Unit 2: Basic Java

2.1 Introduction to Java: Features of Java

Java is a high-level, general-purpose programming language developed by Sun Microsystems (now owned by Oracle Corporation) in the mid-1990s. Java was created by a team of developers led by James Gosling at Sun Microsystems in the early 1990s. It was designed with the goal of creating a language that is platform-independent, secure, and robust, making it suitable for a wide range of applications, from small desktop programs to large-scale enterprise systems.

Feature of Java

1. Simple and Easy to Learn
2. Object-Oriented
3. Portable (Platform Independent)
4. Secure
5. Robust
6. Architecture Neutral
7. Interpreted
8. High Performance
9. Multithreaded
10. Distributed
11. Dynamic

2.2 The Java Virtual Machine (JVM), parts of Java program, Naming Conventions in Java

JVM

The JVM is a software program that acts as an intermediary between Java code and the underlying computer system. It's like a virtual machine running inside your actual computer. Here's how it works:

- Compilation: When you write Java code, you compile it using a Java compiler. This translation process turns your code written in Java (.java files) into bytecode (.class files).
- Bytecode: Bytecode is a special set of instructions that the JVM understands. It's not specific to any particular operating system (OS) like Windows or macOS.
- JVM's Role: The JVM is present on various operating systems. It interprets the bytecode instructions in the .class files and executes them on the specific system. This allows Java programs to run on different platforms, achieving the famous "write once, run anywhere" functionality of Java.

Parts of Java program

- Documentation Section
- Package Declaration
- Import Statements
- Class Definition
- Class Variables and Variables
- Main Method Class
- Methods and Behaviors

Documentation Section

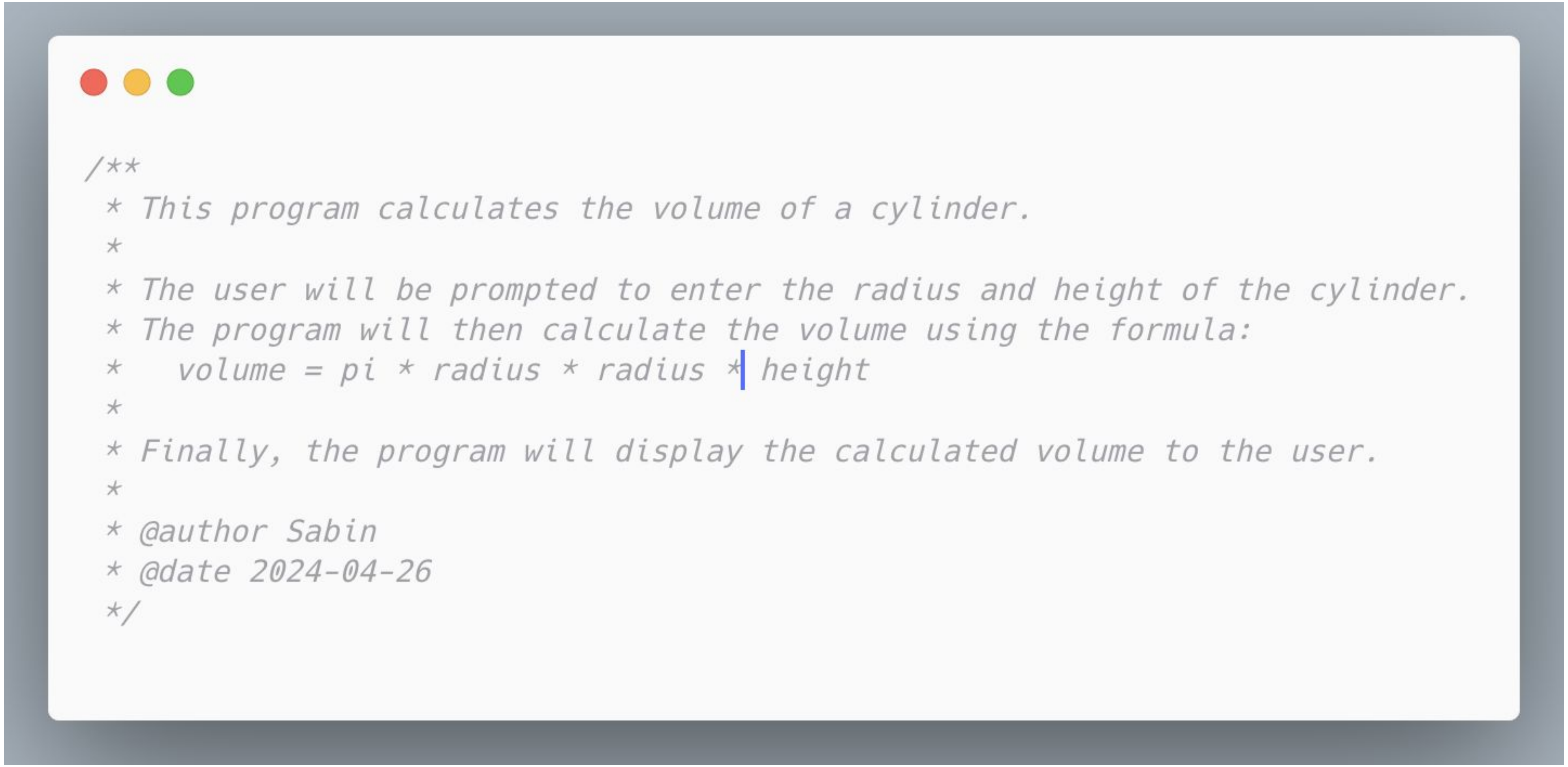
This section consists of comments that provide information about your program. It's not mandatory, but highly recommended for readability and maintainability. Comments are lines of text ignored by the compiler but included in the source code for humans to understand. Documentation might include:

1. A brief description of the program's purpose.

2. The author's name or team.

3. The date the program was created or last modified.

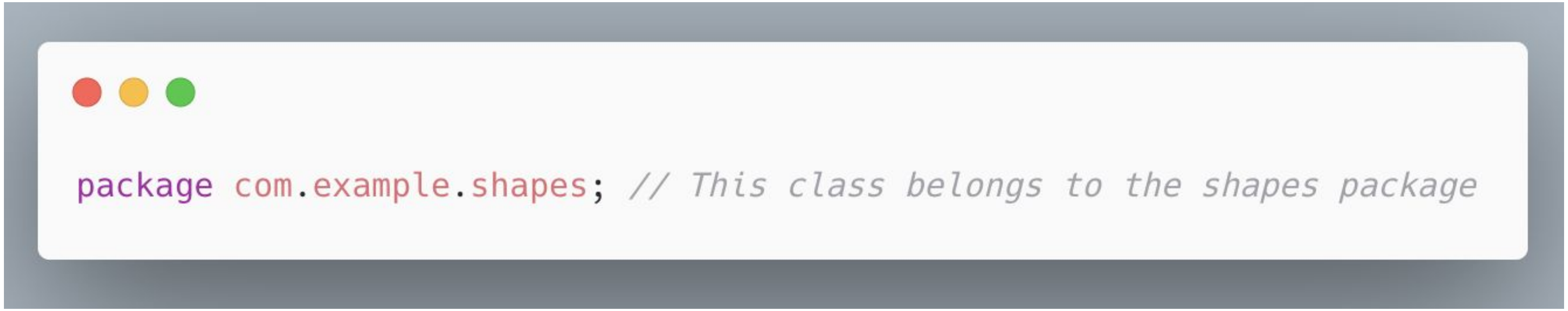
4. Any specific assumptions or requirements for running the program.



```
/**
 * This program calculates the volume of a cylinder.
 *
 * The user will be prompted to enter the radius and height of the cylinder.
 * The program will then calculate the volume using the formula:
 *   volume = pi * radius * radius * height
 *
 * Finally, the program will display the calculated volume to the user.
 *
 * @author Sabin
 * @date 2024-04-26
 */
```

