INTRODUCTION TO JAVA

➲ *JAVA History:*

Java is certainly a good programming language. There is no doubt that it is one of the better language available to serious programmers. Java is a general-purpose, object-oriented programming language developed by Sun Microsystems of USA in 1991. Originally called ***Oak*** by James Gosling, one of the inventors of the language, Java was designed for the development of software for consumer electronics devices like TVs, VCRs, toasters and such other electronic machines. This goal had a strong impact on the development team to make the language simple, portable and highly reliable. The Java team, which included Patrick Naughton, discovered that the existing languages like C and C++ had limitations in terms of both reliability and portability. However, they modeled the new language Java on C and C++ but removed a number of features of C and C++ that were considered as sources of problems and thus made Java a really simple, reliable, portable, powerful language.

➲ *Advantage of JAVA:*

One obvious advantage is a runtime environment that provides platform independence: you can use the same code on Windows, Solaris, Linux, Macintosh, and so on. This is certainly necessary when programs are downloaded over the Internet to run on a variety of platforms.

Another programming advantage is that Java has a syntax similar to that of C++, making it easy for C and C++ programmers to learn

Java is also fully object oriented — even more so than C++. Everything in Java, except for a few basic types like numbers, is an object. (Object-oriented programming has replaced earlier structured techniques because it has many advantages for dealing with sophisticated projects

The key point is this: *It is far easier to turn out bug-free code using Java than using* C++.

They added features to Java that *eliminate the possibility* of creating code with the most common kinds of bugs.

- The Java designers eliminated manual memory allocation and deallocation.

  Memory in Java is automatically garbage collected. You *never* have to worry about memory corruption.
- They introduced true arrays and eliminated pointer arithmetic.
- They eliminated multiple inheritance, replacing it with a new notation of *interface*.

➲ *Features of JAVA:*

The inventors of Java wanted to design a language which could offer solutions to some of the problems encountered in modern programming. They wanted the language to be not only reliable, portable and distributed but also simple, compact and interactive. Sun Microsystems officially describes Java with the following attributes:

- **Simple, Small and Familiar**

  Java is a small and simple language. There is no need for header files, pointer arithmetic, structures, unions, operator overloading, virtual base class and so on.

  Familiarity is another striking feature of Java. To make the language look familiar to the existing programmers, it was modelled on C and C++ languages. Java uses many constructs of C and C++ and therefore, Java code "looks like a C+ + " code. In fact, Java is a simplified version of C+ +.

- **Compiled and Interpreted**

  Usually a computer language is either compiled or interpreted. Java combines both these approaches thus making Java a two-stage system. First, Java compiler translates source code into what is known as *bytecode* instructions. Bytecodes are not machine instructions and therefore, in the second stage, Java interpreter generates machine code, that can be directly executed by the machine that is running the Java program. We can thus say that Java is both a compiled and an interpreted language.

- **Platform-Independent and Portable**

  The most significant contribution of Java over other languages is its portability. Java programs can be easily moved from one computer system to another, anywhere and anytime. Changes and upgrades in operating systems, processors and system resources will not force any changes in Java programs. This is the reason why Java has become a popular language for programming on Internet which interconnects different kinds of systems worldwide. We can download a Java applet from a remote computer onto our local system via Internet and execute it locally. This makes the Internet an extension of the user's basic system providing practically unlimited number of accessible applets and applications.

  Java ensures portability in two ways. First, Java compiler generates bytecode instructions that can be implemented on any machine. Secondly, the size of the primitive data types are machine-independent.

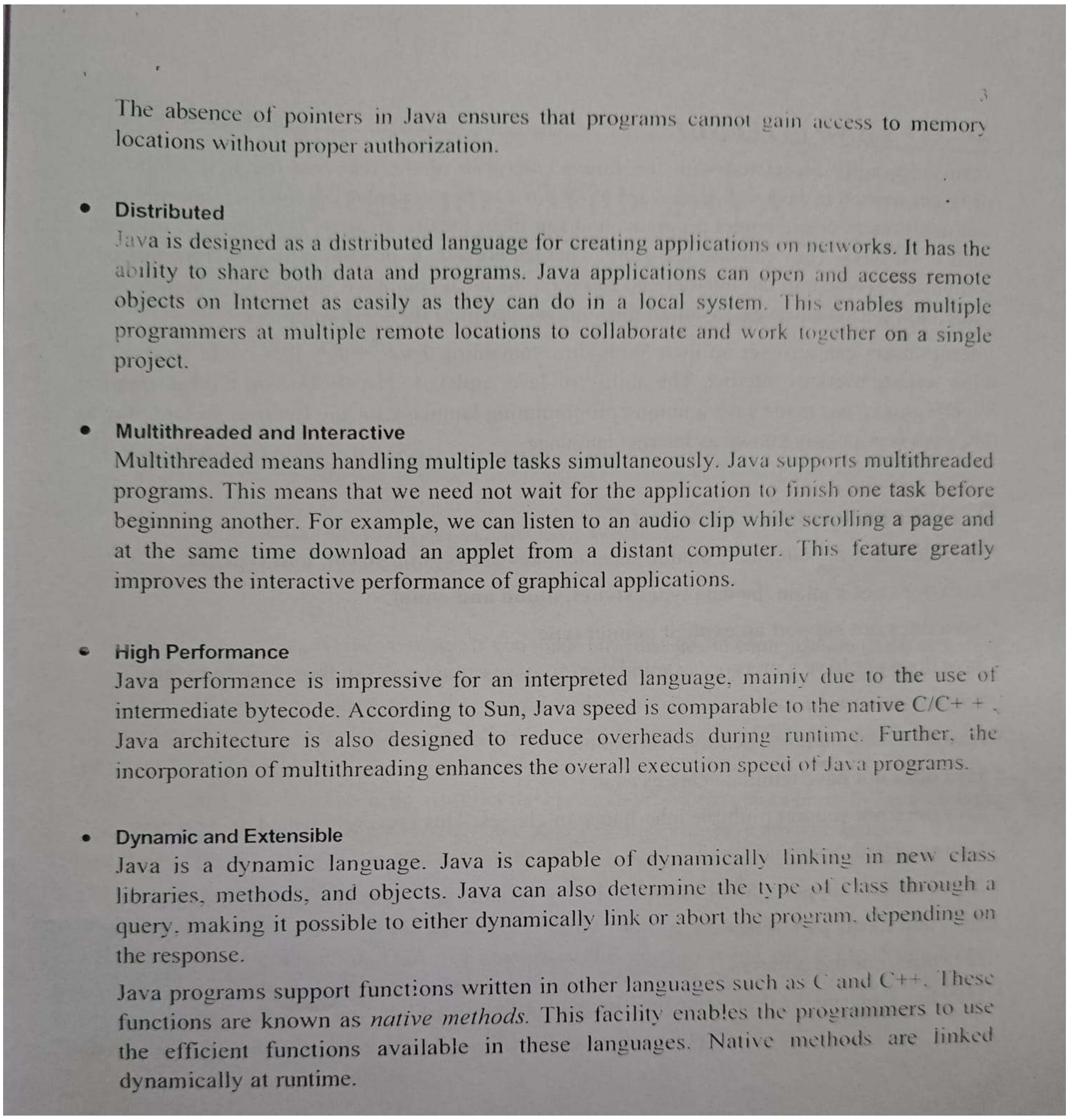
- **Object-Oriented**

  Java is a true object-oriented language. Almost everything in Java is an *object.* All program code and data reside within objects and classes. Java comes with an extensive set of *classes,* arranged in *packages,* that we can use in our programs by inheritance. The object model in Java is simple and easy to extend.

- **Robust and Secure**

  Java is a robust language. It provides many safeguards to ensure reliable code. It has strict compile time and run time checking for data types. Java also incorporates the concept of exception handling which captures series errors and eliminates any risk of crashing the system.

  Security becomes an important issue for a language that is used for programming on Internet. Threat of viruses and abuse of resources is everywhere. Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet.


The absence of pointers in Java ensures that programs cannot gain access to memory locations without proper authorization.

