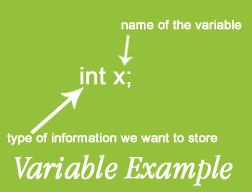
Variables In JAVA With Examples – Complete Tutorials

Variables is something that makes computer program very useful by storing information in memory. When you think of variables think of boxes which is used to store something. In computer world you can have hundreds, thousands or more number of boxes (mean variables) each containing their own pieces of information. That box is called variables in JAVA.

While creating variables in JAVA we need to assign two things. First is the type of information we want to store and second is the name of the variable. Below is an example of variable where int is a type of information (integer in this case) and x is the name:



int x;

Here int is integer [primitive data type](https://abhiandroid.com/java/primitive-data-types-in-java) and x is the variable.

**Table of Contents**[[hide](https://abhiandroid.com/java/variables-with-examples)]

* [1 Syntax Of Variables:](https://abhiandroid.com/java/variables-with-examples" \l "Syntax_Of_Variables)
* [2 Variables Examples In JAVA:](https://abhiandroid.com/java/variables-with-examples" \l "Variables_Examples_In_JAVA)
* [3 Types Of Variables](https://abhiandroid.com/java/variables-with-examples" \l "Types_Of_Variables)
* [4 How To Print Variable Value:](https://abhiandroid.com/java/variables-with-examples" \l "How_To_Print_Variable_Value)
* [5 Importance Of Variables In JAVA:](https://abhiandroid.com/java/variables-with-examples" \l "Importance_Of_Variables_In_JAVA)

**Syntax Of Variables:**

**What Is Syntax:** Syntax in programming language basically means combinations of symbols based on the set of rules defined for that language. JAVA has its own rules for syntax of variables and we need to follow it.

Syntax of variables in JAVA is divided in 3 parts: first assigning type of data you are storing (int,char,byte etc.), second is naming (let’s say x) and third is putting semicolon(;) which is telling program statement has end here.

int x;

You can also store value directly here if you want by putting value after = (which is an assignment operator):

int x = 5;

In above syntax we are storing integer(number without any decimal point). In JAVA “int” is used to assign integer.

**Variables Examples In JAVA:**

Let us create a simple example of variables storing student details:

int rollnumber = 3;

char name = 'X';

In the above example we have defined two variables rollnumber and name for storing student details.

**Important Note:** char only store one character. If you want to store full name then you need to use String.

Defining variable for storing character say A

char name =’A’;

Defining variable for storing decimal number say 500.36

float income = 500.36;

**Above Example Explanation:**

* int, char and float are data type which is telling program to assign area in memory for storing integer, character or decimal number
* rollnumber, name & income are names of variables and value 3, abhishek & 500.36 is stored in them.

**Types Of Variables**

There are 3 types of variables in JAVA:

**1. Local Variables:** These can be defined inside method, constructor or also inside block. The scope or life time of local variable destroyed with end of method completion.  
**2. Instance variables:** These are associated with the object creation. As the object get created instance variable also get created  
**3. Class Variable/static variables:** These are loaded and initialized when class is loaded in JVM and there exists only one copy of class variable

**Examples Of Types Of Variables:**

public class TypeOfVariable{

public static int staticvariable;

int instancevariable;

public void printValue(){

int localvariable;

System.out.println("the value of staticvariable \t"+staticvariable);

System.out.println("the value of instancevariable\t"+instancevariable);

System.out.println("the value of localvariable \t"+localvariable);

}

public static void main(String args[]){

TypeOfVariable object=new TypeOfVariable();

object.printValue();

}

}

For more details read [Type Of Variable Tutorial With Example](https://abhiandroid.com/java/types-of-variables.html)

**How To Print Variable Value:**

Here we will show you how easily we can print variable value in JAVA by using inbuilt JAVA method System.out.println();. Here System is class, out is an object and println is a method for printing. If you don’t understand class, objects and methods don’t worry about it. Right now just remember System.out.println(); is a method for printing in JAVA.

Now lets print a number 5 in JAVA. First we will create a variable and store value 5 in it. Remember 5 is an integer so we will assign int as a data type:

int x = 5;

Now using inbuilt JAVA method for printing it:

System.out.println(x);

Complete code:

int x = 5;

System.out.println(x);

Output:

5

Here JAVA look for value stored in the memory of variable name x, finds it and then print it which is 5.

**Important Note:** Other than simply printing variables we can use them for many other purpose like adding, subtracting etc. which we will share in the Operators topic.

**Importance Of Variables In JAVA:**

* It makes the computer program very useful by storing information which can be easily accessed. For example if you want to create a JAVA program which can calculate current age based on birthday of the person. The person will enter his birthday details which needs to be stored somewhere in computer with proper naming to make it easily accessible for calculating his age. Here you need variables to do that for you.
* In simple words, you need variables in program because it gives the ability to store information which can then be processed. Without variables computer won’t be able to process anything.

Primitive Data Types In JAVA With Examples

When we say Data Type think of type of information you want to store in [variables](https://abhiandroid.com/java/variables-with-examples) like integer, character, decimal etc. Primitive means this data type are pre-defined by JAVA language.

JAVA has 8 reserved keyword for primitive data type for assigning 8 different type of information based on value (type of information) and byte (memory or space). For example int is used for a 32-bit (4-byte) integer value, char for 16-bit character, boolean for true or false value, short for 16-bit (2-byte) integer value etc.

In JAVA this data type must be declared with variables when creating them. This helps compiler to ensure we are inserting the right type of data in variables which we have assigned with a particular data type.

**Table of Contents**[[hide](https://abhiandroid.com/java/primitive-data-types-in-java)]

* [1 Primitive Data Types With Examples:](https://abhiandroid.com/java/primitive-data-types-in-java" \l "Primitive_Data_Types_With_Examples)
* [2 Different Types Of Data Type In JAVA:](https://abhiandroid.com/java/primitive-data-types-in-java" \l "Different_Types_Of_Data_Type_In_JAVA)
* [3  String:](https://abhiandroid.com/java/primitive-data-types-in-java" \l "String)

**Primitive Data Types With Examples:**

Lets try to understand the concept with examples of int data type which is used for assigning integer (number without a decimal point). Suppose we want to store initial value 5 in x variables:

int x = 5;

But what if we want to store 5.2 in integer data type:

int x = 5.2;

The compiler will throw error because we are trying to store decimal value in integer data type.

**Important Note:** One more important thing to note is that [data type](https://abhiandroid.com/java/primitive-data-types-in-java) also assign the limit of space that can be store in variables. In computing terms there is always a limit of value minimum and maximum that can be stored. For example int can store minimum value of -231 and a maximum value of 231-1. If you try to store higher value than 231-1 or lesser value than -231 then compiler will throw an error.

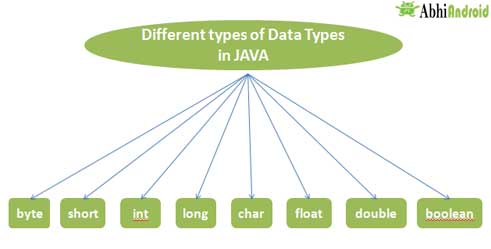
Suppose we try to insert value 2,147,483,648 in integer variable:

int x = 2,147,483,648;

The compiler will give an error because we are trying to store higher value than 231-1. If you want to store that particular value then you have to use long data type instead of int. This concept will make more sense as you proceed in this article where we will discuss 8 different Primitive data types in JAVA.

**Different Types Of Data Type In JAVA:**

In JAVA there are total of 8 different primitive data types. Primitive types means this data types are predefined by JAVA language and has a reserved keyword.



**Below are the 8 different primitive data types:**

**byte:**

It is a 8-bit (1-byte) integer value which has a minimum value of -128 and a maximum value of 127. Byte examples:

byte x = 120;

Suppose if we try to store -200 in byte data type:

byte x = -200;

Compiler will throw an error for exceeding the limits of memory allocated for byte type.

**short:**

It is a 16-bit (2-byte) integer value which has a minimum value of -32,768 and a maximum value of 32,767. Example for short data type:

short x = 30000;

**int:**

It is a 32-bit (4-byte) integer value which has a minimum value of -231 and a maximum value of 231-1. int is the most common, preferred and widely used data type in Android. Example for int data type:

int x = 1245678;

**long:**

It is a 64-bit (8-byte) integer value which has a minimum value of -263 and a maximum value of 263-1. This data type is also preferred in Android for storing very large value like game high score which can be in billions. Example for long data type:

long x = 12345987609L;

Remember to use either ‘l’ or ‘L’ in long data type after the value because it tells the computer we are storing value in long data type.

**char:**

It is a 16-bit character using the Unicode encoding scheme has a minimum value of ‘\u0000’ (or 0) and a maximum value of ‘\uffff’. Example of char data type:

char myChar = ‘0’;

In char we use single quote for representing character.

**Important Note:** In char data type we can only store one character, number, special character or Unicode character. To store more than one character we use String. We will discuss basics of string in this article after finishing eight primitive data type.

Lets try to put more than one character in char data type:

char myChar = ‘AB’;

You will get an error “too many character”.

**float:**

It is a single-precision 32-bit (4-byte) floating-point value. Example for float data type:

float myNumber = 5.25f;

**Important Note:** Put f after float number because it is good practice and by default a decimal number in JAVA is assume as a double.

**double:**

It is a double precision 64-bit (8-byte) floating-point value. Example of double data type:

double myNumber = 5.234d;

or

double myNumber = 5.234;

Both are accepted as by default decimal number is assume as double in JAVA. But prefer putting d because it is considered as good practice in good coding.

**boolean:**

boolean data type has only two possible values: true and false. It is very useful for conditional logic. Example of Boolean data type:

boolean isEven = true;

**String:**

First of all String is not a primitive data type(predefined in JAVA language) but very important to learn as it is used so commonly that it can be considered as 9th data type in JAVA. It is basically a class.

**Important Note:** Here we are only sharing basic of String so that you can understand the next topic in JAVA where we will use it. Just want to add a notice here, String is a big topic and full tutorial on String will be added later.

**What Is String:**

String is a sequence of characters. As we saw previously char is limited to store just one character or Unicode character while String can store sequence of characters.

