Java is a general, all-purpose computer programming language that is circumstantial, class-based, object-oriented, and specially designed to have few application dependencies as possible.

Java applications are compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture.

Java Programming Language was developed by the resolution of 5 great people, James Gosling, Patrick Naughton, Chris Warth, Mike Sheridan and Ed Frank but James Gosling is believed to be the inventor because he did the original design of Java and implemented its original compiler and virtual machine. They all operated for Sun Microsystems, Inc. and developed in 1991.

The language took 18 months to finish and possessed an original name as “Oak” which got renamed to Java in the year 1995, due to copyright matters.

Java versions

The Java language has experienced various modifications since JDK 1.0 as well as numerous inclusions of courses and units to the standard library. Since J2SE 1.4, the development of the Java language has been overseen by the Java Community Process (JCP), which utilizes Java Specification Requests (JSRs) to propose and designate extensions and modifications to the Java program.

The language is stipulated by the Java Language Specification (JLS); changes to the JLS are conducted under JSR 901.

Java 8 is the currently supported long-term-support (LTS) version, and Java 10 is the presently endorsed accelerated release version, as of March 20, 2018. Java 10 support ends on the same date that support for Java 11 begins, planned for September 2018, and **Java 11** will be the next LTS after **Java 8**.

Several java versions have been released, **and they are:**

- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan 1996)
- JDK 1.1 (19th Feb 1997)
- J2SE 1.2 (8th Dec 1998)
- J2SE 1.3 (8th May 2000)
- J2SE 1.4 (6th Feb 2002)
- J2SE 5.0 (30th Sep 2004)
- Java SE 6 (11th Dec 2006)
- Java SE 7 (28th July 2011)
- After Java SE 7 following versions have come and expected to come in future:

Version	Date	End of Free Public Updates	Extended Support Until
Java SE 8 (LTS)	18 March 2014	**January 2019 for Oracle (commercial)** December 2020 for Oracle (personal use) At least May 2026 for AdoptOpenJDK At least May 2026[7] for Amazon Corretto	**December 2030**
Java SE 11 (LTS)	September 2018	At least September 2027 for Amazon Corretto October 2024 for AdoptOpenJDK	**September 2026**
Java SE 12	March 2019	September 2019 for OpenJDK	N/A
Java SE 13	September 2019	March 2020 for OpenJDK	N/A
Java SE 14	March 2020	September 2020 for OpenJDK	N/A
Java SE 15	September 2020	March 2021 for OpenJDK	N/A
Java SE 16	March 2021	September 2021 for OpenJDK	N/A
Java SE 17 (LTS)	September 2021	To be announced	To be announced

JDK (Java Development Kit)

JDK contains everything that will be required to ***develop and run*** Java application.

JRE (Java Runtime Environment)

JRE contains everything required to ***run*** Java application which has already been compiled. It doesn't contain the code library required to develop Java application.

JVM (Java Virtual Machine)

JVM is a virtual machine which works on top of your operating system to provide a recommended environment for your compiled Java code. JVM only works with bytecode. Hence you need to compile your Java application(.java) so that it can be converted to bytecode format (also known as the .class file).

Which then will be used by JVM to run an application. JVM only provide the environment needed to executed Java Bytecode.

Java Portability

In order to understand portability in Java, you need to understand what happens to java code from start to finish.

- Java Source Code (Written by Developer) (Machine Neutral)
- Compiled Code / Byte Code (Compiled by javac). (Machine Neutral)
- Byte Code executed (Executed by JVM) (Machine Specific)

In step 2, javac (Java Compiler) converts Java code to bytecode. This can be moved to any machine(Windows / Linux) and executed by JVM. JVM reads the bytecode and generates machine specific code. In order to generate machine-specific code, JVM needs to be machine specific. So every type of Machine(Windows / Linux / Mac) has a specific JVM. So in this way, the coder doesn't need to bother with generating bytecode. JVM takes care of portability. So the final answer is Java is Portable but JVM is Machine Specific.

public static void main(String args[]) explanation

JVM will always look for a specific method signature to start running an application, and that would be **public static void main(String args[]).** Here args is an argument of the type String array. String array argument can also be written as **String[] args**. Though the type of the argument(String array) is fixed, you can still change the name from args to anything.

