What are Packages in java and how to use them?

Packages in Java is a mechanism to encapsulate a group of classes, interfaces and sub packages. Many implementations of Java use a hierarchical file system to manage source and class files. It is easy to organize class files into packages. All we need to do is put related class files in the same directory, give the directory a name that relates to the purpose of the classes, and add a line to the top of each class file that declares the package name, which is the same as the directory name where they reside.

In java there are already many predefined packages that we use while programming.

For example: java.lang, java.io, java.util etc.
However one of the most useful features of java is that we can define our own packages

Advantages of using a package

Before discussing how to use them Let see why we should use packages.

- **Reusability:** Reusability of code is one of the most important requirements in the software industry. Reusability saves time, effort and also ensures consistency. A class once developed can be reused by any number of programs wishing to incorporate the class in that particular program.
- **Easy to locate the files.**
- In real life situation there may arise scenarios where we need to define files of the same name. This may lead to “name-space collisions”. Packages are a way of avoiding “name-space collisions”.

Types of package:

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

Defining a Package:
This statement should be used in the beginning of the program to include that program in that particular package.

package <package name>;

Example:

```
package tools;
public class Hammer {
  public void id ()
  {
     System.out.println ("Hammer");
  }
}
```

Points to remember:
1. At most one package declaration can appear in a source file.
2. The package declaration must be the first statement in the unit.

Naming conventions:
A global naming scheme has been proposed to use the internet domain names to uniquely identify packages. Companies use their reversed Internet domain name in their package names, like this:
com.company.packageName

How to Use a Package:

1. We can call it by its full name. For instance,

com.myPackage1.myPackage2 myNewClass = new com.myPackage1.myPackage2();

However this method is very time consuming. So normally we use the second method.

2. We use the "import" keyword to access packages. If you say import com.myPackage1.myPackage2, then every time we type "myPackage2", the compiler will understand that we mean com.myPackage1.myPackage2. So we can do:

```
import com.myPackage1.myPackage2;
class myClass {
  myPackage2 myNewClass= new myPackage2 ();
  …
  …
  …
}
```

There are two ways of importing a package:
Importing only a single member of the package

```
//here 'subclass' is a java file in myPackage2
import com.myPackage1.myPackage2.subClass;
class myClass {
  subClass myNewClass= new subClass();
  …
  …
  …
}
```

Importing all members of a package.

```
import com.myPackage1.*;
import java.sql.* ;
```

Also, when we use *, only the classes in the package referred are imported, and not the classes in the sub package.

The Java runtime automatically imports two entire packages by default:
The java.lang package and the current package by default (the classes in the current folder/directory).

Points to remember:

1. Sometimes class name conflict may occur. For example:

There are two packages myPackage1 and myPackage2.Both of these packages contains a class with the same name, let it be myClass.java. Now both this packages are imported by some other class.

import myPackage1.*;
import myPackage2.*;

This will cause compiler error. To avoid these naming conflicts in such a situation, we have to be more specific and use the member's qualified name to indicate exactly which myClass.java class we want:

myPackage1.myClass myNewClass1 = new myPackage1.myClass ();
myPackage2.myClass myNewClass2 = new myPackage2.myClass ();

2. While creating a package, which needs some other packages to be imported, the package statement should be the first statement of the program, followed by the import statement.

Compiling packages in java:

The java compiler can place the byte codes in a directory that corresponds to the package declaration of the compilation unit. The java byte code for all the classes(and interfaces) specified in the source files myClass1.java and myClass2.java will be placed in the directory named myPackage1/myPackage2 , as these sources have the following package declaration

package myPackage1.myPackage2;

