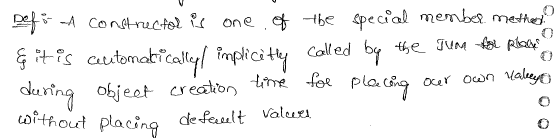
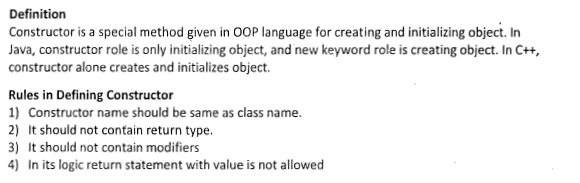
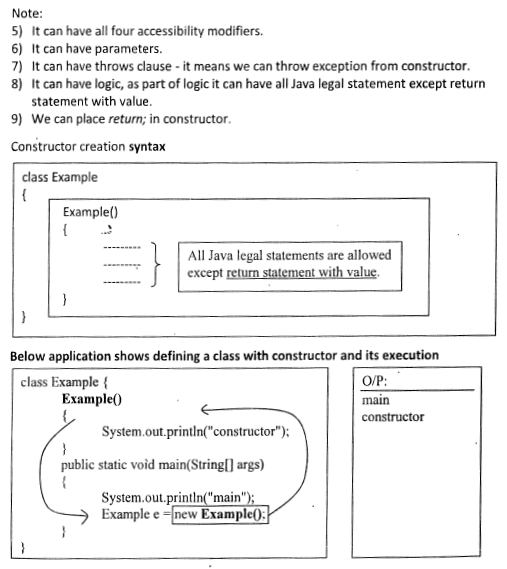
**Java Constructors**

Constructors are the functionalities which would be executed automatically at the time of creating object of the class.

All the statements which are supposed to get executed first i.e. at the time of creating object should be present inside the constructors.

****

****

****

A **constructor** looks more like a method but without return type. Moreover, the name of the constructor and the class name should be the same. The advantage of constructors over methods is that they are **called implicitly** whenever an object is created. In case of methods, they must be called explicitly. To create an object, the constructor must be called. Constructor gives properties to an object at the time of creation itself (else, it takes some method calls with extra code to do the same job). Programmer uses constructor for initializing variables, instantiating objects and setting colors. Constructor is equivalent to [init()](http://way2java.com/applets/life-cycle-of-applet/) method of an[applet](http://way2java.com/applets/applets-vs-applications/).

**Constructors: Initializing an Class Object in Java Programming**

**Constructor in java** is a special type of method that is used to initialize the object.

Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

1. Objects contain their **own copy of Instance Variables**.
2. It is very difficult to **initialize each and every instance variable** of each and every object of Class.
3. Java allows **objects to initialize themselves when they are created**. Automatic initialization is performed through the use of a constructor.
4. A Constructor **initializes an object** as soon as object gets created.
5. Constructor gets **called automatically after creation of object and before completion of new Operator**.

**Some Rules of Using Constructor :**

1. Constructor **Initializes an Object**.
2. Constructor **cannot be called** like methods.
3. Constructors**are called automatically** as soon as object gets created.
4. Constructor **don’t have any return Type.** (even Void)
5. Constructor name is same as that of “**Class Name**“.
6. Constructor **can accept parameter**.

**Example**:

**class** Bike1{

Bike1(){

System.out.println("Bike is created");

}

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

Bike1 b=**new** Bike1();

}

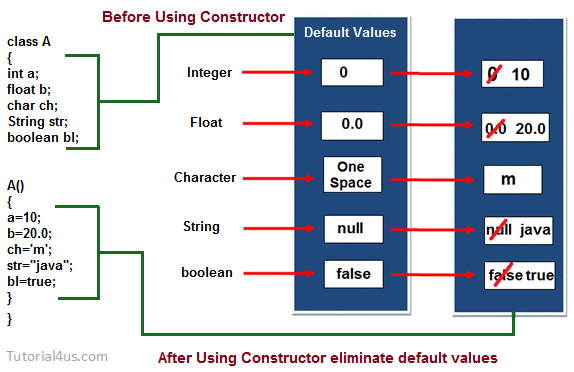
}

**Output**:

**Bike is created**

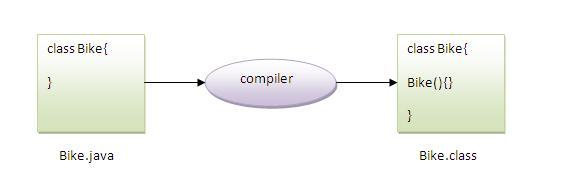
**How Constructor eliminate default values ?**

Constructor are mainly used for eliminate default values by user defined values, whenever we create an object of any class then its allocate memory for all the data members and initialize there default values. To eliminate these default values by user defined values we use constructor.



***Default Constructor****:*

#### If there is no constructor in a class, compiler automatically creates a default constructor.



### Q) What is the purpose of default constructor?

Default constructor provides the default values to the object like 0, null etc. depending on the type.

### Example of default constructor that displays the default values

*class Student3{*

*int id;*

*String name;*

*void display(){*

*System.out.println(id+" "+name);*

*}*

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

*Student3 s1=new Student3();*

*Student3 s2=new Student3();*

*s1.display();*

*s2.display();*

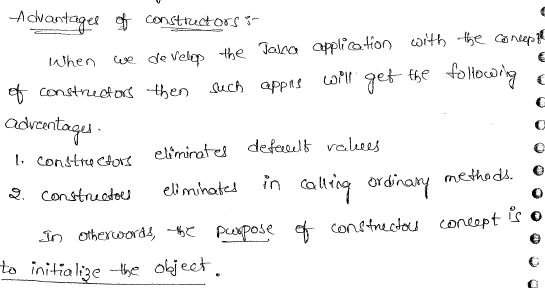
*}*

*}*

**Output:**

0 null

0 null



**Live Example : How Constructor Works ?**

*class Rectangle {*

***int*** *length;*

***int*** *breadth;*

*Rectangle(){*

*length = 20;*

*breadth = 10;*

*}*

*}*

*class RectangleDemo {*

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

*Rectangle r1 = new Rectangle();*

*System.out.println("Length of Rectangle : " + r1.length);*

*System.out.println("Breadth of Rectangle : " + r1.breadth);*

*}*

*}*

Output :

*java RectangleDemo*

*Length of Rectangle : 20*

*Breadth of Rectangle : 10*

Explanation :

1. new Operator will create an object.
2. As soon as Object gets created it will call Constructor-

*Rectangle(){ //This is Constructor*

*length = 20;*

*breadth = 10;*

*}*

1. In the above Constructor Instance Variables of Object r1 gets their own values.
2. Thus Constructor **Initializes an Object as soon as after creation**.
3. It will print Values initialized by Constructor –

*System.out.println("Length of Rectangle : " + r1.length);*

*System.out.println("Breadth of Rectangle : " + r1.breadth);*

**Java constructor example**

Yet Another Constructor Example: More Detailed Example (Java Programming)

class Rectangle {

**int** length;

**int** breadth;

Rectangle(){

length = 20;

breadth = 10;

}

**void** setDiamentions(){

length = 40;

breadth = 20;

}

}

class RectangleDemo {

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

Rectangle r1 = new Rectangle();

System.out.println("Length of Rectangle : " + r1.length);

System.out.println("Breadthof Rectangle: " + r1.breadth);

r1.setDiamentions();

System.out.println("Length of Rectangle : " + r1.length);

System.out.println("Breadthof Rectangle: " + r1.breadth);

}

}

**Output :**

*java RectangleDemo*

*Length of Rectangle : 20*

*Breadth of Rectangle : 10*

*Length of Rectangle : 40*

*Breadth of Rectangle : 20*

**Explanation :**

1. After the Creation of Object , Instance Variables have their own values inside.
2. As soon as we call method , **values are re-initialized**.

***Java parameterized constructor***

### Parameterized Constructors :

A constructor that have parameters is known as parameterized constructor.

### Why use parameterized constructor?

|  |
| --- |
| Parameterized constructor is used to provide different values to the distinct objects. |

1. Constructor Can Take Value , Value is Called as – “**Argument**“.
2. Argument **can be of any type** i.e Integer,Character,Array or any Object.
3. Constructor **can take any number of Argument**.

**Live Example: Constructor Taking Parameter in Java Programming**

class Rectangle {

**int** length;

**int** breadth;

Rectangle(**int** len,**int** bre){

length = len;

breadth = bre;

}

}

class RectangleDemo {

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

Rectangle r1 = new Rectangle(20,10);

System.out.println("Length of Rectangle : " + r1.length);

System.out.println("Breadth of Rectangle : " + r1.breadth);

}

}

**Output :**

Length of Rectangle : 20

Breadth of Rectangle : 10

**Explanation :**

Carefully observe above program – You will found something like this –

Rectangle r1 = new Rectangle(20,10);

This is Parameterized Constructor taking argument.These arguments are used for any purpose inside Constructor Body.

