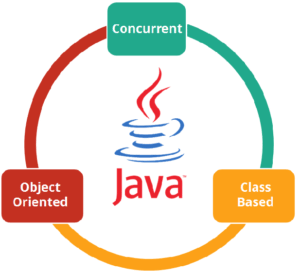
**What is Java?**

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.

**Platform**: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

It is an object-oriented language similar to C++, but with advanced and simplified features. This language is **free to access** and can **run** on **all platforms**.



Java is: –

* **Concurrent** where you can execute many statements instead of sequentially executing it.
* **Class-based** and an **object-oriented** programming language.
* **Independent** programming language that follows the logic of “**Write once, Run anywhere**” i.e. the compiled code can run on all platforms which supports java.

**What is Java used for?**

Before I go ahead with this, let me brief you about why you should choose Java. It is highly popular and has dominated this field from early 2000’s till the present 2022.

Some of the applications are listed below:

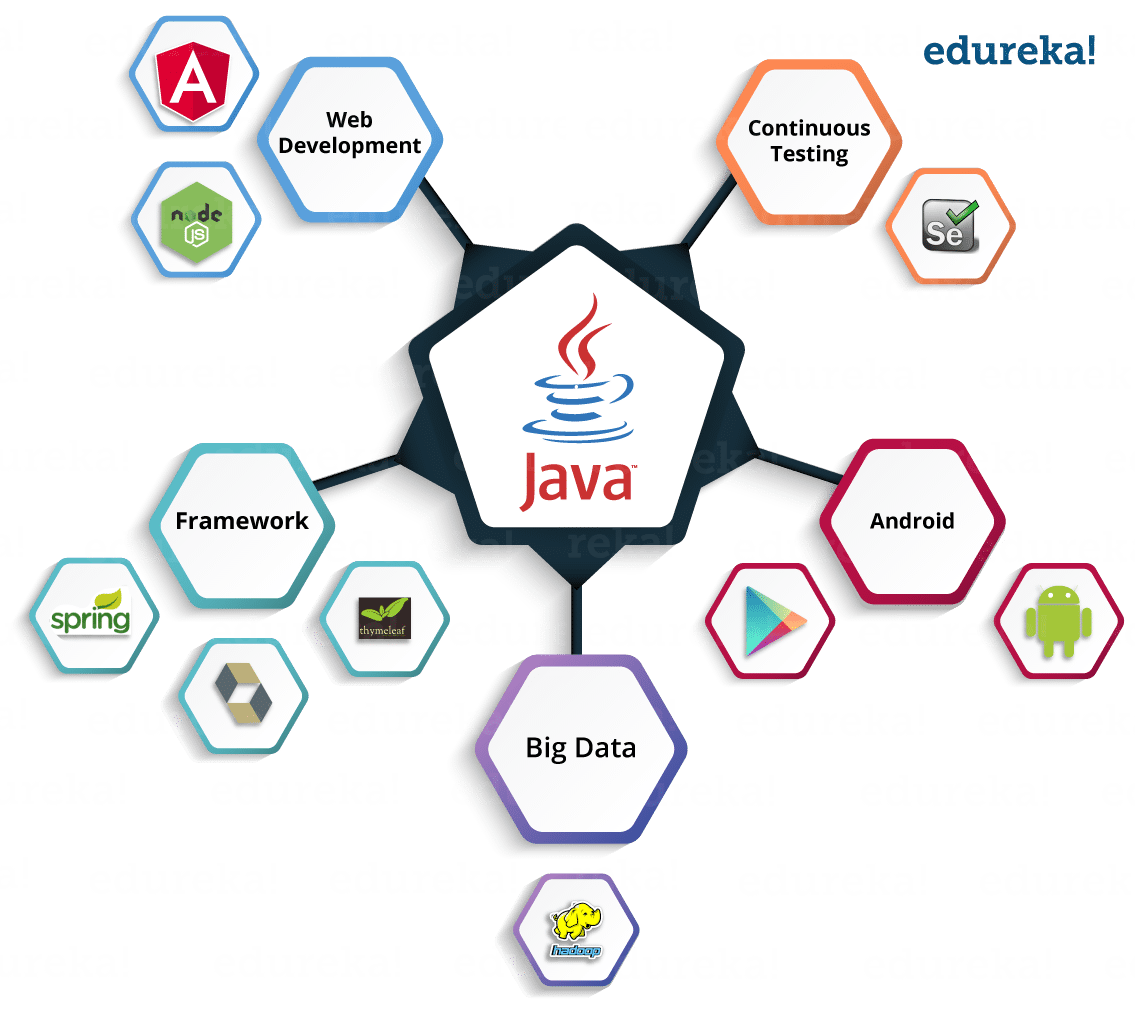
* **Banking**: To deal with transaction management.
* **Retail**: Billing applications that you see in a store/restaurant are completely written in Java.
* **Information Technology**: Java is designed to solve implementation dependencies.
* **Android**: Applications are either written in Java or use Java API.
* **Financial services**: It is used in server-side applications.
* **Stock market**: To write algorithms as to which company they should invest in.
* **Big Data**: Hadoop MapReduce framework is written using Java.
* **Scientific and Research Community**: To deal with huge amount of data.

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as Amazon.com,facebook.com,google.com etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

***Wait! Java can do more.***

Let’s see how some of the technologies make use of Java as an essential core of their functionalities.



You can see in the above image; Java is an ***ocean of opportunities***.

**Features of java**

**Simple:**Java has made life easier by removing all the complexities such as pointers, operator overloading as you see in C++ or any other programming language.

 **Portable:**This is platform independent which means that any application written on one platform can be easily ported to another platform.

**Object-oriented:**Everything is considered to be an “**object**” which possess some state, behaviour and all the operations are performed using these objects.

**Secured:**All the code is converted in **bytecode** after compilation, which is not readable by a human. and java does not use an explicit pointer and run the programs inside the sandbox to prevent any activities from untrusted sources. It enables to develop virus-free, tamper-free systems/applications.

