Java Programming

Packages and Interfaces

Module 5

Agenda

1 Introduction to Packages

2 Importing Classes

3 Introduction to interfaces

4 Applying Interfaces

Objectives

At the end of this module, you will be able to:

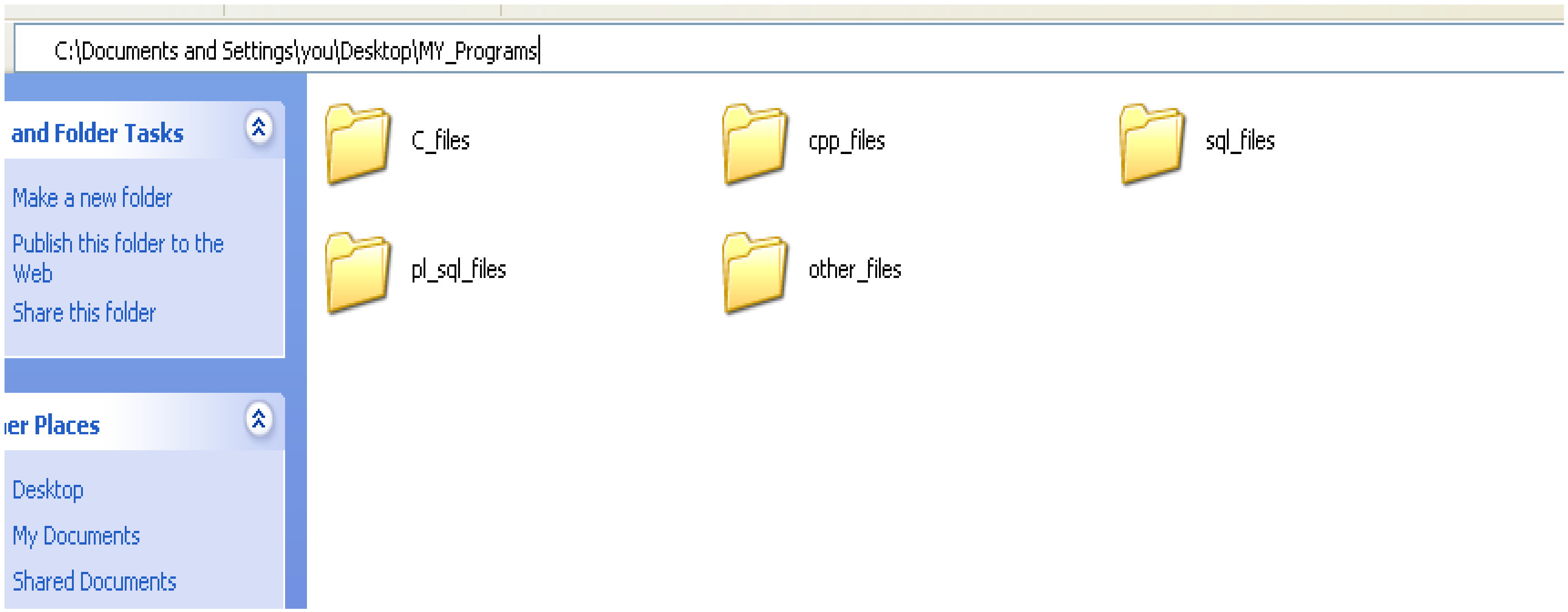
- Understand the use of packages
- Use inbuilt java packages
- Create our own packages
- Import existing packages
- Create interfaces
- Understand the relevance and uses of interfaces in java

Introduction to Packages

Package is similar to folders in your Disk

- People normally group their related data in various folders.
  - For example, a programmer may group his programs into the following folders.
  - C_programs,   CPP_programs, SQL_queries, PLSQL_programs.. etc.
- We also use sub-folders for organizing our data more conveniently.
    - The advantage is we can easily locate the files if they are organized.

Package is similar to folders in your Disk (Contd.)



