

Chapter 1

Introduction to Java



What is Java ?

Java is a general-purpose programming language which is class based and object-oriented, designed for building a wide range of applications, from desktop software to web applications, mobile apps, and enterprise systems.

Java follows the principles of object-oriented programming (OOP). It emphasizes the use of classes, objects, and inheritance for building modular and reusable code.

The primary goal of Java is to provide a platform-independent, robust, and secure programming language that allows developers to write code once and run it anywhere (WORA)

History of Java

- Java was developed by Sun Microsystems in the year 1995, by James Gosling.
- *James Gosling* is known as the father of Java.
- Sun Microsystems was acquired by Oracle in 2010
- It is Originally called 'Oak'. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.
- Java 1.0 was launched in January 1996.

Features of Java

1. Simple :

Java is very easy to learn, and its syntax is simple, clean and easy to understand. Java Syntax is based on C++.

2. Platform Independence:

Platform independence refers to the ability of a programming language or software to run on different platforms (such as different operating systems or hardware architectures) without modification.

Java programs can run on any device or operating system that has a Java Virtual Machine (JVM), adhering to the principle of WORA (Write Once, Run Anywhere)

3. Object Oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

4. Robust and Secure

Java programs are reliable and secure due to features like garbage collection, exception handling, and strict type checking.

5. Portable

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

6. Multithreading

Java supports concurrent programming with built-in features for handling multiple tasks simultaneously. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area.

7. Dynamic

Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. Java supports dynamic compilation and automatic memory management (garbage collection).

8. High Performance

Java achieves high performance through just-in-time (JIT) compilation and optimized execution.

9. Architecture Neutral

Java bytecode (compiled Java code) can be executed on any system, providing flexibility in deployment.

10. Automatic Memory Management

Java handles memory management automatically through garbage collection, reducing the risk of memory leaks and crashes.

11. Community Support

Java has a large and active community that contributes to its ecosystem with frameworks, libraries, and tools, supporting rapid development and innovation.

Java Environment

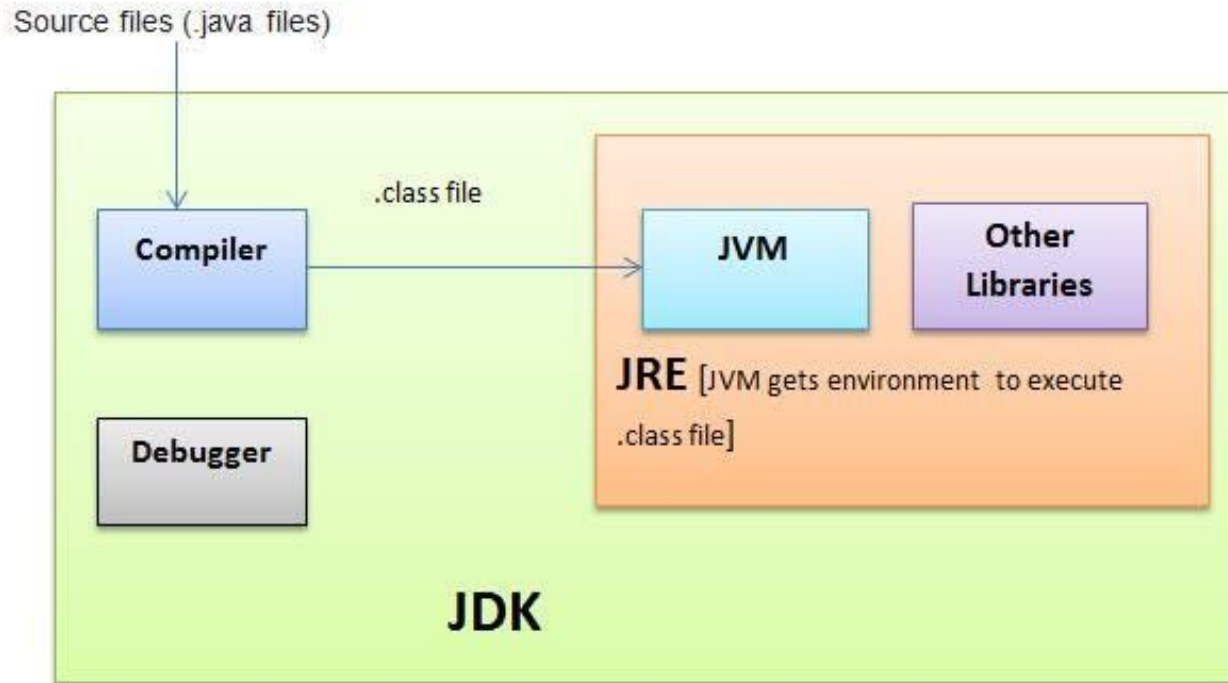
The Java environment consists of the tools and components necessary to develop and run Java applications.

