
## What is Java?

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

## Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

### 1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

### 2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

### 3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

### 4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

## Java Platforms / Editions

There are 4 platforms or editions of Java:

### 1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

### 2) Java EE (Java Enterprise Edition)

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

### 3) Java ME (Java Micro Edition)

It is a micro platform that is dedicated to mobile applications.

### 4) JavaFX

It is used to develop rich internet applications. It uses a lightweight user interface API.

**Features of Java:**

## Simple

**Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:**

- **Java syntax is based on C++ (so easier for programmers to learn it after C++).**
- **Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.**
- **There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.**

---

## 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.**

**Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.**

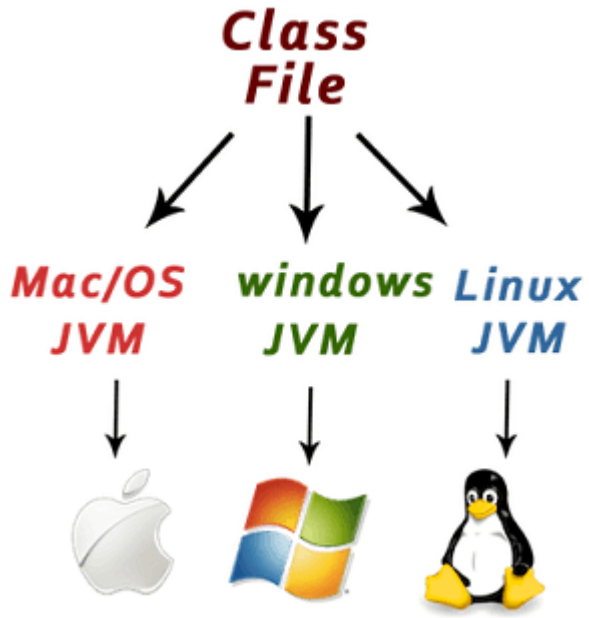
**Basic concepts of OOPs are:**

1. **Object**
2. **Class**
3. **Inheritance**
4. **Polymorphism**
5. **Abstraction**

6. **Encapsulation**

---

## Platform Independent




**Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.**

**There are two types of platforms software-based and hardware-based. Java provides a software-based platform.**

**The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:**

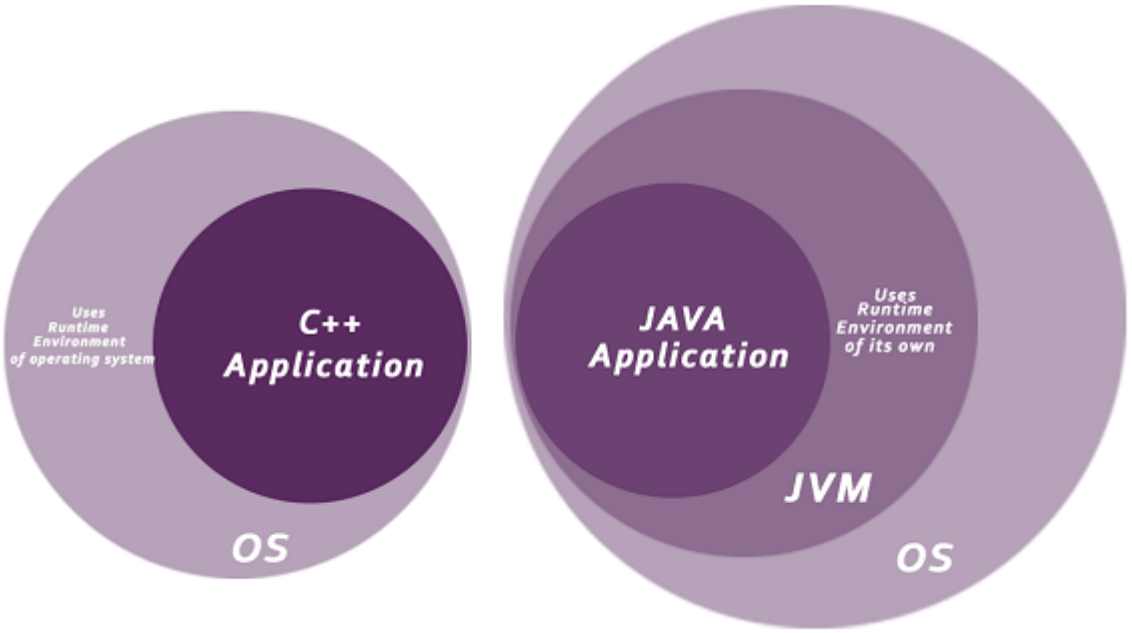
1. **Runtime Environment**
2. **API(Application Programming Interface)**

**Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).**

## Secured

**Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:**

- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**




- **Classloader: Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.**
- **Bytecode Verifier: It checks the code fragments for illegal code that can violate access rights to objects.**
- **Security Manager: It determines what resources a class can access such as reading and writing to the local disk.**

**Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.**

---

## Robust

**The English meaning of Robust is strong. Java is robust because:**

- **It uses strong memory management.**
- **There is a lack of pointers that avoids security problems.**
- **Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.**
- **There are exception handling and the type checking mechanism in Java. All these points make Java robust.**

---

## Architecture-neutral

**Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.**

**In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.**

---

## Portable

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

---

## High-performance

**Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.**

---

## Distributed

**Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.**

---

## Multi-threaded

**A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.**

---

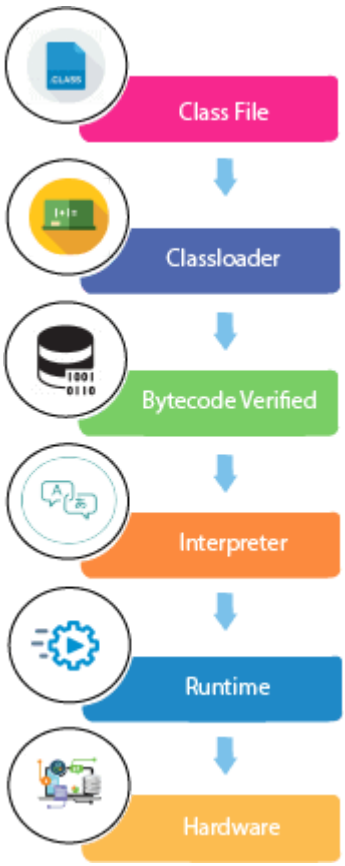
## Dynamic

**Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.**

**Java supports dynamic compilation and automatic memory management (garbage collection).**

# What happens at runtime?

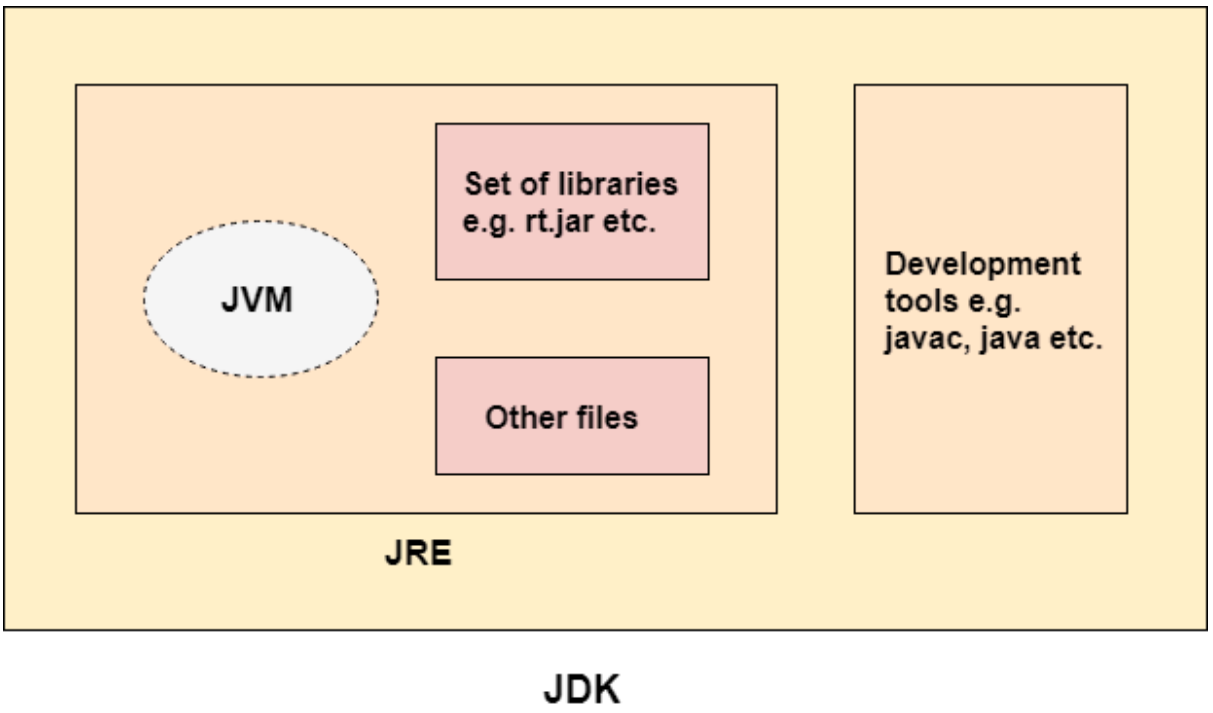
**At runtime, the following steps are performed:**




**JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:**

- **Standard Edition Java Platform**
- **Enterprise Edition Java Platform**
- **Micro Edition Java Platform**

**The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.**




## Java Variables

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

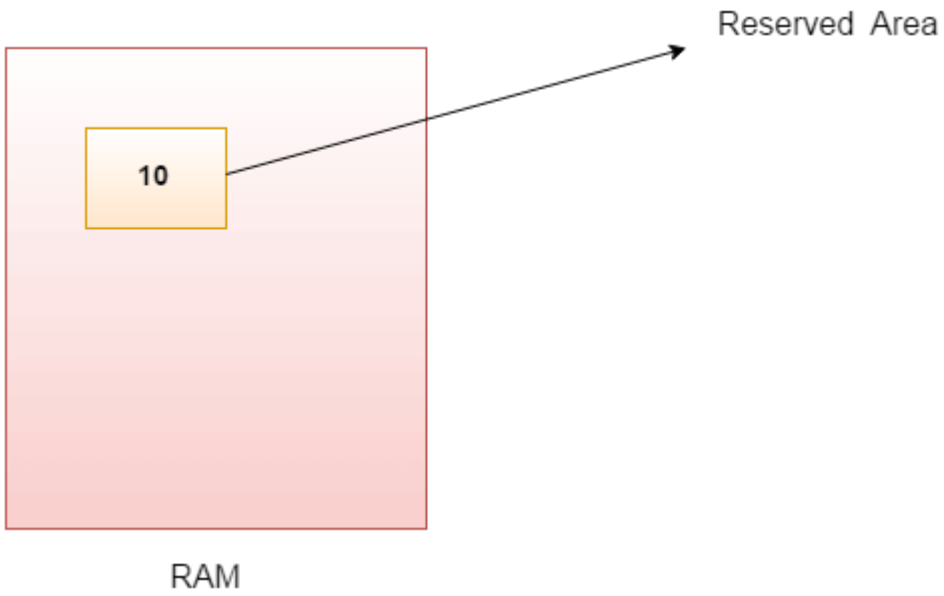
Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in Java: primitive and non-primitive.

---

## Variable

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.

