2.16 Create classes and objects

Java is a true object oriented language and therefore underlying structure of all java programs is classes.

Class: A Class is a user-defined data type. A class contains variables and methods.
Classes create objects and objects use methods to communicate between them.

Creation of class/Defining a class: A class is declared by using class keyword. The general format of class definition is:

```
class classname
{
        [fields declaration;]
        [Methods declaration;]
 }
```

Everything inside the square brackets is optional.

Fields/Variables Declaration:

- The variables defined with in a class are called **instance variables.** These variables are called instance variables because they are created whenever an object of the class is instantiated.
- We declare instance variables exactly the same way as we declare local variables.
- Instance variables declared in the form of

Syntax: type instancevariablename;

Method Declaration:

- The code is contained within methods. A method manipulates the data stored in the fields of an object of the class. The general form of method declaration is

```
syntax:     returntype  methodname([parameter list])
             {
             method body;
              }
```

- The methods and variables defined with in a class are called **members of the class.**

```
Ex:     class Sample
        {
            int  length, width;
            void  getdata(int a ,int b)
            {
                length=a;
                width=b;
            }
        }
```

Creation of objects:

- **Object:** An Object is an instance of a class. (or) An object is a runtime entity. Creating an object is also called as **instantiating an object**.

Creation of objects:-To use a class, we should create an object to the class.

- Object creation represents allocating memory necessary to store the actual data of the variables. Objects in java are created using new operator.
- The new operator creates an object of the specified class and returns a reference to the object.

Syntax: **classname objectname = new classname();** // both declaration and initialization in a single statement.

Ex: Rectangle rect1 = new Rectangle(); ;

(or)

```
classname      objectname;          //declare the object.
objectname = new classname();  //instantiate the object
```

Ex: Rectangle rect1; //declare the object.
rect1= new Rectangle(); //instantiate the object.

2.17 use new operator and methods

Usage of new operator: The new operator allocates memory for an object at run time (dynamically). The general form of new operator is

Syntax: classname objectname = new classname();

Advantages:

1. We can create as many objects as it needs during the execution of your program.
2. No memory wastage.
3. No insufficient memory message (unless total memory is used).

Usage of methods: A class usually contains two things: Instance variables and methods.

The general form of method is

Syntax:
```
returntype methodname(parameter list)
{
    Method body;
}
```

- Method declaration have 4 basic parts
  1. The name of the method(method name)
  2. The type of the value the method returns(type)
  3. A list parameters(parameter list)
  4. The body of the method
- The methods are used to manipulate the data contained in the class.
- Java methods are similar to functions or procedures in other programming languages. Every java program must have one main() method.
- Each method performs a specific task. The data associated with the instance variable is manipulated in the methods.
- The methods may directly output the result or return a value to the calling section.

The general form of calling or accessing a method is

Objectname.methodname(parameter list);

The general form of accessing a variable is

objectname.variablename;

Example program using concept of class,objects and methods:

```
class rectangle
{
        int length,width;
        void getdata( int x,int y)
        {
                length=x;
                width=y;
        }
        int area( )
        {
                int area1=length*width;
                return(area1);
        }
}
class demo
{
        public static void main(String args[])
        {
                int a1,a2,a3;
                rectangle r1=new rectangle();
                rectangle r2=new rectangle();
                r1.width=10;
                r1.length=20;
                a1=r1.width* r1.length;
```

```
                a2=r1.area();
                a3=r2.area();
                System.out.println(a1);
                System.out.println(a2);
                System.out.println(a3);
        }
}
```

2.17 List and explain various types of Constructors

Constructor:

A constructor is a special kind of method .it is similar to a method that is used to initialize variables at the time of creation of an object. The purpose of a constructor is to initialize instance variables.

Properties of Constructor:

1**. The constructor's name and class name should be same.** And the constructor's name should end with a pair of simple braces.

