**Object Oriented Programming Using Java**

**Module-1: OOPS CONCEPTS AND JAVA PROGRAMMING:**

**OOP concepts:** Classes and objects, data abstraction, encapsulation, inheritance, benefits of inheritance, polymorphism, procedural and object-oriented programming paradigm.

**Java programming:** History of java, comments data types, variables, constants, scope and life time of variables, operators, operator hierarchy, expressions, type conversion and casting, enumerated types, control flow statements, jump statements simple java stand alone programs, arrays, console input and output, formatting output, constructors methods, parameter passing, static fields and methods, access control, this reference, overloading methods and constructors, recursion, garbage collection, exploring string class

**Module-2: MULTIPLE INHERITANCE**

**Inheritance:** Inheritance hierarchies, super and subclasses, member access rules, super keyword,

**Preventing inheritance:** final classes and methods, the object class and its methods.

**Polymorphism:** dynamic binding, method overriding abstract classes and methods.

**Module-3: INTERFACES AND PACKAGES**

**Interface:** Interfaces VS Abstract classes, defining an interface, implement interfaces accessing implementations through interface references, extending interface.

**Packages:** Defining creating and accessing a package, understanding CLASSPATH, importing packages

**Module-4: EXCEPTION HANDLING**

**Exception Handling:** Benefits of exception handling, the classification of exceptions exception hierarchy, checked exceptions and unchecked exceptions, usage of try, catch, throw, throws and finally. rethrowing exceptions, exception specification, built in exceptions, creating own exception sub classes.

**Module-5: GUI PROGRAMMING AND APPLETS**

**GUI Programming with Java:** The AWT class hierarchy, introduction to swing. swangs Vs AWT, hierarchy for swing components.

**Containers:** JFrame. JApplet, JDialog. Jpanel, overview of some swing components JButton, JLabel, JTextField, JTextArea, simple applications Layout management Layout manager types, border, grid and flow.

Applets: Inheritance hierarchy for applets, differences between applets and applications, life cycle of an applet, passing parameters to applets.

**What is Programming?:** Writing a Program

**What is Programming Language?:** Is a Collection of **Tokens**.

**What is a Token:**

A Token is a **smallest individual units** of a Program.

In Java, tokens are the smallest **elements** of a program that are **meaningful to the compiler**. Java programs are made up of these tokens.

**Java Tokens:**

1. **Keywords:** Predefined Words like, class, if, switch etc
2. **Identifiers:** Name given to the programming elements such as class, variables and so on.
3. **Literals:** Constant values that are assigned to variables
4. **Operators:** Symbols that perform operations on variables and values
5. **Separators** (or Delimiters): Characters that separate tokens in the code. Ex: {}, [], (), and so on
6. **Comments:** Text within the code that is not executed but provides explanations or annotations.

**Example:**

A screenshot of a computer program

Description automatically generated

**What is Java:**

Java is a High Level, General Purpose, Object Oriented, Platform Independent, Compiled and Interpreted, Statically Typed, Highly Secured programming language.

**Java is Developed By**: **James Gosling**, **Patrick Naughton**, and **Mike Sheridan**

**History of Java:**

**Early Beginnings**

**1991: Project Initiation**

Java began as a project called "**Oak**" by Sun Microsystems, led by James Gosling, Mike Sheridan, and Patrick Naughton. They were part of a team at **Sun Microsystems** that initiated the project in the early 1990s. The goal was to develop a language for **embedded systems in consumer electronics,** , and it eventually evolved into the Java programming language we know today.

**1992: Green Project**

The team, known as the Green Team, worked on creating a platform-independent language. They aimed to create a language that could run on various devices, including **televisions**, **toasters**, and other consumer electronics.

**1994: Transition to the Web**

The team realized the potential of their new language for the burgeoning **World Wide Web**, where the ability to run the same program on different platforms was highly desirable. They renamed **Oak** to **Java**, inspired by **Java coffee**, which was consumed in large quantities by the developers.

**Official Launch**

**1995: Public Introduction**

Java 1.0 was officially released by **Sun Microsystems**. The "Write Once, Run Anywhere" (**WORA**) capability became a significant selling point, allowing Java programs to run on any device with a Java Virtual Machine (**JVM**).

**Evolution and Growth**

**1996: Java Development Kit (JDK) 1.0**

Sun Microsystems released JDK 1.0, providing developers with the **tools needed to develop Java applications**.

**1997-1999: Rapid Evolution**

Java 1.1 was released in 1997, introducing new features like inner classes and JavaBeans.

In 1998, Java 2 (formerly known as JDK 1.2) introduced major enhancements to the platform, including the Swing **graphical API** and **the Collections framework.**

**Java Community Process**

**1999: Establishment of the Java Community Process (JCP)**

The JCP was established to allow for the participation of the **broader Java community** in the development and evolution of Java standards and specifications.

**Open Source and Later Versions**

**2006: Open Sourcing of Java**

Sun Microsystems announced that Java would be released under the GNU General Public License (GPL), making it open-source and allowing the community to **contribute** to its development.

