**Path**

**-**core java 8

-Unit testing

-SQL

-DevOps

-Hibernate 3.0

-Springs 5.0

-HTML,CSS 3 with bootstrap

-JavaScript

-Angular 7

**What is java?**

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

Developed by sun microsystems

High 1 low 0 **Computer**

**-Why 0’s and 1**

**-Electric devices**

**Binary**

**Decimal**

**0 ---------------------------------> 0**

**1 ---------------------------------->1**

**2 --------------------------------->10**

**3 --------------------------------->11**

**4 ---------------------------------->100**

**-High level language comes into the picture.**

**-complier convert code into 0 and 1.**

**Where we can use java?**

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](https://www.javatpoint.com/servlet-tutorial), [JSP](https://www.javatpoint.com/jsp-tutorial), [Struts](https://www.javatpoint.com/struts-2-tutorial), [Spring](https://www.javatpoint.com/spring-tutorial), [Hibernate](https://www.javatpoint.com/hibernate-tutorial), [JSF](https://www.javatpoint.com/jsf-tutorial), 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](https://www.javatpoint.com/ejb-tutorial) 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.

**Program:-**

**For e.g. visit on some place**

**No** Yes

**1. Start**

**2. Search on map**

**3. Options (Vehicles)**

**Search on map**

**4. Timings**

**5. Exit**

**Go**

**Open/Close**

**close**

**Open**

**Options**

**History of java:**

**-James Gosling 1995 oak tree**

**-version JDK 1.0 Jan 23 1996**

**First java program:-**

**Installation of JDK**

**1)JDK**

**2)IDE**

**-Visual Studio**

**-Intellij**

**-Eclipse**

**-notepad**

**-How java program is run?**

**1) Compilation:-**

.class file (Byte code

.java File

**Byte Code**

**Source code**

**2) Execution:-**

Native code

**Byte Code**

**JRE**

**JDK**

**Program component:**

1. **Function**
2. **Class**

**Output**

**\***

**\*\***

**System.out.println(“\*”);**

**System.out.print(“\*\*”);**

**\*\*naming conventions**

**\*\*Fundamentals output,input,variables dataype**

**\*\*conditional statements**

**\*\* loops**

**\*\*\*\*Identifiers and keywords**

## List of Java Keywords

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

A list of Java keywords or reserved words are given below:

1. [**abstract**](https://www.javatpoint.com/abstract-keyword-in-java)**:** Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
2. [**boolean:**](https://www.javatpoint.com/boolean-keyword-in-java) Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. [**break**](https://www.javatpoint.com/java-break)**:** Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.
4. [**byte**](https://www.javatpoint.com/byte-keyword-in-java)**:** Java byte keyword is used to declare a variable that can hold 8-bit data values.
5. [**case**](https://www.javatpoint.com/case-keyword-in-java)**:** Java case keyword is used with the switch statements to mark blocks of text.
6. [**catch**](https://www.javatpoint.com/try-catch-block)**:** Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
7. [**char**](https://www.javatpoint.com/char-keyword-in-java)**:** Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. [**class**](https://www.javatpoint.com/class-keyword-in-java)**:** Java class keyword is used to declare a class.
9. [**continue**](https://www.javatpoint.com/java-continue)**:** Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
10. [**default**](https://www.javatpoint.com/default-keyword-in-java)**:** Java default keyword is used to specify the default block of code in a switch statement.
11. [**do**](https://www.javatpoint.com/java-do-while-loop)**:** Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.
12. [**double**](https://www.javatpoint.com/double-keyword-in-java)**:** Java double keyword is used to declare a variable that can hold 64-bit floating-point number.
13. [**else**](https://www.javatpoint.com/java-if-else)**:** Java else keyword is used to indicate the alternative branches in an if statement.
14. [**enum**](https://www.javatpoint.com/enum-in-java)**:** Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
15. [**extends**](https://www.javatpoint.com/inheritance-in-java)**:** Java extends keyword is used to indicate that a class is derived from another class or interface.
16. [**final**](https://www.javatpoint.com/final-keyword)**:** Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.
17. [**finally**](https://www.javatpoint.com/finally-block-in-exception-handling)**:** Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.
18. [**float**](https://www.javatpoint.com/float-keyword-in-java)**:** Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
19. [**for**](https://www.javatpoint.com/java-for-loop)**:** Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.
20. [**if**](https://www.javatpoint.com/java-if-else)**:** Java if keyword tests the condition. It executes the if block if the condition is true.
21. [**implements**](https://www.javatpoint.com/interface-in-java)**:** Java implements keyword is used to implement an interface.
22. [**import**](https://www.javatpoint.com/package)**:** Java import keyword makes classes and interfaces available and accessible to the current source code.
23. [**instanceof**](https://www.javatpoint.com/downcasting-with-instanceof-operator)**:** Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
24. [**int**](https://www.javatpoint.com/int-keyword-in-java)**:** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
25. [**interface**](https://www.javatpoint.com/interface-in-java)**:** Java interface keyword is used to declare an interface. It can have only abstract methods.
26. [**long**](https://www.javatpoint.com/long-keyword-in-java)**:** Java long keyword is used to declare a variable that can hold a 64-bit integer.
27. **native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
28. [**new**](https://www.javatpoint.com/new-keyword-in-java)**:** Java new keyword is used to create new objects.
29. [**null**](https://www.javatpoint.com/null-keyword-in-java)**:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
30. [**package**](https://www.javatpoint.com/package)**:** Java package keyword is used to declare a Java package that includes the classes.
31. [**private**](https://www.javatpoint.com/private-keyword-in-java)**:** Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
32. [**protected**](https://www.javatpoint.com/protected-keyword-in-java)**:** Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.
33. [**public**](https://www.javatpoint.com/public-keyword-in-java)**:** Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
34. [**return**](https://www.javatpoint.com/return-keyword-in-java)**:** Java return keyword is used to return from a method when its execution is complete.
35. [**short**](https://www.javatpoint.com/short-keyword-in-java)**:** Java short keyword is used to declare a variable that can hold a 16-bit integer.
36. [**static**](https://www.javatpoint.com/static-keyword-in-java)**:** Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.
37. [**strictfp**](https://www.javatpoint.com/strictfp-keyword)**:** Java strictfp is used to restrict the floating-point calculations to ensure portability.
38. [**super**](https://www.javatpoint.com/super-keyword)**:** Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.
39. [**switch**](https://www.javatpoint.com/java-switch)**:** The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
40. [**synchronized**](https://www.javatpoint.com/synchronization-in-java)**:** Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
41. [**this**](https://www.javatpoint.com/this-keyword)**:** Java this keyword can be used to refer the current object in a method or constructor.
42. [**throw**](https://www.javatpoint.com/throw-keyword)**:** The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.
43. [**throws**](https://www.javatpoint.com/throws-keyword-and-difference-between-throw-and-throws)**:** The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.
44. [**transient**](https://www.javatpoint.com/transient-keyword)**:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
45. [**try**](https://www.javatpoint.com/try-catch-block)**:** Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
46. **void:** Java void keyword is used to specify that a method does not have a return value.
47. [**volatile**](https://www.javatpoint.com/volatile-keyword-in-java)**:** Java volatile keyword is used to indicate that a variable may change asynchronously.
48. [**while**](https://www.javatpoint.com/java-while-loop)**:** Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.