It includes:

Java Development Kit (JDK): This is the core component for developing Java applications. It includes the Java compiler, which translates Java source code into bytecode, and other tools like the Java debugger and documentation.

Java Runtime Environment (JRE): The JRE is necessary for running Java applications. It includes the Java Virtual Machine (JVM), which executes Java bytecode. Users need the JRE installed to run Java programs.

Java Virtual Machine (JVM): The JVM is a crucial part of the Java environment. It interprets Java bytecode and translates it into machine-specific code, allowing Java applications to run on any device or operating system that has a compatible JVM.



Structure of Java Programs

Documentation Section
Package Section
Import Statements
Interface Statements
Class Definitions
<pre>main method class { main method definition }</pre>

Main method class is
Essential

Simple Java Program

Hello.java

//This is a simple "Hello, World!" program in Java.

public class Hello

{

public static void main(String args[])

{

//Printing Hello World

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

}

}

To Compile:
Javac Hello.java

To Execute:
Java Hello

Output
Hello World

Understanding the Program

- The public keyword indicates that the class can be accessed from outside its package.
- The main method is the entry point of a Java application.
- String[] args is a parameter that is typically used to pass command-line arguments to the main method of a Java application
- Statements in Java end with a semicolon (;)

Comments in Java

Comments non-executable statement used to explain Java code, and to make it more readable.

Types of Comment in Java

1. Single-Line Comments:

Single-line comments start with `//` and continue until the end of the line. They are used for short comments or annotations on a single line of code.

Syntax:

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

2. Multi-Line Comments:

Multi-line comments start with `/*` and end with `*/`. They can span multiple lines and are used for longer explanations or temporarily disabling blocks of code.

Syntax:

```
/* This is the example of  
Multi-line Comments. */
```

Java Naming Conventions

Classes follows PascalConvention

Example: MyFirstProgram

Functions & Variables follows camelCaseConventions

Example: addTwoNumbers(), myVariable

Constants follows UPPERCASE

Example: MAX_VALUE

Variables

A variable is a container that stores a value. Variables are fundamental units used to store data in computer memory. Value of variables can change during the execution of a program.

Java is a statically-typed programming language. It means, all variables must be declared before they can be used.

Variable Declaration: `dataType varName;`

Variable Initialization: `varName=value;`

Variable Declaration and initialization in single Statement:

`dataType varName= value;`

Rules when declaring a variable name

- Variables name cannot start with a digit. Example: 1num is invalid
- Names are case-sensitive. 'num' and 'NUM' are different.
- Cannot use Java reserved keywords (public, class, etc..)
- Blank spaces are not allowed. Example: 'my Number' is not allowed.
- We can use alphabets(a-z, A-Z), digits (0-9), dollar signs (\$), and underscores (_).

Data Types

Data types are categorized into two main groups:

1. Primitive data type
2. Non-primitive data type

Primitive Data Type

Primitive data types in Java are the most basic data types, and they directly store data values. They are not objects and do not have methods.