* New Operator is used to **Create Object**.
* We are passing **Parameter to Constructor** as 20,10.
* These parameters are assigned to **Instance Variables of the Class**.

**We can Write above statement like –**

*Rectangle(****int*** *length,****int*** *breadth) {*

*length = length;*

*breadth = breadth;*

*}*

OR

*Rectangle(****int*** *length,****int*** *breadth){*

***this****.length = length;*

***this****.breadth = breadth;*

*}*

But if we use Parameter name same as Instance variable then compiler will recognize instance variable and Parameter but user or programmer may confuse. Thus we have used “[this keyword](http://www.c4learn.com/javaprogramming/this-keyword-refer-current-object-in-java-programming/)” to specify that “**Variable is Instance Variable of Object – r1**“.

### Example of parameterized constructor

*class Student4{*

*int id;*

*String name;*

*Student4(int i,String n){*

*id = i;*

*name = n;*

*}*

*void display(){*

*System.out.println(id+" "+name);*

*}*

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

*Student4 s1 = new Student4(111,"Karan");*

*Student4 s2 = new Student4(222,"Aryan");*

*s1.display();*

*s2.display();*

*}*

*}*

**Output:**

111 Karan

222 Aryan

## Constructor Overloading in Java

|  |
| --- |
| Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type. |

### Example of Constructor Overloading

***class Test{***

***int id;***

***String name;***

***int age;***

***Test(int i,String n){***

***id = i;***

***name = n;***

***}***

***Test(int i,String n,int a){***

***id = i;***

***name = n;***

***age=a;***

***}***

***void display(){***

***System.out.println(id+" "+name+" "+age);***

***}***

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

***Test t1 = new Test(111,"Karan");***

***Test t2 = new Test(222,"Aryan",25);***

***t1.display();***

***t2.display();***

***}***

***}***

**Output**:

*111 Karan 0*

*222 Aryan 25*

**Difference between constructor and method in java**

There are many differences between constructors and methods. They are given below.

|  |  |
| --- | --- |
| **Java Constructor** | **Java Method** |
| Constructor is used to initialize the state of an object. | Method is used to expose behaviour of an object. |
| Constructor must not have return type. | Method must have return type. |
| Constructor is invoked implicitly. | Method is invoked explicitly. |
| The java compiler provides a default constructor if you don't have any constructor. | Method is not provided by compiler in any case. |
| Constructor name must be same as the class name. | Method name may or may not be same as class name. |

**Java Copy Constructor**

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

* **By constructor**
* **By assigning the values of one object into another**
* **By clone() method of Object class**

In this example, we are going to copy the values of one object into another using java constructor.

**Example:**

***class*** *Student6{*

***int*** *id;*

*String name;*

*Student6(****int*** *i,String n){*

*id = i;*

*name = n;*

*}*

*Student6(Student6 s){*

*id = s.id;*

*name =s.name;*

*}*

***void*** *display(){*

*System.out.println(id+" "+name);*

*}*

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

*Student6 s1 =* ***new*** *Student6(111,"Karan");*

*Student6 s2 =* ***new*** *Student6(s1);*

*s1.display();*

*s2.display();*

*}*

*}*

**Output:**

111 Karan

111 Karan

## Copying values without constructor

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

**Example:**

***class*** *Student7{*

***int*** *id;*

*String name;*

*Student7(****int*** *i,String n){*

*id = i;*

*name = n;*

*}*

*Student7(){*

*}*

***void*** *display(){*

*System.out.println(id+" "+name);*

*}*

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

*Student7 s1 =* ***new*** *Student7(111,"Karan");*

*Student7 s2 =* ***new*** *Student7();*

*s2.id=s1.id;*

*s2.name=s1.name;*

*s1.display();*

*s2.display();*

*}*

*}*

**Output:**

*111 Karan*

*111 Karan*

### Q) Does constructor return any value?

**Ans:** Yes, that is current class instance (You cannot use return type yet it returns a value).

### Q) Can constructor perform other tasks instead of initialization?

Yes, like object creation, starting a thread, calling method etc. You can perform any operation in the constructor as you perform in the method.

When we declare instance variables and not initialized with any value then they automatically initialized with default values.

But if we want to initialize instance variables with our own values then we can initialize instance variables in following 2 locations,

1. At the time of declaration

2. using contructor

**Constructors**

Constructor is also a method which is especially designed to initialize instance variables.

To create a constructor we have to follow following rules

**Rules**

1. Constructor name must be same as ClassName

2. Constructor cannot take any return type but if we write any return type then the code is valid but it is considered as normal method

3. We cannot return any value from the constructor

4. Constructors take parameters

5. Based on the number of parameters constructors are classified into following 2 types,

**a. 0 parametrized constructor**

If we declare any constructor without any parameters then it is called as 0 parametrized constructor.

**syntax:**

*class ClassName{*

*ClassName(){*

*//statements;*

*}*

*}*

**b. Parametrized constructor**

If we declare any constructor withparameters then it is called as parameterized constructor.

**syntax:**

*class ClassName{*

*ClassName(parameters){*

*//statements;*

*}*

*}*

6. Constructors can be invoked and accessed at the time of object creation.

**// wap to demo on constructors**

*class Student{*

*int rno;*

*String name;*

*double fee;*

***/\*0 parameterized constructor \*/***

*Student(){*

*rno=1;*

*name="sachin";*

*fee=1200.0;*

*}*

***/\* parameterized constructor \*/***

*Student(int rn,String nm,double fe){*

*rno=rn;*

*name=nm;*

*fee=fe;*

*}*

***/\*normal method\*/***

*void display(){*

*System.out.println(rno+"\t"+name+"\t"+fee);*

*}*

*}*

*class Constructor1{*

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

*Student s1 = new Student();*

*s1.display();*

*Student s2 = new Student();*

*s2.display();*

*Student s3 = new Student(2,"sehwagh",1300);*

*s3.display();*

*Student s4 = new Student(3,"dhoni",1500);*

*s4.display();*

*}*

*}*

1. Memory allocated for instance variables at the time of declaration of object

2. Memory allocated for instance variables within the Heap memory area of JVM

3. Memory allocated for instance variables 1 time for Every object.

4. Different copy of instance variables given to every object of the particular class. It means the data changed in one object can not effect the data of other object.

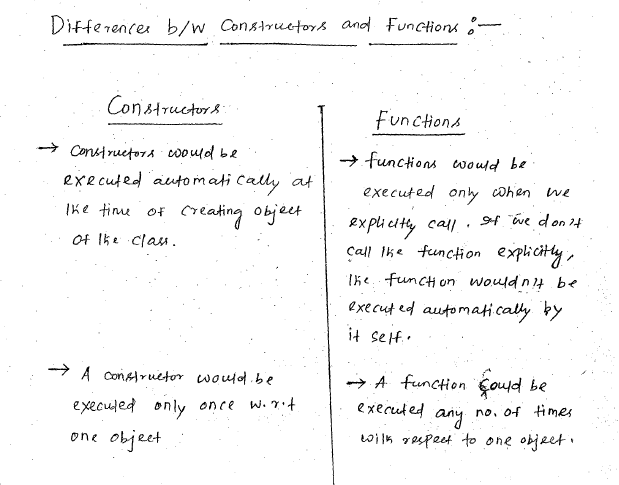
**Note:**

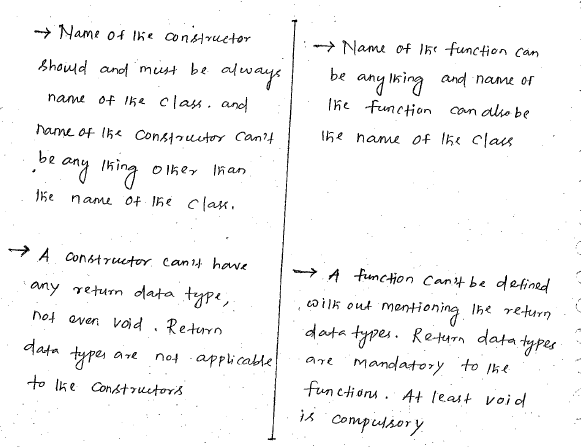
If we want to store same data in all the objects initially then we have to initialize instance variables at the time of declaration and we can also use 0 parametrized constructor

If we want to store different data in all the objects initially then we have to initialize instance variables using parametrized constructor

In java we can not create any object for any class without using constructor.

