Chapter - 4

CONSTRUCTORS

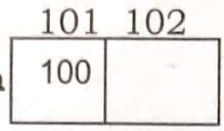
CONSTRUCTORS

Constructors

In general, when a variable is created we want to initialize with a value.

Example1:

int a=100;



When **a** is created it is initialized with **100**.

It is also achieved by assigning value to a variable.

Example2:

int a; ⟶ **a** is created with no value.

a=100; ⟶ **a** is assigned with 100.

From the above two examples we prefer the first method of initializing the values to variables when they are created. In programming it is always recommended that variables created should be initialized with some values.

Similarly, in Java we create objects therefore we want to initialize objects with initial values and it can be done in java using **Constructors**.

Constructors are used to initialize the instance variables of objects with constant values or resources. The resources may be memory, files etc..

Constructors are the special methods of class which have same name as that of classname. Constructors does not have return type because they can't return values. Constructors are automatically executed when an object is instantiated.

Syntax:

```
class classname
{
        public classname( )
        {
                :
                : // code of constructor
        }
}
```

```
classname object;
object=new classname();                 // calls constructor
classname object=new classname();       // calls constructor
```

A program to demonstrate the constructors execution.

Bin>edit cons1.java

```
class A
{
        public A( )
        {
            System.out.print("\n Constructor called..");
        }
}
class cons1
{
    public static void main(String argv[ ])
    {
          A obj1;                  // only obj1 reference variable
                                   // is created not the object
          obj1=new A();            // object is created therefore
                                   // calls constructor
          A obj2=new A();          // object is created therefore
                                   // calls constructor
    }
}
```

Bin>javac cons1.java

Bin>java cons1

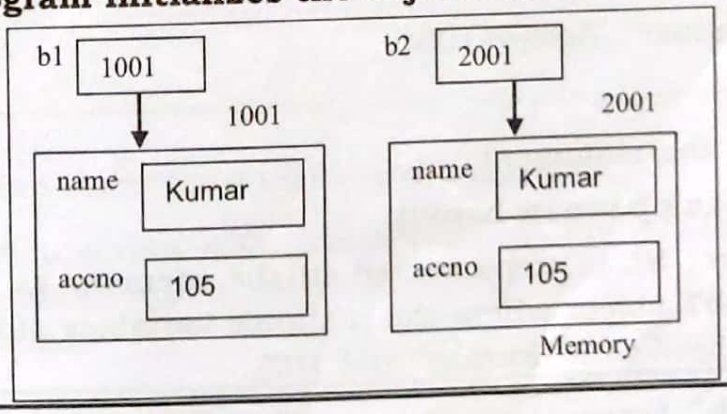
Constructor called..

Constructor called..

Explanation:

In the above program, in the main(); at the statement **obj1=new A();** the **new** operator allocates memory of **A** class (object of **A** is created) as a result it calls constructor as shown in the above program. Similarly, constructor is called for the statement **A obj2=new A();**

The following program initializes the objects with values using constructors.



Bin>edit cons2.java

```
class bank
{
    private String name;
    private int accno;
    public bank() //constructor
    {
          name="kumar";
          accno=105;
    }
    public void showbank()
    {
         System.out.print("\nName="+name+"\tAccno="+accno);
    }
}
class cons2
{
      public static void main(String agrv[])
      {
          bank b1=new bank();
          bank b2=new bank();
          System.out.print("\n b1....");
          b1.showbank();
          System.out.print("\n b2....");
          b2.showbank();
      }
}
```

Bin>javac cons2.java
Bin>java cons2

```
b1 ....
Name=kumar   Accno=105
b2 ....
Name=kumar   Accno=105
```

Explanation:

In the main, at the statement

bank b1=new bank();

The object **b1** is instantiated in the memory as a result it calls constructor for **b1** object where the instance variables of **b1** i.e **name** and **accno** are initialized with "**kumar**" and **105**.

Similarly, when **b2** is instantiated the constructor is called and **b2** object is initialized with "**kumar**" and **105**.

