Unit:II Inheritance – Super classes- sub classes –Protected members – constructors in sub classes- the Object class – abstract classes and methods- final methods and classes – Interfaces – defining an interface, implementing interface, differences between classes and interfaces and extending interfaces - Object cloning -inner classes, Array Lists - Strings

2.1.Inheritance

- Reusability is major concept of OOP paradigm
- Java programs will be reused in different places
- Create new programs by make us of old programs

Definition:

- The process of deriving a new class from an old program is called inheritance.
- Old class of java iscalled as base class or super class or parent class and the new c;as of java is called as subclass/derived class/child class.

Types of Inheritance:

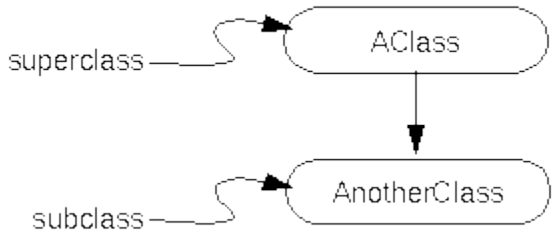
Inheritance can be of any one following types

1. Single inheritance (Only one superclass)
2. Multiple inheritance (Several super classes)
3. Hierarchical inheritance (One superclass, many subclasses)
4. Multilevel inheritance(Derived from a derived class)
5. Hybrid Inheritance(Combination of two inheritances)

Multiple inheritance cannot be used directly in java. This concept is implemented by interface concepts in java

2.1.1.Defining a superclass:

The sub class (the class that is derived from another class) is called a *derived class*. The class from which it's derived is called the *base class or super class*. The following figure illustrates these two types of classes:



2.1.2.Defining a subclass:

Subclass is a class which is formed newly

Syntax for defining a subclass:

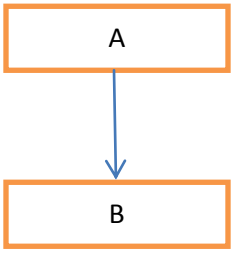
```
Class subclassname extends superclassname
{
        Variable declaration;
        Method declaration;
}
```

Subclassname-subclass or derived class or child class name
Superclassname-superclass or base class or parent class name

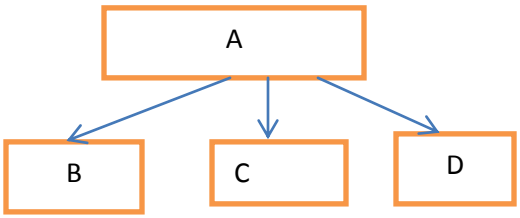
Properties of baseclass are extensive to subclass name.

Pictorial representation of inheritance

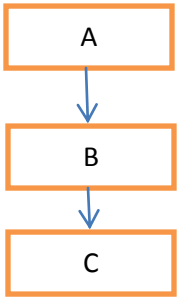
(a)Single inheritance



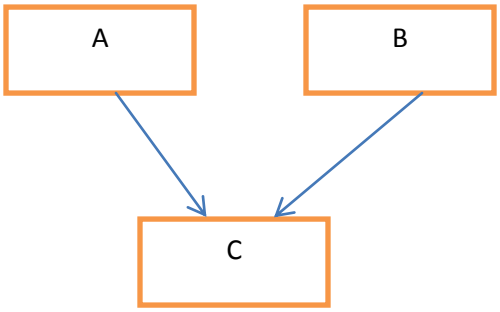
(b)Hierarchical Inheritance



(c)Multilevel inheritance



(d)Multiple inheritance



Single inheritance

The method of inheriting the properties from one super class to one sub class is called single inheritance.
It consists of one base class and one derived class.

Example program1: Single inheritance

```
class Room                          //base class
{
intlength,breadth;
Room(intx,int y)
{
length=x;
breadth=y;
}
int area()
{
return(length*breadth);
}
}
class Bedroom extends Room     //derived class using base class named Room
{
int height;
```

```
Bedroom(intx,inty, int z)
{
super(x,y);
height=z;
}
int volume()
{
Return(length*breadth*height);
}
}

classsingleinheritance
{
public static void main(String ars[])
{
Bedroom room1=new Bedroom(14,12,10);
int area1=room1.area();
int volume1=room1.volume();
System.out.println("Area1="+area1);
System.out.println("Volume1="+volume1);
}
}
```

Output:

```
Area1=168
Volume1=1680
```

Example program2:Single inheritance

```
class A
{
int x;
int y;
int get(int p, int q)
{
x=p;
y=q;
return(0);
}
void Show()
{
System.out.println(x);
}
}

class B extends A
{
public static void main(String args[])
{
B a = new B();
a.get(5,6);
```

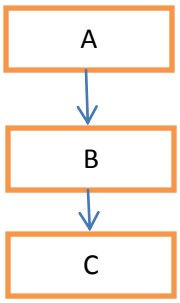
```
a.Show();
}
}
```

Output:

5

Multilevel Inheritance

A generalnecessity in object oriented programming is the use of a derived class as a super class.



A ->B->C is known as inheritance path.