**2009: Acquisition by Oracle**

**Oracle Corporation acquired Sun Microsystems**, becoming the steward of Java. This led to concerns about the future direction of Java, but Oracle continued to develop and support the language.

**2011-Present: Ongoing Evolution**

Java 7 was released in 2011, introducing new language features and performance improvements.

Java 8, released in 2014, brought significant changes, including the introduction of **lambda expressions**, the Stream API, and the new date and time API.

Subsequent versions (Java 9 through Java 20) have continued to add new features, enhance performance, and improve the language's capabilities.

**Java 22**, officially released on **March 19, 2024**

**Flavours of Java:**

**1. Java Standard Edition (Java SE)**

**Purpose**: Java SE provides the core functionality for **general-purpose programming.** It includes the Java Development Kit (JDK), which contains the Java Runtime Environment (JRE), a compiler (javac), and various development tools.

**2. Java Enterprise Edition (Java EE), now Jakarta EE (Web Applications)**

**Purpose**: Java EE is designed for building large-scale, distributed, and component-based applications in the enterprise environment.

**3. Java Micro Edition (Java ME) (Android)**

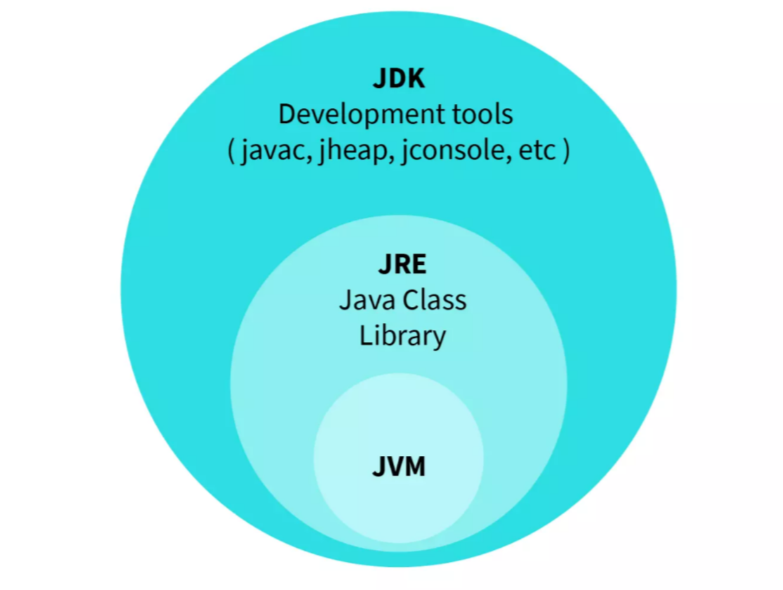
**Purpose**: Java ME is tailored for resource-constrained devices like embedded systems, mobile phones, and Internet of Things (IoT) devices.

**Features of Java:**

1. **Object-Oriented:** Java follows an object-oriented programming model, which allows for the creation of modular, reusable code. It supports features like inheritance, polymorphism, encapsulation, and abstraction.
2. **Platform-Independent**: Java's "write once, run anywhere" capability is facilitated by the Java Virtual Machine (JVM). Java code is compiled into bytecode, which can run on any system equipped with a JVM, making Java applications highly portable.
3. **Secure**: Java provides a secure environment through its runtime environment.
4. **High Performance**: Although Java is an interpreted language, the use of Just-In-Time (JIT) compilers helps improve its performance by converting bytecode into native machine code at runtime.
5. **Multi-Threaded:** A thread is the smallest unit of a process that can be scheduled and executed independently by the operating system. It is a **lightweight process** that shares the same memory space and resources of the parent process but can execute code concurrently.
6. **Compiled & Interpreted:** Java has both Compiler and Interpreter.

**JDK (Java Development Kit)**

The Java Development Kit (JDK) is a **software development environment** used for developing Java applications and applets. It provides the necessary **tools**, **libraries**, and **resources** to **create** and **execute** Java programs.

