Packages: Putting Classes Together

Introduction

- The main feature of OOP is its ability to support the reuse of code:
  - Extending the classes (via inheritance)
  - Extending interfaces
- The features in basic form limited to reusing the classes within a program.
- What if we need to use classes from other programs without physically copying them into the program under development ?
- In Java, this is achieved by using what is known as "packages", a concept similar to "class libraries" in other languages.

Packages

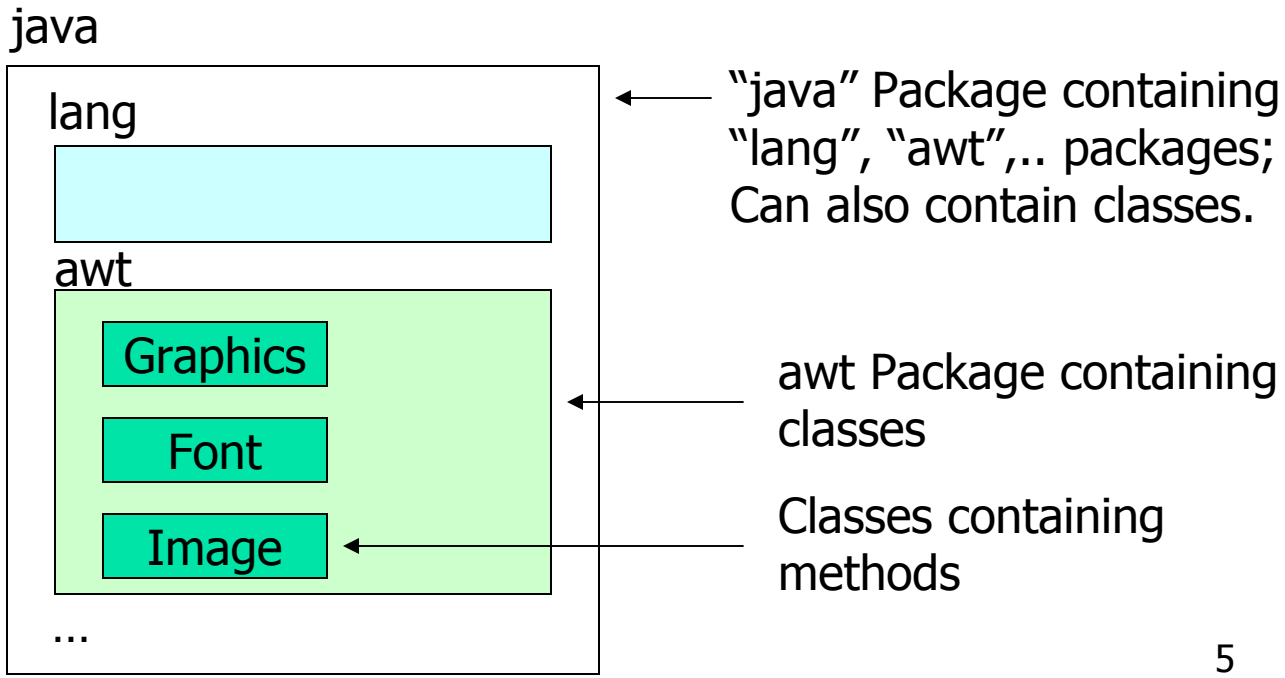
- Packages are Java's way of grouping a number of related classes and/or interfaces together into a single unit. That means, packages act as "containers" for classes.
- The benefits of organising classes into packages are:
  - The classes contained in the packages of other programs/applications can be reused.
  - In packages classes can be unique compared with classes in other packages. That two classes in two different packages can have the same name. If there is a naming clash, then classes can be accessed with their fully qualified name.
  - Classes in packages can be hidden if we don't want other packages to access them.
  - Packages also provide a way for separating "design" from coding.

