Java Package

1. Java Package
2. Example of package
3. Accessing package
    1. By import packagename.*
    2. By import packagename.classname
    3. By fully qualified name
4. Subpackage
5. Sending class file to another directory
6. -classpath switch
7. 4 ways to load the class file or jar file
8. How to put two public class in a 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.

Built-in Packages in Java

Built-in is a part of Java API and it offers a variety of packages are –

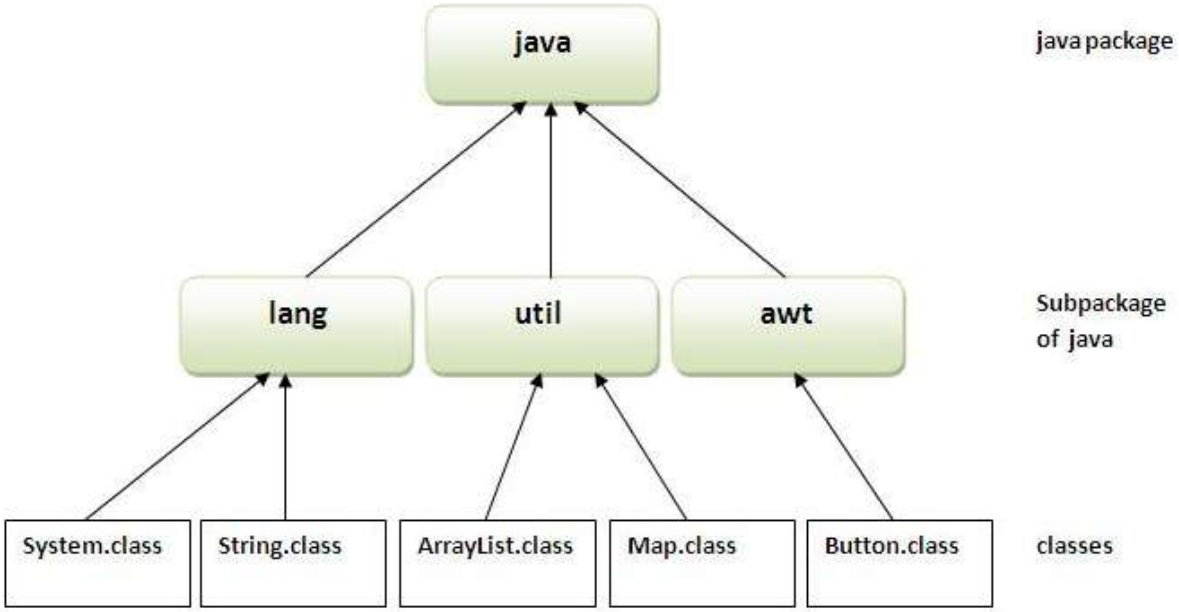
- **lang** – Automatically imported and it contains language support classes.
- **io** – Contains classes for input and output operations.
- **util** – Contains utility classes for implementing data structures.
- **applet** – This package contains classes that create applets.
- **awt** – Contain classes that implement compounds for GUI.
- **net** – This package contains classes that support networking operations.

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.



Simple example of java package

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

```
1. //save as Simple.java
2. package mypack;
3. public class Simple{
4.  public static void main(String args[]){
5.     System.out.println("Welcome to package");
6.    }
7. }
```

How to compile java package

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

```
1. javac -d directory javafilename
```

For **example**

```
1. 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.

Example 1: Java packages

I have created a class Calculator inside a package name letmecalculate. To create a class inside a package, declare the package name in the first statement in your program. A class can have only one package declaration.
Calculator.java file created inside a package letmecalculate

```
package letmecalculate;

public class Calculator {
   public int add(int a, int b){
            return a+b;
   }
   public static void main(String args[]){
            Calculator obj = new Calculator();
            System.out.println(obj.add(10, 20));
   }
}
```

Now let's see how to use this package in another program.

```
import letmecalculate.Calculator;
public class Demo{
   public static void main(String args[]){
            Calculator obj = new Calculator();
            System.out.println(obj.add(100, 200));
   }
}
```

To use the class Calculator, I have imported the package letmecalculate. In the above program I have imported the package as letmecalculate.Calculator, this only imports the Calculator class. However if you have several classes inside package letmecalculate then you can import the package like this, to use all the classes of this package.

```
import letmecalculate.*;
```

Example 2: Creating a class inside package while importing another package

As we have seen that both package declaration and package import should be the first statement in your java program. Let's see what should be the order when we are creating a class inside a package while importing another package.

