Contents

[What is Java? 2](#_Toc152109054)

[Features of Java 2](#_Toc152109055)

[JDK – Java Development Kit 3](#_Toc152109056)

[JRE – Java Runtime Environment 4](#_Toc152109057)

[JVM – Java Virtual Machine 4](#_Toc152109058)

[Java Class 5](#_Toc152109059)

[Java Methods 6](#_Toc152109060)

[Method Overloading 7](#_Toc152109061)

[Constructor in Java 8](#_Toc152109062)

[Constructor Overloading: 9](#_Toc152109063)

# What is Java?

Java is a high-level, platform independent, write once use everywhere programming language used to various including web development, android app development, desktop software and server-side applications.

# Features of Java

1. Platform Independence: Java is designed to be platform-independent, thanks to the use of the Java Virtual Machine (JVM). This means that Java applications can run on any system with a compatible JVM, making it a "write once, run anywhere" language.
2. Object-Oriented: Java is a fully object-oriented programming language, emphasizing the use of objects and classes for code organization and reusability.
3. Strongly Typed: Java enforces strong typing, which helps catch errors at compile time and promotes code reliability.
4. Automatic Memory Management: Java incorporates automatic garbage collection, relieving developers from manual memory management, reducing memory-related errors and vulnerabilities.
5. Multithreading: Java offers robust support for multithreading, making it suitable for concurrent and parallel programming.
6. Security: Java includes various security features, such as a security manager, to create secure and reliable applications.
7. Rich Standard Library: Java provides a vast standard library that simplifies common programming tasks and offers numerous built-in classes and methods.
8. Exception Handling: Java includes a robust exception-handling mechanism to manage and recover from runtime errors.
9. Portability: Java's platform independence and bytecode compilation make it highly portable, ensuring consistent behaviour across different systems.
10. Large Ecosystem: Java has a massive and active community, resulting in a rich ecosystem of libraries, frameworks, and tools for various application domains.
11. Dynamic Class Loading: Java supports dynamic class loading, allowing classes to be loaded on-demand, which is useful for applications that need to adapt to changing requirements.
12. High Performance: Modern Java virtual machines (JVMs) are optimized for high performance, with just-in-time (JIT) compilation and various runtime optimizations.
13. Interoperability: Java can be integrated with other programming languages, often via Java Native Interface (JNI) or through various interop mechanisms.
14. Network Support: Java includes libraries for network programming, enabling the development of networked and distributed applications.
15. Community Support: Java has a strong developer community with extensive documentation, forums, and resources, making it easy to find help and guidance.

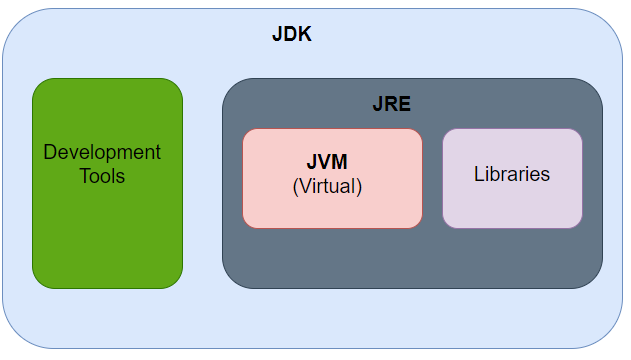
# JDK – Java Development Kit

JDK is a cross-platform software development environment which includes set of tools and libraries required for Java based developments.

JDK = JRE + Development Tools

The Java Development Kit is an implementation of one of the Java Platform:

1. Standard Edition (Java SE),
2. Java Enterprise Edition (Java EE),
3. Micro Edition (Java ME)



# JRE – Java Runtime Environment

The JRE is a software package that provides the runtime environment necessary for executing Java applications on a computer. It contains JVM and Libraries needed to execute JAVA applications.

# JVM – Java Virtual Machine

The Java Virtual Machine is a virtual machine that provides a runtime environment for Java applications and programs.

Java is platform independent due to JVM.

# Java Class

A class in Java is a blueprint or a template that defines the properties and behaviours (methods) common to all objects of a certain type. It serves as a user-defined data type that encapsulates data and methods that operate on that data.

# Java Methods

A method in Java is a block of code that, when called, performs specific actions mentioned in it. For instance, if you have written instructions to draw a circle in the method, it will do that task. A method can accept data as its arguments. To use method, we need to call it.

