Core Java

Day-1

What is Java?

Java is a high-level, object-oriented programming language developed by Sun Microsystems (now owned by Oracle Corporation) in 1995.

It's designed to be platform-independent, meaning that Java programs can run on any device that has a Java Virtual Machine (JVM).

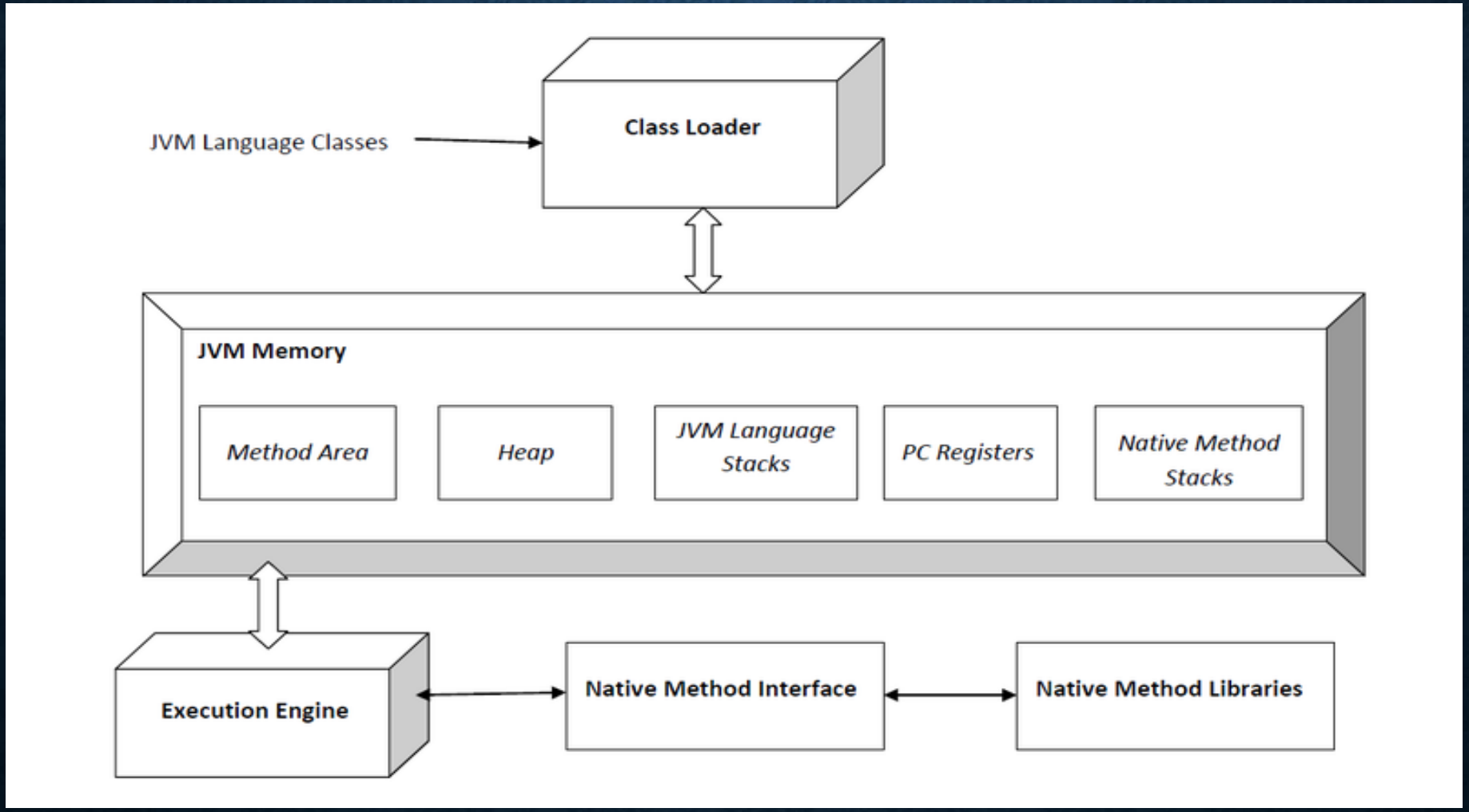
FEATURES OF JAVA

- Simple: Java was designed to be easy to learn and use.
- Object-Oriented: Java supports the principles of object-oriented programming (OOP), including encapsulation, inheritance, and polymorphism.
- Platform-Independent: Java programs can run on any device that has a Java Virtual Machine (JVM), making it highly portable.
- Secure: Java's security features help protect against viruses and other malicious software.
- Robust: Java is designed to be robust and reliable, with features like automatic memory management (garbage collection) to help prevent common programming errors.
- Multithreaded: Java supports multithreading, allowing programs to perform multiple tasks simultaneously.
- Architecture-neutral: Java's architecture-neutral design allows it to run on a variety of hardware platforms.
- Dynamic: Java supports dynamic loading of classes, which allows programs to load classes on demand.

JVM ARCHITECTURE

- The Java Virtual Machine (JVM) is an abstract computing machine that provides a runtime environment in which Java bytecode can be executed. It's responsible for executing Java programs by interpreting the bytecode and translating it into machine code that can be understood by the underlying hardware.
- The JVM architecture consists of three main components:
- Class Loader: The class loader loads Java class files into memory as needed during program execution.
- Runtime Data Area: The runtime data area is where the JVM stores data required for program execution, including the method area, heap, stack, and PC (Program Counter) register.
- Execution Engine: The execution engine is responsible for executing Java bytecode by interpreting it or compiling it into native machine code.

JVM ARCHITECTURE



JDK AND ITS USAGE

- The Java Development Kit (JDK) is a software development kit that includes everything developers need to develop, debug, and run Java applications. It consists of the Java Runtime Environment (JRE), the Java compiler, and other tools and libraries.
- Developers use the JDK to:
  - Compile Java source code into bytecode.