In java every class both user defined or predefined classes contains a constructor whether we specify or not.

****

****

**Example:**

Class Cdemo{

int i,j;

Cdemo(){

System.out.println(“Beginning of Cdemo”);

System.out.println(i);

System.out.println(j);

System.out.println(“End oof Cdemo”);

}

void fun1(){

System.out.println(“fun1 of Cdemo”);

}

public static void main(String[] args){

Cdemo d1=new Cdemo();

//Here we are asking JVM to create object of the class Cdemo by executing the constructor.

//Here Cdemo() represents the constructor of the class Cdemo defined to accept zero arguments.

d1.Cdemo();  
// we can’t execute a constructor by calling it with its name using dot (.) operator and object like a function.

Cdemo d1=new Cdemo();

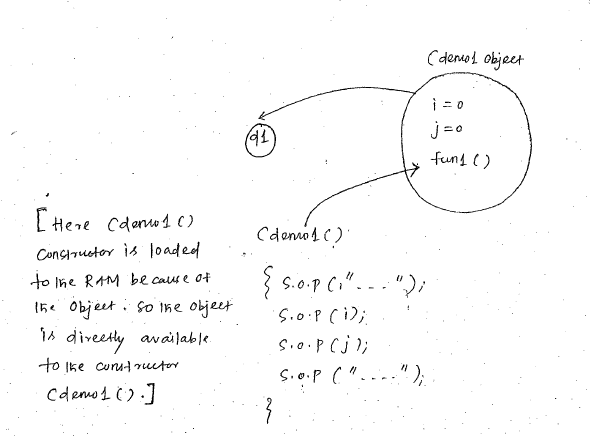
// A constructor can’t be executed more than once w.r.t the same object.

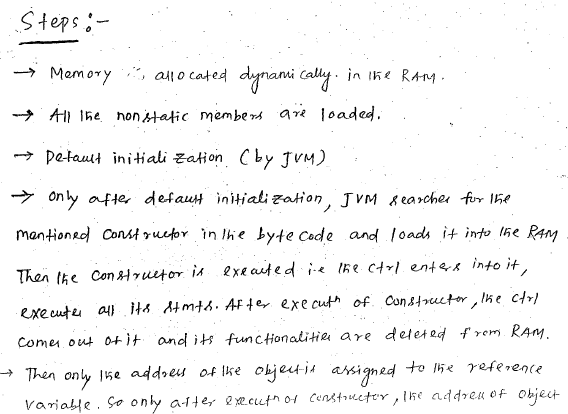
Cdemo d2=ne Cdemo();

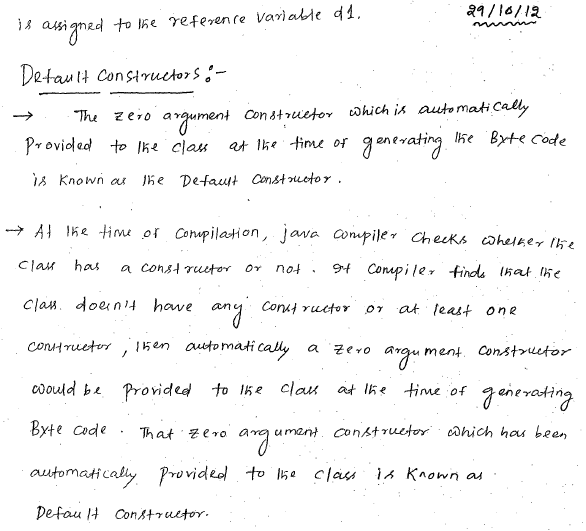
//If we want to call/ execute the same constructor for the second time, we have to create another object.

