CHAPTER-8
PACKAGES: PUTTING CLASSES TOGETHER

Introduction

- One of the main features of OOP is its ability to reuse the code already created. One way of achieving this is by extending the classes and implementing the interfaces. This is limited to reusing the classes within a program.
- If we need to use classes from other programs we have to use packages.
- Packages are Java's way of grouping a variety of classes and/or interfaces together. The grouping is usually done according to functionality.
- In fact, packages act as "containers" for classes.

Java packages are classified into two types.

1. Java API packages
2. User defined packages.

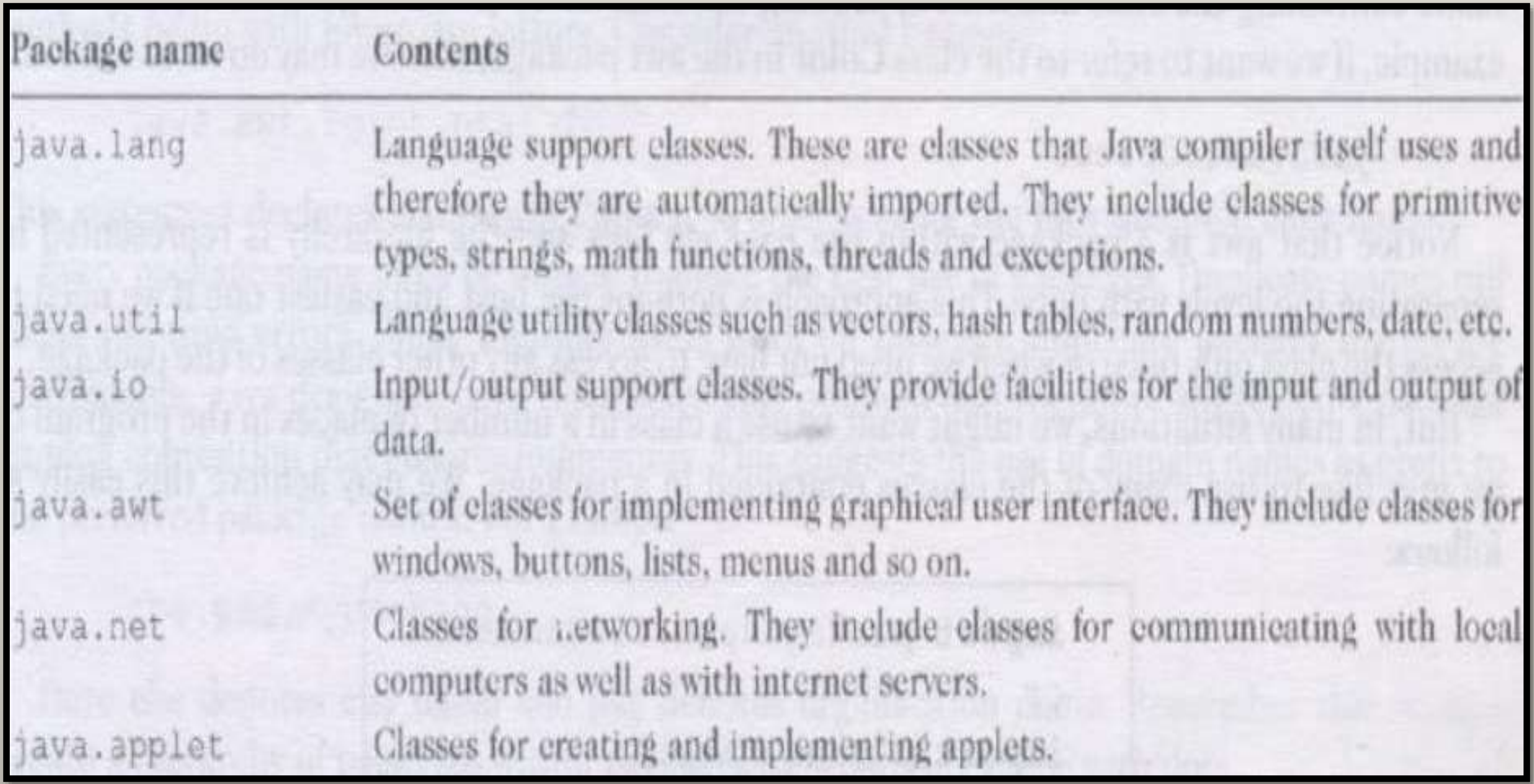
Advantages:

1. The classes contained in the packages of other programs can be easily reused.
2. In packages, classes can be unique compared with classes in other packages. That is, two classes in two different packages can have the same name
3. Packages provide a way to "hide" classes thus preventing other programs or packages from accessing classes that are meant for internal use only.
4. Packages also provide a way for separating "design" from "coding". First we can design classes and decide their relationships, and then we can implement the Java code needed for the methods.

