Packages

CSE 142, Summer 2002

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/02su/

Readings and References

- Reading
  - » Chapter 14.6, *An Introduction to Programming and Object Oriented Design using Java*, by Niño and Hosch
- Other References
  - » "Creating and Using Packages", Java tutorial
  - » http://java.sun.com/docs/books/tutorial/java/interpack/packages.html

Cohesion and Coupling

- Cohesion describes the degree to which the various parts of a class all relate to one another in a logical way - a “cohesive design” is a good thing
- Coupling describes the degree to which different classes are tied together through implementation details and assumptions - a “highly coupled design” is a bad thing
- Goals:
  - » Increase cohesion
  - » Reduce coupling

Cohesion

- Cohesion looks at classes on a high level
  - » do one thing well, rather than doing many things poorly
- Examples
  - » Dog methods - getMealSize(), eat(), toString()
  - » PetSet methods - speak(), dine()
  - » not rover.addMeToPetSet(7)
  - » not theBunch.doAll(3)
- Focus on conceptual task
- Why?
  - » Easier to understand the class function

Coupling

- Coupling looks at the ties between classes
  - » keep it simple and direct - on a “need to know” basis
- Examples
  - » Dog constructor

    `Dog(name,serve,weight)`

    not - `Dog(index,displayType,name,birthDate)`
  - » PetSet method

    `theBunch.add(rover)`

    not - `rover.addMeToPetSet(petNumber,theBunch)`
- Why?
  - » Easier to change your code without ripple effects

Class design

- Focus each class on a particular logical object
  - » control the state and behavior of the object using the methods of the class
- Focus each method on one conceptual task
  - » name the method to indicate the nature of the task
- Avoid passing control data into the methods
  - » deciding how to perform is the method’s job
- Avoid method explosion
  - » Keep number of methods to a manageable number

Structure of Source File

- Simple structure in order
  - » package definition

    Optional, if missing uses the "default" package.

    ```
    package hw7;
    ```
  - » package and/or class import statements

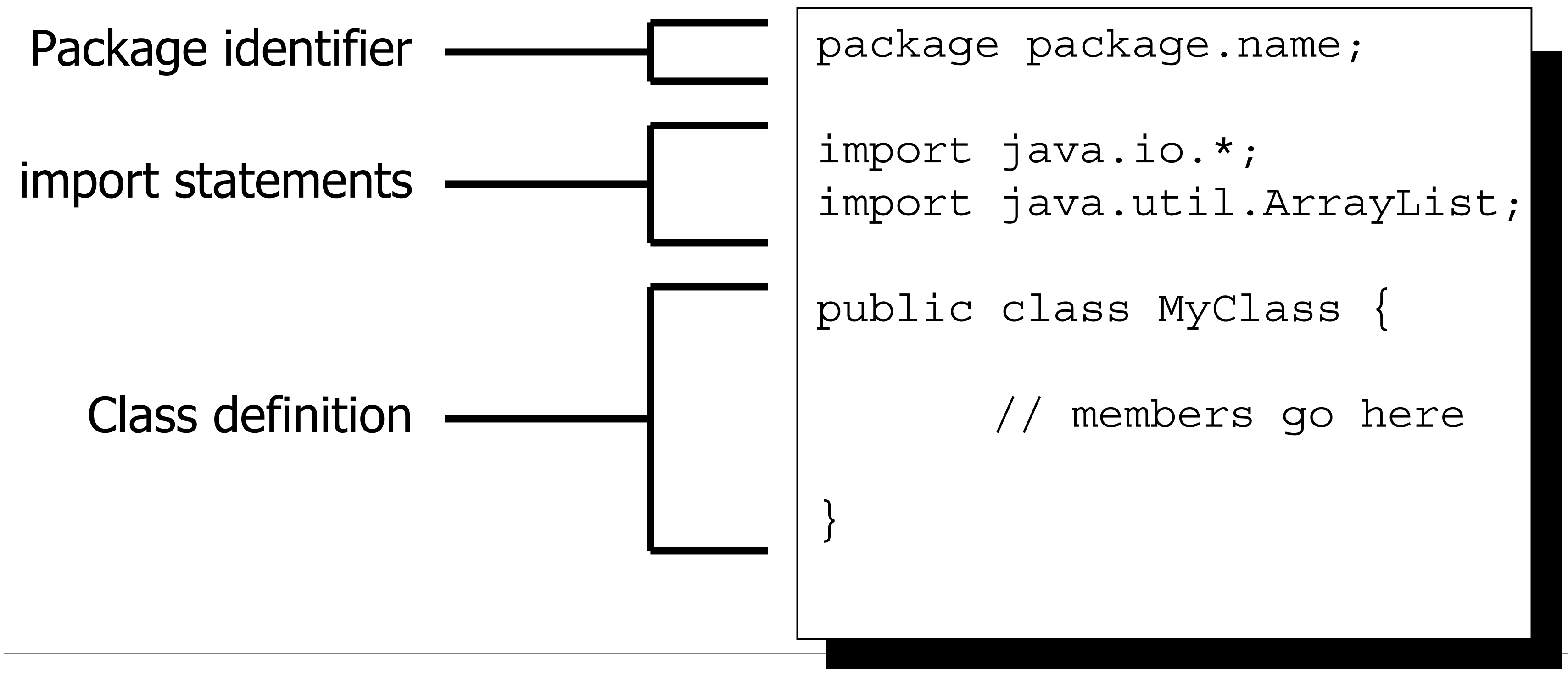
    Optional, use only as desired for simplicity