- **Distributed**
  Java is designed as a distributed language for creating applications on networks. It has the ability to share both data and programs. Java applications can open and access remote objects on Internet as easily as they can do in a local system. This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

- **Multithreaded and Interactive**
  Multithreaded means handling multiple tasks simultaneously. Java supports multithreaded programs. This means that we need not wait for the application to finish one task before beginning another. For example, we can listen to an audio clip while scrolling a page and at the same time download an applet from a distant computer. This feature greatly improves the interactive performance of graphical applications.

- **High Performance**
  Java performance is impressive for an interpreted language, mainly due to the use of intermediate bytecode. According to Sun, Java speed is comparable to the native C/C++. Java architecture is also designed to reduce overheads during runtime. Further, the incorporation of multithreading enhances the overall execution speed of Java programs.

- **Dynamic and Extensible**
  Java is a dynamic language. Java is capable of dynamically linking in new class libraries, methods, and objects. Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program, depending on the response.

  Java programs support functions written in other languages such as C and C++. These functions are known as *native methods*. This facility enables the programmers to use the efficient functions available in these languages. Native methods are linked dynamically at runtime.

JDK -

JDK stands for Java Development Kit. It is a software development kit used for developing Java applications and applets. The JDK includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools needed for Java development.

Here's a breakdown of some key components within the JDK:

Java Runtime Environment (JRE): This includes the Java Virtual Machine (JVM) and the libraries necessary for running Java applications. If you only want to run Java applications, you need the JRE.

Compiler (javac): This is used to compile Java source code (.java files) into bytecode (.class files), which can be executed by the JVM.

Java Virtual Machine (JVM): It is an abstract machine that provides an execution environment for Java bytecode. It interprets and executes the bytecode generated by the Java compiler.

Archiver (jar): This tool is used to package Java classes and resources into a single compressed file called a JAR (Java Archive) file. JAR files are used for packaging and distributing Java applications and libraries.

Documentation Generator (Javadoc): Javadoc is a tool that generates HTML documentation from comments in Java source code. It helps developers create and maintain well-documented code.

JRE-

JRE stands for Java Runtime Environment. It is a set of software tools and libraries that provides the runtime environment needed for executing Java applications and running Java applets. The JRE includes the Java Virtual Machine (JVM), class libraries, and other supporting files required for the execution of Java programs.

Here are the key components of the Java Runtime Environment:

Java Virtual Machine (JVM): This is a virtualized computer that executes Java bytecode. It is responsible for interpreting or compiling Java bytecode into machine code and managing the execution of Java programs.

Java Class Libraries: These are pre-compiled classes and methods that provide a wide range of functionality to Java applications. The class libraries cover areas such as data structures, networking, file handling, graphical user interface (GUI), and more.

Java Deployment Technologies: The JRE includes technologies like Java Web Start and Java Plug-in, which enable the deployment of Java applications on the web and integration with web browsers.

While the JRE is sufficient for running Java applications, it does not include the tools needed for Java development, such as compilers and other development utilities. For Java development, you would need the Java Development Kit (JDK), which includes the JRE along with additional tools for compiling, debugging, and building Java applications.

JIT

When we write a program in any programming language it requires converting that code in the machine-understandable form because the machine only understands the binary language. According to the programming languages, compiler differs. The compiler is a program that converts the high-level language to machine level code. The Java programming language uses the compiler named javac. It converts the high-level language code into machine code (bytecode). JIT is a part of the JVM that optimizes the performance of the application. JIT stands for Java-In-Time Compiler. The JIT compilation is also known as dynamic compilation. In this section, we will learn what is JIT in Java, its working, and the phases of the JIT compiler.

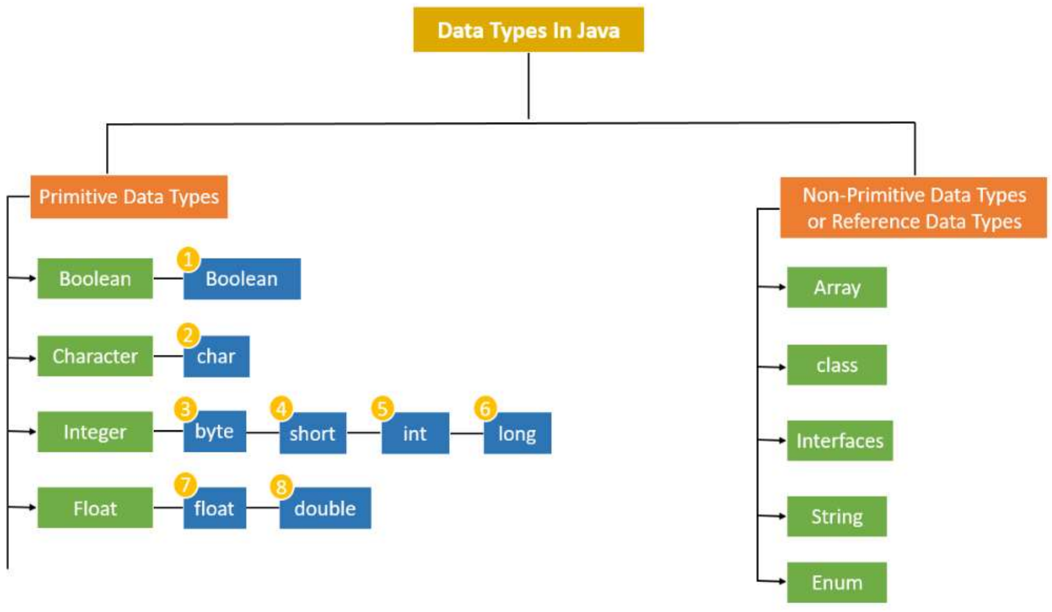
What is JIT in Java?

JIT in Java is an integral part of the JVM. It accelerates execution performance many times over the previous level. In other words, it is a long-running, computer-intensive program that provides the best performance environment. It optimizes the performance of the Java application at compile or run time.

Data Types in Java-

Data types in Java are divided into 2 categories:

- **Primitive Data Types**
- **Non-Primitive Data Types**



Primitive Data Types

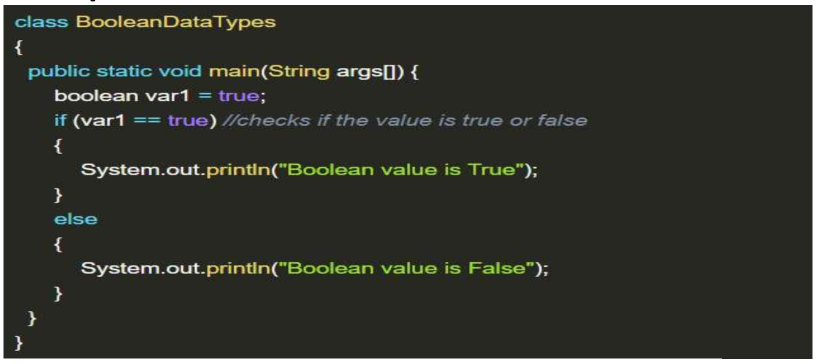
Primitive data types specify the size and type of variable values. They are the building blocks of data manipulation and cannot be further divided into simpler data types.

There are 8 Primitive data types in Java – Boolean, char, byte, int, short, long, float, and double.

Boolean –

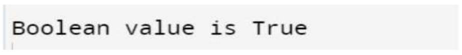
A boolean data type can store either True or False. They can be used to check whether two values are equal or not (basically in conditional statements to return True or False). Typically, programmers use it as a flag variable to track true or false conditions.The **default boolean value is False**.