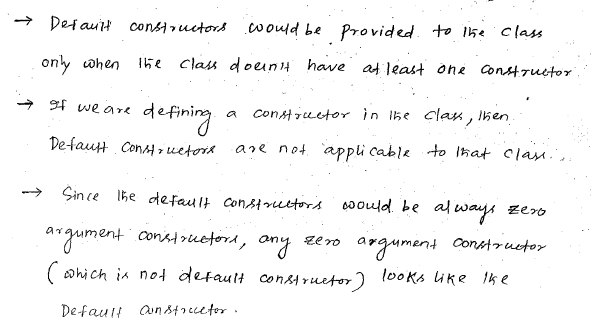
}

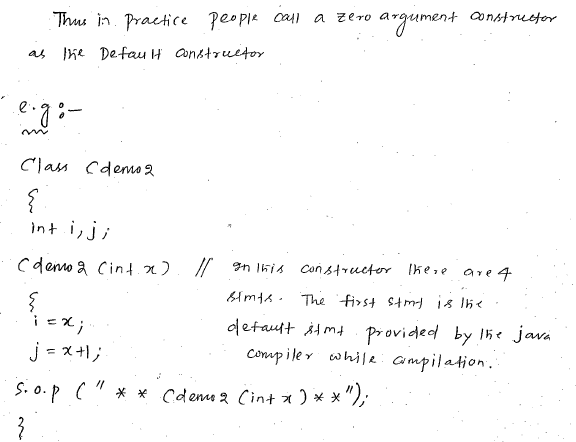
}

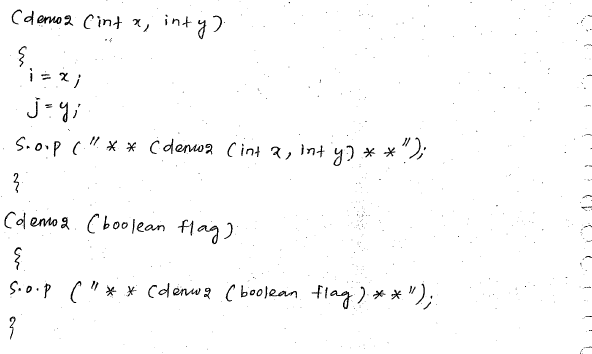


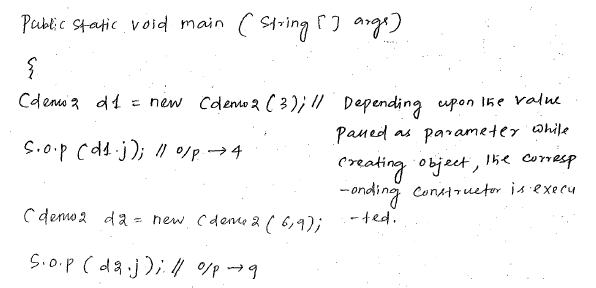


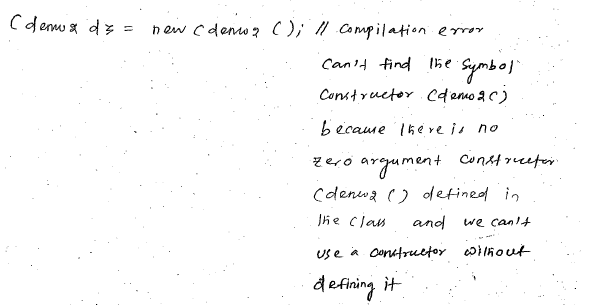


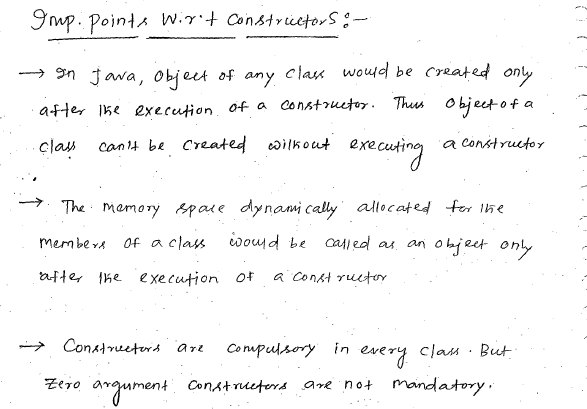


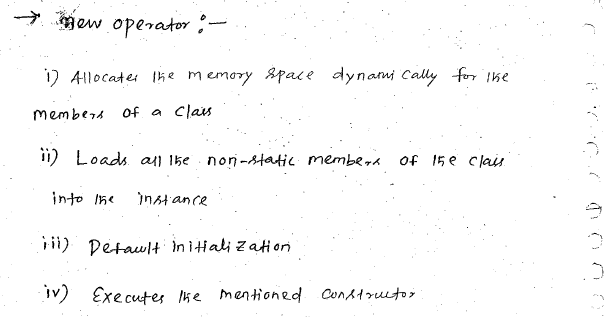


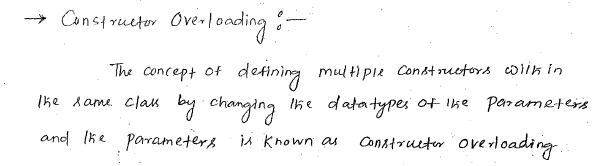


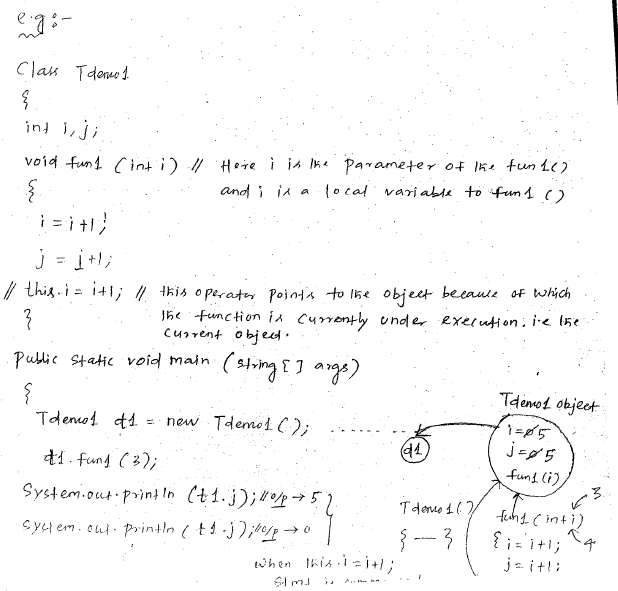


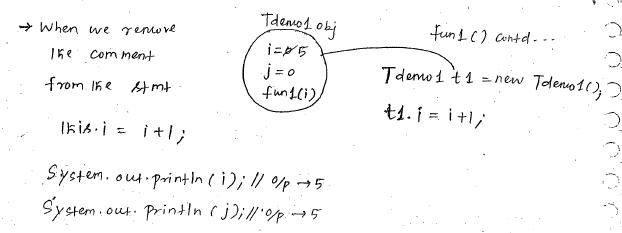


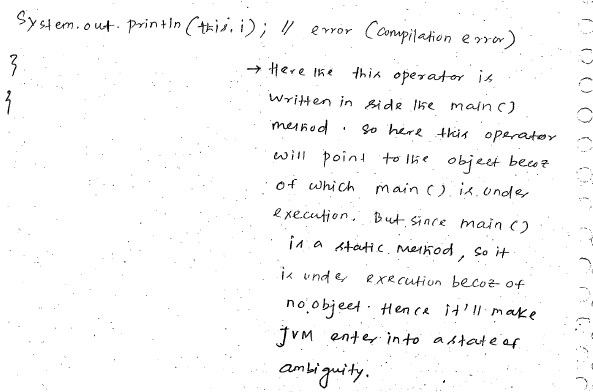


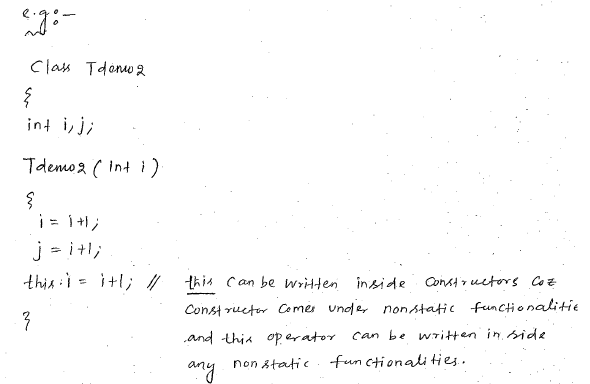


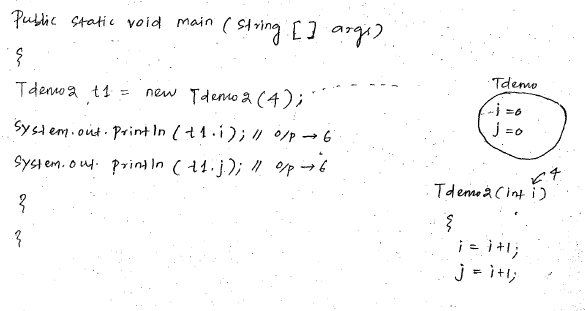


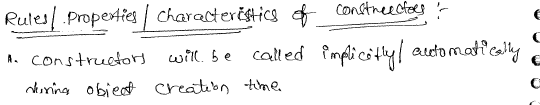


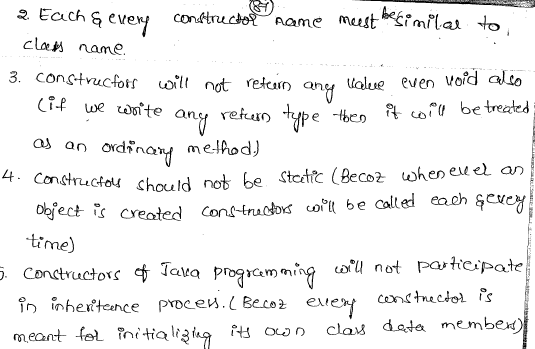


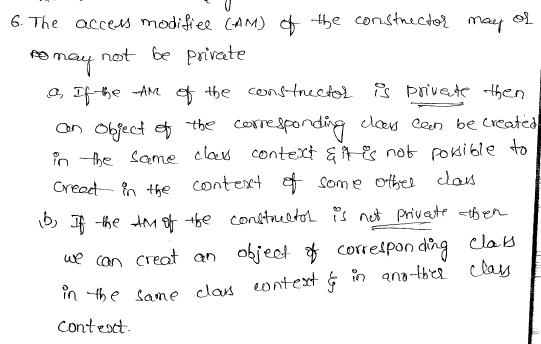


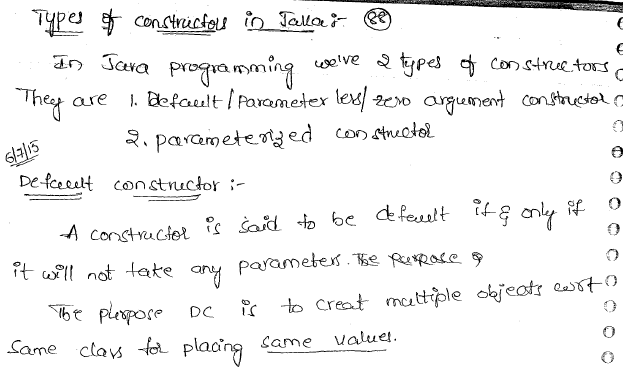


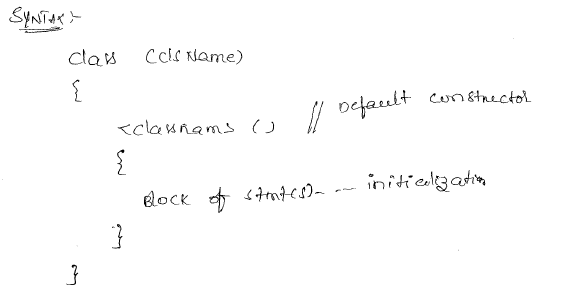


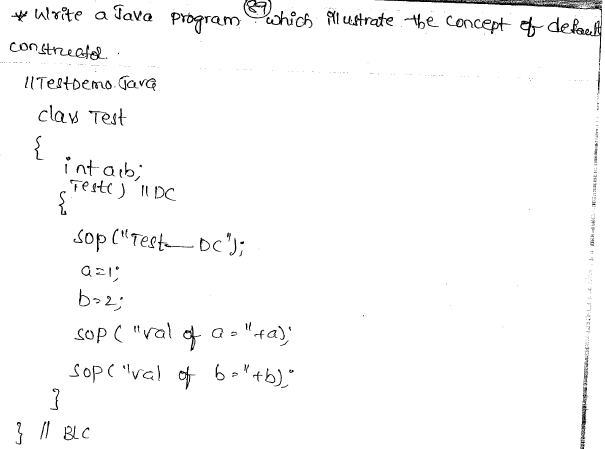


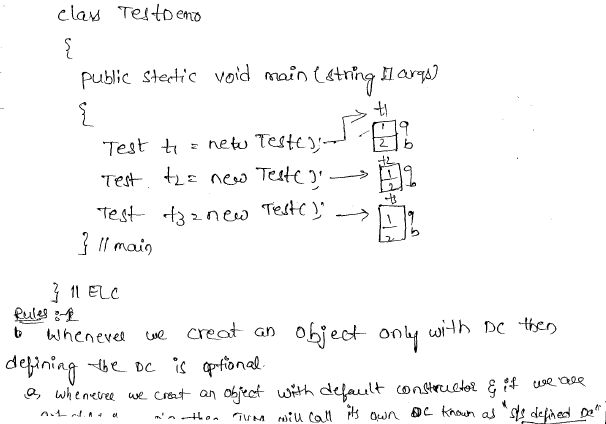


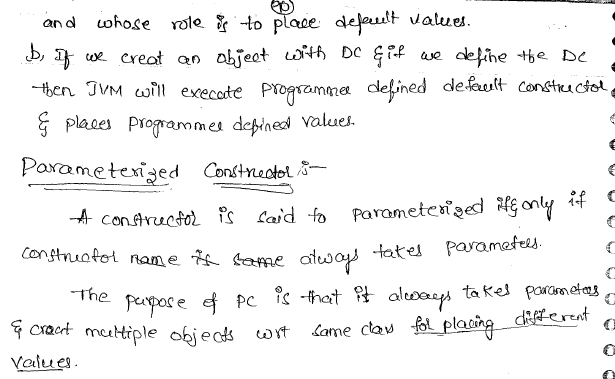


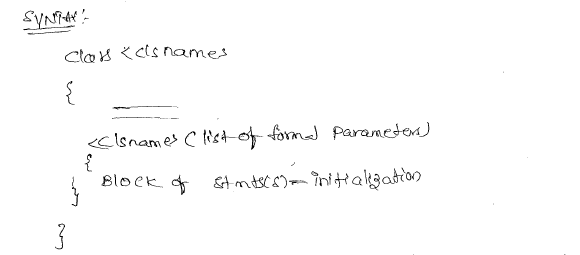


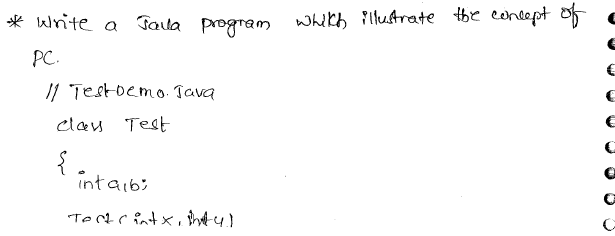


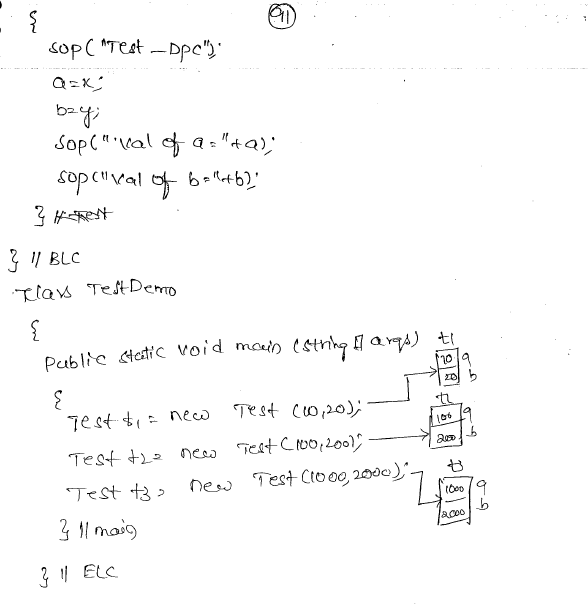


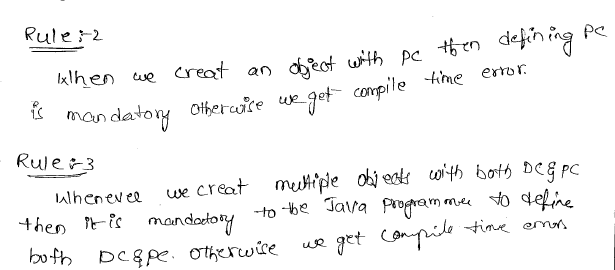


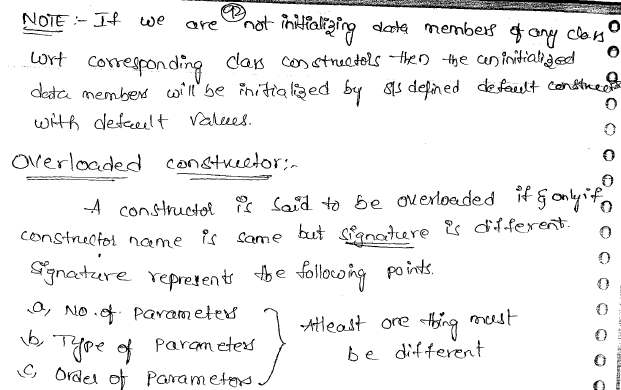


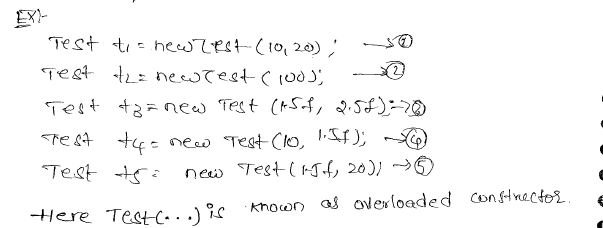


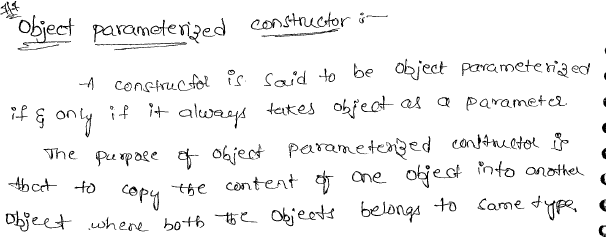


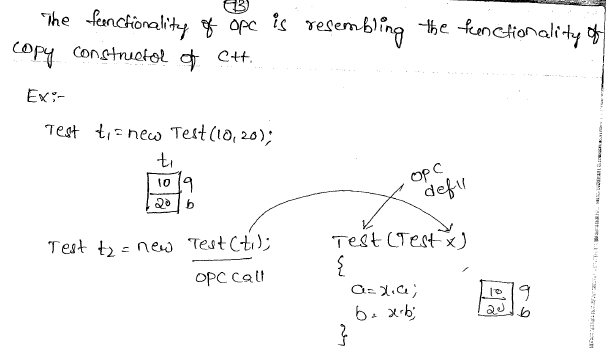


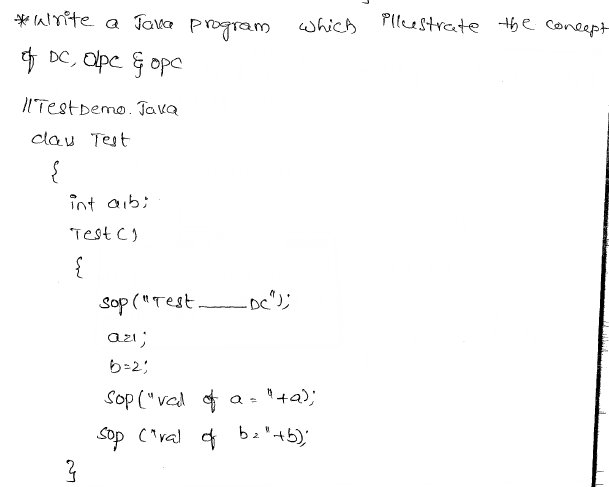


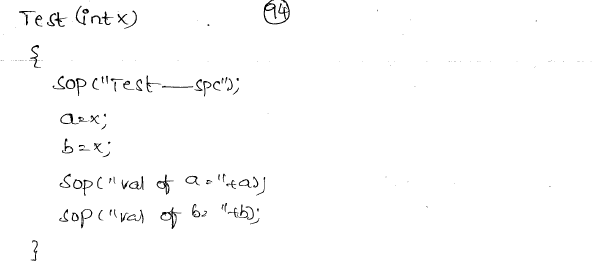


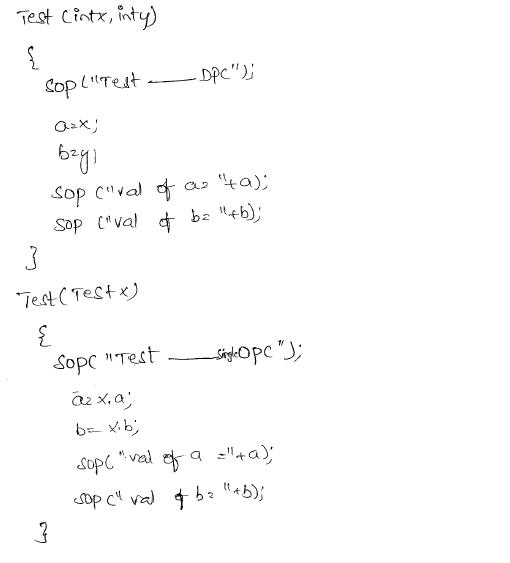


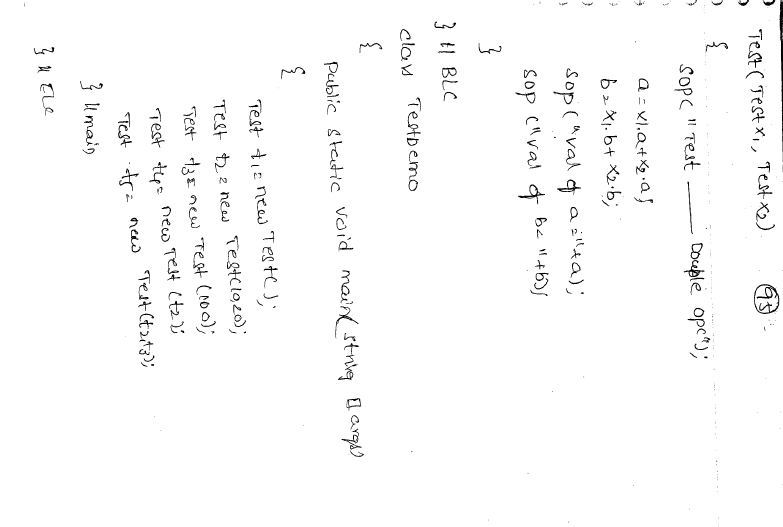


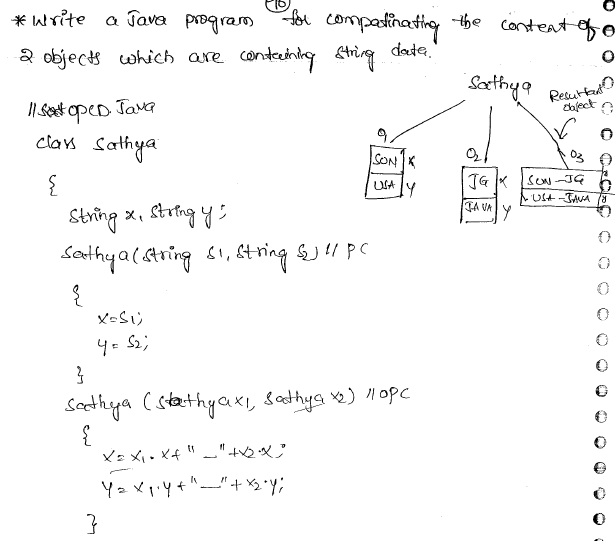


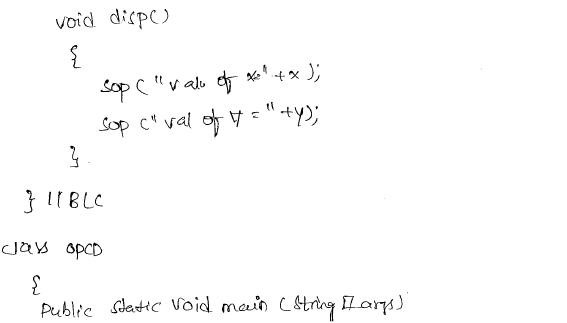


