Java File Structure

Introduction

1. A java Program can contain any no. Of classes but at most one class can be declared as public.
2. "If there is a public class the name of the Program and name of the public class must be matched otherwise we will get compile time error".
3. If there is no public class then any name we can give for java source file.

```
class A{}
class B{}
class C{}
```

Case 1:

- If there is no public class then we can use any name for java source file there are no restrictions.

Case 2:

- If class B declared as public then the name of the Program should be B.java otherwise we will get compile time error saying "class B is public, should be declared in a file named B.java".

Case 3:

- If both B and C classes are declared as public and name of the file is B.java then we will get compile time error saying "class C is public, should be declared in a file named C.java".
- It is highly recommended to take only one class for source file and name of the Program (file) must be same as class name. This approach improves readability and understandability of the code.

- We can compile a java Program but not java class. In that Program for every class one dot class file will be created.
- We can run a java class but not java source file. Whenever we are trying to run a class the corresponding class main method will be executed.
- If the class won't contain main method then we will get runtime exception saying "NoSuchMethodError: main".
- If we are trying to execute a java class and if the corresponding .class file is not available then we will get runtime execution saying "NoClassDefFoundError: Sample".

Import Statements

Types of Import Statements:

There are 2 types of import statements.

1) Explicit class import

Import java.util.Scanner

- This type of import is highly recommended to use because it improves readability of the code.

2) Implicit class import.

import java.util.;*

- It is not recommended to use because it reduces readability of the code.

Whenever we are using fully qualified name it is not required to use import statement. Similarly whenever we are using import statements it is not require to use fully qualified name.

- We may get the ambiguity problem because it may be available in multiple packages.
- While resolving class names compiler will always gives the importance in the following order.

  1. Explicit class import
  2. Classes present in current working directory.
  3. Implicit class import.

- Whenever we are importing a package all classes and interfaces present in that package are by default available but not sub package classes. Explicit import is required.
- In any java Program the following 2 packages are not require to import.

  1. java.lang package
  2. default package(current working directory)

"Import statement is totally compile time concept" if more no of imports are there then more will be the compile time but there is "no change in execution time".

- In the case of C language #include all the header files will be loaded at the time of include statement hence it follows static loading.
- But in java import statement no ".class" will be loaded at the time of import statements in the next lines of the code whenever we are using a particular class then only corresponding ".class" file will be loaded. Hence it follows "dynamic loading" or "load-on -demand" or "load-on-fly".

Usually we can access static members by using class name but whenever we are using static import it is not require to use class name we can access directly.

```
System.out.println(Math.sqrt(4));
System.out.println(Math.max(10,20));
System.out.println(Math.random());
```

With static import:

```
import static java.lang.Math.sqrt;
import static java.lang.Math.*;
System.out.println(sqrt(4));
System.out.println(max(10,20));
System.out.println(random());
```

Packages In Java

Packages

- A **Package** can be defined as a grouping of related types (classes, interfaces, enumerations, annotations and sub-packages) providing access protection and name space management.
- Packages are use to control access of classes, interface, enumeration etc. and avoid namespace collision.
- There can not be two classes with same name in a same Package
- But two packages can have a class with same name.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