Package Declaration

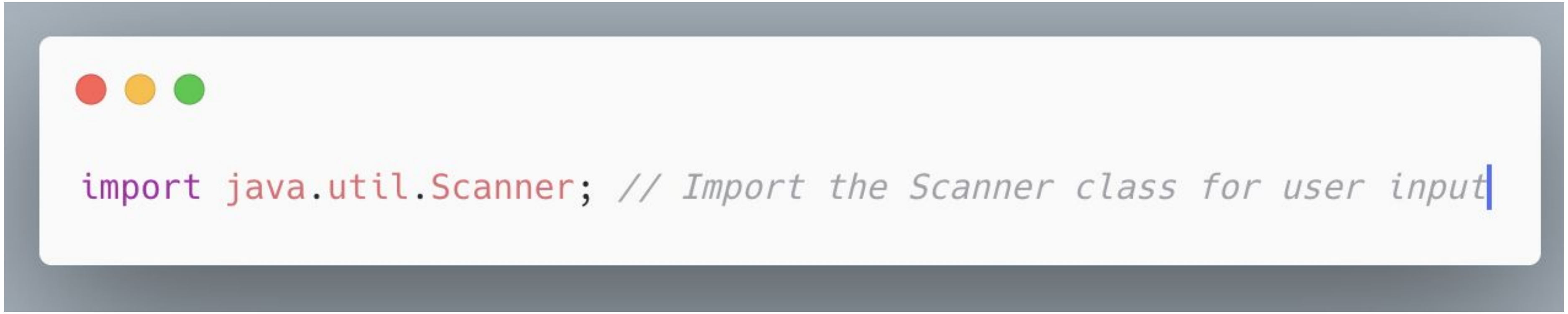
Java programs can be organized into logical groups using packages. A package declaration specifies the package to which a class belongs. This helps prevent naming conflicts between classes from different parts of your code or external libraries. Packages follow a hierarchical naming convention separated by dots (e.g., com.example.myapp).



```java
package com.example.shapes; // This class belongs to the shapes package
```

Import Statement

If your program needs to use functionalities defined in classes from other packages, you'll need to import them using import statements. This allows you to use the class names directly without specifying the entire package path.



```java
import java.util.Scanner; // Import the Scanner class for user input
```

Class Definition

A class acts as a blueprint for creating objects. It defines the properties (variables) and functionalities (methods) that objects of that class will have. Here's what a class definition typically includes:

Access Modifier: Controls the visibility of the class (e.g., public allows access from anywhere).

Class Name: Should start with an uppercase letter and follow camelCase conventions (e.g., Rectangle, Circle).

Class Body: Contains declarations for variables (including class and instance variables) and methods.

Variables

1. Class Variables (Static Variables): Declared within the class but outside any method. They are shared by all instances of the class and typically use the static keyword. (e.g., public static final double PI = 3.14159;).

2. Instance Variables (Object Variables): Declared within a class but inside methods (often constructors). They belong to individual object instances and represent the unique data each object holds. (e.g., private double width; and private double height;).

3. Local Variables: Local variables are declared in methods, constructors, or blocks. Local variables are created when the method, constructor, or block is entered and the variable will be destroyed once it exits the method, constructor, or block.

Main Method Class

The main method is the program's entry point. When you execute a Java program, the JVM starts by looking for the main method and begins execution from there. The main method typically has the following signature:

“public static void main(String[] args) { ... }”

public: Allows the method to be accessed from anywhere.

static: Indicates the method belongs to the class itself, not a specific object.

void: The method doesn't return any value.

String[] args: An optional argument array that can be used to pass command-line arguments to the program when it's run.

Methods and Behaviors

Methods define specific actions or functionalities that objects can perform. They operate on the object's data (using instance variables) and often return results. Methods are like tools that objects can use. Here are some key aspects of methods:

Access Modifier: Controls the visibility of the method (e.g., public for access from anywhere in the program).

Return Type: The data type of the value the method returns (e.g., void for no return value, int for an integer value).

Method Name: Should be descriptive and follow camelCase conventions (e.g., calculateArea(), displayMessage()).

Method Parameters: Optional arguments that can be passed to the method when it's called.

Method Body: The code that defines the method's logic and functionality.

Naming Conventions in Java

In Java, naming conventions are guidelines for naming classes, interfaces, methods, variables, and constants. These conventions help improve code readability, maintainability, and collaboration among developers.

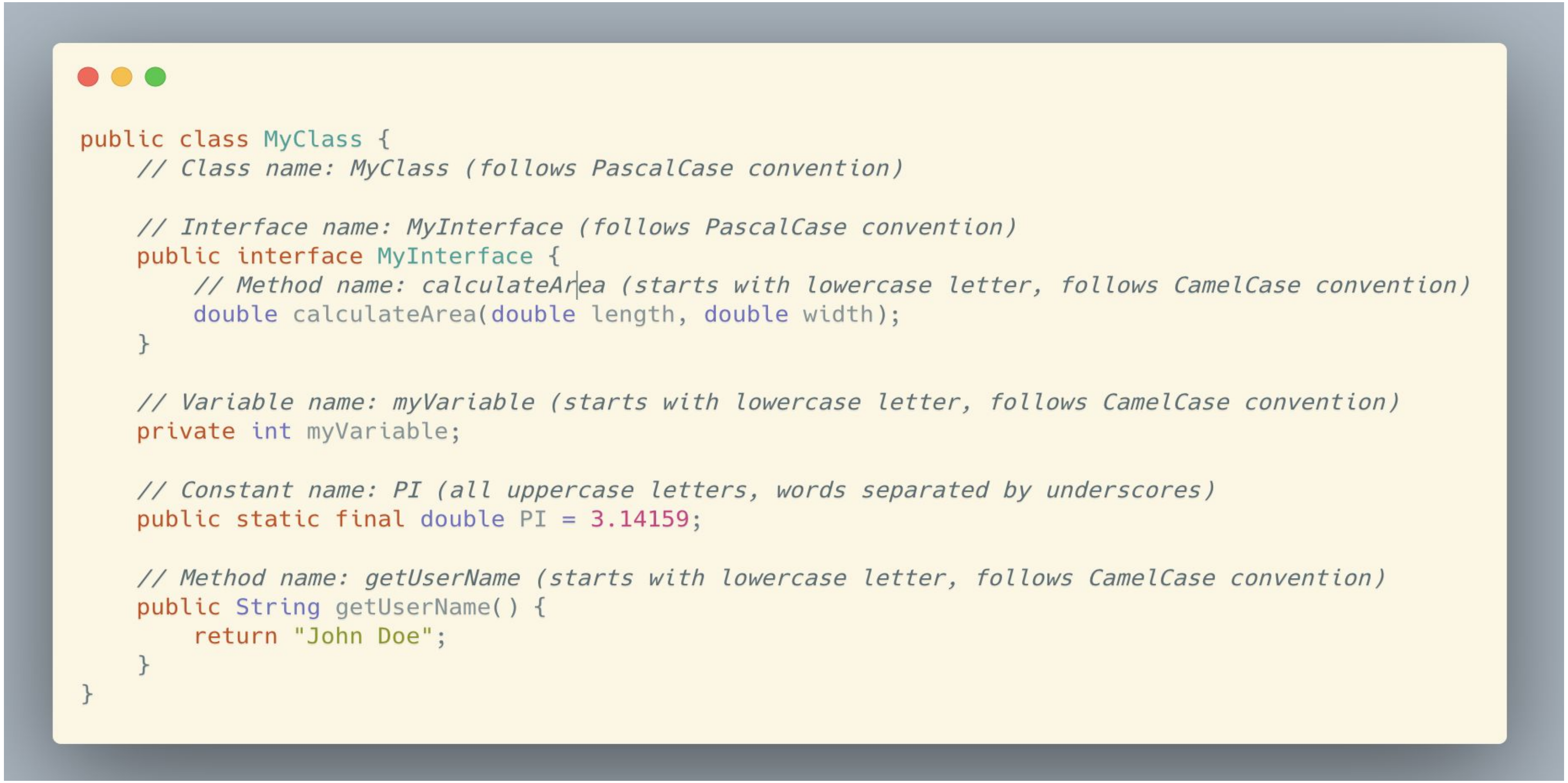
Here are some key naming conventions in Java:

1. Classes: Class names should follow the PascalCase convention. For example, MyClass, Person, Student.
2. Interfaces: Interface names should follow the PascalCase convention. They often include the word "Interface" at the end. For example, MyInterface, DrawableInterface.