Example-



```
class BooleanDataTypes
{
  public static void main(String args[]) {
      boolean var1 = true;
      if (var1 == true) //checks if the value is true or false
      {
         System.out.println("Boolean value is True");
      }
      else
      {
         System.out.println("Boolean value is False");
      }
   }
}
```

Output-

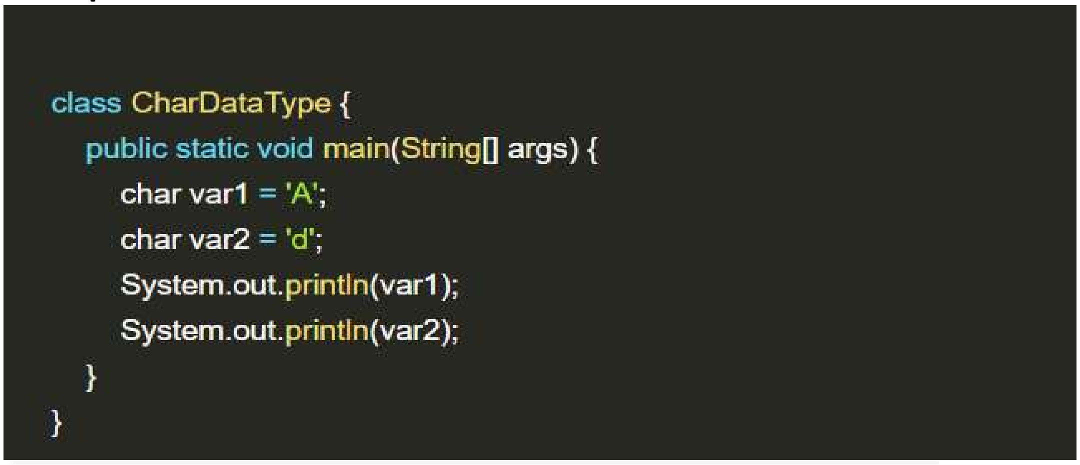


```
Boolean value is True
```

<u>Character type – char</u>

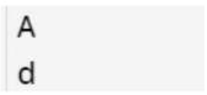
The char data type **stores a single character**. It stores lowercase and uppercase characters, which must be enclosed in single quotes. The char data type in Java supports Unicode characters and provides provision to multiple languages like English, French, German, etc. It takes **memory space of 16 bits or 2 bytes**. The values stored **range between 0 to 65536.**

Example:-



```java
class CharDataType {
    public static void main(String[] args) {
        char var1 = 'A';
        char var2 = 'd';
        System.out.println(var1);
        System.out.println(var2);
    }
}
```

Output:



```
A
d
```

<u>Integer type –</u>

An integer type stores an integer number with no fractional or decimal places. Java has four integer types – byte, short, int, and long.

<u>Byte</u>

The byte is the smallest data type among all the integer data types. It is an 8-bit signed two's complement integer. It stores whole numbers ranging from -128 to 127.

Syntax:
byte byteVariable;

<u>Short</u>

Short is a 16-bit signed two's complement integer. It stores whole numbers with values ranging from -32768 to 32767. Its default value is 0.

Syntax:
short shortVariable;

Int

Int is a 32-bit signed two's complement integer that stores integral values ranging from 2147483648 (-2^31) to 2147483647 (2^31 -1). Its default value is 0.

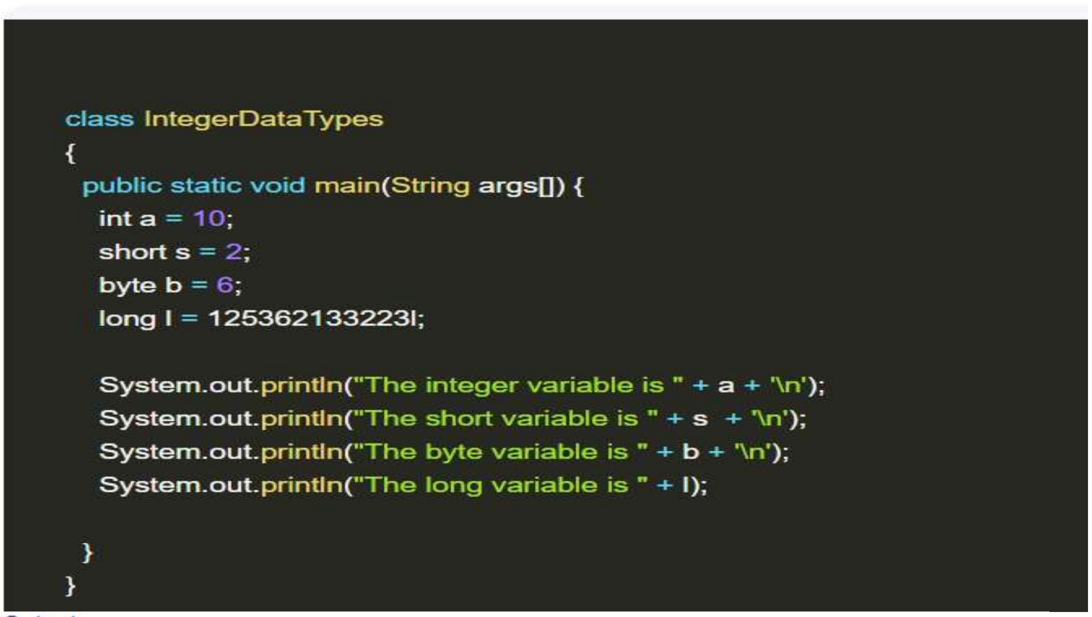
Syntax:
int intVariable;

Long

long is a 64-bit signed two's complement integer that stores values ranging from -9223372036854775808(-2^63) to 9223372036854775807(2^63 -1). It is used when we need a range of values more than those provided by int. Its default value is 0L. This data type ends with 'L' or 'l'.

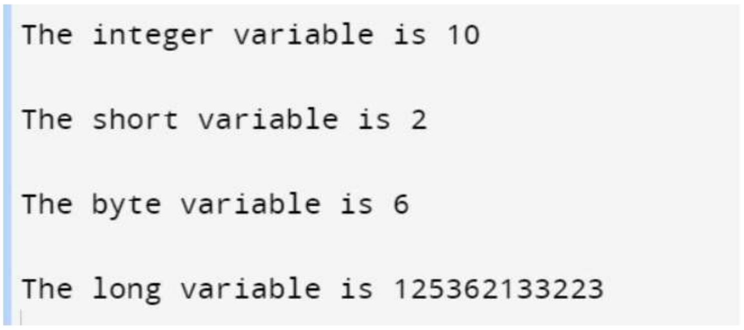
Syntax:
long longVariable;

Example:



```
class IntegerDataTypes
{
  public static void main(String args[]) {
    int a = 10;
    short s = 2;
    byte b = 6;
    long l = 125362133223l;

    System.out.println("The integer variable is " + a + '\n');
    System.out.println("The short variable is " + s  + '\n');
    System.out.println("The byte variable is " + b + '\n');
    System.out.println("The long variable is " + l);

  }
}
```

Output:



```
The integer variable is 10

The short variable is 2

The byte variable is 6

The long variable is 125362133223
```

Float type –

Floating-point is used for expressions involving fractional precision. It has two types: float and double.

Float

It is a floating-point data type that stores the values, including their decimal precision. It is not used for precise data such as currency or research data.

A Float value:

- is a single-precision **32-bit or 4 bytes** IEEE 754 floating-point
- can have a 7-digit decimal precision
- ends with an ‘f’ or ‘F’
- default value = 0.0f

Syntax:

```
float floatVariable;
```

Double

The double data type is similar to float. The difference between the two is that is double twice the float in the case of decimal precision. It is used for decimal values just like float and should not be used for precise values.

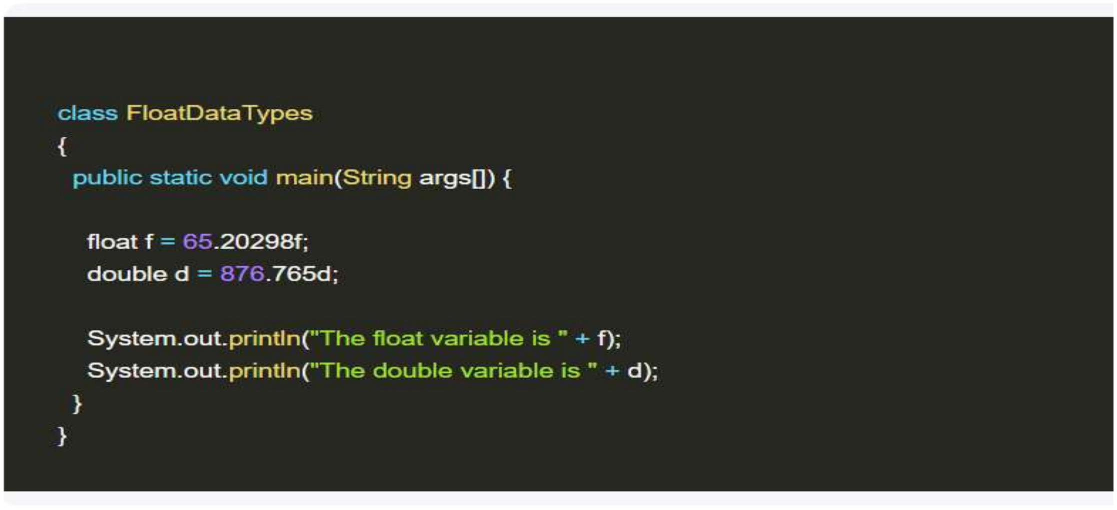
A double value:

- is a double-precision 64-bit or 8 bytes IEEE 754 floating-point
- can have a 15-digit decimal precision
- default value = 0.0d

Syntax:

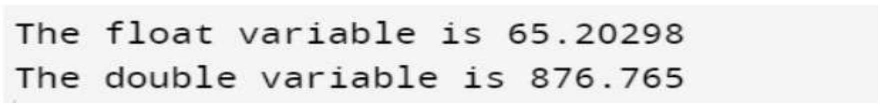
```
double doubleVariable;
```

Example:



```
class FloatDataTypes
{
  public static void main(String args[]) {

    float f = 65.20298f;
    double d = 876.765d;

    System.out.println("The float variable is " + f);
    System.out.println("The double variable is " + d);
  }
}
```

Output:



```
The float variable is 65.20298
The double variable is 876.765
```

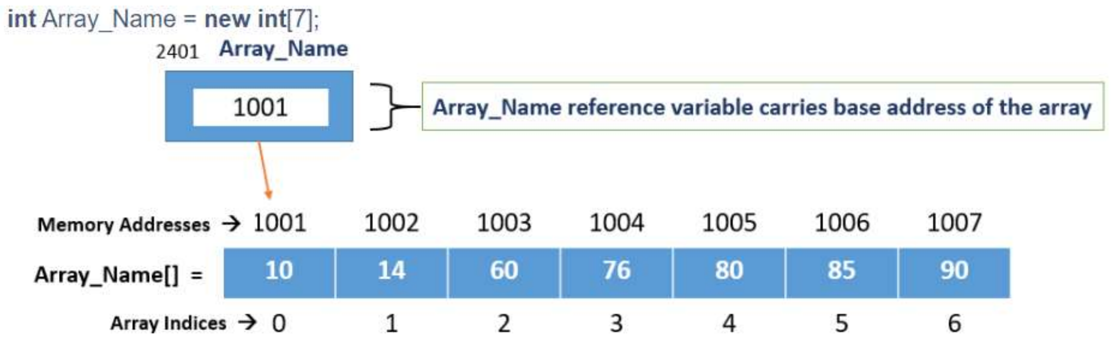
Non-Primitive Data Types

Non-primitive data types or reference data types refer to instances or objects. They cannot store the value of a variable directly in memory. They store a memory address of the variable. Unlike primitive data types we define by Java, non-primitive data types are user-defined. Programmers create them and can be assigned with null. All non-primitive data types are of equal size.

Array

An array holds elements of the same type. It is an object in Java, and the array name (used for declaration) is a reference value that carries the base address of the continuous location of elements of an array.

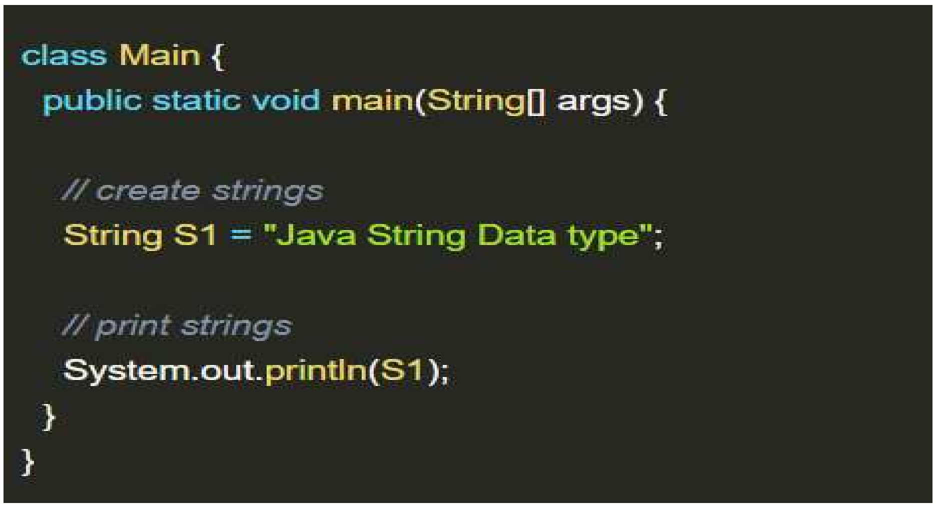
Example:



int Array_Name = new int[7];
2401
Array_Name
1001
Array_Name reference variable carries base address of the array
Memory Addresses → 1001 1002 1003 1004 1005 1006 1007
Array_Name[] = 10 14 60 76 80 85 90
Array Indices → 0 1 2 3 4 5 6

String

The String data type stores a sequence or array of characters. A string is a non-primitive data type, but it is predefined in Java. String literals are enclosed in double quotes.



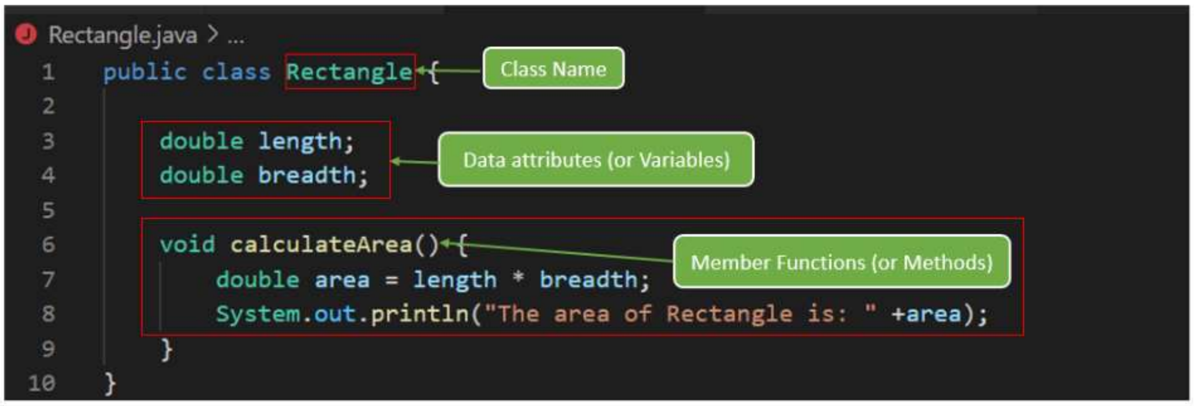
```
class Main {
  public static void main(String[] args) {

    // create strings
    String S1 = "Java String Data type";

    // print strings
    System.out.println(S1);
  }
}
```

Class

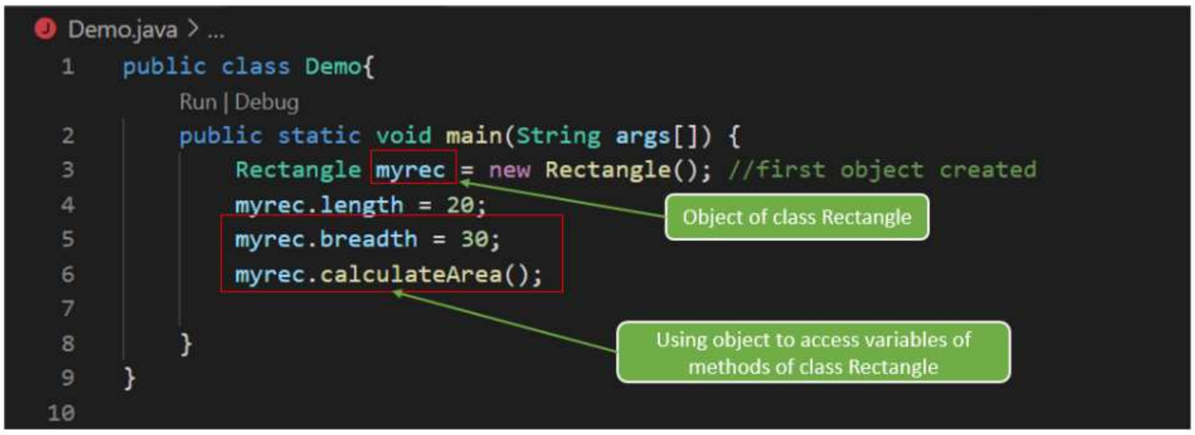
A class is a user-defined data type from which objects are created. It describes the set of properties or methods common to all objects of the same type. It contains fields and methods that represent the behaviour of an object. A class gets invoked by the creation of the respective object.