String x = “This is a String”;

Here we use double quote to represent a string which is storing characters.

String In JAVA With Example

String is the most commonly used class in Java programming language. In java every string we create is an object of String Class that implements *Serializable*, *Comparable* and *CharSequence* interfaces. It is nothing but a character array for example “AbhiAndroid” is a string of 11 characters as shown.

char[] test= { 'A' , 'b' , 'h' , 'i' , 'A' , 'n' , 'd' , 'r' , 'o' , 'i' , 'd' };

String  testString= new String(test);

Strings are Immutable in nature, which means once string is created its value cannot be altered, but we can create a new Instance of it.

**Immutable**: Any object whose state cannot be altered once created are called Immutable objects. String, Integer, Byte, Short, all other wrapper class objects are immutable in nature.

**Table of Contents**[[hide](https://abhiandroid.com/java/string)]

* [1 Creating a String In JAVA:](https://abhiandroid.com/java/string" \l "Creating_a_String_In_JAVA)
* [2 String Important Points In JAVA:](https://abhiandroid.com/java/string" \l "String_Important_Points_In_JAVA)
* [3 String Quick Revision Points:](https://abhiandroid.com/java/string" \l "String_Quick_Revision_Points)

**Creating a String In JAVA:**

There are two ways by which String can be created:

1. Using String literal
2. Using New keyword

**Using String Literal**

Strings can be created very easily, by assigning a string value to the string literal as shown:

String string1 = "Welocome to AbhiAndroid";

String string2 = "Welocome to AbhiAndroid";

As, discussed above String is a class, but we have not created any object above using new keyword, so how new object is created? Don’t worry compiler did your task. But problem here is that, if the object is already present in the memory compiler do not create a new object rather it assigns the same old object to the new instance created, which means even though we have two string instances above string1 and string2, compiler only creates one string object having the value “Welocome to AbhiAndroid” and assigns the same to both the instances String1 and String2.

**Using New keyword**:

As we saw above, by using only String literal compiler assigned the same string object to two different string literals, To overcome this approach we can create string using new keyword as shown

String string1 = new String(" Welocome to AbhiAndroid ");

String string2 = new String(" Welocome to AbhiAndroid ");

In this approach compiler will create two different objects in memory having the same value.

**String Important Points In JAVA:**

1. Strings in Java unlike C, C++ are not null Terminated as String is a Class which is backed by Character array.
2. Strings are Immutable and Final in Java, which means once created we cannot modify its content.
3. Strings are maintained in a String pool, which is a special memory location. Any time we create a new string using String Literal, JVM first checks String pool and if any object which is similar to the content is available, then it returns the same and doesn’t create a new object. However JVM doesn’t perform String pool check if object is created using new operator.
4. For comparing two String objects or we can say to check whether they are same or not, equals()method is used instead of equality(==) operator. String class overrides equals() method that provides content equality, which checks characters, case and order.
5. Avoid storing sensitive data in String like passwords, bank account information etc. As Strings are immutable, there is no way to erase the content from String because data is stored in java heap for longer time and which exposes risk of being seen by anyone who has access to Java memory, like reading from memory dump.

**String Quick Revision Points:**

* String is a sequence of characters, or we can say similar to character array
* String is a java is not null terminated
* Strings are immutable and final
* Strings are maintained in a String pool
* To compare contents of String equals() method is used instead of == operator

Use character array for storing sensitive information likes passwords instead of Strings.

## Operator In JAVA With Examples

In computer programming Operator is a symbol that tells the compiler to perform specific action which can be mathematical or logical. For example: ‘+’ is a additive operator which does addition of two number and can also be used for String concatenation.

Lets take simple example to understand operator:

int x = 4 + 5 – 7;

In the above example:

* ‘+’ and ‘–‘ are addition and subtraction operator respectively which comes under Arithmetic Operator.)
* “4”, “5” and “7” are operand (Operand are objects that are manipulated by operator).
* “=” is an assignment operator which assign result of calculation (value 2 in this case) to x.

**Important Note:** Just want to make a short notice here after learning different types of operator in JAVA don’t miss the next topic operator precedence which is shared just below this topic. Operator learning is incomplete without knowing operator precedence.

**Table of Contents**[[hide](https://abhiandroid.com/java/operators-in-java)]

* [1 Different Types Of Operator In JAVA:](https://abhiandroid.com/java/operators-in-java" \l "Different_Types_Of_Operator_In_JAVA)
* [2 Operator Precedence:](https://abhiandroid.com/java/operators-in-java" \l "Operator_Precedence)

#### ****Different Types Of Operator In JAVA:****

JAVA supports 7 types of operator:

**SIMPLE ASSIGNMENT OPERATOR:**

“=” is assignment operator in JAVA language which assign the value of its right side to its left side value. The right side value can be of variable, constant or [expression](https://abhiandroid.com/java/expression-statement-code-blocks-in-java) that result some value and left side can be a variable or object which has the capacity to possess that right side value . It is evaluated from right to left.

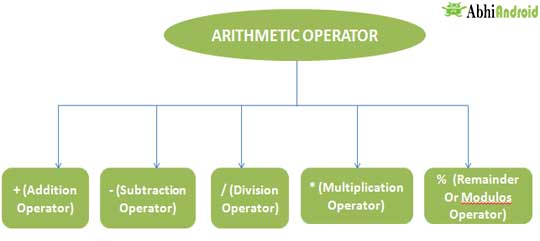
Example of assignment operator:

int x = 5;

Here value 5 is assigned to integer variable x.

**ARITHMETIC OPERATOR:**

There are 5 types of arithmetic operator in JAVA: addition (+), subtraction (-), division (/), multiplication (\*) and remainder (%) operator.



**+ (Addition Operator):**

“+” is binary operator which adds two operands. Example of + operator:

int x = 10;

int y = 5;

int z = x + y;

Here in the above example the value of operands x and y (which is 10 and 5) added by the + operator and total value 15 is assigned to a new variable z.

**– (Subtraction Operator):**

“-“ is a binary operator which subtracts second operand from the first. Example of – operator:

int x = 10;

int y = 5;

int z = x – y;

In the above example value of y operands subtracts from x operands by – operator and result value is then assigned to new variable z.

**/ (Division Operator):**

/ is a binary operator which divide numerator by denominator. Example of division operator:

int x = 10;

int y = 5;

int z = x/y;

In the above example, value of x is divided by y by “/” operator and result value 2 is assigned to new variable z.

**\* (Multiplication Operator):**

\* is a binary operator which multiplies both the operands. Example of Multiplication operator:

int x = 10;

int y = 5;

int z = x\*y;

In the above example, value of x and y operands is multiplied by “\*” operator and result value 50 is assigned to z variable.

**% (Remainder Or Modulos Operator):**

% operator gives the remainder after an integer division of two operands. Example of remainder operator:

int x = 12;

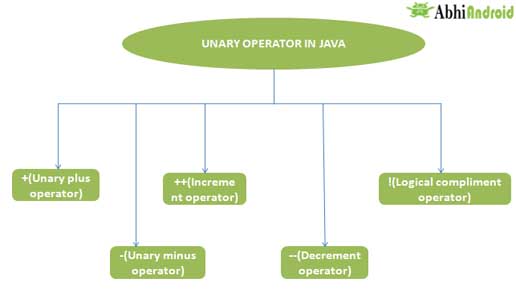
int y = 5;

int z = x%y;

In the above example value % operator gives remainder of x and y division which is then assigned to new variable z. So the remainder value 2 is now stored in z variable.

**UNARY OPERATOR IN JAVA:**

Unary Operator are second type of operator in JAVA which is created to work with only one operand performing operations like incrementing or decrementing the operand value by one, inverting a boolean value or negating an expression. There are total 5 different types of unary operator in JAVA namely +(Unary plus operator), -( Unary minus operator), ++( Increment operator),–( Decrement operator) and !( Logical complement operator).



Now we will discuss each unary operator:

**+(Unary plus operator):**

This operator indicated positive value even though numbers are positive by default. Example of unary plus operator:

int x = +2;

In the above example value of x integer is positive 2.

**-( Unary minus operator):**

This operator negates an expression. Example of unary minus operator:

int x = -5;

In the above example the value of x is minus 5.

**++( Increment operator):**

This operator increment a value of operand by 1. Example of increment operator:

int x = 5;

x++;

In the above example the value of x is incremented by 1 using ++ operator.

**–(Decrement Operator):**

This operator decrement the value of operand by 1. Example of decrement operator:

int x = 5;

x--;

In the above example the value of x is decremented by 1 using — operator. The final value of x is 4.

**!(Logical compliment Operator):**

This operator inverts the value of a Boolean. Example of logical compliment operator:

bollean learningJava = true;

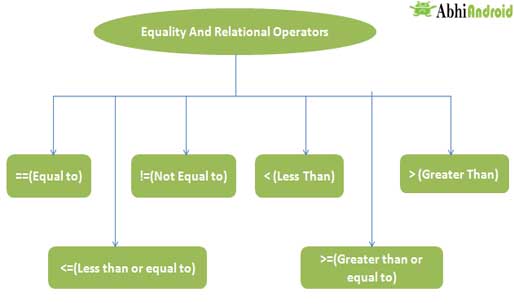
System.out.println(!learningJava);

In the above result output will be false because of using logical compliment operator.

**EQUALITY AND RELATIONAL OPERATOR:**

Equality and relational can be considered as relationship operator because there work is to compare the first operand to the second operand testing the validity of the specified relationship between them. If the result is true then the result will be 1 otherwise for false result is 0. This operator are mostly used in conditions.

There are 6 types of equality and relational operator in JAVA namely ==(Equal to), !=(not Equal to), >(greater than), < (less than), >=(Greater than or equal to) and <= (less than or equal to).



**==(Equal to):**

This operator checks the relationship of first operand is equal to second operand. Remember this operator is checking only if first operand is equal to second or not. It is not assigning value of second operand to first.

**Important Note:** Before we share example of equality and relational operator it is recommended to learn basics of if and else. Here we are only sharing basic and full separate guide will be shared in control flow statement topic.