3. Methods: Method names should start with a lowercase letter and follow the CamelCase convention. They should be verbs or verb phrases. For example, calculateArea, getUserName.

4. Variables: Variable names should start with a lowercase letter and follow the CamelCase convention. They should be nouns or noun phrases. For example, myVariable, userName.

5. Constants: Constant names should be in all uppercase letters with words separated by underscores. For example, PI, MAX_VALUE.


```java
public class MyClass {
    // Class name: MyClass (follows PascalCase convention)

    // Interface name: MyInterface (follows PascalCase convention)
    public interface MyInterface {
        // Method name: calculateArea (starts with lowercase letter, follows CamelCase convention)
        double calculateArea(double length, double width);
    }

    // Variable name: myVariable (starts with lowercase letter, follows CamelCase convention)
    private int myVariable;

    // Constant name: PI (all uppercase letters, words separated by underscores)
    public static final double PI = 3.14159;

    // Method name: getUserName (starts with lowercase letter, follows CamelCase convention)
    public String getUserName() {
        return "John Doe";
    }
}
```

Variables and constants

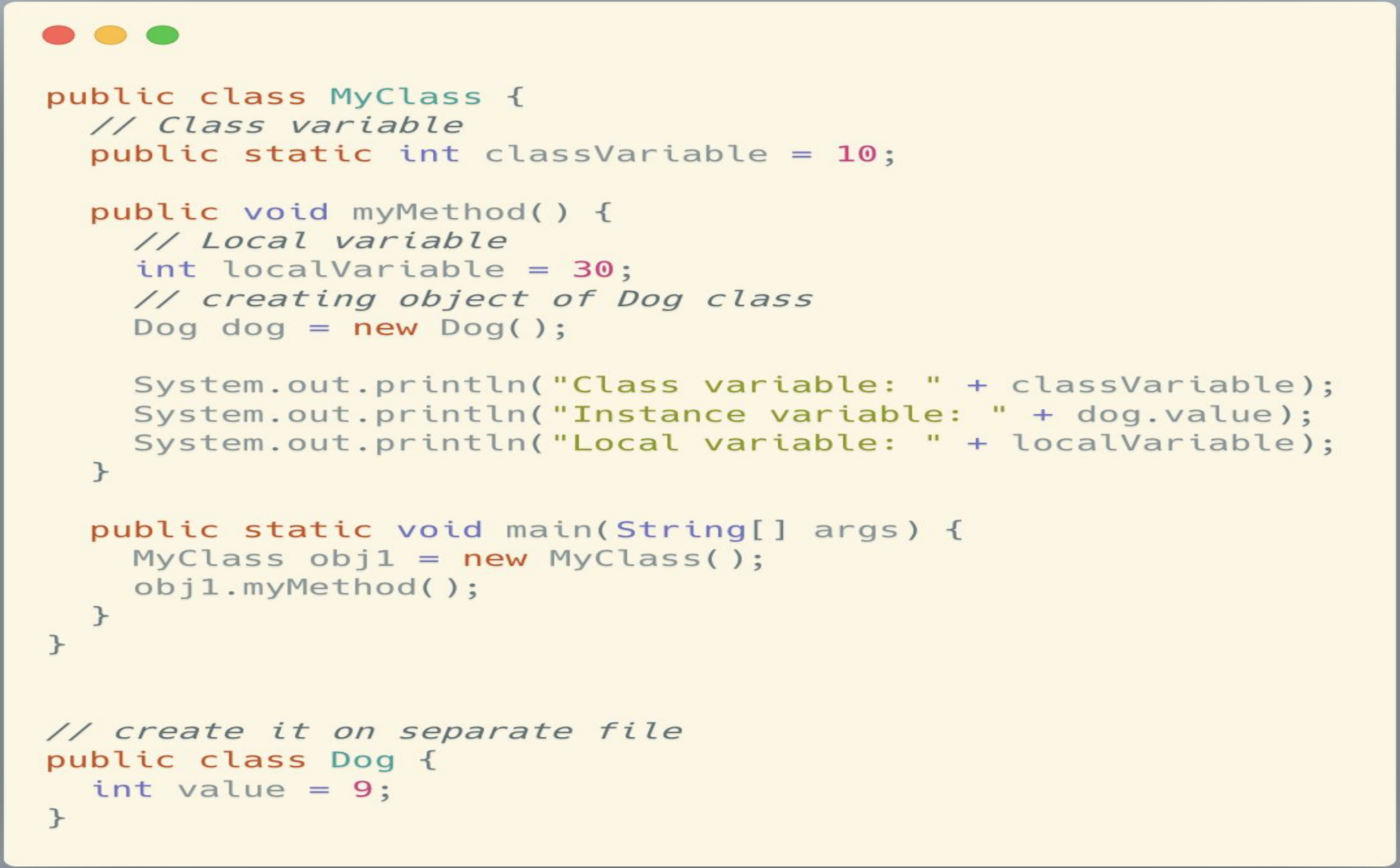
- Variables are used to store information to be referenced and manipulated in a computer program.
- They also provide a way of labeling data with a descriptive name, so our programs can be understood more clearly by the reader and ourselves.
- It is helpful to think of variables as containers that hold information.
- Their sole purpose is to label and store data in memory.
- This data can then be used throughout your program.
- A variable is a value that can change, depending on conditions or on information passed to the program.

Types of variables

1. Class Variables (Static Variables): Class variables, also known as static variables, are variables that are declared with the static keyword in a class, but outside a method, constructor, or block. There would only be one copy of each class variable per class, regardless of how many objects are created from it. Class variables are usually used for constants.

2. Instance Variables (Object Variables): Instance variables are variables declared in a class, but outside a method, constructor, or block. There is a separate copy of an instance variable in each object of the class.

3. Local Variables:Local variables are variables declared in a method, constructor, or block. They are created when the method, constructor, or block is entered, and the variable will be destroyed once it exits the method, constructor, or block.



```java
public class MyClass {
  // Class variable
  public static int classVariable = 10;

  public void myMethod() {
    // Local variable
    int localVariable = 30;
    // creating object of Dog class
    Dog dog = new Dog();

    System.out.println("Class variable: " + classVariable);
    System.out.println("Instance variable: " + dog.value);
    System.out.println("Local variable: " + localVariable);
  }

  public static void main(String[] args) {
    MyClass obj1 = new MyClass();
    obj1.myMethod();
  }
}


// create it on separate file
public class Dog {
  int value = 9;
}
```

Constants

Constant is a value that cannot be changed after assigning it.