Therefore **b1** and **b2** objects are initialized with same values as it is shown in memory.

Default Constructor

Constructor with zero parameters is known as **default constructor**.

Example:

```
class A
{
        A()   // no parameters.
              // default constructor
        {
          :
        }
}
```

Parameterized Constructors

In the above program, the objects **b1** and **b2** are initialized with same values. But we may want different objects to be initialized with different values. This can be achieved by using parameterized constructors.

To the constructors arguments can be passed therefore constructors can have parameters. A constructor with parameters is known as **parameterized constructor**.

Syntax:

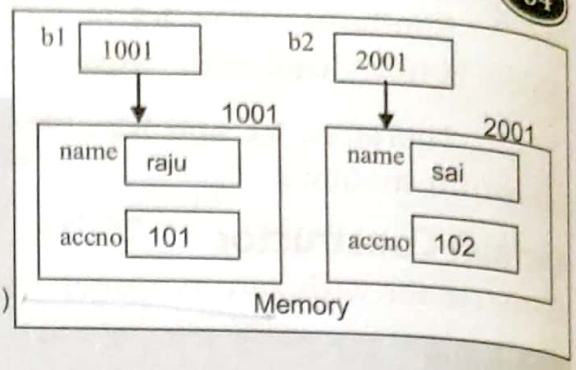
```
class classname
{
        public classname(type para1, type para2, ...)
        {
                :
                :  // code of constructor
        }
}
```

classname object=new classname(arg1,arg2, ..);

To the constructors arguments (values) are passed from object definition. The constructor is called explicitly using the classname and arguments are passed.

Example:

Bin>edit cons3.java




```
class bank
{
    private String name;
    private int accno;
    public bank(String s, int a)
    {
        name=s;
        accno=a;
    }
    public void showbank()
    {
        System.out.print("\n Name = "+name);
        System.out.print("\t Accno= "+accno);
    }
}
class cons3
{
    public static void main(String argv[])
    {
        bank b1=new bank("raju",101);
        bank b2=new bank("sai",102);
        System.out.print("\n b1 values…");
        b1.showbank();
        System.out.print("\n b2 values…");
        b2.showbank();
    }
}
```

Bin>javac cons3.java

Bin>java cons3

b1 values...
Name=raju Accno=101
b2 values...
Name=sai Accno=102

Explanation:

In the above program, in the main(), the statement

bank b1=new bank("raju", 101);

Calls the constructor with two-parameters and "**raju**" is passed to **s** and **101** to **a**. These values are then copied into **name** and **accno** of **b1** object as shown in memory.

Similarly, The statement

bank b2=new bank("sai", 102);

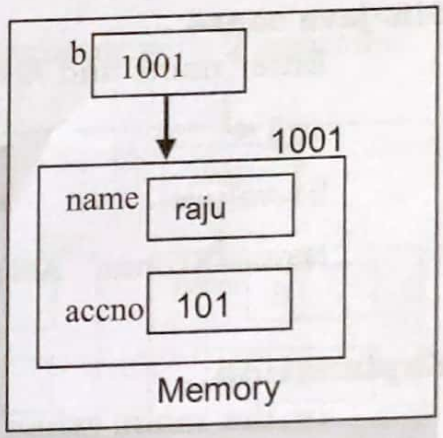
Calls the constructor with two-arguments. This time "**sai**", **102** are passed to **s** and **a**. These values are then copied into **name** and **accno** of **b2** object as shown in memory.

The output can be seen that different objects are initialized with different values and this is achieved using parameterized constructors.

Dynamic initialization using constructors

The above program is initializing the objects at compile time. Therefore the program may be executed for any number of times, in all cases the objects are initialized with same values because the values are given in the program directly which does not change at runtime.

But, at different executions the objects should be initialized with different values. This can be achieved by reading values into objects at Runtime.