There are 8 primitive types available in Java:

1. Byte:

- The 'byte' store whole numbers from -128 to 127.
- Size: 1 byte
- Example: `byte b = 10;`

2. Short:

- The 'short' data type can store whole numbers from -32,768 to 32,767
- Size: 2 bytes
- Example: `short s = 1000;`

3. Int:

- The 'int' data type can store whole numbers from -2^{31} to $2^{31} - 1$.
- Size: 4 bytes
- Example: `int i = 100000;`

4. Long:

- The 'long' data type can store whole numbers from -2^{63} to $2^{63} - 1$.
- Size: 8 bytes
- Example: `long l = 1000000000L; // here L indicating it's a long literal.`

5. Float:

- The 'float' stores fractional numbers. Sufficient for storing 6 to 7 decimal points.
- Size: 4 bytes
- Example: `float f = 3.14f; //here f indicating it's a float literal.`

6. Double:

- The 'double' stores fractional numbers. Sufficient for storing 15 decimal points.
- Size: 8 bytes
- Example: `double d = 3.141592653589;`

7. char

- The char data type is used to store a single character. The character must be surrounded by single quotes, like 'A' or 'c'.
- Size: 2 bytes
- Example: `char ch = 'A';`

8. boolean

- The 'boolean' data type only stores 'true' or 'false' values.
- Mostly used for conditional testing.
- Size: 1 bit
- Example: `boolean flag = true;`

Non Primitive data types

- These data types are used to store multiple values.
- Non-primitive data types are called reference types because they refer to objects.
- Non-primitive types can be used to call methods to perform certain operations.

Non-Primitive data types are:

- **String:** Represents a sequence of characters, such as "Hello World" .
- **Arrays:** Collections of elements of the same type, like `int[]` or `String[]`.
- **Classes:** Custom-defined types that encapsulate data and methods. For example, `Person`, `Car`, or `Book` classes can be created to represent specific types of objects.
- **Interfaces:** Defines a contract for classes to implement. Interfaces can have method signatures but no implementations.

Strings

A string is a sequence of characters used to store text.

Strings are enclosed in double quotes.

Syntax: `String var_name="text";`

Example: `String str="Hello World";`

Final Keyword

- 'Final' keyword is used to declare final variable.
- It's value cannot be changed once it has been assigned.
- Once a final variable is initialized, its value remains constant throughout the execution of the program.
- Once initialized, the value of a final variable cannot be changed. Attempting to re-assign a value to a final variable will result in a compilation error.
- final variables are often used to declare constants in Java programs.

Syntax: `final data_type var_name = value;`

Example: `final int Z = 30;`

`Z= 40; // Compilation error: cannot assign a value to final variable z`

Control Statements

Control statements are used to control the flow of execution within a program. They let you decide which parts of your code should execute based on conditions, loops, or choices you make.

Control statements include the following:

1. Conditional Statements
2. Loops
3. Jump Statements

Conditional Statements

if Statement:

Executes a block of code only if a specified condition is true.

Syntax

```
if (condition) {  
    // Block of code to be executed if the  
    condition is true  
}
```

if-else Statement: Executes one block of code if the condition is true, and another block if the condition is false.

Syntax

```
if (condition) {  
    // Block of code to be executed if the condition  
    is true  
} else {  
    // Block of code to be executed if the condition is  
    false  
}
```

if-else-if Statement:

Allows you to check for multiple conditions sequentially.

```
if (condition1) {
```

```
    // Block of code to be executed if condition1 is true.
```

```
} else if (condition2) {
```

```
    // Block of code to be executed if condition 2 is true.
```

```
} else {
```

```
    // Block of code to be executed if all conditions are false.
```

```
}
```

Switch Statement: It provides a way to execute different blocks of code based on the value of an expression.

Syntax:

```
switch (expression) {  
    case value1:  
        // Code to execute if expression == value1  
        break;  
    case value2:  
        // Code to execute if expression == value2  
        break;  
    // more cases as needed  
    default:  
        // Code to execute if none of the above cases are matched  
}
```

Loops

Loop statements are used to repeat a block of code multiple times until a specified condition is met.

for Loop: Executes a sequence of statements multiple times and is ideal when the number of iterations is known.

Syntax

```
for (initialization; condition; iteration) {  
    // Block of code to be executed repeatedly.  
}
```

while Loop: Continuously executes a block of code as long as a specified condition is true. Use it when the number of iterations is not known in advance.

Syntax

```
while (condition) {
```

```
    // Block of code to be executed repeatedly
```

```
}
```

do-while Loop: Similar to the while loop, but it guarantees that the block of code is executed at least once before checking the condition.

