Constructor

Constructor is a special type of methods in java that enables an object to initialize itself when it is created. Constructors have the same name as the class itself. They do not specify a return type, not even void. This is because they return the instance of the class itself.

Example:

Program to illustrate the use of a constructor method to initialize an object at the time of creation

```
Class Rectangle
{
        int length, width;
        Rectangle( int x, int y)                    // Defining constructor
        {
                length=x;
                width=y;
        }
        int rectArea()
        {
                return(length*width);
        }
}
Class RectangleArea
{
        Public static void main (String args[])
        {
                Rectangle rect1= new Rectangle (15, 10);          // calling constructor
                int area1=rect1.rectArea();
                System.out.println( "Area=   ", + area1);
        }
}
```

Output
Area= 150

Methods Overloading

In java it is possible to create methods that have the same name, but different parameter lists and different definitions. This is called method overloading. Method overloading is used when objects are required to perform similar tasks but using different input parameters.

When we call a method in an object, Java matches up the method name first and then the number and type of parameters to decide which one of the definitions to execute. This process is known as polymorphism.

To create an overloaded method, all we have to do is to provide several different method definitions in the class, all with the same name, but with different

parameter lists. The difference may be in number or type of arguments. Example of creating an overloaded method:

```
Class Room
{
        float length;
        float breadth;
        Room( float x, float y)                         // constructor 1
        {
                length=x;
                breadth=y;
        }
        Room( float x)                          // constructor 2
        {
                length= breadth=x;
        }
        int area()
        {
                return (length*breadth);
        }
}
```

Static Members

The member that is common to all the objects and accessed without using a particular object can be defined as:

```
static int count;
static int max( int x, int y);
```

The members that are declared **static** as shown above are called static members.

Static variables are used when we want to have a variable common to all instances of a class. Like static variables, static methods can be called without using the objects. They are also available for use by other classes.

Example: Defining and using static members

```
Class Mathoperation
{
        static float mult( float x, float y)
        {
                return x*y;
        }
        static float divide( float x, float y)
        {
                return x/y;
```

```
        }
}
Class Mathapplication
{
        public void static main ( string args[])
        {
                float a= Mathoperation.mult( 4.0,5.0);
                float b= Mathoperation.divide(a,2.0);
                System.out.println( "b= " +b);
        }
}
```

OUTPUT

b=10.0

Restrictions on static members

- They can only call other static methods
- They can only access static data
- They cannot refer to **this** or **super** in any way

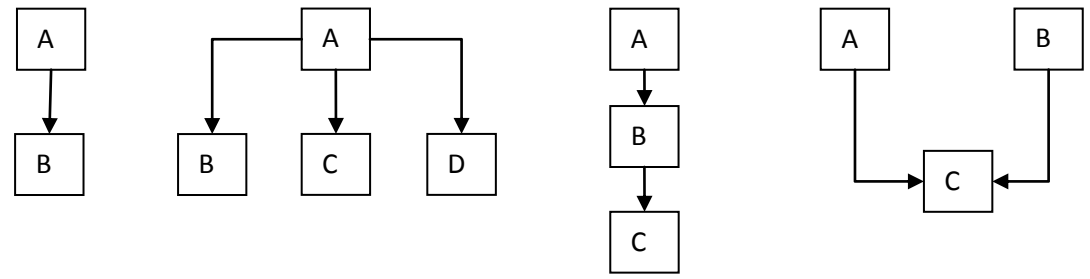
Inheritance

The mechanism of deriving a new class from an old one is called inheritance. The old class is known as the base class or super class or perent class and the new one is called the subclass or derived class or child class.

The inheritance allows subclasses to inherit all the variables and methods of their parent classes. The different forms of inheritance are as follows:

- Single inheritance (only one super class)
- Multiple inheritance (Several super class)
- Hierarchical inheritance (one super class many subclasses)
- Multilevel inheritance (derived from a derived class)

These forms of inheritance are shown below:



Single Hierarchical Multilevel Multiple

Defining a subclass

A subclass is defined as follows:

*Class subclassname **extends** superclass_name*
{
    Variables declaration;
    Methods declarations
}

The keyword **extends** signifies that the properties of the superclass_name are extended to the subclassname.

The subclass will now contain its own variables and methods as well as those of the superclass. This kind of situation occurs when we want to add some more properties to an existing class without actually modifying it.

HOME WORK: Example of Application of Single inheritance