If and else is the most basic and easy to understand control flow statement which is why we are using them here. The function of “if” is to first check the particular test and if it is found to be true then only it allows the code inside it to execute otherwise skips it and moves to the else statement if present or move to code outside the if block. Also remember, if “if” statement is found to be true then it skips all else statement present just after it.

Let us now create a simple equal to program:

int a = 5;

int b = 7;

if (a == b){

System.out.println("a is equal to b");

}

else{

System.out.println("a is not equal to b");

}

In the above case output will be “a is not equal to b” because a operand equality check with b comes out to be false. So statement/code inside if is not executed and code inside else statement gets executed.

**!=(Not equal to):**

This operator checks the relationship of first operand is not equal to second operand. Let us now create a simple not equal to program:

int a = 5;

int b = 7;

if (a != b){

System.out.println("a is not equal to b");

}

else{

System.out.println("a is equal to b");

}

In the above case output will be “a is not equal to b” because a operand equality check with b comes out to be true. So statement/code inside if gets executed.

**< (Less Than):**

This operator checks first operand less than second operand. Example of less than operator:

int a = 5;

int b = 7;

if (a < b)

{

System.out.println("a is less than b");

} else

{ System.out.println("a is greater than or equal to b");

}

**> (Greater than):**

This operator checks first operand is greater than second operand. Example of Greater than operator:

int a = 5;

int b = 7;

if (a > b){

System.out.println("a is greater than b");

}

else{

System.out.println("a is less than or equal to b");

}

In the above case output will be “a is less than or equal to b” because the result of a operand greater than operator checks with b comes out to be false. So statement/code inside if is skipped and else statement gets executed.

**<= (Lesser than or equal to):**

This operator checks first operand is lesser than or equal to the second operand. Example of lesser than or equal to operator:

int a = 5;

int b = 5;

if (a <= b)

{

System.out.println("a is either equal or lesser than b");

} else{

System.out.println("a is greater than b");

}

In the above case output will be “a is either equal or lesser than b” because the result of a operand lesser than or equal to operator checks with b comes out to be true. So statement/code inside if is executed.

**>= (Greater than or equal to):**

This operator checks whether first operand is greater than or equal to the second operand. Example of greater than or equal to operator:

int a = 7;

int b = 5;

if (a >= b){

System.out.println("a is either equal or greater than b");

}

else{

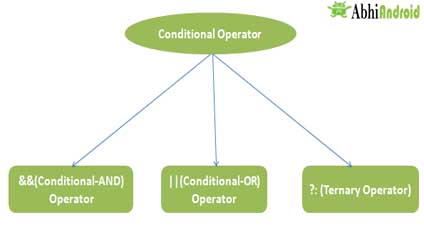
System.out.println("a is lesser than b");

}

In the above case output will be “a is either equal or greater than b” because the result of a operand greater than or equal to operator checks with b comes out to be true. So statement/code inside if is executed.

**CONDITIONAL OPERATOR:**

Conditional Operator gives result based on evaluating two or more boolean expressions. There are three types of conditional operator in JAVA: && (Conditional-AND), || (Conditional-OR) and ?: (Ternary). We will discuss all one by one:



**&& (Conditional-AND) Operator:**

Conditional AND operator takes two operands and both the operand has to be boolean expression (Expression whose output is either true or false). The && operator return true only if both the operands are true otherwise it return false. In other words, if any one or both boolean expression is false, this operator will return false.

Example of Conditional AND operator:

Lets suppose we want to create a program which only accept people of age between 18 and 28:

int xAge = 25;

if(xAge>=18 && xAge <= 28 ){

System.out.println("You are accepted");

}

else{

System.out.println("Not accepted");

}

In the above example both the boolean expression are true so the logical AND(&&) operator will return true and thus the output will be You are accepted.

**|| (Conditional-OR) Operator:**

This operator also accept two operands and both the operands are boolean expression. The operator returns false if both the operand are false otherwise it return true. In other words, if any one boolean expression is true out of two, this operator will return true.

Example of Conditional-OR Operator:

Lets create a simple program where two types of peoples are accepted. Either whose age is less than 18 or greater than 28.

int xAge = 17;

if(xAge < 18 || xAge > 28) {

System.out.println("You are accepted");

}else

{

System.out.println("You are not accepted");

}

In the above example, since first boolean expression is true and we know if any boolean expression is true the logical OR operator will return true. Thus, output will be You are accepted.

**?: (Ternary Operator):**

Ternary Operator takes three operand in which first one is boolean expression, second and third operands are values. It is called ternary because it accepts three operands.

In Ternary Operator, if the first operand which is boolean expression is true, then the value of second operand is return otherwise value of third operand is return. In other words, if boolean expression is false, the value of third operand will be return otherwise second operand will return.

**Important Note:** Ternary operator is a shorter sytax of if then else.

Example of ternary operator:  
Lets create a program to know whether the number is even:

int x = 9;

String output = (x%2 ==0): "Even" : "Not even";

System.out.println(output);

If we divide 9 by 2 the remainder is 1 which is not equal to 0. So in the above example the first operand which is boolean expression is false. Thus the value of third operand is return and out is Not even.

**TYPE COMPARISON OPERATOR:**

**instanceof:**

instanceof is also called type comparison operator. It compares an object to a specified type.

**Important Note:** Before you understand it, you must know What is Object, and Class in JAVA. Below we sharing only the basic of each topic:

Object is any entity that has state & behavior and class is a group of objects that has common properties. In other words Class is a blueprint or template from which objects are created. So objects are result(instance) of Class.

instanceof operator is used to check whether the object is of particular class, subclass or interface type. It returns value true or false. If an object after comparison is find to be a specified type then it returns true otherwise false.

instanceof work on IS~A principle. IS-A is a way of saying, “this object is a type of that class.”

**Syntax of instanceof Operator:**

object instanceof type

It return true if object is of specified type as assigned on the right side. Example of instanceof Operator:

class Abhiandroid{

public static void main(String args[]){

Abhiandroid uidesign =new Abhiandroid();

System.out.println(uidesign instanceof Abhiandroid);

}

}

In the above example the output will be true since object uidesign is of Abhiandroid class type.

**Important Note:** If you found this instanceof operator confusing, we recommend you first understand the concept of class, object, OOPs concept, create few complex programs and comeback to this topic again. Then you will understand it more deeply and clearly.

**BITWISE AND BIT SHIFT OPERATOR:**

**Bitwise Operator**

**Important Note:** Bitwise and Bit Shift Operator are hardly used in JAVA and very rarely used in Android. So you can ignore this operator.

As from the name Bitwise sound these operator performs the operation on bit value. And also you must have heard bit is smallest unit of memory. This is represented by either 0 or 1 which means you have only one option to mark your answer. Also you can understand this concept by an electrical switch which is connected with a bulb. There can only be two conditions, one is bulb can be ON and second is it can be OFF. This is exact representation of a single bit. The combination of 4 bits is called as nibble. **Bitwise common Operator are: Bitwise AND (&), Bitwise OR(|) and Bitwise exclusive OR(^):**

For complete information read [Bitwise Tutorial with example](https://abhiandroid.com/java/bitwise-operator-java-example.html).

**Shift: LeftShift And Rightshift Operator**

Shift Operator is of two types Bitwise Left Shift And Right Shift operator. In the Bitwise shift operator, the result comes applying these operator on the bit values.

For complete information read [Shift: LeftShift And RightShift Tutorial with example](https://abhiandroid.com/java/shift-leftshift-and-rightshift-operator-in-java.html).

#### ****Operator Precedence:****

So now you have learned all types of operator in JAVA. You now also know operator are special symbols in JAVA which perform specific operation to give specific result on operand which can be one, two or three. You now also know how to use this operand.

But before you start using operator in JAVA you must also know one more concept called Operator Precedence. This operator precedence has all operator listed according to precedence order. Operator with higher precedence are executed first relative to the lower precedence operator.

For example, suppose we are using two operator multiplication and addition, then according to operator precedence multiply will be evaluated first and then addition.

int x = 5\*8+5;

Output will be 45 and not 65 because multiply operator(\*) will be executed first relative to addition(+) operator.

**Below is Operator Precedence Table:**

|  |  |
| --- | --- |
| Operator Precedence | |
| **Operator** | **Precedence** |
| postfix | expr++, expr– |
| unary | ++expr, –expr, +expr, -expr, ~, ! |
| multiplicative | \*, /, % |
| additive | +, – |
| shift | <<, >>, >>> |
| relational | <, >, <=, >=, instanceof |
| equality | ==, != |
| bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| logical AND | && |
| logical OR | || |
| ternary | ? : |
| assignment | =, +=, -=, \*=, /=, %=, &=, ^=, |=, <<=, >>=, >>>= |

Operator at the top are have higher precedence and Operator on the same line have equal precedence. When operator of equal precedence appear in the same expression, a rule must govern which is evaluated first. All binary operator except for the assignment operator are evaluated from left to right; assignment operator are evaluated right to left.

Keywords In JAVA With Examples

Keywords in JAVA are predefined list of keywords which has a specific meaning and cannot be used in the Java programming language as an identifier, such as the name of a variable, method, class, function, or label. JAVA has total of 50 such keywords.

**Table of Contents**[[hide](https://abhiandroid.com/java/keywords-in-java-with-examples)]

* [1 List of Keywords In JAVA:](https://abhiandroid.com/java/keywords-in-java-with-examples" \l "List_of_Keywords_In_JAVA)
* [2 Keywords Explanation With Examples:](https://abhiandroid.com/java/keywords-in-java-with-examples" \l "Keywords_Explanation_With_Examples)

**List of Keywords In JAVA:**

Abstract, Continue, For, New, switch, assert, default, goto, package, synchronized, boolean, do, if, private, this, break, double, implements, protected, throw, byte, else, import, public, throws, case, enum\*\*\*\*, instanceof, return, transient, catch extends, int, short, try, char, final, interface, static, void, class, finally, long, strictfp\*\*, volatile, const\*, float, native, super and while

**Keywords Explanation With Examples:**

As we said predefined keywords cannot be used in the JAVA language as an identifier. Lets try to create a [variable](https://abhiandroid.com/java/variables-with-examples) name while:

int while;

You will get Sytax error “while”, invalid VariableDeclarator. So above keywords cannot be used as an identifier and that is also the reason why they are also called as **reserved word or reserved identifier**.