Syntax

```
do {
```

```
    // Block of code to be executed repeatedly
```

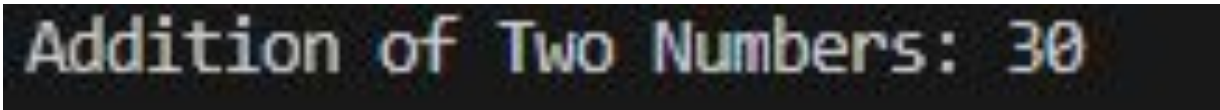
```
} while (condition);
```

Programs

1. Write a program in java to add two numbers.
2. Write a program in java to swap two numbers.
3. Write a program in java to print fibonacci series.
4. Write a program in java to print table of 5.

1. AddTwoNo.java

```
public class AddTwoNo
{
    public static void main(String args[])
    {
        int num1=10;
        int num2=20;
        int mysum=num1+num2;
        System.out.println("Addition of Two Numbers: "+mysum);
    }
}
```



Addition of Two Numbers: 30

A screenshot of a terminal window with a black background. The text "Addition of Two Numbers: 30" is displayed in a light blue, monospaced font.

2. Swap.java

```
public class Swap
{
    public static void main(String args[])
    {
        int a=10,b=20;
        System.out.println("Values before Swapping: "+ "a="+a+" b="+b);
        int temp=a;
        a=b;
        b=temp;
        System.out.println("Values after Swapping: "+ "a="+a+" b="+b);
    }
}
```

Values before Swapping: a=10 b=20

Values after Swapping: a=20 b=10

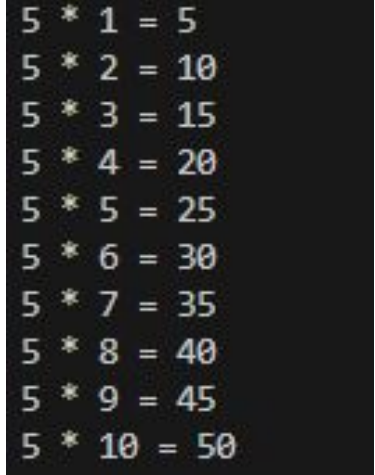
3. Fibo.java

```
public class Fibo
{
    public static void main(String[] args) {
        int n1=0, n2=1;
        System.out.print(n1+ " "+ n2);
        for (int i=3; i<=5; i++)
        {
            int nextnum=n1+ n2;
            System.out.print(" "+nextnum);
            n1=n2;
            n2=nextnum;
        }
    }
}
```

0 1 1 2 3

4. Table5.java

```
public class Table5
{
    public static void main(String args[])
    {
        int number=5;
        for(int i=1;i<=10;i++)
        {
            int ans=number * i;
            System.out.println(number + " * " +i +" = "+ans);
        }
    }
}
```



```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Accepting User Input

In Java, there are several ways to accept user input, each suited for different scenarios:

Methods of Accepting User Input in Java:

1. **Command Line Arguments:**

Arguments passed to the Java program when it is invoked from the command line.

They are Accessed via the main method's parameter `String[] args`.

`args[0]` is the first argument, `args[1]` is the second, and so on.

Example:

```
public class Arguments
{
    public static void main(String[] args)
    {
        System.out.println(args[0]);
    }
}
```

Note: To convert args to integer value, we use Integer.parseInt().

Example: int num=Integer.parseInt(args[0])

2. Scanner Class

- It is used to take user input and is found in the java.util package.
- Allows reading various types of input like int, double, String, etc., from standard input (keyboard).
- To use the Scanner class, we first create an object of the Scanner class.
- It's important to close the Scanner when you're done using it to release underlying resources.
- Provides methods like nextInt(), nextDouble(), nextLine() to read input.

Syntax of creating scanner object:

```
Scanner sc = new Scanner(System.in); // Reading from standard input (keyboard)
```

```
import java.util.Scanner;

public class ScannerExample
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter your name: ");
        String name = sc.nextLine();
        System.out.println("Hello, " + name);

