Java programming
MODULE-2
(Packages)

By

Dr. N. Ramana
Associate Professor
UCE(KSM), Kakatiya University

Course Outcomes

- Able to solve real world problems using OOP techniques.
- Able to understand the use of abstract classes.
- Able to solve problems using java collection framework and I/o classes.
- Able to develop multithreaded applications with synchronization.
- Able to develop applets for web applications.
- Able to design GUI based applications

Agenda

- ✓ Defining a Package,
- ✓ Access protection
- ✓ importing packages.
- ✓ Interfaces- defining an interface
- ✓ implementing interfaces
- ✓ Nested interfaces
- ✓ variables in interfaces and extending interfaces.

Defining Packages in java

- In java, a package is a container of classes, interfaces, and sub-packages. We may think of it as a folder in a file directory.
- We use the packages to avoid naming conflicts and to organize project-related classes, interfaces, and sub-packages into a bundle.
- In java, the packages have divided into two types.

    1. Built-in Packages
    2. User-defined Packages

Built-in Packages

- The built-in packages are the packages from java API. The Java API is a library of pre-defined classes, interfaces, and sub-packages. The built-in packages were included in the JDK.
- There are many built-in packages in java, few of them are as java, lang, io, util, awt, javax, swing, net, sql, etc.
- We need to import the built-in packages to use them in our program. To import a package, we use the import statement.

User-defined Packages

The user-defined packages are the packages created by the user. User is free to create their own packages.

Defining Packages in java

Syntax

```
package packageName;
```

Example

```
package myPackage;
public class DefiningPackage
{
  public static void main(String[] args)
  {
  System.out.println("This class belongs to myPackage.");
   }
}
```

- ✓ Now, save the above code in a file **DefiningPackage.java**, and compile it using the following command.

  javac -d . DefiningPackage.java

- ✓ The above command creates a directory with the package name ***myPackage***, and the ***DefiningPackage.class*** is saved into it.
- ✓ Run the program use the following command. java myPackage.DefiningPackage

Access protection in java packages

- In java, the access modifiers define the accessibility of the class and its members. For example, private members are accessible within the same class members only. Java has four access modifiers, and they are default, private, protected, and public.
- In java, the package is a container of classes, sub-classes, interfaces, and sub-packages. The class acts as a container of data and methods. So, the access modifier decides the accessibility of class members across the different packages.
- In java, the accessibility of the members of a class or interface depends on its access specifiers.
- The following table provides information about the visibility of both data members and methods.

Access control for members of class and interface in java

Accessibility / Access Specifier	Same Class	Same Package		Other Package	
		Child class	Non-child class	Child class	Non-child class
Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	Yes	No
Default	Yes	Yes	Yes	No	No
Private	Yes	No	No	No	No

Importing Packages in java

- In java, the **import** keyword used to import built-in and user-defined packages. When a package has imported, we can refer to all the classes of that package using their name directly.
- The import statement must be after the package statement, and before any other statement.
- Using an import statement, we may import a specific class or all the classes from a package.

Importing specific class

- Using an importing statement, we can import a specific class. The following syntax is employed to import a specific class.

Syntax

```
import packageName.ClassName;
```

Example

```
package myPackage;
import java.util.Scanner;
public class ImportingExample
{
  public static void main(String[] args)
  {
    Scanner read = new Scanner(System.in);
    int i = read.nextInt(); System.out.println("You have entered a number " + i);
   }
 }
```

Importing all the classes

- Using an importing statement, we can import all the classes of a package. To import all the classes of the package, we use * symbol. The following syntax is employed to import all the classes of a package.

Syntax

```
import packageName.*;
```

Example

```
import java.util.*;
```

- The above import statement util is a sub-package of java package. It imports all the classes of util package only, but not classes of java package.

Defining an interface in java

- In java, an **interface** is similar to a class, but it contains abstract methods and static final variables only.
- We may think of an interface as a completely abstract class. None of the methods in the interface has an implementation, and all the variables in the interface are constants.
- All the methods of an interface, implemented by the class that implements it.
- The interface in java enables java to support multiple-inheritance. An interface may extend only one interface, but a class may implement any number of interfaces.
- We use the keyword interface to define an interface. All the members of an interface are public by default. The following is the syntax for defining an interface.

```
interface InterfaceName
  {
      ... members declaration; ... }
```

Implementing an Interface in java

- In java, an **interface** is implemented by a class. The class that implements an interface must provide code for all the methods defined in the interface, otherwise, it must be defined as an abstract class.
- The class uses a keyword **implements** to implement an interface. A class can implement any number of interfaces. When a class wants to implement more than one interface, we use the **implements** keyword is followed by a comma-separated list of the interfaces implemented by the class.

Syntax

```
class className implements InterfaceName
{
    ... body-of-the-class ... }
```

Implementing multiple Interfaces

- When a class wants to implement more than one interface, we use the **implements** keyword is followed by a comma-separated list of the interfaces implemented by the class.
- The following is the syntax for defining a class that implements multiple interfaces.

Syntax

```
class className implements InterfaceName1, InterfaceName2, ...
{
    ... body-of-the-class ...
}
```

Nested Interfaces in java

- In java, an interface may be defined inside another interface, and also inside a class. The interface that defined inside another interface or a class is known as nested interface. The nested interface is also referred as inner interface.
- The nested interface cannot be accessed directly. We can only access the nested interface by using outer interface or outer class name followed by dot(.), followed by the nested interface name.

Nested interface inside another interface

- The nested interface that defined inside another interface must be accessed as OuterInterface.InnerInterface.

Nested interface inside a class

- The nested interface that defined inside a class must be accessed as ClassName.InnerInterface.

Variables in Java Interfaces

In java, an interface is a completely abstract class. An interface is a container of abstract methods and static final variables. The interface contains the static final variables. The variables defined in an interface can not be modified by the class that implements the interface, but it may use as it defined in the interface.

Example

```
interface SampleInterface
{
 int UPPER_LIMIT = 100; //int LOWER_LIMIT;
// Error - must be initialised
 }
 public class InterfaceVariablesExample implements SampleInterface
{
 public static void main(String[] args)
 {
System.out.println("UPPER LIMIT = " + UPPER_LIMIT); // UPPER_LIMIT = 150;
 // Can not be modified
 }
 }
```

Extending an Interface in java

- In java, an interface can extend another interface. When an interface wants to extend another interface, it uses the keyword extends.
- The interface that extends another interface has its own members and all the members defined in its parent interface too.
- The class which implements a child interface needs to provide code for the methods defined in both child and parent interfaces, otherwise, it needs to be defined as abstract class.

Example

```
interface ParentInterface
  {
   void parentMethod();
  }
```

```
interface ChildInterface extends ParentInterface
{
 void childMethod();
}
 class ImplementingClass implements ChildInterface
{
public void childMethod()
{
 System.out.println("Child Interface method!!");
 }
public void parentMethod()
{
System.out.println("Parent Interface mehtod!");
 }
 }
 public class ExtendingAnInterface
{
public static void main(String[] args)
 {
 ImplementingClass obj = new ImplementingClass();
obj.childMethod();
obj.parentMethod(); } }
```

Questions

1. What is a package in Java?
2. How do you define a package in a Java program?
3. Why are packages used in Java?
4. What is package-private access in Java?
5. How does the protected access modifier work in the context of inheritance?
6. What is an interface in Java?
7. How do you define an interface in Java?
8. What are the key differences between an interface and an abstract class?
9. How do you import a package in a Java program?
10. What is the difference between importing a specific class and importing all classes in a package?