For e.g: - in Student class, we can write a constructor as;

```
Student( )
{
}
```

2. **A constructor may have or may not have parameters**. Parameters are variables to receive data from outside into the constructor. If a constructor does not have any parameters, it is called „default constructor".

if a constructor has 1 or more parameters, it is called „parameterized constructor." For example, we can write a default constructor as:

```
Student()
{
}
```

And a parameterized constructor with two parameters , as:

```
Student(String name, int id)
{
}
```

3. **A constructor does not have return-type not even void**.

4. **A constructor is automatically called and executed at the time of creating an object**. For example, if we create the object as:

```
Student s=new Student(); //calling default constructor
Student s=new Student("qwerty", 123); //calling parameterized constructor
```

5 .**A constructor is called and executed only one per object**. This means when we create an object, the constructor is called. When we create second object, again the constructor is called second time.A constructor is a special method which is used to initialize an object when it is created.

- Constructors of base class cannot be inherited to derived class.
- Constructors cannot be private.

Types of Constructor: Two types constructors are available in java.

1. Default constructor.
2. Parameterized constructor.

Default constructor: The constructor which never takes any parameter is called default constructor. If no constructors are defined for a class, the java system automatically generates the default constructor.

syntax:

```
class   classname
{
        classname( ) //default constructor
        {
            Block of statements;
            ……;
            ……;
         }
          ……;
```

```
          ……….……;
}

import java.io.*;

class Student
{
   int num;
    String name;

    Student()
    {
     System.out.println("Constructor called");
    }
}

class StudentCon
{
    public static void main (String[] args)
    {

        Student s1 = new Student();
        System.out.println(s1.name="roopi");
        System.out.println(s1.num=49);
    }
}
```

Parameterized constructor: A constructor with parameters [arguments] is called as parameterized constructor. If any constructors are defined by a class with the parameters, java will not create default constructor for the class.

Syntax:

```
class     classname
{
          classname(list of parameters) //parameterized constructor
          {
                    Block of statements (s);
          }
           ……….……;
          ……….……;
}
```

Example:

```
import java.io.*;

class Student
{

   String name1;
   int id1;
Student(String name, int id)
   {
      name1 = name;
      id1 = id;
System.out.println(name1);
System.out.println(id1);
   }
}

class StudentPara
{
   public static void main (String[] args)
   {
```

```
    Student  s1=new Student("roopi",001);
        }
}
```

What is differences between a constructor and a method?

Constructor	Method
1. A constructor is **used to initialize the instance variables of a class**.	1. A method is **used for any general purpose processing or calculations**.
2. A **constructor's name and class name should be same**.	2. A **method's name and class name can be same or different**.
3. A constructor is **called at the time of creating the object.**	3. A method can be **called after creating the object.**
4. A constructor is **called only one per object.**	4. A method can be **called several times on the object.**
5. A constructor **is called and executed automatically.**	5. A method **is executed only when we call it.**

What is constructor overloading? Writing two or more **constructors with the same name with difference in the parameters is called constructor overloading**. Such constructors are useful to perform different tasks.

Example Program:

```
class Test
{
        int x,y;
        Test()                  // default constructor
        {
                System.out.println("This is default constructor");
                x=10;
                y=20;
                System.out.println("val of x is"+x);
                System.out.println("val of y is"+y);
        }

        Test(int p,int q)               //parameterized constructor
         {
                System.out.println("This is parameterized constructor");
                 x=p;
                 y=q;
                System.out.println("val of x is"+x);
                System.out.println("val of y is"+y);
          }
}
class Testex
{
        public static void main(String args[])
         {
                  Test t1=new Test();
                  Test t2=new Test(35,65);
         }
}
```

Compiling and running:

```
javac Testex.java
java Testex
```

2.18 Explain Method OverLoading:

Method OverLoading:

Writing two or more methods in the same class in such a way that each method has same name but different method signatures.