    ```
    import java.util.*;
    ```
  - » Class definition (multiple are allowed but messy)

    ```
    public class Dog {
      …
    }
    ```

Structure of Source File (Continued)

Three components to a Java source file, <u>in order</u>

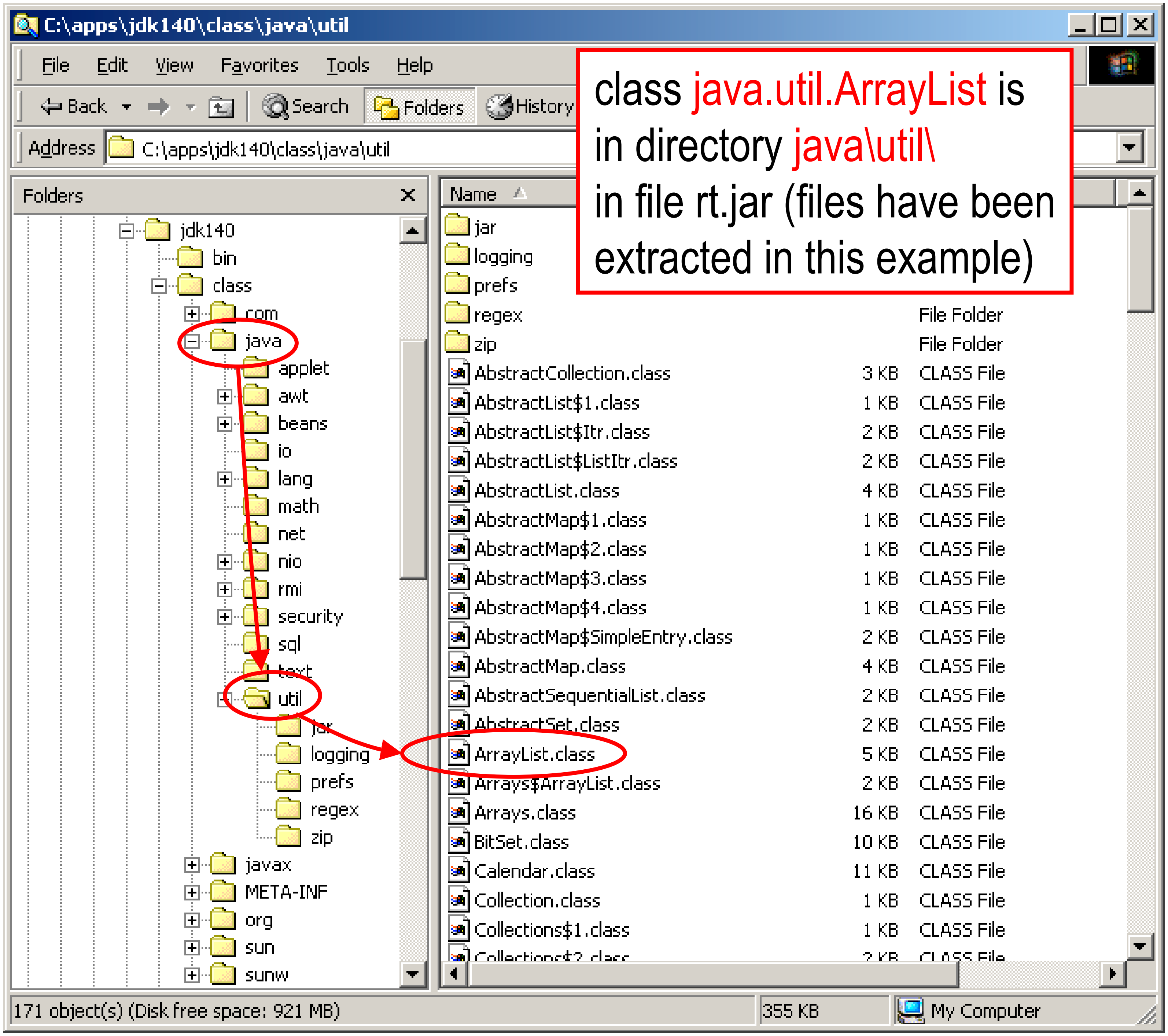


```
package package.name;

import java.io.*;
import java.util.ArrayList;

public class MyClass {

      // members go here

}
```