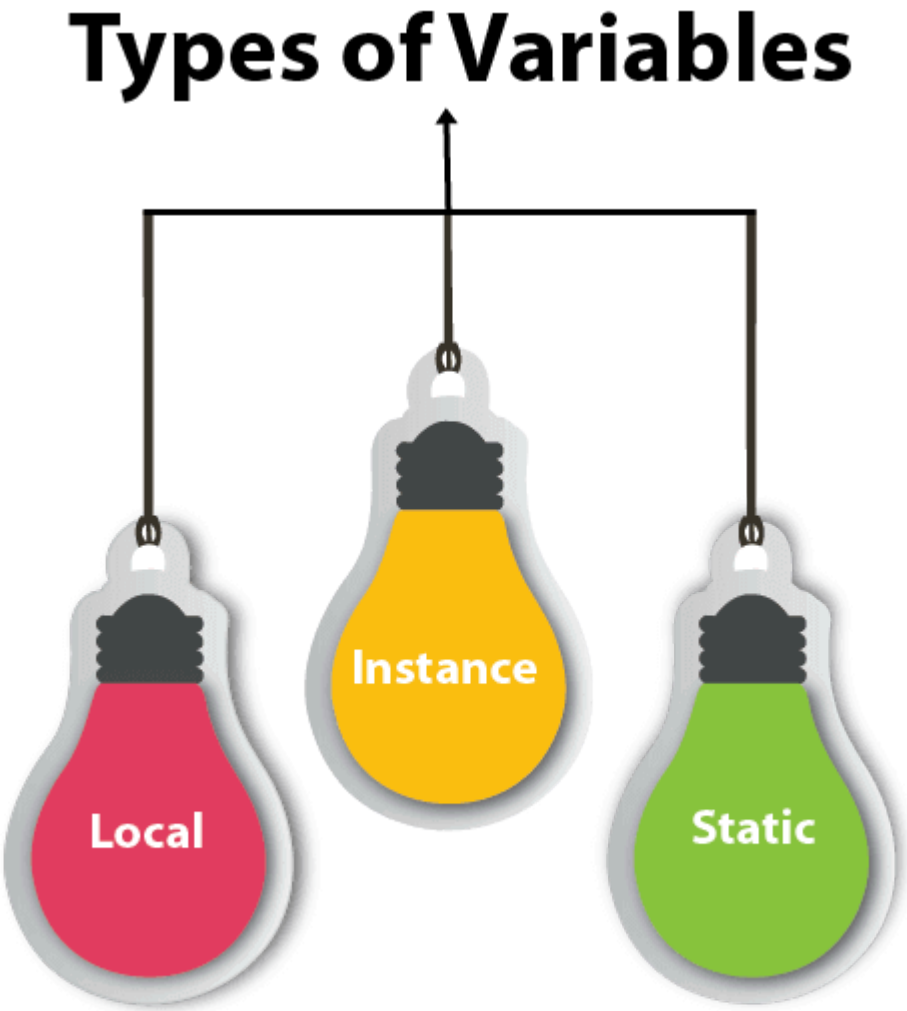



1. **int** data=50;//Here data is variable

## 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.

## Example to understand the types of variables in java

```
1. public class A
2. {
3.     static int m=100;//static variable
4.     void method()
5.     {
6.         int n=90;//local variable
7.     }
8.     public static void main(String args[])
9.     {
10.        int data=50;//instance variable
11.    }
12.}//end of class
```

# Java Variable Example: Add Two Numbers

```
1. public class Simple{
2. public static void main(String[] args){
3. int a=10;
4. int b=10;
5. int c=a+b;
6. System.out.println(c);
7. }
8. }
```

**Output:**

```
20
```

## Java Variable Example: Narrowing (Typecasting)

```
1. public class Simple{
2. public static void main(String[] args){
3. float f=10.5f;
4. //int a=f;//Compile time error
5. int a=(int)f;
6. System.out.println(f);
7. System.out.println(a);
8. }}
```

**Output:**

```
10.5
10
```

## Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

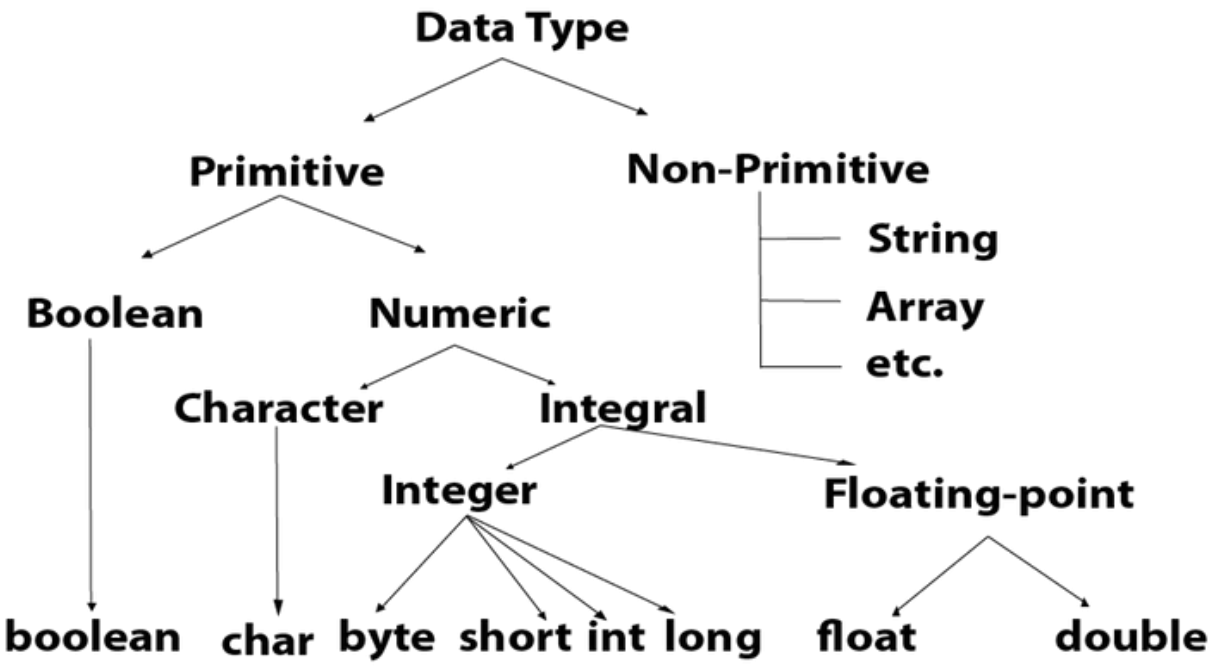
## Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

> Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type




## Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

**Example:**

1. Boolean one = **false**

## Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

**Example:**

1. **byte** a = 10, **byte** b = -20

## Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

**Example:**

1. **short** s = 10000, **short** r = -5000

## Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

**Example:**

```
1. int a = 100000, int b = -200000
```

## Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

**Example:**

```
1. long a = 100000L, long b = -200000L
```

## Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

**Example:**

```
1. float f1 = 234.5f
```

## Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

**Example:**

```
1. double d1 = 12.3
```

## Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

**Example:**

1. `char letterA = 'A'`

# Operators in Java

**Operator** in Java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

## Java Operator Precedence

| Operator Type | Category | Precedence |
|---|---|---|
| Unary | postfix | `expr++ expr--` |
| | prefix | `++expr --expr +expr -expr ~ !` |
| Arithmetic | multiplicative | `* / %` |

| | | |
|---|---|---|
| | additive | `+ -` |
| Shift | shift | `<< >> >>>` |
| Relational | comparison | `< > <= >= instanceof` |
| | equality | `== !=` |
| Bitwise | bitwise AND | `&` |
| | bitwise exclusive OR | `^` |
| | bitwise inclusive OR | `\|` |
| Logical | logical AND | `&&` |
| | logical OR | `\|\|` |
| Ternary | ternary | `? :` |
| Assignment | assignment | `= += -= *= /= %= &= ^= \|= <<= >>= >>>=` |

## Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

## Java Unary Operator Example: ++ and --

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int x=10;
4. System.out.println(x++);//10 (11)
5. System.out.println(++x);//12
6. System.out.println(x--);//12 (11)
7. System.out.println(--x);//10
8. }}
```

**Output:**

```
10
12
12
10
```

## Java Unary Operator Example 2: ++ and --

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=10;
5. System.out.println(a++ + ++a);//10+12=22
6. System.out.println(b++ + b++);//10+11=21
7.
8. }}
```

**Output:**

```
22
21
```

## Java Unary Operator Example: ~ and !

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=-10;
5. boolean c=true;
6. boolean d=false;
7. System.out.println(~a);//-11 (minus of total positive value which starts from
   0)
8. System.out.println(~b);//9 (positive of total minus, positive starts from 0)
9. System.out.println(!c);//false (opposite of boolean value)
10. System.out.println(!d);//true
11. }}
```

**Output:**

```
-11
9
false
true
```

## Java Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

## Java Arithmetic Operator Example

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=5;
5. System.out.println(a+b);//15
6. System.out.println(a-b);//5
```

```
7. System.out.println(a*b);//50
8. System.out.println(a/b);//2
9. System.out.println(a%b);//0
10.}}
```

**Output:**

```
15
5
50
2
0
```

## Java Arithmetic Operator Example: Expression

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. System.out.println(10*10/5+3-1*4/2);
4. }}
```

**Output:**

```
21
```

## Java Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

## Java Left Shift Operator Example

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. System.out.println(10<<2);//10*2^2=10*4=40
4. System.out.println(10<<3);//10*2^3=10*8=80
5. System.out.println(20<<2);//20*2^2=20*4=80
```

```
6. System.out.println(15<<4);//15*2^4=15*16=240
7. }}
```

