Packages and Interfaces

By,
Hitha Paulson
Assistant Professor, Dept. of Computer Science
LF College, Guruvayoor

What is Package

- Packages are containers for classes
- Package is both naming and visibility control mechanism
- Classes defined within a package are not accessible by the classes defined outside the package
- Package helps to interact classes each other and restrict to communicate with the classes defined outside the package

Defining a Package

- Packages are defined by using **package** command
- It should be the first statement in the program
- All the classes defined in that program will belong to this package
- General Form
  - **package** *pkg_name;*
- Java uses filesystem directories to store packages
- Package name is case-sensitive
- Packages can create as a hierarchy.

Finding Packages and CLASSPATH

- The Location of packages can tell to the compiler or Interpreter by
  - Java uses current working directory as starting-point and hence subdirectory can act as Package directory
  - Use environment variable CLASSPATH to specify the package location
  - Compile and Run by using -classpath option with command

Access Protection

- The way in which the members of class are available outside the class
- Access specifiers and Packages are the means for implementing Access Protection
- Different access specifiers makes class members available in the following regions
  - Subclass in the same package
  - Non-subclass in the same Package
  - Subclass in different packages
  - Classes that are neither in the same package nor subclasses

Access Protection contd..

- The main access specifiers are
  - Private, public, protected and Non-modifier (FRIENDLY)
- Class can have two access specifiers: public and default
  - Public class can access anywhere
  - Default class can access only in same package

Access Protection Table

	Private	No Modifier	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package Subclass	No	Yes	Yes	Yes
Same Package Non-Subclass	No	Yes	Yes	Yes
Different Package Subclass	No	No	Yes	Yes
Different package Non-subclass	No	No	No	Yes

Importing Packages

- All the built-in and user-defined classes are stored in named Packages
- In order to refer a particular class, it is required to use its fully qualified classname
  - Eg: java.io.DataInputStream
- Import statement is used to bring certain classes or entire packages into visibility
- Imported classes can refer without prefixing package name
- Import statement is using after Package statement and before class definition
- General Form
  - import pkg1.[pkg2].(classname | *) ;
- If more than one package contains classes of same name, it is required to use full qualified classname

Interfaces

- Interface is a special type of class designed for inheritance
- An interface is a complete abstract class having final variables and abstract methods
- Interfaces are designed to support dynamic method resolution at runtime
- Interfaces are inherited by using “implements” instead of “extends”
- The subclasses inherited from interface should redefine all the abstract methods in interface
- Interfaces helps to implement multiple inheritance

Interfaces contd . . .

General Format

access_specifier **interface** *interface_name*

```
{
        return-type   method_name1(param-list);
    return-type   method_name2(param-list);
    type   final_var1 = value;
    type   final_var2 = value;
// .......
}
```

Implementing Interfaces

- To implement an interface, include the **implements** clause in a class definition
- General Format

class class_*name* *[***extends** *superclass] [***implements** *interface1 [,interface2, ...]*

{

//class body

//definition of methods in interface

}

- Methods that implements an interface must be declared as **public**

Interface Reference variables

- The methods defined in an interface can also invoke by using reference variables of Interface. [Idea of dynamic method despatch]
- Reference variables must be initialized by using object of implemented class
- But the other members of the implemented class cannot be invoked by this object reference
- Any implementing class declared as abstract can leave the methods in Interface redefined. This type of implementations are called **partial Implementation**
- An Interface can be **extended** from another Interface. Here the implementing class must override all the methods defined in the inheritance chain.