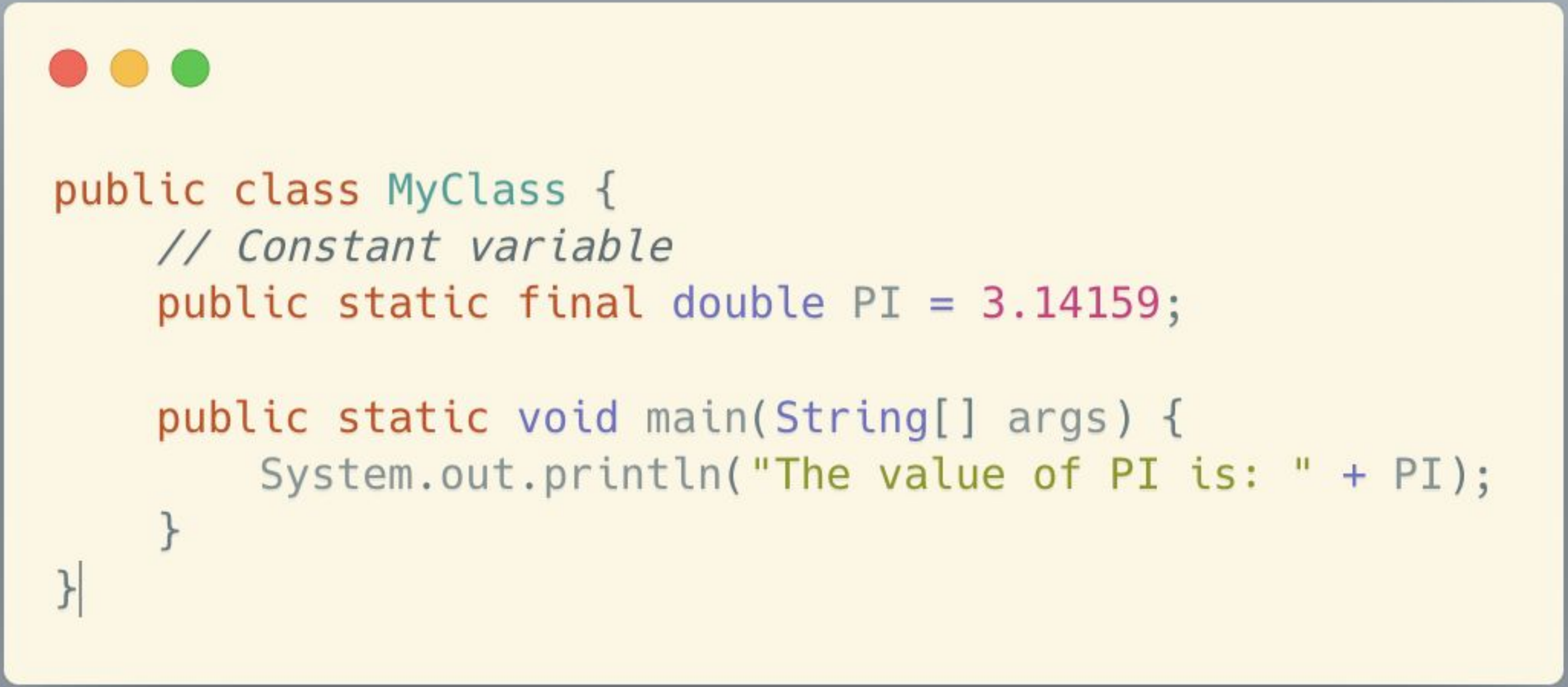
Java does not directly support the constants.

There is an alternative way to define the constants in Java by using the non-access modifiers static and final.

According to the Java naming convention the identifier name must be in capital letters.

Syntax

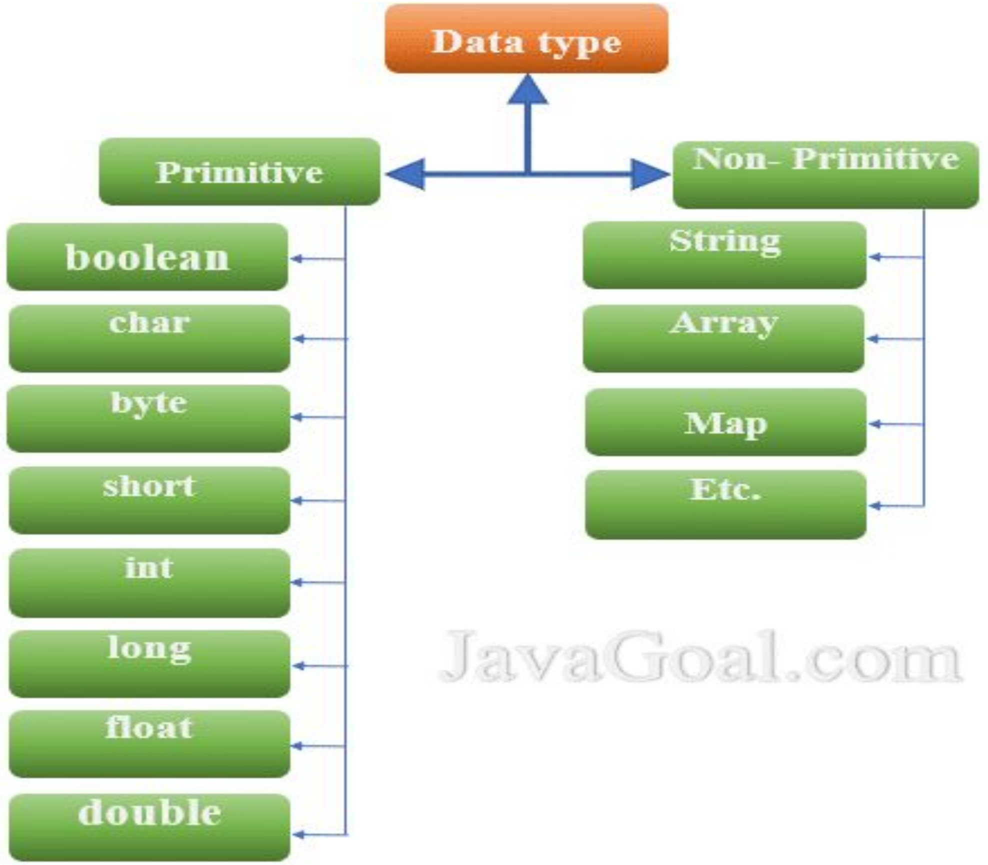
```
static final datatype IDENTIFIER_NAME = value;
```



