JAVA PROGRAMMING

COURSE NAME: MCA – 5TH SEMESTER

COURSE CODE: MCA18501CR

Teacher Incharge: *Dr. Shifaa Basharat*
Contact: *fazilishifaa@gmail.com*

PACKAGES:

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

- **Re-usability:** The classes contained in the packages of another program can be easily reused
- **Name Conflicts:** Packages help us to uniquely identify a class, for example, we can have two classes with the name Employee in two different packages, *company.sales.Employee* and *company.marketing.Employee.*
- **Controlled Access:** Offers access protection such as protected classes, default classes and private class. Protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.
- **Data Encapsulation:** They provide a way to hide classes, preventing other programs from accessing classes that are meant for internal use only
- **Maintenance:** With packages, you can organize your project better and easily locate related classes

Thus, package is a container of a group of related classes where some of the classes are accessible and are exposed and others are kept for internal purpose. We can reuse existing classes from the packages as many time as we need it in our program. Package names and directory structure are closely related. For example if a package name is *college.staff.csc*, then there are three directories, *college*, *staff* and *csc* such that *csc* is present in *staff* and *staff* is present *college*.

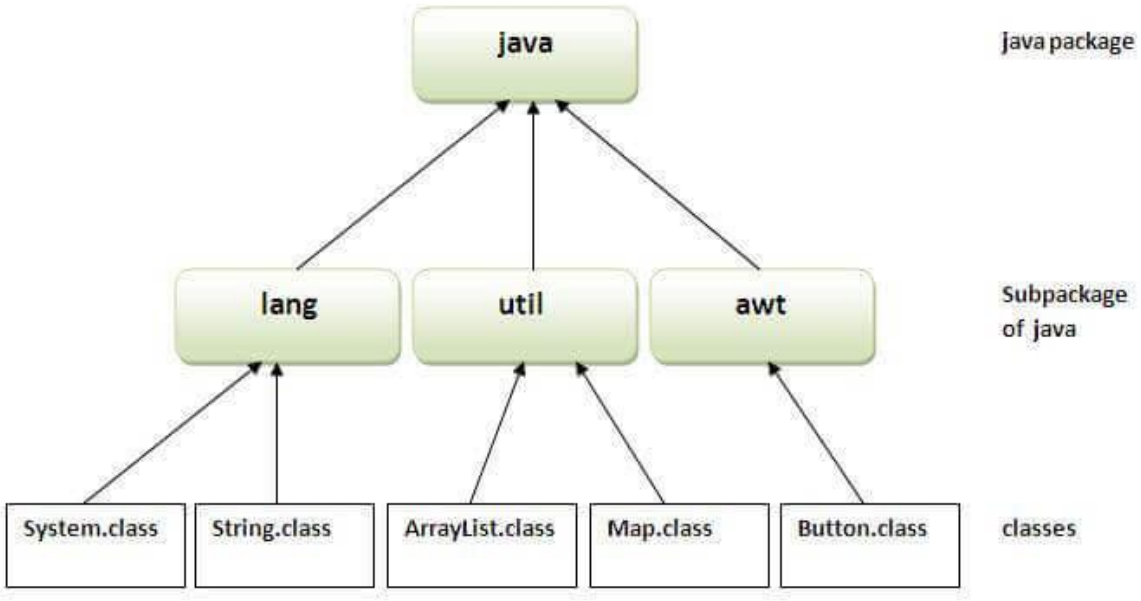
Package naming conventions: Packages are named in reverse order of domain names. For example, in a college, the recommended convention is college.tech.csc, college.art.history, etc.

Types of Packages in Java

Based on whether the package is defined by the user or not, packages are divided into two categories:

1. *Built-in Packages:*
   Built-in packages or predefined packages are those that come along as a part of JDK (Java Development Kit) to simplify the task of Java programmer. They consist of a huge number of predefined classes and interfaces that are a part of Java API's. Some of the commonly used built-in packages are:
   - **java.lang:** Contains language support classes (e.g classed which defines primitive data types, math operations). This package is automatically imported.
   - **java.io:** Contains classes for supporting input / output operations.
   - **java.util:** Contains utility classes which implement data structures like Linked List, Dictionary and support for Date / Time operations.

- j**ava.applet:** Contains classes for creating Applets.
- **java.awt:** Contain classes for implementing the components for graphical user interfaces (like button, menus etc).
- **java.net:** Contain classes for supporting networking operations.