Java API packages:

Java API provides a large number of classes grouped into different packages according to functionality.

Java System Packages and Their Classes



Package name	Contents
`java.lang`	Language support classes. These are classes that Java compiler itself uses and therefore they are automatically imported. They include classes for primitive types, strings, math functions, threads and exceptions.
`java.util`	Language utility classes such as vectors, hash tables, random numbers, date, etc.
`java.io`	Input/output support classes. They provide facilities for the input and output of data.
`java.awt`	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.
`java.net`	Classes for networking. They include classes for communicating with local computers as well as with internet servers.
`java.applet`	Classes for creating and implementing applets.

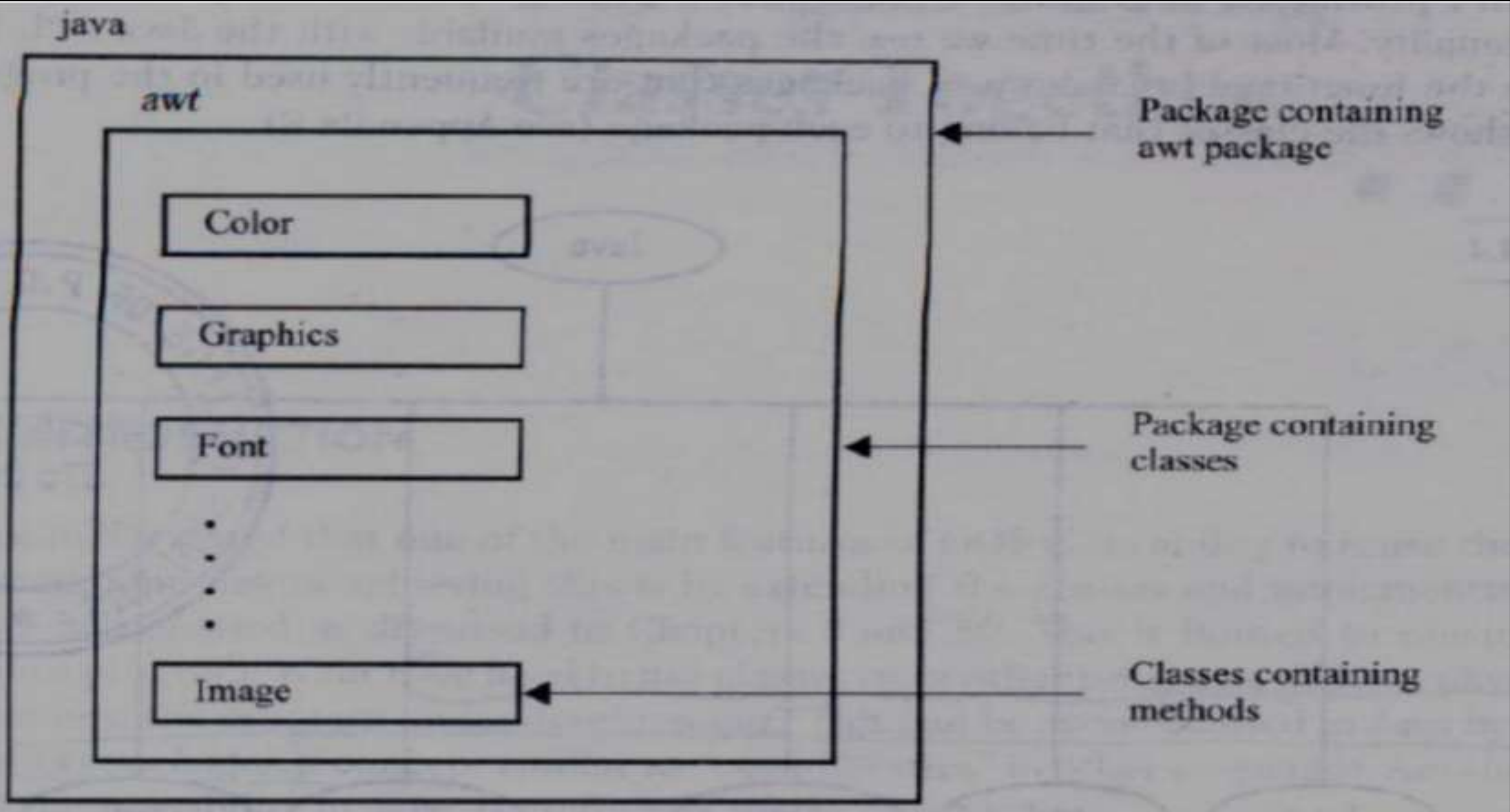
Hierarchical structure of a package in Java with an example