A derived class with multilevel base classes is declared as follows.

```
class A
{
……….
……….
}
class B extends A                    //First Level
{
……..
……..
}
class C extends B                    //Second Level
{
……..
……..
}
```

Example Program: Multilevel inheritance

```
class  students      //base class
{
        private int sno;
        private String sname;
        public void setstud(intno,String name)
        {
                sno=no;
                sname=name;
        }
        public void putstud()
        {
```

```
                System.out.println("Student No:"+sno);
                System.out.println("Student Name:"+sname);
        }
}
class marks extends students //derived or intermediate base class
{
        protected int mark1,mark2;
        public void setmarks(int m1,int m2)
        {
                mark1=m1;
                mark2=m2;
        }
        public void putmarks()
        {
                System.out.println("Mark1:"+mark1);
                System.out.println("Mark2:"+mark2);
        }
}
class finaltot extends marks // derived class
{
        privateint total;
        public void calc()
        {
                total=mark1+mark2;
        }
        public void puttotal()
        {
                System.out.println("Total:"+total);
        }
        public static void main(String args[])
        {
                finaltot f=new finaltot();
                f.setstud(100,"ABC");
                f.setmarks(78,89);
                f.calc();
                f.putstud();
                f.putmarks();
                f.puttotal();
        }
}
```

Example Program2:Multilevel Inheritance

```
class  Base
{
        voidbmsg()
        {
        System.out.println("Welcome to base class");
        }
}
class  Derive1 extends Base
```

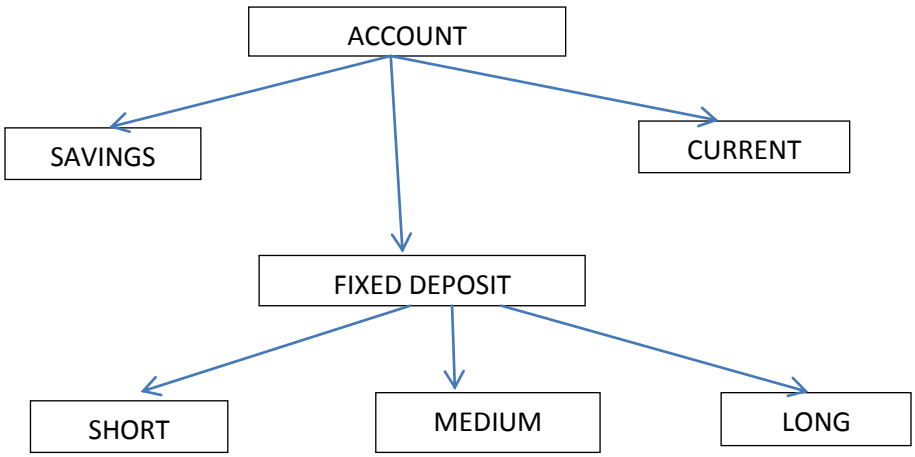
```
{
        void derive1msg()
        {
        System.out.println("Derive1msg");
        }
}
class Derive2 extends Derive1
{
        void  derive2msg()
        {
        System.out.println("Derive2msg");
        }
}
class Multilevel
{
public static void main(String args[])
{
Derive2 d2=new Derive2();
d2.derive2msg();
d2.derive1msg();
d2.bmsg();
}
}
```

Output:

```
Derive2msg
Derive1msg
Welcome to base class
```

Hierarchical inheritance

Class A is a super class of both class B and class C i.e one super class has many sub classes. Some features of one level are shared by many lower level casses



Example Program: Hierarchical inheritance

```
public class A
{
void DisplayA()
```

```
{
System.out.println("I am in A");
}
}

public class B extends A
{
void DisplayB()
{
System.out.println("I am in B");
}
}

public class C extends A
{
void DisplayC()
{
System.out.println("I am in C");
}
}

public class Mainclass
{
System.out.println("Calling for subclass C");
C c=new C();
c.DisplayA();
c.DisplayC();
System.out.println("Calling for subclass B");
B b=new B();
b.DisplayA();
b.DisplayB();
}
}
```

Output:
Calling for subclass C
I am in A
I am in C
Calling for subclass B
I am in A
I am in B

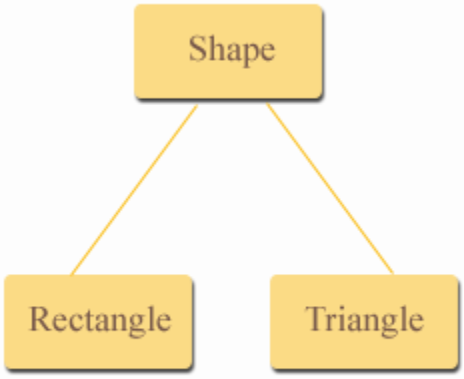
2.1.3. Protected Member:

The private members of a class cannot be openly accessed external class. Only functions of that class can access the private data fields directly. As discussed previously, however, occasionally it may be essential for a subclass to access a private member of a base class. If you make a private member public, then someone can access that member. So, if a member of a base class wants to be (directly) accessed in a subclass and yet still stop its direct access external class, you must declare that member as **protected**.

Following table gives the difference

Modifier	Class	Subclass	World
public	Y	Y	Y
protected	Y	Y	N
private	Y	N	N

Following program illustrates how the functions of a subclass can directly access a protected member of the base class



For example, let's consier a series of classes to describe two types of shapes: rectangles and triangles. These two shapes have definite general properties height and a width (or base).

This could be depicted in the world of classes with a class Shapes from which we can derive the two other ones : Rectangle and Triangle

```
public class Shape
{
  protected double height;  // To hold height.
  protected double width;  //To hold width or base
  public void setValues(double height, double width)
  {
    this.height = height;
    this.width = width;
  }
}

public class Rectangle extends Shape
{
public double getArea()
  {
    return height * width; //accessing protected members
  }
}
```

```
public class Triangle extends Shape
{
public double getArea()
  {
    return height * width / 2; //accessing protected members
  }
}

public class TestProgram
{
  public static void main(String[] args)
  {
    //Create object of Rectangle.
    Rectangle rectangle = new Rectangle();

    //Create object of Triangle.
    Triangle triangle = new Triangle();

    //Set values in rectangle object
    rectangle.setValues(5,4);

    //Set values in trianlge object
    triangle.setValues(5,10);

    // Display the area of rectangle.
    System.out.println("Area of rectangle : " +
                rectangle.getArea());

    // Display the area of triangle.
    System.out.println("Area of triangle : " +
                triangle.getArea());
  }
}
```

Output :
Area of rectangle : 20.0
Area of triangle : 25.0

2.1.4.Subclass Constructor

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

A **Subclass constructor** is used to build the instance variables of both the subclass and the superclass.