2. *User Defined Packages:*
   User-defined packages are those which are developed by users in order to group related classes, interfaces and sub packages

   Example:
   In this example we create a user defined package named 'MyPackage' within which a program to compare two numbers is written. This package is later imported into a demo java program to demonstrate the usage of user-defined packages.

```
package MyPackage;
 public class Compare {
  int num1, num2;

  public Compare(int n, int m) {
     num1 = n;
     num2 = m;
}
public void getmax(){
    if ( num1 > num2 ) {
        System.out.println("Maximum value of two numbers is " + num1);
```

```
  }
   else {
      System.out.println("Maximum value of two numbers is " + num2);
   }
}

 public static void main(String args[]) {
      Compare current[] = new Compare[3];

      current[1] = new Compare(5, 10);
      current[2] = new Compare(123, 120);

        for(int i=1; i < 3 ; i++)
           {
           current[i].getmax();
           }
     }
}
```

Steps to compile and run a user defined package:

- *Save the above program as Compare.java.*
- *Then compile the program using the command :* ***javac Compare.java***
- *Compiling the program creates a file '****Compare.class****' in the same destination were '****Compare.java****' has been placed. However, no package has been created.*
- *Crete a package by executing the following compile command:*
  javac –d . Compare.java . *This creates a folder named* ***'MyPackage'*** *with* ***'Compare.class'*** *inside the folder. The -d option is used to specify the base directory of the compiled class, In this case we have specified the current working directory(using a .) as the base directory.*
- *Next we can execute the program by using the command:*
  java MyPackage.Compare

// Demo program that uses the above package.

```
import MyPackage.Compare;

public class Demo{
   public static void main(String args[]) {
      int n=10, m=10;
      Compare current = new Compare(n, m);
      if(n != m) {
          current.getmax();
      }
      else {
```

```
            System.out.println("Both the values are same");
        }
}
}
```

Steps to compile and run a program importing a user defined package:

- *Case 1: When the program is in the same directory as the package:*
  - *In this case you can compile and run the program with the following commands:*
  - *Compile: javac Demo.java*
  - *Run: java Demo*

- *Case 2: When the program is in a different directory compared to the package:*
  - In this case two sub cases arise:
    - Case a: Specifying the classpath in each command explicitly.
      We use –cp option to specify the base directory of our user defined package 'MyPackage' as shown in the command below. Without it the compiler will display an error message that the package 'MyPackage' doesn't exist.

      javac –cp C:\Users Demo.java

      To execute the program we again need to use the –cp option without which the JRE won't be able to locate the user defined package 'MyPackage'.

      java –cp C:\Users Demo

      However the JRE will still generate an error as it can't even find the Demo.class file, which is located in the current directory. This is because if CLASSPATH is not explicitly set, it is defaulted to the current directory. However, if CLASSPATH is explicitly set, it does not include the current directory unless the current directory is included. Hence, we need to include current directory (denoted as '.') in the CLASSPATH, together with the base directory of user defined package 'MyPackage', separated by ';', as follows:

      java –cp .;C:\Users Demo

    - Case b: Setting the classpath in environmental variable:
      CLASSPATH is an environment variable (i.e., global variables of the operating system available to all the processes) needed for the Java compiler and runtime to locate the Java packages/classes used in a Java program. CLASSPATH can be set in one of the following ways:

CLASSPATH can be set permanently in the environment: In Windows, choose control panel ⇒ System ⇒ Advanced ⇒ Environment Variables ⇒ choose "System Variables" (for all the users) or "User Variables" (only the currently login user) ⇒ choose "Edit" (if CLASSPATH already exists) or "New" ⇒ Enter "CLASSPATH" as the variable name ⇒ Enter the required directories and JAR files (separated by semicolons) as the value (e.g., ".;C:\Users;d:\tomcat\lib\servlet-api.jar"). We also need to include the current working directory (denoted by '.') in the CLASSPATH.
CLASSPATH can be set temporarily for that particular CMD shell session by issuing the following command:

> SET CLASSPATH=.;C:\Users;d:\tomcat\lib\servlet-api.jar

Using Static Import

Static import is a feature introduced in **Java** programming language (versions 5 and above) that allows members (fields and methods) defined in a class as public **static** to be used in Java code without specifying the class in which the field is defined.
Following program demonstrates **static import**:

```
import static java.lang.System.*;

class StaticImportDemo
{
  public static void main(String args[])
  {
      // We don't need to use 'System.out'
      // as imported using static.
      out.println("Static keyword demo");
  }
}
```

Output:
Static keyword demo