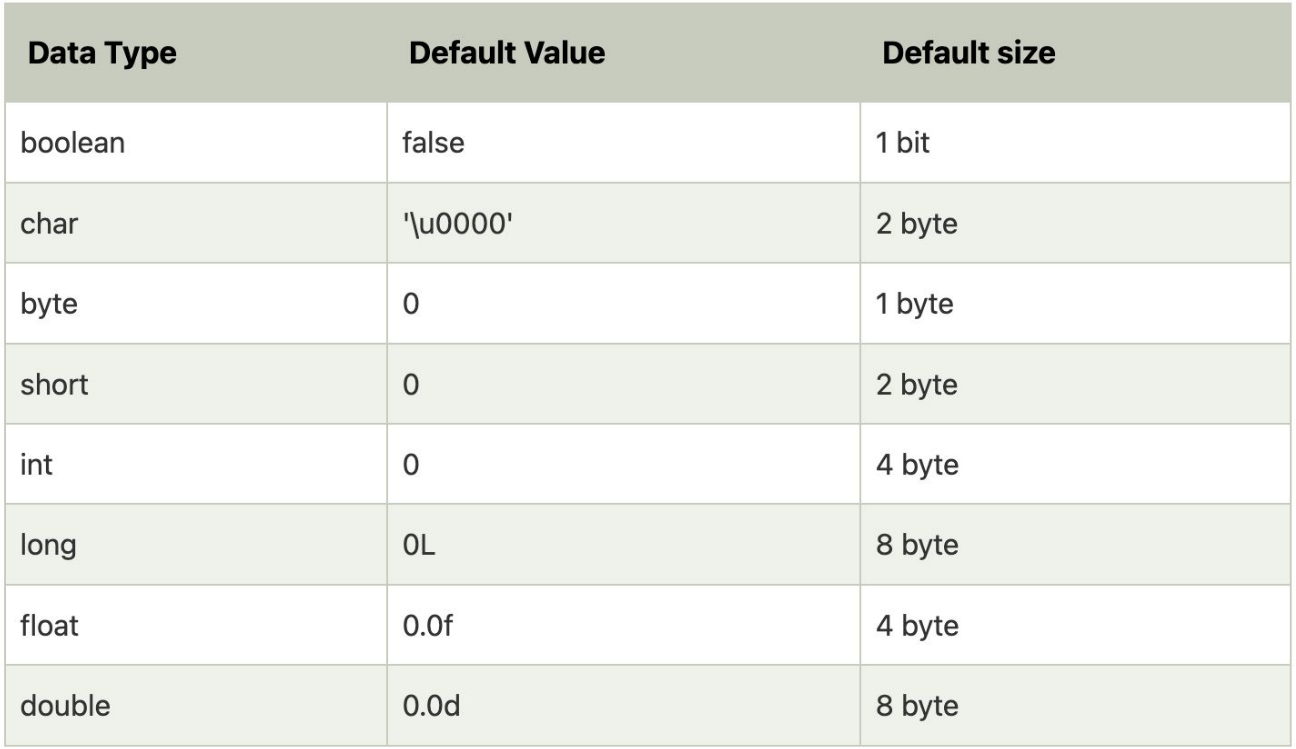
```java
public class MyClass {
    // Constant variable
    public static final double PI = 3.14159;

    public static void main(String[] args) {
        System.out.println("The value of PI is: " + PI);
    }
}
```

Data Types in Java

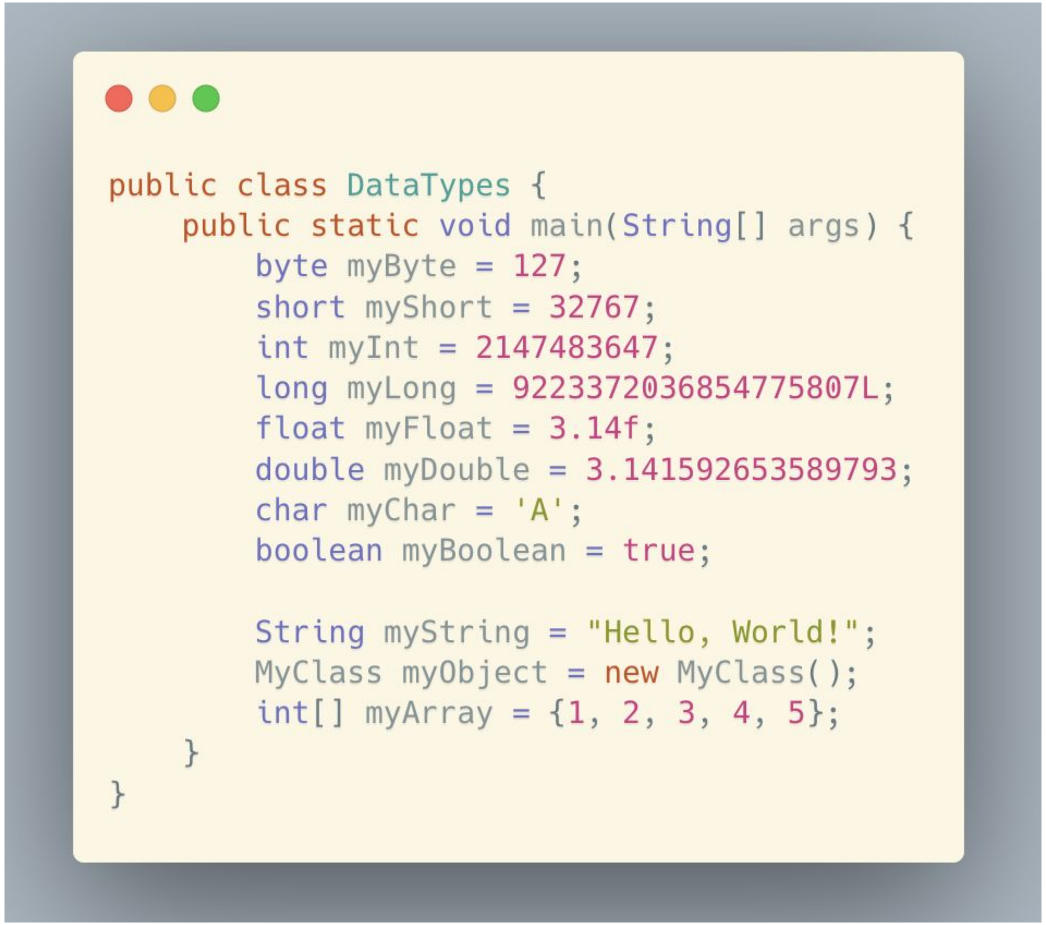
1. Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float, and double.
2. Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.


Data type
Primitive
Non- Primitive
boolean
char
byte
short
int
long
float
double
String
Array
Map
Etc.

Primitive data types



Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

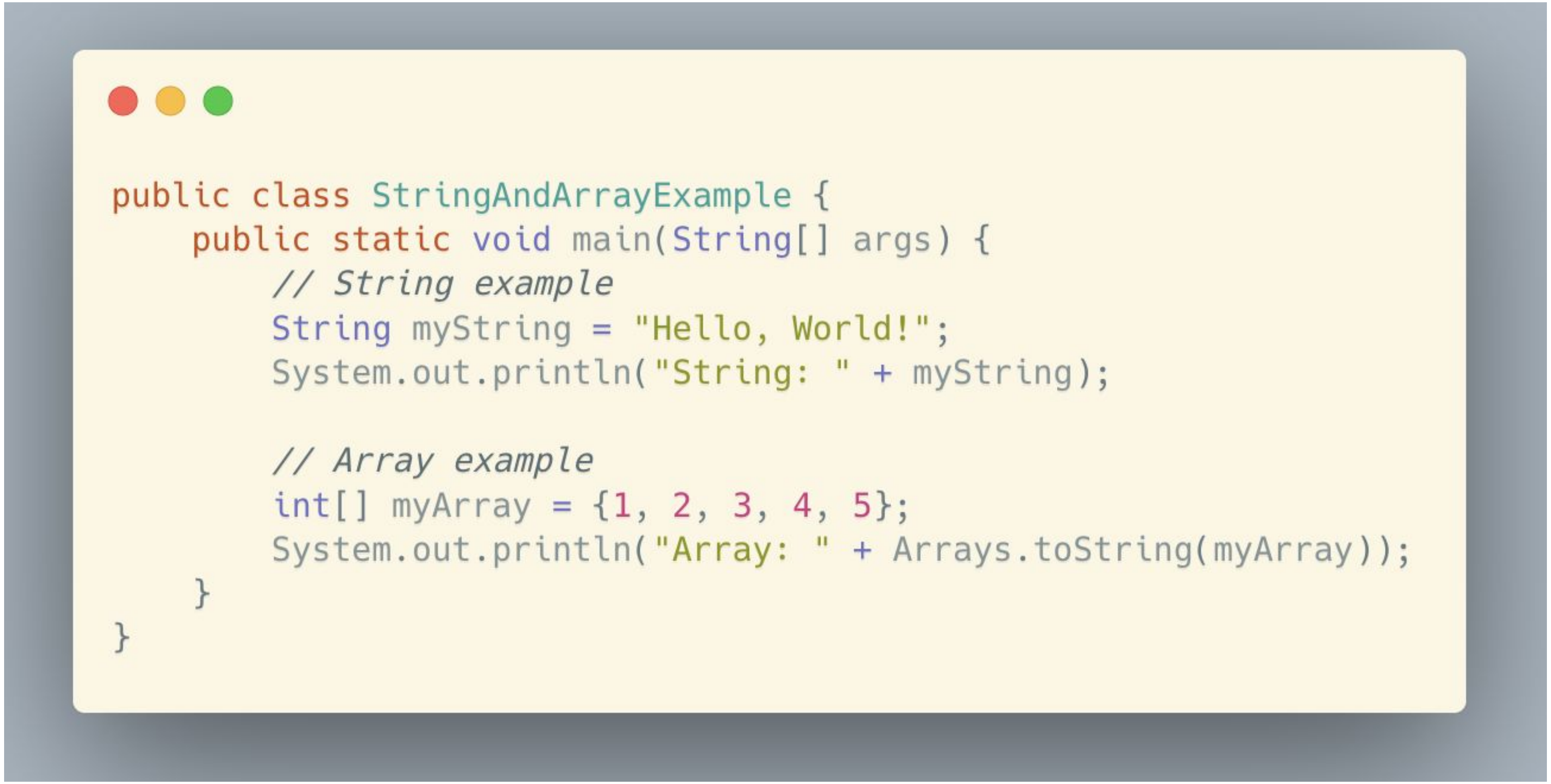


```java
public class DataTypes {
    public static void main(String[] args) {
        byte myByte = 127;
        short myShort = 32767;
        int myInt = 2147483647;
        long myLong = 9223372036854775807L;
        float myFloat = 3.14f;
        double myDouble = 3.141592653589793;
        char myChar = 'A';
        boolean myBoolean = true;

        String myString = "Hello, World!";
        MyClass myObject = new MyClass();
        int[] myArray = {1, 2, 3, 4, 5};
    }
}
```