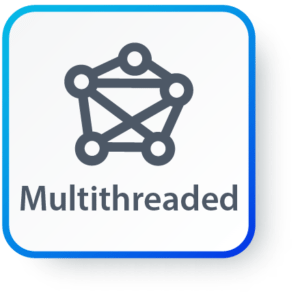
**Dynamic:**It has the ability to adapt to an evolving environment which supports dynamic memory allocation due to which memory wastage is reduced and performance of the application is increased.

**Distributed:**This language provides a feature which helps to create distributed applications. Using Remote Method Invocation (RMI), a program can invoke a method of another program across a network and get the output. You can access files by calling the methods from any machine on the internet.

**Robust:**Java has a strong memory management system. It helps in eliminating error as it checks the code during compile and runtime.

**High Performance:**Java achieves high performance through the use of bytecode which can be easily translated into native machine code. With the use of JIT (Just-In-Time) compilers, it enables high performance.

**Interpreted:** Java is compiled to bytecodes, which are interpreted by a run-time environment.

**Multithreaded:** Java supports multiple threads of execution (a.k.a., lightweight processes), including a set of synchronization primitives. This makes programming with threads much easier.

**Differences between JDK, JRE and JVM**



**Don’t get confused as we are going to discuss all of them one by one.**

**What is Java Virtual Machine (JVM)?**

[Java](https://www.naukri.com/learning/what-is-java-st619-tg17) Virtual Machine (JVM) is an abstract machine responsible for compiling and executing Java code. It is a part of the Java Runtime Environment (JRE) which calls the main function of a program.

* JVM facilitates a **platform-independent** way of executing Java source code. Its basis on WORA (Write Once Run Anywhere).
* It has a class loader, runtime data area, execution engine, and libraries.
* JVM comes with JIT(Just-in-Time) compiler that converts Java source code into machine code.
* When you run the Java program, Java compiler first compiles your Java code to bytecode. Then, the JVM translates bytecode into native machine code (set of instructions that a computer's CPU executes directly).
* Java is a platform-independent language. It's because when you write Java code, it's ultimately written for JVM but not your physical machine (computer). Since JVM ​executes the Java bytecode which is platform-independent, Java is platform-independent.



First, the **java compiler** compiles a Java file into a Java .class file. Then the .class file works as an input into the **JVM**, which loads and executes the class file.

**Features of JVM**

* It converts byte code to the machine language.
* JVM provides basic java functions like memory management, security, garbage collection, etc.
* Runs the program by utilizing JRE’s libraries and files.
* JVM is an integral part of JRE.
* It can execute the java program line by line. Therefore, it is also known as an interpreter.
* The main functions of JVM include loading, linking, initializing, and compiling the program.

**Note:** JVM can’t be installed alone. As JVM is a part of JRE, you need to install JRE. JVM comes within it.

**What is Java Runtime Environment (JRE)?**

JRE is software that includes JVM and class libraries to run java programs independently. Although it can execute the code. Yet, JRE comes bundled with Java development Kit (JDK) to provide a complete application development experience.

* The JRE includes class libraries along with JVM and its supporting files. Note: It does not include separate tools such as a debugger for Java development.
* JRE comprises significant package classes such as util, math, AWT, lang, and various runtime libraries.

**JRE = JVM + Class Libraries (For Running the Java Applications).**



**Features of JRE**

* JRE consists of a set of tools to help the JVM run. In addition, it includes a few deployment tools such as Java Plug-in and Java Web Start.
* A User can efficiently run a java code with JRE only. However, JRE doesn’t allow writing the program.
* JRE appends various integration libraries like the JDBC (Java Database Connectivity), JNDI (Java Naming and Directory Interface), RMI (Remote Method Invocation), etc.
* Along with JVM, it consists of a virtual machine client for Java HotSpot.

The components of JRE are as follows:

1. **Deployment technologies**, including deployment, Java Web Start, and Java Plug-in.
2. **User interface toolkits**, including *Abstract Window Toolkit (AWT), Swing, Java 2D, Accessibility, Image I/O, Print Service, Sound, drag*,*and drop (DnD)*, and *input methods*.
3. **Integration libraries**, including *Interface Definition Language (IDL), Java Database Connectivity (JDBC), Java Naming and Directory Interface (JNDI), Remote Method Invocation (RMI), Remote Method Invocation Over Internet Inter-Orb Protocol (RMI-IIOP)*, and *scripting*.
4. **Other base libraries**, including *international support, input/output (I/O), extension mechanism, Beans, Java Management Extensions (JMX), Java Native Interface (JNI), Math, Networking, Override Mechanism, Security, Serialization*, and *Java for XML Processing (XML JAXP)*.
5. **Lang and util base libraries**, including *lang and util, management, versioning, zip, instrument, reflection, Collections, Concurrency Utilities, Java Archive (JAR), Logging, Preferences API, Ref Objects*, and *Regular Expressions*.

**What is the Java Development Kit (JDK)?**

Java Development Kit (JDK) is a complete software environment for building applications and applets using the Java programming language. It is platform-dependent. Therefore, it has different OS platform versions for Windows, Linux, Mac, etc. It allows to read, write, and execute the java program.

* It includes various tools required for writing Java programs.
* It includes an Applet viewer, Java application launcher, debugger, compiler, etc.
* JDK also consists of JRE for executing Java programs.
* The Java application launcher helps in opening a JRE. Then, it loads the necessary details and executes the main method of the program.

**JDK = Development Tools + JRE (Java Runtime Environment)**



**Features of JDK**

* JDK provides an environment for developing and executing the Java source code.
* It includes all the functionalities of JRE and JVM.
* JDK helps developers to handle the exceptions using multiple extensions in a single catch block.
* It has various other development tools like the debugger, compiler, etc.
* It is platform-dependent. Moreover, any user can easily install JDK on Operating systems like Unix, Mac, Windows, etc.