There are two types of classes: a blueprint and a template. **For instance**, the architectural diagram of a building is a **class**, and the building itself is an **object** created using the architectural diagram.

Example:


Rectangle.java > ...

```java
public class Rectangle{

    double length;
    double breadth;

    void calculateArea(){
        double area = length * breadth;
        System.out.println("The area of Rectangle is: " +area);
    }
}
```

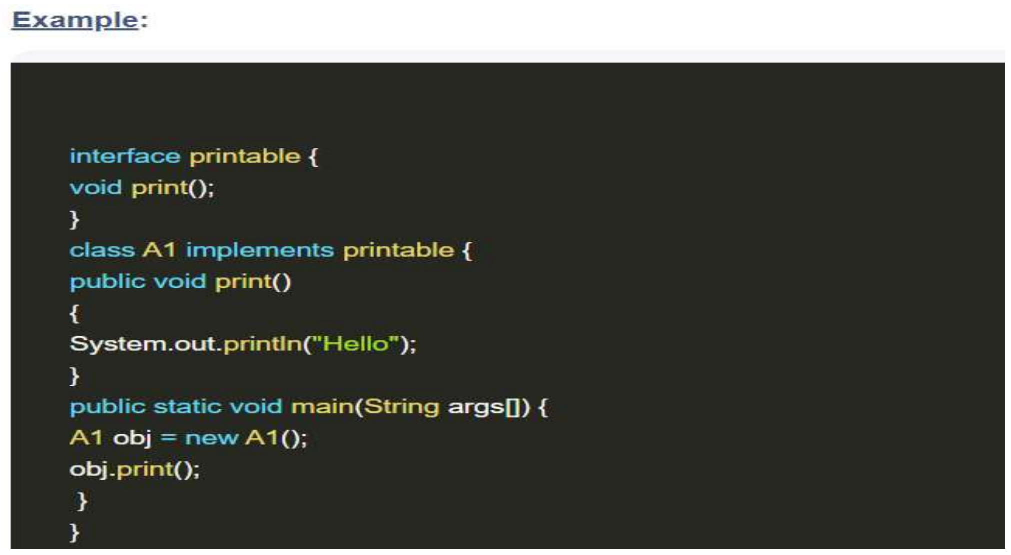

Demo.java > ...

```java
public class Demo{
    Run | Debug
    public static void main(String args[]) {
        Rectangle myrec = new Rectangle(); //first object created
        myrec.length = 20;
        myrec.breadth = 30;
        myrec.calculateArea();

    }
}
```

Object of class Rectangle
Using object to access variables of methods of class Rectangle

Interface

An interface is declared like a class. The key difference is that the interface contains abstract methods by default; they have nobody.


Example:

```java
interface printable {
void print();
}
class A1 implements printable {
public void print()
{
System.out.println("Hello");
}
public static void main(String args[]) {
A1 obj = new A1();
obj.print();
 }
}
```

Enum

An enum, similar to a class, has attributes and methods. However, unlike classes, enum constants are public, static, and final (unchangeable – cannot be overridden). Developers cannot use an enum to create objects, and it cannot extend other classes. But, the enum can implement interfaces.

```
//declaration of an enum
enum Level {
  LOW,
  MEDIUM,
  HIGH
}
```

Variables

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.
A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed.

Types of Variables

There are three types of variables in Java:

- local variable
- instance variable
- static variable

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Comments

The Java comments are the statements in a program that are not executed by the compiler and interpreter.

Types of Java Comments

There are three types of comments in Java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

1) Java Single Line Comment

The single-line comment is used to comment only one line of the code. It is the widely used and easiest way of commenting the statements.

Single line comments starts with two forward slashes **(//)**. Any text in front of // is not executed by Java.

Syntax:

1. //This is single line comment

2) Java Multi Line Comment

The multi-line comment is used to comment multiple lines of code. It can be used to explain a complex code snippet or to comment multiple lines of code at a time (as it will be difficult to use single-line comments there).

Multi-line comments are placed between /* and */. Any text between /* and */ is not executed by Java.

Syntax:

```
/*
This
is
multi line
comment
*/
```

3) Java Documentation Comment

Documentation comments are usually used to write large programs for a project or software application as it helps to create documentation API. These APIs are needed for reference, i.e., which classes, methods, arguments, etc., are used in the code.

To create documentation API, we need to use the **javadoc tool**. The documentation comments are placed between /** and */.

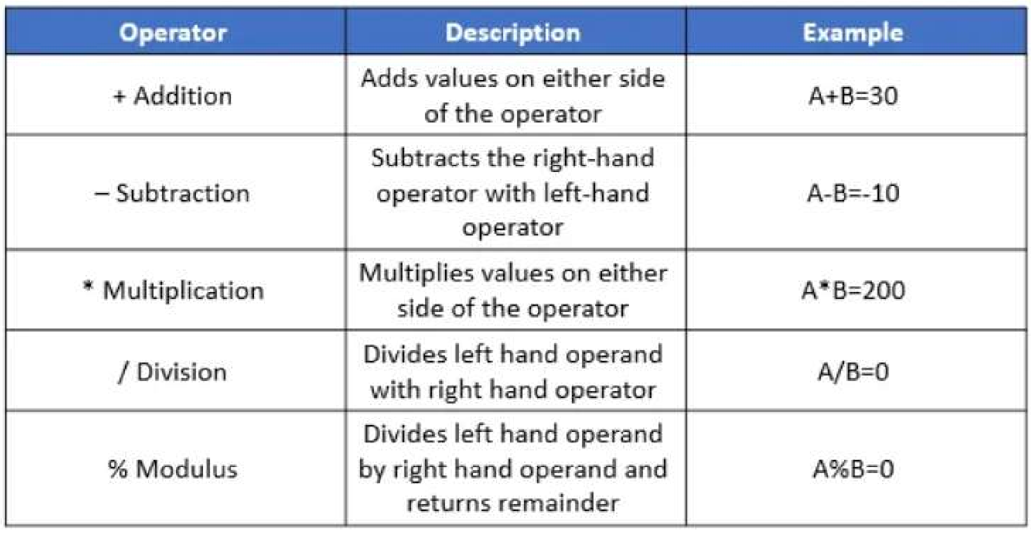
Syntax:

```
/**
*
*We can use various tags to depict the parameter
*or heading or author name
*We can also use HTML tags
*
*/