The subclass constructor uses the keyword super to call up the constructor method of the superclass. Keyword super is used subject to the subsequent conditions.

1."Super" may only be used within a subclass constructor method.

2. The call to super class constructor must show as the first statement inside the subclass constructor.

3.The parameters in the super class must equal to the order and type of the instance variable declared in the base class

```
class Animal
{
Animal()
{
System.out.println("animal is created");
}
}
class Dog extends Animal{
Dog()
{
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}
}
```

Output:

animal is created
dog is created

2.2.The Object class

Object class is a special class in java.If no inheritance is precise for the classes then all those classes are derived class of the **Object** class. We can consider ,**Object** is a superclass of all other classes by default. therefore

Public class A{……}is equal to public class A extends Object{……}

A reference variable of type Object can refer to any object of additional classes.

The package java.lang.Object includes below specified method

Method	Purpose
Object clone()	Creates a new object that is similar to object being cloned

boolean equals(Object object)	Concludes whether one object is similar to another
void finalize()	Called by an unused object is used again
class getclass()	Holds the class of an object at run time
inthashcode()	Returns the hash code associated with the invoking object
void notify()	Resumes execution of a thread waiting on the invoking object
void notifyall()	Resumes execution of all threads waiting on the invoking objecg
String toString()	Returns a string that describes the object
void wait()	Waits on another thread of execution
void wait(long milliseconds)	
void wait(long milliseconds,int nanoseconds)	

toString() Method of Object Class

The toString function returns the string type value.The syntax is**public String toString()**

If we call up the toStringmethod,by default then it gives a string which describes the object.This returned string contains the character "@" and object's memory address in hexadecimal form.

We can identify with the idea of toString() method by using as it is and overriding it with appropriate string.

Example:Illustration1

```
class A extends Object
{
}
class B extends A
{
}
class ObjectClassDemo
{
public static void main(String args[])
{
A obj=new A();
System.out.println("Obj:"+obj);
System.out.println("obj.toString():"+obj.toString());
}
}
```

Output:

```
obj:A@3e25a5
obj.toString():A@3e25a5
```

Example:Illustration2

```
class A extends Object
{
public String toString()          //method is overriden
```

```
{
String str=”Hello”;
returnstr;
}
}
class B extends A
{
}
classObjectClassDemo
{
public static void main(String args[])
{
A obj=new A();
System.out.println(“Obj:”+obj);
System.out.println(“obj.toString():”+obj.toString());
}
}
```

Output:
Obj:Hello
Obj.toString():Hello

Example2:

```
Import java.awt.*;
class StringDemo
{
public static void main(String args[])
{
Point c=new Point(10,20);       //Explicitly call toString() on object as part of string concatenation
System.out.println(“C=”+c.toString()); //Using the default object.toString() method
System.out.println(“C=”+c);     //Implicitly call toString() on object as part of string concatenation
String s=c+”testing”;
System.out.println(s);
}
}
```

Output:
C=java.awt.Point[x=10,y=20]
C=java.awt.Point[x=10,y=20]
java.awt.Point[x=10,y=20] testing

Equals method of Object class

The method equals is helpful for comparing values given by two objects.

Example1:

```
class A extends Object
{
int a=10;
public Boolean equals(Object obj)
{
```

```
if(objinstanceof B)
{
return a==((B)obj).b;
}
else
return false;
}
}

class B extends A
{
int b=10;
}
class ObjectClassDemo1
{
public static void main(String args[])
{
A obj1=new A();
B obj2=new B();
System.out.println("The two values of a and b are equal:"+obj1.equals(obj2));
}
}
```

Output:
The two values of a and b are equal:true

2.3.ABSTRACT CLASS AND METHODS

The base class is supposed to be the most common or less particular.Sometimes base class is so common and less specific that it does not anything but lists out only general features of different classes.Then such a base class is referred as abstract class.

For example,In java program we have formed three classes.

- class A is a base class consists of two methods namely fun1() and fun2(),
- The class B and class C are derived from class A
- .The class A is an abstract class since it contains one abstract method fun1().
- We have defined this method as abstract because,itsdefinition of fun1() is overridden in the derived classes B and C.
- an another function of class A that is fun2() is a regular function.

Definition of function overridden:
When a method of base class and derived classes consists of similar return type ,function name and parameters are called as function overridden.

Example1:Abstract Class and Methods

```
abstract class A  //abstract class
{
abstract void fun1();  //abstract method
void fun2()  //normal method
{
```

```
System.out.println("A:In fun2"):
}
}

class B extends A
{
void fun1()                    //function overridden from abstract class
{
System.out.println("B:In fun1");
}
}

class C extends A
{
void fun1()                     //function overridden from abstract class
{
System.out.println("C:In fun1");
}
}
}

public class AbstractClsDemo
{
public static void main(String args[])
{
B b=new B();
C c=new C();
b.fun1();
b.fun2();
c.fun1();
c.fun2();
}
}
```

Output:

```
B:In fun1
A:In fun2
C:In fun1
A:In fun2
```

Example2:

```
abstract class Base
{
abstract void fun();
}
class Derived extends Base
{
void fun()
{
```

```
System.out.println("Derived fun() called");
}
}
class Main()
{
public static void main(String args[])
{
Base b=new Derived();
b.fun();
}
}
```
Output:
Derived fun() called

Rules for writing abstract classes and abstract methods

1.An abstract method must be there in an abstract class only.It should not be there in an non-abstract class.
2.In all the non-abstract subclasses extended from an abstract base class all the abstract methods must be implemented.An un-implemented abstract method in the derived class is not acceptable.
3.Abstract class cannot be instantiated by the new operator
4.A constructor method of an abstract class can be defined and can be called by the derived classes.
5.A class that consists of abstract method must be abstract but the abstract class may not include an abstract method.This class is simply used as a base class for defining new subclasses
6.A derived class can be abstract but the derived class can be concrete.
7.The abstract class cannot be instantiated by new operator but an abstract class can be used as a datatype.

