PACKAGES

Mr. S. G. Lakhdive
Department of Mathematics,
Savitibai Phule Pune University, Pune

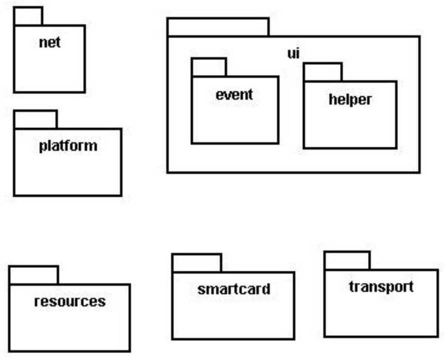
Packages

- 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 packages

- **package**: A collection of related classes.
  - Can also "contain" sub-packages.
  - *Sub-packages* can have similar names, but are not actually contained inside.
    - `java.awt` does not contain `java.awt.event`



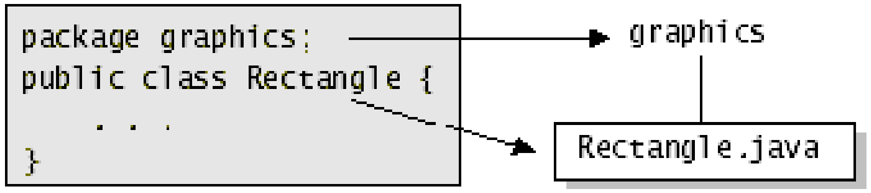
- Uses of Java packages:
  - group related classes together
  - as a *namespace* to avoid name collisions
  - provide a layer of access / protection
  - keep pieces of a project down to a manageable size

Packages and directories

- package ←→ directory (folder)
- class ←→ file

- A class named `D` in package `a.b.c` should reside in this file:

`a/b/c/D.class`



```
package graphics;
public class Rectangle {
    . . .
}
```

  - (relative to the root of your project)

- The "root" directory of the package hierarchy is determined by your *class path* or the directory from which `java` was run.

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 :

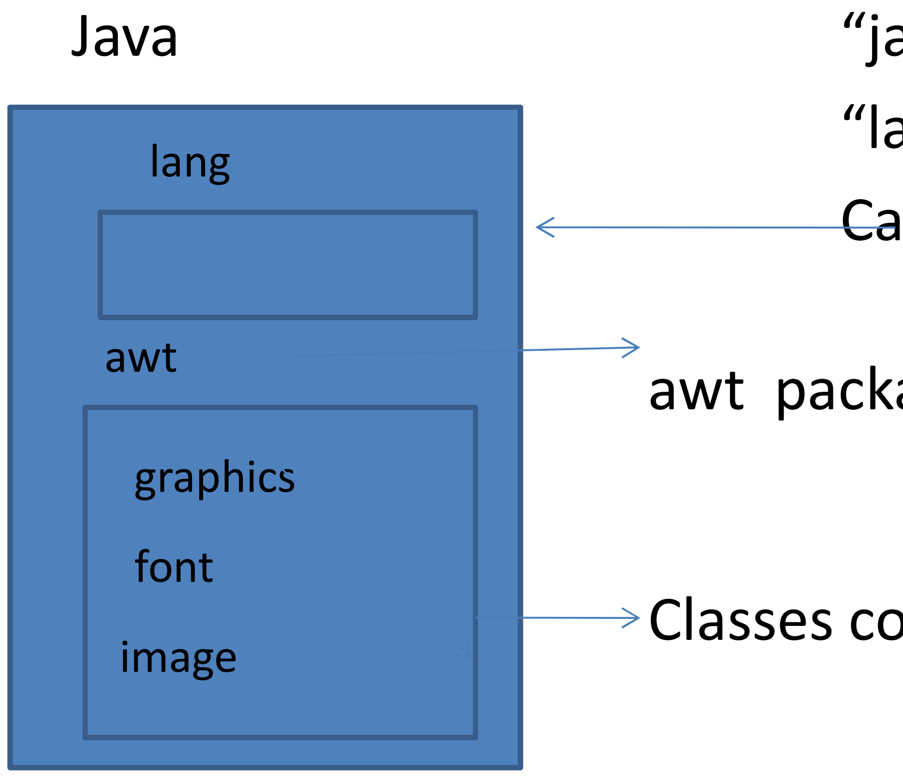
  Classes for networking .