Just relate package concept with directories concept in your file system

Organizing classes into Packages

- Packages are containers for classes and interfaces
- Classes and interfaces are grouped together in containers called **packages**
- To avoid namespace collision, we put the classes into separate containers called **packages**
- Whenever you need to access a class, you access it through its package by prefixing the class with the package name

Need for Packages

- Packages are containers used to store the classes and interfaces into manageable units of code.
- Packages also help control the accessibility of your classes. This is also called as visibility control.
- Example:

```
package MyPackage;
class MyClass {…}
class YourClass{…}
```

Access Protection using Packages

- Packages facilitate access-control
- Once a class is packaged, its accessibility is controlled by its package
- That is, whether other classes can access the class in the package depends on the access specifiers used in its class declaration
- There are four visibility control mechanisms packages offer:
  - private
  - no-specifier (default access)
  - protected
  - public

Packages & Access Control

Specifier	Accessibility
private	Accessible in the same class only
protected	Subclasses and non-subclasses in the same package, and subclasses in other packages
No-specifier (default access)	Subclasses and non-subclasses in the same package
public	Subclasses and non-subclasses in the same package, as well as subclasses and non-subclasses in other packages. In other words, total visibility

Packages & Access Control (Contd.)

Specifier	**Accessibility**
private	same class only
protected	same package and subclasses
No-specifier (default access)	same package only
public	Anywhere in the program

Inbuilt Packages

- We know that the java language support files are available in various in-built packages.
  - java.lang, java.io, java.util, java.awt are some of the in-built packages.
  - Unzip and explore the src directory under jdk1.5 folder to find all the packages.

Quiz

- Which is not a correct inbuilt java package?
  A) java.io
  B) java.sql
  C) java.dbms
  D) java.net

Option C is invalid package;
Others are valid java packages.

Creating our own Packages

- Similarly, in java, we can create our own packages
  - **Package statement helps us to create our own package**
  - We group related classes and interfaces into a package
  - We can have sub-packages inside our packages as required

Importing Classes

Packages & import statement

- Naturally, after creating the packages, we need to use them in our programs. Java provides import statement.
  - Import means, we can including the classes and interfaces of existing packages into our programs.
- For example,
  - import java.awt.*; -- this will be importing awt package
  - import java.awt.event.*; -- this will be importing event package which is a sub package under awt package.
- If you need a sub package, then, you need to issue a separate import statement.

Storing the Packages

- Packages are stored as directories
- All the classes you create in the package MyPack should be saved in the directory MyPack
- First create a directory by the name MyPack (packagename)
- Remember, the case should match exactly

Quiz

- Which is the correct usage of import statement?

  A) import java.*;

  B) import java.lang.*;

  C) import *;

  D) import *.*;

Only Option B is correct;
Others are invalid.

Understanding CLASSPATH

- What is CLASSPATH?
- CLASSPATH is an environment variable that tells the Java runtime system where the classes are present
- When a packages is not created, all classes are stored in the default package
- The default package is stored in the current directory.
- The current directory is the default directory for CLASSPATH.

Understanding CLASSPATH (Contd.).

- When you create your own package for example MyPack, all the .class files including MyClass are saved in the directory MyPack.
- In order for a program to find MyPack, one of two things must be true:
  - Either the program is executed from a directory immediately above MyPack, or
  - CLASSPATH must be set to include the path to MyPack

Creating our own Package Example

```
package empPack;
class EmpClass{
  String empName;
  double salary;
  EmpClass(String name, double sal){
    empName = name;
    salary = sal;
  }
  void display(){
    System.out.println(empName + " : $"+salary);
  }
}
```

Creating our own Package Example (Contd.).

```
class EmpSal{
  public static void main(String args[]){
    EmpClass emp[] = new EmpClass[4];
    emp[0] = new EmpClass("Bill Gates",450.20);
    emp[1] = new EmpClass("D.M Ritchie",725.93);
    emp[2] = new EmpClass("Tagore",630.80);
    emp[3] = new EmpClass("Kalam",545.60);
    for (int i=0; i<4; i++)
        emp[i].display();
  }
}
```

How you will save this file?
<u>In command prompt:</u>
How you will compile?
How you will run?

Importing Classes from Packages

- Java has used the package mechanism extensively to organize classes with similar functionality in one package
- If you want to use these classes in your applications, you can do so by including the following statement at the beginning of your program:
  - *import packagename.classname;*
- *If the packages are nested you should specify the hierarchy.*
  - *import package1.package2.classname;*

Importing Classes from Packages (Contd.).

