

What is constructor?

Rules for Defining a constructor

Constructor overloading

**What is constructor?**

Constructors are definition blocks in java which is used to initialize non-static variables.

In java, every class in java should contain at least one constructor. If not, the compiler writes constructor in the class which is known as “**Default Constructor”**.

**Constructor:**

1. Used to create an object.
2. Used to initialize non-static variables.

**Rules for defining a constructor**

* The constructor name should be same as class name.
* It should not have return type.
* It should not contain non access specifier/modifiers: final, static abstract, synchronized.
* It can have all access modifiers: private, public, protected and default.
* It can have parameters.
* It can have throws clause i.e. we can throw exception from constructor.
* It can contain all java legal statements except return statement i.e. we cannot have return in constructor.

Demo d1 = **new** Demo();

Here new keyword calls the constructor. Constructor creates the object. d1 is the reference variable of type Demo and the Demo class object is assigned to d1. If we print d1 (ref var) we get the address of the object it is referring. You cannot create an object without a class (new is a keyword).

**Example**

class Constructor{

int i, j;

Constructor(){

i=100;

j=200;

}

}

class Demo{

public static void main(String args[]){

Constructor c1 = new Constructor();

System.out.println(“i=”+c1.i);

System.out.println(“j=”+c1.j);

}

}

Constructor does not have a return type and we can have a method with same name as class.

Example

class Demo{

Demo(){ //Constructor

}

void Demo(){ //Method

}

}

**class** Constructor\_Demo{

Constructor\_Demo(){

System.***out***.println("Class Constructor\_Demo");

}

**static**{

System.***out***.println("Class Constructor\_Demo static block");

}

{

System.***out***.println("Class Constructor\_Demo non-static block");

}

}

**public** **class** Demo{

**static**{

System.***out***.println("class Demo static block");

}

{

System.***out***.println("class Demo non-static block");

}

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

System.***out***.println("Main starts");

Constructor\_Demo cd1 = **new** Constructor\_Demo();

Constructor\_Demo cd2= **new** Constructor\_Demo();

System.***out***.println("Main ends");

**new** Demo(); //create default constructor

}

}

Output

class Demo static block

Main starts

Class Constructor\_Demo static block

Class Constructor\_Demo non-static block

Class Constructor\_Demo

Class Constructor\_Demo non-static block

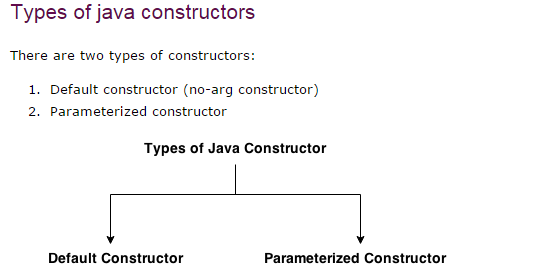
Class Constructor\_Demo

Main ends

class Demo non-static block

**Note:**

Whenever you write any Constructor, the compiler will not create any constructor. If no constructor is created by user then compiler creates a default constructor.



1. **Default Constructor**

The constructor which created by compiler is called

Default Constructor.

Example:

class DefaultConstructor{

pubic static void main(String args[]){

DefaultConstructor dc = new DefaultConstructor();

}

}

Compiler:

DefaultConstructor(){

}

1. **Parameterized Constructor**

The constructor with argument or the constructor that have parameters is called Parameterized Constructor.

Parameterized constructor is used to provide different values to the objects.

Example:

class ParameterizedConstructor{

int i;

int j;

ParameterizedConstructor(int valOfI, int valOfJ){

i=valOfI;

j=valOfJ;

}

void display(){

System.out.println(“Value of i: “+i);

System.out.println(“Value of j: “+j);

}

public static void main(string args[]){

ParameterizedConstructor pc = new ParameterizedConstructor(); //compile time error.

ParameterizedConstructor pc1 = new ParameterizedConstructor(10,20);

ParameterizedConstructor pc2 = new ParameterizedConstructor(30,40);

}

}

**Constructor Overloading**

When we have more than one constructor for a class we call it as constructor overloading.

**Note:** When you overload the constructor the number of argument should be different OR types of argument should be different OR sequence of argument should be different.

Example:

**class** ConstructorOverloading{

**long** i;

**long** j;

/\*ConstructorOverloading(){

System.out.println(“No argument constructor”);

i=10;

j=20;

}\*/

ConstructorOverloading(**int** k){

System.***out***.println("Number of argument is different");

i=k;

}

ConstructorOverloading(**int** a, **int** b){

i=a;

j=b;

}

ConstructorOverloading(**long** a, **long** b){

System.***out***.println("Type of argument is different");

i=a;

j=b;

}

ConstructorOverloading(**int** a, **long** b){

i=a;

j=b;

}

ConstructorOverloading(**long** a, **int** b){

System.***out***.println("sequnce of argument is different");

i=a;

j=b;

}

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

//ConstructorOverloading co1=new ConstructorOverloading();

//System.out.println(co1.i + " " + co1.j);

ConstructorOverloading co2=**new** ConstructorOverloading(10);

System.***out***.println(co2.i + " " + co2.j);

ConstructorOverloading co3=**new** ConstructorOverloading(10,20);

System.***out***.println(co3.i + " " + co3.j);

ConstructorOverloading co4 = **new** ConstructorOverloading(10l,20l);

System.***out***.println(co4.i + " " + co4.j);

ConstructorOverloading co5 = **new** ConstructorOverloading(10,20l);

System.***out***.println(co5.i + " " + co5.j);

ConstructorOverloading co6 = **new** ConstructorOverloading(20l,10);

System.***out***.println(co6.i + " " + co6.j);

}

}

Output:

