08/12/18

Unit-II

⇒ methods

A method represents a group of statements that can be used to perform a task. a task represents, calculation or processing of data.

→ A method have two parts-

① method header (or) method prototype:-

A method header or prototype contains a method name, parameter and a method return type. A method header can be declare as:-

```
returntype methodname(parameter1, parameter2)
```

in the above statement the method name represents the name given to a method, after declaring a method name, we write some variable in the simple braces, this variables are known as parameters. The method parameter can be useful from receiving the data outside into a method. The return data type before a methods indicates the type of a result the method is return. By using the above statement we can declare a method header or method prototype as:-

eg:- int sum (int a, int b)

In the above example Sum is the name of the method, a and b are parameters of a method and the return type of this method is int.

(ii) method body :-

A method body contains a group of statements which contains some logic to perform a task, after declaring the name of the method we can write the method body in b/w the opening and the closing braces.

→ This can be represented as:-

```
Void show()
{
    c=a+b;
    System.out.println(c)
}
```

→ method body

⇒ Constructor

The main purpose of Constructor is to initialize the variables. a Constructor ~~be~~ can have the following characteristics—

(i) A Constructor name and the class name must be the same and the constructor name should be ended with a pair of simple braces.

(ii) A Constructor does not return any value

(iii) A Constructor is automatically called and executed at the time of creating an object.

(iv) while creating an object to a Constructor if nothing is passed the default constructor will be executed

(v) A Constructor is called and executed only once for an object.

*Types of Constructor:-

There are two types of constructors, they are:-

① Default Constructor:-

A constructor without parameters is known as a default constructor. This can be represented as:-

```
class person
{
  ---
  ---
person()
{
  ---
}
}
```

in the above code we create a class with a name person and by using the name of the class we will create a constructor person and the person constructor does not have any argument. so this type of constructor can be known as default constructor.

10/12/2018

program eg:-

```
class person
{
  String name;
  int age;
  person()
  {
    name = "Raju";
    age = 18;
  }
void talk()
{
```

```
    System.out.println("Hello I am:"+name);
    System.out.println("my age is:"+age);
  }
}
class demo
{
  Public static void main(string args[])
  {
    person Raju=new person();
    Raju.talk();
    person sita=new person();
    sita.talk();
  }
}
```

o/p:
```
Hello I am: Raju
my age is:18
Hello I am: Raju
my age is:18
```

② parameterized constructor:-

A constructor ~~with~~ with some parameters is known as parameterized constructor. This can be represented as:-

```
~~person(strings)~~
person(strings s, int (i))
{
  =
  =
}
```

program:- eg:-

```
class person
{
  string name;
  int age;
  person()
  {
   name= "Raju";
   age =18;
  }
  person (string s, int i)
  {
   name=s;
   age=i;
  }
  void talk()
  {
   System.out.println("Hello I am:"+name);
   System.out.println("my age is:"+age);
  }
}
class demo
{
  Public static void main (string args[])
  {
   person Raju= new person();
     Raju.talk();
   person sita= new person("sita", 20);
     sita.talk();
  }
}
```

o/p:-

```
Hello i am: Raju
my age is: 18
hello i am: sita
my age is: 20
```

⇒ Constructor over loading :-

writting two (or) more constructor with the same name but different parameters is known as constructor over loading.

The syntax for creating a constructer overloading is as:-

Syntax:-

```
class classname
{
  class name()
  {
    =
  }
  class name(arg1, arg2)
  {
    =
    =
  }
  class class name
  {
    class name(arg1, arg2, arg3)
    {
      =
      =
    }
  }
```

by using the above syntax we can represents a Constructor Overloading as:-

```
class ab
{
  ab()
  {
```

```
    }
    ab(int a, int b)
    {
     =
    }
    ab(int a, int b, int c)
    {
     =
    }
  }
```

in the above code we declare a class ab, in the class we declare three constructors with a name ab. The first constructor does not contains any parameter. In the second we pass two ~~constructor~~ parameter a, b and in third constructor we pass ~~three~~ three parameter a, b & c. we can identify which constructor at the time of execution depending upon the number of parameters that can be passed to a constructor.

programs

eg:-

```
class ab
{
  ab()
  {
    System.out.println("default constructor");
  }
  ab(int x)
  {
    System.out.println("circle:"+2*math.pi*x);
  }
```

```
ob(int x, int y)
{
System.out.println("rect:"+2*(x+y));
}
}
class overload
{
  Public static void main (string args[])
  {
    ob s1 = new ob();
    ob s2 = new ob(10);
    ob s3 = new ob(10,20);
  }
}
```

o/p:

default Constructor
circle: 62.80000000000004
rect: 60

11/12/018

⇒ Garbage collector :-

The objects are created by using the new operator in C++. The memory allocated to the object can be released by deleting the object with the help of delete operator.