Also with the introduction of java args, instead of writing **String args[],** *String… args* can be used. Keep learning to know more about each and every keyword.

```
class JBT{
  public static void main(String args[])
   {
     System.out.println("Hello JBT");
   }
}
```

In the above application example, we are using the public static void main. Each word has a different meaning and purpose.

public

It is an Access Modifier, which defines who can access this Method. Public means that this Method will be accessible by any Class(If other Classes can access this Class.).

static

Static is a keyword that identifies the class-related thing. It means the given Method or variable is not instance-related but Class related. It can be accessed without creating the instance of a Class.

void

It is used to define the Return Type of the Method. It defines what the method can return. Void means the Method will not return any value.

main

main is the name of the Method**.** This Method name is searched by JVM as a starting point for an application with a particular signature only.

String args[] / String... args

It is the parameter to the main method. The argument name could be anything. You can either use String array (String args[]) or var args variable of String type. Both will work the same way.

Example:- Program to read the number entered by user.

For this we have to use a class Scanner from java.util package. In order to use this class we need to import this package as follows: -

```
import java.util.Scanner;

public class Demo{

        public static void main(String args[]) {

         Scanner obj = new Scanner(System.in);

       System.out.print("Enter any number from Keyboard :  ");

       // This method reads the number provided using keyboard

int num = obj.nextInt();  // Number from keyboard is gets stored in variable num

      // Closing Scanner object buffer after the use (Optional and not necessary)

       obj.close();

        // Displaying the number

       System.out.println("The number entered by You is :  "+num);

   }   //close function main

}    //close class Demo
```

Save this program by name **Demo.java** so that you can compile and run it easily as below:-

javac **Demo.java** → **Compiling**

java **Demo** → **Running**

Output :-

Output:

Enter any number from Keyboard : 765

The number entered by You is : 765

Variables in Java

A variable is a name which is associated with a value that can be changed. For example when I write int i=10; here variable name is i which is associated with value 10, int is a data type that represents that this variable can hold integer values. We will cover the data types in the next tutorial. In this tutorial, we will discuss about variables.

How to Declare a variable in Java

To declare a variable follow this syntax:

data_type variable_name = value;

here value is optional because in java, you can declare the variable first and then later assign the value to it.

For example: Here num is a variable and int is a data type. We will discuss the data type in next tutorial so do not worry too much about it, just understand that int data type allows this num variable to hold integer values. You can read data types here but I would recommend you to finish reading this guide before proceeding to the next one.

int num;

Similarly we can assign the values to the variables while declaring them, like this:

char ch = 'A';

int number = 100;

or we can do it like this:

char ch;

int number;

...

ch = 'A';

number = 100;

Variables naming convention in java

1) Variables naming cannot contain white spaces, for example: int num ber = 100; is invalid because the variable name has space in it.
2) Variable name can begin with special characters such as $ and _
3) As per the java coding standards the variable name should begin with a lower case letter, for example int number; For lengthy variables names that has more than one words do it like this: int smallNumber; int bigNumber; (start the second word with capital letter).
4) Variable names are case sensitive in Java.

Types of Variables in Java

There are **three types of variables** in Java.
1) Local variable 2) Static (or class) variable 3) Instance variable

Static (or class) Variable

Static variables are also known as class variable because they are associated with the class and common for all the instances of class. For example, If I create three objects of a class and access this static variable, it would be common for all, the changes made to the variable using one of the object would reflect when you access it through other objects.