### **Simple Java Hello World Program**

We can simplify the process of creating and running a Java program into three steps:

1. Create the program by typing it into a text editor(like notepad). And, save it to a file named HelloWorld.java.  
2. Compile the file by typing **“javac HelloWorld.java”** in the command prompt window.  
3**.** To execute or run the program, type **“java HelloWorld”** in the command prompt window.

Print Hello World in Java

The below program is the simplest program of Java printing “Hello World” to the screen. Let us try to understand every bit of code step by step.

/\* This is a simple Java program.

FileName : "HelloWorld.java". \*/

public **class** HelloWorld {

//Your program begins by calling the main().

//Prints "Hello, World" to the terminal window.

public static **void** main(String args[]) {

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

}

}

**Output:**

**Hello World**

The above program consists of three primary components: the class definition, main() method and comments. The following section will give you a basic understanding of this code:

**1. Class definition:** We use the ‘class’ keyword to declare a new class. We can also use the access specifier like public before the class keyword:

public **class** HelloWorld

**2. Hello World** is the name of the class that is an identifier in Java. The class definition contains the members of the class that are enclosed within the curly braces{}.

**3. Java Main() method:** Every application in Java programming language must contain a main() method whose signature is:

public static **void** main(String[] args)

* **public:** We declare the main method as public so that JVM can execute it from anywhere.
* **static:** We declare the main method as static so that JVM can call it directly without creating the object of the class.  
  **Note:** We can write the modifiers public and static in any order.
* **void:** The main method does not return anything, therefore we declare it as void.
* **main():** main() is the name that is already configured in the JVM.
* **String[]:** The main() method accepts a single argument which is an array of elements of type String.

The main method is the entry point for any Java application as in C/C++. The main() method will subsequently invoke all the other methods required by the program.

Below is the next line of code. It is present inside the main() method:

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

This line actually prints the string “Hello World” on the screen, followed by a new line on the screen. We can get the output on the screen because of the built-in **println()** method.

The System is a predefined class in Java. This class provides access to the system. out is the variable of type output stream that is connected to the console.

**4. Java Comments:** Comments in Java can either be multi-line or single-line comments.

/\* This is a simple Java program.

Call this file "HelloWorld.java". \*/

This is a multiline comment. It must begin with /\* and end with \*/. For a single line comment, we can directly use // as in C or C++.

Important Points

* The name of the class in the program is HelloWorld. This name is the same as the name of the file, HelloWorld.java. These same names are not a coincidence. In Java, the whole code must reside inside a class. And there must be at most one public class that contains the main() method.
* By convention, the name of the class containing the main method should match the name of the file that holds the program.

Compiling Java Program

1. Firstly, we need to set up the environment. After that, we can open a terminal or command prompt in both Windows or Unix and can go to the directory/folder where we have saved the file: HelloWorld.java.

2**.** Now, to compile Java HelloWorld program, we need to execute the compiler: javac, specifying the name of the source file on the command line, like:

javac HelloWorld.java

3**.** The compiler creates the compiled file called HelloWorld.class in the present working directory. This class file that contains the bytecode version of the program.

Running Java Program

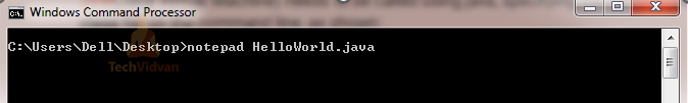
To execute Java program, we need to call JVM(Java Virtual Machine) using java command. After that, we specify the name of the class file on the command line, like:

java HelloWorld

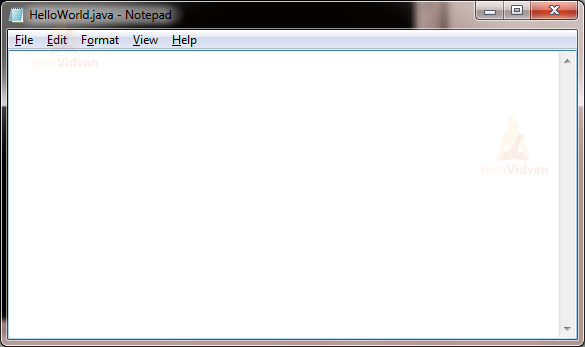
This will print “Hello World” on the terminal screen.

Steps to write HelloWorld Program in Windows

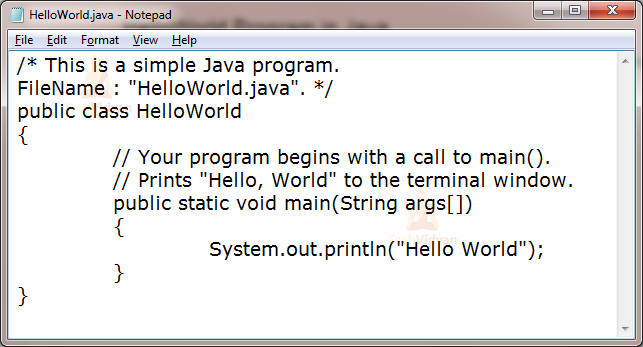
**Step 1:** Open Command Prompt Window, Reach to the desired folder and type **notepad HelloWorld.java**, like this:



**Step 2:** Now, hit **Enter**. As soon as you press enter, you will see a notepad editor screen:

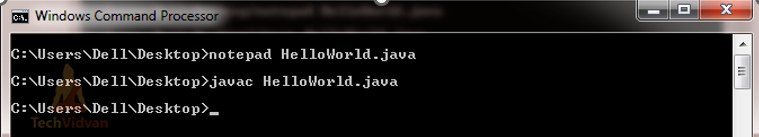
[](https://techvidvan.com/tutorials/wp-content/uploads/sites/2/2020/06/pasted-image-0-1-10.png)

**Step 3:** Start typing the program:

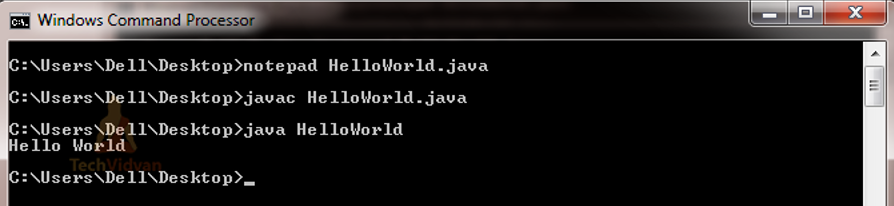
[](https://techvidvan.com/tutorials/wp-content/uploads/sites/2/2020/06/pasted-image-0-2-11.png)

**Step 4:** Go to the**File** option and click the **Save** button to save the file. Close the notepad window. Move to the command prompt window again.