- The class you want to use must be qualified by its package name.
- If you want to use several classes from a package, it would be cumbersome to type so many classes qualified by their packages.
- It can be made easy by giving a star(*****) at the end of the import statement. For example:
  - **import package1.*;**

Static Import

- A static import declaration enables us to refer to imported static members as though they were declared in the current class
- If we use static import, we first have to import this static member in the following way :

```
package p1;
public class Abc {
       public static void xyz() {
       System.out.println("static import demo");
       }
}
```

```
package p2;
import static p1.Abc.xyz;
public class A1 {
       public static void main(String[] args) {
              xyz();
       }
}
```

Output : “static import demo”

Static Import (Contd.).

- If we are invoking multiple static members of the same class, we can also use asterisk(*), which indicates that *all* static members of the specified class should be available for use

```
import static java.lang.Math.*;
public class StaticImportDemo  {
    static float x = 4.556f;
    static double y =4.556;
    public static void main( String args[] )   {
      float a1 = abs(x);
      int r1  = round(x);
      double s1 = sqrt(y);
  System.out.println("absolute value of "+x+" is" +a1);
  System.out.println("When we round off "+x+"we get" +r1);
      System.out.println("Square Root of "+y+ "is" +s1);
    }
  }
```

Quiz

In one java source file, how many package statements can be used?

A) One

B) Two or more

Only Option A is correct;
You can’t have two or more package statements in a java source file

Working with Packages – Example 1

```
package automobile;

public class Vehicle {
public void printname() {
        System.out.println("My name is vehicle");
        System.out.println(" I am defined inside
  automobile package");
    }
}
```

What is the package name?
How you will save this file?

Working with Packages – Example 1 (Contd.).

```
package automobile;

public class Bike extends Vehicle {
     public void printname() {
     System.out.println("My name is bike");
     System.out.println(" I am defined inside 
  automobile package");
     }
}
```

What is the package name?
How you will save this file?

Working with Packages – Example 1 (Contd.).

```
package automobile;

public class Car extends Vehicle {
    public void printname() {
        System.out.println("My name is car");
        System.out.println(" I am defined inside
  automobile package");
    }
}
```

What is the package name?
How you will save this file?

Working with Packages – Example 1 (Contd.).

```
package au_test;
import automobile.*;
public class tester {
public static void main(String s[ ] ) {
System.out.println(" I am tester class defined
  inside au_tester package");
System.out.println(" I had imported all classes
  of automobile package");
System.out.println(" Creating instances of
  Vehicle, Car and Bike ");
System.out.println(" ------------------------");
```

Working with Packages – Example 1 (Contd.).

```
Vehicle v = new Vehicle();
Car c = new Car();
Bike b = new Bike();
System.out.println(" Accessing the functions
  using objects");
System.out.println(" --------------------- ");
 v.printname();
 c.printname();
 b.printname();
}
}
```

How you will save this file?
<u>In command prompt:</u>
How you will compile?
And How you will run?

What is the output of the program?

What will be the result, when you try to compile and execute :

```
class A1 {
    protected void m1() {
        System.out.println(“m1 method of class A1”);
    }
}
class A2 extends A1 {
    void m1() {
        System.out.println(“m1 method of class A2”);
    }
    public static void main(String[] args) {
        A2 x = new A2();
        x.m1();
    }
}
```

Compilation Error…Why?

What will be the result, when you try to compile and execute (Contd.).

```
class A1 {
        protected void m1() {
                System.out.println(“m1 method of class A1”);
        }
}
class A2 extends A1 {
        public void m1() {
                System.out.println(“m1 method of class A2”);
        }
        public static void main(String[] args) {
            A2 x = new A2();
            x.m1();
        }
}
```

The code compiles and executes successfully..! Prints “m1 method of class A2”

What will be the result, when you try to compile and execute : (Contd.).

```
class A1 {
    protected void m1() {
        System.out.println(“m1 method of class A1”);
    }
}
class A2 extends A1 {
    void m1(int i) {
        System.out.println(“m1 method of class A2”);
    }
    public static void main(String[] args) {
      A2 x = new A2();
      x.m1();
    }
}
```

The code compiles and executes successfully..! Prints “m1 method of class A1”

Introduction to interfaces

What is an Interface?

An interface is a named collection of method declarations (without implementations)

- An interface can also include constant declarations
- An interface is syntactically similar to an abstract class
- An interface is a collection of abstract methods and final variables
- A class implements an interface using the **implements** clause

What is an Interface? (Contd.).