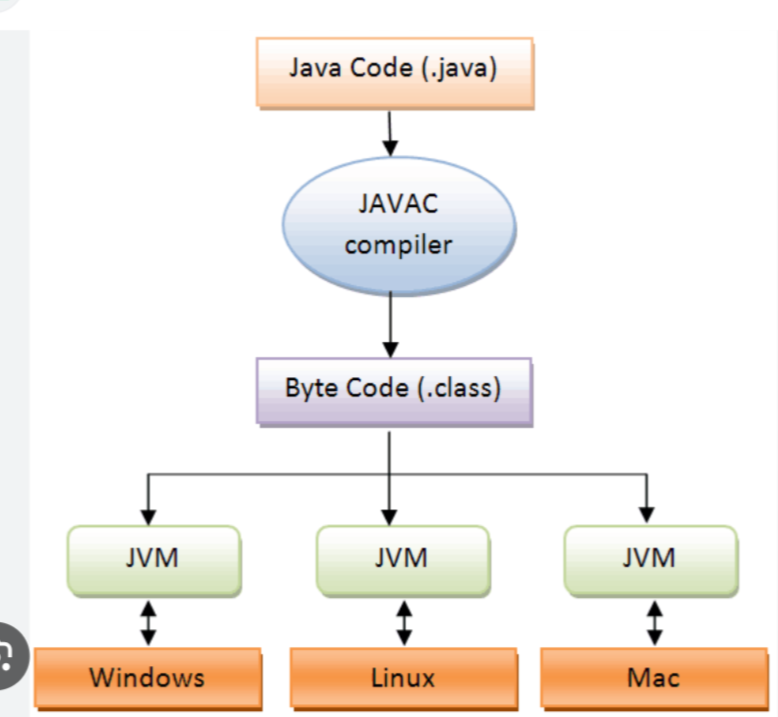


**JRE (Java Runtime Environment)**

The Java Runtime Environment (JRE) is a **software package** that provides the necessary **libraries**, **Java Virtual Machine** (JVM), and other components to **run applications** written in the Java programming language. It is a part of the Java Development Kit (JDK) but can also be installed separately. The JRE is designed to provide an environment for executing Java applications, not for developing them.

**Java Virtual Machine (JVM) :**

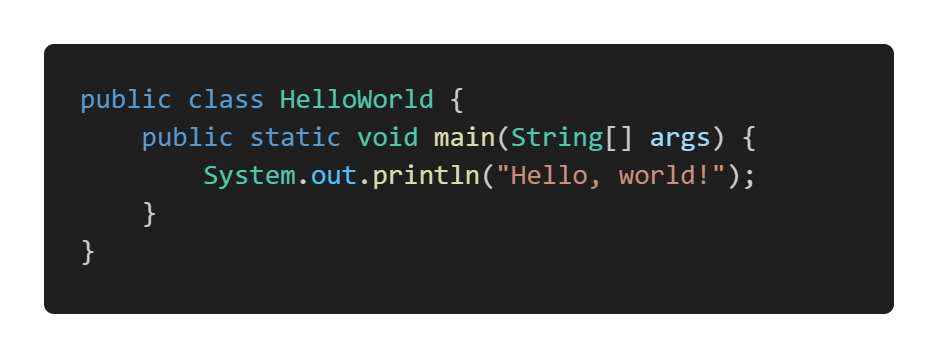
The JVM is the **core component** of the **JRE** that **executes Java bytecode**. It provides a **platform-independent** execution environment, meaning Java applications can run on any device or operating system that has a compatible JVM.



**How to Run the Java Program:**

**1. Write Your Java Program:**

**Example: HelloWorld**.**java**



**2. Compile the Java Program using javac:A black rectangle with blue text

Description automatically generated**

**3. Run the Compiled Java Program using java:**

A black rectangle with blue text

Description automatically generated

**Comments:**

In Java, comments are annotations in the source code that are ignored by the compiler and the runtime environment. They are intended to make the code easier to understand for humans by explaining the purpose, logic, or structure of the code.

**There are 3 types of comments in Java:**

1. **Single-Line Comments:** These comments start with **//** and continue to the end of the line.

A black screen with green text

Description automatically generated

1. **Multi-Line Comments:** These comments start with **/\*** and end with **\*/**. They can span multiple lines.

A screenshot of a computer

Description automatically generated

1. **Documentation Comments:** These comments start with **/\*\*** and end with **\*/**. They are used to generate documentation using the **Javadoc** tool.

**Javadoc** is a tool provided by the Java Development Kit (JDK) that generates **HTML** **documentation** from Java source code with special comments called documentation comments.

**Run Javadoc: javadoc** -d doc <source-files>

**Example:**

A screenshot of a computer program

Description automatically generated

Run:

**javadoc** -d doc Document.java

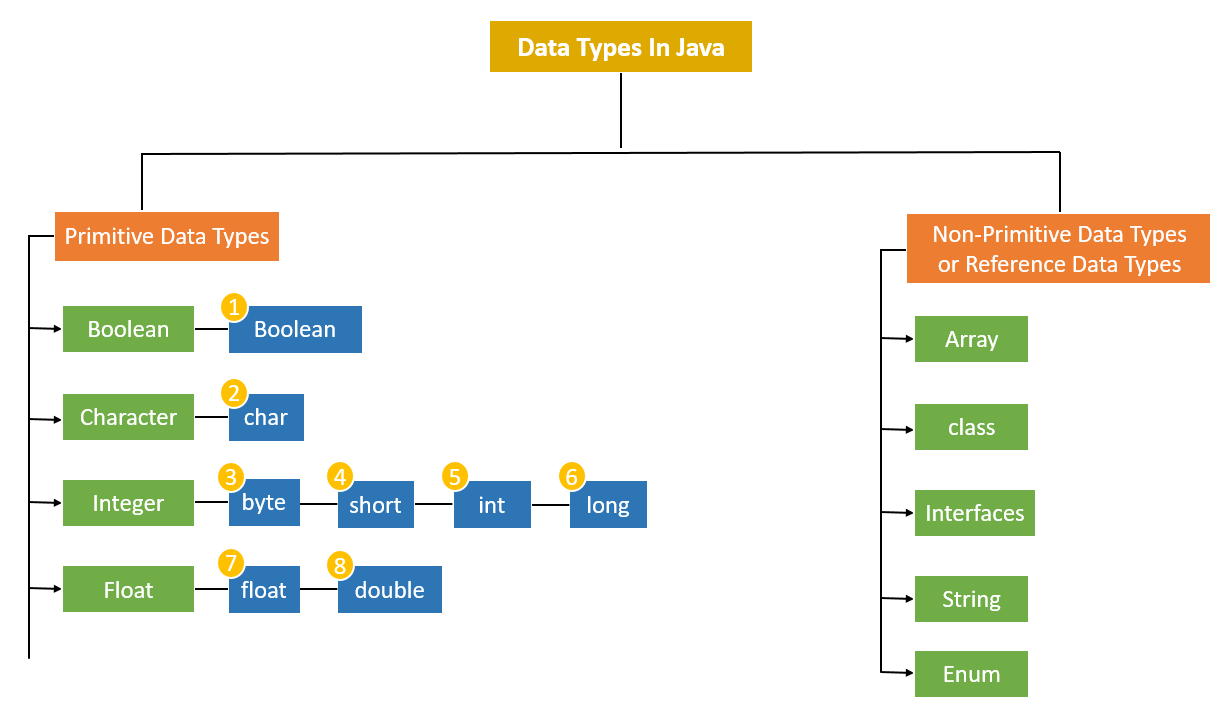
**Evolution of Data Storing Technique:**

