School of Computer Science and IT
JAIN (DEEMED-TO-BE UNIVERSITY)
Department of Bachelor of Computer Applications

Module 3 Introduction to Packages and Exceptions

Chapter 1 Packages and Interfaces

Packages

Definition of Packages

Java packages are defined as a group of similar type of classes, subclasses and interfaces. It provides access protection and name space management which manages the system files and class files.

To create a package is very easy:

- In a Java source file, just insert a package command as the first declaration.
- Any classes declared inside that file will belong to the specified package.
- The package statement defines a namespace in which classes are stored.
- If you delete the package statement, the class titles are put into the default package, which has nameless.
- During the default package is excellent for short, sample programmes, it is inadequate for real applications. Most of the time, you will describe a package for the code.

Definition of Packages

This is the general form of the package statement:

package pkge;

- Here, pkge is the name of the package. For example, the following statement generates a package called MyFirstPackage.

package MyFirstPackage;

- Java uses file system records to save packages. For example, the .class files for whatever classes you declare to be part of MyFirstPackage must be stored in a directory called MyFirstPackage.

- An identity that case is important and the directory name must match the package name accurately.More than one file can include the same package statement. The package statement clearly specifies to which package the classes defined in a file belong.

Definition of Packages

•You can create a hierarchy of packages. To do so, just assign each package name from the one preceding it by use of time.

- **The generic form of a multileveled package statement is given here:**

```
package pkge1[.pkge2[.pkge3]];
```

•A package hierarchy must be reflected in the file system of your Java development system. For example, a package declared as

```
package java.awt.image;
```

- Needs to be stored in java\awt\image in a Windows environment. Be assured to accept your package names carefully. You cannot rename a package without renaming the record in which the classes are saved.

Definition of Packages

Syntax:

Package <package name>;

A simple Package Example:

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

Program To be continued in next slide

```
void show() {
if(balnc<0)
System.out.print("--> ");
System.out.println(name + ": $" + balnc);
}}
class AccountBalance {
public static void main(String args[]) {
Balance current[] = new Balance[3];
 current[0] = new Balance("Jordan", 153.23);
current[1] = new Balance("Hunk", 127.02);
current[2] = new Balance("Mitchel", -19.63);
for(int p=0; p<3; p++) current[p].show();
}}
```

- Call this file AccountBalance.java and put it in a directory called MyFirstPack.Next, execute the file.Be confident that the resulting .class file is also in the MyFirstPackage directory. Then, try executing the AccountBalance class, using the following command line:

java MyFirstPack.AccountBalance

Definition of Packages

Another Simple Example for Package :

```
package MyFirstPackage;
public class FirstPackage
        {
          public static void main (String args [ ])
           {
             System.out.println ("Namaste India");
           }}
```

Output:

Namaste India

In this above example, we are using a package named "MyFirstPackage". Call this file FirstPackage.java and put it in a directory called MyFirstPack.Next, execute the file.Be confident that the resulting .class file is also in the MyFirstPackage directory. Then, try executing the FirstPackage class, using the following command line: java MyFirstPack. FirstPackage.

Access Protection

- Access protection is one of the most important features of Java packages. It can grant access to classes, interface and methods which make them accessible to other members of the same package. This feature gives an ease of modification and provides an ability of internal implementation and an external interface.

- Ease of modification can be achieved through an internal ability (access protection) to change the implementation of the package internally without affecting the code of other packages, which is tied only to the external interface.

The interaction between classes and packages, Java addresses four sections of distinctness for class members in packages:

- Subclasses in the same kind of package.
- Non-subclasses in the same kind of package.
- Subclasses in different kind of packages
- Classes that are neither in the similar package nor subclasses

Access Protection

Access protection in Java Packages:

The three access specifiers/modifiers in the package. These are:

1. Private
2. Default
3. Protected
4. Public

- Private, public, and protected, produce a variation of ways to perform the many levels of access required by these categories. Anything revealed public can be obtained from anyplace.
- Anything published privately cannot be seen outside of its class. When a portion does not produce an unambiguous access specification, it is visible to subclasses as well as to other classes in the same package

Access Protection

Let's disclose about the access modifiers and their permission to use within the programme is shown in the below table.