Non-primitive data types

String: A String is a sequence of characters. In Java, strings are objects that are instances of the String class. Strings are immutable, which means that once a string is created, it cannot be changed. If you need to modify a string, you'll need to create a new string.

Array: An array is a collection of similar data types. In Java, arrays are objects that are instances of the array class. Arrays have a fixed size, which means you cannot add or remove elements once the array is created.



```java
public class StringAndArrayExample {
    public static void main(String[] args) {
        // String example
        String myString = "Hello, World!";
        System.out.println("String: " + myString);

        // Array example
        int[] myArray = {1, 2, 3, 4, 5};
        System.out.println("Array: " + Arrays.toString(myArray));
    }
}
```

Operators in Java

In Java, there are many types of operators which are used to perform various operations on variables and values. These operators are classified based on their functionality and the number of operands they require. Here's a brief explanation of each type of operator in Java:

1. Unary Operator: Unary operators require only one operand. They are used to perform various operations, such as incrementing/decrementing a value by one, negating an expression, or inverting the value of a boolean. Examples of unary operators include ++, --, +, -, and !.

2. Arithmetic Operator: Arithmetic operators are used to perform common mathematical operations, such as addition, subtraction, multiplication, division, and modulus. Examples of arithmetic operators include +, -, *, /, and %.

3. Shift Operator: Shift operators are used to shift the bits of a value to the left or right. They are often used in low-level programming, such as working with binary data or bit manipulation. Examples of shift operators include <<, >>, and >>>.

4. Relational Operator: Relational operators are used to compare two values and return a boolean result. They are often used in conditional statements, such as if statements and loops. Examples of relational operators include <, >, <=, >=, ==, and !=.

5. Bitwise Operator: Bitwise operators are used to perform operations on the individual bits of a value. They are often used in low-level programming, such as working with binary data or bit manipulation. Examples of bitwise operators include &, |, ^, and ~.

6. Logical Operator: Logical operators are used to combine multiple boolean expressions and return a boolean result. They are often used in conditional statements, such as if statements and loops. Examples of logical operators include &&, ||, and !.

7. Ternary Operator: The ternary operator is a shorthand way of writing an if-else statement. It takes three operands and returns one of two values, depending on the result of a boolean expression. The syntax of the ternary operator is: condition ? value1 : value2.

8. Assignment Operator: Assignment operators are used to assign a value to a variable. The most common assignment operator is =, which assigns the value on the right to the variable on the left. There are also compound assignment operators, such as +=, -=, *=, /=, and %=, which perform an operation and assignment in one step.

Operator Type	Category	Precedence
Unary	postfix	`expr++ expr--`
	prefix	`++expr --expr +expr -expr ~ !`
Arithmetic	multiplicative	`* / %`
	additive	`+ -`
Shift	shift	`<< >> >>>`
Relational	comparison	`< > <= >= instanceof`
	equality	`== !=`
Bitwise	bitwise AND	`&`
	bitwise exclusive OR	`^`
	bitwise inclusive OR	`|`
Logical	logical AND	`&&`
	logical OR	`||`
Ternary	ternary	`? :`
Assignment	assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

References: https://www.youtube.com/watch?v=pv1C0_6k78A

https://www.youtube.com/watch?v=pv1C0_6k78A

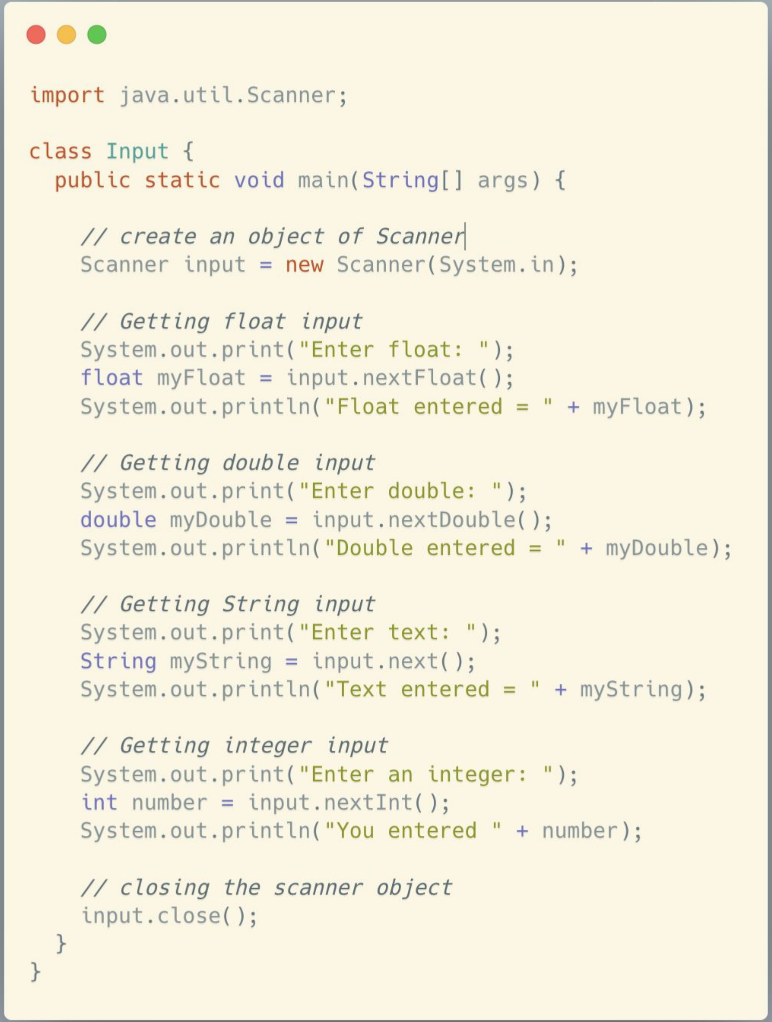
Reading and displaying Output

Reading (Taking user input)

Java provides different ways to get input from the user. However, in this tutorial, you will learn to get input from the user using the object of Scanner class.