Example of static variable

```
public class StaticVarExample {

   public static String myClassVar="class or static variable";

            public static void main(String args[]){

      StaticVarExample obj = new StaticVarExample();

      StaticVarExample obj2 = new StaticVarExample();

      StaticVarExample obj3 = new StaticVarExample();

           //All three will display "class or static variable"
```

```
        System.out.println(obj.myClassVar);
        System.out.println(obj2.myClassVar);
        System.out.println(obj3.myClassVar);
        //changing the value of static variable using obj2
        obj2.myClassVar = "Changed Text";
        //All three will display "Changed Text"
        System.out.println(obj.myClassVar);
        System.out.println(obj2.myClassVar);
        System.out.println(obj3.myClassVar);
    }
}
```

Output:

```
class or static variable
class or static variable
class or static variable
Changed Text
Changed Text
Changed Text
```

Instance variable

Each instance(objects) of class has its own copy of instance variable. Unlike static variable, instance variables have their own separate copy of instance variable. We have changed the instance variable value using object obj2 in the following program and when we displayed the variable using all three objects, only the obj2 value got changed, others remain unchanged. This shows that they have their own copy of instance variable.

Example of Instance variable

```
public class InstanceVarExample {
   String myInstanceVar="instance variable";
   public static void main(String args[]){
         InstanceVarExample obj = new InstanceVarExample();
         InstanceVarExample obj2 = new InstanceVarExample();
         InstanceVarExample obj3 = new InstanceVarExample();
         System.out.println(obj.myInstanceVar);
         System.out.println(obj2.myInstanceVar);
         System.out.println(obj3.myInstanceVar);
         obj2.myInstanceVar = "Changed Text";
         System.out.println(obj.myInstanceVar);
         System.out.println(obj2.myInstanceVar);
         System.out.println(obj3.myInstanceVar);
   }
}
```

Output:

```
instance variable
instance variable
instance variable
instance variable
Changed Text
instance variable
Local Variable
```

These variables are declared inside method of the class. Their scope is limited to the method which means that You can't change their values and access them outside of the method.

In this example, I have declared the instance variable with the same name as local variable, this is to demonstrate the scope of local variables.

Example of Local variable

```
public class VariableExample {
  // instance variable
  public String myVar="instance variable";
  public void myMethod(){
        // local variable
        String myVar = "Inside Method";
        System.out.println(myVar);
  }
  public static void main(String args[]){
    // Creating object
    VariableExample obj = new VariableExample();
    /* We are calling the method, that changes the
     * value of myVar. We are displaying myVar again after
     * the method call, to demonstrate that the local
     * variable scope is limited to the method itself.
     */
    System.out.println("Calling Method");
    obj.myMethod();
    System.out.println(obj.myVar);
```

```
  }
}
```

Output:

```
Calling Method
Inside Method
instance variable
```

Data Types in Java

Data type defines the values that a variable can take, for example if a variable has int data type, it can only take integer values. In java we have two categories of data type: 1) Primitive data types 2) Non-primitive data types – Arrays and Strings are non-primitive data types, we will discuss them later in the coming tutorials. Here we will discuss primitive data types and literals in Java.

Java is a statically typed language. A language is statically typed, if the data type of a variable is known at compile time. This means that you must specify the type of the variable (Declare the variable) before you can use it.

In the last tutorial about Java Variables, we learned how to declare a variable, lets recall it:

```
int num;
```

So in order to use the variable num in our program, we must declare it first as shown above. It is a good programming practice to declare all the variables (that you are going to use) in the beginning of the program.

1) Primitive data types

In Java, we have eight primitive data types: boolean, char, byte, short, int, long, float and double. Java developers included these data types to maintain the portability of java as the size of these primitive data types do not change from one operating system to another.

byte, short, int and long data types are used for storing whole numbers.

float and double are used for fractional numbers.

char is used for storing characters(letters).

boolean data type is used for variables that holds either true or false.

byte:

This can hold whole number between -128 and 127. Mostly used to save memory and when you are certain that the numbers would be in the limit specified by byte data type.

Default size of this data type: 1 byte.

