PACKAGE: by

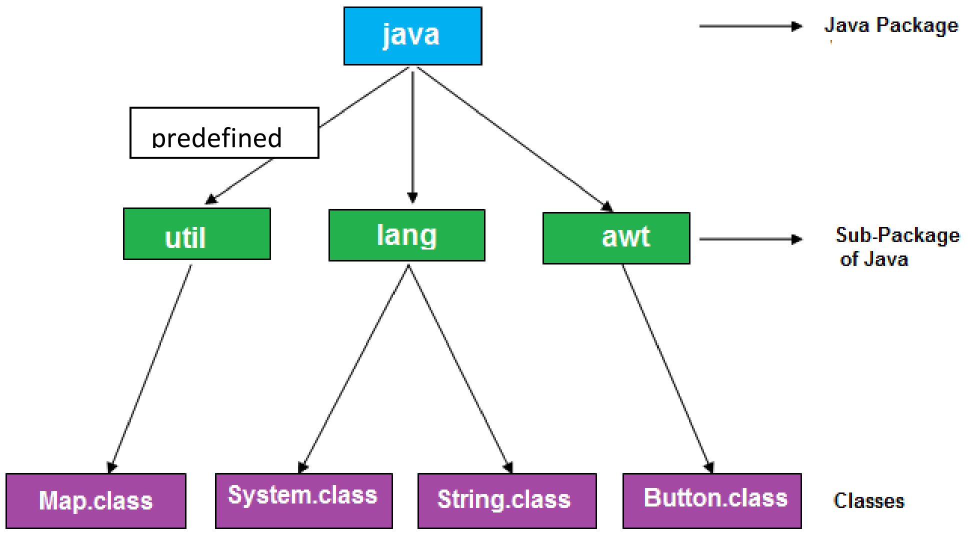
Package is a collection of related classes and interfaces that provides access protection and namespace management.

Generally, we are creating the package to get the following functionality—

1. It provides an inheritance hierarchy, so that we and other programmer are able to find the related classes and interfaces.
2. It provides unlimited access to each other within the package, still it provides restricted access to the outsiders of the package.
3. The name of a class or interface within a package will not conflict with the name of class or interface available in other package, because package namespace collision.

Package in java can be categorized in two form,

(a)built-in package



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

HOW TO CREATE A PACKAGE?

A package is nothing but a folder in java that contains some classes and interfaces, so that it can be used by other classes like a header file.

Let us create a folder called geometry under D:\demo folder which will contain a folder called geometry that will contain Shape interface, Triangle and Rectangle class.

A source file which will remain in a package must provide the package statement as the first executable statement within itself as given below—

```
package packagename;
```

where, package is the keyword to create a package, and packagename is the user defined folder name.

ex- package geometry; → this statement must be the first executable statement of rectangle, triangle and shape.

Creating a package called geometry :

```
/* Store this file as Shape.java under geometry folder */
package  geometry;
public  interface  Shape {
   void calArea();
}
/* Store this file as Rectangle.java under geometry folder */
package  geometry;
public  class  Rectangle  implements  Shape {
int  len , brdth;
Rectangle() {  len = 6;  brdth = 9;  }
public Rectangle ( int  p,  int  q )  {   len = p;   brdth = q;  }
public  void  calArea()  {
                System.out.println("area of rec:"+len*brdth);
             }
}
 int   recArea(){
      return  len*brdth;
}};

/*  Store this file as Tringle.java under geometry folder */
package  geometry;
public class Triangle implements Shape {
int  ht, base;
Triangle () {  ht=5;  base=12;  }
public  Triangle(int p,  int  q) {  ht = p;  base = q;  }
public void calArea() {
          System.out.println("area of trg : " + ( ht*base ) / 2 );
}};
```

The above source files should be compiled in order to create/use the package. If we don't compile the source file then we cannot use the package in other file.

Compiling the Source files of the Package :- If there is any inheritance among the classes & interfaces of the package then we cannot compile the source file individually from the package itself. Hence we have to compile the source file at a time, as given below ..

D:\demo\geometry> **javac *java**

Irrespective of the inheritance relationship among the classes and interface of the package we can always compile the source file of the package individually or at once from the parent of the package.

