JAVA PACKAGE

By :

Subodh Sharma

SUBODH

Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.

Types of packages in Java

Packages are divided into two categories:

- Built-in Packages (packages from the Java API)
- User-defined Packages (create your own packages)

Built-in Packages

The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment.

Ex.

- java. lang.
- java.io.
- java. util.
- java. applet.
- java. awt.
- java.net.

The library contains components for managing input, database programming, and much much more.

The library is divided into **packages** and **classes**. Meaning you can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package.

Syntax

```
import packageName.Class;   // Import a single class

import packageName.*;   // Import the whole package
```

Import a Class

If you find a class you want to use, for example, the `Scanner` class, **which is used to get user input**, write the following code:

Example

```
import java.util.Scanner;
```

In the example above, `java.util` is a package, while `Scanner` is a class of the `java.util` package.

To use the `Scanner` class, create an object of the class and use any of the available methods found in the `Scanner` class documentation. In our example, we will use the `nextLine()` method, which is used to read a complete line:

Example

Using the `Scanner` class to get user input:

```
import java.util.Scanner;

class MyClass {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Enter username");

    String userName = myObj.nextLine();
    System.out.println("Username is: " + userName);
  }
}
```

Import a Package

There are many packages to choose from. In the previous example, we used the `Scanner` class from the `java.util` package. This package also contains date and time facilities, random-number generator and other utility classes.
To import a whole package, end the sentence with an asterisk sign (`*`). The following example will import ALL the classes in the `java.util` package:

Example

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

User-defined Packages

To create your own package, you need to understand that Java uses a file system directory to store them. Just like folders on your computer:

Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as M.java

package iet;
public class M
{
  public static void main(String args[])
  {
   System.out.println("Welcome to package");
  }
}
```

How to compile java package

javac -d directory javafilename
The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder.

How to run java package program

You need to use fully qualified name e.g. iet.M etc to run the class.

To Compile: javac -d . M.java
To Run: java iet.M

How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

*1) Using packagename.**

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
//save by Alpha.java
package iet;
public class Alpha{
  public void show()
  {
   System.out.println("Fn:Show() :Class Alpha :Package iet");
  }
}
//save by Beta.java
package iet;
public class Beta{
  public void show()
  {
   System.out.println("Fn:Show() :Class Beta :Package iet");
  }
}

import iet.*;

class M
{
  public static void main(String args[])
  {
   Alpha obj_a = new Alpha();
   Obj_a.show();

   Beta obj_b = new Beta();
   Obj_b.show();
```

```
  }
}
```

2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
    //save by Alpha.java
    package iet;
    public class Alpha
    {
      public void show()
      {
       System.out.println("Fn:Show() :Class Alpha :Package iet");
      }
    }
//save by Beta.java
    package iet;
    public class Beta
    {
      public void show()
      {
       System.out.println("Fn:Show() :Class Beta :Package iet");
      }
    }

import iet.Alpha;
import iet.Beta;

 class M
{
   public static void main(String args[])
    {
     Alpha obj_a = new Alpha();
     Obj_a.show();

     Beta obj_b = new Beta();
     Obj_b.show();

  }
```

```
}
```

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

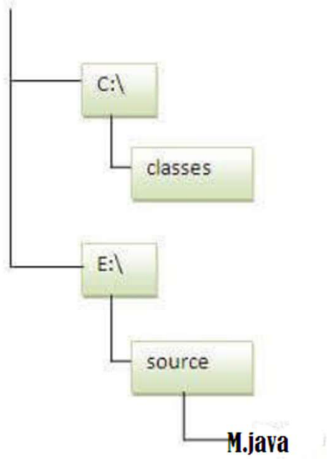
It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

```
//save by Alpha.java
package iet;
public class Alpha
  {
  public void show()
  {
   System.out.println("Fn:Show() :Class Alpha :Package iet");
  }
}

 class M
{
  public static void main(String args[])
  {
   iet.Alpha obj_a = new iet.Alpha();
   Obj_a.show();
  }
}
```

How to send the class file to another directory or drive?



There is a scenario, I want to put the class file of M.java source file in classes folder of c: drive. For example:

```
//save as M.java

package iet;
public class M
{
```

```
    public static void main(String args[])
    {
     System.out.println("Welcome to package");
    }
}
```

To Compile:

e:\sources> javac -d c:\classes M.java

To Run:

To run this program from e:\source directory, you need to set classpath of the directory where the class file resides.

e:\sources> set classpath=c:\classes;.;

e:\sources> java mypack.Simple

Another way to run this program by -classpath switch of java:

The -classpath switch can be used with javac and java tool.

To run this program from e:\source directory, you can use -classpath switch of java that tells where to look for class file. For example:

e:\sources> java -classpath c:\classes mypack.Simple

```
Output:Welcome to package
```

Ways to load the class files or jar files

There are two ways to load the class files temporary and permanent.

- Temporary
  - By setting the classpath in the command prompt
  - By -classpath switch
- Permanent
  - By setting the classpath in the environment variables
  - By creating the jar file, that contains all the class files, and copying the jar file in the jre/lib/ext folder.

Access Modifiers in Java

1. Private access modifier
2. Role of private constructor
3. Default access modifier
4. Protected access modifier
5. Public access modifier
6. Access Modifier with Method Overriding

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

1) Private

The private access modifier is accessible only within the class.

Simple example of private access modifier

In this example, we have created two classes Alpha and M. Alpha class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

```
class Alpha{
private int data=40;
private void msg()
  {System.out.println("Hello java");
  }
}

public class M
{
 public static void main(String args[])
{
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

Role of Private Constructor

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

```
class Alpha{
private Alpha()   {}//private constructor
void show()
  {System.out.println("Hello java");}
  }

public class M{
 public static void main(String args[]){
  Alpha obj=new Alpha();       //Compile Time Error
 }
}
```

Note: A class cannot be private or protected except nested class.

2) Default

If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

Example of default access modifier

In this example, we have created two packages iet and mypack. We are accessing the Alpha class from outside its package, since Alpha class is not public, so it cannot be accessed from outside the package i.e. mypack.

```
//save by Alpha.java
package iet;
class Alpha{
  void show()    {System.out.println("Hello");   }
}
//save by M.java
package mypack;
import iet.*;
class M{
  public static void main(String args[])    {
      Alpha obj = new Alpha();//Compile Time Error
   obj.show();//Compile Time Error
  } }
```

In the above example, the scope of class Alpha and its method show() is default so it cannot be accessed from outside the package.

3) Protected

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifer.

Example of protected access modifier

In this example, we have created the two packages iet and mypack. The Alpha class of iet package is public, so can be accessed from outside the package. But show() method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
//save by Alpha.java
package pack;
public class Alpha
 {
   protected void show()
    {System.out.println("Hello");
    }
}

//save by M.java
package mypack;
import iet.*;

class M extends Alpha{
  public static void main(String args[])
   {
     M obj = new M();
      obj.show();
  }
}
   Output:Hello
```

4) Public : **The** public access modifier **is accessible everywhere. It has the widest scope among all other modifiers.**

Example of public access modifier

```
//save by Alpha.java
  package iet;
public class Alpha
{
public void show()
   {System.out.println("Hello");
   }
}
//save by M.java
  package mypack;
  import pack.*;

class M
{
  public static void main(String args[])
  {   Alpha obj = new Alpha();
      obj.show();
  }
}
```

```
Output:Hello
```

Java Access Modifiers with Method Overriding

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

```
class Alpha
{
protected void show()
  {System.out.println("Hello java");  }
}
 public class M extends Alpha
{
    void show()
   {System.out.println("Hello java");}//C.T.Error

  public static void main(String args[])
   {     M obj=new M();
         Obj.show();
   }
}
```