Packages in JAVA

By: Prasad. M. Pujar

PACKAGES

- Packages in Java is a mechanism to encapsulate a group of classes, interfaces and sub packages.
- In java there are already many predefined packages that we use while programming.
    - For example: java.lang, java.io, java.util etc.
- One of the most useful feature of java is that we can define our own packages

PACKAGES Cont...

Advantages of using a package

- Reusability: Reusability of code is one of the most important requirements in the software industry. Reusability saves time, effort and also ensures consistency. A class once developed can be reused by any number of programs wishing to incorporate the class in that particular program.
- Easy to locate the files.
- In real life situation there may arise scenarios where we need to define files of the same name. This may lead to "name-space collisions". Packages are a way of avoiding "name-space collisions".

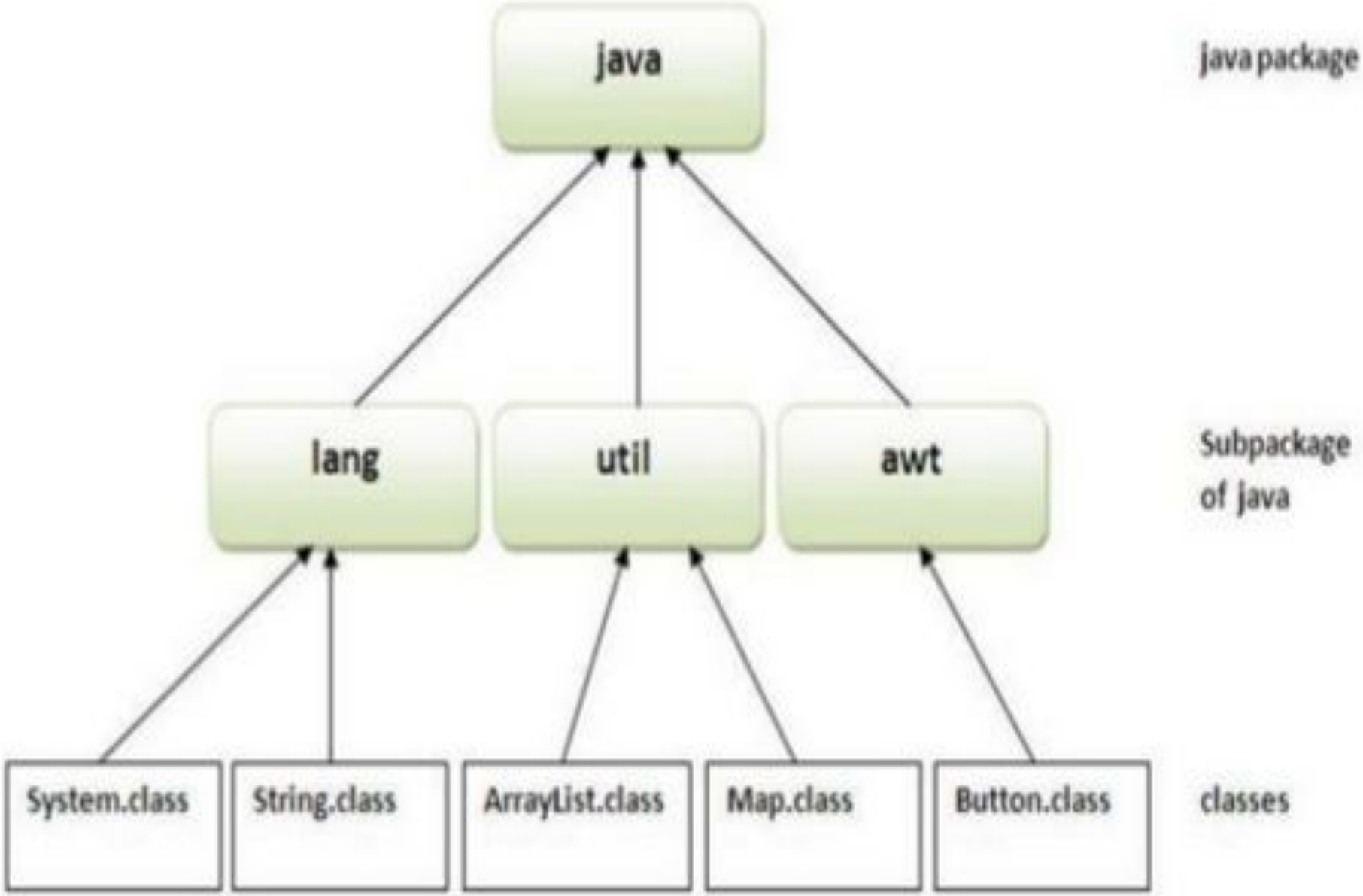
PACKAGES Cont...