In Java the memory allocated to an object is automatically released. This concept is known as garbage collector.

In garbage collector after creating an object the JVM will create a reference to an object. as long as the programmer is working with the object the reference object is stay in JVM. once the programmer has stopped

working with the object. The reference of that object is deleted by JVM. after deleting the reference object, the memory that is allocated the object is automatically realeased by JVM.

Note:-

as long as you're working with the object the memory of the object is not released by JVM.

⇒ Arrays:-

An array represents a group of elements of same data type (or) an array can store a group of homogeneous data types means we can store a group of int values and a group of float values, but it is not possible to store some int values & some float values in an array.

* Types of array:-

An array can be classified into two parts:-

① Single [] array:-

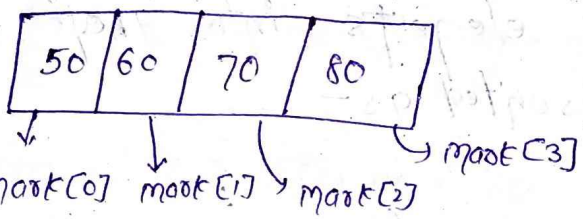
a. Single[] arrays represents a single row (or) a single coloumn of a elements.

→ creating a single[] array:- we can create a single[] array intwo ways-

① we can create a single[] arrays by directly storing the elements into an array. This can be represented as-

```
int marks[] = {50,60,70,80};
```

in the above statement int represents an integer type of element which can be stored into an array name as marks to represents a single[] arrays, we should use [] after the array name. The elements of an array can be represented inside the curly braces. By considering the above statement the JVM will creates four blocks of memory for storing the above element in an array. The array initialization is starting from zero. The element in an array block can be represented as:-



(ii) another way of for creating for a single[] array as, first declare the array and then allowed the memory to store the elements in the array by using a new operator. This can be represented as:-

```
int marks[] = new int[5];
```

program:-
eg:-

```
class Array
{
 Public static void main (string args[])
 {
  int marks[] = {50,60,70,80,90};
  for (int i=0; i<5; i++)
  {
   System.out.println(marks[i]);
  }
 }
}
```

Output:- 20
30
40
50
60

14/12/18

② multidimensional (or) Two-[] array:-

A Two dimensional array can be represented in the form of several rows and columns.

→ Creating a two-dimensional array:-

we can create 2-[] array in two ways

① we can declare a 2-[] array and directly storing the elements into that array. This can be represented as-

```
int marks[][]={{50, 60, 70}, {80, 90, 100}, {10, 20, 30}};
```

in the above statement we create a 2-[] array with a name marks and we use two square brackets before the variable name for representing a 2-[] arrays where as first [] represents the row of an element in an array and the second[] represents the colomn an element in an array.

The arrangement of an elements in a 2-[] array can be represented as-

	j=0	j=1	j=2
i=0	50	60	70
i=1	80	90	100
i=2	10	20	30

in the above diagram we create a 2-[] array with three rows and three colomns. The Total size of an array is 3x3=9 blocks. The Jvm will allocate sufficient memory for storing the element into the blocks.

(ii) another way for creating a 2-[] array is—

```
int marks[][] = new int[3][3]
```

program

eg: program:—

```
class Array2
{
  Public static void main(String args[])
  {
   int marks[][] ={{50,60,70},{80,90,100},{10,20,30}};
   for(int i=0; i<3; i++)
   {
   for(int j=0; j<3; j++)
   {
   System.out.println(marks[i][j]+" ");
   }
  }System.out.println();
  }
}
```

o/p:-

```
50 60 70
80 90 100
10 20 30
```

⇒ Command line arguments

When we are passing input of a program at the time of running the program is known as command line arguments.

eg: program:-

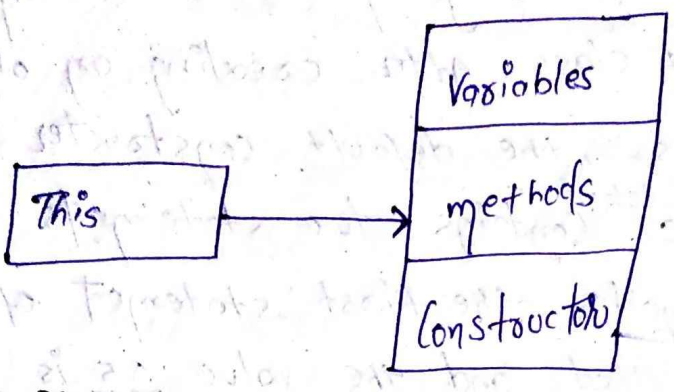
```
class prog
{
  Public static void main (string args[])
  {
    int n = args.length;
    system.out.println ("no. of arguments:"+n);
    System.out.println("the arguments are:");
    for(int i=0; i<n; i++)
    {
      System.out.println(args[i]);
    }
  }
}
```

o/p:-

```
Javac prog.Java
Java prog 10 20 30
no. of arguments : 3
The arguments are: 10
                   20
                   30
```