```

Operators

Arithmetic Operators in Java

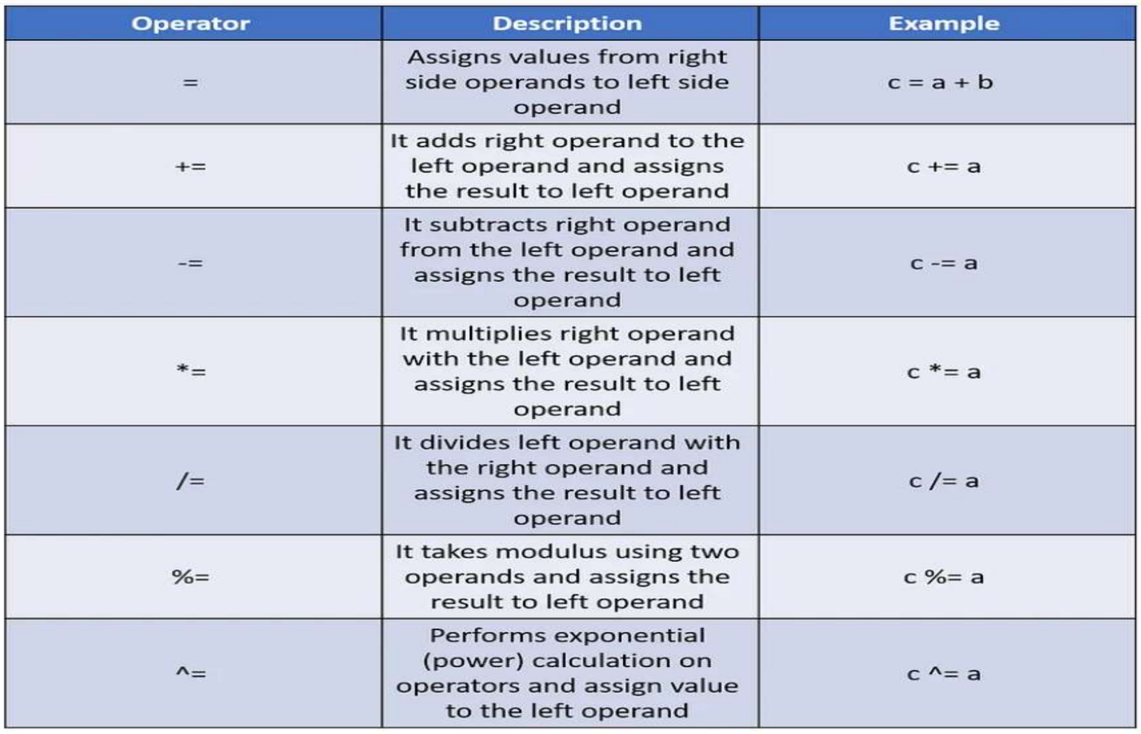
Arithmetic Operators are used to performing mathematical operations like addition, subtraction, etc. Assume that A = 10 and B = 20 for the below table.



Operator	Description	Example
+ Addition	Adds values on either side of the operator	A+B=30
– Subtraction	Subtracts the right-hand operator with left-hand operator	A-B=-10
* Multiplication	Multiplies values on either side of the operator	A*B=200
/ Division	Divides left hand operand with right hand operator	A/B=0
% Modulus	Divides left hand operand by right hand operand and returns remainder	A%B=0

Assignment Operators in Java

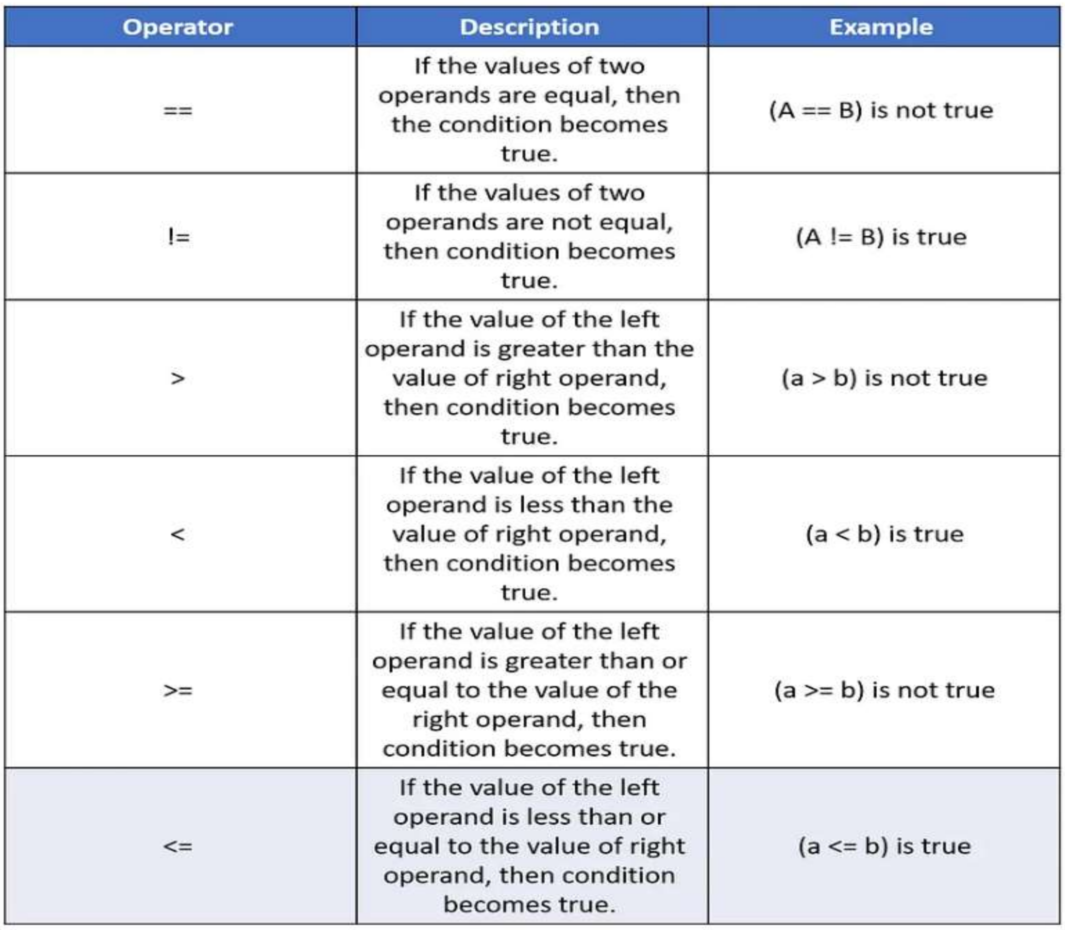
An *Assignment Operator* is an *operator* used to *assign* a new value to a variable. Assume A = 10 and B = 20 for the below table.



Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b
+=	It adds right operand to the left operand and assigns the result to left operand	c += a
-=	It subtracts right operand from the left operand and assigns the result to left operand	c -= a
*=	It multiplies right operand with the left operand and assigns the result to left operand	c *= a
/=	It divides left operand with the right operand and assigns the result to left operand	c /= a
%=	It takes modulus using two operands and assigns the result to left operand	c %= a
^=	Performs exponential (power) calculation on operators and assign value to the left operand	c ^= a

Relational Operators in Java

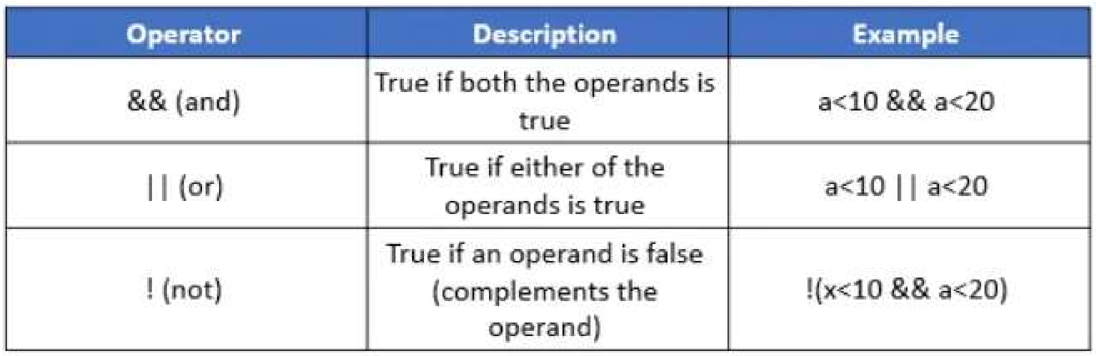
These operators compare the values on either side of them and decide the relation among them. Assume A = 10 and B = 20.



Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(A == B) is not true
!=	If the values of two operands are not equal, then condition becomes true.	(A != B) is true
>	If the value of the left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true
<	If the value of the left operand is less than the value of right operand, then condition becomes true.	(a < b) is true
>=	If the value of the left operand is greater than or equal to the value of the right operand, then condition becomes true.	(a >= b) is not true
<=	If the value of the left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true

Logical Operators in Java

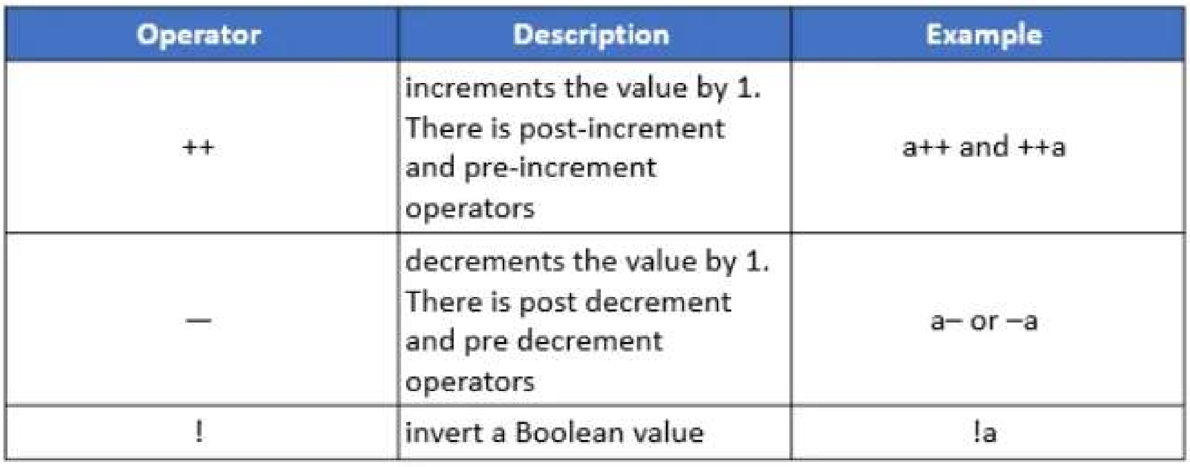
The following are the Logical operators present in Java:



Operator	Description	Example
&& (and)	True if both the operands is true	a<10 && a<20
|| (or)	True if either of the operands is true	a<10 || a<20
! (not)	True if an operand is false (complements the operand)	!(x<10 && a<20)

Unary Operator in Java

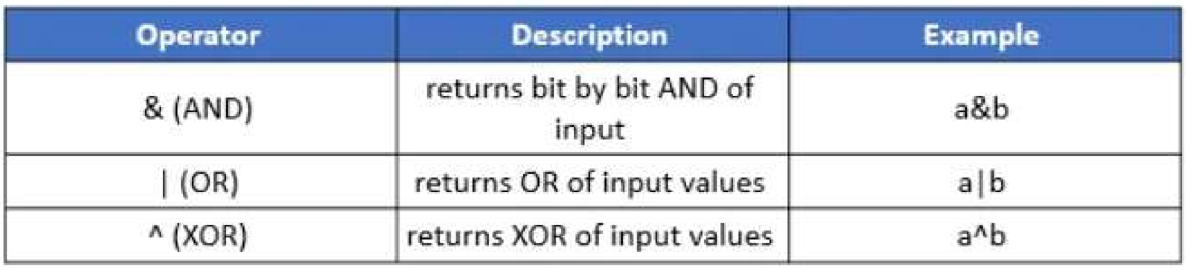
Unary operators are the one that needs a single operand and are used to increment a value, decrement or negate a value.



Operator	Description	Example
++	increments the value by 1. There is post-increment and pre-increment operators	a++ and ++a
—	decrements the value by 1. There is post decrement and pre decrement operators	a– or –a
!	invert a Boolean value	!a

Bitwise Operator in Java

Bitwise operations directly manipulate **bits**. In all computers, numbers are represented with bits, a series of zeros and ones. In fact, pretty much everything in a computer is represented by bits. Assume that A = 10 and B = 20 for the below table.



Operator	Description	Example
& (AND)	returns bit by bit AND of input	a&b
| (OR)	returns OR of input values	a|b
^ (XOR)	returns XOR of input values	a^b

Java provides three types of control flow statements.

1. Decision Making statements
   - if statements
   - switch statement

2. Loop statements
    - do while loop
    - while loop
    - for loop
    - for-each loop
3. Jump statements
    - break statement
    - continue statement

1) If Statement:

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

1. Simple if statement
2. if-else statement
3. if-else-if ladder
4. Nested if-statement

Let's understand the if-statements one by one.

1) Simple if statement:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax of if statement is given below.

```
if(condition) {
statement 1; //executes when condition is true
}
```

2) if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

```
if(condition) {
statement 1; //executes when condition is true
}
else{
statement 2; //executes when condition is false
}
```

3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax of if-else-if statement is given below.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false
}
```

4. Nested if-statement

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

Syntax of Nested if-statement is given below.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
if(condition 2) {
statement 2; //executes when condition 2 is true
}
else{
statement 2; //executes when condition 2 is false
}
}

```

Switch Statement:

In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

The syntax to use the switch statement is given below.

```
switch (expression){
    case value1:
     statement1;
     break;
    .
    .
    .
    case valueN:
     statementN;
     break;
    default:
     default statement;
}
```

Loop Statements

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

1. for loop
2. while loop
3. do-while loop

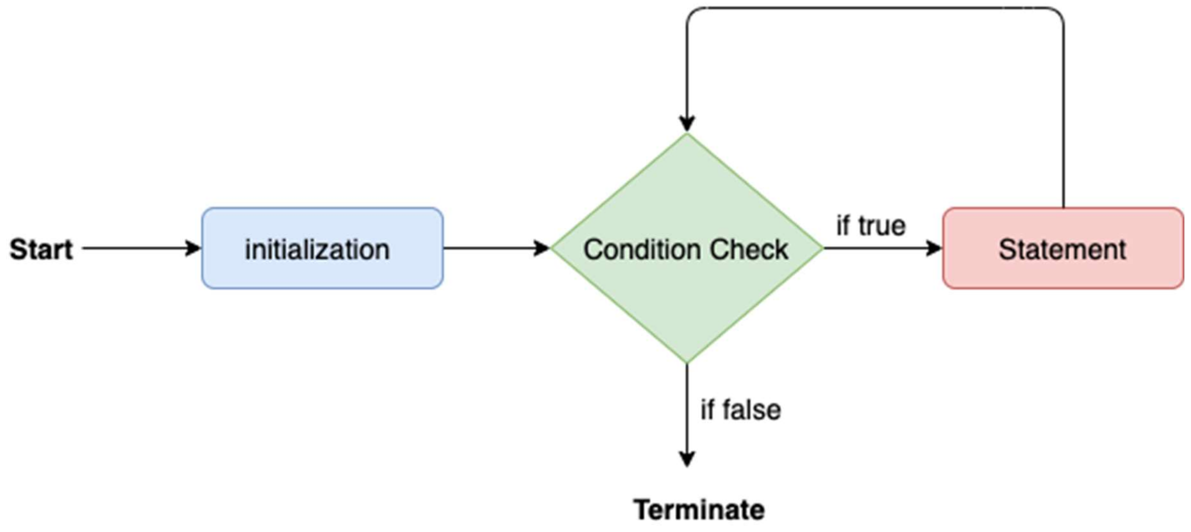
Let's understand the loop statements one by one.

Java for loop

In Java, for loop is similar to C and C++. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

```
for(initialization, condition, increment/decrement) {
//block of statements
}
```

The flow chart for the for-loop is given below.



Java while loop

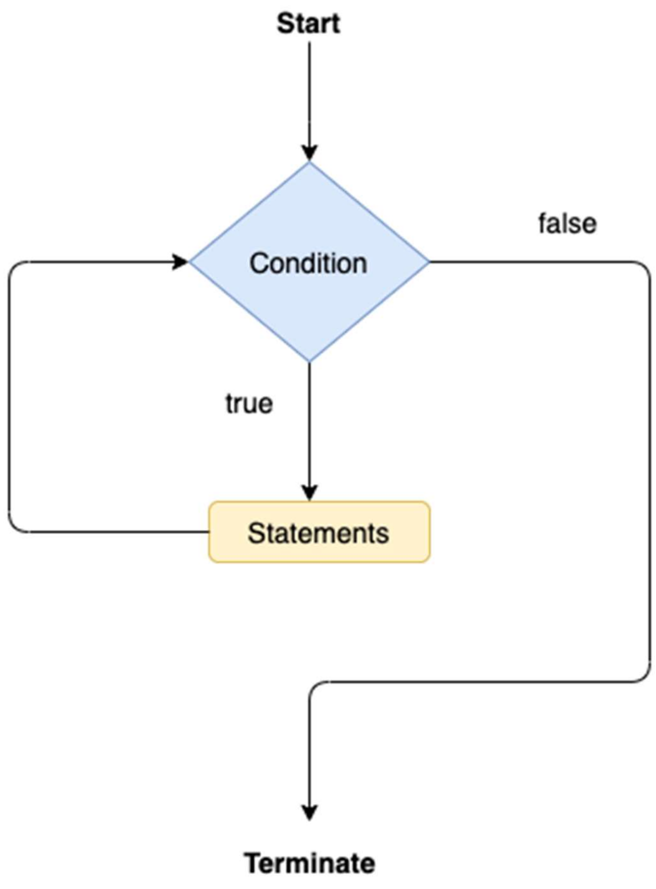
The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.