  - Run Java applications on their local machine.
  - Debug Java programs.
  - Package Java applications for distribution.

JAVA TOKENS

- Java tokens are the smallest individual units of a Java program. These tokens are the building blocks of a Java program. There are five types of Java tokens:

Keywords:

- Keywords are reserved words that have predefined meanings in Java. These words cannot be used as identifiers (variable names or method names). Examples of keywords include class, public, static, void, if, else, etc.

JAVA TOKENS

Identifiers:

Identifiers are names given to classes, methods, variables, etc. in a Java program. They must follow certain rules:

- They cannot start with a digit.
- They can contain letters, digits, underscores, and dollar signs.
- They cannot be Java keywords.
- They are case-sensitive.

JAVA TOKENS

Literals:

Literals represent constant values that are used in Java. There are different types of literals:

- Integer literals (e.g., 10, 0xFF)
- Floating-point literals (e.g., 3.14, 2.5f)
- Boolean literals (true and false)
- Character literals (e.g., 'a', '5')
- String literals (e.g., "Hello", "Java")

JAVA TOKENS

Operators:

- Operators perform operations on variables and values. Examples include arithmetic operators (+, -, *, /), assignment operators (=, +=, -=), comparison operators (==, !=, <, >), logical operators (&&, | |, !), etc.

Separators:

- Separators are characters used to separate tokens in a Java program. Examples include parentheses (), braces {}, square brackets [], commas ,, semicolons ;, etc.

DECLARING VARIABLES AND METHODS

- In Java, variables and methods must be declared before they can be used.
- Variables are containers for storing data values. They must be declared with a data type and an optional initial value.
- Example: int num = 10; String name = "John";
- Methods:
- Methods are blocks of code that perform a specific task. They are declared within a class and can optionally return a value.
- Example:

```
public int add(int a, int b)
{
    return a + b;
}
```

WORKING WITH PRIMITIVE DATA TYPES

Java supports eight primitive data types, which are divided into four categories:

- Integer Types: byte, short, int, long
- Floating-Point Types: float, double
- Character Type: char
- Boolean Type: Boolean

Each primitive data type has a predefined size and range of values, as well as default values.