**Identifiers in Java**

Identifiers in Java are symbolic names used for identification. They can be a class name, variable name, method name, package name, constant name, and more. However, In [Java](https://www.javatpoint.com/java-tutorial), There are some reserved words that can not be used as an identifier.

For every identifier there are some conventions that should be used before declaring them. Let's understand it with a simple Java program:

### Rules for Identifiers in Java

There are some rules and conventions for declaring the identifiers in Java. If the identifiers are not properly declared, we may get a compile-time error. Following are some rules and conventions for declaring identifiers:

* A valid identifier must have characters [A-Z] or [a-z] or numbers [0-9], and underscore(\_) or a dollar sign ($). for example, @javatpoint is not a valid identifier because it contains a special character which is @.
* There should not be any space in an identifier. For example, java tpoint is an invalid identifier.
* An identifier should not contain a number at the starting. For example, 123javatpoint is an invalid identifier.
* An identifier should be of length 4-15 letters only. However, there is no limit on its length. But, it is good to follow the standard conventions.
* We can't use the Java reserved keywords as an identifier such as int, float, double, char, etc. For example, int double is an invalid identifier in Java.
* An identifier should not be any query language keywords such as SELECT, FROM, COUNT, DELETE, etc.

**\*\*\*Naming conventions\*\*\*\***

Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc

-not forced to follow

**printPattern();**

**printOddNumbers();**

|  |  |  |
| --- | --- | --- |
| **Identifiers Type** | **Naming Rules** | **Examples** |
| Class | It should start with the uppercase letter. It should be a noun such as Color, Button, System, Thread, etc. Use appropriate words, instead of acronyms. | public  class **Employee** { //code  snippet } |
| Interface | It should start with the uppercase letter. It should be an adjective such as Runnable, Remote, ActionListener. Use appropriate words, instead of acronyms. | interface **Printable** { //codesnippet } |
| Method | It should start with lowercase letter. It should be a verb such as main(), print(), println(). If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed(). | Class  Employee { //method void **draw()** { //codesnippet } } |
| Variable | It should start with a lowercase letter such as id, name. It should not start with the special characters like & (ampersand), $ (dollar), \_ (underscore). If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName. Avoid using one-character variables such as x, y, z. | classEmployee { //variable int **id**; //codesnippet } |
| Package | It should be a lowercase letter such as java, lang. If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang. | //package package **com.javatpoint;** classEmployee { //codesnippet } |
| Constant | It should be in uppercase letters such as RED, YELLOW. If the name contains multiple words, it should be separated by an underscore(\_) such as MAX\_PRIORITY. It may contain digits but not as the first letter. | classEmployee { //constant static final int **MIN\_AGE** = 18; //codesnippet } |

**\*\*\*\*\*\*Variables\*\*\*\*\*\***

**-bulding blocks**

**Float price=2.50;**

**-**

**a b**

**a=25 b=10**

**2\*(a+b)**

**Int a = 25;**

**String name = “Ms Dhoni”;**

a b

**Every difference data stored in different size**

Ms

Dhoni

30

10

.