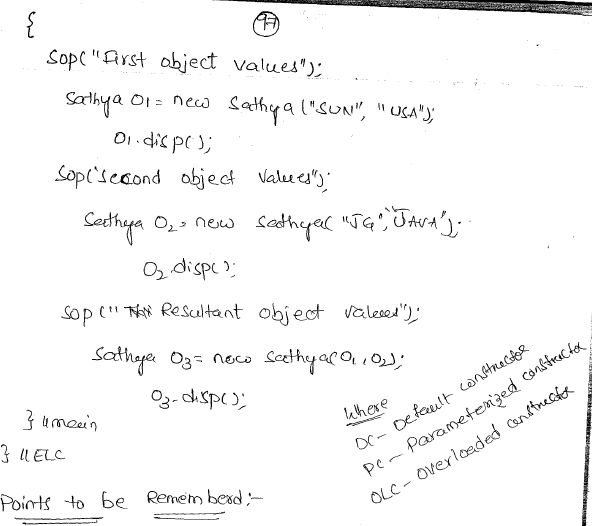


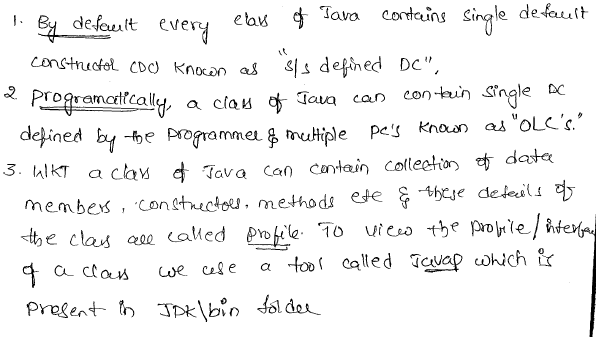


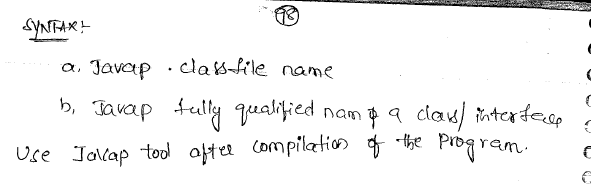


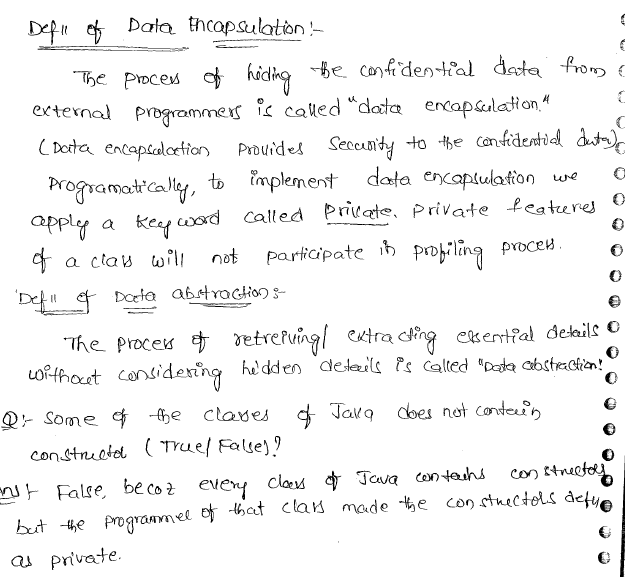












**this() with Constructors**

Suppose by accessing one constructor, the programmer may require the functionality of other constructors also but by creating one object only. For this, Java comes with this(). "**this()**" is used to access one constructor from another "within the same class". Depending on the parameters supplied, the suitable constructor is accessed.

*public class Perimeter{*

*public Perimeter(){// I*

*System.out.println("From default");*

*}*

*public Perimeter(int x){ // II*

*this();*

*System.out.println("Circle perimeter: " + 2\*Math.PI\*x);*

*}*

*public Perimeter(int x, int y) { // III*

*this(100);*

*System.out.println("Rectangle perimeter: " +2\*(x+y));*

*}*

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

*Perimeter p3 = new Perimeter(10, 20); // III*

*}*

*}*



In the code, creating object **p3**, the III constructor is accessed. From III, with "this(100)" statement, the II constructor is accessed. Again from II, the I is accessed without the statement "this()". As per the parameter supplied to this(), the appropriate or corresponding constructor is accessed.

**Rules of using this()**

A few restrictions exist for the usage of this().

1. If included, this() statement must be the first one in the constructor. You cannot write anything before this() in the constructor.
2. With the above rule, there cannot be two this() statements in the same constructor (because both cannot be the first).
3. this() must be used with constructors only, that too to call the same class constructor (but not super class constructor).

**Constructors in Inheritance**

Constructors come with some implicit behavior. Most of the time, the programmer would like to have the behavior and properties of super class constructor also being in subclass. For this reason, "a subclass constructor calls super class default constructor implicitly".

*class Hello{*

*public Hello(){*

*System.out.println("From Hello constructor");*

*}*

*}*

*class Test extends Hello{*

*public Test(){*

*System.out.println("From Test constructor");*

*}*

*}*

*public class Demo extends Test{*

*public Demo(){*

*System.out.println("From Demo constructor");*

*}*

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

*Demo d1 = new Demo();*

*}*

*}*



In the above program **Demo**extends **Test**and in turn **Test** extends **Hello**. It is [**multilevel**](http://way2java.com/oops-concepts/types-of-inheritance/)inheritance. In the main() method, we are accessing Demo() constructor by creating the object**d1** of Demo class. The Demo() constructor calls implicitly Test() constructor because it is the super class of Demo. Similarly Hello is the super class of Test and thereby Test() constructor calls Hello() constructor. Now you get the output of all the three constructors. The output you get in reverse order.