NAMING CONVENTIONS

- Packages:
  - Package names are written in lowercase.
  - Use your organization's domain name in reverse order as the prefix for package names (e.g., com.example.mypackage).
- Classes and Interfaces:
  - Class and interface names should be nouns and written in CamelCase.
  - Start class names with an uppercase letter.
  - Use meaningful names that describe the class's purpose.
  - Avoid abbreviations unless they are widely understood (e.g., HTTPServer).
- Methods:
  - Method names should be verbs and written in camelCase.
  - Start method names with a lowercase letter.
  - Use meaningful names that describe the method's action or behavior.
  - Follow the JavaBeans conventions for getter and setter methods (e.g., getName(), setName()).

NAMING CONVENTIONS

- Enums:
  - Enum types should be written in CamelCase and follow the class naming conventions.
  - Enum constants should be written in uppercase letters with underscores separating words.
- Packages, Classes, Interfaces, and Methods:
  - Use meaningful and descriptive names.
  - Avoid using acronyms or abbreviations unless they are widely understood.
  - Be consistent throughout your codebase.
- Indentation and Formatting:
  - Use consistent indentation (typically four spaces).
  - Use braces {} for block statements, even if they contain only a single statement.
  - Follow standard Java code formatting conventions.

NAMING CONVENTIONS

- Variables:
  - Variable names should be written in camelCase.
  - Start variable names with a lowercase letter.
  - Use meaningful names that describe the variable's purpose.
  - Avoid using single-letter variable names except for loop counters.
- Constants:
  - Constants should be written in uppercase letters with underscores separating words (e.g., MAX_SIZE).
  - Use meaningful names that describe the constant's purpose.
  - Declare constants using the final keyword (e.g., static final int MAX_SIZE = 100;).

DATA TYPE COMPATIBILITY

- In Java, data type compatibility refers to the ability to perform operations between different data types.
- Widening Conversion:
  - Widening conversion occurs when a smaller data type is converted to a larger data type automatically without loss of information.

```
Example:
int numInt = 10;
double numDouble = numInt; // Widening conversion from int to double
```

- Narrowing Conversion:
  - Narrowing conversion occurs when a larger data type is converted to a smaller data type. It may result in loss of information and requires explicit casting.

```
Example:
double numDouble = 10.5;
int numInt = (int) numDouble; // Narrowing conversion from double to int
```

OPERATORS

Operators in Java are symbols that perform operations on variables and values.

Unary Operators

- Unary Minus(-),++,--,!

Binary Operators

- Arithmetic Operators
  - +,-,*,/,%
- Relational Operators
  - >,<,>=,<=,==,!=
- Logical Operators
  - &&,||,!

Ternary Operator

?:

. CONTROL STATEMENTS

Control statements in Java are used to control the flow of execution in a program.

- Conditional Statements (if-else)
- Selection Statements(switch-case)
- Looping Statements (for, while, do-while)
- Branching Statements(break,continue)

ARRAYS

Arrays (1-D and Multidimensional)

Arrays are used to store multiple values of the same data type. They can be one-dimensional or multidimensional.