1. **Variables** (Single Value)
2. **Array** (Multi-Value but Homogeneous)
3. **Structure & Union** (No OOP’s and No Methods)
4. **Class**

**Data Types:**

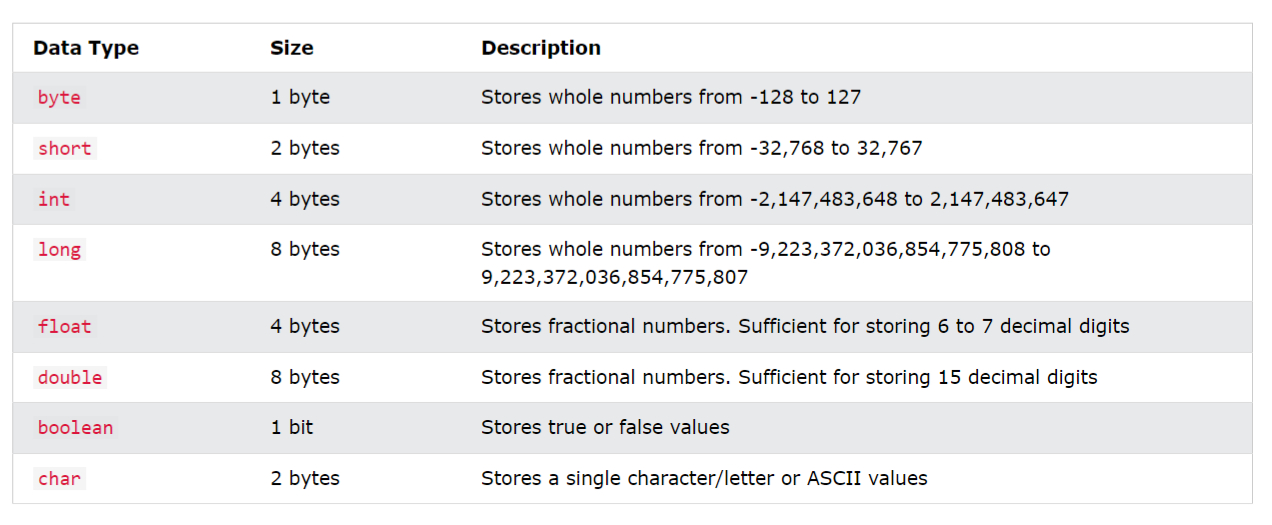
Data types in Java specify the different sizes and values that can be stored in a variable. They determine what kind of data can be held in a variable, such as integers, floating-point numbers, characters, or boolean values. Each data type requires different amounts of memory and has specific operations that can be performed on it.

**Categories of Data Types in Java:**



**Primitive Types:**

Java has **8** primitive data types, which are the most basic data types available within the language. They are **predefined by the language** and **named by a reserved keyword**.



**Non-Primitive** or **Reference Types**:

In Java, non-primitive or reference types are more **complex** **data** **types** that refer to objects and arrays. Unlike primitive types, which store actual values, reference types **store references to memory locations** where the **objects are stored.**

**How Variables of type Primitive and Non-Primitive Types are Stored in the Memory:**

A diagram of a stack and heart

Description automatically generated

1. **String:**

In Java, a String is an **object** (of String class) that represents a **sequence of characters**. Strings are widely used in Java programming and are a crucial part of the Java language. The String class is **immutable**, **meaning that once a String object is created**, its **value cannot be changed.**

**Key Features of Strings in Java**

1. **Immutable:** The content of a String object cannot be changed after it is created. Any modification of a string results in the creation of a new String object. So we can reassign new sequence of character to the String Object but we cannot modify the content or char of the String directly by using index.
2. **String Pool:** Java uses a special memory region called the string pool to store string literals. When a new string literal is created, Java checks the string pool first. If the string already exists in the pool, a reference to the existing string is returned, otherwise, the new string is added to the pool.
3. **String Class:** The String class is part of the **java.lang** package and is **implicitly imported.**