**Output:**

```
40
80
80
240
```

## Java Right Shift Operator

The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

## Java Right Shift Operator Example

```
1. public OperatorExample{
2. public static void main(String args[]){
3. System.out.println(10>>2);//10/2^2=10/4=2
4. System.out.println(20>>2);//20/2^2=20/4=5
5. System.out.println(20>>3);//20/2^3=20/8=2
6. }}
```

**Output:**

```
2
5
2
```

## Java Shift Operator Example: >> vs >>>

```
1. public class OperatorExample{
2. public static void main(String args[]){
3.     //For positive number, >> and >>> works same
4.     System.out.println(20>>2);
```

```
5.      System.out.println(20>>>2);
6.      //For negative number, >>> changes parity bit (MSB) to 0
7.      System.out.println(-20>>2);
8.      System.out.println(-20>>>2);
9.  }}
```

**Output:**

```
5
5
-5
1073741819
```

## Java AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

```
1.  public class OperatorExample{
2.  public static void main(String args[]){
3.  int a=10;
4.  int b=5;
5.  int c=20;
6.  System.out.println(a<b&&a<c);//false && true = false
7.  System.out.println(a<b&a<c);//false & true = false
8.  }}
```

**Output:**

```
false
false
```

## Java AND Operator Example: Logical && vs Bitwise &

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=5;
5. int c=20;
6. System.out.println(a<b&&a++<c);//false && true = false
7. System.out.println(a);//10 because second condition is not checked
8. System.out.println(a<b&a++<c);//false && true = false
9. System.out.println(a);//11 because second condition is checked
10.}}
```

**Output:**

```
false
10
false
11
```

## Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=5;
5. int c=20;
6. System.out.println(a>b||a<c);//true || true = true
7. System.out.println(a>b|a<c);//true | true = true
8. //|| vs |
9. System.out.println(a>b||a++<c);//true || true = true
```

```
10. System.out.println(a);//10 because second condition is not checked
11. System.out.println(a>b|a++<c);//true | true = true
12. System.out.println(a);//11 because second condition is checked
13. }}
```

**Output:**

```
true
true
true
10
true
11
```

## Java Ternary Operator

Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

## Java Ternary Operator Example

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=2;
4. int b=5;
5. int min=(a<b)?a:b;
6. System.out.println(min);
7. }}
```

**Output:**

```
2
```

Another Example:

```
1. public class OperatorExample{
```

```
2. public static void main(String args[]){
3. int a=10;
4. int b=5;
5. int min=(a<b)?a:b;
6. System.out.println(min);
7. }}
```

**Output:**

```
5
```

## Java Assignment Operator

Java assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left.

## Java Assignment Operator Example

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=20;
5. a+=4;//a=a+4 (a=10+4)
6. b-=4;//b=b-4 (b=20-4)
7. System.out.println(a);
8. System.out.println(b);
9. }}
```

**Output:**

```
14
16
```

## Java Assignment Operator Example

```
1. public class OperatorExample{
2. public static void main(String[] args){
3. int a=10;
4. a+=3;//10+3
5. System.out.println(a);
6. a-=4;//13-4
7. System.out.println(a);
8. a*=2;//9*2
9. System.out.println(a);
10. a/=2;//18/2
11. System.out.println(a);
12. }}
```

**Output:**

```
13
9
18
9
```

## Java Assignment Operator Example: Adding short

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. short a=10;
4. short b=10;
5. //a+=b;//a=a+b internally so fine
6. a=a+b;//Compile time error because 10+10=20 now int
7. System.out.println(a);
8. }}
```

**Output:**

```
Compile time error
```

After type cast:

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. short a=10;
4. short b=10;
5. a=(short)(a+b);//20 which is int now converted to short
6. System.out.println(a);
7. }}
```

**Output:**

```
20
```

**User input in java:**

```
1. import java.util.*;
2. class UserInputDemo
3. {
4. public static void main(String[] args)
5. {
6. Scanner sc= new Scanner(System.in);    //System.in is a standard input
   stream
7. System.out.print("Enter first number- ");
8. int a= sc.nextInt();
9. System.out.print("Enter second number- ");
10. int b= sc.nextInt();
11. System.out.print("Enter third number- ");
12. int c= sc.nextInt();
13. int d=a+b+c;
14. System.out.println("Total= " +d);
15. }
16. }
```