There are two types of methods based on parameters accepted by method: (a) Parameterless method (b) Parameterized method.

Syntax:

1. Declaration :

//Paramterless method declaration

returnType methodName() {

    //Write instructions here

}

//Parameterized method declaration

returnType paramterizedMethodName(int x, String y) {

    //Write instructions here

}

1. Calling :

objectName.methodName();

objectName.parameterizedMethodName(5, “abhishek”);

Example:

class example {

    // This is a parameterless method declaration.

    void printHelloWorld() {

        System.out.println("Hello, World!");

    }

    // This is a parameterized method declaration.

    void printGiven(int x, String y) {

        System.out.println(x + " " + y);

    }

    public static void main(String[] args) {

        //creating object

        example e = new example();

        // This is a method call.

        e.printHelloWorld();

        e.printGiven(5, "Hi");

    }

}

## Method Overloading

Method Overloading in Java refers to the capability of a class to define multiple methods with the same name but different parameter lists. This allows a class to have more than one method with the same name, provided that the number or types of parameters are different.  
public class Calculator {

    // Overloaded methods for addition

    public int add(int a, int b) {

        return a + b;

    }

    public double add(double a, double b) {

        return a + b;

    }

    // Overloaded methods for multiplication

    public int multiply(int a, int b) {

        return a \* b;

    }

    public double multiply(double a, double b) {

        return a \* b;

    }

    public static void main(String[] args) {

        Calculator calculator = new Calculator();

        // Calling overloaded methods

        int resultInt = calculator.add(5, 10);

        double resultDouble = calculator.add(3.5, 2.5);

        int productInt = calculator.multiply(4, 7);

        double productDouble = calculator.multiply(2.5, 1.5);

    }

}

# Constructor in Java

A constructor is a special type of method in Java. It shares the same name as the class. Constructors are responsible for initializing the state of an object, setting initial values to its fields. Constructors are automatically invoked when an object is created using the new keyword. Unlike methods, constructors do not have a return type, not even void.

Multiple Constructors (Overloading): A class can have multiple constructors, a concept known as constructor overloading, where each constructor has a different parameter list.

Syntax:

//Paramterless default constructor declaration

className() {

    //Write instructions here

}

//Parameterized constructor declaration

className(int x, String y) {

    //Write instructions here

}

Example:

class Car {

    // Fields

    String model;

    int year;

    // Default Constructor (if not provided by the programmer, the compiler adds a default one)

    public Car() {

        model = "Unknown";

        year = 0;

    }

    // Parameterized Constructor

    public Car(String carModel, int carYear) {

        model = carModel;

        year = carYear;

    }

    // Method to display car information

    void displayInfo() {

        System.out.println("Model: " + model);

        System.out.println("Year: " + year);

    }

    public static void main(String[] args) {

        // Creating objects using different constructors

        // Default constructor

Car car1 = new Car();

        // Parameterized constructor

Car car2 = new Car("Toyota", 2022);

        // Calling method on objects

        car1.displayInfo();

        car2.displayInfo();

    }

}

## Constructor Overloading

Constructor overloading in Java is a feature that allows a class to define multiple constructors with different parameter lists. This enables objects of the class to be initialized in various ways, providing flexibility in object creation.

public class Student {

    String name;

    int age;

    double grade;

    // Default constructor

    public Student() {

        name = "Unknown";

        age = 0;

        grade = 0.0;

    }

    // Parameterized constructor with name and age

    public Student(String studentName, int studentAge) {

        name = studentName;

        age = studentAge;

        grade = 0.0;

    }

    // Parameterized constructor with name, age, and grade

    public Student(String studentName, int studentAge, double studentGrade) {

        name = studentName;

        age = studentAge;

        grade = studentGrade;

    }

    // Method to display student information

    void displayInfo() {

        System.out.println("Name: " + name);

        System.out.println("Age: " + age);

        System.out.println("Grade: " + grade);

    }

    public static void main(String[] args) {

        // Creating objects using different constructors

        // Default constructor

        Student student1 = new Student();

        // Parameterized constructor

        Student student2 = new Student("John", 20);

        // Parameterized constructor

        Student student3 = new Student("Alice", 22, 85.5);

        // Calling method on objects

        student1.displayInfo();

        student2.displayInfo();

        student3.displayInfo();

    }

}