Ch – 2
Inheritance , Java Packages , Interface

Prepared By :

Ms. Kakadiya Jainam A.

Universal Class

- The Object Class is known as universal super class of Java.
- This is so because Every class you create in Java automatically inherits the Object class.
- The Object class is super class of all the classes in Java either directly or Indirectly.

Example of Universal Class

```
public class ObjectClass { public static void
   main(String[] args)
{ ObjectClass oc = new ObjectClass();
   System.out.println(oc.toString());
   System.out.println(oc.getClass());
ObjectClass oc1 = new ObjectClass();
   System.out.println(oc.equals(oc1));
   System.out.println(oc1.toString());
ObjectClass oc2 = oc;
   System.out.println(oc.equals(oc2)); } }
```

Inheritance

- Inheritance is important OOP principle by which you can do hierarchical classification of your program.
- By, inheritance, a class can use some or all properties of another class.
- The class which is begin inherited is called Super class and the class that inherited the properties of super class is called the subclass.

Example of Inheritance

- To inherit a class **extends** keyword is used.
  For example :

```
Class superclass { // super class
   int a,b;
   void displaySuper() {
        System.out.println(“Displaying superclass”);
        System.out.println(“a=”+a+”and b=”+b); }
   }
Class subclass extends  superclass { // sub class
Int c,d;
Void displaysub() {
```

Example of Inheritance

```
    System.out.println("Display subclass");
    System.out.println("a="+a+"and b="+b);
    System.out.println("c="+c+"and d="+d); } }
Class Inheritance {
    public void main(String arg[]) {
      subclass sub=new subclass();
      sub.a=10; sub.b=20; // subclass can access
                                  super class variable
     sub.c=30; sub.d=40; // subclass can call super
                                  class method
     sub.displaySub();
} }
```

The super keyword

- There are two uses of super keyword :
  - 1. To call the constructor of super class.
  - 2. To access the members of super class.

- **1) To call the constructor of super class :** Using super keyword you can call the constructor of the super class.

- **2) Use super to access members of super class :** You can use super keyword to access members of super class. **For example :**
  - super.member;
  - Super.methodName();

Type of Inheritance in Java

- There are three types of inheritance in java:

- **1) Single Inheritance**
  - In single inheritance, there is one super class and one subclass.

- **2) Hierarchical Inheritance**
  - In this type, there is one super class and more than one subclass of it.

- **3) Multilevel Inheritance**
  - In this type one class extends super class and another subclass extends this subclass.

Dynamic Method Dispatch

- We have seen that the method overriding is an important concept concept in java as it offers the concept of polymorphism. But this was the compile time polymorphism not runtime.

- The main power of method overriding is in runtime polymorphism.

- Means the call to a method should be resolved at runtime. “The Dynamic method dispatch is the mechanism by which the call to an overriden method is resolved at runtime rather than compile time.”

Example of Dynamic Method Dispatch

```
Class shape()
{  double height,width;
   shape(double d,double w)
   {    width=w;
        height=h;
   }
   double area()
   {    return o;
   }
}
   class Rectangle extends shape
   {
```

Example of Dynamic Method Dispatch

```
Recgtangle(double d, double h)
{  super(w,h); }
   double area()
    { return width*height; }
}
Class DynamicMeth {
Public static void void main(String arg[]) {
   shape ref; // creating reference of subclass
   rectangle r=new Rectangle(20,30);
   ref=r; // r is assigned ref. to refer to Rectangle class
   System.out.println("Area of Rect:"ref.area());
```

Abstract class and Methods

- There may be a situation when a method in the super class does not define a real implementation, but it is for its subclass to define it properly. Means super class define just a structure or a general form that will be implemented by its subclass.

- A method can be declared as abstract by keyword '*abstract'.* The class which has abstract methods must be declared as abstract.

Abstract class and Methods

- There are some rules for abstract class and methods :
  - The abstract methods must be in abstract class.
  - The abstract methods do not have body.
  - No objects can be created of abstract class.
  - The abstract class must be extended by at least one subclass.
  - All the abstract methods must be overridden.

Package

- Package is a very important concept of java. Simply saying package is just a container or collection of classes. So it is a directory or folder which contains the source files and its class file.

- Using package you can have more than one class of same name.

How to define a Package ?

- To create a package, package keyword is used as the first statement of your program. After package keyword you have to give the package name.