Example: void add(int a,int b)
void add(float a,float b)

Program:

```
class sample
{
void add(int a,int b)
{
        System.out.println(" sum=" +(a+b));
}
void add(int a,int b.int c)
{
        System.out.println(" sum=" +(a+b+c));
}

class demo
{
        public static void main(String args[])
        {
                Sample s=new sample();
                s.add(10,15);
                s.add(10,15,10);
        }
}
```

JVM executes methods based on method signatures

2.20 Use of 'this' pointer

'this' keyword:

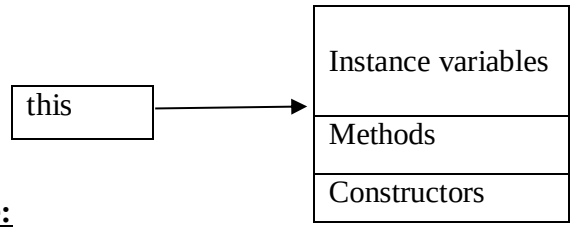
- **"this"** is a keyword that refers to the object of the class where it is used. In other words,
- **this** refers to the object of the present class. All the members of a class are referenced by **this**.
- When an object is created to a class a by default reference is also created internally to the object. This default reference is nothing but "this".

'this' is created by JVM for two purposes.

- To point to current class object
- Whenever formal parameters and data members of a class are similar, there is a possibility of getting ambiguity by JVM. In order to differentiate between formal parameters and data members of the class, the data members of the class must be preceded by a keyword 'this'.

Syntax :

this.currentclass datamember name



Example:

```
class Test
{
   int a;
   int b;

   // Parameterized constructor
   Test(int a, int b)
   {
      this.a = a;
```

```
        this.b = b;
    }

    void display()
    {
        //Displaying value of variables a and b
        System.out.println("a = " + a + "  b = " + b);
    }

    public static void main(String[] args)
    {
        Test object = new Test(10, 20);
        object.display();
    }
}
```

Output: a=10 b=20

Program 2:

```
class sample
{
        int x;
        sample( )
        {
                this(55);  //calls present class parameterized constructor
                this.access( ); //calls present class method
        }
        sample(int x)
        {
                this.x=x;   //refer present class instance variable.
        }
        void access( )
        {
                System.out.println("x= " +x);
        }
}
class ThisDemo
{
        public static void main(String args[ ])
        {
                sample s1=new sample();
        }
}
```

2.21 Explian the working of static and final.

Working of static members:

- static is a keyword used **to create a class member that will be used independently without creating any object of the class**
- To create such a member, precede its declaration with the keyword 'static'. We can create both methods & variables to be static. The most common example of a static member is main(). main() is declared as static because it must be called without any objects.

Syntax:

static variable declaration: **static datatype variablename;**

static method declaration: **static returntype methodname([parameters]);**

- **Methods declared as static have restrictions:**
  1. They can't access non static methods /data(variables) directly
  2. They cannot refer to this.
- If we wish to call a static method from outside its class, we can do so using following general from.

Classname.methodname();

Here, classname is the name of the class in which the static method is declared. A static variable can also be accessed in the same way by use of dot operator.

Program On Static:

```
class UseStatic
{
        static int a=3;
        static int b;
        static void meth(int x)
        {
                System.out.println("x=" +x);
                System.out.println("a=" +a);
                System.out.println("b=" +b);
                System.out.println("static block is initialized");
                b=a*4;
                }
        public static void main(String args[])
        {
                meth(55);
        }
}
```

Example Program(calling static method from outside of a class)

```
class demo
{
  static int x=55;
            static void access()
            {
                        System.out.println("x=   " +x);
            }
}
class demo1
{
            public static void main(String args[])
            {
                        demo.access( );   //classname.methodname( )
}
}
```

Example program using static variables,methods,blocks:

```
class UseStatic
{
static int a=3; static int b;
static void meth(int x)
{
System.out.println("x=" +x); System.out.println("a=" +a); System.out.println("b=" +b);
}
static
{
System.out.println("static block is initialized");
b=a*4;
System.out.println("b=" +b);
}
public static void main(String arg[])
{
meth(55);
}
}
```

Working of final:

In order to make anything as constant we use a keyword **final**. The final keyword can be applied to:

1. Variables
2. Methods
3. Class

final members are declares as follows:

final datatype variablename=value; **//final variable declaration**

final returntype methodname([parameters]); **//final method declaration**

EX:
```
final int SIZE=50;
final void getdata([parameters])
{
      -----
      ----
 }
```

Final Variables:

- if we make any variable as final, **we can't change the value of final variables** (it will be constant)
  eg: final int PI=3.14f ; **Note:** we must initialize a final variable when it is declared.

Final methods :- If we make any method as final, we **can't override it**.

Final class:- If we make any class final, **we can't extend it**.

2.22 Explain string classes and methods .

String classes:

- String represents a sequence of characters.
- In java, **a string is an object of string class**. It is not a character array.
- String class is present in "java.io" package.
- java, strings are class objects and implemented using two classes, namely
  1. String
  2. StringBuffer

Creating strings: There are 2 ways to create strings in java:

1. Declare string type variable & assigning a group of characters to it.

Ex: **String s;**
s="Hello";
(or)
String s="Hello";

In this case JVM creates an object & store the string in that object.

2. We can create an object to String class by allocating memory using 'new' operator.

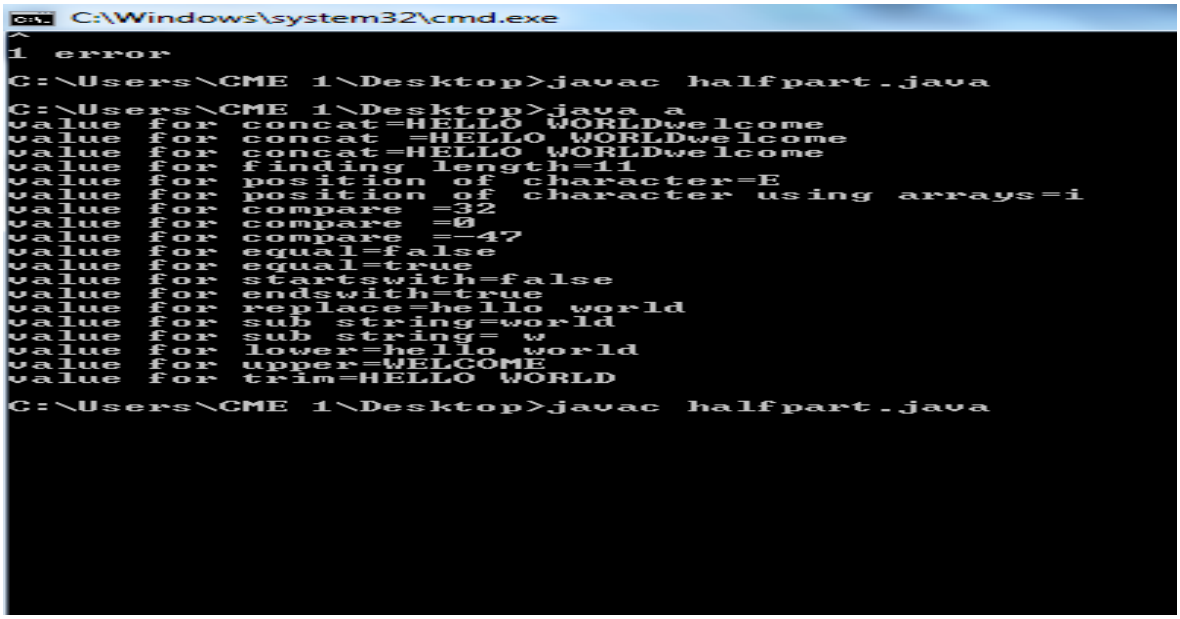
Syntax: **String stringname=new String("string");**
Ex: String s = new String (Hello");

String Methods:

Program:

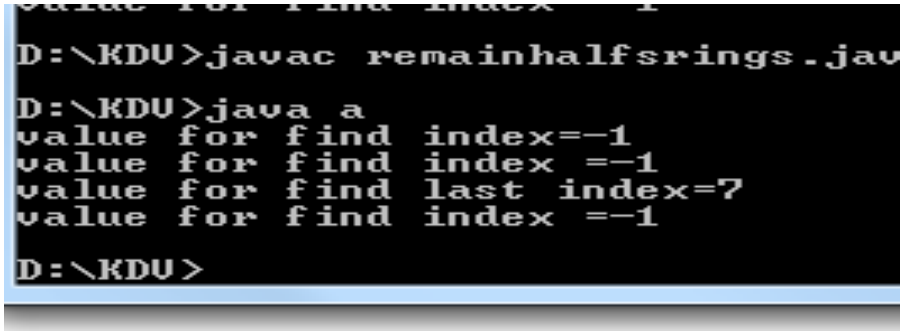
```
class a
{
public static void main(String args[])
{
String s="hello world";
String s1="HELLO WORLD";
String s2="welcome",s3;
char arr[]={'c','h','a','i','r','s'};
//concept on concat
System.out.println("value for concat="+s1.concat("welcome"));
System.out.println("value for concat ="+s1.concat(s2));
s3=s1+s2;
```

```
System.out.println("value for concat="+s3);
//concept on length
System.out.println("value for finding length="+s1.length());
//concept on characters
System.out.println("value for position of character="+s1.charAt(1));
System.out.println("value for position of character using arrays="+arr[3]);
//concept on compareTo
System.out.println("value for compare ="+s.compareTo(s1));
System.out.println("value for compare ="+s.compareToIgnoreCase(s1));
System.out.println("value for compare ="+s1.compareTo(s2));
//concept on equals
System.out.println("value for equal="+s.equals(s1));
System.out.println("value for equal="+s.equalsIgnoreCase(s1));
//concept on startswith,endswith
System.out.println("value for startswith="+s1.startsWith("h"));
System.out.println("value for endswith="+s1.endsWith("D"));
//concept on replace
System.out.println("value for replace="+s.replace('L','y'));
//concept on substring
System.out.println("value for sub string="+s.substring(6));
System.out.println("value for sub string="+s.substring(5,7));
//concept on case sensitive
System.out.println("value for lower="+s.toLowerCase());
System.out.println("value for upper="+s2.toUpperCase());
//concept on trim
System.out.println("value for trim="+s1.trim());
}
}
```




```
class a
{
public static void main(String args[])
{
String s="hello world";
String s1="HELLO WORLD";
String s4="O";
String s2="welcome";
//concept on indexOf
System.out.println("value for find index="+s.indexOf(s4));
System.out.println("value for find index ="+s.indexOf(s1));
```

```
//concept of lastindexof value
System.out.println("value for find last index="+s1.lastIndexOf(s4));
System.out.println("value for find index ="+s2.indexOf(s));
}
}
```



program on String class (Alphabetical ordering of strings)

```
import java.io.*;
classstringsort
{
        static String name[]={"madras","delhi","ahmedabad","Calcutta","Bombay"};
        public static void main(String args[])
        {
                inti,j;
                int size=name.length;
                String temp=null;
                for(i=0;i<size;i++)
                {
                        for(j=i+1;j<size;j++)
                        {
                                if(name[j].compareTo(name[i])<0)
                                {
                                        temp=name[i];
                                        name[i]=name[j];
                                        name[j]=temp;
                                }
                        }
                }
                System.out.println("string after sorting");
                for(i=0;i<size;i++)
                {
                        System.out.println(name[i]);
                }
        }
}
```

String Buffer class:

- StringBuffer is a peer class of **String.** While **String** creates strings of fixed length, **StringBuffer**creates strings of flexible length that can be modified in terms of both length and content.
- We can insert characters and substrings in the middle of a string, or append another string to the end.
- Some of the commonly used string buffer methods are

	Method	task
1.	s1.setCharAt(n,'x')	Modifies the n^{th} character to x
2.	s1.append(s2)	Appends the string s2 to s1 at the end
3.	s1.insert(n,s2)	Insert the string s2 at the position n of the string s1
4.	s1.Length()	It returns the length of the String If n>s1.length() Zeros are added at the end of s1 5
5.	s1.reverse()	It reverses the given String .