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.

A variable is a container which holds the value while the [Java program](https://www.javatpoint.com/simple-program-of-java) 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.

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. }

#### 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](https://www.javatpoint.com/static-keyword-in-java).

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.

**\*\*\*\*\*Datatype\*\*\*\***

**\*\*Data type\*\*\***

**Primitive Non-Primitive**

**1)byte 🡪1 byte String name=”prachi”;**

**2)short Array**

**3)char 2 bytes = ‘a’ class**

**4)Boolean 1byte object**

**5)int 4 interface**

**6)long 8**

**7)float 4**

**8)double 8**

|  |  |  |
| --- | --- | --- |
| **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 |

## 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 = 10, **int** b = -20

## 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:**

**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'

**Type casting:**

In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer. In this section, we will discuss **type casting** and **its types** with proper examples.

Convert a value from one data type to another data type is known as **type casting**

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. }}

There are two types of type casting:

* Widening Type Casting
* Narrowing Type Casting

### Widening Type Casting

Converting a lower data type into a higher one is called **widening** type casting. It is also known as **implicit conversion** or **casting down**. It is done automatically. It is safe because there is no chance to lose data. It takes place when:

1. **public** **class** WideningTypeCastingExample
2. {
3. **public** **static** **void** main(String[] args)
4. {
5. **int** x = 7;
6. //automatically converts the integer type into long type
7. **long** y = x;
8. //automatically converts the long type into float type
9. **float** z = y;
10. System.out.println("Before conversion, int value "+x);
11. System.out.println("After conversion, long value "+y);
12. System.out.println("After conversion, float value "+z);
13. }
14. }

### Narrowing Type Casting

Converting a higher data type into a lower one is called **narrowing** type casting. It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer. If we do not perform casting then the compiler reports a compile-time error.

1. **double** -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**

Let's see an example of narrowing type casting.

In the following example, we have performed the narrowing type casting two times. First, we have converted the double type into long data type after that long data type is converted into int type.

1. **public** **class** NarrowingTypeCastingExample
2. {
3. **public** **static** **void** main(String args[])
4. {
5. **double** d = 166.66;
6. //converting double data type into long data type
7. **long** l = (**long**)d;
8. //converting long data type into int data type
9. **int** i = (**int**)l;
10. System.out.println("Before conversion: "+d);
11. //fractional part lost
12. System.out.println("After conversion into long type: "+l);
13. //fractional part lost
14. System.out.println("After conversion into int type: "+i);
15. }
16. }

**\*\*\*\*\*\*\*\*\*\*Input\*\*\*\*\*\*\*\*\***

**Public class Example{**

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

**Scanner sc = new Scanner(System.in);**

**String name =sc.next();**

**Sout(name);**

**}**

**}**

**Name of student:**

**Subj 1**

**Subj 2**

**Sc.nextLine();**

**Sc.nextFloat()**

**\*\*\*\*take input and do addition\*\*\***

# Operators in Java

**Operator** in [Java](https://www.javatpoint.com/java-tutorial) 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 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
* ++a

### 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++);
5. System.out.println(++x);
6. System.out.println(x--);
7. System.out.println(--x);
8. }}

### 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. }}

### 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. }}

### 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. }}

### 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. }}

### 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. }}

**\*\*\*Conditional statements\*\*\***

* if statement

The Java if statement tests the condition. It executes the *if block* if condition is true.

**Syntax:**

1. **if**(condition){
2. //code to be executed
3. }

Eg age

## Java if-else Statement

The Java if-else statement also tests the condition. It executes the if block if condition is true otherwise else block is executed.

**Syntax:**

1. **if**(condition){
2. //code if condition is true
3. }**else**{
4. //code if condition is false
5. }

Odd or even

## Java if-else-if ladder Statement

1. **if**(condition1){
2. //code to be executed if condition1 is true
3. }**else** **if**(condition2){
4. //code to be executed if condition2 is true
5. }
6. **else** **if**(condition3){
7. //code to be executed if condition3 is true
8. }
9. ...
10. **else**{
11. //code to be executed if all the conditions are false
12. }

**Switch:-**

**India USA Japan**

**switch**(expression){

**case** value1:

 //code to be executed;

**break**;  //optional

**case** value2:

 //code to be executed;

**break**;  //optional  ......

**default**:

  code to be executed **if** all cases are not matched;

}

Scanner sc = new Scanner(System.in);

Int button = sc.nextInt();

switch