It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

The syntax of the while loop is given below.

```
while(condition){
//looping statements
}
```

The flow chart for the while loop is given in the following image.



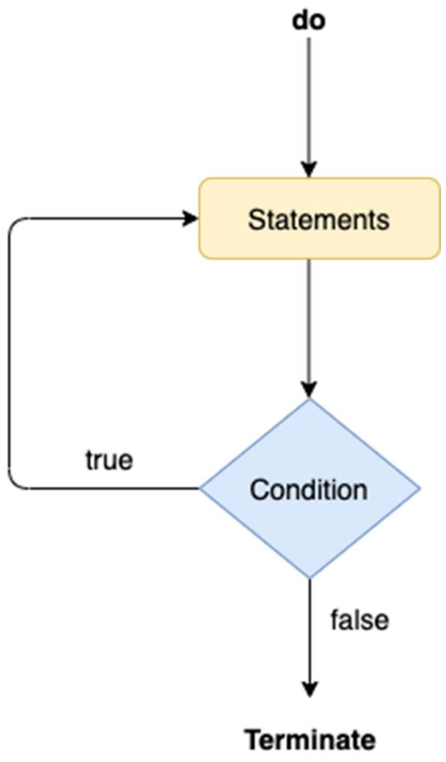
Java do-while loop

The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

```
do
{
//statements
} while (condition);
```

The flow chart of the do-while loop is given in the following image.



Compiling and running java programs using command line

Step 1:

Write a program on the notepad and save it with **.java** (for example, DemoFile.java) extension.

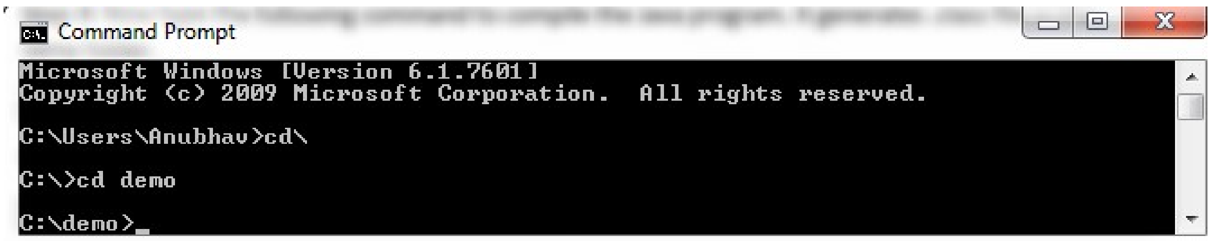
```
class DemoFile
{
public static void main(String args[])
{
System.out.println("Hello!");
System.out.println("Java");
}
}
```

Step 2:

Open Command Prompt.

Step 3:

Set the directory in which the .java file is saved. In our case, the .java file is saved in C:\\demo.

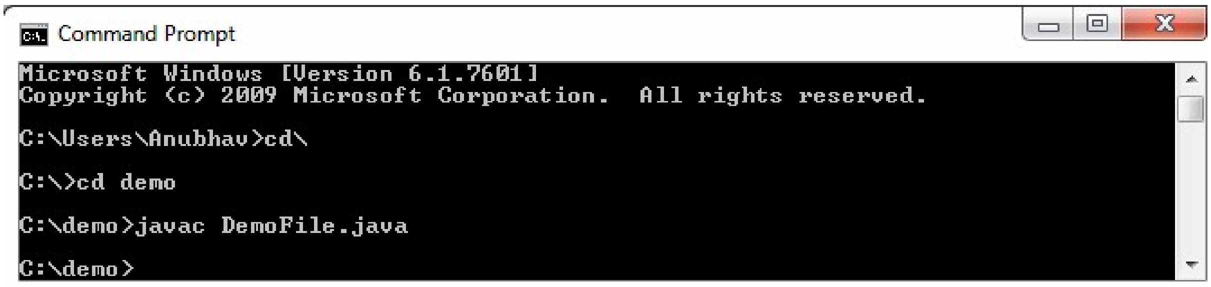


```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Anubhav>cd\

C:\>cd demo

C:\demo>_
```

Step 4:

Use the following command to compile the Java program. It generates a .class file in the same folder. It also shows an error if any.

```
javac DemoFile.java
```

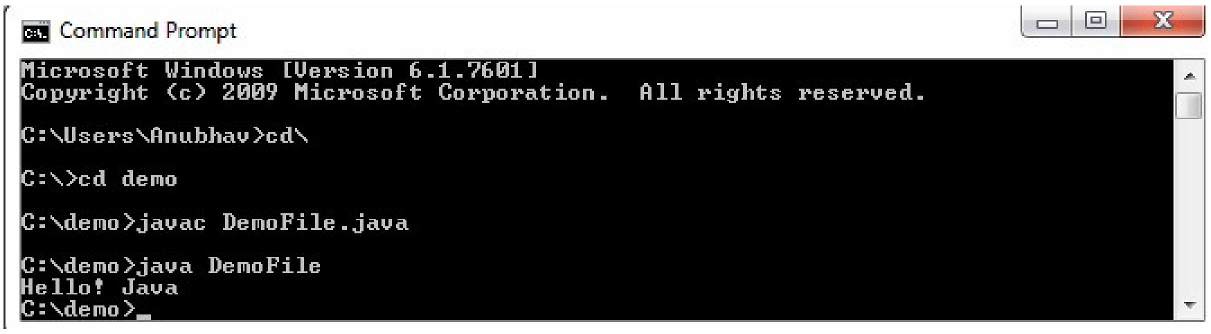


```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Anubhav>cd\

C:\>cd demo

C:\demo>javac DemoFile.java

C:\demo>
```

Step 5:

Use the following command to run the Java program:

1. java DemoFile



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Anubhav>cd\

C:\>cd demo

C:\demo>javac DemoFile.java

C:\demo>java DemoFile
Hello! Java
C:\demo>_
```

1.Using Buffered Reader Class

This is the Java classical method to take input, Introduced in JDK1.0. This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader, we can read input from the user in the command line.

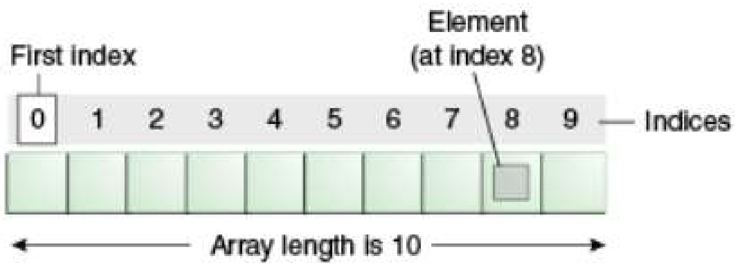
- The input is buffered for efficient reading.
- The wrapping code is hard to remember.

2. Using Scanner Class

This is probably the most preferred method to take input. The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however, it is also can be used to read input from the user in the command line.

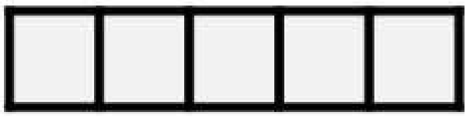
Array

array is a collection of similar type of elements which has contiguous memory location. **Java array** is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array. Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.



1. One-dimensional Array

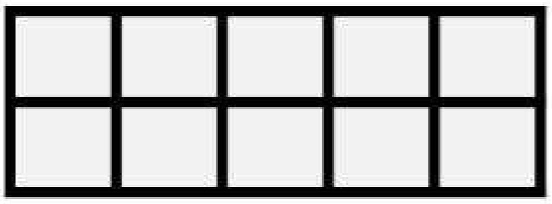
Also known as a linear array, the elements are stored in a single row. For example:



```
int Array = new int[5];
```

2 Two-dimensional Array

Two-dimensional arrays store the data in rows and columns:



```
int[][] Array = new int[2][5];
```