Example:

Bin>edit cons4.java

```
import java.io.*;
class bank
{
     private String name;
     private int accno;
     public bank()throws IOException
     {
       InputStreamReader in = new InputStreamReader(System.in);
       BufferedReader br=new BufferedReader(in);
           try
           {
                 System.out.print("\nEnter name,accno :");
                 name=br.readLine();
                 accno=Integer.parseInt(br.readLine());
           }
           catch(NumberFormatException e)
           {
```

```
            System.out.print("\n invalid input.");
        }
    }
    public void showbank()
    {
      System.out.print("\n Name="+name+"\t Accno="+accno);
    }

}
class cons4
{
    public static void main(String argv[]) throws IOExceptio
      {
      bank b1=new bank();   // calls default constructor
          System.out.print("\n b1.values..");
          b1.showbank();
    }
}
```

Bin> javac cons4.java
Bin>java cons4

Enter name and Accno: Kumar ↵

105 ↵

b1.values..

Name=Kumar Accno=105

Explanation:

In the main, when **b1** object is created the default constructor is called where it reads values from keyboard and stores into **name** and **accno**. The output can be seen above.

Constructor overloading

A class can have any number of constructors provided they should differ either in number of arguments or data types of arguments. In this way a class can be overloaded with any number of constructors. A class containing two or more constructors is known as **constructor overloading**.

Syntax:

```
class classname
{
        public classname()
        {
                : // constructor with 0 argument
        }
        public classname(type para1)
        {
                : // constructor with 1 argument
        }
        public classname(type para1, type para2)
        {
                : // constructor with 2 arguments
        }
}
```

classname object=new classname(); // calls constructor with 0 argument

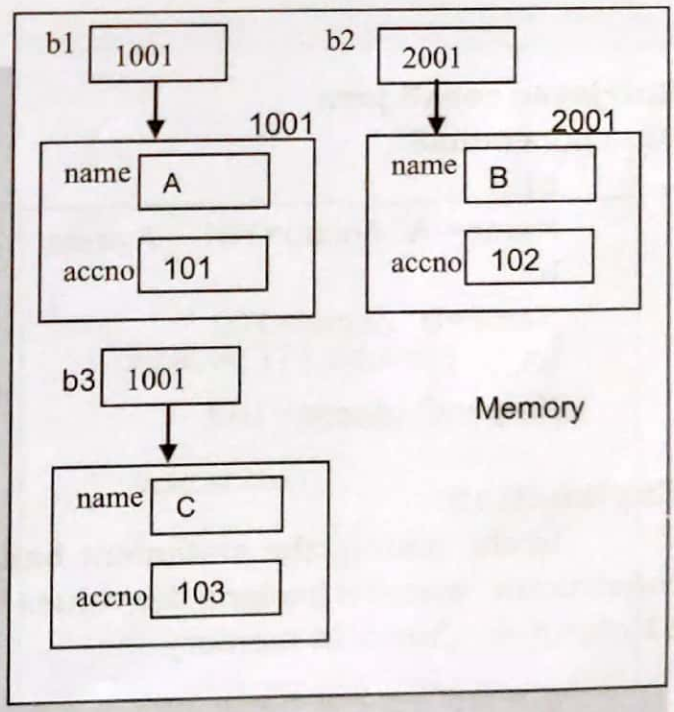
classname object=new classname(arg1);
// calls constructor with 1 argument

classname object=new classname(arg1, arg2);
// calls constructor with 2 argument

Example:

Bin>edit cons5.java

```
class bank
{
    private String name;
    private int accno;
    public bank()
    {
          name= "A";
          accno=101;
    }
    public bank(String s)
    {
          name= s;
          accno=102;
    }
    public bank(String s, int a)
    {
```



```
            name= s;
            accno=a;
    }
    public void showbank()
    {
       System.out.print("\nName="+name+"\tAccno="+accno);
    }
}
class cons5
{
    public static void main(String argv[ ] )
    {
            bank b1=new bank();
            bank b2=new bank("B");
            bank b3=new bank("C", 103);
            System.out.print("\n b1…");
            b1.showbank();
            System.out.print("\n b2… ");
            b2.showbank();
            System.out.print("\n b3… ");
            b3.showbank();
    }
}
```

Bin>javac cons5.java
Bin> java cons5

```
b1...
Name= A  Accno=101
b2...
Name=B  Accno=102
b3...
Name=C  Accno=103
```

Explanation:

In the main(), the statement **bank b1=new bank( );** calls the default constructor, where it assigns the values **"A"** and **101** into **name** and **accno** of **b1** object as shown in memory.