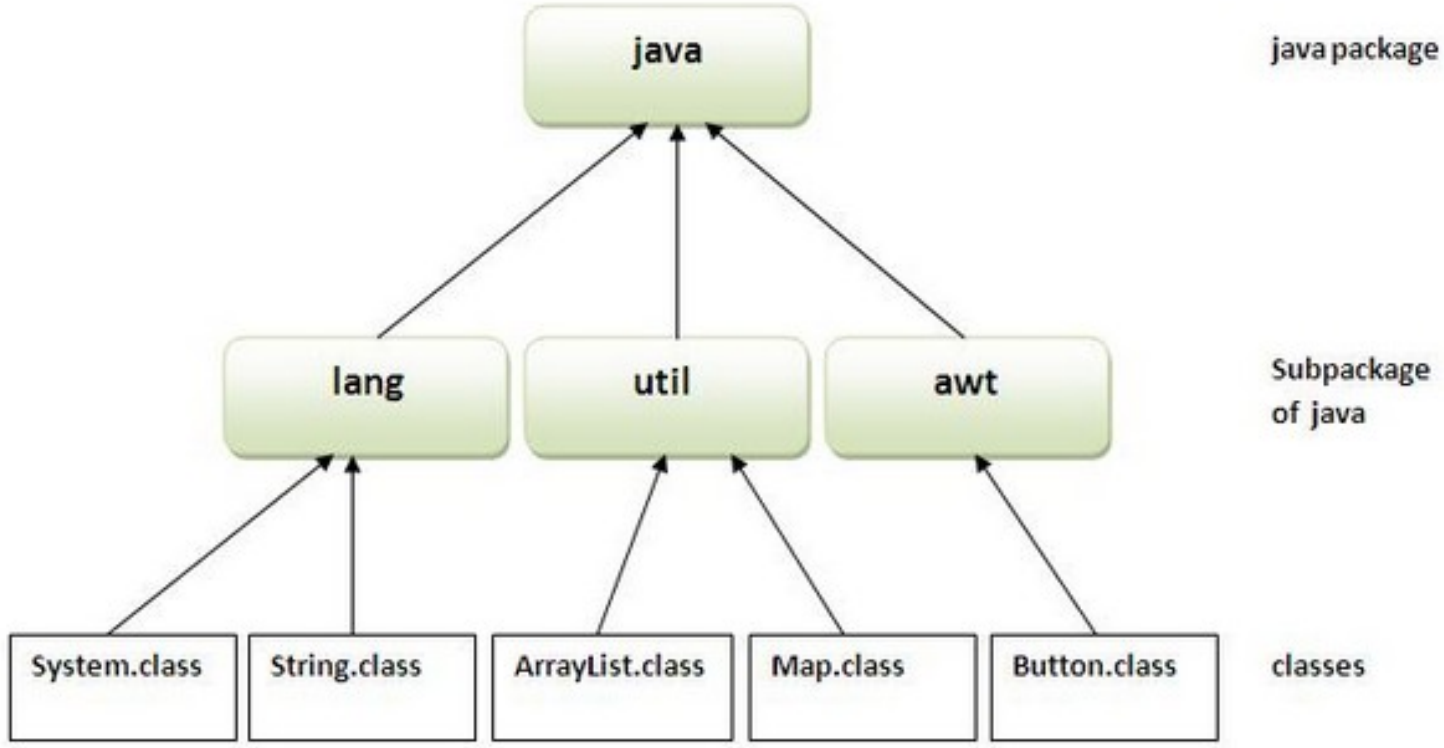
Packages

- Package names are dot separated, e.g., java.lang.
- Package names have a correspondence with the directory structure.
- Exact Name of the class is identified by its package structure.
- << Fully Qualified Name>>
- java.lang.String ;
- java.util.Arrays;
- java.io.BufferedReader ;
- java.util.Date

Advantage of Java Package

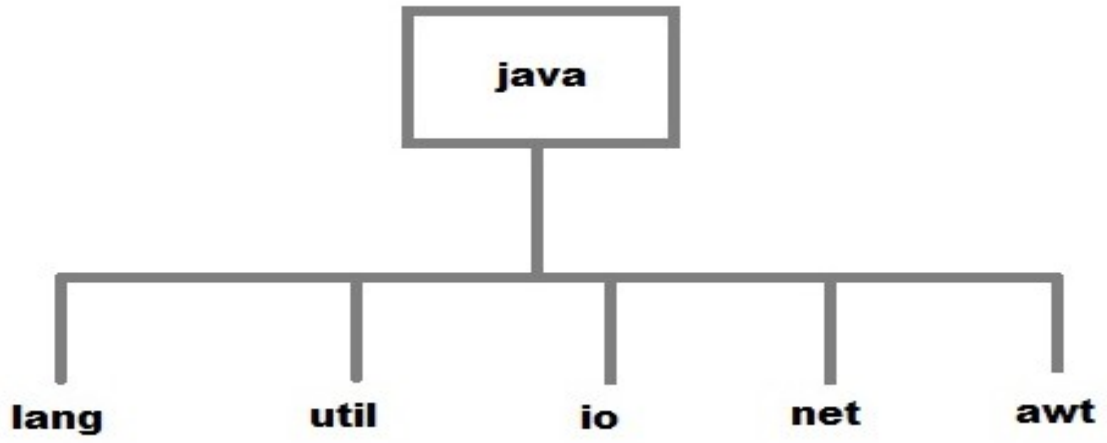
- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.

Package Hierarchy



Con't

- Package are categorized into two forms:
  - **Built-in** java package : java.lang, java.util etc.