In order to use the object of Scanner, we need to import java.util.Scanner package.

Then, we need to create an object of the Scanner class. We can use the object to take input from the user.

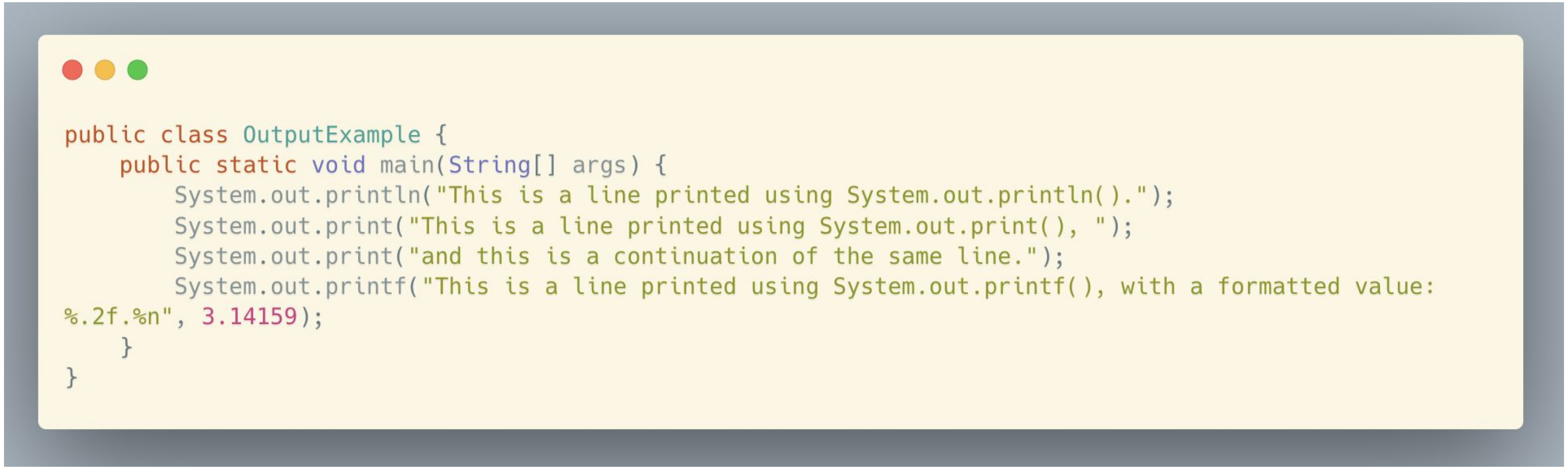


```java
import java.util.Scanner;

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

    // create an object of Scanner
    Scanner input = new Scanner(System.in);

    // Getting float input
    System.out.print("Enter float: ");
    float myFloat = input.nextFloat();
    System.out.println("Float entered = " + myFloat);

    // Getting double input
    System.out.print("Enter double: ");
    double myDouble = input.nextDouble();
    System.out.println("Double entered = " + myDouble);

    // Getting String input
    System.out.print("Enter text: ");
    String myString = input.next();
    System.out.println("Text entered = " + myString);

    // Getting integer input
    System.out.print("Enter an integer: ");
    int number = input.nextInt();
    System.out.println("You entered " + number);

    // closing the scanner object
    input.close();
  }
}
```

Displaying Output

In Java, System.out.println(), System.out.print(), and System.out.printf() are methods used to display output to the console.

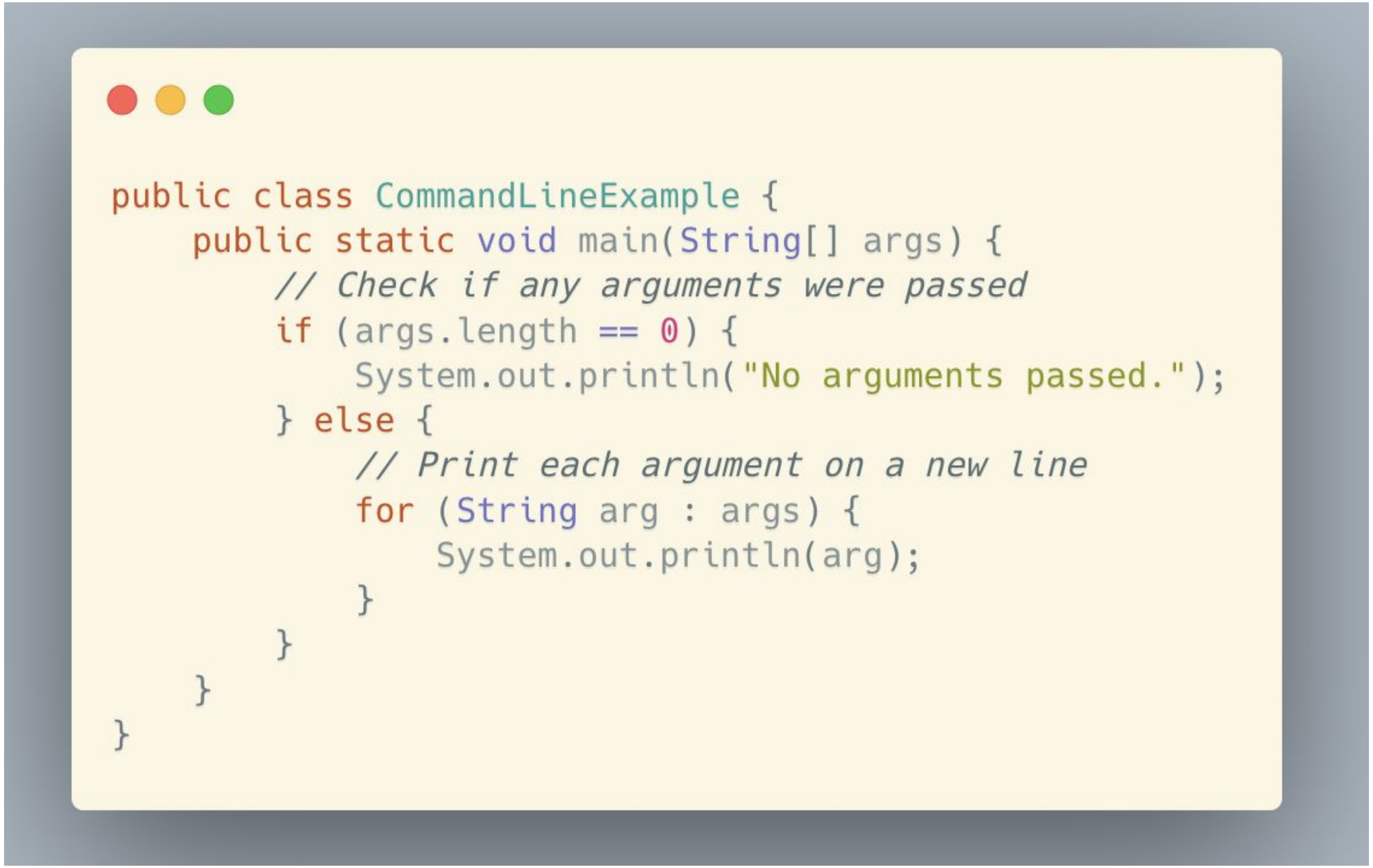
1. System.out.println(): This method prints the specified text to the console and moves the cursor to the next line. It's useful when you want to print each output on a new line.
2. System.out.print(): This method prints the specified text to the console without moving the cursor to the next line. It's useful when you want to print multiple outputs on the same line.
3. System.out.printf(): This method prints the specified text to the console using a format string. It allows you to format the output using placeholders and format specifiers.


```java
public class OutputExample {
    public static void main(String[] args) {
        System.out.println("This is a line printed using System.out.println().");
        System.out.print("This is a line printed using System.out.print(), ");
        System.out.print("and this is a continuation of the same line.");
        System.out.printf("This is a line printed using System.out.printf(), with a formatted value:
%.2f.%n", 3.14159);
    }
}
```

Command Line Arguments

In Java, command-line arguments are the arguments that are passed to the main method when a Java program is executed from the command line. These arguments can be used to provide input to the program or to control its behavior.