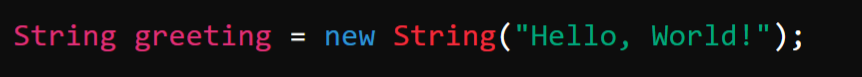
**Creating Strings:**

1. **Using String Literals**:

A black background with green and white text

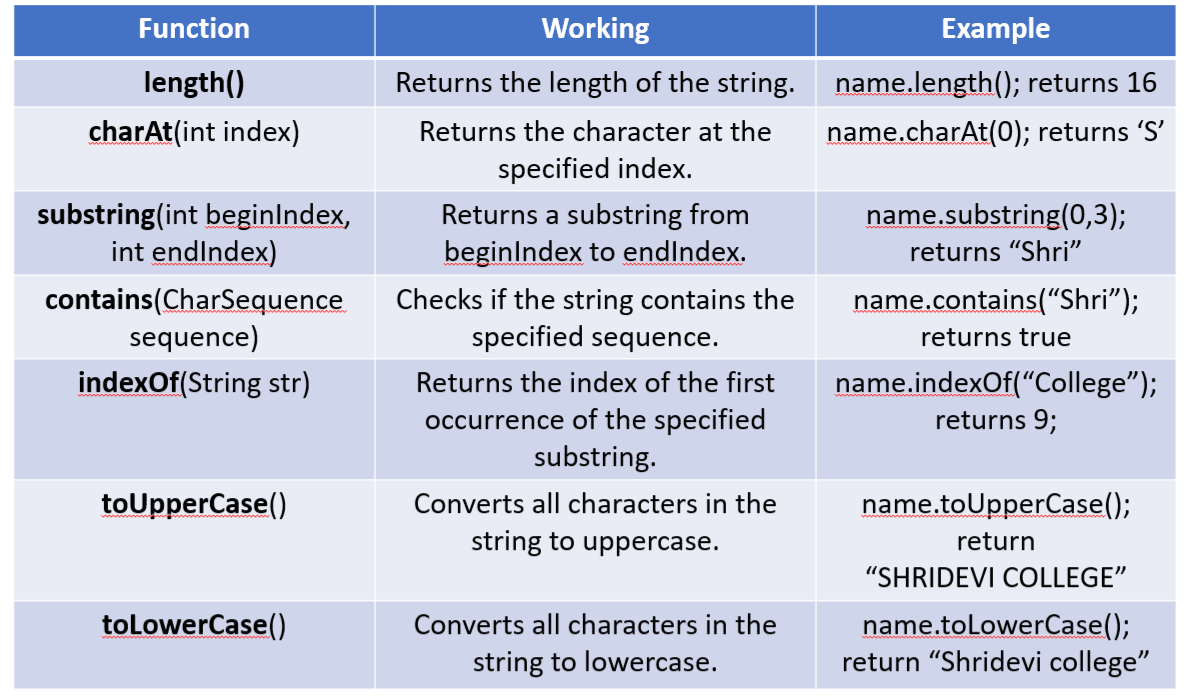
Description automatically generated

1. **Using the new Keyword:**



**Commonly Used Methods:**

**Example:** String name = “Shridevi College”



1. **Array:**

An array in Java is a **data structure** that allows you to store multiple values of the **same type** in a single variable. Arrays are useful for managing collections of data that can be accessed using an index.

**Key Features of Arrays in Java:**

1. **Fixed Size:** Once an array is created, its size cannot be changed. You must specify the number of elements it will hold when you create it.
2. **Indexed Access:** Each element in an array can be accessed using its index, with the first element at index 0 and the last element at index **length-1**.
3. **Homogeneous Elements:** All elements in an array must be of the same type, such as int, String, boolean, etc.
4. **Zero-Based Indexing:** The index of the first element is 0, and the index of the last element is **length-1.**

**Declaring and Initializing Arrays:**

**Declaring-**> **Syntax**:

A black background with white text

Description automatically generated

**Example**:

A black background with white text

Description automatically generated

**Initializing**-> **Syntax**:

A black background with blue and red text

Description automatically generated

**Example**:

A black background with white text

Description automatically generated

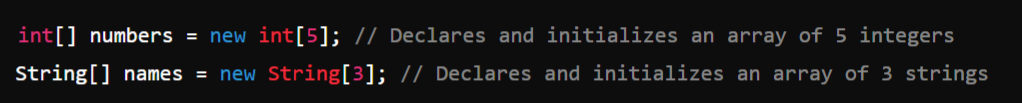
**We can combine declaration and initialization in one step**:

**Syntax:**

**A black background with white text

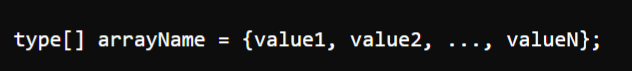
Description automatically generated**

**Example:**

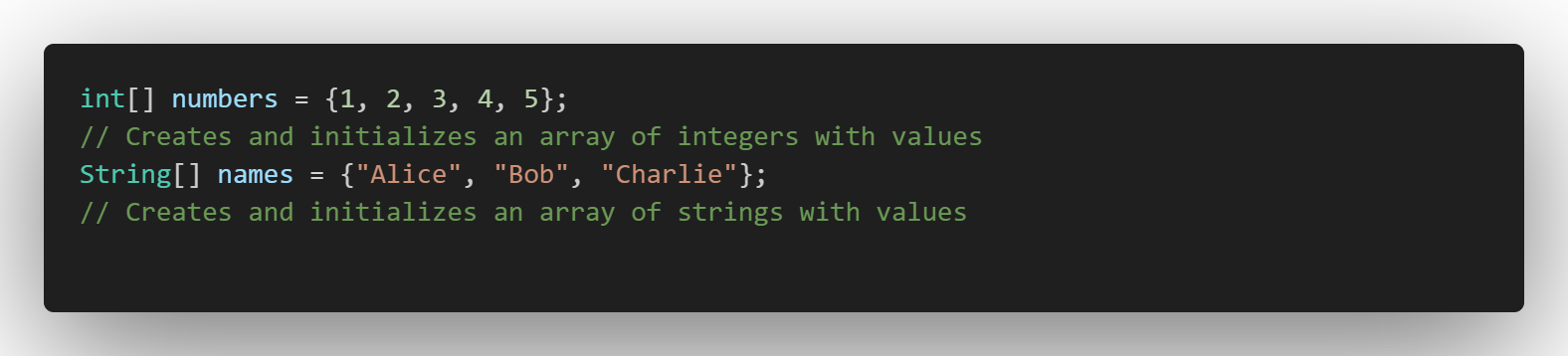


**Directly Assign the Values and the size will be detected by the compiler:**

**Syntax:**



**Example:**

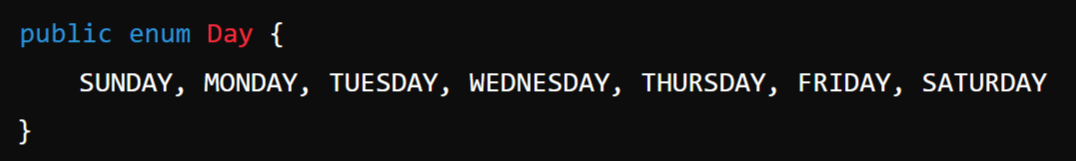


1. **Enum: (Creating our Own Type)**

In Java, an **enum** (short for "**enumeration**") is a special data type that enables a variable to be a set of predefined constants. It is used to represent a **fixed set of related constants more efficiently** and **readably**.

**Definition:** Enums are defined using the **enum** **keyword**. Each value in an enum is called an **enum constant**.

**Example:**

****

Note:

In Enum we are creating **our own data type** which can hold only the **constant** **specified** to it while defining it which is called **enum constants**.

**Example**:

A computer screen with text on it

Description automatically generated

**Example**: We can have **Functions** and **Constructor** for our Enum

A screen shot of a computer program

Description automatically generated

1. **Class:**

A class is a blueprint for creating objects. It defines a datatype by bundling data and methods that work on the data into one single unit. Classes are fundamental to the object-oriented programming paradigm in Java.

Defining a Class:

A class is defined using the **class keyword** followed by the class name and a body enclosed in curly braces. It typically **includes** **fields** (variables) and **methods** to define the behavior and state of the objects created from the class.

**Example:**



In the example above:

* Car is a class with fields color, model, and year.
* It has a constructor to initialize these fields.
* The displayDetails() method prints out the details of the car.
* The main method creates an instance of Car and calls the displayDetails method.

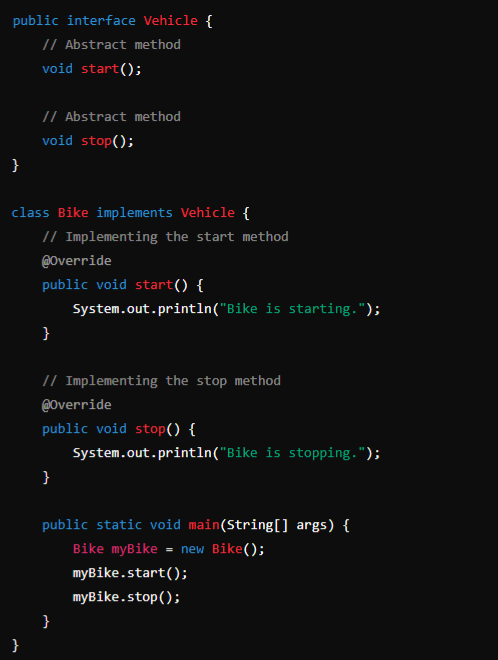
1. **Interface:**

An interface in Java is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Interfaces cannot contain instance fields or constructors. Interfaces provide a way to achieve abstraction and multiple inheritance in Java.

**Example of an Interface:**

In the example below:

* Vehicle is an interface with two abstract methods: start and stop.
* Bike is a class that implements the Vehicle interface, providing concrete implementations for the start and stop methods.
* The main method creates an instance of Bike and calls its start and stop methods.