Class And Object In JAVA With Examples – Tutorial

You might have already heard many times JAVA is an Object Oriented Programming which simply means coding in JAVA constantly involve classes and objects. In other words coding in JAVA is not possible without object and class. Even the smallest Hello world program requires declaration of class and method work on object. So let’s understand these two concepts which are really very important in JAVA.

**Table of Contents**[[hide](https://abhiandroid.com/java/class-objects)]

* [1 Definition of Class and Object:](https://abhiandroid.com/java/class-objects" \l "Definition_of_Class_and_Object)
* [2 Real Life Example Of Class And Objects:](https://abhiandroid.com/java/class-objects" \l "Real_Life_Example_Of_Class_And_Objects)
* [3 Another Program With Explanation To illustrate The Concept Of Class And Object:](https://abhiandroid.com/java/class-objects" \l "Another_Program_With_Explanation_To_illustrate_The_Concept_Of_Class_And_Object)
* [4 Important Points about class and object:](https://abhiandroid.com/java/class-objects" \l "Important_Points_about_class_and_object)

**Definition of Class and Object:**

**Class:** The concept of class comes into role when we see certain type of objects or things around us and the common idea or a blueprint behind this type of objects is called Class. In other words class is a properties behind each of the objects or things possess.

**For example:**Consider you have iPhone, Samsung and Sony devices and you want to represent them in JAVA. To do that you first need to find out what can be the blueprint behind these devices. And here blueprint can be a Mobile because they all are a type of Mobile. So Mobile is a class which can represent iPhone, Samsung and Sony devices here.

**How To Declare Class in JAVA:**

class Mobile{

/\*\*ToDo Code Here\*/

}

**Important Note:** Class is a representation of similar types of objects or it is a implementation of encapsulation.

* In terms of java, a class is type declaration means when you want to define a specific type of data for special use then it can be easily obtained with the help of class.
* Basically, class is blueprint for specific type. In the human language across the world, if you count, there are lots of classes e.g. car, bank, bird, student, employee etc. are very simple example to understand.
* Class is created with the word class when you want to define it.  
  Ex: class Mobile{}
* Class itself consists of various methods and variable.
* To call upon class objects of other classes there must be main method with static keyword. Because with the static word method will be called even if you do not create its objects.
* Classes are provided with special access modifiers that are **default, public, private**and **protected**.

**Object:** Object is an instance of class. Understanding the concept of object is lot easier when considering real life examples around us because the concept is actually based on real life objects. So just look around yourself and you will find yourself surrounded with lots of objects which has a certain characteristics and behaviors.

**For example:** Your mobile, it’s an example of object which has a lots of characteristics like color, RAM, camera etc and behaviors like calling, messaging etc.

**How To Create Object:**

Suppose Mobile is the class for which we want to create object name abhi. Below is the code:

//Object abhi is declared

Mobile abhi;

//Object is created using new keyword

abhi = new Mobile();

Alternatively you can create object in one line:

Mobile abhi = new Mobile();

* Object can be defined as state, behavior of class.
* Objects are the instance variable of class.
* Objects are stored through references.
* Objects are created with new keyword in java. Ex: Mobile abhi = new Mobile();
* Every object has its own memory.
* Objects of class of same type can interact with each other means you can pass your message from one object to other.

**Important Note:** Objects are run time entity. They are always getting a memory at run time.

**Important Note:** Object are created from a class and methods or actions are performed on object.

**Real Life Example Of Class And Objects:**

Here in this example we will display the details of Mobile which three person Abhishek, Rahul and Ravi own. To do that in JAVA first we will create a class Mobile, declare fields for mobile specs (i.e brand, color, camera) and initialized constructor.

Then in main method we will three object for Mobile class. Each object will have the specification details of Mobile which he owns. And finally using System.out.println() method we display the details:

//Class Mobile

class Mobile{

    String brand, color;

    int camera;

//Constructor initialized

    public Mobile(String brand, String color, int camera){

        this.brand = brand;

        this.color = color;

        this.camera = camera;

    }

    public static void main(String args[]){

//Object created

        Mobile abhishek = new Mobile("iPhone","Gold",8);

        Mobile rahul = new Mobile("Samsung","White",13);

        Mobile ravi = new Mobile("Nexus","Black",8);

//Smartphone details displayed for each user

        System.out.println("Abhishek own " + abhishek.brand +" "+ abhishek.color + " color smartphone having "+ abhishek.camera+ "MP"+ " camera");

        System.out.println("Rahul own " + rahul.brand +" "+ rahul.color + " color smartphone having "+ rahul.camera+ "MP"+ " camera");

        System.out.println("Ravi own " + ravi.brand +" "+ ravi.color + " color smartphone having "+ ravi.camera+ "MP"+ " camera");

    }

}

**Output:**

Abhishek own iPhone Gold color smartphone having 8MP camera

Rahul own Samsung White color smartphone having 13MP camera

Ravi own Nexus Black color smartphone having 8MP camera

**Another Program With Explanation To illustrate The Concept Of Class And Object:**

Here in this program class **ClassInJava** is taken to create data type of student who is having Name, Roll number, Fathers’ Name, Contact Number, Address as fields. As from the name it is clear that these are of different primitive data type and all these are taken together in a class.

Here declaration of **constructor**of the class is used which initialize the object with predefined values which are passed at the creation of object.

public class ClassInJava {

    String name;

    int rollno;

    String fathername;

    String contactno;

    String address;

    //Constructor

    ClassInJava(String name,int rollno,String fathername,String contactno,String address){

        this.name=name;

        this.rollno=rollno;

        this.fathername=fathername;

        this.contactno=contactno;

        this.address=address;

    }

}

And next class is **ObjectOfClass**which is used to create the object of first class. So the main method with the **static** written in ObjectOfClass. And this is also good prevention to declare first letter of each word of class name as capital.

public class ObjectOfClass {

    /\*\*Simple illustration of how to create an object of given class and how it works

     \*/

    public static void main(String[] args) {

        //Object of class ClassInJava

        ClassInJava object=new ClassInJava("Mr. Abhishek",123,"Mr. Sulekh", "+1-8745733445","#321, South Street, No-3, Ontario");

        System.out.println("Student Name is: " + object.name);

        System.out.println("Roll Number is: " + object.rollno);

        System.out.println("Fathers' Name is: "+ object.fathername);

        System.out.println("Contact Number is: "+ object.contactno);

        System.out.print("Student Address is: "+ object.address);

    }

}

**Output:**

Student Name is: Mr. Abhishek

Roll Number is: 123

Fathers' Name is: Mr. Sulekh

Contact Number is: +1-8745733445

Student Address is: #321, South Street, No-3, Ontario

**Important Points about class and object:**

* The word “Class” came from simula language.
* Class is blueprint for an entity.
* In class there are variables and methods.
* The access modifiers like public, private and protected used in different situation.
* Objects represent the state and behavior of class.
* Object is just a memory area or a buffer in heap area in which all the instance data members are getting a memory.
* In Java “new” operator is only used for allocating a memory for an object only.
* Memory associated with object is automatically collected by garbage collector.
* Class with main method having static keyword is mandatory to call upon the object of other classes.
* All the objects are sharing a same memory location for each static data members.

**History about word “Class”:**Aristotle was initially consider deeply the concept of type. **Simula** is language used for simulation of “bank teller problem”. There were lots of object like customer account, bank detail, employee detail and many more. So all these objects were put together in “classes of objects”. And hence the word class was invented for programming languages**.**That was main origin of word class.

Method In Java With Example

Methods are truly the heart and soul of the java programs. A method is a self contained block of code that performs a specific task. They provide a way of defining the behavior of an object i.e. what the object does. Methods break up large and complex calculations in program that might involve many lines of code into more manageable chunks. All the execution that takes place in any application is within a method.

**Table of Contents**[[hide](https://abhiandroid.com/java/method)]

* [1 Syntax of Method:](https://abhiandroid.com/java/method" \l "Syntax_of_Method)
* [2 Parts of Method Definition](https://abhiandroid.com/java/method" \l "Parts_of_Method_Definition)
* [3 Program Example of Method](https://abhiandroid.com/java/method" \l "Program_Example_of_Method)
* [4 Returning a Value From Method](https://abhiandroid.com/java/method" \l "Returning_a_Value_From_Method)

**Syntax of Method:**

Following is the syntax of method which consists of method header and method body:

returnType methodName(Parameter list)  //Method Header

{

//declarations and statements

//Method Body

}

**Parts of Method Definition**

Method definition consists of two parts:

* Method header
* Method body

**Method Header:** A method header consists of method’s return type followed by the method name and optional parameter list enclosed in the parenthesis.

* The **returnType**in a method header specifies the type of value if any, that the method returns. If the method does not return a value then the return type must be void. For example: return type may be void, int etc.
* The **methodName**in the method header specifies the name of the method. It follows the same rules and conventions as that of identifiers. For example: method name may be area, display, show etc.
* The **parameter list** in the method header must be enclosed in parenthesis. A parameter is a variable that temporarily stores data values known as arguments that are being passed to the method whenever it is called. If no arguments are being passed to the method then the parenthesis remains empty. The parameter list takes the following form: (datatype varName1, datatype varName2, …….)
* **Examples**of method header is:

int area(int l, int b)

void show();

**Important Note:**The parameters specified in the method header are also known as formal parameters and the arguments that are passed to the method are also known as actual parameters.

**Method Body:**The method’s body enclosed in a pair of curly braces consists of local variables and constant declarations for use within the method and sequence of executable statements that takes some kind of action when the method is invoked.

**Program Example of Method**

Let us take an example to show the method definition in the class.

**Step 1:**First we create a class **MethodDemo** in which we define methods:

class MethodDemo

{

int length,breadth;

void getData(int l,int b) //method definition

{

length=l;

breadth=b;

}

int area() //method definition

{

int rectArea=length\*breadth;

return rectArea;

}

}

**Step 2:**Second we create a class **Rectangle** in which we call the methods of above class:

class Rectangle

{

public static void main(String[] args)

{

MethodDemo obj=new MethodDemo();

obj.getData(6,9);

int result=obj.area();

System.out.println("Area of rectangle is:"+result);

}

}

**Output:**

Area of rectangle is: 54

**Returning a Value From Method**

To return a value from a method, the return statement is used. Its **syntax** is as follows:

return(expr);

Here expr is a variable, constant value or an expression.