**Step 5:** Type here**javac HelloWorld.java**, and enter. If the program compiles successfully, then the cursor will start blinking on the next line, otherwise, there will be an error message.



**Step 6:** Now type **java HelloWorld**. Press Enter to get the output:



You can see the printed output as Hello World on the screen.

**Alternatively, if you want to use IDE like Eclipse , NetBeans, IntelliJ... for creating Java programs, then you can create a project and add a class with the main function.**

In how many ways we can write a Java program?

There are many ways to write a Java program. The modifications that can be done in a Java program are given below:

**1) By changing the sequence of the modifiers, method prototype is not changed in Java.**

Let's see the simple code of the main method.

1. **static** **public** **void** main(String args[])

**2) The subscript notation in the Java array can be used after type, before the variable or after the variable.**

Let's see the different codes to write the main method.

1. **public** **static** **void** main(String[] args)
2. **public** **static** **void** main(String []args)
3. **public** **static** **void** main(String args[])

**3) You can provide var-args support to the main() method by passing 3 ellipses (dots)**

Let's see the simple code of using var-args in the main() method. We will learn about var-args later in the Java New Features chapter.

1. **public** **static** **void** main(String... args)

**4) Having a semicolon at the end of class is optional in Java.**

Let's see the simple code.

1. **class** Simple{
2. **static** **public** **void** main(String... args){
3. System.out.println("hello java4");
4. }
5. };

Valid Java main() method signature

1. **public** **static** **void** main(String[] args)
2. **public** **static** **void** main(String []args)
3. **public** **static** **void** main(String args[])
4. **public** **static** **void** main(String... args)
5. **static** **public** **void** main(String[] args)
6. **public** **static** **final** **void** main(String[] args)
7. **final** **public** **static** **void** main(String[] args)
8. **final** **strictfp** **public** **static** **void** main(String[] args)

Invalid Java main() method signature

1. **public** **void** main(String[] args)
2. **static** **void** main(String[] args)
3. **public** **void** **static** main(String[] args)
4. **abstract** **public** **static** **void** main(String[] args)

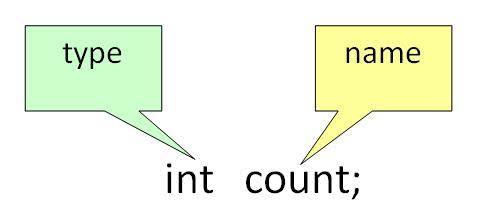
**Variable in Java** is a data container that saves the data values during Java program execution. Every variable is assigned a data type that designates the type and quantity of value it can hold. A variable is a memory location name for the data.

A variable is a name given to a memory location. It is the basic unit of storage in a program.

* The value stored in a variable can be changed during program execution.
* A variable is only a name given to a memory location. All the operations done on the variable affect that memory location.
* In Java, all variables must be declared before use.

**How to declare variables?**

We can declare variables in Java as pictorially depicted below as a visual aid.



From the image, it can be easily perceived that while declaring a variable, we need to take care of two things that are:

**1. datatype**: Type of data that can be stored in this variable.

**2. data\_name:** Name given to the variable.

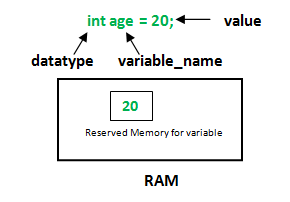
In this way, a name can only be given to a memory location. It can be assigned values in two ways:

* Variable Initialization
* Assigning value by taking input

**How to initialize variables?**

It can be perceived with the help of 3 components that are as follows:

* **datatype**: Type of data that can be stored in this variable.
* **variable\_name**: Name given to the variable.
* **value**: It is the initial value stored in the variable.



**Illustrations:**

float simpleInterest;

// Declaring float variable

int time = 10, speed = 20;

// Declaring and initializing integer variable

char var = 'h';

// Declaring and initializing character variable

**Types of Variables in Java**

Now let us discuss different types of variables which are listed asfollows:

1. Local Variables
2. Instance Variables
3. Static Variables

Let us discuss the traits of every type of variable listed here in detail.

**1. Local Variables**

A variable defined within a block or method or constructor is called a local variable.

* These variables are created when the block is entered, or the function is called and destroyed after exiting from the block or when the call returns from the function.
* The scope of these variables exists only within the block in which the variables are declared, i.e., we can access these variables only within that block.
* Initialization of the local variable is mandatory before using it in the defined scope.
* Java

|  |
| --- |
| /\*package whatever //do not write package name here \*/  // Contributed by Shubham Jain  **import** java.io.\*;  **class** GFG {  **public** **static** **void** main(String[] args)      {  **int** var = 10; // Declared a Local Variable          // This variable is local to this main method only          System.out.println("Local Variable: " + var);      }  } |

**Output**

Local Variable: 10

**2. Instance Variables**

Instance variables are non-static variables and are declared in a class outside of any method, constructor, or block.

