   

**JAVA II**

Constructor :

So for every object created the constructor will provide a seperate date memebers for ex : if a class student has data memebers name and roll number and we are creating 600 objects of a class so for evry class created constructor will make a sperate data memebers for each 600 objects and jvm will intialize them with the value null and 0 respectively in this case . So we need to perform intialization then the object creaation is meaningful imagine having all the 600 objects name as null and roll number 0 its meaningless . So it is compulsory to intialize the object when created so that it respond us properly .

Constructor main purpose is to perofrm initialization of an object . Constructor job is to intialize the object and not to create it .

Student s1 = new Student ();

Reference variable. : s1 is the referencevariable of type student ; Student () ; : will intialize the object

new : will create the object . For every object created seperate copies of instances variable will be created ( String name , int roll). For evry instamce variable created jvm will provide default values.

This means current object . Constructor provides values for our instance variables .

Student s1 = new Student ("Ravi" , 102); till new the jvm will by default provide null value to String name and value 0 to int roll number and when call the constructor Student ("Ravi" , 102); then it intialize the instance variable String name with Ravi and roll number with 102.

Constructor sample program :

class Student

{ String name ; int rollno ; // instance variable

Student (String name , int rollno) //constructor

{ [this.name](http://this.name) = name ; this.rollno = rollno; }

psvm(String a[])

{ Student s1 = new Student("Yash", 127);

Student s2 = new Student("kamal" , 129);

[Sout(s1.name](about:blank) + " " s1.rollno);

[Sout(s2.name](about:blank) + " " s2.rollno);

}

}

After creating the object with the help of new immediately constructor will execute . For every object constructor will be executed seperately. If we create 3 objects the constructor will be called 3 times .

Rules for cosntructor :

1. Name of the constructor and name of the class must be same.

2 . Whenever we create object constructor is called immediately . As we are not calling the constructor the JVM calls it immediately thus return type for constructor is not applicable.

If we provide a return type to a constructor the compiler will not give left and right it will treat it as a method .

So from this we came to know that we can create a method whose name is same as that of the class name

class Test

{ void Test ()

{ Sout ("Display"); }

psvm(String []a)

{ Test t1 = new Test();

Test t2 = new Test();

t1.test();

} } but it is not recommended to use the method name which is same as that of the class name . As class name and constructor name should be same . And we should follow this comvention.

3. In java there are almost 12 modifiers are available (public ,default ,private , protected, final , static , abstract , strictfp,synchronized...)

4. So fir constructors the allowed modifiers i.e. the modifier which we can use with the constructor are public , protected , default and private .

Singleton classes : in which we have to create only one object for the class and its constructor modifier should be private . It also increase the performance. So while creating singleton classes private constructor are useful.

Default constructor:

Every class in java contains constructor . Simultaneously default constructor (provided by compiler ) and the constructor cretaed by user cannot exist . If we create a constructor of our own i.e. explicit constructor the compiler will not provide us with the default constructor .

It is always no args constructor i.e. unparameterized constructor . So default constructor is always no args constructor but every no arg constructor is not default constructor.

The default access modifier of the constructor is same as the access modifier of the class . If the class is public the constructor is public if the class is private the constructor is private . So don't get confuse by the name that default constructor access modifer is default always it depends on class access modifier. So the access modifier of the default constructor is same as that of the class modifier. But this rule is only applicable for [ public ,<default>].

Default constructor only contains one line super(); it does not contain any other code . It is a no argument call to a super() class constructor. So the parent class constructor will be executed because of this call .

The first line inside a constructor should be super(); or this(); if you dont write anything then by default the constructor will generate or write super(); inside the cosntructor. If you miss any of this above mentioned points the compiler will generate the code by itself .

Case 1:

class Test

{ Test ()

{ Sout ("Constructor");

super();

}

} the compiler will give left and right and will give error as call to super must be first statement in constructor . Thus it will give compile time error.

Case 2:

class Test

{ Test (){

super();

this();

 Sout ("Constructor");

}

} the compiler will give left and right and will give error as call to this must be first statement in constructor . Thus it will give compile time error. As we cannot use super and this simultaneously in a constructor. We can use either of them.

Case 3:

class Test

{public void m1 (){

super();

Sout ("Constructor");

}

} super () : it means we are calling super class constructor.

this () : it means we are calling current class constructor .

So we can call a constructor directly from a constructor but not with the method.Thus it will give compile time error. super must be first statement in constructor so this the compile time error we would get.

Super () and this () :

1. We can only use it inside a constructor .

2. We should use them in the first line of the constructor.

3. We can use only one of them at a time not both simultaneously.

Super and this as keyword :

class P

{

String s = "Parent class variable "

}

class C extends P

{

String s = "Child class variable " ;

public void m1()

{

     Sopln(s); // Child class variable

     Sopln(this.s) ; // Child class variable

     Sopln(super.s) ; // Parent class variable

}}

class Test

{

psvm (String a[])

{

C t = new C();

t.m1();

}}

So this refers to the current class instance variable and and super refers to super class instance variable .

We can use super and this anywhere except static area .

Constructor overloading :

Within a class we can write any number of constructor.

Multiple constructors with same name but with different argument types are called Constructor overloading.

Constructor inheritance and overriding :

The inheritance and overriding concepts are not available for constructors it is available for methods .

class P

{

P()

{}

}

class C extends P

{

C(int i) here only one constructor is present

{

}}

C c = new C (); it will give compile time error as the parent class constructor is not available by default to the child class constructor . So if it is not available by default so there is no chance of overriding.

class P

{

public void m1()

{Sopln("Method1 ");

}

}

class C extends P

{

public void m2()

{ Sopln("Method 2"); //by default it has 2 methods m1 as well as m2

}}

C c = new C (); so methods support overriding and inheritance .

Interface can't have constructor . Every variable presnt inside an interface is public static final so there is no chance of existence of instance variabke . So constructor concept is not applicable for interface as the main job of the constructor is to intialize the instance variable.

But abstract class can contain instance variable so constructor is applicable incase of abstract class.

Inheritance:

Is a relationship

Code reusability

extends keyword

On the parent reference child specific method cannot be called

class Parent {

    void parentMethod() {

        System.out.println("Parent method");

    }

}

class Child extends Parent {

    void childMethod() {

        System.out.println("Child method");

    }

}

public class Main {

    public static void main(String[] args) {

        Parent parentRef = new Child();

        parentRef.parentMethod(); // This is allowed

        // parentRef.childMethod(); // This will give a compilation error

    }

}because child memebers are not by default available to the parents .

Parent reference can be used to store child class object but by using parent class reference we cannot call child class method.