object belongs. We'll see it much more clearly in the example I show you in the next point.

Basic Example of a Class and Object

- Create the class:

```
public class Person {  
    String name;  
    int age;  
  
    // Constructor  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
}
```

```
// Method to show the person's information
public void showInformation() {
    System.out.println("Name: " + name + ", Age: " + age);
}
}
```

- Now for the object:

```
public class Main {
    public static void main(String[] args) {
        // Create an object of the Person class
        Person person1 = new Person("John", 25);

        // Use the object to call the showInformation method
        person1.showInformation();
    }
}
```

As you can see, in the Person class we have two methods. The first is the constructor. Thanks to it, later in the Main class, we'll be able to create objects and assign them values. The next method will simply display the values on screen. It uses the information provided by the constructor method.

In the Main class, we create an object, which in this case we call Person1, and assign it the name and age values. For this specific case, the name is John and the age is 25 years. Once our object has been created and has assigned values, we use the second method to display the information on screen.

Definition of Getter Methods

Purpose: A getter method is used to obtain the value of a private attribute of a class. It's a safe way to access data without allowing it to be directly modified..

```
public class Person {  
    private String name;  
    // Getter method for the name attribute  
    public String getName() {  
        return name;  
    }  
}
```

Definition of Setter Methods

Purpose: A setter method is used to establish or modify the value of a private attribute of a class. It allows control over how the value is changed and can include validations.

```
public class Person {  
    private String name;  
    // Setter method for the name attribute  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

Example of Getters and Setters

Let's see a complete example with a Person class that has an age attribute and uses getter and setter methods to handle it.

```
public class Person {  
    private int age;  
  
    // Getter method for the age attribute  
    public int getAge() {  
        return age;  
    }  
  
    // Setter method for the age attribute  
    public void setAge(int age) {  
        if (age > 0) {  
            this.age = age;  
        }  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Person person1 = new Person();  
        person1.setAge(25); // Set the age using the setter  
        System.out.println("Age: " + person1.getAge()); // Get the age using the getter  
    }  
}
```

This time, in the Person class we haven't created any constructor. Therefore, if we created an object in the Main class, it would be impossible to assign values to its attributes.

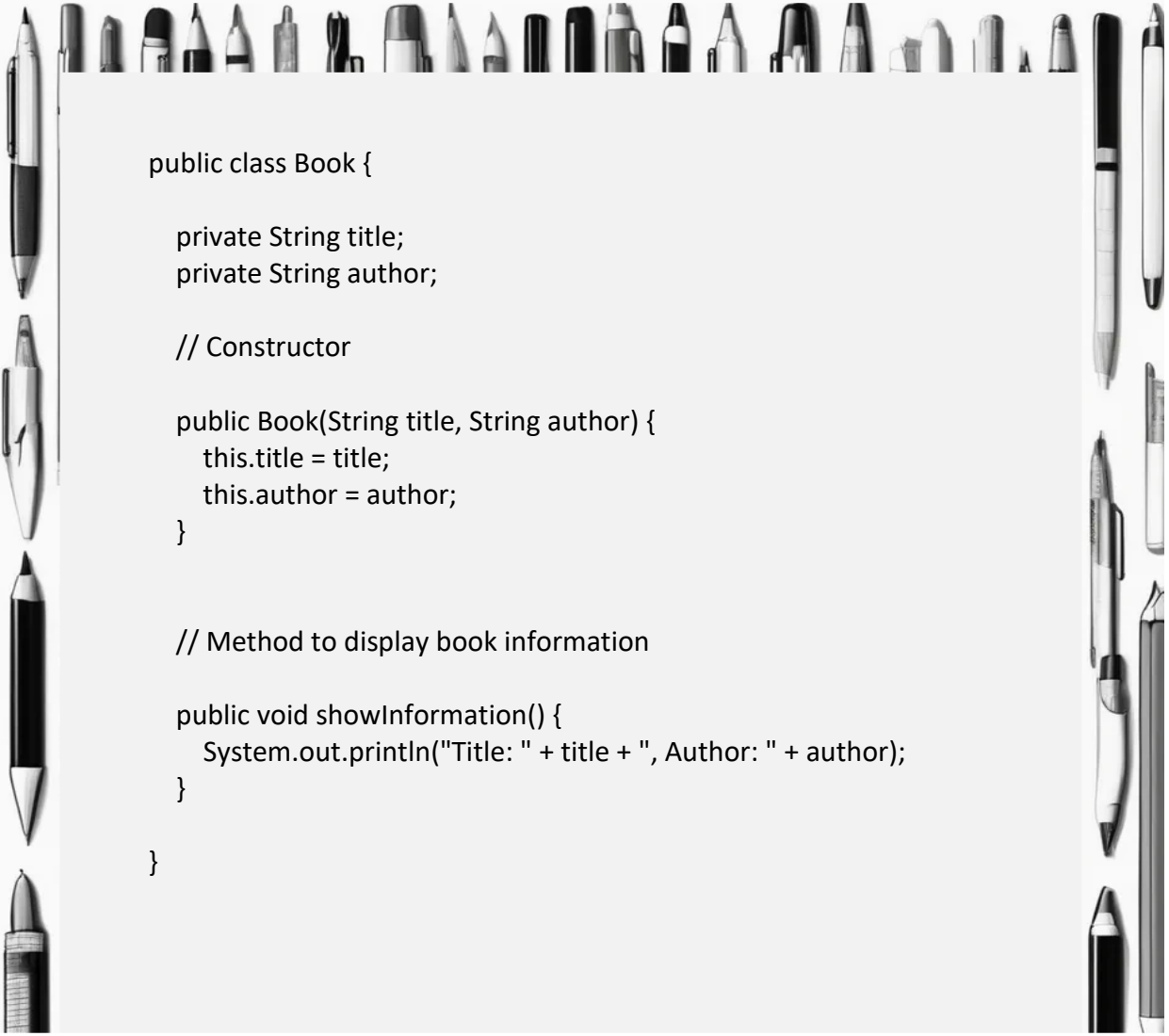
But there's no problem; instead of a constructor method, we'll use a Getter and a Setter method. We'll create it in the Person class and then call it from Main. In this case, we only have one variable, so a Getter and a Setter are created for each variable. Both always follow the same structure that you can see in the example.

In the Main class, the first thing we do is create an object, again Person1. To call the Setter, we put the name of our object, a dot, the name of our Setter, and in parentheses the value we want to assign. As you can observe, to call the Getter, we follow exactly the same structure.

Typical object exercises using constructors

Exercise 1: Create and Display Book Information. Create a Book class with title and author attributes. Use a constructor to initialize these attributes and a method to display the book's information.

Solution:



```
public class Book {  
  
    private String title;  
    private String author;  
  
    // Constructor  
  
    public Book(String title, String author) {  
        this.title = title;  
        this.author = author;  
    }  
  
    // Method to display book information  
  
    public void showInformation() {  
        System.out.println("Title: " + title + ", Author: " + author);  
    }  
}
```

```
public class Main {  
  
    public static void main(String[] args) {  
        Book book1 = new Book("One Hundred Years of Solitude", "Gabriel G.  
Márquez");  
        book1.showInformation();  
  
    }  
}
```

Exercise 2: Create a Student class with name and age attributes. Use a constructor to initialize these attributes and a method to display the student's information.

Solution:

```
public class Student {  
  
    private String name;  
    private int age;  
  
    // Constructor  
  
    public Student(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
}
```

```
// Method to display student information

public void showInformation() {
    System.out.println("Name: " + name + ", Age: " + age);
}

public class Main {

    public static void main(String[] args) {
        Student student1 = new Student("Ana", 20);
        student1.showInformation();
    }
}
```

Exercise 3. Create a Product class with name and price attributes. Use a constructor to initialize these attributes and a method to display the product's information.

Solution:

```
public class Product {  
  
    private String name;  
    private double price;  
  
    // Constructor  
  
    public Product(String name, double price) {  
        this.name = name;  
        this.price = price;  
    }  
  
    // Method to display product information  
  
    public void showInformation() {  
        System.out.println("Name: " + name + ", Price: $" + price);  
    }  
}  
  
public class Main {  
  
    public static void main(String[] args) {  
        Product product1 = new Product("Laptop", 799.99);  
        product1.showInformation();  
    }  
}
```

Exercise 4. Create a Car class with brand and model attributes. Use a constructor to initialize these attributes and a method to display the car's information.

Solution:

```
public class Car {  
  
    private String brand;  
    private String model;  
  
    // Constructor  
  
    public Car(String brand, String model) {  
        this.brand = brand;  
        this.model = model;  
    }  
  
    // Method to display car information  
  
    public void showInformation() {  
        System.out.println("Brand: " + brand + ", Model: " + model);  
    }  
}  
  
public class Main {  
  
    public static void main(String[] args) {  
        Car car1 = new Car("Toyota", "Corolla");  
        car1.showInformation();  
    }  
}
```


Exercise 5. Create a Movie class with title and director attributes. Use a constructor to initialize these attributes and a method to display the movie's information.

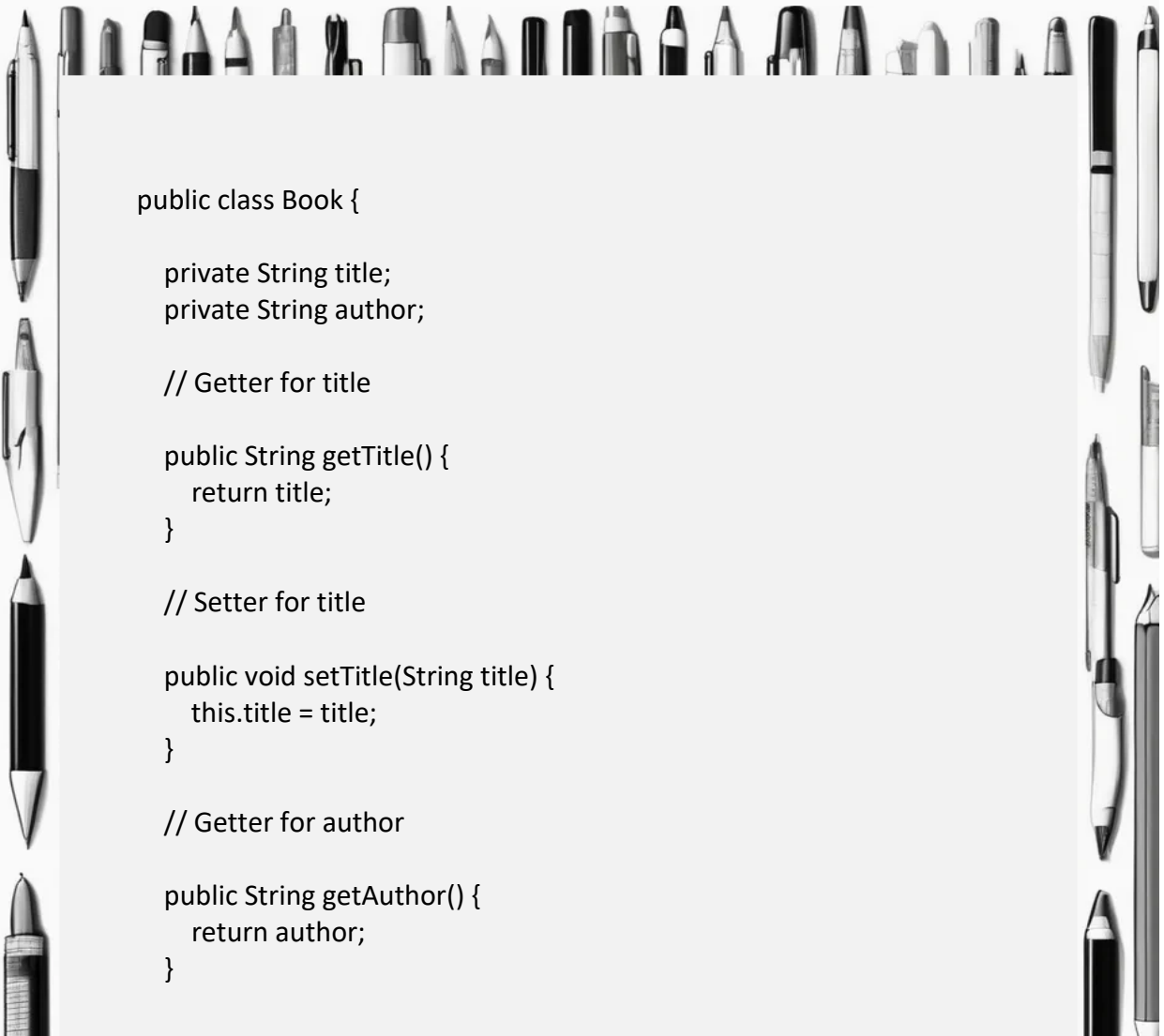
Solution:

```
public class Movie {  
  
    private String title;  
    private String director;  
  
    // Constructor  
  
    public Movie(String title, String director) {  
        this.title = title;  
        this.director = director;  
    }  
  
    // Method to display movie information  
  
    public void showInformation() {  
        System.out.println("Title: " + title + ", Director: " + director);  
    }  
}  
  
public class Main {  
  
    public static void main(String[] args) {  
        Movie movie1 = new Movie("Inception", "Christopher Nolan");  
        movie1.showInformation();  
    }  
}
```

Typical object exercises using Getter and Setter

Exercise 1: Create a Book class with title and author attributes. Use getter and setter methods to access and modify these attributes, and a method to display the book's information

Solution:



```
public class Book {  
  
    private String title;  
    private String author;  
  
    // Getter for title  
  
    public String getTitle() {  
        return title;  
    }  
  
    // Setter for title  
  
    public void setTitle(String title) {  
        this.title = title;  
    }  
  
    // Getter for author  
  
    public String getAuthor() {  
        return author;  
    }  
}
```

```
// Setter for author

public void setAuthor(String author) {
    this.author = author;
}

// Method to display book information

public void showInformation() {
    System.out.println("Title: " + title + ", Author: " + author);
}


}

public class Main {

    public static void main(String[] args) {
        Book book1 = new Book();
        book1.setTitle("One Hundred Years of Solitude");
        book1.setAuthor("Gabriel García Márquez");
        book1.showInformation();
    }
}
```

Exercise 2: Create a Student class with name and age attributes. Use getter and setter methods to access and modify these attributes, and a method to display the student's information.

Solution:



```
public class Student {  
  
    private String name;  
    private int age;  
  
    // Getter for name  
  
    public String getName() {  
        return name;  
    }  
  
    // Setter for name  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    // Getter for age  
  
    public int getAge() {  
        return age;  
    }  
  
    // Setter for age  
  
    public void setAge(int age) {  
        this.age = age;  
    }  
}
```

```
// Method to display student information

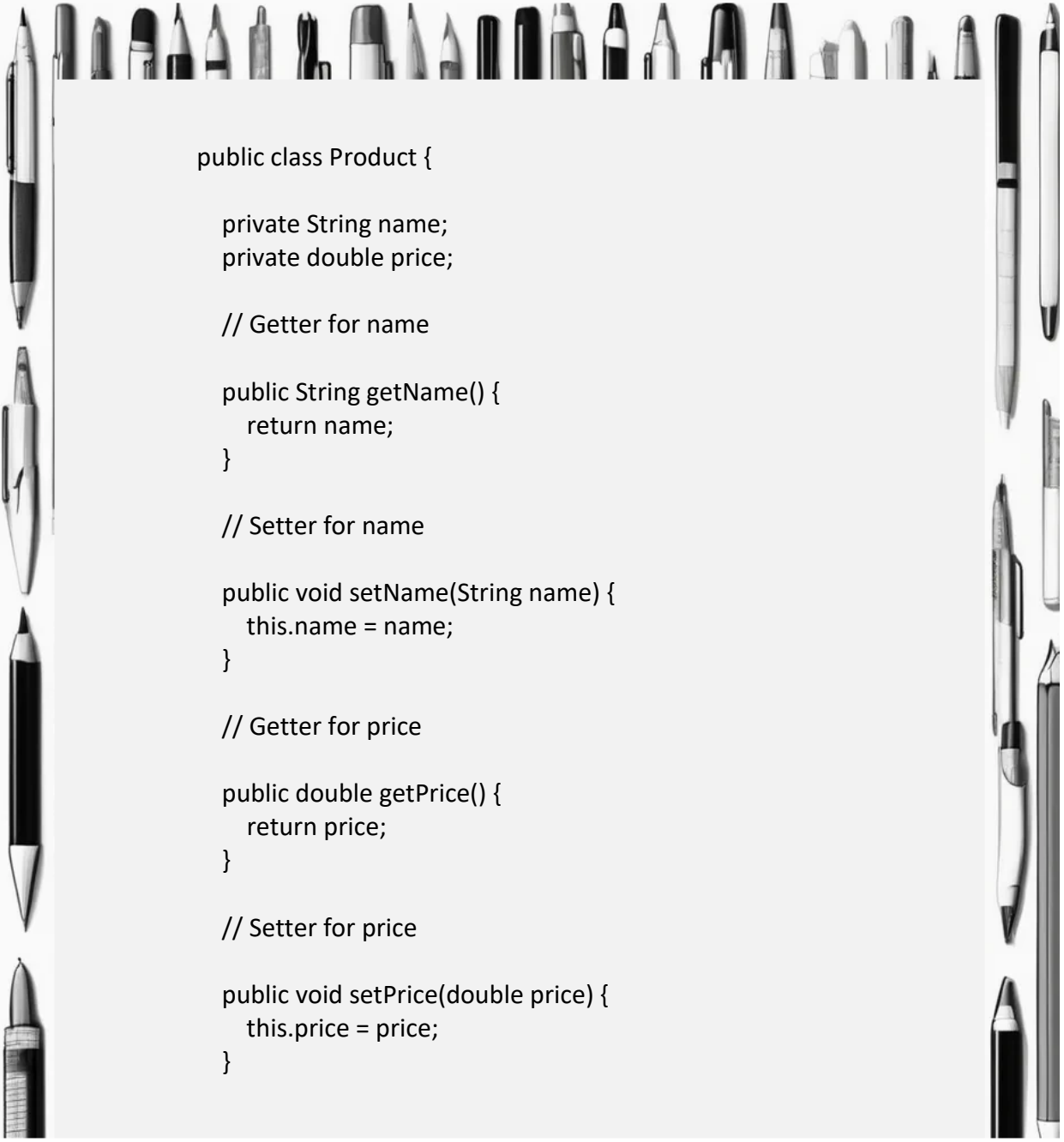
public void showInformation() {
    System.out.println("Name: " + name + ", Age: " + age);
}

public class Main {

    public static void main(String[] args) {
        Student student1 = new Student();
        student1.setName("Ana");
        student1.setAge(20);
        student1.showInformation();
    }
}
```

Exercise 3: Create a Product class with name and price attributes. Use getter and setter methods to access and modify these attributes, and a method to display the product's information.

Solution:



```
public class Product {  
  
    private String name;  
    private double price;  
  
    // Getter for name  
  
    public String getName() {  
        return name;  
    }  
  
    // Setter for name  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    // Getter for price  
  
    public double getPrice() {  
        return price;  
    }  
  
    // Setter for price  
  
    public void setPrice(double price) {  
        this.price = price;  
    }  
}
```

```
// Method to display product information

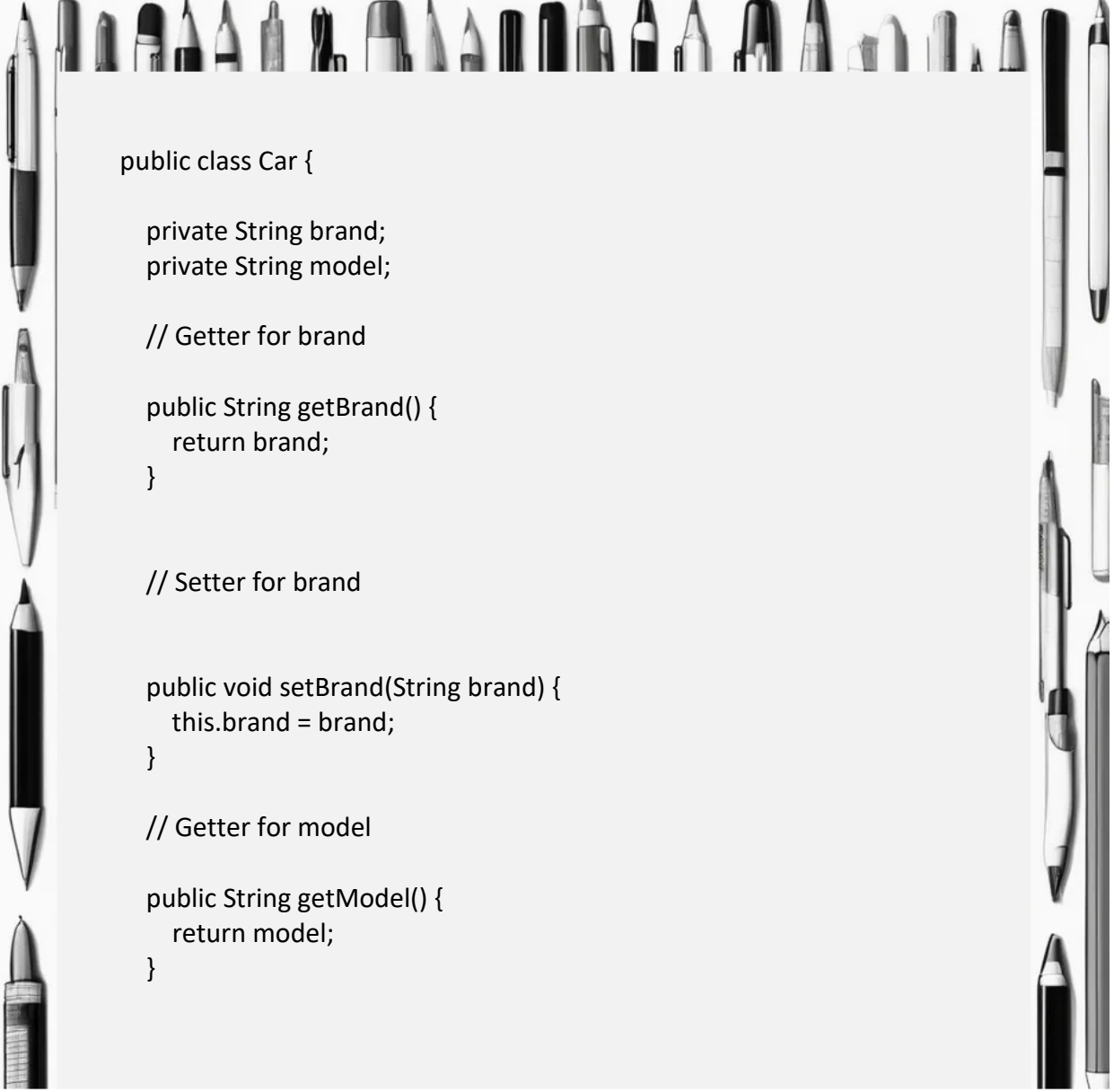
public void showInformation() {
    System.out.println("Name: " + name + ", Price: $" + price);
}