On execution of the return statement, the program control immediately returns to the point where the method was called. If the expression is the part of the return statement then the value of that expression is returned as the value of the method invocation.

**For Example:**Let us take a small segment from the program example of method in which we have used return statement:

int area()

{

int rectArea=length\*breadth;

return rectArea;

}

Here this method area calculates the area of rectangle whose length and breadth is given and returns the area as an integer.

Since we know that **void** does not return any value then you can omit the return statement or just use the keyword **return**by itself to the end of the execution of method like this:

return;

**Important Note:** There can be more than one return statement in a method. **For example**:

int maximumVal(int a, int b)

{

if(a>b)

{

return a;

}

else

return b;

}

If Then Else In JAVA With Examples – Complete Tutorials

**Table of Contents**[[hide](https://abhiandroid.com/java/if-then-else-in-java)]

* [1 What Is If Then](https://abhiandroid.com/java/if-then-else-in-java" \l "What_Is_If_Then)
* [2 If Then Syntax:](https://abhiandroid.com/java/if-then-else-in-java" \l "If_Then_Syntax)
* [3 If Then Else:](https://abhiandroid.com/java/if-then-else-in-java" \l "If_Then_Else)
* [4 Else If:](https://abhiandroid.com/java/if-then-else-in-java" \l "Else_If)
* [5 Ternary Operator:](https://abhiandroid.com/java/if-then-else-in-java" \l "Ternary_Operator)

**What Is If Then**

If then allow us to control the flow of program based on conditions, whether to execute a code or not. In simple word it gives the freedom to take decisions on what code to execute or not. The biggest advantage of if then is that it makes the program to take action based on user input.

**If Then Syntax:**

If then execute a code of block or single statement if a Boolean [expression](https://abhiandroid.com/java/expression-statement-code-blocks-in-java) inside braces is true. The basic syntax of if then is as follows:

if(Boolean expression is true)

Execute this single statement;

Boolean expression is enclosed in parenthesis and single statement is terminated with semicolon.

Or

if(Boolean expression is true){

Execute all statement inside code block;

}

If you are using code block then each statement inside block should also be terminated with semicolon.

**Example of If then:**

Lets create a simple program when water evaporate if temperature is 100 degree Celsius:

int tempWater = 100;

if(tempWater==100)

System.out.println(“Water start evaporating”);

In the above example, since if Boolean expression is true, so single statement just below will be executed. Thus output will be Water start evaporating.

**Lets see same example with different input:**

int tempWater = 50;

if(tempWater==100)

System.out.println("Water start evaporating");

System.out.println("This Will Be Executed");

In the above example, since the if Boolean expression is false so single [statement](https://abhiandroid.com/java/expression-statement-code-blocks-in-java) just below if won’t be executed. But the next statement just after it will be executed. Thus the output will be:

This Will Be Executed

**Important Note:** It is important to understand without using code block only single statement after if is executed, if Boolean expression is evaluated to be true. Otherwise it will jump to the next statement.

**Lets see same example with block code:**

int tempWater = 100;

if(tempWater==100){

System.out.println("Water start evaporating");

System.out.println("This will also be executed");

}

In the above example, since if Boolean expression is true so both statement inside code block will be executed. So the output will be:

Water start evaporating

This will also be executed

**Important Note:** Using code block is meant to use when you want to execute more than single statement. But we recommend to use code block even if you are using single statement because it comes in good programming practice.

**If Then Else:**

What if you want to execute some code if the if condition evaluates to false, that’s when you need if then else in JAVA. The else statement says to execute the single statement or code of block if the if statement evaluates to false.

**Example of if then else:**

Lets a create a simple program to test whether x number is even or not:

int x = 19;

if(x%2==0){

System.out.println("x is even number");

}

Else{

System.out.println("x is not an even number");

}

In the above example, since the if boolean expression evaluates to false, so the else statement will be executed. Thus the output will be:

x is not an even number

**Else If:**

Sometime we have more than two conditions to evaluate, for such situations we use “else if” statement following an if statement and its body. We can use as many as else if ensuring only one code block is executed.

**Lets create a program to find out largest of three number us else if:**

int x = 10;

int y = 20;

int z = 30;

If(x>=y && x>=z){

System.out.println("x is the greatest number");

else if(y>=x && y>=z){

System.out.println("y is the greatest number");

}

else{

System.out.println("z is the greatest number");

}

In the above example, first if expression will be evaluated to false. After that else if expression will also be evaluated to false. And the third else statement will be executed.

So the output will be:

z is the greatest number

**Ternary Operator:**

This topic is already discussed in conditional [operators](https://abhiandroid.com/java/operators-in-java) but we want to repeat it once again as now it will be easier to understand because it is a shorter syntax of if then else.

Ternary Operator takes three operand in which first one is boolean expression, second and third operands are values. It is called ternary because it accepts three operands.

In Ternary Operator, if the first operand which is boolean expression is true, then the value of second operand is return otherwise value of third operand is return. In other words, if boolean expression is false, the value of third operand will be return otherwise second operand will return.

**Example of Ternary Operator:**

int x = 10;

(x%2 ==0): "Even" : "Not even";

In the above example since the Boolean expression evaluate to true so the value of second operand will be return. So the output will be:

Even

For Loop In JAVA With Examples – Complete Tutorial

**Table of Contents**[[hide](https://abhiandroid.com/java/for-loop-in-java)]

* [1 What Is For Loop:](https://abhiandroid.com/java/for-loop-in-java" \l "What_Is_For_Loop)
* [2 For Loop Syntax:](https://abhiandroid.com/java/for-loop-in-java" \l "For_Loop_Syntax)
* [3 Example of For Loop:](https://abhiandroid.com/java/for-loop-in-java" \l "Example_of_For_Loop)
* [4 Importance of For Loop:](https://abhiandroid.com/java/for-loop-in-java" \l "Importance_of_For_Loop)

**What Is For Loop:**

In JAVA For statement is the most commonly used lopping statement which iterate over a range of numbers. It is different from if then else in a way that it forces the program to go back up again repeatedly until termination [expression](https://abhiandroid.com/java/expression-statement-code-blocks-in-java) evaluate to false. For loop is another way to control flow of program.

**For Loop Syntax:**

The most basic syntax of For loop is:

for (initialization-expression; termination-expression;increment-or-decrement-expression) {

statements-or-code-to-be-executed-here;

}

**initialization-expression –** The initialization expression is executed once before the loop begins. It initializes the loop.

**termination-expression –** This is executed each time to check whether looping should terminate or continue. The looping continues to execute the code until it evaluates to true and terminate when evaluate to false.

**increment-or-decrement-expression –** It is executed after each iteration which increment or decrement the value of initialized variable.

**statements-or-code-to-be-executed-here –** Here we put the code or statement which we want to be iterated until for looping termination expression evaluate to false.

**Example of For Loop:**

Lets create a program to find the sum of 1 to 100 number using for Loop:

public class ForExample {

public static void main(String[] args) {

int i;

int total = 0;

for(i=1;i<=100;i++){

total = total + i;

}

System.out.println("Sum of Value from 1 to 100 number is: " + total);

}

}

Output is:

Sum of Value from 1 to 100 number is: 5050

**Importance of For Loop:**

As we know in JAVA, code is executed from top to bottom executing every line of code unless we use control flow statement. In our previous section, we already learned one way is to use [if then else](https://abhiandroid.com/java/if-then-else-in-java) which tells the compiler to skip executing code if if condition is evaluated to false.

Another way to tell JAVA not to execute the code is using Loopings. It is different from if then else in a way that it forces the program to go back up again repeatedly until termination expression evaluate to false.

For example we need to create a program to find factorial of 10. You can do something like:

int x = 10\*9\*8\*7\*6\*5\*4\*3\*2\*1;

You can do it easily in this case but what if we need to find factorial of 1000 or What if we need to add 1 to 1000 number or even more. Here we need for loop to iterate a particular code for a range of values.

While And Do While In JAVA With Examples – Complete Tutorials

The While is a type of loop which evaluate the condition first. If condition evaluate to true the code inside the block{} will be executed and if it evaluate to false it will jump outside the while loop. The While in JAVA has two parts, first defining a Boolean expression inside parenthesis which will be tested and second is a statement or code which will be executed continually until Boolean expression evaluate to be true.

**Table of Contents**[[hide](https://abhiandroid.com/java/while-do-while-in-java)]

* [1 Syntax Of While:](https://abhiandroid.com/java/while-do-while-in-java" \l "Syntax_Of_While)
* [2 Examples Of While In JAVA:](https://abhiandroid.com/java/while-do-while-in-java" \l "Examples_Of_While_In_JAVA)
* [3 Do While In JAVA:](https://abhiandroid.com/java/while-do-while-in-java" \l "Do_While_In_JAVA)
* [4 Examples Of Do While In JAVA:](https://abhiandroid.com/java/while-do-while-in-java" \l "Examples_Of_Do_While_In_JAVA)

**Syntax Of While:**

The syntax of While in JAVA is:

initialize-value;

while(Boolean-Expression){

statement-or-code; //This will be executed continually until Boolean expression evaluate to true

increment-decrement-value;

}

**initialize-value –** First initialize the value first before starting while loop. **Boolean-Expression –** Boolean expression will be evaluated. If it comes out to be true then code inside it will executed otherwise compiler jumps outside the loop. **statement-or-code –** The code that you want to execute after Boolean expression evaluate to true. **increment-decrement-value –** increment or decrement the value of initialized variable

**Examples Of While In JAVA:**

Lets create a simple program to print 1 to 10 number using While loop:

public class WhileExample {

public static void main(String[] args) {

int i=1;

while(i<=10){

System.out.println(i);

i++;

}

}

}

Output:

1

2

3

4

5

6

7

8

9

10

**Do While In JAVA:**

The Do While loop will first execute the code inside do{} and then evaluate the while Boolean expression. The statements will continue to execute until Boolean expression evaluate to false. In do while, we first put statement or code that we want to execute inside the do block and then we put the Boolean expression inside While parenthesis just outside do block.