- The packages are organized in a hierarchical structure. This shows that the package named java contains the package awt, which in turn contains various classes required for implementing graphical user interface

There are two ways of accessing the classes stored in a package.

- The first approach is to **use the fully qualified class name** of the class that we want to use. This is done by using the **package name containing the class** and then appending the class name to it using the **dot operator**.
- **For example,** if we want to refer to the class Color in the awt package, then we may do so as follows: **java.awt.Colour**
- **Importing the Package:** Instead of using the fully qualified class name every time, we can import the package at the beginning of our Java file. **For example: import java.awt.Color;**
- We can use Color in our code without specifying the package every time

Color myColor = new Color(255, 0, 0); // Creating a red color object


java
awt
Color
Graphics
Font
Image
Package containing awt package
Package containing classes
Classes containing methods

Hierarchical representation of java.awt package.

Naming conventions

- Packages can be named using the standard Java naming rules. By convention, however, packages begin with lowercase letters. This makes it easy for users to distinguish package names from class names .
- Again by convention, all class names begin with an uppercase letter.

For example: double y= java.lang.Math.sqrt(x);

This statement uses a fully qualified class name Math to invoke the method sqrt ().

- Every package name must be unique to make the best use of packages. Duplicate names will cause run-time errors.
- To ensure uniqueness, Java designers suggest using domain names as a prefix to predefined package names.

For example: cbe.psg.mypackage

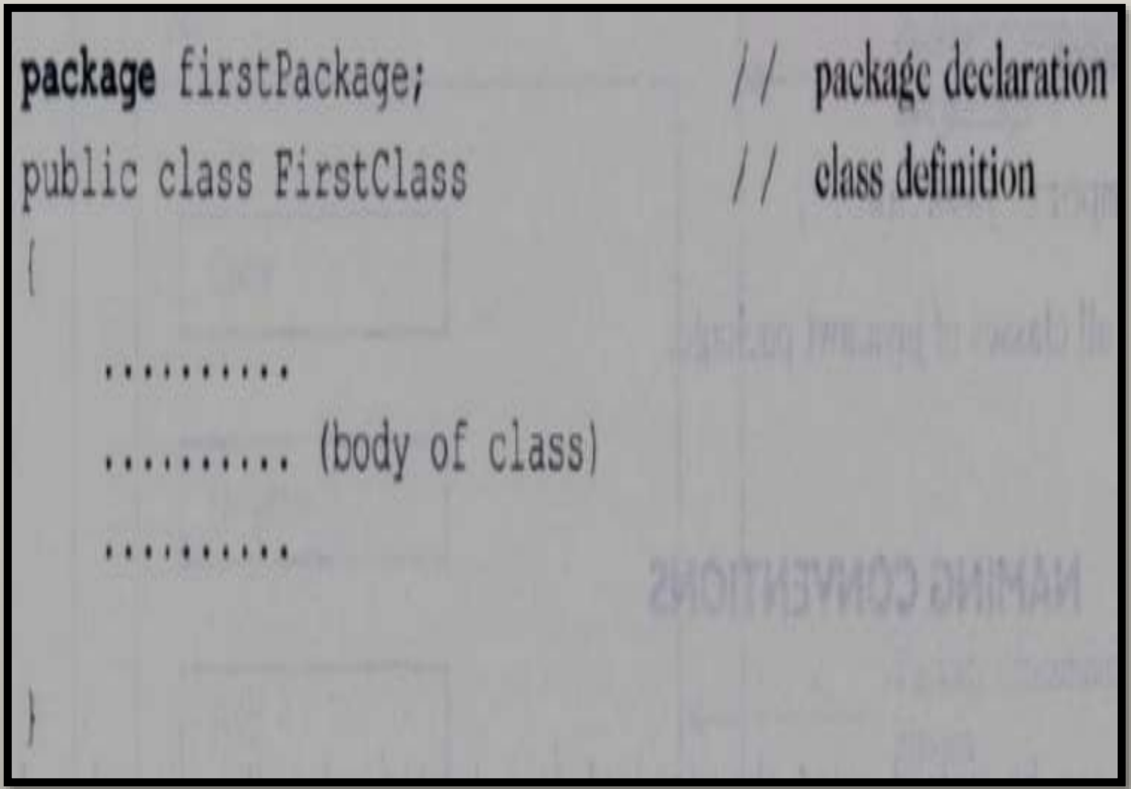
Here cbe denotes city name and psg denotes organization name.