**super() with Constructors**

We came to know in the previous program that subclass constructor calls super class constructor implicitly. Let us make the rule more clear. Either the "**default or the overloaded constructor of the subclass calls super class default constructor implicitly**". If the programmer would like to call the overloaded constructor of the super class from subclass constructor, he can use **super()**. Compiler comes to know which  
super class constructor is to be called depending upon the parameter list of super().

*class Test{*

*public Test() // I*

*{*

*System.out.println("From Test default");*

*}*

*public Test(int x) // II*

*{*

*this();*

*System.out.println(x);*

*}*

*}*

*public class Demo extends Test*

*{*

*public Demo() // III*

*{*

*super(100);*

*System.out.println("From Demo default");*

*}*

*public Demo(int x) // IV*

*{*

*this();*

*System.out.println(x);*

*}*

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

*{*

*Demo d1 = new Demo(10); // IV*

*}*

*}*



In the above code, super class Test and subclass Demo come with two constructors each – one default and one overloaded.

*this();*

We know earlier, this() calls same class default constructor as there are no parameters.

*super(100);*

From III constructor, we are calling super class II constructor with super(100). "**super(100)**" calls super class matching integer parameter constructor, I.

if the statement super(100) is omitted in III, the III calls implicitly I.

**Rules of using super()**

Same restrictions of this() exist for the usage of super() also.

1. If included, super() statement must be the first one in the constructor. You cannot write anything before super() in the constructor.
2. With the above rule, there cannot be two super() statements in the same constructor (because both cannot be the first). That is, we can have only one of either this() or super() in the constructor.
3. super() must be used with constructors only, that too to call the super class constructor from subclass constructor.

# [Basic Rules Need to Follow When Writing Constructors In Java](http://javaconceptoftheday.com/constructors-in-java/)

* Name of the constructor must be same as that of a class name. If you give another name it will give compile time error. If you give another name, it is neither a method because of no return type, nor constructor because name is different from class name.

*class A*

*{*

*A()*

*{*

*// Constructor of Class A*

*}*

*A1()*

*{*

*// Compile time error, It is neither a constructor nor a method*

*}*

*}*

* Constructors must not have a return type. If you keep return type for the constructor, it will be treated as another method.But compiler gives a warning saying that this method has a constructor name. That means, it is legal to have method name same as constructor name or same as class name but it is not recommended.

*class A*

*{*

*A()*

*{*

*// Constructor of Class A, not having any return type.*

*}*

*void A()*

*{*

*// constructor having a return type, It will be treated as method but with a warning.*

*}*

*}*

* Every class should have at least one constructor. If you don’t write constructor for your class, compiler will give default constructor. Default constructor is always public and it has no arguments (No-Arg Constructor).

*class A*

*{*

*// No Constructors written*

*}*

*Compiler will treat the above code as,*

*class A*

*{*

*public A()*

*{*

*//   Constructor provided by the compiler.*

*}*

*}*

* Constructor can be declared as private. If you declare constructor as private, you can’t use it outside that class.

*class A*

*{*

*private A()*

*{*

*// Private Constructor*

*}*

*void methodOne()*

*{*

*//You can use private constructor inside the class*

*A a1 = new A();*

*}*

*}*

*class MainClass*

*{*

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

*{*

*//You can't use private constructor ouside the class like this*

*// A a1 = new A();*

*}*

*}*

* One class can have more than one constructors. It is called **Constructor Overloading.**Through constructor overloading, you can have multiple ways to create objects.

*class A*

*{*

*A()*

*{*

*// First Constructor*

*}*

*A(int i)*

*{*

*// Second Constructor*

*}*

*A(int i, int j)*

*{*

*// Third Constructor*

*}*

*}*

you can create the objects to the above class in three ways like below,

*class MainClass*

*{*

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

*{*

*A a1 = new A();      //Using First Constructor*

*A a2 = new A(10);    // Using Second Constructor*

*A a3 = new A(10, 20);    // Using Third Constructor*

*}*

*}*

* Duplicate Constructors not allowed. If you keep duplicate constructors, you will get compile time error.

*class A*

*{*

*A(int i)*

*{*

*// Duplicate Constructor*

*}*

*A(int i)*

*{*

*// Duplicate Constructor*

*}*

*}*

* Multiple arguments of the constructors can’t have same name. If the two arguments have the same name, you will get compile time error.

*class A*

*{*

*A(int i, int i)*

*{*

*// Duplicate Arguments Passed. It gives compile time error*

*}*

*}*

* Only public, protected and private keywords are allowed before a constructor name. If you keep any other keyword before a constructor name, it gives compile time error.

*class A*

*{*

*final A()*

*{*

*//Constructor can not be final*

*}*

*static A()*

*{*

*//Constructor can not be static*

*}*

*abstract A()*

*{*

*//Constructors can not be abstract*

*}*

*}*

* First statement in a constructor must be either super() or this(). If you put any other statements you will get compile time error.If you don’t include these statements, by default compiler will keep super() calling statement. super() – It is a calling statement to default constructor of super class. this()- it is a calling statement to constructor of the same class.

*class A*

*{*

*A()*

*{*

*//By Default, Compile will keep super() calling statement here.*

*System.out.println("First Constructor");*

*}*

*A(int i)*

*{*

*//Compiler will not keep any statement here*

*super();*

*System.out.println("Second Constructor");*

*}*

*A(int i, int j)*

*{*

*//Compiler will not keep any statement here*

*this();*

*System.out.println("Third Constructor");*

*}*

*A(int i, int j, int k)*

*{*

*System.out.println("Fourth Constructor");*

*// super(); It will give error if you keep super() here*

*}*

*}*

* Recursive constructor calling is not allowed.

*class A*

*{*

*A()*

*{*

*this();*

*// It gives compile time error*

*}*

*}*

* No Cylic calling of constructors.

*class A*

*{*

*A()*

*{*

*this(10);*

*// It gives compile time error*

*}*

*A(int i)*

*{*

*this();*

*// It gives compile time error*

*}*

*}*

**Questions:**

**1) Can you create an object without using new operator in Java?**

Yes, We can create an object without using new operator. There are some other ways to create objects other than using new operator. But, 95% of object creation in java is done through new operator only.

**a) Using newInstance() Method**

|  |  |
| --- | --- |
|  | *Class c = Class.forName("packageName.MyClass");*  *MyClass object = (MyClass) c.newInstance();* |

**b) Using clone() method.**

|  |  |
| --- | --- |
|  | *MyClass object1 = new MyClass();*    *MyClass object2 = object1.clone();* |

**c) Using object deserialization**

|  |  |
| --- | --- |
|  | *ObjectInputStream inStream = new ObjectInputStream(anInputStream );*    *MyClass object = (MyClass) inStream.readObject();* |

**d) Creating string and array objects :**

|  |  |
| --- | --- |
|  | *String s = "string object";*    *int[] a = {1, 2, 3, 4};* |

**2) What is constructor chaining?**

Constructor Chaining is a technique of calling another constructor from one constructor. **this()** is used to call same class constructor where as **super()** is used to call super class constructor.

|  |  |
| --- | --- |
|  | *class SuperClass*  *{*  *public SuperClass(int i)*  *{*  *System.out.println("Super Class Constructor");*  *}*  *}*    *class SubClass extends SuperClass*  *{*  *public SubClass()*  *{*  *this(10);      //Calling same class constructor*  *}*    *public SubClass(int i)*  *{*  *super(i);      //Calling super class constructor*  *}*  *}* |

**3) Can we call sub class constructor from super class constructor?**

No. There is no way in java to call sub class constructor from a super class constructor.