- Many implementations of Java use a hierarchical file system to manage source and class files. It is easy to organize class files into packages. All we need to do is put related class files in the same directory, give the directory a name that relates to the purpose of the classes, and add a line to the top of each class file that declares the package name, which is the same as the directory name where they reside.

- Types of package:

  - 1) User defined package: The package we create is called user-defined package.

  - 2) Built-in package: The already defined package like java.io.*, java.lang.* etc are known as built-in packages.


java
java package
lang
util
awt
Subpackage
of java
System.class
String.class
ArrayList.class
Map.class
Button.class
classes

PACKAGES Cont...

Defining a Package:

- This statement should be used in the beginning of the program to include that program in that particular package.

  package <package name>;

- The **package keyword** is used to create a package in java.

To Compile: javac -d directory javafilename.java

To Run: java packagename.classname

- The -d switch specifies the destination where to put the generated class file
- If you want to keep the package within the same directory, you can use . (dot).

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 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.

Using packagename.classname

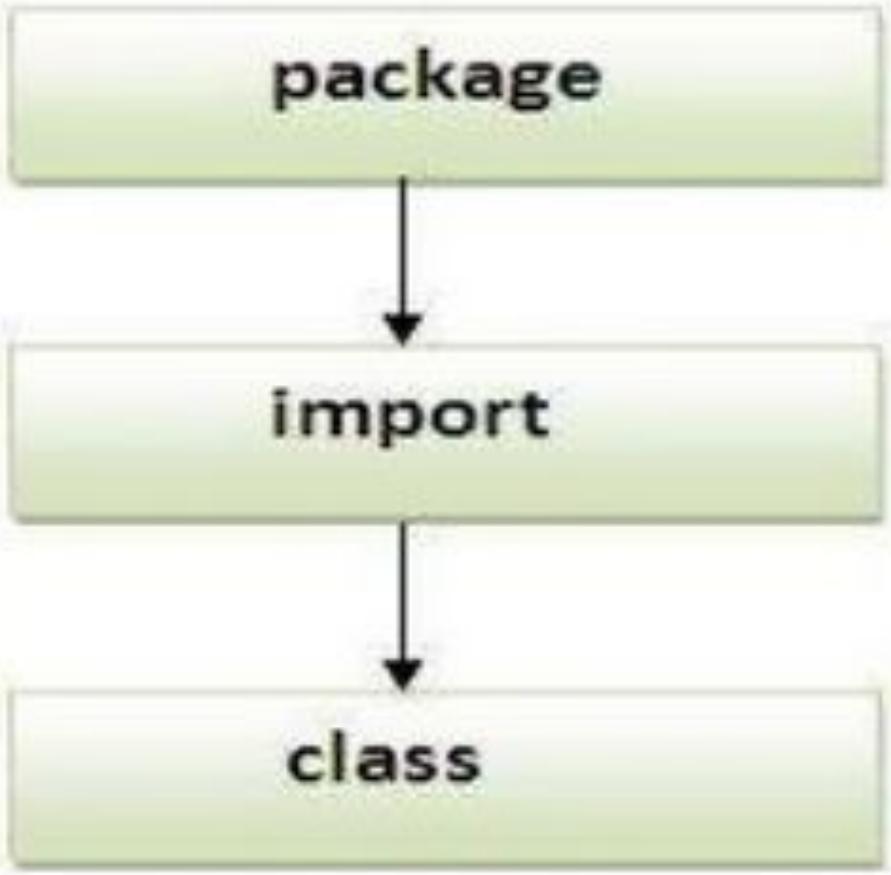
- If you import package.classname then only declared class of this package will be accessible.

