**Assignment 12**

**Title of Assignment :Write a Java program which will demonstrate a concept of Interfaces and packages: In this assignment design and use of customized interfaces and packages for a specific application are expected.**

**Relevant Theory:**

**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.

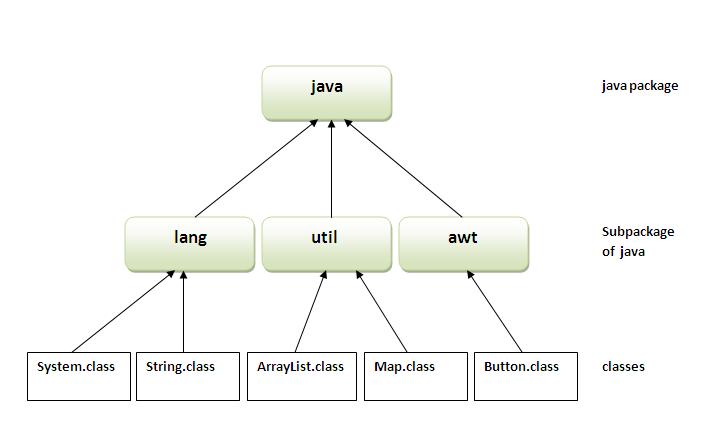
Here, we will have the detailed learning of creating and using user-defined packages.

**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.

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. |

## 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. }
6. //save by B.java
7. package mypack;
8. import pack.\*;
10. class B{
11. public static void main(String args[]){
12. A obj = new A();
13. obj.msg();
14. }
15. }

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
3. package pack;
4. public class A{
5. public void msg(){System.out.println("Hello");}
6. }
7. //save by B.java
8. package mypack;
9. import pack.A;
11. class B{
12. public static void main(String args[]){
13. A obj = new A();
14. obj.msg();
15. }
16. }

Output:Hello

**Interface in Java**

An **interface in java** is a blueprint of a class. It has static constants and abstract methods.

The interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.

Java Interface also **represents IS-A relationship**.

It cannot be instantiated just like abstract class.

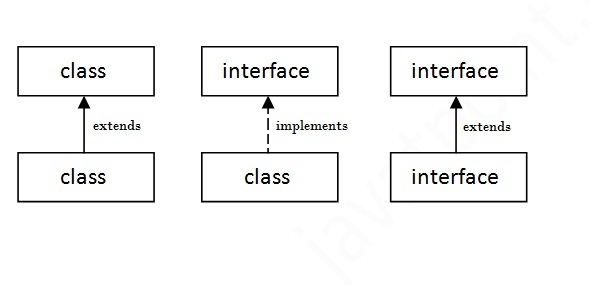
**Why use Java interface?**

There are mainly three reasons to use interface. They are given below.

* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.

#### Understanding relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface but a **class implements an interface**.



## Java Interface Example

In this example, Printable interface has only one method, its implementation is provided in the A class.

1. interface printable{
2. void print();
3. }
4. class A6 implements printable{
5. public void print(){System.out.println("Hello");}
7. public static void main(String args[]){
8. A6 obj = new A6();
9. obj.print();
10. }
11. }

Output:

Hello

**Conclusion: Understood concepts of package and interface and performed application for the same.**