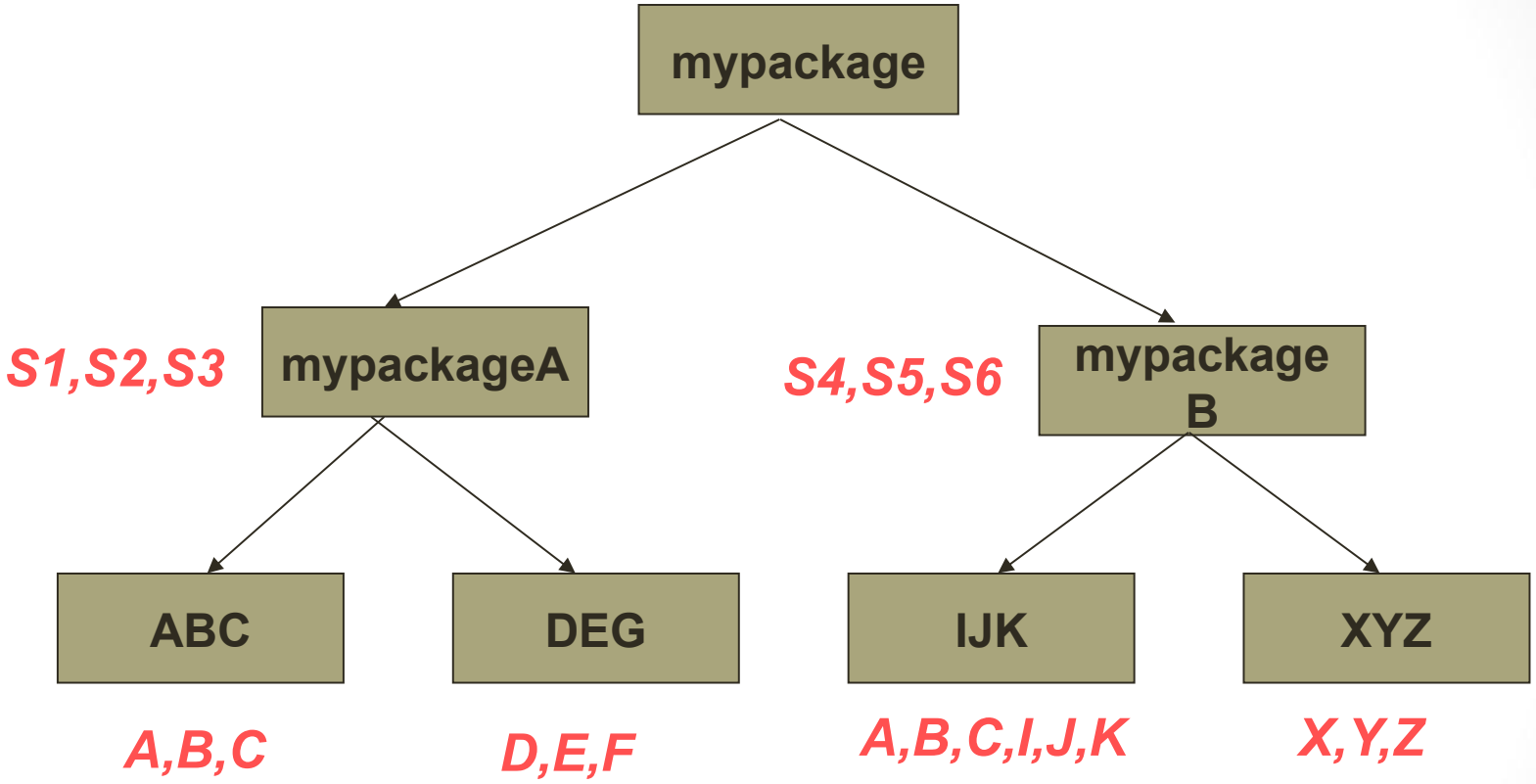
  - **User-defined** packages : Java Package created by user to categorize classes and interface.

How To Create a Package

- To create a package, First we have to create a directory /directory structure that matches the package hierarchy.
- To make a class belongs to a particular package include the package statement as the first statement of source file.
- There can be only one package statement in each source file, and it applies to all types in the file.

- I:\5\mypack\Student.java
- In mypack folder, file Student.java (we need to be in - I:\5)
  - Compile: javac mypack/Student.java (compiler operates on files)
  - Run : java mypack.Student (interpreter loads a class)

Creating Packages



- Package ABC and IJK have classes with same name.
- A class in ABC has name mypackage.mypackageA.ABC.A
- A class in IJK has name mypackage.mypackageB.IJK.A

How to make a class Belong to a Package

- Include a proper package statement as first line in source file

Make class S1 belongs to mypackageA

```
package mypackage.mypackageA;
   public class S1    {
      public S1( )       {
      System.out.println("This is Class S1");
      }
}
```

Name the source file as S1.java and compile it and store the S1.class file in mypackageA directory

Make class S2 belongs to mypackageA

```
package mypackage.mypackageA;
    public class S2
    {
        public S2( )
        {
            System.out.println("This is Class S2");
         }
    }
```

Name the source file as S2.java and compile it and store the S2.class file in mypackageA directory

Make class A belongs to IJK

```
package mypackage.mypackageB.IJK;
    public class A
    {
        public A( )
        {
            System.out.println("This is Class A in IJK");
        }
}
```

Name the source file as A.java and compile it and store the A.class file in IJK directory

<< Same Procedure For all classes>>

Make class A belongs to IIK

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

```
package mypack;
import pack.*;

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