- An interface defines a protocol of behavior
- An interface lays the specification of what a class is supposed to do
- How the behavior is implemented is the responsibility of each implementing class
- Any class that implements an interface adheres to the protocol defined by the interface, and in the process, implements the specification laid down by the interface

Interface: Example

```
   Calculate_salary
 IN: emp_id String(10)
OUT:  salary float (6,2)
```

Interface 'lif_salary'

Write code to Calculate salary for US employees

Sheldon

class lcl_salary_US

Implements lif_salary

class lcl_salary_IN

Implements lif_salary

Write code to Calculate salary for India employees

Rahul

Why interfaces are required ?

- Interfaces allow you to implement common behaviors in different classes that are not related to each other

- Interfaces are used to describe behaviors that are not specific to any particular kind of object, but common to several kind of objects

Why interfaces are required ? (Contd.).

- Defining an interface has the advantage that an interface definition stands apart from any class or class hierarchy

- This makes it possible for any number of independent classes to implement the interface

- Thus, an interface is a means of specifying a consistent specification, the implementation of which can be different across many independent and unrelated classes to suit the respective needs of such classes

- Interfaces reduce coupling between components in your software

Why interfaces are required ? (Contd.).

- Java does not support multiple inheritance
- This is a constraint in class design, as a class cannot achieve the functionality of two or more classes at a time
- Interfaces help us make up for this loss as a class can implement more than one interface at a time
- Thus, interfaces enable you to create richer classes and at the same time the classes need not be related

Interface members

- All the methods that are declared within an interface are always, by default, public and abstract

- Any variable declared within an interface is always, by default, public static and final

Abstract Classes v/s Interfaces

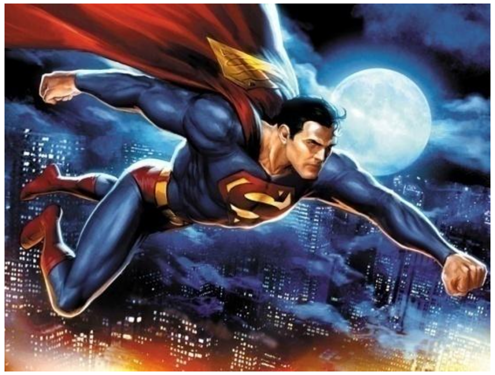
Abstract Classes	Interfaces
Abstract classes can have non-final non-static variables.	Variables declared within an interface are always static and final.
Abstract Classes can have abstract methods as well as concrete methods.	Interfaces can have only method declarations(abstract methods). You cannot define a concrete method.
You can declare any member of an abstract class as private, default, protected or public. Members can also be static.	Interface members are by default public. You cannot have private or protected members. Interface methods cannot be static.
Abstract class is extended by another class using “extends” keyword.	An interface is “implemented” by a java class using “implements” keyword.

Contd..

Abstract Classes v/s Interfaces (Contd.).

Abstract Classes	Interfaces
An abstract class can extend another class and it can implement one or more interfaces.	An interface can extend one or more interfaces but cannot extend a class. It cannot implement an interface.
An abstract class can have constructors defined within it.	You cannot define constructors within an interface.
An abstract class cannot be instantiated using “new” Keyword	An interface cannot be instantiated.
You can execute(invoke) an abstract class, provided it has public static void main(String[] args) method declared within it.	You cannot execute an interface

What will you choose..?



What is the behavior which is common among the entities depicted in the pictures above?

Yes..You are right. All of them can fly.

Requirement : You have to develop 3 classes, Bird, Superman and Aircraft with the condition that all these classes must have a method called fly().

What is the mechanism, using which you can ensure that the method fly() is implemented in all these classes?

An Abstract class or An Interface?

Defining an Interface

- An interface is syntactically similar to a class
- It’s general form is:

```
public interface FirstInterface {
  int addMethod(int x, int y);
  float divMethod(int m, int n);
  void display();
   int VAR1 = 10;
   float VAR2 = 20.65;
  }
```

Implementing Interfaces

- A class implements an interface
- A class can implement more than one interface by giving a comma- separated list of interfaces

```
class MyClass implements FirstInterface{
 public int  addMethod(int a, int b){
  return(a+b);
 }
 public float divMethod(int i, int j){
  return(i/j);
 }
 public void display(){
  System.out.println(“Variable 1 :” +VAR1);
  System.out.println(“Variable 2 :” +VAR2);
 }
}
```

Quiz

Will the following code compile successfully ?

```
interface I1 {
        private int a=100;
        protected void m1();
}

class A1 implements I1 {
        public void m1() {
        System.out.println(“In m1 method”);
        }
}
```

It will throw compilation errors.. Why?