2.4.FINAL CLASS AND METHODS

The final keyword can be usedin three places

- For declaring variables
- For declaring the methods
- For declaring the class

Final Variables and Methods

A variable can be given as final.If a specific variable is declared as final then it cannot be changed again. The final variable is constant always.

For example: final int a=10;

The final keyword can also be useful to the method.The method using final keyword cannot be overridden.

Java program which makes use of the keyword final for declaring the method

```
public class FinalVariableDemo
{
final int number=10;   //final keyword used in variable
public void showFinalValue()
{
```

```
System.out.println("Final variable value:"+number);
}
public static void main(String args[])
{
finalVariableDemo  obVariableDemo=new FinalVariableDemo();
obVariableDemo . showFinalValue();
}
}
```

Output:

Final variable value:10

Java program which makes use of the keyword final for declaring the method

```
class Test
{
final void fun()    //final keyword used in method
{
System.out.println("Hello,this function declared using final");
}
}
class Test1 extends Test
{
final void fun()
{
System.out.println("Hello,this function declared using final");
}
}
```

Output:

Test.Java:10:fun() in Test1 cannot override fun() in Test;overridden method is final
final void cun()
1 error

Example mentioned above,on execution shows the error. sincefun method is declared with the keyword final and it cannot be overridden in sub class.

Final Class

If we declare specific class as final,no class can be derived from it.

Example1:Final Class

```
final class Test
{
void fun()
{
System.out.println("This is the function of base class");
}
}

class Test1 extends Test
```

```
{
final void fun()
{
System.out.println("This is the function of derived class");
}
}
```

Output:

```
Test.java:8 :cannot inherit from final Test
class  Test1 extends Test
1 error
```

Example2:Final Class

```
class point
{
intx,y;
}

classColoredPoint extends Point
{
int color;
}

final class Colored3dPoint extends ColoredPoint
{
int z;
}

Class FinalClassDemo
{
public static void main(String args[])
{
Colored3dPoint cObj=new Colored3dPoint();
cObj.z=10;
cObj.color=1;
cObj.x=5;
cObj.y=8;
System.out.println("x="+cObj,x);
System.out.println("y="+cObj,y);
System.out.println("z="+cObj,z);
System.out.println("Color="+cObj,color);

}
}
```

Output:

```
x=5
y=8
z=10
Color=1
```

2.5. INTERFACES

- Java does not support multiple inheritance.
- Classes in java cannot have more than one base class.
- Java gives an alternate method known as interfaces to implement the concept of multiple inheritance.

2.5.1.Defining interfaces

It is a type of a class but cannot be instantiated the new operator.Like classes,interface will havefunctions and variables but with a most important differenceInterfaces can have only abstract functions and final members.It won't be instantiated/implemented or extended.This means that interfaces do not identify any code to execute these functions and data membershave only constants.Therefore, it is the duty of the class that implements an interface to develop the code for implementation of such functions

Syntax:

```
interface interfacename
{
Variables declaration;
Methods declaration;
}
```

Interface is a keyword

Variable declaration-**static final type variablename=value**.
All variables declared as constants
Method declaration-Contains only l ist of methods.
returntypemethodname(parameter_list)

Example1:

```
interface Item
{
static final int code=100;
static final String name=fun;
void display();
}
```

Example2:

```
interface Area
{
final static float pi=3.14F;
float compute(float x,float y);
void show();
}
```

2.5.2.Implementing interfaces

Interfaces can be consider as base class.Properties are inherited by classes.

Syntax:

class classname implements interfacename
{
body of class
}

classclassname extends superclass implements interface1,interface2…
{
body of class
}

Example program1:

```
interface Area
{
final static float pi=3.14F;
float compute(float x,float y);
}

class Rectangle implements Area
{
public float compute(float x,float y)
{
return(x*y);
}
}

class Circle implements Area
{
public float compute(float x,float y)
{
return(pi*x*x);
}
}

class interfacetest
{
public static void main(String args[])
{
Rectangle rect=new Rectangle();
Circle cr=new Circle();
Area area;
area=rect;
System.out.println("Area of Rectangle:"+area.compute(10,20));
Area=cir;
System.out.println("Area of Circle:"+area.compute(10,0));
}
}
```

Output:

```
Area of Rectangle:200
Area of Circle:314
```

Implementing multiple and Hybrid inheritance

```
class student
{
int rollno;
void getno(int no)
{
Rollno=no;
}
Void putno()
{
System.out.println("Rollno:"+rollno);
}
}

class Test extends student
{
float mark1,mark2;
void getmarks(float m1,float m2)
{
mark1=m1;
mark2=m2;
}
void putmarks()
{
System.out.println("Mark1:"+mark1);
System.out.println("Mark2:"+mark2);
}
}

interface sports
{
floatsportwt=6.0F;
voidputwt();
}

class Results extends test implements sports
{
float total;
public void putwt()
{
System.out.println("Sportswt:"+sportwt);
}
void display()
{
total=mark1+mark2;
putno();
putmarks();
putwt();
```

```
System.out.println("Total Score:"+total);
}
}

class Hybrid
{
public static void main(String args[])
{
Results s1=new Results();
s1.getno(100);
s1.getmarks(50.0F,50.F);
s1.display();
}
}
```