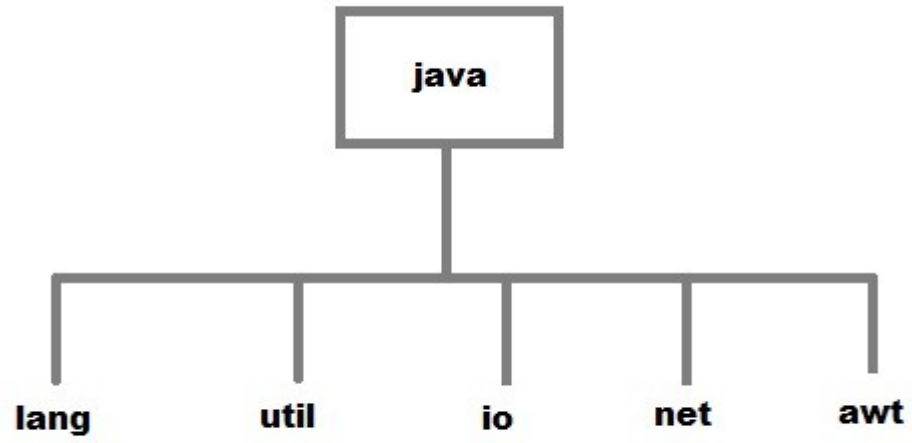
Java Package

Package are used in Java, in-order to avoid name conflicts and to control access of class, interface and enumeration etc. A package can be defined as a group of similar types of classes, interface, enumeration and sub-package. Using package it becomes easier to locate the related classes.

Package are categorized into two forms

- Built-in Package:-Existing Java package for example java.lang, java.util etc.

- User-defined-package:- Java package created by user to categorized classes and interface



Creating a package

Creating a package in java is quite easy. Simply include a package command followed by name of the package as the first statement in java source file.

```
package mypack;
public class employee
{
 ...statement;
}
```

The above statement create a package called **mypack**.

Java uses file system directory to store package. For example the .class for any classes you to define to be part of **mypack** package must be stored in a directory called mypack

---

Example of package creation

```
package mypack;
class Book
{
 String bookname;
 String author;
 Book(String b, String c)
 {
```

```
  this.bookname = b;
  this.author = c;
 }
 public void show()
 {
  System.out.println(bookname+" "+ author);
 }
}

class test
{
 public static void main(String[] args)
 {
  Book bk = new Book("java","Herbert");
  bk.show();
 }
}
```

To run this program :

- create a directory under your current working development directory(i.e. JDK directory), name it as **mypack**.
- compile the source file
- Put the class file into the directory you have created.
- Execute the program from development directory.

NOTE : Development directory is the directory where your JDK is install.

Uses of java package

Package is a way to organize files in java, it is used when a project consists of multiple modules. It also helps resolve naming conflicts. Package's access level also allows you to protect data from being used by the non-authorized classes.

---

import keyword

import keyword is used to import built-in and user-defined packages into your java source file. So that your class can refer to a class that is in another package by directly using its name.

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

1. **Using fully qualified name** (But this is not a good practice.)

   Example :

```
class MyDate extends java.util.Date
{
 //statement;
}
```

2. **import the only class you want to use.**

   Example :

```
import java.util.Date;
class MyDate extends Date
{
 //statement.
}
```

3. **import all the classes from the particular package**

   Example :

```
import java.util.*;
class MyDate extends Date
{
//statement;
}
```

---

import statement is kept after the package statement.

Example :

```
package mypack;
import java.util.*;
```

But if you are not creating any package then import statement will be the first statement of your java source file.

Static import

static import is a feature that expands the capabilities of **import** keyword. It is used to import **static** member of a class. We all know that static member are referred in association with its class name outside the class. Using **static import**, it is possible to refer to the static member directly without its class name. There are two general form of static import statement.

- The first form of **static import** statement, import only a single static member of a class

  Syntax

import static *package.class-name.static-member-name;*

Example

import static java.lang.Math.sqrt; //importing static method **sqrt** of **Math** class

- The second form of **static import** statement,imports all the static member of a class

Syntax

import static *package.class-type-name.*;*

Example

import static java.lang.Math.*; //importing all static member of **Math** class

---

Example without using static import

```
public class Test
{
   public static void main(String[] args)
   {
      System.out.println(Math.sqrt(144));
   }
}
```

Output :

12

Example using static import

```
import static java.lang.Math.*;
public class Test
{
   public static void main(String[] args)
   {
      System.out.println(sqrt(144));
   }
}
```

Output :

12