- java.applet :

  Classes for creating and implementing app.

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).



“java” Package containing
“lang”, “awt”,.. packages;
Can also contain classes.

awt package containing classes.

Classes containing methods

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 {
}
```

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

- en 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”).

```
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 ClassY
- {
- public static void main(String args[])
- {
- ClassA objA = new ClassA();
- ClassC objC = new ClassC();
- objA.display();
- objC.display();
- }
- }

Output

- Hello, I am ClassA
- Hello, I am ClassC

Package access

- Java provides the following access modifiers:
  - `public` : Visible to all other classes.
  - `private` : Visible only to the current class (and any nested types).
  - `protected` : Visible to the current class, any of its subclasses, and any other types within the same package.
  - default (package): Visible to the current class and any other types within the same package.

- To give a member default scope, do not write a modifier:

```
package pacman.model;
public class Sprite {
    int points;          // visible to pacman.model.*
    String name;         // visible to pacman.model.*
```

Packages Advantages

Packages are groups of classes.

Often you put an application in its own package.

Also ‘libraries’ of useful classes.

Packages allow us to use other people’s code.

- Saves name clashes with other people.

User Defined Packages:

- Like built-in packages, user defined packages can be developed as follows:
- Steps to create a package:

  1. Create a directory i.e. package.

  2. Include the package command as the first line of code in your Java Source File.

  3. The Source file contains the classes, interfaces, etc you want to include in the package.

  4. **Everything in package must be public except variable declaration.**

  5. Compile to create the Java packages.

Example

```
package IMCA;
public class MyCollege
{
        public void disp()
        {
        System.out.println (“Yes..!..I m IMCA Student”;);
        }
}
```

Example

Save this file as name MyCollege.java in a IMCA directory and compile it as follow:

```
javac -d directory javafilename
```

Example: `javac –d . MyCollege.java`

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

We can also compiled java program from IMCA directory through command prompt as follow:

```
javac MyCollege.java
```

How to Access Package?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. Fully qualified name.

Example

```
import IMCA.*;
class Demo
{
        public static void main (String args[])
        {
        MyCollege obj=new MyCollege();
        obj.disp();
        }
}
```

Thanks you

new student();

- Assume Student is a Java Class.
- **anonymous object:** Anonymous objects are those objects in Java which are created without any reference variable.
- As a result, after creation, we have no way to access the anonymous object.
- In the above code, even though an object of Student class is created, it has no reference variable and cannot be used after creation, but it acquires memory.

Student stud = new Student();

- Assume Student is a Java Class.
- stud = null;
- **nulling reference:** When an object is referenced to null value, it is considered an unreferenced object as it holds only null value.
- Here, s is an unreferenced object that holds null value. It is considered garbage by JVM.

Student s1 = new Student();
Student s2 = new Student();
s1 = s2;

- Assume Student is a Java Class.
- **assigning a reference to another object:** When one object is assigned to another, first object is of no use and can be considered as garbage.
- After the last statement s1 = s2 is executed, the first object becomes unreferenced and can be considered for garbage collection.

Garbage Collector

- The garbage collector is a part of Java Virtual Machine(JVM).
- Garbage collector checks the heap memory, where all the objects are stored by JVM, looking for unreferenced objects that are no more needed and automatically destroys those objects.
- Garbage collector calls **finalize()** method for clean up activity before destroying the object.
- Java does garbage collection automatically; there is no need to do it explicitly, unlike other programming languages.