6. s1. delete(int startIndex, int endIndex) is used to delete the string from specified startIndex and endIndex.

//program on StringBuffer class (Manipulation of strings)

```
class stringmanipulation
{
        public static void main(String args[])
        {
                StringBuffer str1=new StringBuffer("object oriented");
                StringBuffer str2=new StringBuffer("programming language");
                //getting string length
                System.out.println("length of string1 is:"+str1.length());
                //modifying the character
                str1.setCharAt(6,'-');
                System.out.println("modified string:"+str1);
                //Inserting a string
                str1.insert(15,str2);
                System.out.println("after inserting string is:"+str1);
                //Appending a string
                str1.append("example is java");
                System.out.println("After appending:"+str1);
                System.out.println("in reverse=" +str1.reverse());
        }}
```

```
C:\Users\CME 1\Desktop>javac srt.java

C:\Users\CME 1\Desktop>java stringmanipulation
length of string1 is:15
modified string:object-oriented
after inserting string is:object-orientedprogramming language
After appending:object-orientedprogramming languageexample is java
in reverse=avaj si elpmaxeegaugnal gnimmargorpdetneiro-tcejbo
```

2.23 Use of command line arguments:

Command line arguments **represents the values passed to main() method**. To catch and to store these values, main() has a parameter, **String args[]** as:

```
Public static void main(String args[])
```

Here, args[] is a 1-dimensional array of String type. So it can store a group of Strings, passed to main() form outside by the user. The user should pass the values from outside , at the time of running the program at command prompt as:

```
C:/>java prog 11 22 xyz
```

These 3-values are automatically stored in main() method's agrs [] in the form of Strings. This is because , args[] is String type array. Thus 11 is stored as a string in args[0] ,22 as a string in args[1] and xyz is stored in args[2].

Program on command line arguments:

```
class ExCommanLine
{
        public static void main(String args[ ])
        {
                int i;
                for(i=0;i< args.length;i++)
                System.out.println("java is "+args[i]);
         }
}
```

Execution:

```
C:/>Java  ExCommandLine  simple robust object-oriented
```

UNIT-3

KNOW THE CONCEPTS OF INHERITANCE ,PACKAGES AND INTERFACES

3.1 LIST THE TYPES OF INHERITANCE

Inheritance:

- The process of deriving a new class from an already existing class is called "inheritance".

Types of inheritance:

Inheritance is categorized into the following types

1. Single Inheritance
2. Multiple Inheritance
3. Hierarchical Inheritance
4. Multilevel Inheritance
5. Hybrid Inheritance

3. 2 Implement of inheritance

Inheritance:

•The process of deriving a new class from an already existing class is called "inheritance".

•In object oriented programming the concept of inheritance provides the idea of reusability. This means we can add additional features to an existing class without modifying it.

•We can derive a new class from existing class and add new features to it.

•The new class will have the combined features of both these classes

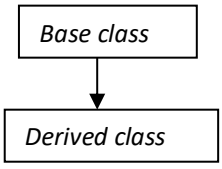
•The old class is known as base class (or) parent class (or) super class.

•The new class is called as derived class (or) child class (or) sub class.