Output:

```
Rollno:100
Mark1:50.0
Mark2:50.0
Sportswt:6.0
Total Score:100.0
```

Example2:

```
interface interface1
{
public void show_val();
}

class Base
{
int val;
public void set_val(int i)
{
val=i;
}
}

class A extends Base implements interface1
{
public void show_val()
{
System.out.println("The value of a="+val);
}
}

class B extends Base implements interface1
{
public void show_val()
{
System.out.println("The value of b="+val*5);
}
```

```
}

class multipleinherit
{
public static void main(String args[])
{
interface1 obj_A=new A();
interface1 obj_B=new B();
obj_A.set_val(10);
obj_B.set_val(20);
obj_A.show_val();
obj_B.show_val();
}
}
```

Output:
The value of a=10
The value of b=100

2.5.3.Difference between class and interface

Class	**Interface**
The class is represented by a keyword **class**	The interface is represented by a keyword **interface**
The class consists data members and methods.But the methods are defined in class implementation.Thus class consists ofan executable code	The interfaces may have data members and methods but the methods will not be defined.The interface serves as an summarize for the class
With the help of instance of a class ,class members can be accessed	Not possible to create an instance of an instance
The class can use different access specifiers like public,private or protected	The interface will use only public access specifier
The data members of a class can be constant or final	The data members of interfaces are constantly declared as final

2.5.4.Difference between abstract class and interface

Abstract Class	**Interface**
The new class can inherit only one abstract class	The class can implement more than one interfaces
Members of abstract class can have any access modifier such as public,private and protected.	Members of interface are public by default
The methods in abstract class may or may not have implementation	The methods in interface have no implementation at all.Only declaration of the methods is given
Java abstract classes are comparatively efficient	The interfaces are comparatively slow and implies extra level of indirection
Java abstract class is extended using the keyword abstract	Java interface can be implemented by using the keyword implements
The member variables of abstract class can be non final	The member variables of interface are by default final

2.5.5.Extending interfaces

One interface can able to extend with another one interfaces

The sub interface will take over all the data members of the base interface using 'extends' keyword

Syntax:

```
interface name2 extends name1
{
body of name2
}
```

2.6.OBJECT CLONING

Object cloning is a new object that has the similar state as the original but a dissimilar identity

Copying

When a replica of a variable is made,the original and the replica are references to the same object.This means a modify to either variable also affects the otherConsider the below coding:

```
Employee original=new Employee("ABC",5000);
Employee copy=original;
copy.raiseSalary(10);
```

Clone method is used when replica is made to be a new object that starts its life being equal to original but whose state can differ over time.

```
//must cast-clone returns an object
Employee copy=(Employee)original.clone();
copy.raiseSalary(10);
```

Types of cloning

1.Shallow copy
2.Deep copy

Shallow Copy

- It is a bitwise replica of an object.
- It has exact replica of values in the original.
- If any of the fields of the object are references to another object,just the references are duplicated.
- If the object that is copied holds references to other objects,a shallow copy refers to the similar subobjects.

Deep Copy

- It is a complete replica copy of an object.
- If an object has references to other objects, total new copies of those objects are also made.
- A deep copy generates a duplicate not only of the primitive values of the original object,but duplicate of all subobjects as well,all the way to the bottom.
- If you need a true, total copy of the original object,then you will have to to execute a full deep replica for the object.

- To construct a clone of an object,declare that an object implements cloneable and then give an override of the clone function of the typical java object super class

Example1:Object Cloning

```
Import java.util.*;

public class clonetest
{
public static void main(String args[])
{
Employee original=new Employee("ABC");
original.setHireDay(2000,1,1);
Employee copy=(Employee)original.clone();
copy.setHireDay(2002,12,31);
System.out.println("Original:"+original);
System.out.println("Copy:"+copy);
}
}

class Employee implements cloneable
{
private String name;
private Date hireDay;

public Employee(String n)
{
name=n;
}
public Object clone()
{
try
{
Employee cloned=(Employee)super.clone();
cloned.hireDay=(Date)hireDay.clone();
return cloned;
}
catch(CloneNotSupportedException e)
{
return null;
}
}
public void setHireDay(intyear,intmonth,int day)
{
hireDay=new GC(year,month-1,day).getTime();
}
public String toString()
{
return "Employee[name="+name+",hireDay="+hireDay+"]";
}
```

Example2: Object Cloning

```
public class CloneDemo
{
public static void main(String args[])
{
Person p1=new Person();
p1.setfirstname(“Bob”);
p1.setlastname(“Roy”);
Person p2=(Person)p1.clone();
System.out.println(“Person1”);
System.out.println(“First Name:”+p1.getfirstname());
System.out.println(“Last Name:”+p1.getlastname());

System.out.println(“Person2”);
System.out.println(“First Name:”+p2.getfirstname());
System.out.println(“Last Name:”+p2.getlastname());
}
}

class Person implements Cloneable
{
private String firstname;
private String lastname;

public Object clone()
{
Person obj=new Person();
obj.setfirstname(this.firstname);
obj.setlastname(this.lastname);
return obj;
}
public String getfirstname()
{
returnfirstname;
}
public void setfirstname(String firstname)
{
this.firstname=firstname;
}

public String getlastname()
{
returnlastname;
}
public void setlastname(String lastname)
{
this.lastname=lastname;
}
}
```

Output:
Person1
FirstName:Bob
LastName:Roy

Person2
FirstName:Bob
LastName:Roy

2.7. INNER CLASSES

Inner classes are the nested classes. We can simply represent that are defined inside the other classes.The below syntax defining the inner class is

```
Access_modifier class outerClass
{
        //code
Access_modifier class InnerClass
{
        //code
}
}
```

Properties of Inner class:

1.The outer class can take over as many number of innerclass objects as it needs.
2.If the outer class and the equivalent inner class both are public then any new class can create an instance of this inner class.
3.The inner class objects do not get instantiated with outer class object
4.outer class can be able to call the private functions of inner class.
5.Inner class code has openway to all members of the outer class object that contains it.
6.If the inner class has a variable with same name then the variable of outer class's can be accessed in given syntax

Outerclassname.this.variable_name

Types of Inner class:

There are four kinds of inner classes

1.Static member classes
2.Member classes
3.Local classes
4.Anonymous classes

1.Static member classes

- This inner class can be defined as the static member variable of other class.
- Static members of the outer class can access the static inner class.
- The non-static members of the outer class are not accessible to inner class.