Creating packages

In order to create a package we must first declare the name of the package using the package keyword followed by a package name.

This must be the first statement in a Java source file (except for comments and white spaces). Then we define a class, just as we normally define a class.

Example:



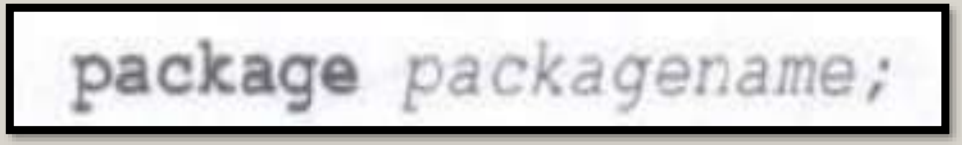
```
package firstPackage;                 // package declaration
public class FirstClass               // class definition
{
    ..........
    .......... (body of class)
    ..........

}
```

Here the package name is **firstPackage**. The class FirstClass is now considered a part of this package. This listing would be saved as a file called **FirstClass.java,** & located in a directory named **firstPackage**. When the source file is compiled, Java will create a .class file and store it in the same directory.

Creating our own package involves the following steps:

1.Declare the package at the beginning of a file using the form



```
package packagename;
```

2. Define the class that is to be put in the package and declare it public.

3. Create a subdirectory under the directory where the main source files are stored.

4. Store the listing as the classname.java file in the subdirectory created.

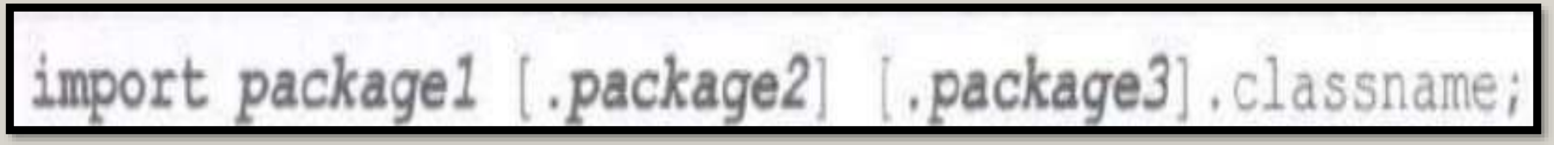
5. Compile the file. This creates .class file in the subdirectory.

Remember that case is significant and therefore the subdirectory name must match the package name exactly Java also supports the concept of package hierarchy. This is done by specifying multiple names in a package statement, separated by dots.

Accessing a Package

- The import statement can be used to search a list of packages for a particular class.

The general form of import statement for searching a class:



```
import package1 [.package2] [.package3].classname;
```

- package1 is the name of the top level package,package2 is the name of the package that is inside the package1, and so on.
- We can have any number of packages in a package hierarchy. Finally, the explicit class name is specified.
- The statement must end with a semicolon (;).
- The import statement should appear before any class definitions in a source file. Multiple import statements are allowed.

Example of importing a particular class:

import firstPackage.secondPackage.MyClass;

After defining this statement, all the members of the class MyClass can be directly accessed using the class name or its objects (as the case may be) directly without using the package name.

Another approach :

import packagename.*;

- Here, packagename may denote a single package or a hierarchy of packages .
- The star (*) indicates that the compiler should search this entire package hierarchy where it encounters a class name. This implies that **we can access all classes contained in the above package directly.**
- The major drawback of the shortcut approach is that it is difficult to determine from which package a particular member came.
- But the advantage is that we need not have to use long package names repeatedly in the program.

<u>**Adding Class To Package**</u>:

The package p1 contains one public class by name A.

Suppose we want to add another class B to this package.

