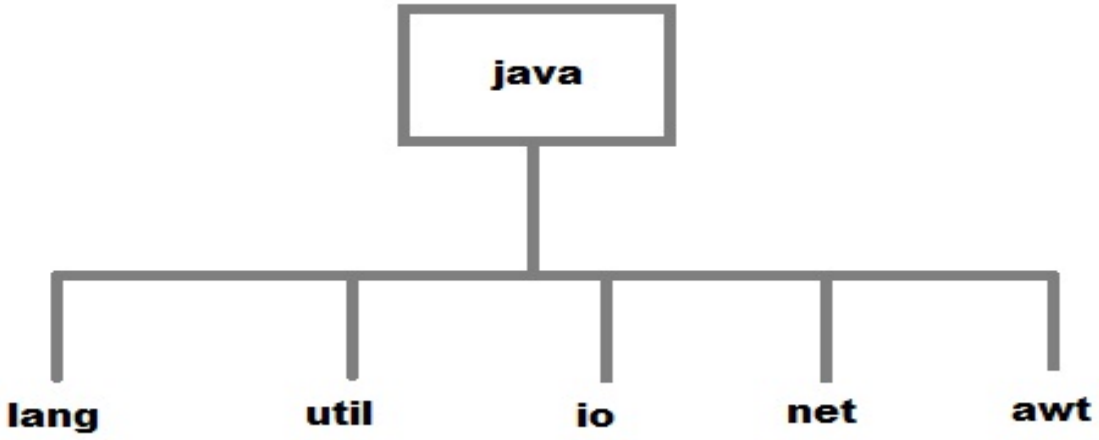
Packages In Java

Packages

- Packages enable grouping of functionally related classes(i.e. classes, interface, enumeration, annotation)
- Package names are dot separated, e.g., java.lang.
- Package names have a correspondence with the directory structure.
- Packages are use to control access of classes, interface, enumeration etc. and avoid name space collision.
  - There can not be two classes with same name in a same Package
  - But two packages can have a class with same name.
- Exact Name of the class is identifed by its package structure. << Fully Qualified Name>>
  - java.lang.String ;
  - java.util.Arrays;
  - java.io.BufferedReader ;
  - java.util.Date

Con't

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



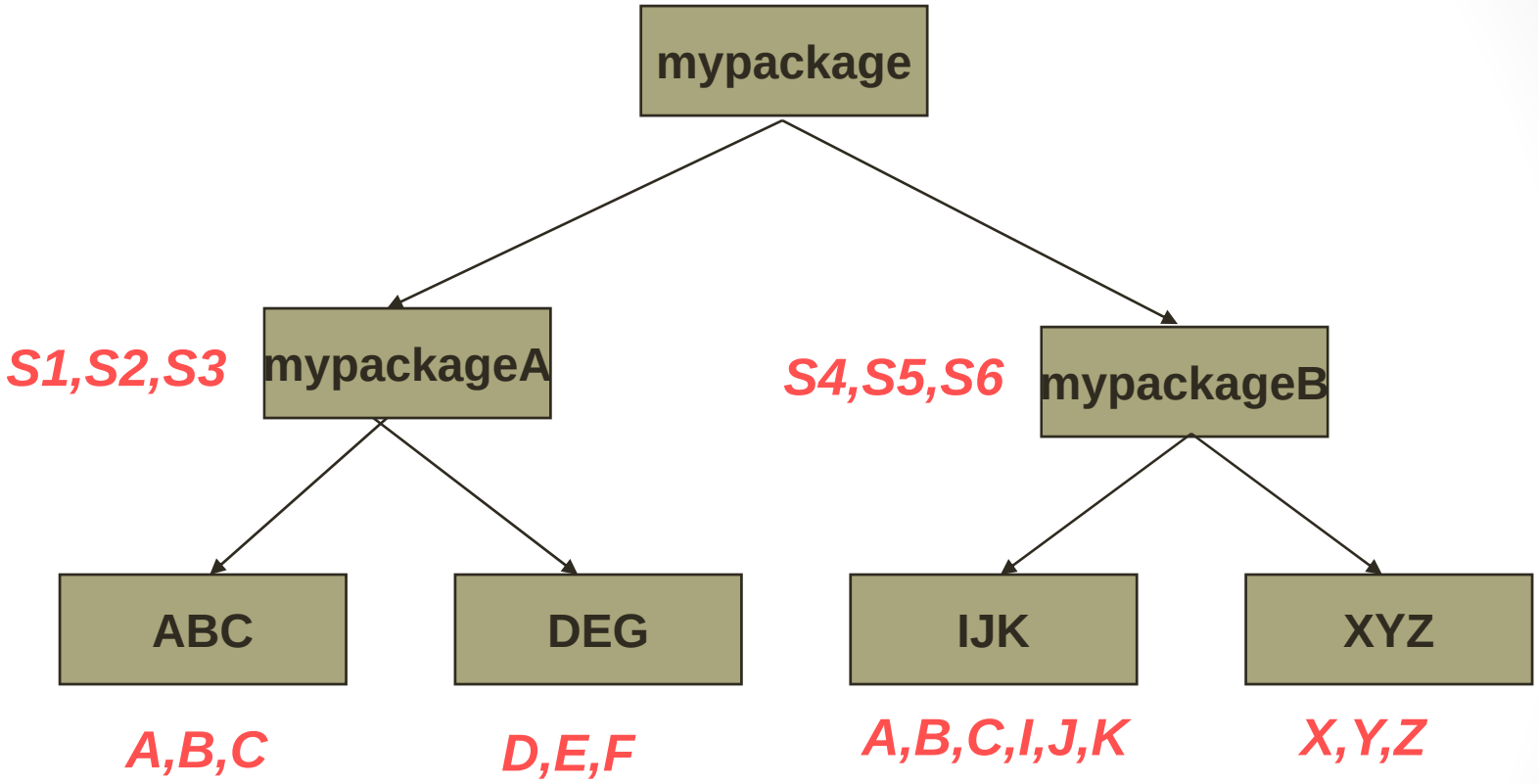
  - User-defined packages : Java Package created by user to categorize classes and interface.