* As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
* Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier, then the default access specifier will be used.
* Initialization of an instance variable is not mandatory. Its default value is 0.
* Instance variables can be accessed only by creating objects.
* Java

|  |
| --- |
| /\*package whatever //do not write package name here \*/  **import** java.io.\*;  **class** GFG {  **public** String my; // Declared Instance Variable  **public** GFG()      { // Default Constructor  **this**.my = "Suresh"; // initializing Instance Variable      }  //Main Method  **public** **static** **void** main(String[] args)      {          // Object Creation          GFG name = **new** GFG();          // Displaying O/P          System.out.println("My name is: " + name.my);      }  } |

**Output**

My name is: Suresh

**3. Static Variables**

Static variables are also known as class variables.

* These variables are declared similarly as instance variables. The difference is that static variables are declared using the static keyword within a class outside of any method, constructor or block.
* Unlike instance variables, we can only have one copy of a static variable per class, irrespective of how many objects we create.
* Static variables are created at the start of program execution and destroyed automatically when execution ends.
* Initialization of a static variable is not mandatory. Its default value is 0.
* If we access a static variable like an instance variable (through an object), the compiler will show a warning message, which won’t halt the program. The compiler will replace the object name with the class name automatically.
* If we access a static variable without the class name, the compiler will automatically append the class name.
* Java

|  |
| --- |
| /\*package whatever //do not write package name here \*/  **import** java.io.\*;  **class** GFG {  **public** **static** String my = "Suresh";        //Declared static variable  **public** **static** **void** main (String[] args) {        //my variable can be accessed without object creation        //Displaying O/P        //GFG.my --> using the static variable          System.out.println("My Name is : "+GFG.my);      }  } |

**Output**

My Name is : Suresh

**Differences between the Instance variables and the Static variables**

Now let us discuss the differences between the Instance variables and the Static variables:

* Each object will have its own copy of an instance variable, whereas we can only have one copy of a static variable per class, irrespective of how many objects we create.
* Changes made in an instance variable using one object will not be reflected in other objects as each object has its own copy of the instance variable. In the case of a static variable, changes will be reflected in other objects as static variables are common to all objects of a class.
* We can access instance variables through object references, and static variables can be accessed directly using the class name**.**

**Syntax:** Static and instance variables

class GFG

{

// Static variable

static int a;

// Instance variable

int b;

}

**Rules to Declare a Variable**

1. A variable name can consist of Capital letters **A-Z**, lowercase letters **a-z** digits **0-9**, and two special characters such as **\_** underscore and **$** dollar sign.
2. The first character must not be a digit.
3. Blank spaces cannot be used in variable names.
4. Java keywords cannot be used as variable names.
5. Variable names are case-sensitive.
6. There is no limit on the length of a variable name but by convention, it should be between 4 to 15 chars.
7. Variable names always should exist on the left-hand side of assignment operators.

**List of Java keywords**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | continue | for | new | switch |
| assert | package | synchronized | default | goto |
| boolean | do | if | private | this |
| break | else | import | public | throw |
| byte | enum | implements | protected | throws |
| case | double | instanceof | return | transient |
| catch | extends | int | short | try |
| char | final | interface | static | void |
| class | finally | long | strictfp | volatile |
| const | float | native | super | while |

The table above shows the list of all java keywords that programmers can not use for naming their variables, methods, classes, etc. The keywords const and goto are reserved, but they are not currently used. The words true, false, and null might seem like keywords, but they are actually literals, you cannot use them as identifiers in your programs.

**Example**

* Java

|  |
| --- |
| **import** java.io.\*;  **class** GFG {  **public** **static** **void** main(String[] args)      {          // Declaring all the          // possible combinations of          // variable format  **int** \_a = 10;  **int** $b = 20;  **int** C = 30;  **int** c = 40;  **int** result = \_a + $b + C + c;          // Displaying O/P          System.out.println("Result: " + result);      }  } |

**Output:**

Result: 100

***Here are a few valid Java variable name examples:***

* ***myvar***
* ***myVar***
* ***MYVAR***
* ***\_myVar***
* ***$myVar***
* ***myVar1***
* ***myVar\_1***

# 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

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Default Value** | **Default size** |
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

* 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 is an 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. 64bit range: −9,223,372,036,854,775,808 to +9,223,372,036,854,775,807

**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'

### **Why char uses 2 byte in java and what is \u0000 ?**

It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system. To get detail explanation about Unicode visit next page.

# Unicode System

|  |
| --- |
| Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages. |

### **Why java uses Unicode System?**

|  |
| --- |
| Before Unicode, there were many language standards: |
| * **ASCII** (American Standard Code for Information Interchange) for the United States. * **ISO 8859-1** for Western European Language. * **KOI-8** for Russian. * **GB18030 and BIG-5** for chinese, and so on. |

## Problem

|  |
| --- |
| **This caused two problems:**   1. A particular code value corresponds to different letters in the various language standards. 2. The encodings for languages with large character sets have variable length.Some common characters are encoded as single bytes, other require two or more byte. |

## Solution

|  |
| --- |
| To solve these problems, a new language standard was developed i.e. Unicode System. |
| In unicode, character holds 2 byte, so java also uses 2 byte for characters. |
| **lowest value:**\u0000 |
| **highest value:**\uFFFF |

# 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 --**

**public** **class** OperatorExample{

**public** **static** **void** main(String args[]){

**int** x=10;

System.out.println(x++); //10 (11)

System.out.println(++x);//12

System.out.println(x--);//12 (11)

System.out.println(--x);//10

}}

**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
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**

**public** **class** OperatorExample{

**public** **static** **void** main(String args[]){

**int** a=10;

**int** b=5;

System.out.println(a+b);//15

System.out.println(a-b);//5

System.out.println(a\*b);//50

System.out.println(a/b);//2

System.out.println(a%b);//0

}}

**Output:**

15

5

50

2

0

### **Java Arithmetic Operator Example: Expression**

**public** **class** OperatorExample{

**public** **static** **void** main(String args[]){

System.out.println(10\*10/5+3-1\*4/2);