1. Define the class and make it public.

2. Place the package statement package p1; before the class definition

package p1;

public class B

{ / / body of B }

3. Store this as B.java file under the directory p1.

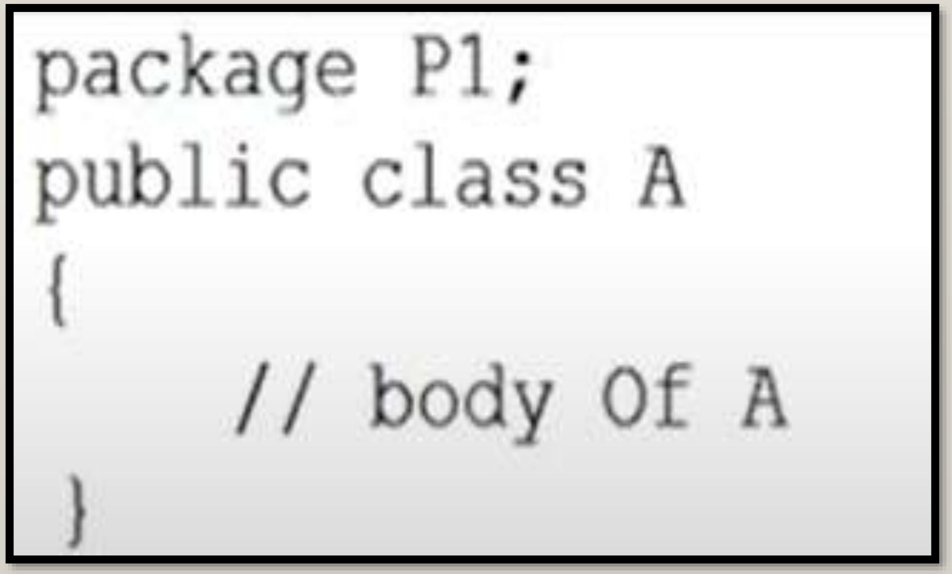
4. Compile B.java file. This will create a B.class file and place it

in the directory pl.

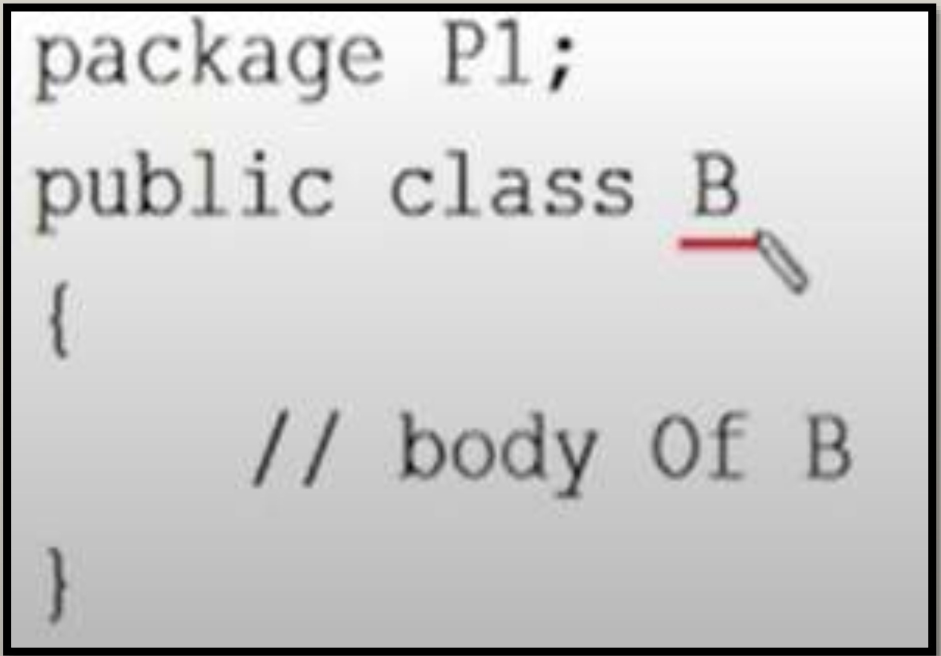
Now the package p1 contain both class A and B.