Java Foundation Packages

- Java provides a large number of classes groped into different packages based on their functionality.
- The six foundation Java packages are:
  - java.lang
    - Contains classes for primitive types, strings, math functions, threads, and exception
  - java.util
    - Contains classes such as vectors, hash tables, date etc.
  - java.io
    - Stream classes for I/O
  - java.awt
    - Classes for implementing GUI – windows, buttons, menus etc.
  - java.net
    - Classes for networking
  - java.applet
    - Classes for creating and implementing applets

Using System Packages

- The packages are organised in a hierarchical structure. For example, a package named "java" contains the package "awt", which in turn contains various classes required for implementing GUI (graphical user interface).



Accessing Classes from Packages

- There are two ways of accessing the classes stored in packages:
  - Using fully qualified class name
    - java.lang.Math.sqrt(x);
  - Import package and use class name directly.
    - import java.lang.Math
    - Math.sqrt(x);
- Selected or all classes in packages can be imported:

```
import package.class;
import package.*;
```

- Implicit in all programs: import java.lang.*;
- package statement(s) must appear first

Creating Packages

- Java supports a keyword called "package" for creating user-defined packages. The package statement must be the first statement in a Java source file (except comments and white spaces) followed by one or more classes.

```
package myPackage;
public class ClassA {
        // class body
}
class ClassB {
 // class body
}
```

- Package name is "myPackage" and classes are considred as part of this package; The code is saved in a file called "ClassA.java" and located in a directory called "myPackage".

Creating Sub Packages

- Classes in one ore more source files can be part of the same packages.
- As packages in Java are organised hierarchically, sub-packages can be created as follows:
  - package myPackage.Math
  - package myPackage.secondPakage.thirdPackage
- Store "thirdPackage" in a subdirectory named "myPackage\secondPackage". Store "secondPackage" and "Math" class in a subdirectory "myPackage".

Accessing a Package

- As indicated earlier, classes in packages can be accessed using a fully qualified name or using a short-cut as long as we import a corresponding package.
- The general form of importing package is:
  - import package1[.package2][…].classname
  - Example:
    - import myPackage.ClassA;
    - import myPackage.secondPackage
  - All classes/packages from higher-level package can be imported as follows:
    - import myPackage.*;

Using a Package

- Let us store the code listing below in a file named "ClassA.java" within subdirectory named "myPackage" within the current directory (say "abc").

```
package myPackage;
public class ClassA {
 // class body
 public void display()
 {
                System.out.println("Hello, I am ClassA");
  }
}
class ClassB {
 // class body
}
```

Using a Package

- Within the current directory ("abc") store the following code in a file named "ClassX.java"

```
import myPackage.ClassA;

public class ClassX
{
        public static void main(String args[])
        {
                ClassA objA = new ClassA();
                objA.display();
        }
}
```

Compiling and Running

- When ClassX.java is compiled, the compiler compiles it and places .class file in current directly. If .class of ClassA in subdirectory "myPackage" is not found, it comples ClassA also.
- Note: It does not include code of ClassA into ClassX
- When the program ClassX is run, java loader looks for ClassA.class file in a package called "myPackage" and loads it.

Using a Package

- Let us store the code listing below in a file named "ClassA.java" within subdirectory named "secondPackage" within the current directory (say "abc").

```
package secondPackage;
public class ClassC {
 // class body
        public void display()
        {
                System.out.println("Hello, I am ClassC");
        }
}
```

Using a Package

- Within the current directory ("abc") store the following code in a file named "ClassX.java"

```
import myPackage.ClassA;
import secondPackage.ClassC;
public class ClassX
{
        public static void main(String args[])
        {
                ClassA objA = new ClassA();
                ClassC objC = new ClassC();
                objA.display();
                objC.display();
        }
}
```

Output

[raj@mundroo] package % java ClassX

```
Hello, I am ClassA
Hello, I am ClassC
```

Protection and Packages

- All classes (or interfaces) accessible to all others in the same package.
- Class declared public in one package is accessible within another. Non-public class is not
- Members of a class are accessible from a difference class, as long as they are not *private*
- *protected* members of a class in a package are accessible to subclasses in a different class

Visibility - Revisited

- *Public* keyword applied to a class, makes it available/visible everywhere. Applied to a method or variable, completely visible.
- *Private* fields or methods for a class only visible within that class. Private members are *not* visible within subclasses, and are *not* inherited.
- *Protected* members of a class are visible within the class, subclasses and *also* within all classes that are in the same package as that class.

