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.outprintin("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.outprintin("Hello,this function declared using final");
}
}
class Test1 extends Test
{
final void fun()
{
System.outprintin("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.outprintln("This is the function of base class”);
}
}

class Test1 extends Test
{
final void fun()
{
System.outprmtin("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.outprmtin("x=”+cObj,x);
System.outprmtin("y=”+cObj,y);
System.outprmtin("z=”+cObj,z);
System.outprmtin("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 interface l,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.outprmtln("Area of Rectangle:”+area.compute(10,20));
Area=cir;
System.outprmtln("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.outprmtin("RoHno:”+roHno);
}
}

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

interface sports
{
floatsportwt=6.0F;
```

```
voidputwt();
}

class Results extends test implements sports
{
float total;
public void putwt()
{
System.outprmtin("Sportswt:”+sportwt);
}
void display()
{
total=mark1+mark2;
putno();
putmarks();
putwt();
System.outprmtin("'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.outprmtin("The value of a=”+val);
}
}

class B extends Base implements interface1
{
public void show_val()
{
System.outprmtin("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.outprmtln("Origmal:"+original);
System.outprintln("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");
p 1 .setlastname ("Roy");
Person p2=(Person)p1.clone();
System.outprmtin("Person1");
System.outprmtin("First Name:"+p1.getfirstname());
System.outprmtin("Last N ame: "+p1.getlastname ());

System.outprmtin("Person2");
System.outprmtin("First Name:"+p2.getfirstname());
```

```
System.outprmtin("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