**4) What happens if you keep return type for a constructor?**

It will be treated as a normal method. But compiler gives a warning saying that method has a constructor name.

|  |  |
| --- | --- |
|  | *class MyClass*  *{*  *int MyClass()*  *{*  *return 0;    //No Compile time error but just a warning*  *}*  *}* |

**5) What is No-arg constructor?**

Constructor without arguments is called no-arg constructor. Default constructor in java is always a no-arg constructor.

|  |  |
| --- | --- |
|  | *class MyClass*  *{*  *public MyClass()*  *{*  *//No-arg constructor*  *}*  *}* |

**6) What is the use of private constructor?**

Private constructors are used to restrict the instantiation of a class. When a class needs to prevent other classes from creating it’s objects then private constructors are suitable for that. Objects to the class which has only private constructors can be created within the class. A very good use of private constructor is in singleton pattern. This ensures only one instance of a class exist at any point of time. Here is an example of singleton pattern using private constructor.

|  |  |
| --- | --- |
|  | *class MyClass*  *{*  *private static MyClass object = null;*    *private MyClass()*  *{*  *//private constructor*  *}*    *public MyClass getObject()*  *{*  *if(object == null)*  *{*  *object = new MyClass();   //Creating object using private constructor*  *}*    *return object;*  *}*  *}* |

**7) Can we use this() and super() in a method?**

No, We can’t use this() and super() in a method.

|  |  |
| --- | --- |
|  | *class SuperClass*  *{*  *public SuperClass()*  *{*  *System.out.println("Super Class Constructor");*  *}*  *}*    *class SubClass extends SuperClass*  *{*  *public SubClass()*  *{*  *System.out.println("Sub Class Constructor");*  *}*    *void method()*  *{*  *this();     //Compile time error*  *super();    //Compile time error*  *}*  *}* |

**Difference between Method and Constructor**

|  |  |  |
| --- | --- | --- |
|  | **Method** | **Constructor** |
| 1 | Method can be any user defined name | Constructor must be class name |
| 2 | Method should have return type | It should not have any return type (even void) |
| 3 | Method should be called explicitly either with object reference or class reference | It will be called automatically whenever object is created |
| 4 | Method is not provided by compiler in any case. | The java compiler provides a default constructor if we do not have any constructor. |

### Why overriding is not possible at constructor level. The scope of constructor is within the class so that it is not possible to achieved overriding at constructor level.

### Example of default constructor, parameterized constructor and overloaded constructor

**class** Test{

**int** a, b;

Test (){

System.**out**.println("I am from default Constructor...");

a=1;

b=2;

System.**out**.println("Value of a ="+a);

System.**out**.println("Value of b ="+b);

}

Test (**int** x, **int** y){

System.**out**.println("I am from double Paraceterized Constructor");

a=x;

b=y;

System.**out**.println("Value of a ="+a);

System.**out**.println("Value of b ="+b);

}

Test (**int** x){

System.**out**.println("I am from single Parameterized Constructor");

a=x;

b=x;

System.**out**.println("Value of a ="+a);

System.**out**.println("Value of b ="+b);

}

Test (Test T){

System.**out**.println("I am from Object Parameterized Constructor...");

a=T.a;

b=T.b;

System.**out**.println("Value of a ="+a);

System.**out**.println("Value of b ="+b);

}

};

**class** TestDemo2{

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

Test t1=**new** Test ();

Test t2=**new** Test (10, 20);

Test t3=**new** Test (1000);

Test t4=**new** Test (t1);

}

};

**Note** By default the parameter passing mechanism is call by reference.

#### Constructor Overloading

Like methods, a constructor can also be overloaded. Overloaded constructors are differentiated on the basis of their type of parameters or number of parameters. Constructor overloading is not much different than method overloading. In case of method overloading you have multiple methods with same name but different signature, whereas in Constructor overloading you have multiple constructor with different signature but only difference is that Constructor doesn't have return type in Java.

#### Q. Why do we Overload constructors ?

Constuctor overloading is done to construct object in different ways.

#### Example of constructor overloading

class Cricketer

{

String name;

String team;

int age;

Cricketer () **//default constructor**.

{

name ="";

team ="";

age = 0;

}

Cricketer(String n, String t, int a) **//constructor overloaded**

{

name = n;

team = t;

age = a;

}

Cricketer (Cricketer ckt) **//constructor similar to copy constructor of c++**

{

name = ckt.name;

team = ckt.team;

age = ckt.age;

}

public String toString()

{

return "this is " + name + " of "+team;

}

}

Class test:

{

public static void main (String[] args)

{

Cricketer c1 = new Cricketer();

Cricketer c2 = new Cricketer("sachin", "India", 32);

Cricketer c3 = new Cricketer(c2 );

System.out.println(c2);

System.out.println(c3);

c1.name = "Virat";

c1.team= "India";

c1.age = 32;

System .out. print in (c1);

}

}

**output:**

this is sachin of india

this is sachin of india

this is virat of india

#### Q What's the difference between constructors and normal methods?

Constructors must have the same name as the class and can not return a value. They are only called once while regular methods could be called many times and it can return a value or can be void.

#### Q. What is constructor chaining in Java?

Constructor chaining is a phenomena of calling one constructor from another constructor of same class. Since constructor can only be called from another constructor in Java, constructor chaining is used for this purpose.

class Test

{

Test()

{

this(10);

}

Test(int x)

{

System.out.println("x="+x);

}

}

#### Q. Does constructors return any value?

Yes, constructors return current instant of a class. But yet constructor signature cannot have any return type.

**Note:**

*If a Class defines an explicit Constructor, it no longer has a default Constructor to set the state of the Objects. If such a Class requires a default Constructor, its implementation must be provided. Any attempt to call the default Constructor will be a compile time error if an explicit default Constructor is not provided in such a case.*

**Constructor Chaining:**

Every Constructor calls its Superclass constructor. **An implied *super()* is therefore included in each constructor which does not include either *this()* or an *explicit super()*call as its first statement.** The *super()* statement invokes a constructor of the super class.

The implicit super() can be replaced by an explicit super(). The super statement must be the first statement of the Constructor. The explicit super allows parameter values to be passed to the Constructor of its superclass and must have matching parameter types. A super() call in the constructor of a subclass will result in the call of the relevant constructor from the superclass, based on the signature of the call. This is called constructor chaining.

**super() or this() construct**: If used in constructor, must occur as the first statement in a constructor, and it can only be used in a constructor declaration. This implies that *this()*and *super()* calls cannot both occur in the same constructor. Just as the *this()* construct leads to chaining of constructors in the same class, the *super()* construct leads to chaining of subclass constructors to Superclass Constructors. If a constructor has neither a *this()*nor a *super()* construct as its first statement, then a *super()* call to the default constructor in the superclass is inserted.

*If a class only defines non-default constructors, then its subclasses will not include an implicit super() call. This will be flagged as a compile-time error. The subclasses must then explicitly call a Superclass constructor, using the super() construct with the right arguments to match the appropriate Constructor of the Superclass.*

**Calling a Constructor from another Constructor**

A constructor may call another constructor of the same class. Let's consider the following Test class. It declares two constructors; one accepts no parameters and one accepts an int parameter.

We must use the keyword this to call a constructor from another constructor.

The following code calls the constructor with an int parameter from the constructor with no parameter using the statement, "this(1);".

**class** Test {

Test() {

this(1); // OK. Note the **use** of the keyword this.

}

Test(int x) {

}

}

If a constructor calls another constructor, it must be the first executable statement in the constructor's body.

A constructor cannot call itself because it will result in a recursive call.

**return from a Constructor**

A constructor cannot have a return type in its declaration.

We can use a return statement without a return expression inside a constructor body.

When a return statement in a constructor is executed, the control returns to the caller, ignoring the rest of the constructor's code.

The following code shows an example of using a return statement in a constructor.

***class*** *Test {*

***public*** *Test(****int*** *x) {*

***if*** *(x < 0) {*

***return****;*

*}*

*System.out.println(****"here"****);*

*}*

*}*

**Access Level Modifier for a Constructor**

Access level for a constructor determines the program that can use that constructor in an object creation expression.

We can specify one of the four access levels for a constructor: public, private, protected, and package-level.

The following code declares four constructors for the Test class.

*// Class Test has* ***public*** *access level*

***public******class*** *Test {*

*// Constructor #1 - Package-level access*

*Test() {*

*}*

*// Constructor #2 -* ***public*** *access level*

***public*** *Test(int x) {*

*}*

*// Constructor #3 -* ***private*** *access level*

***private*** *Test(int x, int y) {*

*}*

*// Constructor #4 -* ***protected*** *access level*

***protected*** *Test(int x, int y, int z) {*

*}*

*}*

A constructor with a public access level can be used in any part of the program.

A constructor with private access level can be used only inside the same class in which it is declared.

A constructor with protected access level can be used in the program with the same package in which its class is declared and inside any descendant class in any package.

A constructor with package-level access can be used inside the same package in which its class is declared.

**Default Constructor**

Java compiler adds a constructor to class if we do not declare one.

The constructor that is added by the compiler is called the default constructor.

The default constructor does not have any parameters.

The default constructor is also called a no-args constructor.

If the class already had a constructor, the compiler does not add any constructor.