public class Main {

    public static void main(String[] args) {
        Product product1 = new Product();
        product1.setName("Laptop");
        product1.setPrice(799.99);
        product1.showInformation();
    }
}
```

Exercise 4: Create a Car class with brand and model attributes. Use getter and setter methods to access and modify these attributes, and a method to display the car's information.

Solution:



```
public class Car {  
  
    private String brand;  
    private String model;  
  
    // Getter for brand  
  
    public String getBrand() {  
        return brand;  
    }  
  
    // Setter for brand  
  
    public void setBrand(String brand) {  
        this.brand = brand;  
    }  
  
    // Getter for model  
  
    public String getModel() {  
        return model;  
    }  
}
```



```
// Setter for model

public void setModel(String model) {
    this.model = model;
}

// Method to display car information

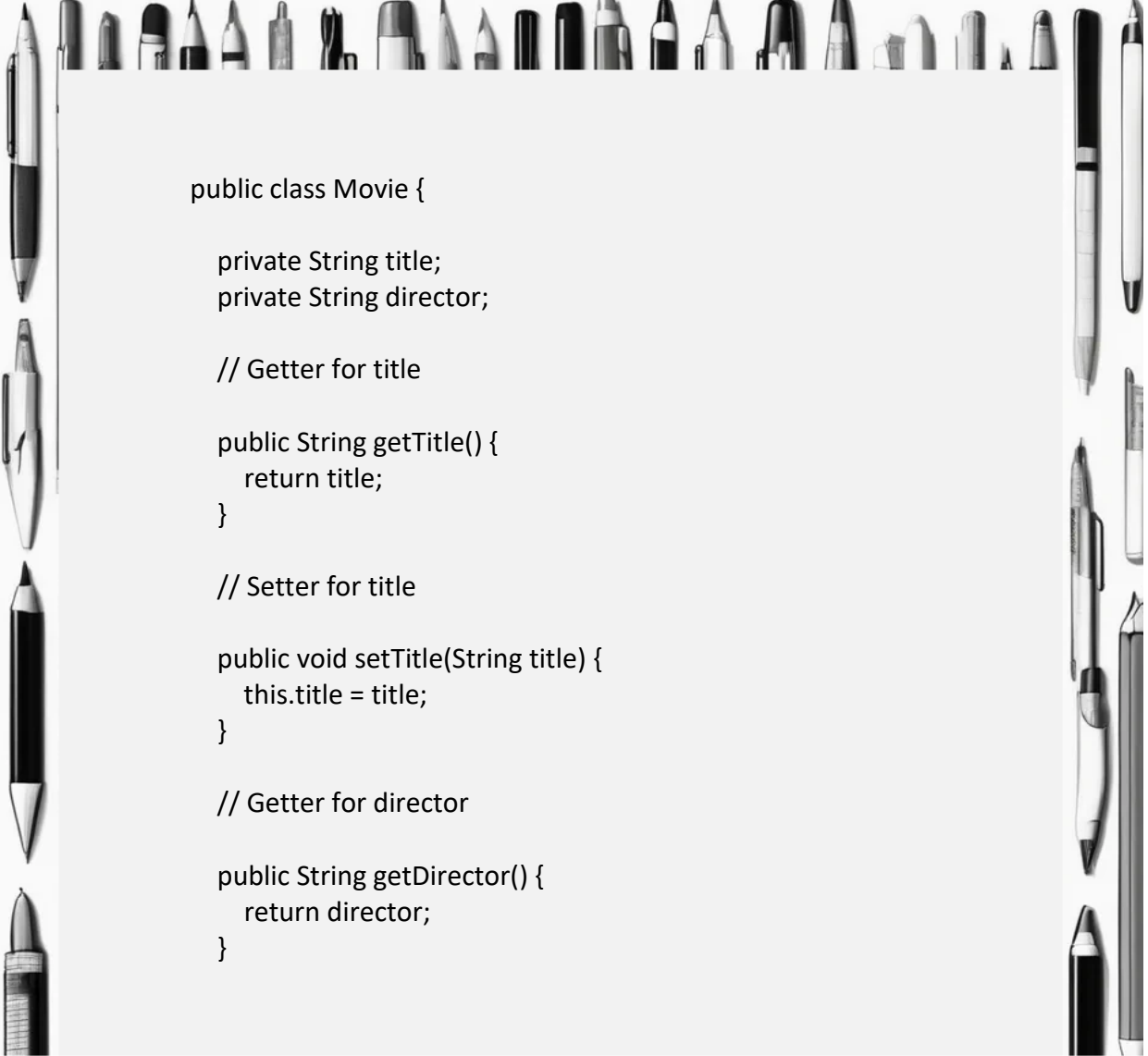
public void showInformation() {
    System.out.println("Brand: " + brand + ", Model: " + model);
}

public class Main {

    public static void main(String[] args) {
        Car car1 = new Car();
        car1.setBrand("Toyota");
        car1.setModel("Corolla");
        car1.showInformation();
    }
}
```

Exercise 5: Create a Movie class with title and director attributes. Use getter and setter methods to access and modify these attributes, and a method to display the movie's information.

Solution:



```
public class Movie {  
  
    private String title;  
    private String director;  
  
    // Getter for title  
  
    public String getTitle() {  
        return title;  
    }  
  
    // Setter for title  
  
    public void setTitle(String title) {  
        this.title = title;  
    }  
  
    // Getter for director  
  
    public String getDirector() {  
        return director;  
    }  
}
```

```
// Setter for director

public void setDirector(String director) {
    this.director = director;
}

// Method to display movie information

public void showInformation() {
    System.out.println("Title: " + title + ", Director: " + director);
}

}

public class Main {

    public static void main(String[] args) {
        Movie movie1 = new Movie();
        movie1.setTitle("Inception");
        movie1.setDirector("Christopher Nolan");
        movie1.showInformation();
    }

}
```

Chapter 20.

The Final Phase, Creating a Door Unlocking Tool

We have everything ready for the escape. As I've already told you, I will be in charge of opening the final door. Right next to the exit there's a desk with a computer from which you can access the entire system and, by the way, open the main door. Normally, five security keys are used and, without them, it's impossible to open it. Needless to say, with my current computer knowledge it would be impossible for me to hack it.

However, Bud and Pedro know the system perfectly. They have been working with it for years and have even Programmed much of the code that controls the main security functions. They wanted to make things easier for me to ensure we can get out quickly and without complications. For this, they have created five Programs that are housed in five different classes. These classes are already hidden within the system, so my only task is to create a Main class and call the methods that are in them.



It's a humble task, but of vital importance. I'm not going in blind; Bud has taken care of showing me the five Programs and explaining what they're for. Additionally, we've already practiced how I should call the methods, so you could say I know by heart what I have to do. I wouldn't want to get nervous and go blank right at the most critical moment.

To give you an idea, I'm going to show you the classes and give you an explanation of what each one is for. Obviously, Bud has spent a lot of time working on them and has checked many times that there are no errors that could compromise our escape.

1. Key Class

I had told you that five keys are needed to open the door normally. We only have one; we got it when we accessed Vicente's office. We're lucky, as it's the most important one; thanks to it we can follow the other steps to, in the end, manage to completely disarm it. This class allows the system to verify that the key we have is still active and valid. Therefore, this class represents a key and verifies if the key is valid.

```
public class Key {  
    private String key;  
  
    public Key(String key) {  
        this.key = key;  
    }  
  
    public boolean Valid() {  
        // Simple key verification  
        return "correctKey".equals(this.key);  
    }  
}
```

Once the access key is entered, the system wants to know who is the person trying to access. We will pretend to be the director. For this, we have his personal identification code. It's very easy to know the identification number of any prison worker; it's no secret, as it appears on any employee document or record.

2. AccessCode Class

This class represents an access code and verifies if the code is correct.

```
public class AccessCode {  
    private int code;  
  
    public AccessCode(int code) {  
        this.code = code;  
    }  
  
    public boolean isValid() {  
        // Simple access code verification (can be any logic)  
        return this.code == 1234;  
    }  
}
```

3. Light class

For the door to open, it is absolutely mandatory that the light beam illuminating the door from the outside is on. This helps increase security and visibility in the area. However, we prefer darkness, so the cameras don't detect us. Therefore, we must make the system believe that the light is on. We'll do this with the following method.

```
public class Light {  
    public void turnOn() {  
  
        // Simulates turning on the light  
  
        System.out.println("The light is on.");  
  
    }  
}
```

4. Alarm class

When the door opens without authorization, an alarm sounds, and we don't want that to happen. We must create a method that indicates that the alarm is activated, although we have already taken care of deactivating it previously.

```
public class Alarm {  
    public void enable() {  
  
        // Simulates that the alarm is active.  
  
        System.out.println("The alarm is activated.");  
  
    }  
}
```

5. Door class

Finally, we want to deceive the security system and indicate that the door is open. We'll simply have to create a class that indicates that, indeed, the door is open. That way, it really will be open and, at last, we'll be able to get out.

```
public class Door {  
    public void open() {  
  
        // Simulates opening the door  
  
        System.out.println("The door is open.");  
    }  
}
```

Main class

In this class, we create the necessary objects and call their methods to open the door.

```
public class Main {  
    public static void main(String[] args) {  
  
        // Create objects  
  
        Key key = new Key("67898");  
        AccessCode accessCode = new AccessCode(787656);  
    }  
}
```



```
Light light = new Light();
Alarm alarm = new Alarm();
Door door = new Door();

// Process to open the door

if (key.isValid() && accessCode.isValid()) {
    light.turnOn();
    alarm.enable();
    door.open();
} else {

    System.out.println("The key or access code is incorrect. The door
cannot be opened.");

