This image cannot currently be displayed.

Chapter 9

Package and Interfaces

Declaration

- These slides are made for UIT, BU students only. I am not holding any copy write of it as I had collected these study materials from different books and websites etc. I have not mentioned those to avoid complexity.

Topics

- Packages
  - Defining a Package
  - Creating Package
  - Importing Package
- Interfaces
  - General form
  - Implementing Interfaces
  - Partial Implementation
  - Use of interfaces
  - Interfaces can be extended

Packages

- Packages are java’s way of grouping a variety of classes and/or interfaces together.
- Packages are containers for classes and interfaces.

Defining a Package

- General form of package statement

  package pkg;

- If you omit package statements then class names are put into default package, which has no name.
- Java uses file system directories to store packages.
- myPackage can be stored in the directory myPackage.
- Within a package there may be several package.
- The general form of multi level package statement is

  package pkg1[.pkg2[.pkg3]];

- To call a method under a package and under a class we can use following type of instruction.

double y= java.lang.Math.sqrt(x);

package name   class name   method name

Creating Package

```
// A simple package
package MyPack;
class Balance {
  String name;
  double bal;
  Balance(String n, double b) {
    name = n;
    bal = b;
  }
  void show() {
    if(bal<0)
      System.out.print("-->> ");
    System.out.println(name + ": $" + bal);
  }
}
```

Creating Package

```
class AccountBalance {
 public static void main(String args[]) {
   Balance current[] = new Balance[3];

   current[0] = new Balance("K. J. Fielding", 123.23);
   current[1] = new Balance("Will Tell", 157.02);
   current[2] = new Balance("Tom Jackson", -12.33);

   for(int i=0; i<3; i++) current[i].show();
 }
}
```

Creating Package

- Create a directory by md MyPack (in DOS/command prompt) command.
- Save java file by name AccountBalence.java in MyPack directory.
- Compile the java program by command javac MyPack/AccountBalence.java
- From bin directory of jdk run the java program by java MyPack. AccountBalence.
- You can also run the program by java MyPack/AccountBalence.

Importing Package

- Java includes the import statement to bring certain classes, using only packages into visibility.
- Once imported, a class can be referred to directly, using only its name.
  - import java.util.Date;
  - import java.io.*;
- * indicates that the java compiler should import the entire package.
- It may increase compilation time.
- However it has no effect on the run-time performance or size of your class.
- Java's basic language function is stored in java lang. It is implicitly imported by the compiler.

Importing Package

- If a class with same name exits in two different packages that you import using the * form, the compiler will remain silent, unless you are trying to use one of the classes. In that cases it will generate compile time error and have to explicitly name the class specifying its package.
- When a package is imported, only those items within the package declared as public will be available to non-subclasses in the importing node.

Example

```
package package1;
public class A {
  public void displayA(){
     System.out.println(“Class A");
     }
}
package package2;
public class B {
  public void displayB(){
     System.out.println(“Class B");
     }
}
```

Example

```
import package1.A;
import package2.*;
  class packageTest{
     public static void main(String args[]) {
     A a=new A();
     B b=new B();
     a. displayA();
     b. displayB();
     }
}
```

- Output

```
Class A
Class B
```

Interfaces

- Using the keyword interface, you can abstruct a class from its implementation. That is, using interface, you can specify what a class must do, but not how it does it.
- Interfaces are syntactically similar to classes but they lack instance variables and their methods are defined without any body.
- Once it is defined, any number of classes can implement an interface. Also one class can implement any number of interfaces.
- To implement an interface, a class must create complete set of methods defined by the interface.
- Interfaces are designed to support dynamic method resolution at run time.

Interfaces

- Normally, in order for a method to be called from one class to another, both classes need to be present at compile time so the java compiler can check to ensure that the method signatures are compatible.
- Interfaces are designed to avoid this problem.
- This disconnect the definition of method or set of methods from the inheritance hierarchy.
- Since interfaces are in a different hierarchy from classes, it is possible for classes that are unrelated in terms of the class hierarchy to implement the same interface.
- Interfaces support the concept of multiple inheritance.