The statement **bank b2=new bank("B");** calls the constructor with one parameters where **"B"** is passed to **s** and then the constructor assigns **s** into **name** and **102** into **accno** of **b2** object as shown in memory.

The statement **bank b3=new bank("C",103);** calls the constructor with two parameters where it passes **"C"** and **103** to **s** and **a,** these values of **s** and **a** are then assinged into **name** and **accno** of **b3** object.

Finally,

```
b1.showbank()   //displays the values of b1.
b2.showbank()   //displays the values of b2.
b3.showbank()   //displays the values of b3.
```

Dummy Constructor

The rule in OOP is that for each object there should be a constructor in the class. When an object is instantiated it is compulsory to execute the constructor. If there is no matching constructor for the object then the compiler generates error.

A class should contain minimum one constructor and can contain any number of constructors. If no constructor is defined in the class then the java runtime system(jvm) adds a **dummy constructor** in the class. **Dummy constructor** is a default constructor which is an empty constructor i.e which does not have statements and does not perform any operation.

Syntax:

```
public classname( ) { }
```

Example:

```
class A
{
   //no constructors

      void show()
      {
         :
      }
}
```

JVM

```
class A
{
      public A( ) { }  //dummy
                       //constructor added
       void show()
       {
          :
       }
}
```

Example:

Bin>edit cons6.java

```
class A
{
        public void show()
    {
            System.out.print("\nNo constructor in the class");
    }
}
class cons6
{
        public static void main(String argv[])
        {
            A obj=new A(); //calls dummy constructor
            obj.show();
        }
}
```

Bin>javac cons6.java
Bin>java cons6
No constructor in the class

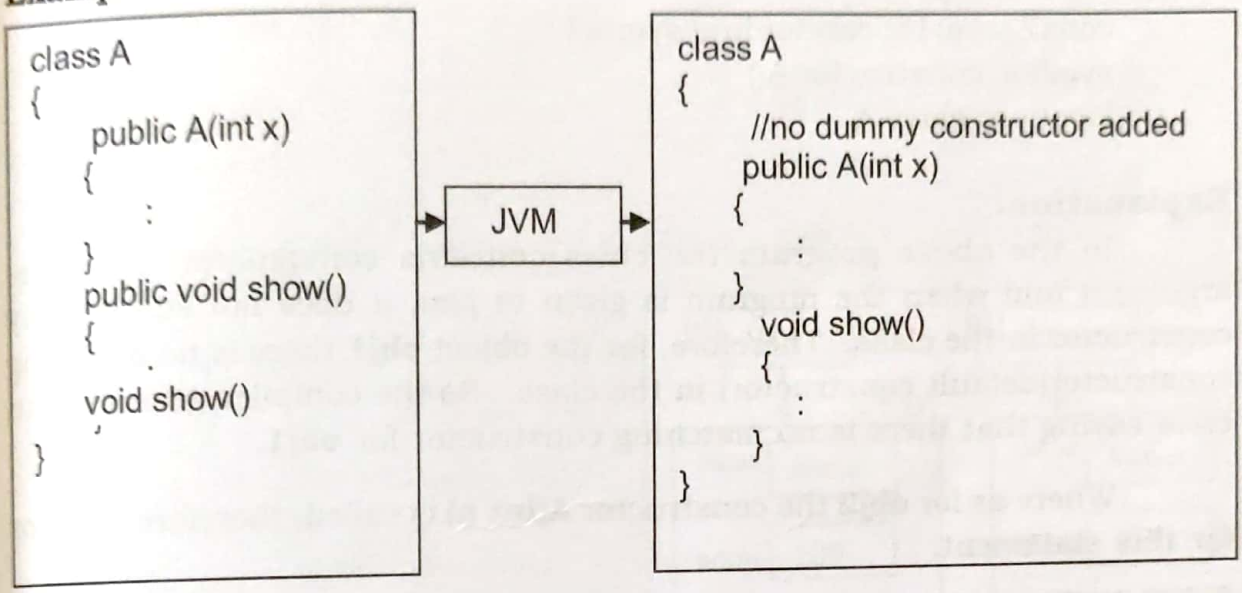
Explanation:

In the above program, class **A** does not contain any constructor. If this program is given to jvm then JVM (Java runtime system) adds a dummy constructor in the class and we get the code as shown below.

```
class A
{
    public A(){ } // dummy constructor added in the class
    void show()
    {
        System.out.print("\nNo constructor in the class");
    }
}
class cons6
{
        public static void main(String argv[])
        {
            A obj=new A(); //calls dummy constructor
            obj.show();
        }
}
```

If the class contains minimum one constructor then the Java does not add a dummy constructor in the class.

Example:



Example:

Bin>edit cons7.java

```
class A
{
    public A(int x)
    {
        System.out.print("\n Constructor with 1-arg..x="+x);
    }
    public void show()
    {
        System.out.print("\nNo constructor in the class");
    }
}
class cons7
{
    public static void main(String argv[])
    {
        A obj1=new A(); //Error: calls a default constructor
                        //which is not present in the class
        A obj2=new A(100);
        obj1.show();
```

```
            obj2.show();
        }
    }
```

Bin>javac cons7.java

```
cons7.java:16: cannot find symbol
symbol: constructor A()
location: class A
```

Explanation:

In the above program the class contains constructor with one-argument and when the program is given to jvm, it does not add dummy constructor in the class. Therefore, for the object **obj1** there is no matching constructor(default constructor) in the class. So the compiler generates an error saying that there is no matching constructor for **obj1**.

Where as for **obj2** the constructor **A(int x)** is called, therefore no error for this statement.

Points to Remember:

1. If a class does not contain default constructor and containing constructors with arguments, it is recommended that we should write a default constructor (dummy constructor) in the class so that the above problem can be solved.

For example, execute the above program by adding a dummy constructor shown below in the **class A** and the program executes successfully.

```
public A() {}
```

Copy Constructor

A constructor can have parameters of same class and such constructor is known as **copy constructor**. Copy constructor is used to copy (or) initialize values of one object into another object.

Syntax:

```
class classname
{
        public classname(classname ob)
        {
                // code of copy constructor
        }
}
```

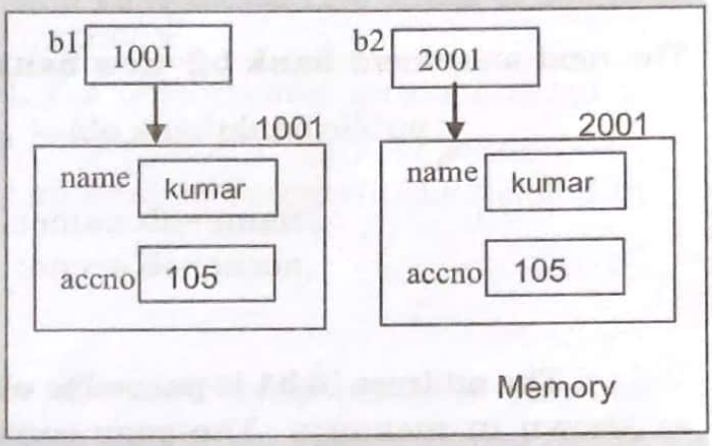
```
classname obj1=new classname();
classname obj2=new classname(obj1); //calls copy constructor
```

Example:

Bin>edit cons8.java

```
class bank
{
    private String name;
    private int accno;
    public bank(){ }
    public void setbank(String s, int a)
    {
          name=s;
          accno=a;
    }
    public bank(bank ob)
    {
          name=ob.name;
          accno=ob.accno;
    }
    public void showbank()
    {
        System.out.print("\nName="+name+"\tAccno="+accno);
    }
}
class cons8
{
    public static void main(String argv[])
    {
          bank b1=new bank();  // calls dummy constructor
          b1.setbank("kumar",105);
          bank b2=new bank(b1); // calls copy constructor
          System.out.print("\n b1 values…");
          b1.showbank();
          System.out.print("\n b2 values…");

          b2.showbank();
    }
}
```



Bin>javac cons8.java

```
Bin>java cons8
        b1 values...
        Name=kumar   Accno=105
        b2 values...
        Name=kumar   Accno=105
```

Explanation:

In the main, the statement **bank b1=new bank();** creates the object **b1** and calls the dummy constructor. The statement **b1.setbank("kumar",105);** calls the **setbank(String s,int a)** in the class where it passes "**kumar**" and **105** to **s** and **a** parameters. These values are then assigned to **name** and **accno** of **b1** object.

The next statement **bank b2=new bank(b1);** calls the copy constructor

```
public bank(bank ob)
{
        name=ob.name;
        accno=ob.accno;
}
```

The address in **b1** is passed to **ob** i,e. **ob** is also pointing to object of **b1** as shown in memory. The copy constructor copies the values of **ob** i,e. **name("kumar")** and **accno(105)** into **name** and **accno** of implicit object **b2** as shown in memory

The statements **b1.showbank()** and **b2.showbank()** displays the values of **b1** and **b2** objects as shown in the ouput.

Points to remember:

1) Copy constructors are used to copy or initialize the members of one object to another object.

Memories in Java

Memories in java are of two types. They are:

1. Stack memory
2. Heap memory

1. Static memory

1. Any memory that is allocated at compile time is created on stack memory.

2. All the primitive type variables (byte, short, char, int, long, float, double, boolean) and reference variables of objects, arrays memory is allocated on stack.
3. For each method separate stack memory is created in RAM.
4. When the method is terminated all variables of method created on the stack memory are deleted.

2. Heap memory

1. Any memory that is allocated at runtime is created on heap memory.
2. The objects and arrays memory is created on heap.
3. Heap memory is common for all methods.
4. When the method is terminated, the object's and arrays created in the method are not deleted from heap.
5. The objects and arrays memory on heap is automatically deleted by the garbage collector.

Garbage Collection

In java, memories of objects are allocated dynamically at runtime using **new** operator. This dynamically allocated objects memories can not be deleted manually by the programmer in Java. In java the memories of objects are automatically deleted by garbage collector. *The garbage collector is a special program which is a part of JVM that is automatically executed by jvm and comes into the heap memory where it deletes all unwanted/waste memories of objects and arrays.* The unwanted/waste objects/arrays are identified if the object /array does not have any reference variables on stack.

This process of de-allocation of memory by the garbage collector is called as **Garbage Collection**. The garbage collector is not simply executed if one or more waste objects exist in heap. The garbage collector is executed periodically at any time and its execution is under the control of JVM. The different java-run-time(JVM) implementations have different approaches to garbage collection. Therefore we need not to think about the memory de-allocation while writing the program as it is handled by the JVM.

Advantage of garbage collector

1. No wastage of memory:

The waste objects memories are automatically deleted by garbage collector.

Dis-Advantage of garbage collector

1. Loss of resources:

If the object is maintaining resources from outside the object then such resources are also lost when the object is deleted by garbage collector. The resources may be a file, font, etc.

To overcome the dis-advantage of garbage collector we can use **finalize()** method.

The finalize() method

When the objects are deleted by the garbage collector, we want to perform some operation for the object to save the resources and this can be achieved by using **finalize()** method. The finalize() whose prototype is predefined in java and this method is automatically executed for the object by the jvm before the object is deleted by the garbage collector. Therefore we can write the code in the **finalize()** method to save the resources of object before it is deleted.

The **finalize()** overcomes the dis-advantage of garbage collection.