**Control Flow Statements:**

Java control flow statements are constructs that **dictate the order in which instructions are executed**.

They can be categorized into 3 main types:

1. Decision-making statements
2. Loop statements
3. Branching statements
4. **Decision-Making Statements:**

Decision-Making statements control the flow of execution based on **certain conditions**. These statements allow the program to choose different **paths** of execution based on the evaluation of Boolean expressions.

**Main decision-making statements:**

1. **if**
2. **if-else**
3. **if-else-if ladder**
4. **nested if**
5. **switch**

**1. if statement**:

The if statement **evaluates a condition** (a **Boolean expression**). If the condition is **true**, the block of code within the if statement is executed. If the condition is **false**, the block is skipped.

**Syntax:**

**A black background with white text

Description automatically generated**

**Example:**

A black background with white text

Description automatically generated

**2. if-else statement**:

The if-else statement provides **two blocks of code**: one that executes if the condition is **true** and another that executes if the condition is **false**.

**Syntax:**

**A black screen with white text

Description automatically generated**

**Example:**

A computer screen with white text

Description automatically generated

**3. if-else-if ladder**:

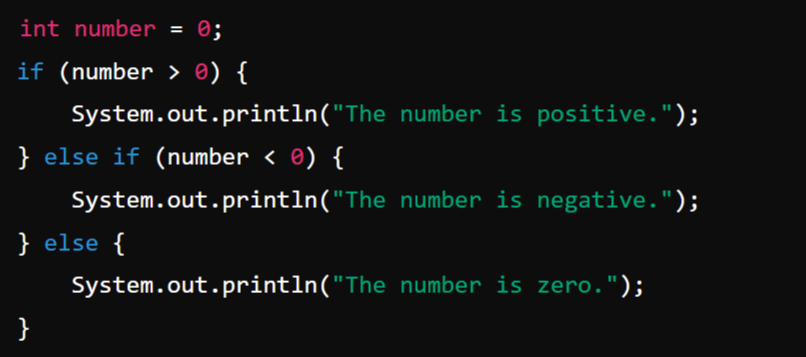
The if-else-if ladder allows for **multiple conditions** to be evaluated **sequentially**. The first condition that evaluates to true will have its corresponding block executed, and the rest of the ladder will be skipped. If none of the conditions are true, the **else** block (if present) will be executed.

**Syntax:**

**A computer screen with white text

Description automatically generated**

**Example:**

****

**4. Nested if statement:**

Nested if statements allow an **if statement** to be placed **inside another if statement**. This allows for **more complex decision-making** processes where multiple conditions must be true for a block of code to execute.

**Syntax:**

**A computer screen with white text

Description automatically generated**

**Example:**

**A computer code on a black background

Description automatically generated**

**5. Switch statement:**

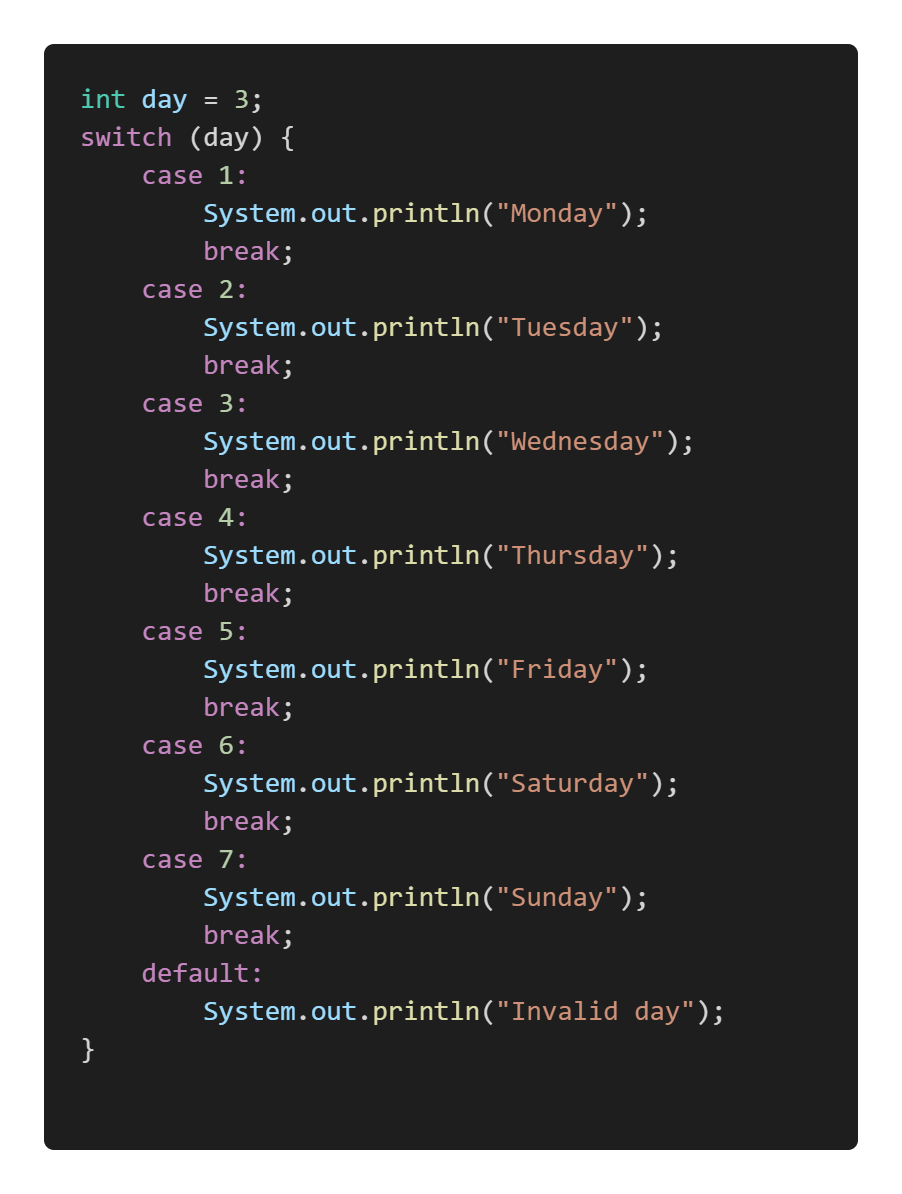
The switch statement evaluates an expression and compares it to a **list of case** values. When a match is found, the corresponding block of code is executed. The **break** statement is used to exit the switch block after the matched case has been executed. If no match is found, the default block (if present) is executed. The switch statement is often used as an alternative to the **if-else-if ladder** for better readability and performance when dealing with multiple possible values of an expression.