```
//Declaring a package
package anotherpackage;
//importing a package
import letmecalculate.Calculator;
public class Example{
   public static void main(String args[]){
            Calculator obj = new Calculator();
            System.out.println(obj.add(100, 200));
   }
}
```

So the order in this case should be:
→ package declaration
→ package import

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.*

```
1. //save by A.java
2. package pack;
3. public class A{
4.   public void msg(){System.out.println("Hello");}
5. }
1. //save by B.java
2. package mypack;
3. import pack.*;
4.
5. class B{
6.   public static void main(String args[]){
7.    A obj = new A();
8.    obj.msg();
9.   }
10. }
```

```
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

```
1. //save by A.java
2.
```

```
3. package pack;
4. public class A{
5.   public void msg(){System.out.println("Hello");}
6. }
```

```
1. //save by B.java
2. package mypack;
3. import pack.A;
4.
5. class B{
6.   public static void main(String args[]){
7.    A obj = new A();
8.    obj.msg();
9.   }
10. }
```

```
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

Handling name conflicts

The only time we need to pay attention to packages is when we have a name conflict . For example both, java.util and java.sql packages have a class named Date. So if we import both packages in program as follows:

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


//And then use Date class, then we will get a compile-time error :


Date today ; //ERROR-- java.util.Date or java.sql.Date?
```

The compiler will not be able to figure out which Date class do we want. This problem can be solved by using a specific import statement:

```
import java.util.Date;
import java.sql.*;
```

If we need both Date classes then, we need to use a full package name every time we declare a new object of that class. For Example:

```
java.util.Date deadLine = new java.util.Date();
java.sql.Date today = new java.sql.Date();
```

Example of package by import fully qualified name

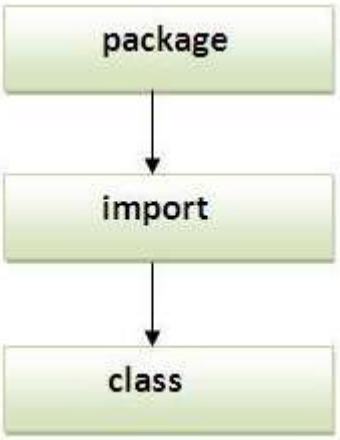
```
1. //save by A.java
```

```
2. package pack;
3. public class A{
4.   public void msg(){System.out.println("Hello");}
5. }
1. //save by B.java
2. package mypack;
3. class B{
4.   public static void main(String args[]){
5.    pack.A obj = new pack.A();//using fully qualified name
6.    obj.msg();
7.   }
8. }
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.

> ***The standard of defining package is domain.company.package e.g. com.javatpoint.bean or org.sssit.dao.***

Example of Subpackage

```
1. package com.javatpoint.core;
2. class Simple{
3.   public static void main(String args[]){
4.    System.out.println("Hello subpackage");
5.   }
6. }
```

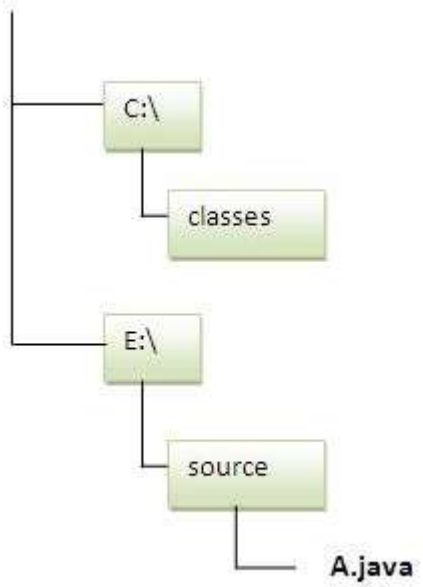
To Compile: javac -d . Simple.java

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:




```
1. //save as Simple.java
2. package mypack;
3. public class Simple{
```

```
4.  public static void main(String args[]){
5.    System.out.println("Welcome to package");
6.   }
7. }
```

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
```

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.

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

```
1. //save as C.java otherwise Compile Time Error
```

```
2.
3. class A{}
4. class B{}
5. public class C{}
```

How to put two public classes in a package?

If you want to put two public classes in a package, have two java source files containing one public class, but keep the package name same. For example:

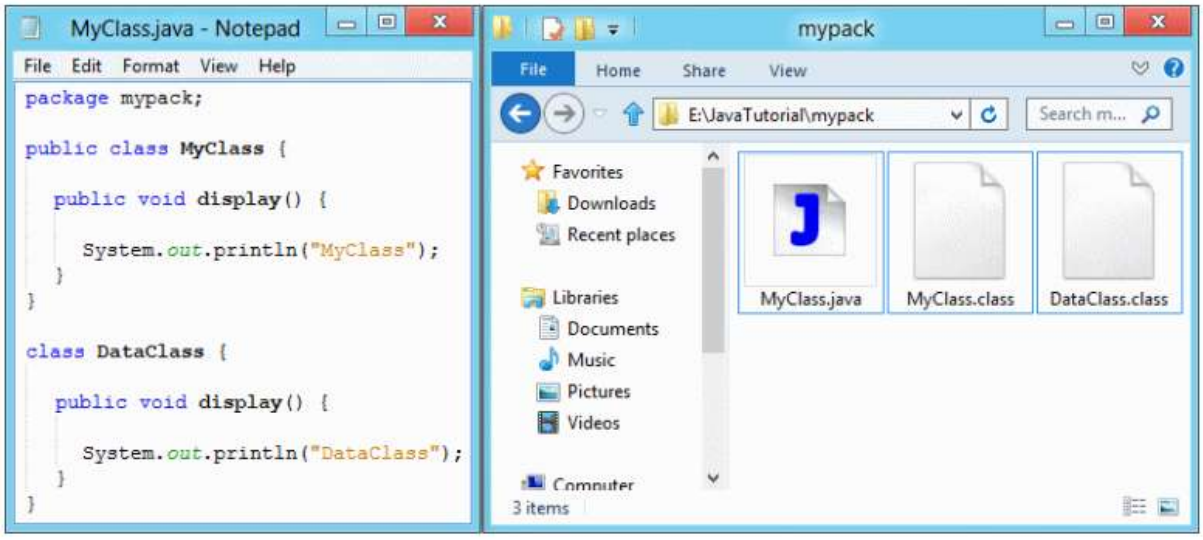
```
1. //save as A.java
2.
3. package javatpoint;
4. public class A{}
1. //save as B.java
2.
3. package javatpoint;
4. public class B{}
```

Java-Hiding Classes

Hiding Classes

When we import a package within a program, only the classes declared as **public** in that package will be made accessible within this program. In other words, the classes not declared as public in that package will not be accessible within this program.

We shall profitably make use of the above fact. Sometimes, we may wish that certain classes in a package should not be made accessible to the importing program. In such cases, we need not declare those classes as public. When we do so, those classes will be hidden from being accessed by the importing class. Here is an example :



Here, the class **DataClass** which is not declared public is hidden from outside of the package **mypack.** This class can be seen and used only by other classes in the same package.

Note that a Java source file should contain **only one public class** and may include any **number of non-public classes**.

Program

```
import mypack.*;

public class Javaapp {

    public static void main(String[] args) {

        DataClass da = new DataClass();
        da.display();
    }
}
```

Program Output

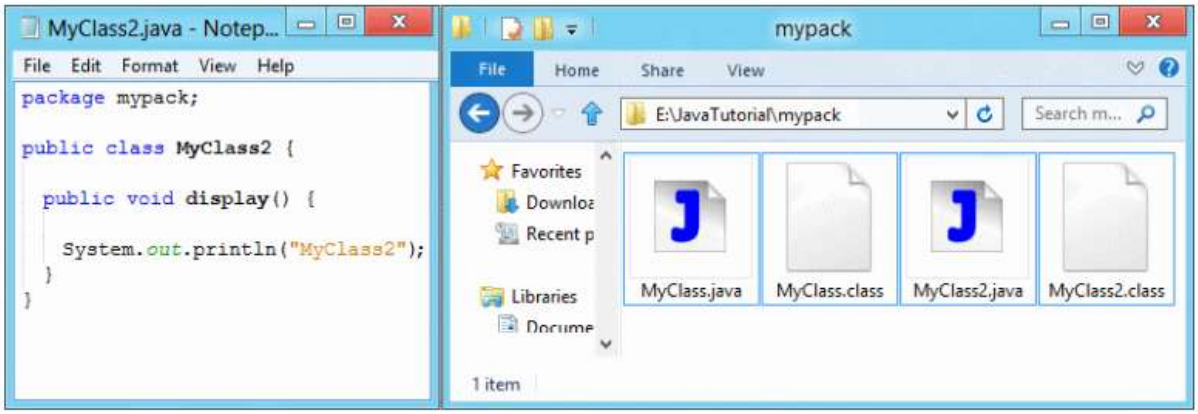
```
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code
mypack.DataClass is not public in mypack; cannot be accessed from outside package
```

Java-Package with Multiple Public Classes

Package with Multiple Public Classes

A Java source file can have only one class declared as **public**, we cannot put two or more public classes together in a **.java** file. This is because of the restriction that the file name should be same as the name of the public class with **.java** extension. If we want multiple classes under consideration are to be declared as **public**, we have to store them in **separate source files** and attach the **package** statement as the first statement in those source files.
If we want to add class to an existing package. For example the package **mypack** contains one public class by name **MyClass**. Suppose we want to add another public class **MyClass2** to this package. This can be as follows :

1)	Define the class and make it **public**.
2)	Place the **package statement** before the class definition.
3)	Store this as **MyClass2.java** file under directory **mypack**.
4)	Compile **MyClass2.java** file. This will create a **MyClass2.class**.



Program

```
import mypack.MyClass2;

public class Javaapp {

    public static void main(String[] args) {

        MyClass2  my2 = new MyClass2();
        my2.display();
    }
}
```

Program Output

```
MyClass2
```