Syntax:

```
protected void finalize()
{
        : //code of finalize method
}
```

Example:

Bin>edit finalize1.java

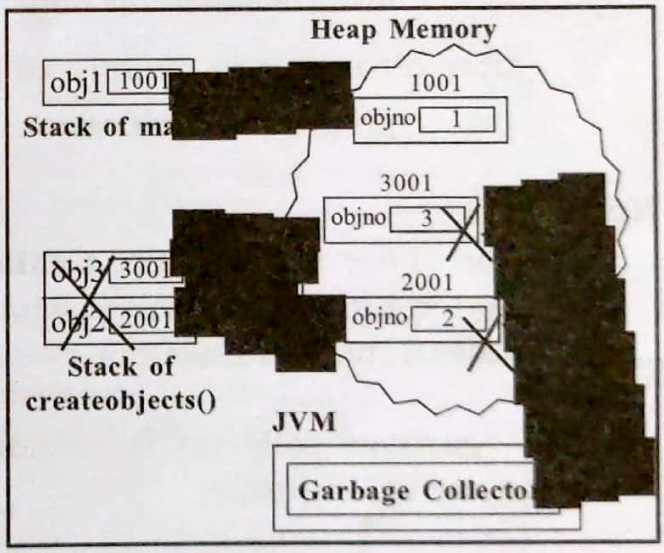
```
class myclass
{
    private int objno;
    public myclass(int n)
    {
        objno=n;
        System.out.print("\nExecuting constructor
                                    of obj"+objno);
    }
    protected void finalize()
    {
        System.out.print("\nfinalize called for obj"+objno);
    }
}
```

```
class finalize1
{
    public static void main(String argv[])
    {
        myclass obj1=new myclass(1);
        createobjects();
        //System.gc(); //calls garbage collector explicitly
        System.out.print("program terminates...");
    }

    public static void createobjects()
    {
        System.out.print("\ncreateobjects() started...");
        myclass obj2=new myclass(2);
        myclass obj3=new myclass(3);
        System.out.print("\ncreateobjects() terminates...");
    }
}
```

Bin> javac finalize1.java
Bin> java finalize1

Executing constructor of obj1
createobjects() started...
Executing constructor of obj2
Executing constructor of obj3
createobjects() terminates...
finalize called for obj3
finalize called for obj2
program terminates...



Explanation:

When the main() starts its execution the JVM creates Stack memory for it. At the statement **myclass obj1=new myclass(1);** the **obj1** reference variable creates on stack and new operator creates object of myclass on heap and whose address is assigned to **obj1** on stack as shown in memory. When **obj1** object is created, it calls the constructor and **1** is passed to **n**. The constructor assigns **n** to **objno** of **obj1** and prints the message

Executing constructor of **obj1**

The statement **createobjects();** in main() calls the **public static viod createobjects(){ .. }** and before its execution the JVM create statck memory for the createobjects() as shown in memory. In the createobjects() **obj2** and **obj3** objects are instantiated similar to **obj1** as shown in above memory and prints the messages.

```
createobjects() started...
Executing constructor of obj2
Executing constructor of obj3
createobjects() terminates...
```

When the **createobjects()** terminates the stack memory along with **obj2** and **obj3** reference variables is deleted from RAM where as the objects(**2001** and **3001**) present in the heap are not deleted. At this time **if garbage collector is executed** then it comes into heap where it finds and deletes the waste objects (**2001**,**3001**) because these objects does not have any reference variables on stack, but before deletion of objects the jvm calls **finalize()** method for each object and prints the messages.

```
finalize called for obj3
finalize called for obj2
```

After returning to main(), we get the message

```
program terminates...
```

Note:

1. In the above program the **finalize()** method is executed if the garbage collector program executes while execution of the program otherwise **finalize()** method does not execute.

2. The garbage collector is executed implicitly by the jvm and it may execute at any time.

3. We can also call the garbage collector explicitly using the method **System.gc();**

4. In the above program, to see the execution of **finalize()** method we can add the statement **System.gc();** in the main() as shown below.

```
myclass obj1=new myclass(1);
createobjects();
System.gc();
System.out.print("program terminates...");
```