Access Modifiers	Access inside class	Access inside package	Access by sub class	Outside package
Private	Yes	No	No	No
Default	Yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	No	Yes	Yes

Access Protection

Let's see the following source code for the package, Demo2. The two classes (A and B)represented in Package, Demo2 cover the other two conditions that are affected by access control.A's variables besides for msg(), the variable declared with the default security. Eventually, class B has entrance to only one object, Obj.msg (); which was declared public.

Sample Program for Public Access Modifier:

```
Package Demo2;
public class A
   {
public void msg( )
System.out.println ("My First Package");
   }
```

Program To be continued in next slide

```
Packagemydemo;
Import demo.*;
Class B
{
Public staic void main (string args [ ])
{
  A obj = new A( );
  Obj.msg ( );
 }}
```

Output:

My First Package

Import Packages in Java

- Given that packages exist and are a useful mechanism for compartmentalising diverse classes from each other, it is easy to see why all of the built-in Java classes are stored in packages.There are no core Java classes in the default unnamed package; all of the standard classes are stored in some named package

- Since classes within packages must be complete with their package name or names, it could become tedious to type in the long dot-separated Package path names for each class you require to use. For this reason, Java combines the import statement to bring several classes, or entire packages, into clarity.

- Once imported, a class can be transferred to directly, using only its name. The import statement is a convenience to the programmer and is not technically needed to write a complete Java programme.

Import Packages in Java

• Import statements occur immediately following the package statement and earlier any class representations. It is the simple general form of the import statement:

import pkge1[.pkge2].(classname|*);

• Here, pkge1 is the name of an excellent package, and pkge2 is the name of a subordinate package inside the outer package separated by a dot (.). There is no practical limit on the depth of a package hierarchy, except that imposed by the file system.

• Ultimately, you specify either an explicit class name or a star (*), which indicates that the Java compiler should import the complete package. This code piece shows both forms in use:

import java.util.Date;

import java.io.*;

Import Packages in Java

- The '*'(star) form can increase compilation time. It is a good idea to name the classes explicitly that coder wants to use rather than importing entire packages. But, the '*' (star) form has no effect on the run-time display/performance or size of the classes.

- All of the formal Java classes included with Java are stored in a package called java.java.lang is the basic language functions stores in packages inside of the java package. Java is useless without significant of the functionality in 'java.lang', The compiler implicitly imports it for all programmes.

This is similar to the following sequence being at the top of all of the java programs:

import java.lang.*;

- Wherever a class with the same name exists in two separate packages that you import using the '*' form, the compiler device will remain quiet unless you examine to use one of those classes.

Import Packages in Java

For example, this fragment uses an import statement:

```
import java.util.*;
class ImportDate extends Date
{
}
```

The similar example without the import statement looks like this:

```
class ImportDate extends java.util.Date
{
}
```

- For instances, if you want the Balance class of the package, MyFirstPack showed earlier to be available as a stand-alone class for general use outside of MyFirstPack, then you will need to declare it as public and put it into its file, as shown here:

package MyFirstPack;

Import Packages in Java

/* Now, the Balance class, its constructor, and it show() are public.It means that non-subclass code can use constructors outside their package. */

```
public class Balance {
 String name;
double balnc;
public Balance(String m, double b) {
name = m;
balnc= b;
}
public void show() {
if(balnc<0)
System.out.print("--> ");
System.out.printing(name + ": $" + balnc);
}
}
```

Import Packages in Java

- **Note-The output for above code is based on** "The Balance class is immediately public, as you can see in above code. Also, its constructor and its show( ) method are public, too. It means that they can be accessed by any code outside the MyFirstPack package.

For example, here TestBalance imports MyFirstPack and is then able to make use of the Balance class:

```
import MyFirstPack.*;
class Balance {
public static void main(String args[]) {
Balance testing = new Balance("Jordan Hunk", 45.65);
testing .show();
}}
```

- **Note for Output :**As an experiment, remove the public specifier from the Balance class and then try compiling TestBalance. As explained, ***errors will result."***

Interfaces

Defining an interface

An Interface is defined as a blueprint of class. It looks similar to that of class, but it is not a class. Like class, it can have methods and variables but the methods declared in an interface are by default abstract. The variables declared in an interface are public, static and final by default.