D:\demo> javac d:\demo\geometry\Rectangle.java
(absolute path)
D:\demo> javac .\geometry\Triangle.java
(relative path)

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.
Using the geometry package :-- A source file which will use a package must provide the following import statement as the first statement before the class declaration.

```
     import  packagename.*;                    -> it will include all dotclass files

Ex..  import  geometry.*;
       import java.util.*;
Or      import packagename.dotclassfilename;      -> It will include one dotclass file of the
package
Ex     import   geometry.Rectangle;
         import   java.util.Scanner;
```

Using the geometry package :
Lets create a class test.java which will use the package

```
import geometry.*;
class  Test {
public static void main(String  args[]){
          Rectangle rec = new   Rectangle( 10, 12);
          Triangle trg = new  Triangle( 8, 10);
          rec.calArea();
          trg.calArea();
}};
```

The above source file should not be stored on the geometry folder otherwise compilation error will occur. Lets store Test.java under different folder as given bellow.
Case->1 : When implementation file (Test.java) is stored under the parent of the package(D:\demo), then it should be compiled and execute like a simple java program.
Case->2 : If the implementation file (Test.java) is stored under any folder except the parent of the package, then we have to compile and execute the implementation of file as given bellow (lets stored Test.java under e:\help)

(1)Compile as:- E:\help> **javac –classpath D:\demo;. Test.java**
Execute as:- E:\help> **java –classpath D:\demo;. Test**
or
(2) Compile as:- E:\help> **javac –cp D:\demo;. Test.java**
Execute as:- E:\help> **java –cp D:\demo;. Test**
or
(3) E:\help>set classpath=D:\demo;.
Compile as:- E:\help>javac Test.java
Execute as:- E:\help>java Test

<u>classpath :</u> It is an enviornment variable because application program uses this variable. It is used to hold the path of a package ,zip file, jar file, war file etc. so that the application program will use this variable to find the path of the desire file

<u>-classpath :</u> It is an option for the java compiler and interpreter, when they are unable to find the path of a package then this temporary path is used.

D:\demo : - It is the parent of the package

; (semicolon) :- It is the path separator between different path.

. (dot) :- it represent the current folder value so that the compiler and interpreter will remember from where they have started compilation and execution?

Case->3 : If we set the permanent classpath of package, then we can compile & execute the implementation file like a simple java program. (Under the user variable list an environment variable dialog box look for classpath, if it is not available then click the new button, a dialog box will appear known as new user variable. Write **classpath** in the variable name and **.;d:\demo;** in the variable value click all ok or apply button after setting the permanent will can compile and execute Test.java like a simple java program .even if it is stored under any folder or drive.

<u>How To Create a Sub-package :</u> The stpes describes how to create the subpackages,

Stpe 1 : Create a folder called drawing under geometry foldet, so that the drawing folder will behave like a sub package.

Step 2 : The drawing package should contain a class Square which will inherit from Rectangle class. Hence the default constructor and the Rectangle must be public or protected, because when the object of Square class will be created, then the default constructor of Rectangle class will be invoked implicitly.

Step 3 : Set the permanent classpath of main(geometry) package in environmental variable.

Step 4 : Since the source file that stored in a package contains the package statement as the 1st statement, hence the source file of drawing package i.e. Square class must provide package & sub package name as the package statement within it.

```
 package  geometry.drawing;
 public class Square  extends  geometry.Rectangle {
   int side;
   public Square(){ side=100;  }
   public Square(int p){ side=p;  }
   public void calArea(){
        System.out.println("area of Square "+ side* side);
}};
```

Store the above file as Square.java in drawing package,

Step 5: Compile the Square.java from the parent of main(geometry) package as given below.

```
D:\demo> javac    .\geometry\drawing\Square.java
```

Step 6 : The implementation file(Test.java) should be written as given below :

```
import  geometry.*;
import  geometry.drawing.*;
class  Test    {
  public static void main(String args[]){
    Rectangle rec = new Rectangle(10,12);
    Triangle trg = new Triangle(8,10);
    rec.calArea();
    trg.calArea();
    Square sq = new  Square();
    sq.calArea();
}};
```

Step 7 : Since we have set the permanent classpath of the main(geometry) package, therefore , the Test.java can be stored any location and it can be compiled and executed like a simple java program.

Note : The following points should be remembered about a package—

1. The package name in package statement and in import statement must remain in similar case, otherwise compilation error will occur.
2. When we import a package, then we can use the classes and interfaces of the package , but we cannot use the classes and interfaces of the sub-package. Hence to use the classes and interfaces of the sub-package we have to import the sub-package.
3. We can use a class or interface of a package without import statement , but we have to use long qualified name of the package.