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.

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

Exercise 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>>

Import keyword

- A class can use all classes from its own package and all public classes from other packages.

3 ways to refer to a class present in different package:

- Using fully qualified name
  - Class MyDate extends java.util.Date { … }
- Import the only class you want to use
  - import java.util.date;
  - class MyDate extends Date{ … }
- Import all the classes from the particular package
  - import java.util.*;
  - class MyDate extends Date{ … }
- Import statement is kept after package statement
  - E.g. package mypack;

    import java.util.*;

Importing the Package

- import statement allows the importing of package
- Library packages are automatically imported irrespective of the location of compiling and executing program
- JRE looks at two places for user created packages

  (i) Under the current working directory

  (ii) At the location specified by CLASSPATH environment variable
- Most ideal location for compiling/executing a program is immediately above the package structure.

Employee.java

Boss.java

static import

- To import static member of class.
- Using static import it is possible to refer static member without even using class name.
- Import static package.class-name.static-member-name;
  - E.g. import static java.lang.Math.sqrt;
- Import static package.class-type-name.*;
  - E.g. import static java.lang.Math.*;

- Test.java

Example importing

import mypackage.mypackageA.ABC;

```
import mypackage.mypackageA.ABC.*;
class packagetest
{
        public static void main(String args[])
        {
            B b1 = new B();
            C c1 = new C();
        }
}
```

<< packagetest.java>>

This is Class B

This is Class C

<< Store it in location above the package structure. Compile and Execute it from there>>

```
import mypackage.mypackageA.ABC.*;
import mypackage.mypackageB.IJK.*;
class packagetest
{
        public static void main(String args[])
        {
                A a1 = new A();
        }
}
```

<< What's Wrong Here>>

```
mypackage.mypackageA.ABC.A a1 = new mypackage.mypackageA.ABC.A();
```

OR

```
mypackage.mypackageB.IJK.A a1 = new mypackage.mypackageB.IJK.A();
```

<< class A is present in both the imported packages ABC and IJK. So A has to be fully qualified in this case>>

CLASSPATH Environmental Variables

- CLASSPATH Environmental Variable lets you define path for the location of the root of the package hierarchy
- Consider the following statement :

  package mypack;

  What should be true in order for the program to find mypack.

  (i) Program should be executed from the location immediately above mypack

  OR

  (ii) mypack should be listed in the set of directories for CLASSPATH

The Directory Structure of Packages:

- In general, a company uses its reversed Internet domain name for its package names.
  - Eg: A company's Internet domain name is apple.com, then all its package names would start with com.apple.
- Example: The company had a com.apple.computers package that contained a Dell.java source file.
  - // File Name: Dell.java
  - package com.apple.computers;
  - public class Dell{ }
  - class Ups{ }

What is jar files? How to create it?

- JAR : It is a java archive files used to package classes, files etc. as single file.
  - This is similar to zip file in windows.
- To create a jar file with all class files under the current directory.
  - jar –cvf jarfilename.jar *.class
  - c - to *create* a JAR file.
  - f - the output goes to a *file* rather than to stdout.
  - v -Produces *verbose* output on stdout while the JAR file is being built.
    - The verbose output tells you the name of each file as it's added to the JAR file.
- Can jar code be retrieved from byte code?
  - Yes, class files can be decompiled using utilities like JAD(JAva Decompiler). This takes the class files as an input and generates a java file.