}}

**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**

**public** **class** OperatorExample{

**public** **static** **void** main(String args[]){

System.out.println(10<<2);//10\*2^2=10\*4=40

System.out.println(10<<3);//10\*2^3=10\*8=80

System.out.println(20<<2);//20\*2^2=20\*4=80

System.out.println(15<<4);//15\*2^4=15\*16=240  (2\*2\*2\*2)

}}

**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.

**public** **class** OperatorExample{

**public** **static** **void** main(String args[]){

**int** a=10;

**int** b=5;

**int** c=20;

System.out.println(a<b&&a<c);//false && true = false

System.out.println(a<b&a<c);//false & true = false

}}

**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.

**public** **class** OperatorExample{

**public** **static** **void** main(String args[]){

**int** a=10;

**int** b=5;

**int** c=20;

System.out.println(a>b||a<c);//true || true = true

System.out.println(a>b|a<c);//true | true = true

//|| vs |

System.out.println(a>b||a++<c);//true || true = true

System.out.println(a);//10 because second condition is not checked

System.out.println(a>b|a++<c);//true | true = true

System.out.println(a);//11 because second condition is checked

}}

**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**

**public** **class** OperatorExample{

**public** **static** **void** main(String args[]){

**int** a=10;

**int** b=20;

a+=4; //a=a+4 (a=10+4)

b-=4; //b=b-4 (b=20-4)

System.out.println(a);

System.out.println(b);

}}

**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

**What are Keywords in java?**

Keywords are special tokens that add a different meaning to the language compiler. Each keyword has its own individual function and performs a specific task assigned to it. In Java, we have 50 such reserved words, out of which 48 are in use and 2 are reserved but not in use. Keywords cannot be used as identifiers in a program, so it is essential that a programmer knows all the keywords to avoid any kind of syntax or logical errors.

Let us study all the keywords that java provides us with in detail.

**1. abstract:** Using the abstract keyword in java we can create abstract classes and methods. Abstract keywords are essential to implement abstraction into a program.

For example:

abstract **class** DataFlair

**2. assert:** Using the assert keyword we can implement assertion in a program. Using it we can check the correctness of any assumptions made in a program. The assert keyword was added in JDK 1.4.

For Example:

int i = 10;

assert i > 5 : "The value is greater than 5";

**3. boolean:** Using the boolean keyword we can declare a boolean variable. A boolean variable is a variable that has two values, true and false.

For example:

boolean flag = **true**;

**4. break:** The break keyword is a jump statement using which we can break out of a loop or switch statement. The break statement terminates the currently executing block of code.

For example:

i=0;

**while**(**true**)

{

**if**(i==10)

break;

i++;

}

**5. byte:** Using the byte keyword in java we can declare a variable that can hold a value of 1 byte or 8 bit.

For example:

byte num=1;

**6. case:** Using the case statement we can declare each case inside a switch-case block.

For example:

{

case 1:

System.out.println(“ONE”);

break;

}

**7. catch:** The catch keyword is part of exception handling in java. Using the catch keyword we can catch or capture the error caught by the try block. A catch block can exist only after the try block.

For Example:

**catch** (Exception e)

{

System.out.println(“An exception was caught by the **try** block”);

}

**8. char:** Using the char keyword we can declare a character variable.

For example:

char ch=’A’;

**9. class:** Using the class keyword we can declare a class in java.

For example:

**class** DataFlair

**10. continue:** Continue is also a jump statement in java. Using it we can terminate the current iteration and continue from the next iteration inside the loop.

For example:

i=0;

**while**(i<=10)

{

**if**(i==5)

continue;

i++;

}

**11. default:** default is the keyword using which we can declare the default statement inside a switch case. It is executed when none of the cases match.

For Example:

Switch(num)

{

case 1:

System.out.println(“ONE”);

break;

default:

System.out.println(“Not ONE”);

}

**12. do:** Using the do keyword we can declare a do-while loop. It is an exit controlled loop, so it doesn’t have any entry condition.

For example:

i=0;

**do**{

i++;

}**while**(i<=10);

**13. double:** Using the double keyword we can declare a double variable. A double variable can hold a 64bit long floating point number.

For Example:

double d= 1.23456;

**14. else:** Using the else keyword we can write statements that will be executed when the if block doesn’t execute successfully.

For example:

i=5;

**if**(i!=5)

System.out.println(“Not Five”);

**else**

System.out.println(“FIVE”);

**15. enum:** Using the enum keyword we can declare an enumerated class. It consists of a fixed set of constants. Enum constructors are always private by default. Enum keyword was added to java in JDK 5.0.

For Example:

enum Rainbow

{

Violet, Indigo, Blue, Green,

Yellow, Orange, Red;

}

**16. extends:** extend keyword is used in inheritance. Using it we can inherit one class into another.

For Example:

**class** subclass **extends** superclass

**17. final:** Using the final keyword we can finalize the value of a variable. It means that the value cannot be updated by the user whatsoever.

For example:

final int num = 1000;

**18. finally:** The finally keyword is used after the try-catch block. It indicates the end of the block. Unlike the try-catch block, the finally block will be executed regardless of whether an exception is found or not.

For Example:

finally{

System.out.println("This block is always executed whatsoever!");

}

**19. float:** Using the float keyword we can declare a float variable in java. The float variable holds a 32-bit long floating-point number.

For example:

float num= 1.324f;

**20. for:** Using the for keyword we can declare a for loop. It is an entry controlled loop, for this kind of loop the number of iterations should be known beforehand.

For example:

**for**(i=0;i<10;i++)

{

System.out.println(i);

}

**21. if:** In java, if keyword is used to declare if conditional statement. The if statement is used to check the viability of a condition.

For Example:

i=5;

**if**(i==5)

{

System.out.println(“The Number is Five”);

}

**22. implements:** Using the implements keyword in java we can declare an interface. It is like inheritances extend keyword, here to access an interface we have to use the implements keyword.

