Raja N.L.Khan Women's College (Autonomous)
Department of BCA
2nd Semester
Paper: C3T
Prepared By: Moumita Mondal

JAVA - PACKAGES:

What is package:

Packages 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 or sub-package.

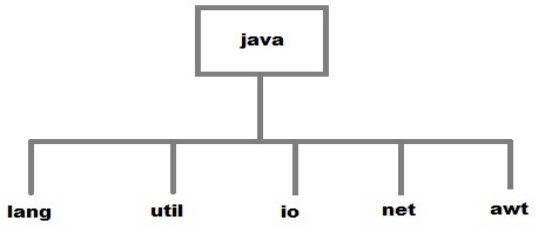
Using package it becomes easier to locate the related classes and it also provides a good structure for projects with hundreds of classes and other files.

Benefits of using package in java:

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.
4) This **packages** can be provide reusability of code.
5) We can create our own **package** or extend already available **package**.

Java Packages: Types:

- **Built-in Package:** Existing Java package for example `java.lang`, `java.util` ,java.io etc.
- **User-defined-package:** Java package created by user to categorize their project's classes and interface.



How to Create 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 mypackage;

public class student

{

Statement;

}
```

The above statement will create a package name **mypackage** in the project directory.

Java uses file system directories to store packages. For example the `.java` file for any class you define to be part of **mypackage** package must be stored in a **directory** called **mypackage**.

Additional points about package:

- A package is always defined as a separate folder having the same name as the package name.
- Store all the classes in that package folder.
- All classes of the package which we wish to access outside the package must be declared public.
- All classes within the package must have the package statement as its first line.
- All classes of the package must be compiled before use (So that they are error free)

Example of Java packages:

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

How to compile Java packages:

This is just like compiling a normal java program. If you are not using any IDE, you need to follow the steps given below to successfully compile your packages:

1. java -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

To Run: java mypack.Simple

Output: Welcome to package

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 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 java");}
}

//save by B.java
package mypack;
import pack.*;
  class B
{
  public static void main(String args[])
{
   A obj = new A();
   obj.msg();
  }
}
```

Output: Hello java

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.

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.

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

To Compile: javac -d . Simple.java

To Run: java com.javatpoint.core.Simple

Output: Hello subpackage

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

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

For example:

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

}

To Compile:

e:\sources> javac -d c:\classes Simple.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