General form

```
access interface name{
return-type methodname1(parameter_list);
return-type methodname2(parameter_list);
type final-varname1=value;
type final-varname2=value;
//---
return-type methodnameN(parameter_list);
type final-varnameN=value;
}
```

- The access specifier is either public or not used.
- Variables inside the interfaces are final and static, meaning they cannot be changes by the implementing class. They must also be initialized with a constant value.
- All methods and variables are implicitly public if the interface, itself, is declared as public.

Implementing Interfaces

- Here access is either public or not used. The method that implement an interface must be declared public.

access class classname[extends superclass][implements interface[,inteface]]{

//class-body

}

Implementing Interfaces

```
interface Callback {
  void callback(int param);
}

class Client implements Callback {
  // Implement Callback's interface
  public void callback(int p) {
    System.out.println("callback called with " + p);
  }
}
```

Implementing Interfaces

- You can declare variables as object references that use interface rather than a class type.
- Dynamic lookup of a method at run time incurs a significant overhead when compared with the normal invocation in java, you should be careful not to use intefaces casually in performance critical code.

```
class TestIface {
 public static void main(String args[]) {
  Callback c = new Client();
  c.callback(42);
 }
}
```

- Output

```
callback called with 42
```

Implementing Interfaces

- To demonstrate the polymorphic power of reference see the following program.

```
// Another implementation of Callback.
class AnotherClient implements Callback {
  // Implement Callback's interface
  public void callback(int p) {
    System.out.println("Another version of callback");
    System.out.println("p squared is " + (p*p));
  }
}
```

Implementing Interfaces

```
class TestIface2 {
  public static void main(String args[]) {
    Callback c = new Client();
    AnotherClient ob = new AnotherClient();
    c.callback(42);
    c = ob; // c now refers to AnotherClient object
    c.callback(42);
  }
}
```

- Output

```
callback called with 42
Another version of callback
p squared is 1764
```

Partial Implementation

- If a class includes an interface but does not fully implement the methods defined by that interface, then that class must be declared as abstract.

Use of interfaces

- Interfaces can be used to declare a set of constant that can be used in different classes.
- Multiple inheritance can also be implemented by means of interfaces.

Example

```
class student {
  int rollNumber;
  void getNumber(int n){
     rollNumber=n;
  }
  void putNumber(){
     System.out.println("Roll no:"+ rollNumber);
  }
}
```

Example

```
class Test extends student {
  float part1, part2;
  void getMarks(float m1,float m2){
      part1=m1;
      part2=m2;
  }
  void putMarks(){
      System.out.println(“Marks obtained”);
     System.out.println(“part1=”+ part1);
     System.out.println(“part2=”+ part2);
  }
}
```

Example

```
interface sports {
  final float sportWt=6.0;
  void putWt();
  }
}
```

Example

```
class result extends Test implements Sports{
     float total;
     public void putWt(){
               System.out.println("Sports wt="+ sportWt);
     }
     void display(){
               total=part1+part2+ sportWt;
                putNumber();
                putMarks();
               putWt();
               System.out.println("Total score="+ total);
     }
}
```

Example

```
class Hybrid {
  public static void main(String args[]) {
     Result student1=new Results();
      student1.getNumber(1234);
      student1.getMarks(27.5,33.0);
      student1.display();
     }
}
```

- Output

```
Roll No: 1234
Marks obtained
Part1=27.5
Part2=33
Sports wt = 6
Total score =66.5
```

Interfaces can be extended

```
interface A {
  void meth1();
  void meth2();
}
interface B extends A {
  void meth3();
}
class MyClass implements B{
  //body
}
class IfExtend{
  public static void main(String args[]) {
}
}
```

Example

```
interface car
{
        final int speed=90;
        public void distance();
}
interface bus
{
        final int distance=100;
        public void speed();
 }
```

Example

```
class vehicle implements car,bus
{
        public void distance()
        {
                int distance=speed*100;
                System.out.println("distance travelled is"+distance);
        }
        public void speed()
        {
                int speed=distance/100;
        }
}
```

Example

```
class maindemo
{
        public static void main(String args[])
        {
                System.out.println("Vehicle");
                Vechicle v1=new Vehicle();
                v1.distance();
                v1.speed();
         }
}
```

End of Chapter 9

Questions?