Default value: 0

Example:

```
class JavaExample {
    public static void main(String[] args) {
        byte num;
        num = 113;
        System.out.println(num);
    }
}
```

Output:

113

Try the same program by assigning value assigning 150 value to variable num, you would get type mismatch error because the value 150 is out of the range of byte data type. The range of byte as I mentioned above is -128 to 127.

short:

This is greater than byte in terms of size and less than integer. Its range is -32,768 to 32767.

Default size of this data type: 2 byte

short num = 45678;

int: Used when short is not large enough to hold the number, it has a wider range: -2,147,483,648 to 2,147,483,647

Default size: 4 byte

Default value: 0

Example:

```
class JavaExample {
    public static void main(String[] args) {
        short num;
        num = 150;
        System.out.println(num);
    }
}
```

Output:

150

The byte data type couldn't hold the value 150 but a short data type can because it has a wider range.

long:

Used when int is not large enough to hold the value, it has wider range than int data type, ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

size: 8 bytes

Default value: 0

Example:

```
class JavaExample {
    public static void main(String[] args) {
        long num = -12332252626L;
        System.out.println(num);
    }
}
```

Output:

-12332252626

double: Sufficient for holding 15 decimal digits

size: 8 bytes

Example:

```
class JavaExample {
    public static void main(String[] args) {
        double num = -42937737.9d;
        System.out.println(num);
    }
}
```

Output:

-4.29377379E7

float: Sufficient for holding 6 to 7 decimal digits

size: 4 bytes

```
class JavaExample {
```

```
    public static void main(String[] args) {
        float num = 19.98f;
        System.out.println(num);
    }
}
```

Output:

19.98

boolean: holds either true of false.

```
class JavaExample {
    public static void main(String[] args) {
        boolean b = false;
        System.out.println(b);
    }
}
```

Output:

false

char: holds characters.

size: 2 bytes

```
class JavaExample {
    public static void main(String[] args) {
        char ch = 'Z';
        System.out.println(ch);
    }
```

}

Output:

Z

Literals in Java

A literal is a fixed value that we assign to a variable in a Program.

int num=10;

Here value 10 is a Integer literal.

char ch = 'A';

Here A is a char literal

Integer Literal

Integer literals are assigned to the variables of data type byte, short, int and long.

byte b = 100;

short s = 200;

int num = 13313131;

long l = 928389283L;

Float Literals

Used for data type float and double.

double num1 = 22.4;

float num2 = 22.4f;

Note: Always suffix float value with the “f” else compiler will consider it as double.

Char and String Literal

Used for char and String type.

char ch = 'Z';

String str = "BeginnersBook";

Operators in Java

An operator is a character that represents an action, for example + is an arithmetic operator that represents addition.

Types of Operator in Java

1) Basic Arithmetic Operators

2) Assignment Operators

3) Auto-increment and Auto-decrement Operators

4) Logical Operators

5) Comparison (relational) operators

6) Bitwise Operators

7) Ternary Operator

1) Basic Arithmetic Operators

Basic arithmetic operators are: +, -, *, /, %

+ is for addition.

– is for subtraction.

* is for multiplication.

/ is for division.

% is for modulo.

Note: Modulo operator returns remainder, for example 10 % 5 would return 0

Example of Arithmetic Operators

```
public class ArithmeticOperatorDemo {
   public static void main(String args[]) {
      int num1 = 100;
      int num2 = 20;
      System.out.println("num1 + num2: " + (num1 + num2) );
      System.out.println("num1 - num2: " + (num1 - num2) );
      System.out.println("num1 * num2: " + (num1 * num2) );
      System.out.println("num1 / num2: " + (num1 / num2) );
      System.out.println("num1 % num2: " + (num1 % num2) );
   }
}
```

Output:

```
num1 + num2: 120
num1 - num2: 80
num1 * num2: 2000
num1 / num2: 5
num1 % num2: 0
```

Thanks for today.

Any Queries can be asked on whatsapp.

Next Volume of Revision will be in the next Lecture.