<u>Namespace Management :</u>

The name of a class/interface along with the memory space/folder/package where it is stored is know as namespace. Java allows us to provide same name to different source files by storing them in different forder/package.

```
import   java.util.*;
import   java.sql.*;
class Test{
static public void main(String args[]){
    Date d = new Date();
}}
```

The Date class in above program is available in util & in sql package. Since we are using both package, then compiler will face an ambiguity to create the Date object. It is known as namespace collision/conflict. Java avoids the namespace collision/conflict by using the long qualified name of the package as given bellow

java.util.Date d = new java.util.Date();   // java.util.Date   is called long qualified name.

Static import :

1. If we write the static keyword in between the import keyword and the package name, then it is called static import. It is used to import the static members of a class, so that we can use the static members in our class as if we have defined it.
2. When we import the static members of a class by static import, then we should not define any member in our class with the same name as that of the static import.

Simple Example of static import

```
import static java.lang.System.*;
class StaticImportExample{
  public static void main(String args[]){

   out.println("Hello");//Now no need of System.out
   out.println("Java");

 }
}
```

Q->What is the difference between import and static import?

The import allows the java programmer to access classes of a package without package qualification whereas the static import feature allows to access the static members of a class without the class qualification. The import provides accessibility to classes and interface whereas static import provides accessibility to static members of the class.

ACCESS SPECIFIER :-A class may be preceded by public or default access specifier, when it default then it can not be accessed by any body outside of the package. When it is public then it can be accessed by anybody outside of the package.

Accessible of the members of class	**Private**	**Default**	**protected**	**Public**
Within the same class	Yes	Yes	Yes	Yes
Within the same package & by subclass	No	Yes	Yes	Yes
Within the same package but by non subclass	No	Yes	Yes	Yes
Outside the package by subclass	No	No	Yes	Yes
Outside the package by non subclass	No	No	No	Yes

PRIVATE :- private members of the class are available with in the class only. The scope of private members of the class is “CLASS SCOPE”.

DEFAULT:- A member of a class preceded by default access specifier can be access by any other class of the same package .Hence the default access specifier behaves like public for the other classes form the same package. The member of a class presided by default access specifier can't be accessed by any other class that remains outside of the package hence default behaves like private for the classes outside of the package.

PROTECTED :- The member of the class preceded by the protected access specifier can be access by any other class within the same package .Hence the protected access specifier can behaves like public. The member of a class preceded by protected access specifier can't be by any non sub class that remain outside of the of a package, hence protected behaves like private for the non subclasses that remains outside of the package. The member of a class preceded protected access specifier can be accessed by the subclass that remain out side of the package , even if the class is not allowed to access its own protected member outside of the package

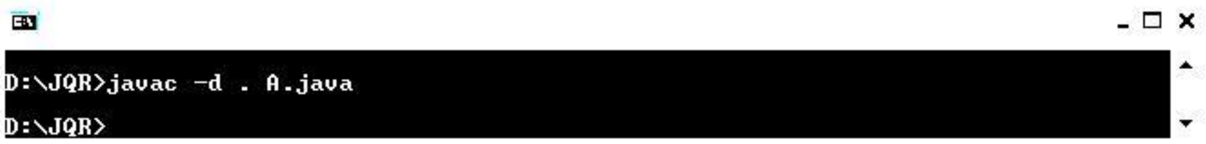
The protected and default access specifier allows everybody within the package but they does not allows anybody to access outside of package where as the protected will allows the child class to access even if the child class is outside of the package hence package provides access protection.

PUBLIC:- public members of the class are available anywhere . The scope of public members of the class is "GLOBAL SCOPE".

Program 1: Write a program to create class A with different access specifiers.

```
//create a package same
package same;
public class A
{ private int a=1;
public int b = 2;
protected int c =3;
int d = 4;
}
```

Compiling the above program:



```
D:\JQR>javac -d . A.java
D:\JQR>
```

D:\JQR>javac -d . *.java

The –d option tells the Java compiler to create a separate directory and place the .class file in that directory (package).