A keyword 'interface' is used to declare an interface.

Syntax:

```
Interface MyInterface
{
//statements;
//statements;
public void method1( );
public void method2( );
}
```

Defining an interface

Declaration of an interface:

An interface is described much like a class. This is the general form of an interface:

```
access interface name {
return-type method-nameFirst(parameter-List);
return-type method-nameSecond(parameter-List);
type final-varname1 = value;
type final-varname2 = value;
// ...
return-type method-name(parameter-list);
type final-varname = value;
}
```

Defining an interface

- When no entrance specifier is included, then access failure results and the interface will be available only to other segments of the package in which it is held. When it is revealed as public, the interface can be used by another code.

- The interface must be the only public interface declared in the file, and the file must have the same name as the interface. The name is the name of the interface, and can be any valid identifier. They are, essentially, abstract methods; within an interface.Each class that includes an interface must implement all of the methods.

- Java Variables can be declared within the interface declarations in the code. They are inherently final and static, meaning the implementing class can not change them. They must also be initialized. It declares a simple interface that contains one method called callback( ) that takes a single integer parameter.

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

Implementing Interface

- An interface cannot implement another interface. If required, it has to extend the other interface, or it has to designate a set of abstract methods for classes to implement. If a class implements an interface, it inherits all the abstract methods declared within its scope.

Sample Program:

```
public Interface P1
{
public void method1();
}
public Interface P2 extends P1 //provides the implementation of all the methods of interface P1
{
public void method2();
}
public class A implements P2
{
```

Program To be continued in next slide

```
public void method1()
{
//statements
}
public void method2( ){
//statements}
```

- An interface has been defined once, one or more than one class can perform that interface.

To implement an interface, include the implements clause in a class definition, and then create the methods defined by the interface. The general form of a class that includes the implements sentence looks like this:

```
class classname [extends superclass] [implements interface [,interface...]]
{
// class-body
}
```

Implementing Interface

Let's see a small example class that implements the Callback interface shown earlier:

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

Notice that callback( ) is declared using the public access specifier.

Note –When you implement an interface method,it must be declared as 'public'.

Implementing Interface

• It is both permissible & standard for classes that implement interfaces to define additional members of their own. For example, the following version of Client implements callback( ) and adds the method nonIfaceMeth( ):

```
class Customer implements Callback
{
// Implement Callback's interface
public void callback(int x)
{
System.out.println("callback called with " + x);
}
void nonIfaceMeth() {
System.out.println("To implement interfaces's classes " +"can be define other members, too.");
}
}
```

Program To be continued in next slide

The resulting example calls the callback( ) method via an interface reference variable:

```
class Testface
{
public static void main(String args[])
{
Callback b = new Client();
b.callback(24);
}
}
```

Output of this program :

callback called with 24

Collection Framework

Collection Framework

The most basic Java interface is **collection**. Java provides some classes like Dictionary, vector, stack and properties for some essential purpose to store and manipulate groups of objects. To meet these set of essential goals collection framework is used. Some features of collection frame work are:

- It consists of highly efficient collection fundamentals like dynamic arrays, linked lists, trees and hash tables.
- It has to be highly performable because it must have an ability to allow different types of collections to work with a high degree of interoperability.
- It should be adaptable to extend to the different type of collections

Collection Framework

- The Collection interface is the structure upon which the Collections Framework is built because it must be implemented by any class that determine a collection. Collection is a generic interface that has this declaration:

interface Collection<P>

- Here, P defines the type of objects that the collection will hold. The collection extends the Iterable interface. It means that all groups can be sequenced through by use of the for-each style for loop.

Advantages of Collection Framework:

- It reduces programming effort and increases the performance.
- Provides interoperability between unrelated application programme interfaces.
- It reduces the effort of learning, designing and implementing application programme interface.
- It is reusable any number of time.

Summary

- ✓ The package is a collection of similar type of classes.
- ✓ Access protection is one of the most important features of Java packages.
- ✓ Keyword 'interface' is used to declare an interface.
- ✓ An interface cannot implement another interface.
- ✓ The most basic Java interface is a collection.