Syntax:

```
Access_modifier class OuterClass
{
        //Code
```

```
        public static class InnerClass
        {
                //Code
        }
}
```

2.Member classes

This kind of inner class is non-static fields of outer class.

3.Local classes

This class is defined inside a Java code just similar to a local variable.
Local classes are not at all declared with an access specifier.
The life time of inner classes is constantlylimited to the block in which they are declared.
The local classes are totally hidden from the outside world.

Syntax:

```
Access_modifier class OuterClass
{
        //Code
        Access_modifierreturn_typemethodname(arguments)
        {
                classInnerclass
                {
                        //Code
                }
        //Code
}
```

4.Anonymous classes

Anonymous class is a local class with nospecified name.
Anonymous class is a one-shot class formed exactly where required.
The anonymous class is created in below situations

- When the class has very small body.
- Only one instance of the class is desired.
- Class is used directly after defining it.

The anonymous inner class can be able to extend the class,it can implement the interface or it can be stated in function argument.

Example:

```
class MyInnerClass implements Runnable
{
public void run()
{
System.out.println(“Hello”);
}

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

```
MyInnerClass my=new MyInnerClass();
Thread th=new Thread(my);
my.start();
}
}
}
```

Example2:

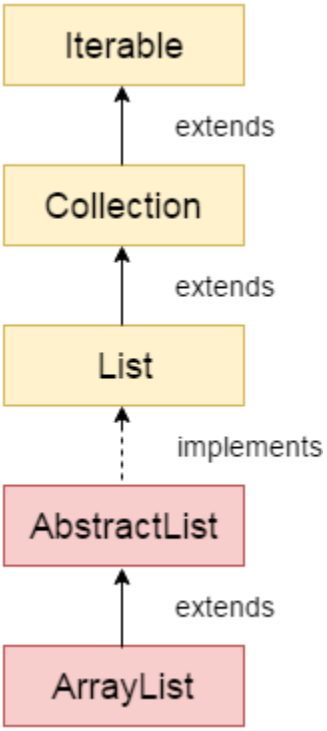
```
class Outer
{
int x=100;
void test()
{
Inner inner=new Inner();
Inner.display();
}

class Inner
{
void display()
{
System.out.println("display x:"+x);
}
}
}

class IC
{
public static void main(String args[])
{
Outer outer=new Outer();
outer.test();
}
}
```

2.8.ARRAY LIST CLASS

Dynamic array is used in ArrayList class for storing the elements. It derivesAbstractList class and implements List interface.

Features of Java ArrayList class are:

- Java ArrayList class can hold duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non-synchronized.
- Java ArrayList allows random access since array works at the index basis.
- In Java ArrayList class, operation is slow because a lot of shifting wants to be occurred if any element is deleted from the array list.ArrayList class hierarchy is specified in below



Example:

```
import java.util.*;
class TestCollection1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  list.add("Ravi");//Adding object in arraylist
list.add("Vijay");
  list.add("Ravi");
  list.add("Ajay");
  //Traversing list through Iterator
  Iterator itr=list.iterator();
  while(itr.hasNext()){
```

Two possible ways to iterate the elements of collection in java

There are two ways to visit collection elements:

1. By Iterator interface.
2. By for-each loop.

In the above example, we have seen visting ArrayList by Iterator. Let's see the example to go across ArrayList elements using for-each loop.

Iterating Collection through for-each loop

```
import java.util.*;
class TestCollection2{
 public static void main(String args[]){
  ArrayList<String> al=new ArrayList<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ravi");
  al.add("Ajay");
  for(String obj:al)
    System.out.println(obj);
 }
}
```

```
Ravi
Vijay
Ravi
Ajay
```

User-defined class objects in Java ArrayList

Let we see another example where we are storing Student class object in array list.

```
class Student{
  int rollno;
  String name;
  int age;
  Student(int rollno,String name,int age){
   this.rollno=rollno;
   this.name=name;
   this.age=age;
  }
}

import java.util.*;
public class TestCollection3{
 public static void main(String args[]){
  //Creating user-defined class objects
```

```
 Student s1=new Student(101,"Sonoo",23);
 Student s2=new Student(102,"Ravi",21);
 Student s2=new Student(103,"Hanumat",25);
 //creating arraylist
 ArrayList<Student> al=new ArrayList<Student>();
 al.add(s1);//adding Student class object
 al.add(s2);
 al.add(s3);
 //Getting Iterator
 Iterator itr=al.iterator();
 //traversing elements of ArrayList object
 while(itr.hasNext()){
  Student st=(Student)itr.next();
  System.out.println(st.rollno+" "+st.name+" "+st.age);
 }
}
}
```

Output

```
101 Sonoo 23
102 Ravi 21
103 Hanumat 25
```

Example of addAll(Collection c) method

```
import java.util.*;
class TestCollection4{
 public static void main(String args[]){
  ArrayList<String> al=new ArrayList<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ajay");
  ArrayList<String> al2=new ArrayList<String>();
  al2.add("Sonoo");
  al2.add("Hanumat");
  al.addAll(al2);//adding second list in first list
  Iterator itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Output

```
Ravi
Vijay
Ajay
Sonoo
Hanumat
```

The other methods of ArrayList are also used such as removeAll() methodretainAll() method.

```
  System.out.println(itr.next());
 }
}
}
```

Output:

```
Ravi
    Vijay
    Ravi
    Ajay
```

2.9.STRINGS

String is a collection of characters.
Easiest way to represent a collection of characters in java is by using a character array.

```
char chararray[]=new char[4];
chararray[0]='J';
chararray[1]='a';
chararray[2]='v';
chararray[3]='a';
```

In java,strings are class objects and implemented using two classes that is strings and stringbuffer.
A java string is an instantiated object of the string class.

```
String stringname;
Stringname=new String("string");
```

Example

```
String fname;
firstname=new String("CSE");
```

String Arrays

```
String items[]=new String[3];
```

String Methods

The string class defines a number of methods that allow us to accomplish a variety of string manipulation task.