⇒ This keyword

In a class we declare some variables, methods and constructor. By using the this keyword we can refer the current class variables, methods and constructor. This can be represented as:-



15/12/18 Program:-

```
eg: class Sample
    {
      int x;
      Sample()
      {
        this(55);
        this.accessc();
      }
    Sample (int x)
    {
      this.x = x;
    }
    void accessc()
    {
      System.out.println("x=" +x);
    }
}
class demo
{
  Public static void main(String args[])
  {
    Sample s=new Sample();
  }
}
```

o/p:- x=55

in the above program we create an object to a sample class, After creating an object to a sample class, the default constructer is executed. The DC (default) contains two statements among those two statements, the first statement of the constructor is called and the value 55 is passed to a parameterized Constructor, in that parameterized constructor this.x = x represents the present class for calling access we can use this.x access which displays the value of x=55 on output.

⇒ Importance of static keywood

*Static Block:- static block is a block of statements that can we declare by using the keywood static. The syntax of creating a static Block is:-

Syntax:-

```
static
{
  statements;
}
```

eg: program:-

```
class test
{
 static
 {
  System.out.println("static Block");
 }
 Public static Void main(string args[])
 {
  System.out.println("main Block");
} }
```

o/p:- static Block
main Block

we know that the JVM is executed, the statements that are return in the main method:-

(1) if static block is not present in the program. if static block is present in the program then the JVM first executed the static block statement after that it will execute the main block statements.

18/12/2018

* static method :- A static method is declared by using the keywood static and the static method can be called by using :- classname.methodname();

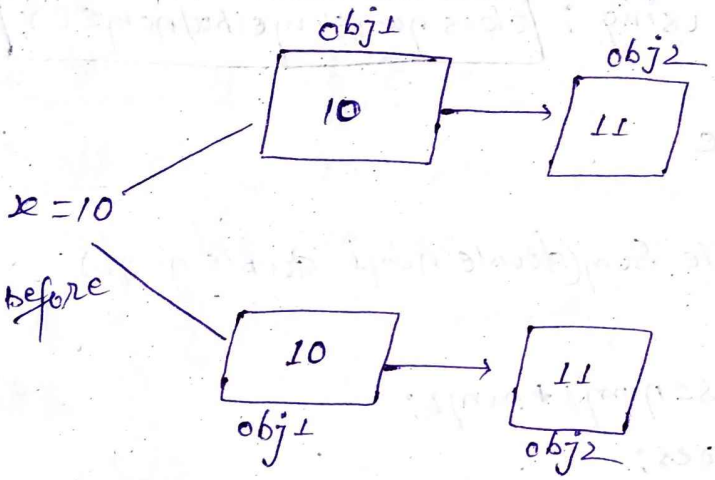
program:-

eg:-
```
class Sample
{
    static double Sum(double num1, double num2)
    {
        double res=num1+num2;
        return res;
    }
}
class As
{
    Public static Void main (string arg[])
    {
        double x = Sample.Sum (10.0, 22.5);
        System.out.println("Sum ="+x);
    }
}
```

o/p:- Sum = 32.5

⇒ class Variables (or) static Variables :—

The class variables can also known as static variables. The static variables are declared by using keyboard static. The difference b/w an instance variable & static variable is an instance variables is a variable whose seperate copy is available to each object. A static variable is a variable whose single copy in a memory is shared by all the objects. This can be represented as :—



eg: program

```
class Test
{
  static int x=10;
  static void display()
  {
    System.out.println(x);
  }
}
class demo
{
  Public static void main(string args[])
```

```
{
  Test obj1, obj2;
  Test obj1 = new Test();
  Test obj2 = new Test();
    ++ obj1.x;
  System.out.println("x in obj1:"+ obj1.x);
  System.out.println("x in obj2:"+ obj2.x);
  }
}
```

O/P:

```
10
x in obj1:11
x in obj2:11
```

⇒ Nested class

A Java programming language allows you to define a class with another class is known as a nested class (or) an inner class. A nested class can be represented as:-

```
class outer
{
  ==
  ==
  ==
  class ss
  {
    ==
    ==
  }
}
```

in the above code the class with a name ss is known as a nested class (or) an inner class.