}
}
```

Basically, this is everything. Let's hope everything goes well tonight. It has taken a lot of effort to get here, and it would be a big problem not to achieve our escape. Although it would be much worse if we got caught. If that happened, I would be facing a long time locked up, Rich would go unpunished, and justice would never be served.

Chapter 21.

The Outcome

Everything has gone as we expected. We've managed to get out of the prison without any complications. We're continuing with the plan and we're already at Pedro's friend's house. I don't have much time, so the farewell will be short. I have no idea what the next destination of my escape companions will be. They never told me. What I'm clear about is that I'm not going to see them again. It's a very desolate feeling, since, although I haven't known them for long, I've created a great friendship with them and they've been a great support.

From this house, I'm heading on foot to the library. As I walk, I think that being on the street is a very strange sensation. I don't know how to describe it, but the mere fact of walking down any street makes me happy. I hope my new friends do well and I hope the same happens to me. I'm quite nervous because I don't know exactly what's going to happen, and it's a very distressing feeling.



Although Chani is helping me prove my innocence, my relationship with him will never be the same. I plan to leave this city and forget everything that has happened to me here. That also includes Chani. As for Rich, better not to even talk about it. He was one of my best friends and he betrayed me without thinking for some money.

I keep walking and my head won't stop spinning. I guess it's normal, but I have a very bad feeling. Maybe it's nerves. I enjoy what could be my last minutes of freedom, although I can't enjoy them as I would like. I must walk very quickly, as I don't have much time.

Then, without warning, someone grabs my arm and I almost have a heart attack. Nobody knows me in this area, so I wasn't expecting to be stopped for any reason. When I turn around, I got a small surprise: it was Rich. I thought it would be a coincidence, since we had arranged to meet at the same place and were following the same path. But in reality, it wasn't like that. He was waiting there to warn me.



As soon as I saw him, an immense feeling of anger came over me. I thought about hitting him, but before I could say anything, he started talking hurriedly. I could hardly understand him.

I tried hard to understand what he was saying and came to the conclusion that he was leading me into a trap. I don't have much time, so I asked him to explain himself better and more slowly.

Then he began to explain a story. According to him, Chani had planned everything together with his Korean girlfriend. Apparently, it had all been the latter's idea. She had planned it from afar. Additionally, they had Chani's cousin as an accomplice. The main idea was to frame both me and Rich, but it seems they didn't have enough time to access his computer and they gave up.

Continuing with the story, theoretically, Rich didn't know anything until Chani and his girlfriend contacted him a few weeks ago. They offered him a good sum of money if he testified against me and corroborated their story. According to Rich, he refused to do it and then the threats came. First, they suggested they could have him imprisoned and then, seeing that their victim was still hesitant, the physical threats towards him and his family began. They advised him to stay out of it, and that's why he hadn't received any news from him for a long time.

Meanwhile, Chani and his accomplices had been weaving a plan to ensure I stayed locked up for a long time. The idea was to convince me to escape from prison and, when I arrived at the agreed place, the police would be waiting for me there. Apparently, that's what awaits me there: a lot of undercover police officers, from the surroundings to inside the library.

The story seems credible; even Rich's way of expressing himself is very convincing, but this plot is quite hard to believe. I've been planning the prison escape for months based on information, and I can't get my head around things being different. On the other hand, it's normal for Rich to want to deceive me; in theory, he's the one who benefits from me being in prison.

After thinking about it for less than a minute, I tell him I don't believe it and that I plan to get to the library. Although in reality doubts have already entered my mind and I'm extremely nervous. But, having come this far, I don't know what else to do. The only option I have is to continue with the plan and trust that everything will work out. The story they just told me makes a lot of sense, but if I were in his place, I would also try to deceive me.

Luckily for me, Rich had evidence to show me. The first time he contacted Chani and his girlfriend, he managed to get a recording. At one point during the meeting, he went to the bathroom and activated his phone's audio recording. At the beginning of the conversation, you can hear two people speaking in Korean, although at some point they start speaking in English with Rich. It's definitely Chani's voice; he starts by saying that I'm a criminal and that I must pay for my crimes. Then he advises him not to help me and ends with a threat, making it clear that if he helps me in any way, he'll end up explaining himself to justice.



Chani is smart, so at no point does he reveal his involvement and is very measured in his threats. This recording has opened my eyes, although I'm still not 100% convinced. It might have been part of the plan to get Rich to trust them and thus obtain incriminating evidence. Whatever the case, I'm more inclined to believe Rich's version. But I'm still doubting; I weigh my alternatives and mentally review all the conversations I've had with those involved since I entered prison.

As Rich sees that I'm still doubtful, he decides to show me one more piece of evidence. Something irrefutable. He asks me to follow him, although he suggests that I keep my distance. He carefully scans the horizon and stays close to the wall. He seems to be looking for something. He reaches a corner dumpster; from there you can see the library and the entire main street in front of it. I don't know what we're looking for, but almost without telling me anything, he points to a van. There are people inside, they seem to be watching. Then he shows me another van parked on the other side. It's clear they're police officers and they're waiting for my arrival.



At that moment, I start feeling an amount of anxiety that I can barely control. I think that if the police are waiting, it's because they already know I've escaped from prison and the whole plan has gone down the drain. The easiest thing would be to try to run away. I wish I knew where Bud had gone; maybe he could take me with him or at least help me leave the country.

I didn't know yet that Bud was still monitoring the communication reaching the prison. Inside it, the police still didn't know I had escaped, so they weren't looking for me. If I returned before they did the cell check, no one would have proof of my escape and no one would have reason to doubt me.

Suddenly I remembered the recording that Chani's cousin showed me to incriminate Rich, where he asked Chani to leave the country and never come back. I mentioned it to Rich, and he told me it was a phrase taken out of context. The conversation wasn't with Chani, but with an English friend from his same city. It's such an absurd excuse that it must be true.