String Methods	**Task Performed**
s2=s1.toLowercase;	Converts the string s1 to all lowercase
s2=s1.toUppercase;	To all uppercase
s2=s1.replace('x','y');	Replaces all appearances of x with y
s2=s1.trim();	Remove white spaces at the beginning and end of string
s1.equals(s2)	Return true if s1 is equal to s2
s1.length()	Gives the length of s1
s1.charAt(n)	Gives n^{th} character of s1
s1.CompareTo(s2)	Returns negative if s1<s2,positive if s1>s2 & zero if s1 is equal to s2.

s1.concat(s2)	Concatenate s1 and s2
s1.substring(n)	Gives substring starting from n^{th} character
s1.substring(n,m)	Gives substring starting from n^{th} character up to m^{th}.
s1.indexOf('x')	Gives the position of the first occurrence of 'x' in the string s1.
s1.indexOf('x',n)	Gives the position of 'x' that occurs after n^{th} position in the string s1.
String.valueOf(variable)	Converts the parameter value to string representation.

Example Program

```
class Stringordering
{
static stringname[]={"Madrs","Delhi","Calcutta","Bombay"};
public static void main(String args[])
{
int size=name.length;
String temp=null;
for(int 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;
}
}
}
for (int i=0;i<size;i++)
{
System.out.println(name[i]);
}
} }
```

Output:

Bombay
Calcutta
Delhi
Madras

String Buffer Class:

String buffer creates strings of flexible length that can be modified

Method	Task
s1.charAt(n,'x')	Modifies the nth character to x
s1.append(s2)	Appends the string s2 to s1 at the end
s1.insert(n,s2)	Inserts the string s2 at the position n of the

	string s1
s1.setLength(n)	Sets the length of the string s1 to n.

Manipulation of strings: Example

```
class Stringmanipulation
{
public static void main(String args[])
{
StringBuffer str=new StringBuffer("Object Language");
System.out.println("Original String:"+str);
System.out.println("Length of string"+str.length());
for(int i=0i<str.length();i++)
{
int p=i+1;
System.out.println("character aat position"+p"is"+str.charAt());
}
String astringnew String(str.toString());
int pos=astring.indexOf("language");
str.insert(pos,"oriented");
System.out.println("Modified String"+str);
str.setcharAt(6,'-');
System.out.println("String now"+str);
str.append("improved security");
System.out.println("appended string"+str);
}
}
```

Output

Original String: Object Language
Character at Position : 1 is o
.
.
Modified String : Object Oriented Language
String now:Object –oriented language
Appended String : Object-Oriented language improves security

2 MARK QUESTIONS AND ANSWERS

1. Define Inheritance. May/June 2012

Inheritance can be defined as the process where one object acquires the properties of another. With the use of inheritance the information is made manageable in a hierarchical order.The resulting classes are known as derived classes, subclasses, or child classes. Older class is known as super class.

2.What are the conditions to be satisfied while declaring abstract classes

Java Abstract classes are used to declare common characteristics of subclasses. An abstract class cannot be instantiated. It can only be used as a superclass for other classes that extend the abstract class. Abstractclasses are declared with the abstract keyword. Abstract classes are used to provide a template or design for concrete subclasses down the inheritance tree.

3. You can create an abstract class that contains only abstract methods. On the other hand, you can create an interface that declares the same methods. So can you use abstract classes instead of interfaces?

Sometimes. But your class may be a descendent of another class and in this case the interface is your only option

4. Explain the concept of Polymorphism.

Polymorphism means when an entity behaves differently depending upon the context its being used. Moreover In other words Polymorphism is the capability of an action or method to do different things based on the object that it is acting upon. Means polymorphism allows you define one interface and have multiple implementations. That being one of the basic principles of object oriented programming.

5. Explain about Virtual methods.

A child class can override a method in its parent. An overridden method is essentially hidden in the parent class, and is not invoked unless the child class uses the super keyword within the overriding method.

6. What is Static Binding?

Connecting a method call(i.e. Function Call) to a method body(i.e. Function) is called binding.

When binding is performed before the program is run (by the compiler and linker, if there is one), it's called early binding or static Binding.

7. What is Dynamic binding?

The runtime system [JVM]during runtime determines the appropriate method call based on the class of the object. This feature is called as Polymorphism. All the methods in java are dynamically resolved. This cannot be determined by the Compiler.

8. What is an Abstract classes and Methods?

An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

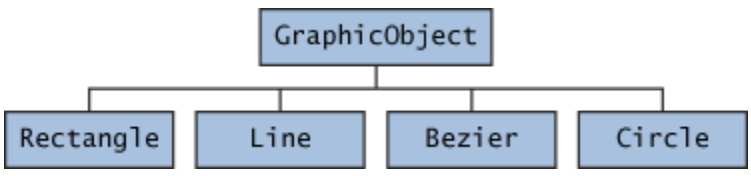
```
abstract void moveTo(double deltaX, double deltaY);
```

If a class includes abstract methods, the class itself *must* be declared abstract, as in:

```
public abstract class GraphicObject {
// declare fields
// declare non-abstract methods
abstract void draw();
}
```

When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, the subclass must also be declared abstract.