For Example:

interface DataFlair{

public **void** intern();

}

**class** internship **implements** DataFlair

{

Public **void** intern()

{

System.out.println(“Internship Granted”);

}

}

**23. import:** Using the import keyword in java we can access classes and interfaces present inside a package to the current code.

For Example:

import java.util.Arrays;

**24. instanceof:** Using the instanceof keyword in java we can check whether a given object is an instance of another class or not. It returns true if the given object is an instance and false if not.

For Example:

public **class** DataFlair {

public static **void** main(String[] args) {

DataFlair Obj1 = **new** DataFlair();

System.out.println(Obj1 instanceof DataFlair);// It will return True.

}

}

**25. int:** Using the int keyword we can declare a variable of integer datatype. It can hold a value of 32-bit long.

For Example:

int num=100;

**26. interface:** Using the interface keyword in java we can declare an interface inside the code.

For Example:

interface DataFlair{}

**27. long:** Using the long keyword we can declare variables with long data type. The long data type is integers that can hold 64-bit of data.

For Example:

long num=100000;

**28. native:** The native keyword in java is used to indicate that a method is implemented in native code using JNI[Java Native Interface].

For Example:

Public native **void** DataFlair();

**29. new:** Using the new keyword we can create a new instance for a class.

For Example:

public **class** DataFlair {

public static **void** main(String[] args) {

DataFlair Obj1 = **new** DataFlair();

}

**30. Package:** Using the package keyword in java we can create a new package in the java library.

For Example:

package com.DataFlair.Keywords;

**31. private:** private keyword is an access modifier that declares a class or method as private; i.e; it can only be accessed by its local variables.

For Example:

private **class** DataFlair{}

**32. protected:** protected is also an access modifier that declares a class or method as protected; i.e; it can be accessed by files in the PWD(Present Working Directory).

For Example:

protected **class** DataFlair{}

**33. public:** public is another access modifier that declares a class or method as public; i,e; it can be accessed globally.

For Example:

public **class** DataFlair{}

**34. return:** The return keyword is used inside a method when it requires to return a value of a certain return type except void.

For Example:

int sum(int a, int b)

{

**return** (a+b);

}

**35. short:** In java, the short keyword declares variables that can hold an integer value of 16-bit long.

For Example:

short num=1;

**36. static:** In java, the static keyword declares a static variable or method. A static variable or method is stored in a static memory location.

For Example:

public static **void** main()

**37. strictfp:** In java, strictfp keyword ensures that every platform gets the same result for floating-point operation. The strictfp keyword is used on classes, methods and interfaces. This keyword was introduced in JDK 1.2.

For Example:

strictfp **class** DataFlair{}

**38. super:** The super keyword distinguishes variables with the same name in both parent class and child class in inheritance.

For Example:

**class** DataFlair

{

String name=”Company”;

}

**class** internship **extends** DataFlair/

{

String name=”Intern”;

System.out.println(name);//Intern will be printed

System.out.println(super.name);//Company will be printed

}

**39. switch:** Using the switch keyword we can initiate the switch case block in a java code.

For Example:

Switch(num)

{

case 1:

System.out.println(“ONE”);

break;

default:

System.out.println(“Not ONE”);

}

**40. synchronized:** Using the synchronized keyword in java we can specify the critical section to be executed during multithread coding.

For Example:

synchronized **void** print(int num)

**41. this:** this keyword in java is used to distinguish local variables from global variables.

For example:

**class** DataFlair{

String name;

DataFlair(String name)

{

**this**.name=name;

}

}

**42. throw:** The throw keyword in java is used to throw custom made exceptions explicitly into the code.

For Example:

throw **new** IOException("The Input is Not Acceptable”);

**43. throws:** The throws keyword is used to propagate a checked exception.

For Example:

import java.io.\*;

**class** DataFlair{

**void** internship()throws IOException{

throw **new** IOException("Internship Not Found");

}

}

**44. transient:** Using the transient keyword in java we can deserialize a variable. When a variable is declared transient, it is not considered for serialization.

For Example:

transient int num;

**45. try:** The try block is used in exception handling to test a block of code for exceptions. The try block is always followed by a catch block or finally block.

For Example:

**try**{

//statement

}**catch**(exception e){}

**46. void:** Using the void keyword we can declare that the method will not return any value.

For Example:

**void** main()

**47. volatile:** volatile keyword in java is used to indicate that a variable can change asynchronously.

For Example:

static volatile int num=100;

**48. while:** Using the While keyword we can declare a while loop. While loop is an entry controlled loop, where the number of iterations is not fixed.

For example:

i=5;

**while**(i<=10)

{

i++;

}

**Note:** There are two other keywords that are reserved but not used, they are **const** and **goto.** Words like true, false and null might seem like keywords, but they are actually literals.

# Java Control Statements | Control Flow in Java

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

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

### **Decision-Making statements:**

As the name suggests, decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in Java, i.e., If statement and switch 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.JDK, JRE, and JVM

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.

1. **if**(condition) {
2. statement 1;
3. statement 2; //executes when condition is true
4. }

Consider the following example in which we have used the **if** statement in the java code.

Student.java

**Student.java**

**public** **class** Student {

**public** **static** **void** main(String[] args) {

**int** x = 10;

**int** y = 12;

**if**(x+y > 20) {

System.out.println("x + y is greater than 20");

}

}

}

**Output:**

x + y is greater than 20

### **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:**

1. **if**(condition) {
2. statement 1; //executes when condition is true
3. }
4. **else**{
5. statement 2; //executes when condition is false
6. }

Consider the following example.

**Student.java**

**public** **class** Student {

**public** **static** **void** main(String[] args) {

**int** x = 10;

**int** y = 12;

**if**(x+y < 10) {

System.out.println("x + y is less than      10");

}   **else** {

System.out.println("x + y is greater than 20");

}

}

}

**Output:**

x + y is greater than 20

### **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.

1. **if**(condition 1) {
2. statement 1; //executes when condition 1 is true
3. }
4. **else** **if**(condition 2) {
5. statement 2; //executes when condition 2 is true
6. }
7. **else** {
8. statement 2; //executes when all the conditions are false
9. }

Consider the following example.