- Following is the syntax to define a package :
  - Package packagename;

- Example :
  - Package p1;

Example

```
Class player {
   string nm;
   int score;
   player(String nm, int score)
   {    this.nm=nm;
        this.score=score;
   }
   void display()
   {
        System.out.println(“Player name:”+nm);
        System.out.println(“Highest score”+score); }
   }
```

Example

```
Class packtest
{
   public static void main(string args[])
   {
     player p=new player(“Sachin”,200);
     player p1=new player(“Rahul”,180);
     p.display();
     p1.display();
   }
}
```

Output :

Player name: Sachin

Height score : 200

Player name : Rahul

Height score : 180

Access control

- In previous chapter, we have already seen java's access control mechanism. Here they are discussed again to understand how they affect members of other package also.

Specifier	private	Default	Protecte	Public
1) Same class	Y	Y	Y	Y
2) Same pack	N	Y	Y	Y
Diff. class				
3) Diff. pack	N	N	Y	Y
Same class				
4) Diff pack non-sub class	N	N	N	Y

How to import a package ?

- After creating a package, you can use its classes in other java files.
- To do this you have to import that package in which the class is resided.
- Importing a package is similar to including header file in C or C++, in C we include <stdio.h> to use printf() and scanf() functions.
- Similar we can import a package to use classes and methods of it. To import a package ***import*** keyword is used.

Example

```
Package mypack; // defining package mypack

Public class player
{
   String name;
   int score;
   public player(Strinng name,int score)
    { this.name=name;
   this.score=score;
   public void display() {
   System.out.println(“Player name:”+name);
   System.out.println(“Highest score”+score);
}   }
```

Example

```
//import mypacck.*;
Class importex
{
   public static void main(String arg[])
   {
   player p=new player(“Saurav”,200);
   player p1=new player(“Sehwag”,300);
   p.display();
   p1.display();
   }
}
```

Output :

Player name: Saurav

Height score : 200

Player name : Sehwag

Height score : 300

Interfaces

- The interface is a fully abstract class since its all methods are abstract.

- So an interface defines a structure that what a class has to do to implement that interface. An interface is a prototype for a class to implement it.

- Your class can not extend more than one class but it can implement more than one interface.

How to define an Interface ?

- Defining an interface is quite similar to defining a class. Here is the general form of it :

```
accessSpecifier interface innterfaceName {
    Type variable1=value;
    Type variable2=value;
    returnType method1(parameter list);
    returnType method2(parameter list);
```

- The ***interface*** keyword is used to define an interface.

How to implement an Interface ?

- After defining an interface you can implement it by implements keyword.

- Example :

```
Interface myinterface
{  int val=100;
   void print(); //method declared
   int getvalue(); }
Class myclass implements myinterface {
   public void print() { // method of interface must be public
   System.out.println(“this is print method”);
}
```

Output :

This is print method

Value is :200

This is own method of myclass

```
   public int getvalue() { //method of interface public
   int newval=val+100; //val can be used but can't changed
   return newval;
}
Void ownMethod { //this is method of myclass
   System.out.println("this is own method of myclass"); }
}
Class interfaceEx
{
Public static void main(String arg[]) {
   myclass m=new myclass();
   m.print();
   System.out.println("value is:"+m.getvalue());
   m.ownMethod(); }
}
```

An interface can extends another interface

- An interface can inherit another interface also. That is the inheriting interface adds some extra method declaration of its own.

- So any class implementating this inheriting interface must implement all the methods of both the interface.

```
Interface superInterface {
   void method1();
   void mthod2(int val);
}
Interface subInterface extends superinterface {
   void method3(string msg);  }
Class Testclass implemets subInterface {
   public void method1() {
   system.out.println(“implementation of method1”);
   }
   public void method2(int val) {
   system.out.println(“the value is”+val) ;
   }
```

```
    public void method3(string msg) {
    system.out.println("the message is "+msg); }
    }
Class democlass {
    public static void main(String arg[]) {

Testclass t=new Testclass();
    t.method1();
    t.method2(100);
    t.method3("Hello friends");
```

Output :

Implementation of method1

The value is : 100

The message is : Hello friends

Nested classes and inner classes

- You can define a class within another class. These classes are known as nested classes.

- For example :

```
Class Outerclass {
   Int o=10;
   Void test() {
        inner in=new inner();
        in.display(); }
Class inner {
```

Example of Nested classes and inner classes

```
int i=100;
Void display() {
    System.out.println(“Outer is:”+o); }  }
void showInner() {
    inner in=new inner();
    System.out.println(“Inner is:”+in.i) ; } }
class InnerClassEx {
public static void main(String args[]) {
    OuterClass outer=new OutreClass();
    outer.test();
    outer.showInner(); } }
```