9. With example Explain Abstract Classes?



First, you declare an abstract class, GraphicObject, to provide member variables and methods that are wholly shared by all subclasses, such as the current position and the moveTo method. GraphicObject also declares abstract methods for methods, such as draw or resize, that need to be implemented by all subclasses but must be implemented in different ways.

10. When an Abstract Class Implements an Interface?

It was noted that a class that implements an interface must implement *all* of the interface's methods. It is possible, however, to define a class that does not implement all of the interface methods, provided that the class is declared to be abstract. For example,

```
abstract class X implements Y {
// implements all but one method of Y
}

class XX extends X {
// implements the remaining method in Y
}
```

In this case, class X must be abstract because it does not fully implement Y, but class XX does, in fact, implement Y.

11. What is Object Class?

The Object class defines the basic state and behavior that all objects must have, such as the ability to compare oneself to another object, to convert to a string, to wait on a condition variable, to notify other objects that a condition variable has changed, and to return the object's class.

12. What is an interface (Nov/Dec 2011)

An interface is a collection of method definitions (without implementations) and constant values. In Java, an interface is a reference data type and, as such, can be used in many of the same places where a type can be used (such as in method arguments and variable declarations)

13. What are the uses of Interfaces?(Nov/Dec2010)

- Capturing similarities between unrelated classes without forcing a class relationship.
- Declaring methods that one or more classes are expected to implement.
- Revealing an object's programming interface without revealing its class. (Objects such as these are called anonymous objects and can be useful when shipping a package of classes to other developers.)

14. Why Interfaces Do not Provide Multiple Inheritances?

- You cannot inherit variables from an interface.
- You cannot inherit method implementations from an interface.
- The interface hierarchy is independent of a class hierarchy--classes that implement the same interface may or may not be related through the class hierarchy. This is not true for multiple inheritances.

15. What is meant by Reflection API?

Reflection is commonly used by programs which require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine. This is a relatively advanced feature and should be used only by developers who have a strong grasp of the fundamentals of the language.

16. What are the uses of Reflection?

Extensibility Features

An application may make use of external, user-defined classes by creating instances of extensibility objects using their fully-qualified names.

Class Browsers and Visual Development Environments

A class browser needs to be able to enumerate the members of classes. Visual development environments can benefit from making use of type information available in reflection to aid the developer in writing correct code.

Debuggers and Test Tools

Debuggers need to be able to examine private members on classes. Test harnesses can make use of reflection to systematically call a discoverable set APIs defined on a class, to insure a high level of code coverage in a test suite.

17. What are the Drawbacks of Reflection?

Reflection is powerful, but should not be used indiscriminately. If it is possible to perform an operation without using reflection, then it is preferable to avoid using it. The following concerns should be kept in mind when accessing code via reflection.

- **Performance Overhead**
- **Security Restrictions**
- **Exposure of Internals**

18. What is object cloning in Java? (May/June 2013)

Objects in Java are referred using reference types, and there is no direct way to copy the contents of an object into a new object. The assignment of one reference to another merely creates another reference to the same object. Therefore, a special clone() method exists for all reference types in order to provide a standard mechanism for an object to make a copy of itself. Here are the details you need to know about cloning Java objects.

19.Define Inner classes. (Apr/May 2011)

- An inner class is a class that is defined inside another class
- Inner class methods can access the data from the scope in which they are defined including data that would otherwise be private.
- Inner classes can be hidden from other classes in the same package.
-

20. What are the properties of proxy class ?

- Proxy classes are created on the fly in the running program.
- Once they are created they are just like any other class in the V.M.
- All proxy classes extends the class proxy.
- A proxy class as only one instant field. The innovation handles which is defined in the proxy super class.

21. What is final modifier?(May/June 2013)

The final modifier keyword makes that the programmer cannot change the value anymore. The actual meaning depends on whether it is applied to a class, a variable, or a method.
final Classes- A final class cannot have subclasses.
final Variables- A final variable cannot be changed once it is initialized.
final Methods- A final method cannot be overridden by subclasses.

PART B -13 MARK QUESTIONS

1. Explain the concept of inheritance and its types.
2. Explain the concept of overriding with examples.
3. What is dynamic binding? Explain with example.
4. Explain the uses of reflection with examples.
5. Define an interface. Explain with example.
6. Explain the methods under "object" class and "class" class.
7. What is object cloning? Explain deep copy and shallow copy with examples.
8. Explain static nested class and inner class with examples.
9. With an example explain proxies.
10. Develop a message abstract class which contains playMessage abstract method. Write a different sub-classes like TextMessage, VoiceMessage and FaxMessage classes for to implementing the playMessage method.
11. Develop a abstract Reservation class which has Reserve abstract method. Implement the sub-classes like ReserveTrain and ReserveBus classes and implement the same.
12. Develop an Interest interface which contains simpleInterest and compInterest methods and static final field of Rate 25%. Write a class to implement those methods.
13. Develop a Library interface which has drawbook(), returnbook() (with fine), checkstatus() and reservebook() methods. All the methods tagged with public.
14. Develop an Employee class which implements the Comparable and Cloneable interfaces. Implement the sorting of persons (based on name in alphabetical). Also implement the shallow copy (for name and age) and deep copy (for DateOfJoining).
15. Explain the different methods supported in Object class with example.

Part-C 15 MARK QUESTIONS

1. Develop a static Inner class called Pair which has MinMax method for finding min and max values from the array.
2. Explain the following with examples(NOV/DEC 2010)
    i. The clone able interface(8)
    ii. The property interface. (7)
3. Develop an application using inheritance and interfaces
4. Develop a book application and apply various string operations to find particular string.
5. With the help of real time application explain object cloning in java.