Packages in java

What is Package in Java?

Java package is a mechanism of grouping similar type of classes, interfaces, and sub-classes collectively based on functionality. When software is written in the Java programming language, it can be composed of hundreds or even thousands of individual classes. It makes sense to keep things organized by placing related classes and interfaces into packages.

Using packages while coding offers a lot of advantages like:

- Re-usability: The classes contained in the packages of another program can be easily reused
- Name Conflicts: Packages help us to uniquely identify a class, for example, we can have *company.sales.Employee* and *company.marketing.Employee* classes
- Controlled Access: Offers access protection such as protected classes, default classes and private class
- Data Encapsulation: They provide a way to hide classes, preventing other programs from accessing classes that are meant for internal use only
- Maintainance: With packages, you can organize your project better and easily locate related classes

Types of Packages in Java

Based on whether the package is defined by the user or not, packages are divided into two categories:

1. **Built-in Packages**
2. **User Defined Packages**

Built-in Packages

Built-in packages or predefined packages are those that come along as a part of JDK (Java Development Kit) to simplify the task of Java programmer. They consist of a huge number of predefined classes and interfaces that are a part of Java API's. Some of the commonly used built-in packages are java.lang, java.io, java.util, java.applet etc.

To use a class or a package from the library, you need to use the import keyword:

Syntax

import *package.name.Class*; // Import a single class

import *package.name*.*; // Import the whole package

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);

  }

}
```

User Defined Packages

User-defined packages are those which are developed by users in order to group related classes, interfaces and sub packages.

Note: The package name should be written in lower case to avoid conflict with class names.

Simple example of java package

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

//save as Simple.java

```
package mypack;
```

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

How to compile java package

If you are not using any IDE, you need to follow the syntax given below:

javac -d directory javafilename

For example

javac -d . Simple.java

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot) .

How to run java package program

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac -d . Simple.java

(Hint: d:\javaprogram> -d D:\javaprograms\ Simple.java)

To Run: java mypack.Simple

Output:Welcome to package

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 A.java
package pack;
public class A{
  public void msg()
{
System.out.println("Hello! I am from class A");
}
}
//save by B.java
package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello

---

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 A.java
```

```
package pack;
public class A{
  public void msg()
{
System.out.println("Hello");
}
}
//save by B.java
package mypack;
import pack.A;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello

---

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 A.java
package pack;
public class A{
public void msg()
{
System.out.println("Hello");
}
}
//save by B.java
package mypack;
class B{
```

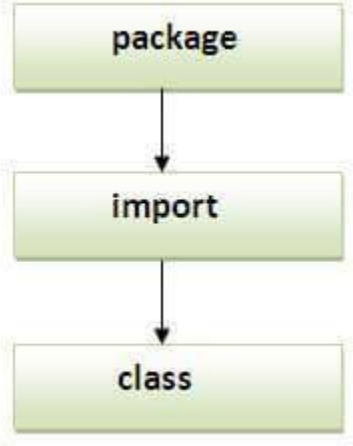
```
public static void main(String args[])
{
 pack.A obj = new pack.A(); //using fully qualified name
 obj.msg();
}  }
```

Output:Hello

Note: If you import a package, subpackages will not be imported.

If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.

Note: Sequence of the program must be package then import then class.



Subpackage in java

Package inside the package is called the **subpackage**. It should be created **to categorize the package further**.

Let's take an example, Sun Microsystem has definded a package named java that contains many classes like System, String, Reader, Writer, Socket etc. These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking etc and so on. So, Sun has subcategorized the java package into subpackages

such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.

Example of Subpackage

```
package com.notes.core;
class Simple
{
  public static void main(String args[])
  {
   System.out.println("Hello subpackage");
  }
}
```

To Compile: javac -d . Simple.java

To Run: java com.notes.core.Simple

Output:Hello subpackage

Note: There can be only one public class in a java source file and it must be saved by the public class name.

Access Protection in Java Packages

Packages in Java add another dimension to access control. Both classes and packages are a means of data encapsulation. While packages act as containers for classes and other subordinate packages, classes act as containers for data and code. Because of this interplay between packages and classes, Java packages addresses four categories of visibility for class members:

- Sub-classes in the same package
- Non-subclasses in the same package
- Sub-classes in different packages
- Classes that are neither in the same package nor sub-classes

The table below gives a picture of which type access is possible and which is not when using packages in Java:

	Private	Default	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package Subclasses	No	Yes	Yes	Yes
Same Package Non-Subclasses	No	Yes	Yes	Yes
Different Packages Subclasses	No	No	Yes	Yes
Different Packages Non-Subclasses	No	No	No	Yes

CLASSPATH in Java

Classpath in Java is the path to directory or list of the directory which is used by `ClassLoaders` to find and load class in Java program. Classpath can be specified using `CLASSPATH` environment variable which is case insensitive, `-cp` or `-classpath` command line option or `Class-Path` attribute in `manifest.mf` file inside JAR file in Java.

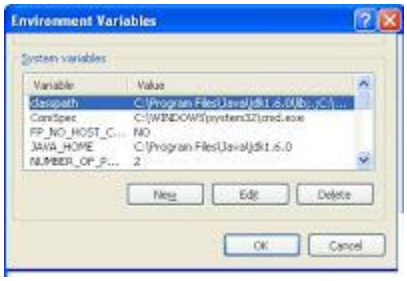
Note: The main difference between PATH and CLASSPATH is that former is used to locate Java commands while later is used to locate Java class files.

Setting Java Classpath in Windows

In order to set Classpath for Java in Windows (any version either Windows XP, Windows 2000 or Windows 7) you need to specify the value of environment variable ***CLASSPATH***, the name of this variable is not case sensitive and it doesn't matter if the name of your *environment variable is Classpath, CLASSPATH or classpath in Java*.

Here is *Step by Step guide for setting Java Classpath in Windows*:

1. Go to Environment variable window in Windows by right click on my computer than choosing properties and then Advanced and then Environment variable this will open Environment variable window in windows.

2. 

3. Now specify your environment variable `CLASSPATH` and put the value of your `JAVA_HOME\lib` and also include current directory by including (dot or period sign).
4. Now *to check the value of Java classpath in windows* type **"echo %CLASSPATH"** in your DOS command prompt and it will show you the value of directory which is included in CLASSPATH.

   You can also set classpath in windows by using DOS command like:

   ```
   set CLASSPATH=%CLASSPATH%;JAVA_HOME\lib;
   ```

   This way you can set the classpath in `Windows XP`, windows 2000 or Windows 7 and 8, as they all come with command prompt.