Java Package

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.

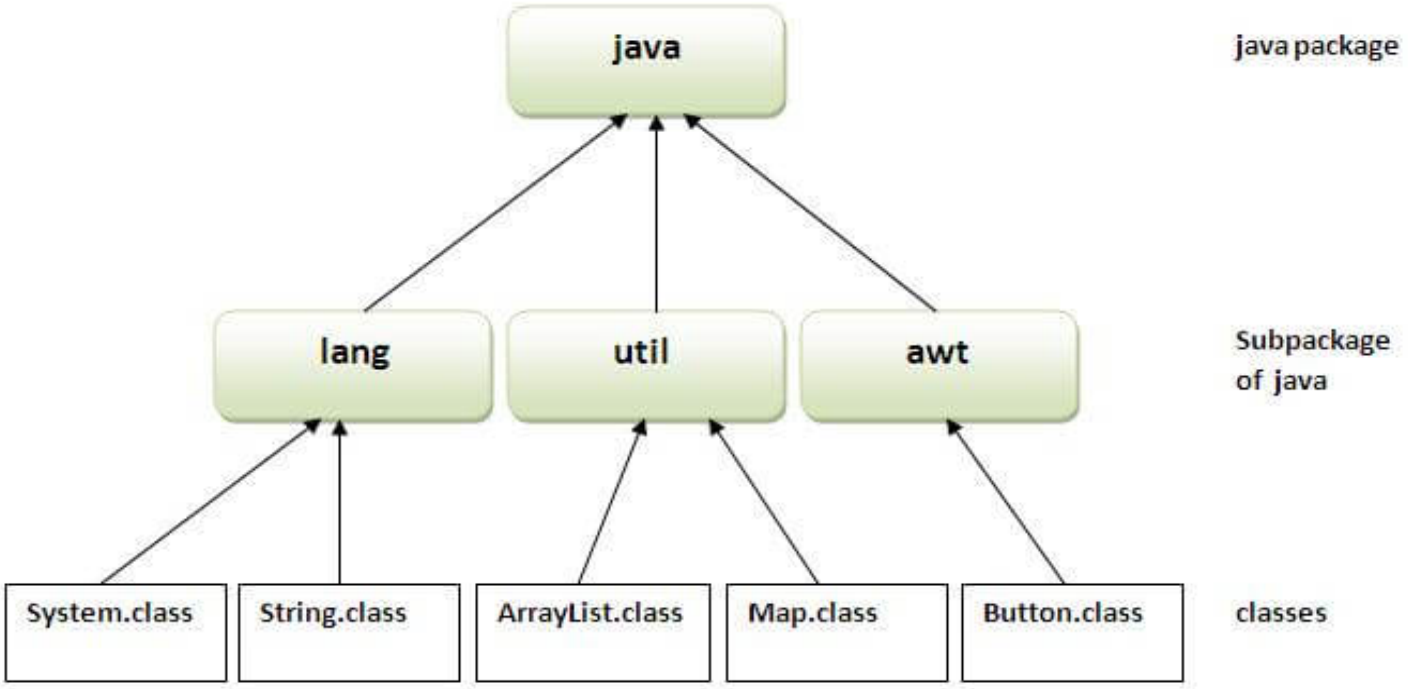
Java Package

Advantage of Java Package :

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

Java Package

Structure of Java Package :



Java Package

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");
   }
}
```

Java 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).

Java Package

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

Java Package

How to access package from another package ?

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

- **import package.*;**
- **import package.classname;**
- **fully qualified name.**

Java Package

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.

Java 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 Package

Example of package that import the packagename.*

```
//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 Package

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");}
}
```

Java Package

Example of package by import package.classname :

```
//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

Java Package

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.

Java Package

Example of package by import fully qualified name

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

Java Package

Example of package by import fully qualified name

```
//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

Java Package

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.

Java Package

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.

Java Package

Example of Subpackage :

```
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

Java Package

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

```
//save as C.java otherwise Compilte Time Error

class A{}
class B{}
public class C{}
```

Java Package

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:

```
//save as A.java
package javatpoint;
public class A{}

//save as B.java
package javatpoint;
public class B{}
```

Access Modifiers in java

Access Modifiers in java :

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

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

- **private**
- **default**
- **protected**
- **public**

Access Modifiers in java

Access Modifiers in java :

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

Access Modifiers in java

private access modifier :

The private access modifier is accessible only within class.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

Access Modifiers in java

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 A{
   private A(){}//private constructor
   void msg(){System.out.println("Hello java");}
   }
   public class Simple{
    public static void main(String args[]){
      A obj=new A();//Compile Time Error
    }
   }
```

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

Access Modifiers in java

Default access modifier :

If you don't use any modifier, it is treated as default bydefault. The default modifier is accessible only within package.

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

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

Access Modifiers in java

Default access modifier :

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

Access Modifiers in java

Protected access modifier

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.

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

Access Modifiers in java

Protected access modifier

```
//save by B.java
package mypack;
import pack.*;

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

Output:Hello

Access Modifiers in java

Public access modifier

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

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

Access Modifiers in java

Public access modifier

```
//save by B.java
package mypack;
import pack.*;

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

Output:Hello

Access Modifiers in java

Understanding all java access modifiers

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

Access Modifiers in java

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 A{
protected void msg(){System.out.println("Hello java");}
}
public class Simple extends A{
void msg(){System.out.println("Hello java");}//C.T.Error
 public static void main(String args[]){
   Simple obj=new Simple();
   obj.msg();  }  }
```

The default modifier is more restrictive than protected. That is why there is compile time error.