Lesson 9.2

User Defined Package

By the end of this lesson you will learn:

- What is Package
- Why We Need Package
- Types of Package
- Define a Package
- Hierarchy of Packages
- Package Naming Convention
- Access Protection
- Importing a Package Member

What is Package

A **package** is a grouping of related types providing access protection and name space management. Java provides this mechanism for partitioning the class name space into more manageable chunks. We can define classes inside a package that are not accessible by code outside that package. We can also define class members that are exposed only to other members of the same package. This allows your classes to have intimate knowledge of each other, but not expose that knowledge to the rest of the world.

This is the general form of the package statement:

```
package pkg;
```

Here **package** is the keyword and **pkg** is the name of the pacakage.

Why We Need Package

1. **Preventing naming conflicts.** Till now, we were using **default package**, which has no name. That is why, a unique name had to be used for each class to avoid name collisions. For example, we cannot declare employee class twice in default package or **unnamed package**. But using package we can keep both employee class in separate package.
2. Making searching/locating and usage of classes and interfaces easier.
3. **Providing controlled access**. Protected and default have package level access control. A protected member is accessible by classes in the same package and its

subclasses. A default member (without any access specifier) is accessible by classes in the same package only.

Types of Package

There are two types of packages:

1. **Built-in package**: Already defined packages like java.io.*, java.lang.* etc. are known as built-in packages.
2. **User defined package**: The package we create is called user-defined package.

Define a Package

Follow the below steps to define a package:

- Include a **package** command as the first statement in a Java source file.
  For example:

```
package greeting;

public class HelloWorld {
     public static void main(String[] args) {
          System.out.prinlnt("Hello World");
     }
}
```

  So, **greeting** is our package name.
- Java uses **file system directories** to store packages. So, the **.class** files for any classes (in the above example, it is HelloWorld.class) we declare to be part of **greeting** must be stored in a directory called **greeting.** So, our compiled HelloWorld.class file needs to be moved to greeting directory.

Hierarchy of Packages

It is possible to create hierarchy of packages. The general form of a multileveled package statement is shown here:

```
package pkg1[.pkg2[.pkg3]];
```

For example:

```
package demo.greeting;

public class HelloWorld {
     public static void main(String[] args) {
          System.out.prinlnt("Hello World");
     }
}
```

As java uses **file system directories** to store packages, so this hierarchy must be reflected in the file system of your Java development system. We must create a folder **demo.** Under demo folder we must create **greeting** folder. In that folder we must keep our compiled HelloWorld.class file.

Package Naming Convention

- Package names are written in all **lower case** to avoid conflict with the names of classes or interfaces.
- Companies use their reversed Internet domain name to begin their package names—for example, ***com.example.mypackage*** for a package named ***mypackage*** created by a programmer at ***example.com***.
- Packages in the Java language itself begin with **java**. or **javax**.

Importing a Package Member

There are 3 different ways to refer to any class that is present in a different package.

1. **Using fully qualified name:** For this approach, there is no need to use the import statement. But must use the fully qualified name every time when accessing the class or the interface.

```
// HelloWorld.java
package demo.greeting;

public class HelloWorld {
     public void doGreeting() {
          System.out.prinlnt("Hello World");
     }
}

// Main.java
package demo.main;

public class Main {
     public static void main(String[] args) {
          demo.greeting.HelloWorld hw =
               new demo.greeting.HelloWorld();
          hw.doGreeting();
     }
}
```

Since classes within packages must be fully qualified with their package name or names, it could become tedious to type in the long dot-separated package path name for every class you want to use. For this reason, Java includes the **import** statement to bring certain classes, or entire packages, into visibility.

2. **Using import statement:** In a Java source file, import statements occur immediately following the package statement (if it exists) and before any class definitions. This is the general form of the import statement:

```
import pkg1 [.pkg2].(classname | *)
```

Using import statement in the last example:

```
// HelloWorld.java
package demo.greeting;

public class HelloWorld {
     public void doGreeting() {
          System.out.prinlnt("Hello World");
     }
}

// Main.java
package demo.main;
import demo.greeting.HelloWorld;

public class Main {
     public static void main(String[] args) {
          HelloWorld hw = new HelloWorld();
          hw.doGreeting();
     }
}
```

As we are using import statement, we do not need to provide full qualified name for every class.

3. **Using import all:** If you use ***packagename.*,*** then all the classes and interfaces of this package will be accessible but the classes and interface inside the subpackages will not be available for use.

   See the following example:

```
// HelloWorld.java
package demo.greeting;

public class HelloWorld {
     public void doGreeting() {
          System.out.prinlnt("Hello World");
     }
}

// HelloWorld.java
package demo.greeting;

public class Test {
     public void doGreeting() {
          System.out.prinlnt("Test");
     }
}


// Main.java
package demo.main;
import demo.greeting.*;

public class Main {
     public static void main(String[] args) {
          HelloWorld hw = new HelloWorld();
          hw.doGreeting();
          Test t = new Test();
          t.doGreeting();
     }
}
```

In the above example, **demo.greeting.*** brings both the **HelloWorld** and **Test** in visibility of **Main** class.

Exercise

1. Extend the below code to proof the sentence

   "After importing a complete package, the classes and interfaces inside the subpackages will not be available for use"

```
// HelloWorld.java
package demo.greeting;

public class HelloWorld {
     public void doGreeting() {
          System.out.prinlnt(“Hello World”);
     }
}

// HelloWorld.java
package demo.greeting;

public class Test {
     public void doGreeting() {
          System.out.prinlnt(“Test”);
     }
}


// Main.java
package demo.main;
import demo.greeting.*;

public class Main {
     public static void main(String[] args) {
          HelloWorld hw = new HelloWorld();
          hw.doGreeting();
          Test t = new Test();
          t.doGreeting();
     }
}
```