**Syntax:**

A computer screen with white text

Description automatically generated

**Example:**

****

**2. Loop Statements:**

In Java, loop statements **allow** the **execution of a block of code repeatedly** **based on a condition**. They are essential for tasks that require iteration, such as processing elements in an array or repeatedly performing an operation until a certain condition is met.

**Looping Statements are:**

1. **for** loop
2. **while** loop **(Entry Controlled Loop)**
3. **do-while** loop **(Exit Controlled Loop)**
4. **for-each** loop (enhanced for loop)

**1. for loop:**

The for loop provides a **concise** **way** of writing the loop structure. It is used when the **number of iterations is known beforehand**.

**Syntax:**

**A black background with white text

Description automatically generated**

**Example:**

**A black background with white text

Description automatically generated**

**2. while loop:**

The while loop continually executes a block of code as long as a specified condition is true. The condition is checked before the execution of the loop body, making it a pre-test loop.

**Syntax:**

**A black background with white text

Description automatically generated**

**Example:**

**A computer screen with white text

Description automatically generated**

**3. do-while loop:**

The do-while loop is similar to the while loop, but it guarantees that the loop body will be executed at **least once** because the condition is checked **after the execution of the loop body**.

**Syntax:**

**A black background with white text

Description automatically generated**

**Example:**

A computer screen with white text

Description automatically generated

**4. for-each loop (Enhanced for loop)**

The for-each loop, introduced in **Java 5,** is used to **iterate over elements in an array or a collection**, making the code more readable and reducing the risk of errors.

**Syntax:**

**A black background with white text

Description automatically generated**

**Example:**

A black screen with white text

Description automatically generated

**3. Branching** or **Jump Statements:**

In Java, branching statements are used to **alter the flow of execution** within a program based on certain conditions. They allow for more **complex** and **dynamic** behavior by enabling jumps to different parts of the code.

**The main branching statements in Java are:**

1. **break**
2. **continue**
3. **return**

**1. break statement:**

The break statement is used to terminate the execution of a loop or switch statement prematurely. It can be used in for, while, and do-while loops, as well as in switch statements.

**Syntax:**

**A black background with blue letters

Description automatically generated**

**Example:** When a break statement is encountered inside a loop or switch, control is transferred to the statement immediately following the loop or switch.

A computer screen shot of a code

Description automatically generated

**2. continue statement:**

The continue statement is used to skip the current iteration of a loop and proceed to the next iteration. It can be used in for, while, and do-while loops.

**Syntax:**

**A blue and black text

Description automatically generated**

**Example:** When a continue statement is encountered inside a loop, control is immediately transferred to the next iteration of the loop, bypassing the remaining code in the current iteration.

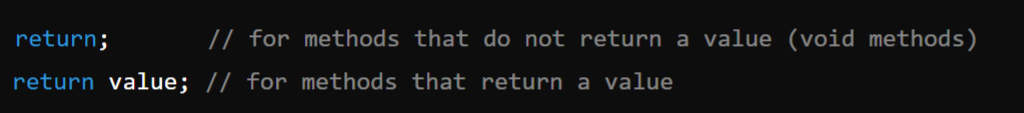
A computer screen with numbers and symbols

Description automatically generated

**3. return statement**

The return statement is used to exit from the current method and optionally return a value to the calling method. It can be used in any method to terminate its execution and, if needed, provide a value back to the caller.

**Syntax:**

****

**Example-1:** When a return statement is encountered, the method **execution is terminated**, and control is returned to the caller. If the method has a return type other than void, a **value** **must be returned**.

A screen shot of a computer code

Description automatically generated

Example-2: Method that returns a value:

A black background with white text

Description automatically generated