The (.) dot after –d indicates that the package should be created in the current directory.

program 2: Write a program for creating class SC in the same

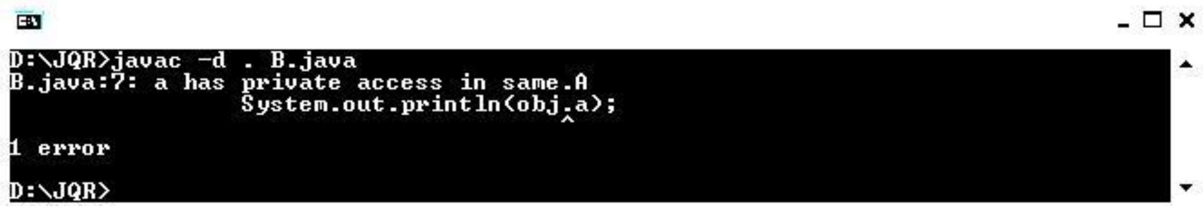
```
package same;
import same.A;
public class SC extends A
{
```

```
public static void main(String args[])
 {
SC obj = new SC();
System.out.println(obj.a);
System.out.println(obj.b);
System.out.println(obj.c);
System.out.println(obj.d);
}
}
D:\JQR>javac -d . SC.java
SC.java:9: error: a has private access in A
System.out.println(obj.a);
                  ^
1 error
```

Program 3: Write a program for creating class B in the same package. //class B of same package

```
package   same;
import  same.A;
public class B
{
          public static void main(String args[])
          {
                    A obj = new A();
                    System.out.println(obj.a)
                    System.out.println(obj.b);
                    System.out.println(obj.c);
                    System.out.println(obj.d);
          }
}
```

Compiling the above program:



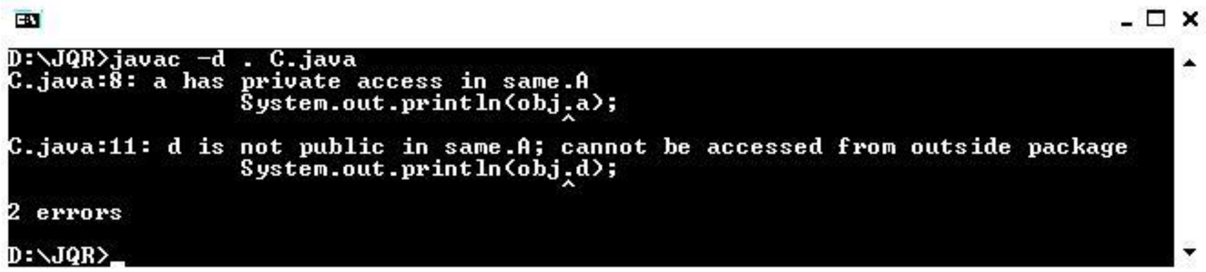
```
D:\JQR>javac -d . B.java
B.java:7: a has private access in same.A
                System.out.println(obj.a);
                                      ^
1 error

D:\JQR>
```

Program 4: Write a program for creating class C of another package.

```
package another;
import same.A;
public class C extends A
{ public static void main(String args[]) { C obj = new C();
                    System.out.println(obj.a);
                    System.out.println(obj.b);
                    System.out.println(obj.c);
                    System.out.println(obj.d);
          }
}
```

Compiling the above program:



```
D:\JQR>javac -d . C.java
C.java:8: a has private access in same.A
                System.out.println(obj.a);
                                      ^
C.java:11: d is not public in same.A; cannot be accessed from outside package
                System.out.println(obj.d);
                                      ^
2 errors

D:\JQR>_
```

Program 5: Write a program for creating class C of another package.

```
package another;
import same.A;
public class E
{
public static void main(String args[])
 {
A obj = new A();
System.out.println(obj.a);

System.out.println(obj.b);
System.out.println(obj.c);
System.out.println(obj.d);
}
}
```

```
D:\JQR>javac -d . E.java
E.java:11: error: a has private access in A
System.out.println(obj.a);
                      ^
E.java:15: error: c has protected access in A
System.out.println(obj.c);
                      ^
E.java:17: error: d is not public in A; cannot be accessed from outside package
System.out.println(obj.d);
                      ^
3 errors
```

Java Naming conventions

Java **naming convention** is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

But, it is not forced to follow. So, it is known as convention not rule.

All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

Advantage of naming conventions in java

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that **less time** is spent to figure out what the code does.

Name	**Convention**
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

If name is combined with two words, second word will start with uppercase letter always e.g. actionPerformed(), firstName, ActionEvent, ActionListener etc.