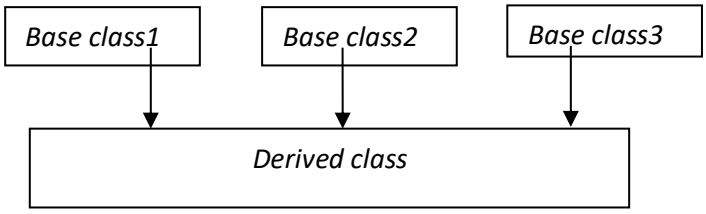
Types of inheritance:

Inheritance is categorized into the following types

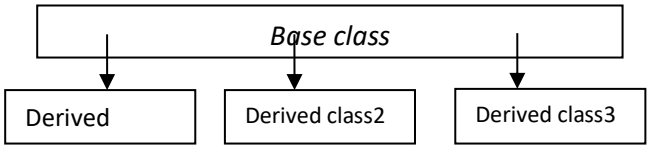
1. **Single Inheritance:** The process of deriving a new class from only one base class is called single inheritance.



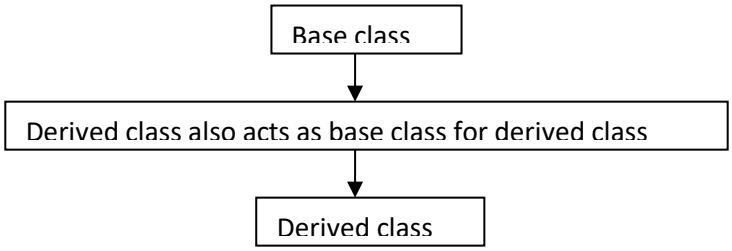
2. **Multiple Inheritance**: The process of deriving a new class from more than one base class is called multiple inheritance.



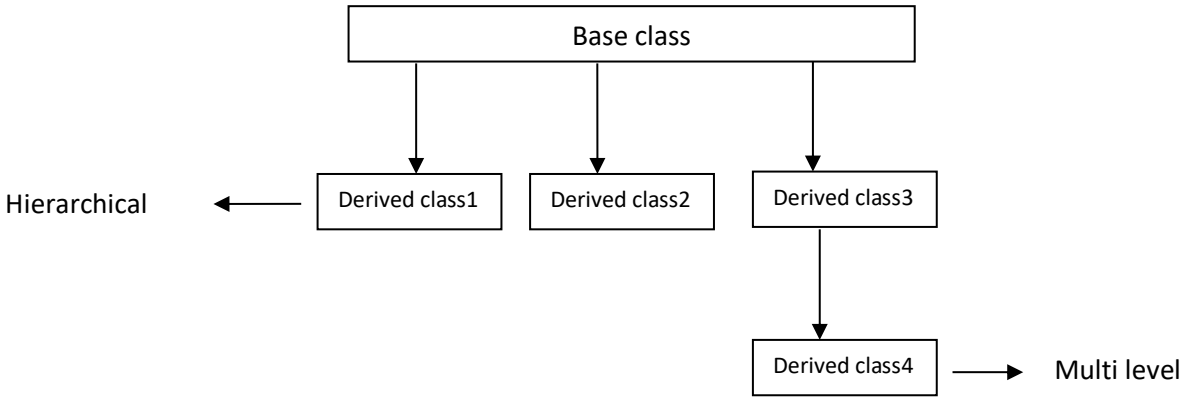
3.**Hierarchical Inheritance**: The process of deriving more than one derived classes from the same base class is called hierarchical inheritance.



4.**Multilevel Inheritance:** The process of deriving a new class from already derived class is called multilevel inheritance.



Hybrid Inheritance: The process of deriving a new class involving more than one form of inheritance is called as hybrid inheritance.



Syntax for inheritance: The general form of class declaration that inherits a super class is as follows:

```
Syntax:   class  subclassname extends superclassname
          {
                Variable declaration;
                Method declaration;
          }
Ex:       class B extends A
          {
              ------
              ------
          }
```

The keyword 'extends' tells that the properties of super class are extended into the sub class. The sub class will contain own variables and methods as well as variables and methods of super class.