Example (1-D array):

```
int[] numbers = {1, 2, 3, 4, 5};
```

Example (Multidimensional array):

```
int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING (OOP) IN JAVA:

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code.

Java is a powerful, class-based, object-oriented language that is designed to be platform-independent and to have as few implementation dependencies as possible.

KEY CONCEPTS

- Classes: In Java, classes are the blueprint for creating objects. They define the properties (attributes) and behaviors (methods) that objects of the class will have.

- Objects: Objects are instances of classes. They represent real-world entities and have their own unique data and behavior.

OOP PARADIGMS

Abstraction in Java:

Abstraction in Java is the process of hiding the implementation details and showing only the necessary features of an object. It is achieved using abstract classes and interfaces.

Encapsulation in Java:

Encapsulation in Java is achieved by bundling the data (attributes) and methods that operate on the data into a single unit (class). Access to the data is controlled through methods.

OOP PARADIGMS

Inheritance in Java

Inheritance in Java allows a class to inherit properties and behavior from another class. The class that is being inherited from is called the superclass or parent class, and the class that inherits is called the subclass or child class.

Polymorphism in Java:

Polymorphism in Java allows objects of different classes to be treated as objects of a common superclass. It is achieved through method overriding and method overloading.

STRUCTURE OF A JAVA CLASS

- A Java class is a blueprint for creating objects. It consists of the following components:
- Class Declaration: The class declaration specifies the name of the class and can include modifiers such as public, private, or abstract.
- Fields (Variables): Fields represent the state of an object and store data.
- Methods: Methods define the behavior of an object and represent actions that an object can perform.
- Constructors: Constructors are special methods used to initialize objects.
- Inner Classes: Java allows classes to be nested within other classes.

ACCESS MODIFIERS

- In Java, access modifiers are keywords used to control the accessibility or visibility of classes, methods, and variables.
- They specify how classes, methods, or variables can be accessed by other classes or components in the same or different packages.
- Public:
  - The public modifier makes a class, method, or variable accessible from any other class.
  - Classes: Public classes can be accessed from any other class.
  - Methods: Public methods can be called from any other class.
  - Variables: Public variables can be accessed and modified from any other class.
- Private:
  - The private modifier restricts access to the class, method, or variable to only the class in which it is declared.
  - Classes: Private classes cannot be accessed from outside the containing class.
  - Methods: Private methods can only be called from within the same class.
  - Variables: Private variables cannot be accessed or modified from outside the containing class.

ACCESS MODIFIERS

- Protected:
  - The protected modifier restricts access to the class, method, or variable to the same package or subclasses (even if they are in different packages).
  - Classes: Protected classes can be accessed from within the same package and subclasses.
  - Methods: Protected methods can be called from within the same package and subclasses.
  - Variables: Protected variables can be accessed and modified from within the same package and subclasses.
- Default (Package-private):
  - When no access modifier is specified (also known as package-private), the class, method, or variable is accessible only within the same package.
  - Classes: Default classes can be accessed from within the same package.
  - Methods: Default methods can be called from within the same package.
  - Variables: Default variables can be accessed and modified from within the same package.
  - Access modifiers provide a way to control the visibility and accessibility of classes, methods, and variables, contributing to the encapsulation and security of Java programs

CONSTRUCTORS

- Constructor is a special type of method that is automatically called when an object of a class is created. It is used to initialize the newly created object.
- The purpose of a constructor is to ensure that all the necessary setup for an object is done when it is created, such as initializing instance variables, allocating resources, or performing any other necessary tasks.

CONSTRUCTORS

- Constructor Name: The constructor method must have the same name as the class.
- No Return Type: Constructors don't specify a return type, not even void.
- Accessibility: Constructors can have different access modifiers:
- Overloading: Constructors can be overloaded, meaning you can have multiple constructors with different parameter lists in the same class.
- Chaining Constructors (Constructor Overloading): Java allows constructor chaining, meaning you can call one constructor from another using this() keyword. This is useful when you want to reuse some initialization code. However, if you use this() to call another constructor, it must be the first statement in the constructor.
- Default Constructor: If no constructor is explicitly defined in a class, Java provides a default constructor with no parameters. However, if you define any constructor, the default constructor is not provided automatically.
- Initialization: Constructors are typically used to initialize instance variables and perform any necessary setup when an object is created.
- Cannot be Inherited or Overridden: Constructors are not inherited like methods. Subclasses cannot override constructors from their superclass.