The Statement like: Import p1.*;

Will import both the classes



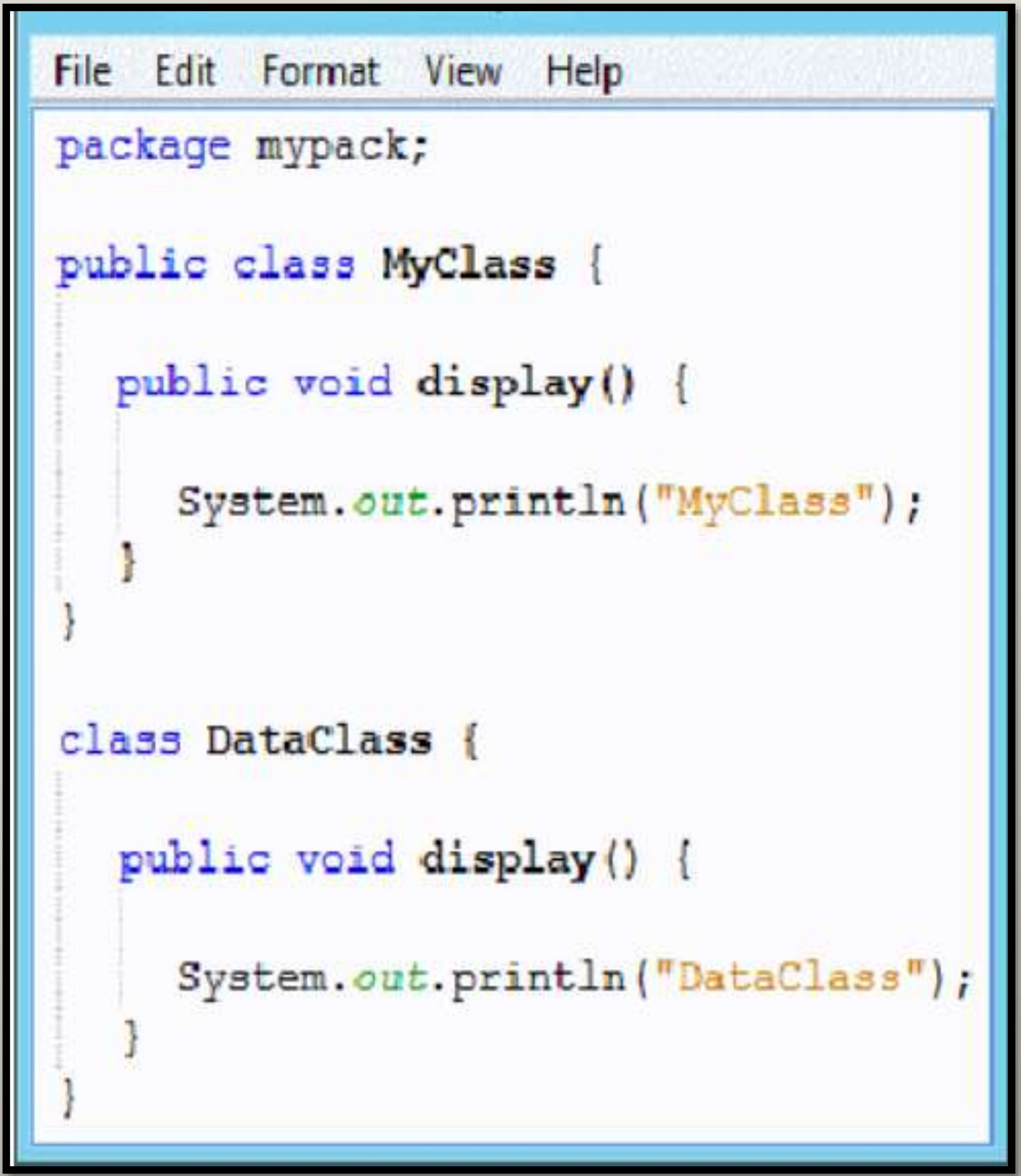
```
package P1;
public class A
{
      // body Of A
 }
```



```
package P1;
public class B
{
      // body Of B
}
```

Hiding classes

- When we import a package within program, only the classes declared as **public** will be made accessible.
- Sometimes we wish certain classes in a package should not be made accessible to importing program. In such case, we need not declare those classes as public. Those classes will be **hidden** from being accessed by the importing class.
- Here, the class DataClass which is not declared public is hidden from outside of the package p1. This class can be seen and used only by other classes in the same package.
- Java compiler would generate an error message for this code because the class DataClass, which has not been, declared public, is not imported and therefore not available for creating its objects.



```
package mypack;

public class MyClass {

  public void display() {

    System.out.println("MyClass");
  }
}

class DataClass {

  public void display() {

    System.out.println("DataClass");
  }
}
```