Quiz (Contd.).

Will the following code compile successfully ?

```
interface I1 {
        static int a=100;
        static void m1();
}

class A1 implements I1 {
        public void m1() {
        System.out.println("In m1 method");
        }
}
```

It will throw compilation error.. Why?

Applying Interfaces

Applying Interfaces

- Software development is a process where constant changes are likely to happen
- There can be changes in requirement, changes in design, changes in implementation
- Interfaces support change
- Programming through interfaces helps create software solutions that are reusable, extensible, and maintainable

Applying Interfaces (Contd.).

```
interface IntDemo{
  void display();
}
class classOne implements IntDemo{
  void add(int x, int y){
    System.out.println("The sum is :" +(x+y));
  }
  public void display(){
    System.out.println("Welcome to Interfaces");
  }
}
```

Applying Interfaces (Contd.).

```
class classTwo implements IntDemo{
  void multiply(int i,int j, int k) {
    System.out.println("The result:" +(i*j*k) );
  }
  public void display(){
    System.out.println("Welcome to Java ");
  }
}
class DemoClass{
  public static void main(String args[]) {
    classOne c1= new classOne();
    c1.add(10,20);
    c1.display();
    classTwo c2 = new classTwo();
    c2.multiply(5,10,15);
    c2.display();
  }
}
```

Interface References

- When you create objects, you refer them through the class references. For example :
  - ClassOne c1= new classOne(); /* Here, c1 refers to the object of the class classOne. */
- You can also make the interface variable refer to the objects of the class that implements the interface
- The exact method will be invoked at run time
- It helps us achieve run-time polymorphism

Interface References (Contd.).

```
interface IntDemo{
 void display();
}
class classOne implements IntDemo{
 void add(int x, int y){
   System.out.println("The sum is :" +(x+y));
 }
 public void display(){
   System.out.println("Class one display method
 ");
 }
}
```

Interface References (Contd.).

```
class classTwo implements IntDemo {
  void multiply(int i,int j, int k){
    System.out.println("The result:" +(i*j*k) );
  }
  public void display(){
    System.out.println("Class two display method"
  );
  }
}
class DemoClass{
  public static void main(String args[]){
    IntDemo c1= new classOne();
    c1.display();
    c1 = new classTwo();
    c1.display();
  }
}
```

Extending Interfaces

- Just as classes can be inherited, interfaces can also be inherited
- One interface can extend one or more interfaces using the keyword **extends**
- When you implement an interface that extends another interface, you should provide implementation for all the methods declared within the interface hierarchy

Marker Interface

- An Interface with no method declared in it, is known as Marker Interface

- Marker Interface is provided as a handle by java interpreter to mark a class, so that it can provide special behavior to it at runtime

- Examples of Marker Interfaces :
  - java.lang.Cloneable
  - java.io.Serializable
  - java.rmi.Remote

Quiz

Will the following code compile successfully ?

```
interface I1 {
     int a=100;
     void m1();
}

class A1 extends I1 {
     public void m1() {
     System.out.println("In m1 method");
     }
}
```

It will throw compilation error.. Why?

Quiz (Contd.).

Will the following code compile successfully ?

```
interface I1 {
     int a=100;
     void m1();
}

interface A1 implements I1 {
     public void m2();
}
```

It will throw compilation error.. Why?

Quiz (Contd.).

Will the following code compile successfully ?

```
interface I1 {
     int a=100;
     void m1();
}

interface A1 extends I1 {
     public void m2();
}


class Aimp implements I1 {
     public void m1() {
     System.out.println("In m1 method");
     }
}
```

This code will compile successfully..!

Summary

In this session, you were able to:

- Understand the use of packages
- Use inbuilt java packages
- Create our own packages
- Import existing packages
- Create interfaces
- Understand the relevance and uses of interfaces in java

References

1. Oracle (2012). The Java Tutorials: Creating and Using Packages. Retrieved on April 5, 2012, from, http://docs.oracle.com/javase/tutorial/java/package/packages.html

2. Oracle (2012). The Java Tutorials: Interfaces. Retrieved on April 5, 2012, from, http://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html

3. Schildt, H. Java: *The Complete Reference. J2SETM*. Ed 5. New Delhi: McGraw Hill-Osborne, 2005.

Thank You