

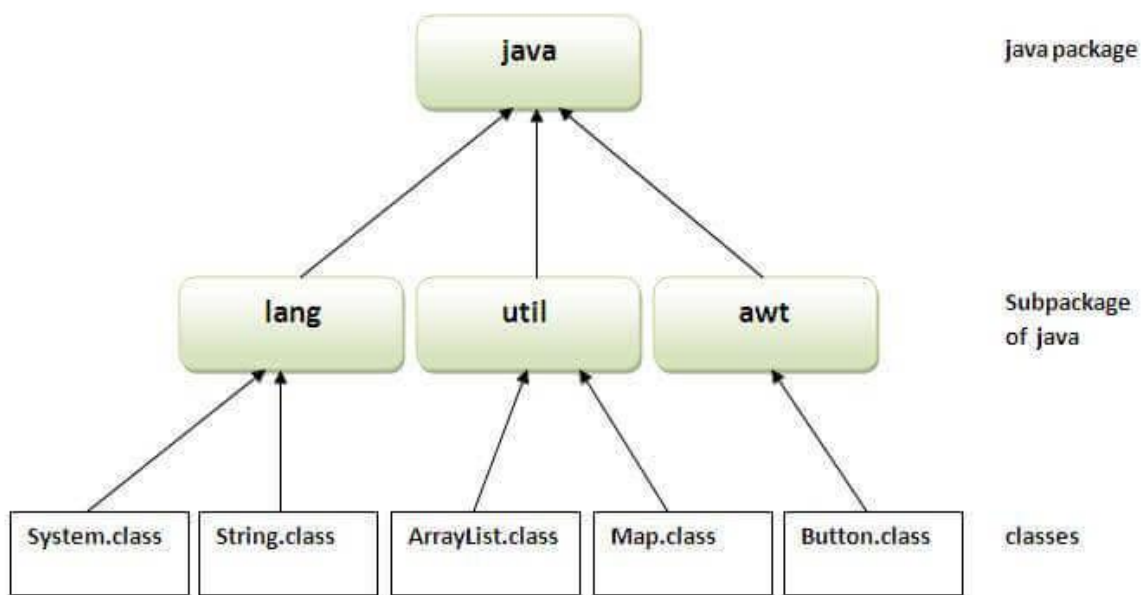
Use of Package, Import statement, Access Modifiers

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.

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.

```
//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:

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

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");}
}

//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.

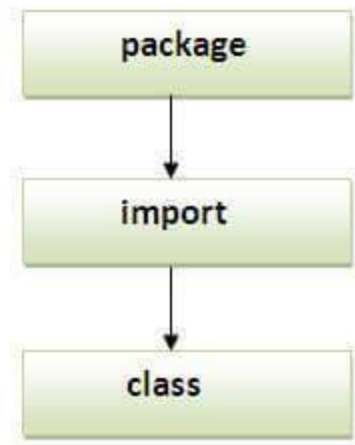
It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

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



Access Modifiers

The access modifier in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.
5. Understanding Java Access Modifiers
6. Let's understand the access modifiers in Java by a simple table.

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

1) Private

The private access modifier is accessible only within the class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

```

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
    }
}

```

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
}
}
```

2) Default

1. If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.
2. 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");}
}

//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
}
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

3) Protected

- 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.
- It provides more accessibility than the default modifier.

Example of protected access modifier

- In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

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

//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

4) Public

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

Example of public access modifier

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



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

```
package mypack;
import pack.*;

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

Output:Hello