**Important Note:** The major difference between While and do While is that statement will be executed atleast once in do while.

**Examples Of Do While In JAVA:**

We can create the same program to print 1 to 10 number with do while in JAVA:

public class DoWhileExample {

public static void main(String[] args) {

int i=1;

do{

System.out.println(i);

i++;

} while(i<=10);

}

}

Output:

1

2

3

4

5

6

7

8

9

10

Array In JAVA With Examples

Array can be defined as a contiguous memory locations used to store the homogeneous data types. In simple words, it is a [variable](https://abhiandroid.com/java/variables-with-examples) that can store multiple values of single data type.

* The integer array can be declared as int[] intArray; (recommended by JAVA) or int intArray[]; (not recommended by JAVA). Here intArray is the name of Array variable.
* The length/size of array is fixed and defined when array is created. For example, intArray = new int[5]; fix the length of Array with variable name intArray up to 5 values which means it can store 5 values of integer data type.
* int[] intArray; // Array Declared
* intArray = new int[5]; // Array length is established gets memory for 5 integers which is fix
* intArray[0] = 10; // Value stored at index 0 position
* intArray[1] = 20; // Value stored at index 1 position

//So on... we can store upto 5 values and index 4 position

* Alternatively we can also declare array using shorter syntax: int[] intArray = {10,20,30,40,50}; . In this case the total number of values is the size of array and also values are directly stored in it starting with index 0.

**Table of Contents**[[hide](https://abhiandroid.com/java/arrays)]

* [1 Array Explanation: Index And Element Value:](https://abhiandroid.com/java/arrays" \l "Array_Explanation_Index_And_Element_Value)
* [2 Array Declaration In JAVA](https://abhiandroid.com/java/arrays" \l "Array_Declaration_In_JAVA)
* [3 Array Examples, Program And Code:](https://abhiandroid.com/java/arrays" \l "Array_Examples_Program_And_Code)
* [4 Multidimensional Array In JAVA:](https://abhiandroid.com/java/arrays" \l "Multidimensional_Array_In_JAVA)
* [5 Array Important Points, Summary And Quick Revision:](https://abhiandroid.com/java/arrays" \l "Array_Important_Points_Summary_And_Quick_Revision)

**Array Explanation: Index And Element Value:**

**int[] intArray = {10,20,30,40,50};**

|  |  |
| --- | --- |
| **Index Value** | **Element Value** |
| 0 | 10 |
| 1 | 20 |
| 2 | 30 |
| 3 | 40 |
| 4 | 50 |

Let’s now understand array concept from the above table which shows an array with 5 integer values stored in it. Two things are very clear at seeing on first sight from the table i.e. *array is having an index value which starts from 0 and other is element value*. Both index and element value have their own meaning. **Index value is simple to take the track of your array. For example, we can track value 20 by using intArray[1];. Element value is the value stored in array.**

**Important Note 1:** Index value starts with zero and move on in increasing order. It has upper bound and lower bound. Upper bound is always one less than the size of given array.

**Important Note 2:** The first element of array is accessed at index 0 i.e. intArray[0] in the above table. And the last element can be accessed by one less index value of total i.e. intArray[4] in the above table.

**Array Declaration In JAVA**

To assign some value in the memory there are two thing mandatory in program. First is its declaration and then its initialization. So same principle is applied on Array also. To declare it we can simply write as

int[] onedimensionalarray= new int[5];

//OR

int onedimendionalarray[]=new int[5];

Both the ways are right, but first one is preferred as recommended by JAVA. This is to declare an one dimensional array of integer type.

Now the second thing is to initialize that particular variable. Here it is assigned the values using index of that particular array. E.g.

onedimensionalarray[0]=10; // Particular value is initialized

Here first index of array is initialized with value 10. In similar way you can simply assign the value to the different index values.

**Important Note:**The important thing about the array is to remember its size is fixed after its creation i.e. when memory has been allocated.

***Another way to declare and initialize the given array:***

We can also use shorter index to declare and directly initialize values:

int[] onedimensionalarray= {10,30,20,50,15};

**Similarly we can declare other primitive data types and String array in JAVA.**

byte[] anArrayOfBytes; // Byte Array Declaration

short[] anArrayOfShorts; // Short Array Declaration

long[] anArrayOfLongs; // Long array declaration

float[] anArrayOfFloats; //Float array declaration

double[] anArrayOfDoubles; //Double array declaration

boolean[] anArrayOfBooleans; // Boolean Arrray Declaration

char[] anArrayOfChars; //Character array declaration

String[] anArrayOfStrings; //String Array Declaration

**Array Examples, Program And Code:**

Now lets build some Array programs to understand the Array concept more deeply.

**Example 1:** This is a very simple program in which integer Array is declared, initialized and finally all its value is printed using For loop.

public class ArraysInJAVA {

public static void main(String[] args) {

int[] onedimensionalarray= new int[5]; // Array declared with size

onedimensionalarray[0]=10; // Value initialized

onedimensionalarray[1]=20;

onedimensionalarray[2]=30;

onedimensionalarray[3]=40;

onedimensionalarray[4]=50;

// Values are printed in output

for(int i=0; i<5;i++){

System.out.println("the value in onedimensioanlarray are " +onedimensionalarray[i]);

}

}

}

**Output:**

the value in onedimensioanlarray are 10

the value in onedimensioanlarray are 20

the value in onedimensioanlarray are 30

the value in onedimensioanlarray are 40

the value in onedimensioanlarray are 50

**Example 2:** The second example of Array is of String because lots of JAVA beginners are confused with it though the process is simple and same.

If you consider the case for array of string, in this there is also same principle apply as array of integers. You can access any string from the array using the particular index value of for any string to which you want to access.

Like if we take an array of string of size 4. First of all we must know how to declare array of string. This can be written as:

**String[] onedimensionalarray= new String[4];**

And after doing this the next thing is to assign the different string elements to this array which we can access later on when we have its requirement. To access the elements from this array simply we can use For loop which will follow 4 iterations and in these iterations array elements can be printed very easily.

public class ArrayStringJava {

public static void main(String[] args) {

String[] onedimensionalarray=new String[4]; //String Array Declared

onedimensionalarray[0]="Java "; // Value intialized

onedimensionalarray[1]="Is ";

onedimensionalarray[2]="High Level ";

onedimensionalarray[3]="Language";

//String value printed

for(int i=0; i<onedimensionalarray.length;i++){

System.out.print(onedimensionalarray[i]);

}

}

}

**Output:**

Java Is High Level Language

**Multidimensional Array In JAVA:**

Array is continuous memory locations used to store homogeneous data means a data of similar type and Multi-Dimensional Array is used to store the values in the rows as well as in columns.

**Important Note:** Java does not support multidimensional array. Multidimensional array in java is basically **“array of arrays”.**

Below is multi-dimensional array declaration syntax:

int[][] twoDimensionalArray= new int[2][3];

Multi-dimensional Array initialization:

int[][] twoDArray=new int[2][2];

    /\*value assigned are 1,2,4,5 at [0][0],[0][1],[1][0],[1][1] index values respectively\*/

        twoDArray[0][0]=1;

        twoDArray[0][1]=2;

        twoDArray[1][0]=4;

        twoDArray[1][1]=5;

For complete details read [Multidimensional tutorial with example](https://abhiandroid.com/java/multidimensional-array-example.html)

**Array Important Points, Summary And Quick Revision:**

* Array is data structure which is static in nature. This is different from the type of array used in c/c++/Fortran where its size put no limit on entering the element. In case of java if you put the element beyond its limit then it shows compile time exception of out of index array exception.
* Array is used to store multiple values of same data type in single variable
* The size of array is fixed and defined when it is created
* Array index value start with 0

Inheritance Tutorial With Example In JAVA

Inheritance is a mechanism that allows the class to use the states and behavior of another class. In simple words a class derive field and methods from another class. The derived class in inheritance is called sub class (also called derived class or extended class, or child class) and the class from which it is derived is called super class (also called base class or a parent class).

**Important Note:** A super class can have any number of sub class in inheritance but sub class can only extend one super class. Multiple inheritance is not supported in JAVA.

Inheritance is one of the important concept of object oriented programming. The simple definition says inheritance provides mechanism that allows a class to inherit properties of another class.

**Inheritance Concept:** Inheritance means inherit the properties and behavior of existing object in a new object.

**Way To Achieve Inheritance:**It is achieved by deriving a new class from existing class using extends keyword. Example, Class Child extends Base { }.

**Table of Contents**[[hide](https://abhiandroid.com/java/inheritance)]

* [1 Explanation With Example:](https://abhiandroid.com/java/inheritance" \l "Explanation_With_Example)
* [2 Types Of Inheritance](https://abhiandroid.com/java/inheritance" \l "Types_Of_Inheritance)
* [3 Inheritance Example:](https://abhiandroid.com/java/inheritance" \l "Inheritance_Example)
* [4 Importance Of Inheritance:](https://abhiandroid.com/java/inheritance" \l "Importance_Of_Inheritance)
* [5 Inheritance Important Points To Remember:](https://abhiandroid.com/java/inheritance" \l "Inheritance_Important_Points_To_Remember)
* [6 Inheritance Quick Revision:](https://abhiandroid.com/java/inheritance" \l "Inheritance_Quick_Revision)

**Explanation With Example:**

* In Inheritance by default all data members and member functions of a parent class are available to child class if they are not private
* Inheritance defines **is-a**relationship between a super class(parent) and its sub class(child).
* **extends**keyword is used to show inheritance in java.

For example :

Suppose a class name Base.java

class Base

{

//Code of Base class

}

Another class Child.java use Inheritance to extends properties from Base class.

Class Child extends Base

{

//extends the properties of base class

}

In the above example, Child class will inherit field and methods of Base class.

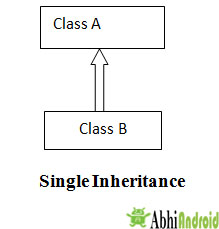
**Types Of Inheritance**

There are 3 types of inheritance:

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance

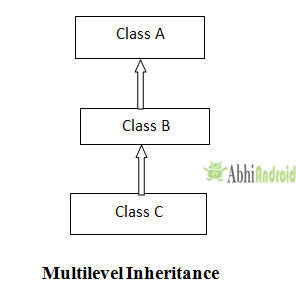
**1. Single Inheritance:**

In Single inheritance one class is derived from one parent class. The below diagram shows single inheritance where Class B inherits from Class A.