References: https://www.youtube.com/watch?v=Up17-azeuyE



```java
public class CommandLineExample {
    public static void main(String[] args) {
        // Check if any arguments were passed
        if (args.length == 0) {
            System.out.println("No arguments passed.");
        } else {
            // Print each argument on a new line
            for (String arg : args) {
                System.out.println(arg);
            }
        }
    }
}
```

In this example, the main method takes a String array called args as an argument. This array contains the command-line arguments passed to the program.

To run this program with command-line arguments, you can use the following command:

```
java CommandLineExample arg1 arg2 arg3
```

Control Statements

In Java, control statements are used to control the flow of execution of a program based on certain conditions. These statements are used to cause the flow of execution to advance and branch based on changes to the state of a program.

Java provides three types of control statements: decision-making statements, looping statements, and branching statements.

Decision-making statements:

1. if statement: The if statement is used to execute a block of code if a specified condition is true.
2. if-else statement: The if-else statement is used to execute a block of code if a specified condition is true, and another block of code if the condition is false.
3. switch statement: The switch statement is used to execute different blocks of code based on different conditions.

Looping statements:

1. for loop: The for loop is used to execute a block of code repeatedly, as long as a specified condition is true.
2. while loop: The while loop is used to execute a block of code repeatedly, as long as a specified condition is true.
3. do-while loop: The do-while loop is used to execute a block of code at least once, and then repeatedly, as long as a specified condition is true.

Branching statements:

1. break statement: The break statement is used to terminate the loop or switch statement and transfer the execution to the next statement immediately following the loop or switch.
2. continue statement: The continue statement is used to skip the current iteration of a loop and continue with the next iteration.

Assignment

WAP to demonstrate example of all the control structure with explanation.

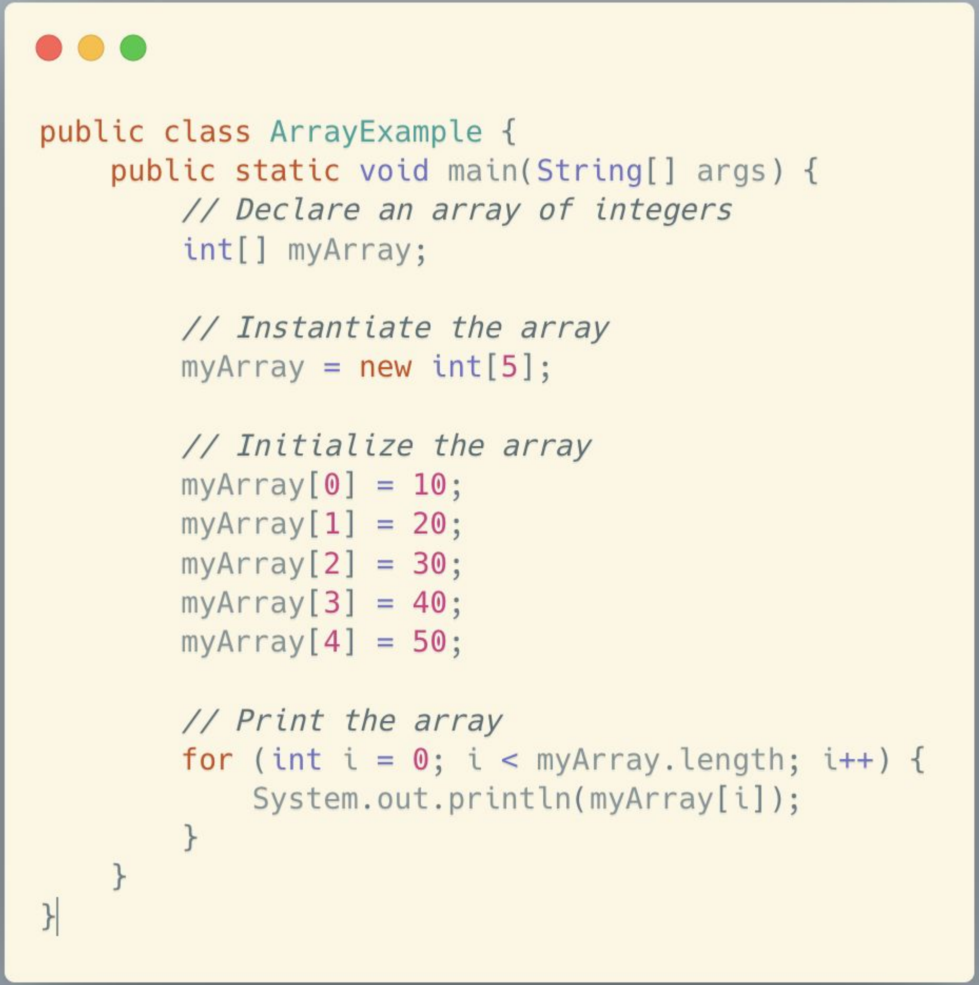
Array

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator.

Used to store multiple values in a single variable, instead of declaring separate variables for each value.



```java
public class ArrayExample {
    public static void main(String[] args) {
        // Declare an array of integers
        int[] myArray;

        // Instantiate the array
        myArray = new int[5];

        // Initialize the array
        myArray[0] = 10;
        myArray[1] = 20;
        myArray[2] = 30;
        myArray[3] = 40;
        myArray[4] = 50;

        // Print the array
        for (int i = 0; i < myArray.length; i++) {
            System.out.println(myArray[i]);
        }
    }
}
```

In this example, an array of integers called myArray is declared, instantiated, and initialized. The for loop is used to print each element of the array.

WAP to find Average of Array Elements

int[] numbers = {2, -9, 0, 5, 12, -25, 22, 9, 8, 12, -100, 234};

Assignment: Multidimensional Arrays

https://www.programiz.com/java-programming/multidimensional-array

Practical

1. Write a program to define a structure of a basic Java program.
2. Write a Java program to calculate the area of a rectangle. Prompt the user to enter the length and width of the rectangle as input.
3. Write a Java program to check whether a given number is odd or even. Prompt the user to enter the number.
4. Write a Java program to check whether a given number is prime or not.
5. Implement a Java program to reverse a given string. Get string as argument.
6. Develop a Java program to find the sum of digits of a given number.
7. Create a Java program to display the Fibonacci series up to a certain limit.
8. Implement a Java program to convert temperature from Celsius to Fahrenheit and vice versa. Prompt the user to enter the temperature and the scale (Celsius or Fahrenheit).
9. Develop a Java program to find the largest and smallest elements in an array.
10. Write a Java program to calculate the factorial of a number without using recursion.
11. Create a Java program to find the factorial of a given number using a recursive function.

References:

- Book
- https://www.javatpoint.com/java-tutorial
- https://www.geeksforgeeks.org/java/
- https://www.programiz.com/java-programming