Packages in Java

- Package is a collection of related classes.
- Packages provide naming and visibility
- Java uses package to group related classes, interfaces and sub-packages .
- Package can be assumed as a folder or a directory that is used to store similar files.
- In Java, packages are used to avoid name conflicts and to control access of class, interface and enumeration etc.
- 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.

Types Of Java Packages

- Package can be built-in and user-defined.
- Java provides rich set of built-in packages that stores related classes and sub-packages.
    - **Built-in Package:** util, lang, io etc are the example of built-in packages.
    - **User-defined-package:** Java packages created by user to categorize their project's classes and interfaces are known as user-defined packages.

How to Create a Package

To create a package in java , include a package command followed by name of the package as the first statement in java source file.

Syntax: package packagename;

Ex:

```
package mypack;

public class employee

{

   String empId;

   String name;

}
```

- The above statement will create a package with name **mypack**
- Java uses file system directories to store packages.
- For example the .java file for any class you define to be part of **mypack** package must be stored in a **directory** called **mypack**.

NOTE:

- Package statement must be first statement in the program even before the import statement.
- 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.

Example:

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

- to compile Java programs inside packages

Method-1:

1. Create a folder with the same name as the package
2. Save the file with the package statement in that folder
3. While compiling first goto that folder using cd command
4. Then compile as javac filename
5. Now come back to the parent directory using cd .. command
6. Now run the program as packagename.classname

Method-2:

1. Save the java file in cwd i.e current working directory
2. Now compile it as javac –d . filename.java
3. This will create a directory with the same name as the package in the current working directory
4. Now simply run as java packagename.filename

javac -d . FirstProgram.java

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

Method-3:

Changing the environmental variable CLASSPATH makes the changes permanent.

Importing a package

- To import java package into a class, we need to use java **import** keyword which is used to access package and its classes into the java program.
- We can use import to access built-in and user-defined packages into our java source file so that our class can refer to a class that is in another package by directly using its name.

There are 3 different ways to refer to any class that is present in a different package:

1. without import the package
2. import package with specified class
3. import package with all classes

Accessing package without import keyword

- use fully qualified name to import any class into your program, then that particular class of the package will be accessible in the program
- other classes in the same package will not be accessible.
- For this approach, there is no need to use the import statement.
- But you will have to use the fully qualified name every time you are accessing the class or the interface.
- This is generally used when two packages have classes with same names.

Example

```
package pack;
public class A {  // should be compulsorily saved as A.java
   public void msg() {
      System.out.println("Hello");
   } }
package mypack;
class B {
   public static void main(String args[]) {
      pack.A obj = new pack.A();  //using fully qualified name
      obj.msg();    } }
```

Import the Specific Class

- Package can have many classes but sometimes we want to access only specific class in our program .
- Then specify class name along with package name.
- If we use import packagename.classname statement then only the class with name *classname* in the package will be available for use.

Example:

```
package pack;
public class Demo {
   public void msg() {
      System.out.println("Hello");    } }
```

```
package mypack;

import pack.Demo;

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

Import all classes of the package

- If we use **packagename.* statement**, then all the classes and interfaces of this package will be accessible but the classes and interface inside the sub-packages will not be available for use.
- The import keyword is used to make the classes of another package accessible to the current package.

Example :

```
package learnjava;

public class First{
    public void msg() {
        System.out.println("Hello");
    }
```

```
}
```

```
package Java;

import learnjava.*;

class Second {

    public static void main(String args[]) {

        First obj = new First();

        obj.msg();

    }

}
```

Access Modifiers in Java

- Access modifiers are keywords in Java that are used to set accessibility.
- An access modifier restricts the access of a class, constructor, data member and method in another class.

Java language has four access modifiers to control access level for classes and its members.

- **Default:** Default has scope only inside the same package
- **Public:** Public has scope that is visible everywhere
- **Protected:** Protected has scope within the package and all sub classes
- **Private:** Private has scope only within the classes

Default Access Modifier

- If we don't specify any access modifier then it is treated as default modifier.

- It is used to set accessibility within the package.
- It means we can not access its method or class from outside the package.
- It is also known as package accessibility modifier.

Public Access Modifier

- public access modifier is used to set public accessibility to a variable, method or a class.
- Any variable or method which is declared as public can be accessible from anywhere in the application.

Protected Access Modifier

- Protected modifier protects the variable, method from accessible from outside the class.
- It is accessible within class, and in the child class (inheritance) whether child is located in the same package or some other package.

Private Access Modifier

- Private modifier is most restricted modifier which allows accessibility within same class only.
- We can set this modifier to any variable, method or even constructor as well.

Accessibility / Access Specifier	Same Class	Same Package		Other Package	
		Child class	Non-child class	Child class	Non-child class
Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	Yes	No
Default	Yes	Yes	Yes	No	No
Private	Yes	No	No	No	No

```
// ACCESS PROTECTION IN PACKAGES

//PROTECTION.JAVA

package p11;

public class Protection

{

int a=1;

private int b=2;

protected int c=3;

public int d=4;

 public Protection()

{

System.out.println(a);

System.out.println(b);
```

```
System.out.println(c);
System.out.println(d);
}}
//DERIVED.JAVA
package p11;
 class Derived extends Protection
{

Derived()
{
System.out.println(a);
//System.out.println(b);
System.out.println(c);
System.out.println(d);
}}
//SAMEPACKAGE.JAVA
package p11;

 class SamePackage
{
SamePackage()
{
Protection ob=new Protection();
System.out.println(ob.a);
//System.out.println(ob.b);
System.out.println(ob.c);
System.out.println(ob.d);
```

```
}}
//DEMO1.JAVA
package p11;
 class Demo1
{
public static void main(String args[])
{
 Protection ob1=new Protection();
 Derived ob2=new Derived();
SamePackage ob3=new SamePackage();
test ob4=new test();
}}
//DERIVED2.JAVA
package p12;

 class Derived2 extends p11.Protection
{

Derived2()
{
//System.out.println(a);
//System.out.println(b);
System.out.println(c);
System.out.println(d);
}}
//OTHERPACKAGE.JAVA
package p12;
```

```
import p11.Protection;


 class OtherPackage
{
OtherPackage()
{
Protection ob=new Protection();
//System.out.println(ob.a);
//System.out.println(ob.b);
//System.out.println(ob.c);
System.out.println(ob.d);
}}
 //DEMO2.JAVA
package p12;
 class Demo2
{
public static void main(String args[])
{
 Derived2 ob2=new Derived2();
OtherPackage ob3=new OtherPackage();
}}
```