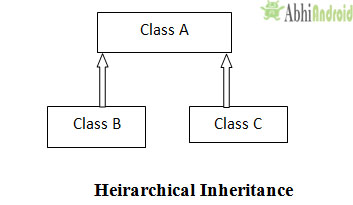
**2. Multilevel Inheritance:**

In multilevel inheritance there is a concept of grand parent class as shown in below diagram class C inherits class B and class B inherits class A.



**3. Hierarchical Inheritance:**

In Hierarchical inheritance more than one sub classes is derived from a single parent class as shown in below diagram class B and C both are derived from single parent class A.

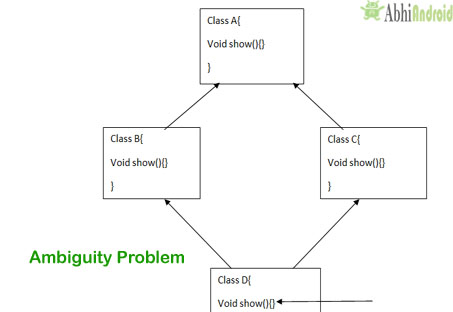


**Important Note:**Java does not support multiple Inheritance .

**Why multiple inheritance is not supported in java:**

**To remove ambiguity**:

Multiple inheritance is not supported in java as it causes ambiguity in few scenarios. The most common scenario is **Diamond problem.**



Consider the above diagram which shows multiple inheritance. In this class D extends both class B & C.Here class B & C inherit the same method of class A. Now the problem comes when class D is extending both class B & C and if class D wants to use same method then which method would be called? That’s why multiple inheritance is not supported in java as to remove ambiguity.

**Inheritance Example:**

Below is the program to show you the use of inheritance in java. For coding this we have used eclipse IDE.

**Example 1:** Let’s inherit some fields and methods in Child class from Base class.

Base class is having 2 fields and 1 method:

class Base

{

int x=50;

int y = 60;

//Addition method return integer value of x+y

public int addition(){

return x+y; //return 110

}

}

Child class inherit addition method and x field from Base class:

public class Child extends Base

{

int z;

public void subtraction(){

//addition method and x filed is inherited from Base Class

z = addition() - x;

System.out.println(z);

}

}

In the Main class we create Child object and calls subtraction method on it.

public class InheritanceAbhiandroid {

public static void main(String[] args) {

Child child = new Child(); //Child object

child.subtraction();//Subtraction method called on child object

}

}

**Output:**

60

**Example 2:**

**Base.java:**

class Base

{

int x=50;

}

**Child.java:**

public class Child extends Base

{

int x=20;

void show()

{

System.out.println(x);

}

}

**InheritanceAbhiandroid.java**

public class InheritanceAbhiandroid {

public static void main(String[] args) {

Child c = new Child();

c.show();

}

}

**OUTPUT:**

20

**Important Note:** The above program prints value of x=20 because priority always goes to local variables. So in show() method of child class,we can access member of parent class i.e base class using **super**keyword. It means if we want to print value of x=50 also from the same above program then by using super keyword.

**Example 3 of Inheritance using super keyword:**

**Base.java:**

class Base

{

int x=50;

}

**Child.java:**

class Child extends Base

{

int x=20;

void show()

{

System.out.println(x);

System.out.println(super.x);

}

}

public class InheritanceAbhiAndroid{

public static void main(String[] args)

{

Child c=new Child();

c.show();

}

}

**OUTPUT:**

20

50

**Importance Of Inheritance:**

* **Reusuability of code**: It is one of the important feature of inheritance. It is a good way to reuse the already existing code rather than creating the same code again and again. This feature not only saves time and money as we are reusing the properties but it also increase reliability of code.
* **Method Overriding**: With the help of inheritance, it is possible to override the methods of base class so that base class method is easily used in derived class.

**Inheritance Important Points To Remember:**

* Whenever a parent class and a child class both are having same data members then this concept is known as **data hiding**.
* Whenever a parent class and a child class both are having ditto same functions then this concept is known as **[method overriding](https://abhiandroid.com/java/method-overriding).**
* Whenever a parent class and a child class both are having same static functions then this concept is known as **function hiding.**
* We cannot print **super**, there is a syntax error. Always data members of parent class is inherited by **super**
* If you make any non static function of a class as **final** then it cannot be overridden by the child class that means to stop method overriding makes a function **final**.

**Inheritance Quick Revision:**

* Inheritance allows the class to use the states and behavior of another class using extends keyword
* Inheritance **is-a**relationship between a Base class and its child class.
* Multiple inheritance is not supported in JAVA.

Abstraction Tutorial With Example In JAVA

In Java, Abstraction is one of the major building block. It is a process of hiding internal working and showing only necessary details. In simple form abstraction means:

* Show Functionality
* Hide Complexity

**Important Note:** Interface is used to achieve 100% Abstraction in JAVA. Here we only discuss Abstraction in full details but we recommend you to learn Abstraction and Interface topic simultaneously as both topics are similar with little bit difference.

**Table of Contents**[[hide](https://abhiandroid.com/java/abstraction)]

* [1 Explanation Of Abstraction:](https://abhiandroid.com/java/abstraction" \l "Explanation_Of_Abstraction)
* [2 Key Learning Points On Abstraction:](https://abhiandroid.com/java/abstraction" \l "Key_Learning_Points_On_Abstraction)
* [3 Syntax Of Abstraction:](https://abhiandroid.com/java/abstraction" \l "Syntax_Of_Abstraction)
* [4 Examples Of Abstraction:](https://abhiandroid.com/java/abstraction" \l "Examples_Of_Abstraction)
* [5 Importance Of Abstraction](https://abhiandroid.com/java/abstraction" \l "Importance_Of_Abstraction)
* [6 Difference Between Abstract Class And Interface:](https://abhiandroid.com/java/abstraction" \l "Difference_Between_Abstract_Class_And_Interface)

**Explanation Of Abstraction:**

The best explanation of Abstraction is using the common example of sending SMS. When a user send SMS, he simply enter the address, type message in the text box and send it. He doesn’t know what is happening behind the scene when this message is getting send to the other user. So here implementation details has been hided from the user and only functionality is presented to user. This is achieved in JAVA either through Abstraction or Interface.

**Key Learning Points On Abstraction:**

Before we share syntax you first need to understand abstract class, abstract methods and why/how to extend Abstract class:

**What Is Abstract Class:**

An abstract class is declared with abstract keyword which may contain methods without body, with body or mixture of both. If a class have at least one method without body then it has to be declared abstract.

**Important Note:** Method without body means method without any functionality or implementation.

In the below example you can see one method calling is without body. So here class has to be declared abstract because it has one method without body.

abstract class Mobile{

abstract void calling();

abstract void messaging()

{

System.out.println("Messaging");

}

}

**What Is Abstract Method:**

The methods without body or methods with only signatures are called abstract methods. The abstract method doesn’t have any implementation details. It is declared using abstract keyword before method name. For example, calling method is an abstract method.

abstract void calling();

**Syntax: of Abstract Method**

abstract void method\_name();

**Important Note:** In order to use abstract method we first need to override that method in subclass (i.e. child class).

**Why And How To Extend Abstract Class:**

As now you already know Abstract class may contains abstract method without any implementation and thus we cannot instantiate abstract class. In case we try instantiating it, the object might be useless because methods inside abstract class may not have any implementation at all. So to avoid this situation, JAVA doesn’t allow us to instantiate abstract class.

We first need to extend abstract class where we will do the implementation of those abstract method and then we instantiate this new class.

**So let’s see how to extend Abstract class:**

**Step 1:** First we define abstract class with abstract method:

abstract class Mobile{

abstract calling(); //Method without body

}

**Step 2:** Now we extends abstract class and implement abstract methods:

class Samsung extends Mobile{

void calling();

{

System.out.println("Start Calling");

}

}

**Pro Note:** An example of abstract class is AbstractMap which is part of collection framework include TreeMap, HashMap and ConcurrentHashMap and share many methods like isEmpty(), put() , get(), containValue() , containKey() etc that AbstractMap defines.

**Syntax Of Abstraction:**

The syntax of abstraction start with abstract keyword before class name. It then contains mixture of abstract and non-abstract methods.

abstract class <ClassName>{

//Mixture of abstract methods(Without body/implementation) and non-abstract methods(with body/implementation)

//If it contains atleast one abstract method then a class has to be declared abstract

}

**Examples Of Abstraction:**

**Example 1:** Lets now understand abstraction concept using real life examples of different sounds created by animals. For example Cat does Meow and Lion Does Roar. We will display different sounds using Abstraction in JAVA.

**Step 1:** First create a new project in Eclipse and create a abstract class Sound. Below is the code of Sound.java

abstract class Sound {

//Non-abstract method i.e. method with implementation

    public void soundmessage(){

        System.out.print("Animal Sound");

    }

//Abstract method i.e. without body/implementation

    abstract void sound();

}

**Step 2:** Now create another class name Cat which extends Sound class. Here we will implement abstract sound() method for Cat. Below is the code of Cat.java

class Cat extends Sound{

void sound(){

soundmessage();

System.out.println(" of Cat: Meow");

}

}

**Step 3:** In the same way we are creating another Lion class which extends Sound class. Here also we will implement the sound() method but for Lion. Below is the code of Lion.java:

class Lion extends Sound {

void sound(){

soundmessage();

System.out.println(" of Lion: Roar");

}

}

**Step 4:** Now create AnimalSound main class. Here first we will create object of Cat & Lion class and then we will call sound() method on these objects. Below is the code of AnimalSound.java

public class AnimalSound {

public static void main(String[] args) {

Cat cat = new Cat();

cat.sound();

Lion lion = new Lion();

lion.sound();

}

}

**Output:**

Now run the program and you will see sounds of Cat and Lion printed.