I don't know what to do; I'm frozen. Luckily, Rich starts speaking in a low voice and tells me he has a plan. It seems he had planned it since Chani asked him to participate in this trap meeting. In reality, I barely have to do anything; everything will depend on him. Well, to be honest, if he asks me to do something, it's to hit him in the face. It surprises me, but he only had to ask twice. He's bleeding from his nose, he's wiped his shirt and even got part of his clothes muddy. Finally, he tells me to escape from there as soon as possible. My intention is to return to prison with Phil; even if they already know in jail that I'm not there, I can make up some story, like I fell asleep in the laundry room. I tell Rich my meeting point with Phil, and he simply says goodbye, telling me he'll give me a two-minute head start.

After those two minutes, Rich ran out to the main street desperately asking for help. It was then that all the police officers approached to see what was happening. He told them that someone had assaulted him and had run away. Of course, he indicated the opposite direction to the one I used to escape. At that moment, I was already reaching the parking lot, where I would wait for Phil crouched down.

Thanks to this strategy, the library was left unprotected. The police told Rich that he should stay there waiting for the ambulance, but that wasn't what he had in mind. Inside the library, they hadn't heard about what had happened, so Chani and his girlfriend still thought the meeting was on and that the authorities were out there waiting to arrest me.

He explained to them that he had met me at the entrance, that I had attacked him and then fled upon seeing the police presence. This made them think that at that moment I was about to be captured and that their plan had been a complete success. On the other hand, after the attack, they thought that from now on Rich was totally on their side.

They began to talk casually, and Rich explained to them in detail how he had been attacked by surprise. He knew he had to gain their trust before bringing up any incriminating topics. If they suspected anything, all would be lost. There's only one chance to get a confession from them.



Chani's girlfriend began saying that I was dangerous and that, thanks to this, they were going to sentence me for a long time. Chani thanked Rich for his help and they shook hands. Then the tone of the conversation changed and became more relaxed. For a few minutes, both parties were discussing their immediate future plans. Then, Rich took advantage of the situation.

He began complaining that, despite being an accomplice to the Korean couple's crime, he hadn't received any money, while they had gotten a good sum, enough to live well for a long time. These comments were followed by a long silence. They were quite distrustful and didn't want to corroborate Rich's words.

Once the tension passed, they continued by saying they hadn't received any money and that they were simply happy to have helped the authorities arrest a criminal. Things were going badly and it was time to make the final attempt. Seeing that the couple was very secretive, he decided to try a new tactic: making them angry. He was sure that at some point they would explode. They had already shown aggressive behavior towards him in the past, and he knew it could happen again.

He began playing his last card, threatening to talk if he didn't receive money. The important thing was for them to think that I didn't matter to him at all and that all he wanted was money. This way, they wouldn't suspect anything. At this point, no one could blame him if he claimed a share of the loot. Besides, it was fair, since he was risking a lot being an accomplice and had even helped them set this trap for me.

Regarding my unfair incrimination, Chani, extremely cautious, neither confirmed nor denied Rich's words. He simply said that the important thing was that everyone was physically well, which sounded like a subtle threat. He emphasized that all he wanted for everyone was that they could continue being friends and go on with their lives, trying to forget what had happened.

Although Chani remained cautious, Rich tried one last time. This time, directly, he told them that if he didn't receive fair compensation, he would confess to the police that very night and even testify against them in court. That would exculpate me and put both of them in prison. It was then that Chani's girlfriend lost control and became disproportionately furious. She began shouting, asserting that he would never receive any of the money they had stolen and that if he didn't want to suffer an "accident," he'd better keep his mouth shut.

They had taken the bait, and Rich already had what he wanted. Now, to avoid problems, he needed to reduce the tension and leave there with the recorded evidence. He simply said that he wasn't stupid enough to confess, as that could also bring problems for him. Finally, he added that they would be in contact because he wasn't giving up his share of the money and that they would have time to discuss it more calmly. Chani, calmer now, accepted. He shook his hand and they said goodbye. Rich left there with his objective accomplished.

Meanwhile, I was crouched in the parking lot, waiting for Phil. I had no way to communicate with him, only the hope that he would appear at the agreed time. With the blackmail we had done to him, I had no doubt he would comply. Five minutes before the agreed time, he arrived and turned off the car lights. For at least two minutes, I observed the area looking for any suspicious movement. Seeing that everything was quiet, I cautiously approached the car. Phil got out and put me in the trunk. It wouldn't be a comfortable journey and, to be honest, it was quite claustrophobic, but it was the only way to return to prison without being detected.



The parking lot for prison staff is underground and from there you can access several rooms. Before escaping, Bud showed me a route back to my cell. The door should still be open and, once there, I'll be able to close it from the inside. It was the perfect plan, although I'm worried that the police might have alerted the prison guards about my possible escape. At this point, they've probably already checked my cell and discovered I wasn't there. Even so, I trust the plan. No one has seen me outside these walls and they can't prove that I've escaped. If I manage to return without being detected, I'll be able to continue with our strategy.

Returning to my humble quarters was quite simple. I closed the cell from the inside and now I'm just waiting for the morning inspection. I don't see much commotion, which means they're not looking for anyone; a very good sign. When the guard arrived, his expression completely changed.

For a moment, I was scared, but then I remembered that Bud had escaped and they didn't know it yet.

Quickly, the guard raised the alarm and the search began. It was then that I realized that Bud, from outside, had helped me by blocking communications between the police and the prison. The plan had worked perfectly.

Of course, I was interrogated about the disappearance of my escape companions, especially because I shared a cell with one of them. I simply said that I was asleep and that, when I woke up, I found myself alone in the cell. It was an unlikely explanation, but impossible to refute.

Rich, for his part, handed the audio to my lawyer and together they started working on my defense. Thanks to the new evidence and Rich's testimony, it didn't take long for me to regain my freedom. Finally, I could prove my innocence. Now I have a whole new life ahead of me and, additionally, I've learned to Program in Java.

Chani and his girlfriend weren't so lucky. They were found guilty of extortion and sentenced to several years in prison. The affected companies recovered their money, and everything was finally cleared up.

Thank you for having accompanied me on this arduous journey. Together we've learned to Program in Java, and I hope it's as useful for you as it has been for me.