//program on single inheritance:

```
class A
{
        int a,b;
        void msg()
        {
                System.out.println("the value of a="+a +"and b="+b);
        }
}
class B extends A
{
        int c;

        void sum()
        {
          C=a+b;
                System.out.println("sum of a,b="+c);
        }
```

```
}
class SingleInheritance
{
        public static void main(String args[])
        {
                B obj=new B();
                obj.a=10;
                obj.b=20;
                obj.msg();
                obj.sum()
        }
}
```

Super: The usage of keyword "super" in inheritance:

•Whenever we inherit the base class features into derived class, there is a possibility that base class features are similar to derived class features. So that there is a possibility of getting confusion by the JVM.

•In order to differentiate the base class features with derived class features, the base class features must be preceded by a keyword 'super' in the context of derived class.

•Super keyword can be used in three places:

1) super can be used to refer to super class variables, as:

super.variable;

2) super can be used to refer to super class methods, as:

super.method();

3) super can be used to refer to super class constructor , as:

super(values);

Super( ) must always be first statement executed inside a sub class constructor

Example Program:

```
class One
{
int i=10;
void show()
{
System.out.println("Super class method,i:"+i);
}
}
class Two extends One
{
int i=20;
void show()
{
System.out.println("sub class method,i="+i);
super.show();     //calling super class show
System.out.println("Super class method,i:"+super.i);
}
}
class Ex
{
public static void main(String args[])
{
Two t=new Two();
t.show();
}
}
```

3.3 Create multilevel hierarchy

Multi – level Hierarchy:

In simple inheritance a subclass or derived class derives the properties from its parent class, but in multilevel inheritance a subclass is derived from a derived class. One class inherits only single class. Therefore, in multilevel inheritance, every time ladder increases by one. The lower most class will have the properties of all the super classes'.

It is common that a class is derived from another derived class. The class student serves as a base class for the derived class student1, which in turn serves as a base class for the derived class student 2..

The chain is known as inheritance path. When this type of situation occurs, each subclass inherits all of the features found in all of its super classes. In this case, percentage inherits all aspects of marks and student.

program on Multi level inheritance:

```
class Student
{
        int pin;
        Student(int p)
        {
                pin=p;
        }
        void display()
        {
                System.out.println("pinno="+pin);
        }
}
class Student1 extends Student
{
        int marks1,marks2,marks3;
        Student1(int p,int m1,int m2,int m3)
        {
                super(p);
                marks1=m1;
                marks2=m2;
                marks3=m3;
        }
        void display()
        {
                super.display();
                System.out.println("marks1="+marks1+"marks2="+marks2+"marks3="+marks3);
        }
}

class Student2 extends Student1
{
        int marks4,marks5,marks6;
        Student2(int p,int m1,int m2,int m3, int m4,int m5,int m6)
        {

                super(p,m1,m2,m3);
                marks4=m4;
                marks5=m5;
                marks6=m6;
        }
        void display()
        {
                super.display();
                System.out.println("marks4="+marks4+"marks5="+marks5+"marks6="+marks6);
```

```
	}
}
class Student3
{
	public static void main(String args[])
	{
		Student2 s2=new Student2(239,88,86,84,82,80,78);
		s2.display();
	}
}
```

3.4 Use 'final' to avoid overriding.

Method overriding: A method in a subclass has the same name, same arguments and same return type as a method in the super class, then the method in the subclass is said to be override the method in the super class.

(OR)

The process of redefining the original method into various derived classes for performing various operations is known as method overriding

Use 'final' to avoid overriding.

- All methods and variables can be overridden by default in sub class. If you want to prevent the subclass from overriding the member of the super class, we can declare them as final using the keywords 'final'.
- A method that is declared as final cannot be overridden in a subclass method.

Ex:

```
class A
{

	int a=10;
	final void display()
	{
		System.out.println("this is super class");
		System.out.println("value of a is" +a);

	}
}
class B extends A
{
	int b=20;
	void display()	// compile time error (final method cannot be overridden)
	{
		System.out.println("this is sub class");
		System.out.println("value of b is" +b);


	}
}
class override
{
	public static void main(String args[])
	{
		B obj=new B();
		obj.display();	// Error
	}	}
```