Visibility Modifiers

Accessible to:	public	protected	Package (default)	private
Same Class	Yes	Yes	Yes	Yes
Class in package	Yes	Yes	Yes	No
Subclass in different package	Yes	Yes	No	No
Non-subclass different package	Yes	No	No	No

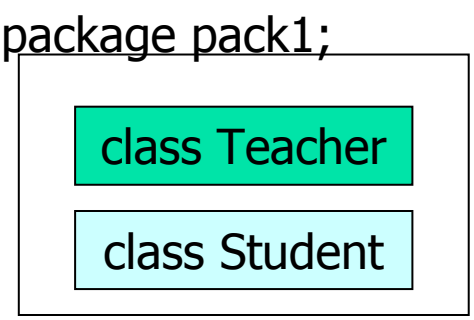
Adding a Class to a Package

- Consider an existing package that contains a class called "Teacher":

```
package pack1;
public class Teacher
{
  // class body
}
```

- This class is stored in "Teacher.java" file within a directory called "pack1".
- How do we a new public class called "Student" to this package.

Adding a Class to a Package



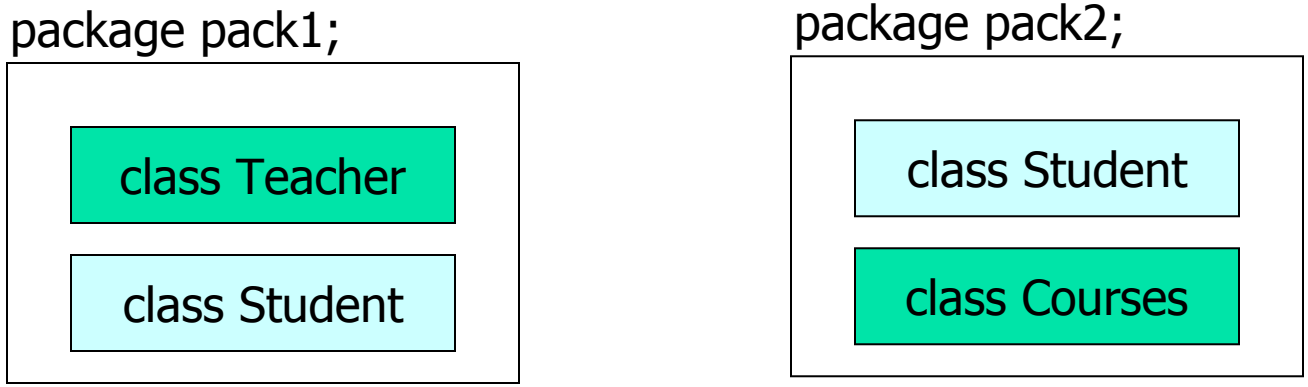
- Define the public class "Student" and place the package statement before the class definition as follows:

```
package pack1;
public class Student
{
  // class body
}
```

- Store this in "Student.java" file under the directory "pack1".
- When the "Student.java" file is compiled, the class file will be created and stored in the directory "pack1". Now, the package "pack1" will contain both the classes "Teacher" and "Student".

Packages and Name Clashing

- When packages are developed by different organizations, it is possible that multiple packages will have classes with the same name, leading to name classing.



- We can import and use these packages like:
  - import pack1.*;
  - import pack2.*;
  - Student student1; // Generates compilation error

Handling Name Clashing

- In Java, name classing is resolved by accessing classes with the same name in multiple packages by their fully qualified name.
- Example:

```
import pack1.*;
import pack2.*;
pack1.Student student1;
pack2.Student student2;
Teacher teacher1;
Courses course1;
```

Extending a Class from Package

- A new class called "Professor" can be created by extending the "Teacher" class defined the package "pack1" as follows:

```
import pack1.Teacher;
public class Professor extends Teacher
{
  // body of Professor class
  // It is able to inherit public and protected members,
  // but not private or default members of Teacher class.
}
```

Summary

- Packages allow grouping of related classes into a single united.
- Packages are organised in hierarchical structure.
- Packages handle name classing issues.
- Packages can be accessed or inherited without actual copy of code to each program.