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.

Note: If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.

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



Packaging up multiple classes

Package with Multiple Public Classes

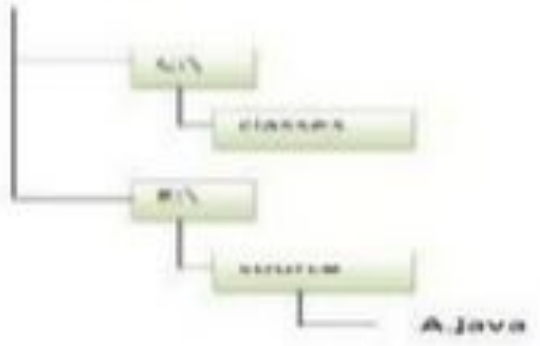
- A Java source file can have only one class declared as **public**, we cannot put two or more public classes together in a **.java** file. This is because of the restriction that the file name should be same as the name of the public class with **.java**extension.
- If we want to multiple classes under consideration are to be declared as **public**, we have to store them in **separate source files** and attach the **package** statement as the first statement in those source files.

```
//Save as A.java
package javapoint;
Public class B{}
```

```
//Save as B.java
package java
public class a{}
```

CLASSPATH

- It is a environmental variable, which contains the path for the default-working directory (.).
- The specific location that java compiler will consider, as the root of any package hierarchy is, controlled by Classpath
- The package name is closely associated with the directory structure used to store the classes. The classes (and other entities) belonging to a specific package are stored together in the same directory.
- There is a scenario, I want to put the class file of A.java source file in classes folder of c: drive. For example:



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

To Compile:

e:\sources> javac -d c:\classes Simple.java

To Run:

To run this program from e:\source directory, you need to set classpath of the directory where the class file resides.

e:\sources> set classpath=c:\classes;

e:\sources> java mypack.Simple

Another way to run this program by -classpath switch of java:

- The -classpath switch can be used with javac and java tool.
- To run this program from e:\source directory, you can use -classpath switch of java that tells where to look for class file. For example:
    - **e:\sources> java -classpath c:\classes mypack.Simple**

Access Protection in Packages

- Access modifiers define the scope of the class and its members (data and methods).
- There are four categories, provided by Java regarding the visibility of the class members between classes and packages:
    - Subclasses in the same package
    - Non-subclasses in the same package
    - Subclasses in different packages
    - Classes that are neither in the same package nor subclasses

The three main access modifiers *private*, *public* and *protected* provides a range of ways to access required by these categories.

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
same package subclass	No	Yes	Yes	Yes
same package non - subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package mon-subclass	No	No	No	Yes

Packages	Description
Java.lang	It is a default package which contain primitive data type, displaying result on console screen, obtaining garbage collector etc.
java.io	It used for developing file handling applications, such as, opening the file in read or write mode, reading or writing the data, etc.
java.awt	This package is used for developing GUI (Graphic User Interface) components such as buttons, check boxes, scroll boxes, etc.
Java. applet	This package is used for developing browser oriented applications.
java.net	This package is used for developing client server applications.
java.util	Contains utility classes which implement data structures like Hash Table, Dictionary, etc.
java.sql	This package is used for retrieving the data from data base and performing various operations on data base.

Static Import

- Static Import is a new feature added in java.
- In order to access static members, it is necessary to qualify references with the class they came from.
- That means in order to access the static members of a class, it is a must to write the member along with its belonging class name.

Static Import

- For Example:

```
class A
{
Public static void main(String a[])
{
Int a=36;
Int val=Math.sqrt(a);
System.out.println("The square root of"+a"is"+val);
}
}
```

o/p : The square root of 36 is 6

In order to avoid unnecessary use of static class members like Math and System, we should Static import.

Static Import

```
Import static java.lang.System.out;

Import static java.lang.Math.sqrt;

class A

{

Public static void main(String a[])

{
```

```
Int a=36;

Int val=sqrt(a);

System.out.println("the square root of" +a"is"+val);

}

}
```

- O/p : the square root of 36 is 6