Animal Sound of Cat: Meow

Animal Sound of Lion: Roar

**Conclusion:** So you can see how useful is Abstraction. We just define mixture of abstract and non-abstract methods in Abstract class and then implement abstract method in child class (i.e. sub class) according to requirement. In the end same method gives different result depending on Objects of which sub-class. Remember we can’t instantiate Abstract class as discussed earlier.

**Example 2:** In second Abstraction example we simply display a text “AbhiAndroid” by using an abstract method of an abstract Base class. In this, Child class extends the Base class and override the display method and then prints the text as shown in below example.

Here in this example we have provided lots of explanation in the code itself as comments.

**Below is the code of Base.java**

abstract class Base //abstract class

{

abstract void display(); //abstract method

}

**Below is the code of Child.java and also main function of class**

class Child extends Base //child class which extends the property of base class

{

void display() // override method of base class

{

System.out.println("AbhiAndroid"); //prints the text “AbhiAndroid” as an output

}

public static void main(String args[]) //main function of class

{

Child c=new Child(); //create an object of child class

c.display(); //call the display method of child class with the help of object

}

}

**OUTPUT:**

AbhiAndroid

**Conclusion:** In the above example, Base is abstract class that contain abstract method display(). Its implementation is provided by the Child class.

**Importance Of Abstraction**

Below is the some major important points of abstraction in java.

* Abstraction helps to reduce the complexity and also improves the maintainability of the system.
* Abstraction gives more power to the object oriented programming languages when used with the concepts like encapsulation and polymorphism.
* Abstraction is used to provide solution to real world problem.

**Difference Between Abstract Class And Interface:**

Abstract class is quite similar to interface like both contain abstract method and also we can’t initiate both of them. But below are the some key differences between  both the topics:

**Abstract class:**

1. Abstract class can extend only one class at a time.
2. Abstract class can have both abstract (method without implementation) and concrete methods (method with implementation).
3. In abstract class ‘abstract’ keyword is used to declare a method as abstract.
4. Abstract class can have protected, public and public abstract methods.
5. Abstract class can have static, final or static final variables with any access specifier.

**Interface:**

1. Interface can extend any number of interfaces at a time.
2. Interface can only have abstract methods, they cannot have concrete methods.
3. In interfaces, “abstract” keyword is optional to declare a method as an abstract because all the methods are abstract by default.
4. Interfaces can have only public abstract methods i.e by default.
5. Interfaces can have only static final variable i.e by default.

Method Overriding Tutorial With Examples In JAVA

Overriding means to extend or to pass over something, especially to overlap the previous described functionality. So Method Overriding means to re-write the previous described method again of Parent Class in Sub class with different functionality.

In Method Overriding, we overrides the method of Super Class in Sub Class. The method overriding is also called Run time Polymorphism or Dynamic Binding.

**Table of Contents**[[hide](https://abhiandroid.com/java/method-overriding)]

* [1 Method Overriding Explanation With Example:](https://abhiandroid.com/java/method-overriding" \l "Method_Overriding_Explanation_With_Example)
* [2 Writing @Override Annotation Is Optional](https://abhiandroid.com/java/method-overriding" \l "Writing_Override_Annotation_Is_Optional)
* [3 Real life example of Method Overriding :](https://abhiandroid.com/java/method-overriding" \l "Real_life_exampleof_Method_Overriding)
* [4 Output:](https://abhiandroid.com/java/method-overriding" \l "Output)
* [5 Advantages of using Method Overriding:](https://abhiandroid.com/java/method-overriding" \l "Advantages_of_using_Method_Overriding)
* [6 Rules of Method Overriding :](https://abhiandroid.com/java/method-overriding" \l "Rules_of_Method_Overriding)
* [7 Difference between Method Overloading and Method Overriding:](https://abhiandroid.com/java/method-overriding" \l "Difference_between_Method_Overloading_and_Method_Overriding)
* [8 Important Points:](https://abhiandroid.com/java/method-overriding" \l "Important_Points)
* [9 Quick Revision points:](https://abhiandroid.com/java/method-overriding" \l "Quick_Revision_points)

**Method Overriding Explanation With Example:**

Before discussing **Method Overriding**, let us revise our concept of**[Inheritance](https://abhiandroid.com/java/inheritance)**. Inheritance is a phenomena in which one class extends the properties( i.e. methods and fields) of another class. It makes our programming much easier as previously defined code is reused, and we can add our own functionality.

The class which extends the properties of another class is called Sub-Class or Child-Class or Derived-Class. And the class whose methods and fields or any other property is inherited is called Super-Class or Base-Class or Parent-Class.

**Why Method Overriding?** Now let us understand the problem that we face without method overriding.

In the following program we have Animal and Horse classes. In **Animal Class** we have defined a method which **prints “Animal is running”.** In **Horse class** we have used the already defined **method run()** of Animal Class as shown:

Below is code of Animal class:

class Animal {

void run(){

System.out.println("Animal is running");

}

}

Below is the code of Horse class which extend Animal class

public class Horse extends Animal

{

public static void main(String args[]){

Horse horse = new Horse(); //Horse class object is created

horse.run();

}

}

**Output:**

Animal is running

***Problem & Solution:****But the problem here is if we have to give specific implementation to Horse Class object, that “Horse is running” then how can we do that? The answer is Method overriding! So to give specific implementation to parent class method in sub-class we need Method Overriding.*

**So let’s come to Method Overriding:**

* In this process when a method defined in super class the same method is defined again in sub class
* In sub class new functionality is defined for that method which overlap the functionality defined in Parent Class. So here method is overridden.
* In other words we can say defining and giving a new functionality to the same method in sub class which is already present in parent class is called Method Overriding.

**Lets discuss the same example again using method overriding:**

class Animal {

void run()

{

System.out.println("Animal is running");

}

}

Below is the Horse class code which override run method of Animal class after extending it:

public class Horse extends Animal

{

//run() Method is overridden in Sub-Class

void run(){

System.out.println("Horse is running");

}

public static void main(String args[]){

Horse horse = new Horse();

horse.run();

}

}

**Output:**

Horse is running

Here we see method run() is defined again in sub class Horse or in other words method run() is overlapped in the child class, giving it a different functionality. So now we have given specific implementation to the child class run() method.

**Important Note:**Method Overriding is also called as Run time Polymorphism or Dynamic Binding. This is due to the fact that the compiler doesn’t necessarily know what type of object is being passed at compile-time.

**Writing @Override Annotation Is Optional**

When you override any method, it is optional but recommended in JAVA to write @Override just above the method which you are overriding. This helps JAVA compiler to know that we want to override a method here which is already present in Parent class. But in-case if it is not present the compiler will give error letting us know that there is no method like that in Parent class.

**Below is the Syntax of Method Overriding using @Override**

class Parent{

method(){

//Add Functionality here

}

}

class Subclass extends Parent{

@Override

method(){

//New Functionality here for method

}

}

**Real life example of Method Overriding :**

Lets take a real life situation where we will need Method Overriding to code in JAVA.

Consider a case, where Hospital provides no. of patients admitted in it. But number of patients varies with the different hospitals, For example Health India hospital has 1657 patients, IVY hospital has 2965 patients  and Apollo Hospital has 1631 patients.

Below is code of Hospital parent class which has one method getNumberOfPatients() and sub class HealthIndia, IVY and Apolo class which extends parent class & override its method.

class Hospital{

int getNumberOfPatients()

{

return 0;

}

}

class HealthIndia extends Hospital{

@Override

int getNumberOfPatients()

{

return 1657 ;

}

}

class IVY extends Hospital{

@Override

int getNumberOfPatients()

{

return 2965 ;

}

}

class Apollo extends Hospital{

@Override

int getNumberOfPatients()

{

return 1631 ;

}

}

Below is the code of MethodOverriding class which is the main class. Here we will create object for each sub class and run getNumberOfPatients() method on those objects to get patient details of each Hospital.

public class MethodOverriding{

public static void main(String args[]){

HealthIndia healthIndia=new HealthIndia();

IVY ivy=new IVY();

Apollo apollo=new Apollo();

int healthIndiaPatients= healthIndia.getNumberOfPatients();

int iVYPatients= ivy.getNumberOfPatients();

int apolloPatients= apollo.getNumberOfPatients();

System.out.println("Health India hospital patients: "+ healthIndiaPatients);

System.out.println("IVY hospital Patients: "+ iVYPatients);

System.out.println("Apollo hospital Patients: " +apolloPatients);

}

}

**Output:**

Health India hospital patients: 1657

IVY hospital Patients: 2965

Apollo hospital Patients: 1631

Now we have seen different hospitals are calling same method, but at run time it has to be decided which class method is called depending on which class object is calling the **getNumberOfPatients()** method.

**Advantages of using Method Overriding:**

* Method overriding is made to achieve Dynamic Binding or Runtime polymorphism.
* Method overriding is used to give more specific definition to the method which is already defined in the Base Class.

**Rules of Method Overriding :**

* Extended class must have same name of method as in base class.
* Method which is overridden must have same parameters.
* Method which is to be overridden should not be declared Final and static.

**Difference between Method Overloading and Method Overriding:**

**Method overloading:**

* Method Overloading happens at compile time.
* Static methods can be overloaded i.e a class can have more than one static method of same name.
* Method Overloading is done in the same class.
* Method Overloading gives better performance.
* In case of Method Overloading, return type of a method does not matter (means it will be same or different).

**Method Overriding:**

* Method Overriding happens at run time.
* Static methods cannot be overridden.
* For method overriding base and child class is required. Overriding is all about giving a specific implementation to the inherited method of a super class.
* Method Overriding gives less performance as compared to method overloading. The reason is binding of overridden method is being done at run time.
* In case of Method Overriding, Overriding methods  have more specific return type.

**Important Points:**

* Final method can’t be overridden, as final is a keyword which when applied to class prohibits it from extending in other class, when applied to method it prohibits from overriding and when applied to keyword it prohibits from getting changed.
* Static method can’t be overridden as static method belongs to the class and not to object of that class.
* We cannot override main method, as it is static method.

**Quick Revision points:**

* Method overriding is also called as Run time Polymorphism or Dynamic Binding.
* It increases the readability of code.
* It helps classes to be more specific in nature.
* Final and Static methods cannot be overridden.