**Student.java**

**public** **class** Student {

**public** **static** **void** main(String[] args) {

String city = "Delhi";

**if**(city == "Meerut") {

System.out.println("city is meerut");

}

**else** **if** (city == "Noida") {

System.out.println("city is noida");

}**else** **if**(city == "Agra") {

System.out.println("city is agra");

}**else** {

System.out.println(city);

}

}

}

**Output:**

Delhi

### **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

}

}

Consider the following example.

**Student.java**

**public** **class** student {

**public** **static** **void** main(String[] args) {

String address = "Delhi, India";

**if**(address.endsWith("India")) {

**if**(address.contains("Meerut")) {

System.out.println("Your city is Meerut");

}**else** **if**(address.contains("Noida")) {

System.out.println("Your city is Noida");

}**else** {

System.out.println(address.split(",")[0]);

}

}**else** {

System.out.println("You are not living in India");

}

}

}

**Output:**

Delhi

### **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.

Points to be noted about switch statement:

* The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java
* Cases cannot be duplicate
* Default statement is executed when any of the case doesn't match the value of expression. It is optional.
* Break statement terminates the switch block when the condition is satisfied.  
  It is optional, if not used, next case is executed.
* While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

The syntax to use the switch statement is given below.

1. **switch** (expression){
2. **case** value1:
3. statement1;
4. **break**;
5. .
6. .
7. .
8. **case** valueN:
9. statementN;
10. **break**;
11. **default**:
12. **default** statement;
13. }

Consider the following example to understand the flow of the switch statement.

**Student.java**

**public** **class** Student **implements** Cloneable {

**public** **static** **void** main(String[] args) {

**int** num = 2;

**switch** (num){

**case** 0:

System.out.println("number is 0");

**break**;

**case** 1:

System.out.println("number is 1");

**break**;

**default**:

System.out.println(num);

}

}

}

**Output:**

2

While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value. The switch permits only int, string, and Enum type variables to be used.

### **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.

1. **for**(initialization, condition, increment/decrement) {
2. //block of statements
3. }

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



Consider the following example to understand the proper functioning of the for loop in java.

**Calculation.java**

**public** **class** Calculattion {

**public** **static** **void** main(String[] args) {

 // print sum of first 10 natural number

**int** sum = 0;

**for**(**int** j = 1; j<=10; j++) {

sum = sum + j;

}

System.out.println("The sum of first 10 natural numbers is " + sum);

}

}

**Output:**

The sum of first 10 natural numbers is 55

### **Java for-each loop**

Java provides an enhanced for loop to traverse the data structures like array or collection. In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

1. **for**(data\_type var : array\_name/collection\_name){
2. //statements
3. }

Consider the following example to understand the functioning of the for-each loop in Java.

**Calculation.java**

**public** **class** Calculation {

**public** **static** **void** main(String[] args) {

// TODO Auto-generated method stub

String[] names = {"Java","C","C++","Python","JavaScript"};

System.out.println("Printing the content of the array names:\n");

**for**(String name:names) {

System.out.println(name);

}

}

}

**Output:**

Printing the content of the array names:

Java

C

C++

Python

JavaScript

### **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.

1. **while**(condition){
2. //looping statements
3. }

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



Consider the following example.

**Calculation .java**

**public** **class** Calculation {

**public** **static** **void** main(String[] args) {

// Printing the list of first 10 even numbers

**int** i = 0;

System.out.println("Printing the list of first 10 even numbers \n");

**while**(i<=10) {

System.out.println(i);

i = i + 2;

}

}

}

**Output:**

Printing the list of first 10 even numbers

0

2

4

6

8

10

### **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.

1. **do**
2. {
3. //statements
4. } **while** (condition);

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



Consider the following example to understand the functioning of the do-while loop in Java.

**Calculation.java**

**public** **class** Calculation {

**public** **static** **void** main(String[] args) {

// TODO Auto-generated method stub

**int** i = 0;

System.out.println("Printing the list of first 10 even numbers \n");

**do** {

System.out.println(i);

i = i + 2;

}**while**(i<=10);

}

}

**Output:**

Printing the list of first 10 even numbers

0

2

4

6

8

10

### **Jump Statements**

Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

### **Java break statement**

As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

**The break statement example with for loop**

Consider the following example in which we have used the break statement with the for loop.

**BreakExample.java**

**public** **class** BreakExample {

**public** **static** **void** main(String[] args) {

// TODO Auto-generated method stub

**for**(**int** i = 0; i<= 10; i++) {

System.out.println(i);

**if**(i==6) {

**break**;

}

}

}

}

**Output:**

0

1

2

3

4

5

6

**break statement example with labeled for loop**

**Calculation.java**

1. **public** **class** Calculation {
3. **public** **static** **void** main(String[] args) {
4. // TODO Auto-generated method stub
5. a:
6. **for**(**int** i = 0; i<= 10; i++) {
7. b:
8. **for**(**int** j = 0; j<=15;j++) {
9. c:
10. **for** (**int** k = 0; k<=20; k++) {
11. System.out.println(k);
12. **if**(k==5) {
13. **break** a;
14. }
15. }
16. }
18. }
19. }

22. }

**Output:**

0

1

2

3

4

5

### **Java continue statement**

Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Consider the following example to understand the functioning of the continue statement in Java.

**public** **class** ContinueExample {

**public** **static** **void** main(String[] args) {

// TODO Auto-generated method stub

**public** **class** example {

**public** **static** **void** main(String[] args) {

**for**(**int** i =1;i<=10;i++) {

**if**(i>4 && i<9) {

**continue**;

}

System.***out***.println(i);

}

}

}**Output:**

1

2

3

4

9

10