        System.out.println("\nEnter Your Roll Number: ");
        int number=sc.nextInt();
        System.out.println("Roll Number: "+ number);

        System.out.println("\nEnter a double: ");
        double decimal = sc.nextDouble();
        System.out.println("You entered: " + decimal);

        sc.close(); // Close the scanner to release resources
    }
}
```

3. BufferedReader Class

- It is the part of java.io package.
- Reads text from a character-input stream, usually faster than Scanner.
- Used with InputStreamReader to read from standard input (keyboard).

Syntax

```
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
```

Some methods of BufferedReader

- `int read()`: It is used for reading a single character.
- `String readLine()`: It is used for reading a line of text.
- `String ready()`: It is used to test whether the input stream is ready to be read.
- `void close()`: It closes the input stream and releases any of the system resources associated with the stream.

Arrays

- An array is a collection of similar types of data.
- Arrays are used to store multiple values in a single variable, making it easier to iterate over and manipulate data.

Types of Array in java

1. Single Dimensional Array
2. Multidimensional Array

Syntax to declare a 1D Array:

```
dataType[] arrayName;
```

Example: `int[] nums;`

After declaring an array, we need to allocating memory to array it by specifying the number of elements it can hold.

Syntax: `arrayName= new dataType[size];`

Example: `nums= new int[5];`

Declaring and allocating memory to array in one line

Syntax: **`dataType[] arrayName= new dataType[size];`**

Example: `int[] numbers = new int[5];`

Array initialization

```
numbers[0]=1; // Sets the element at index 0 to value 1.  
numbers[1]=2;  
numbers[2]=3;  
numbers[3]=4;  
numbers[4]=5;
```

Array Initialization with Values:

Arrays can also be initialized with specific values using an initializer list enclosed in curly braces {}:

Example:

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

Accessing Array Elements

Array elements are accessed using zero-based indexing (starting from 0).

Example: `int[] nums = {1, 2, 3, 4, 5};`

`int myFavNum= nums[2] // Retrieves the element at index 2.`

Array Length

You can determine the length (number of elements) of an array using the length property:

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

`int length = nums.length; // gives the length of array`

Accessing Array Elements using for loop

We can loop through the array elements with the for loop, and use the length property to specify how many times the loop should run.

Example:

```
public class MyArray {  
    public static void main(String[] args) {  
        String[] cars = {"Ford", "BMW", "Audi", "Lexus"};  
        for (int i = 0; i < cars.length; i++) {  
            System.out.println(cars[i]);  
        }  
    }  
}
```

Multidimensional Array

- A multidimensional array is an array of arrays.
- Each element of a multidimensional array is an array itself.
- It allows you to store data in a tabular format consisting of rows and columns

Syntax to declare a 2D Array: `dataType[][] arrayName;`

Example: `int[][] matrix;`

Allocate memory to a 2D Array:

Syntax: `arrayName= new dataType[row][cols];`

Example: `matrix= new int[3][3];`

Declaring and allocating memory to a 2D array in one line

Syntax: `dataType[][] arrayName= new dataType[row][cols];`

`int[][] matrix = new int[3][3];`

2D Array initialization

`matrix[0][0] = 1;` // Sets the element at row 0, column 0 to 1.

Array Initialization with Values:

Arrays can also be initialized with specific values using an nested initializer list enclosed in curly braces {}:

Example:

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

Accessing Array Elements

Array elements are accessed using zero-based indexing. Both rows and cols starts with zero index(starting from 0).

Example:

```
int value = matrix[1][2]; // Retrieves the element at row 1, column 2
```

`array.length`: Gives the number of rows in the array.

`array[i].length` Gives the number of columns in the i-th row of the array.

Accessing 2D Array Elements using for loop

To access elements of 2D Array we use two loops one loop for iterating through the rows and another nested loop for iterating through the columns of each row.

Example:

```
public class MyArray2
{
    public static void main(String[] args)
    {
        int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
        for (int i = 0; i < matrix.length; i++)
        {
            for (int j = 0; j < matrix[i].length; j++)
            {
                System.out.print(matrix[i][j] + " "); // Access and print each element
            }
            System.out.println();
        }
    }
}
```