Packages

- Packages are a way to group related classes
  - » A key part of Java’s encapsulation mechanism
  - » A class is permanently associated with its package
- Period (.) separated name of the package mirrors directory structure where classes are stored
- “Default” package is the current directory
  - » Classes without a package identifier are considered to be in the default package
  - » That's why we can ignore package in simple programs

class java.util.ArrayList is in directory java\util\ in file rt.jar (files have been extracted in this example)



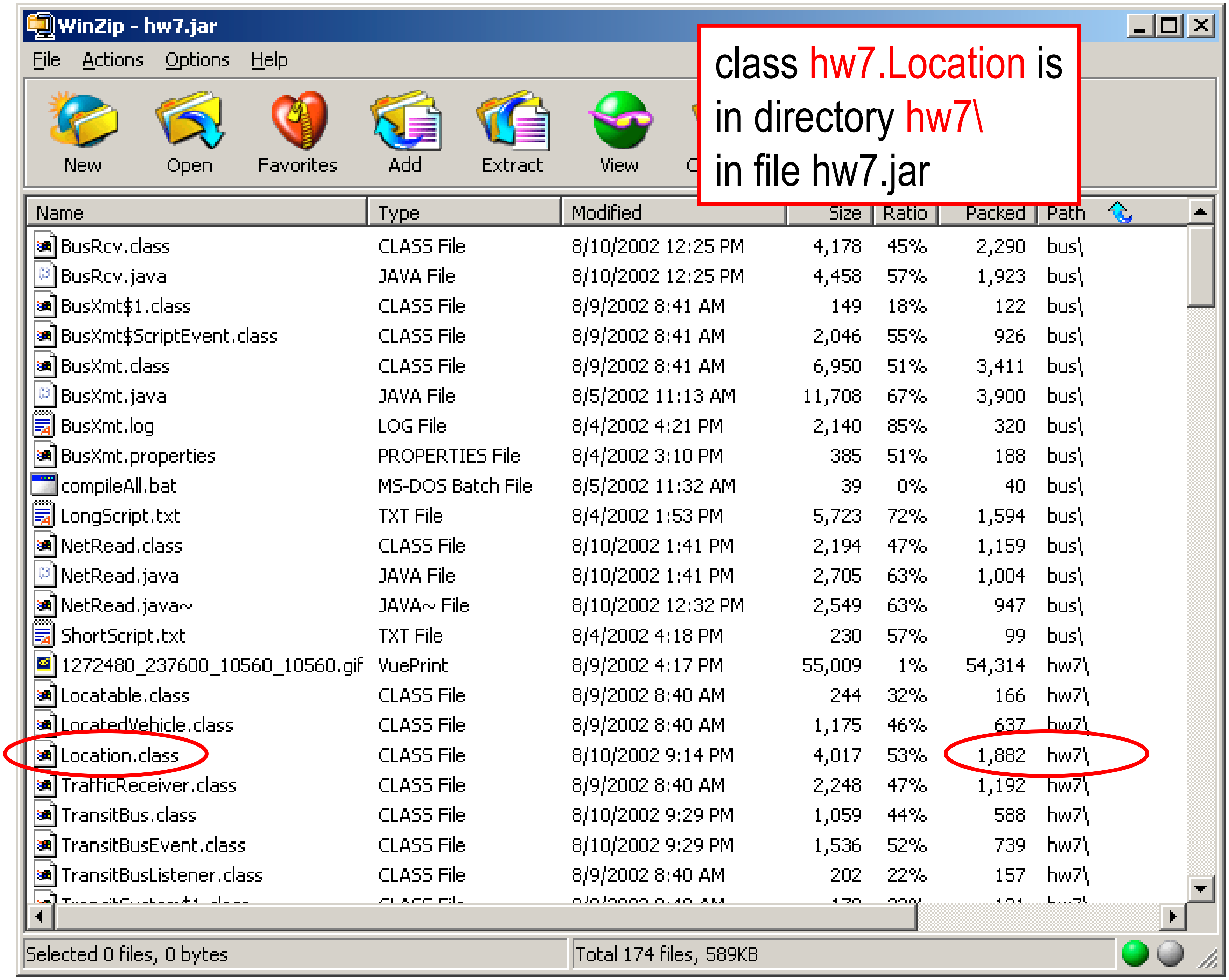
C:\apps\jdk140\class\java\util

```
jdk140
  bin
  class
    com
    java
      applet
      awt
      beans
      io
      lang
      math
      net
      nio
      rmi
      security
      sql
      text
      util
        jar
        logging
        prefs
        regex
        zip
    javax
    META-INF
    org
    sun
    sunw
```

Name	Size	Type
jar		
logging		
prefs		
regex		File Folder
zip		File Folder
AbstractCollection.class	3 KB	CLASS File
AbstractList$1.class	1 KB	CLASS File
AbstractList$Itr.class	2 KB	CLASS File
AbstractList$ListItr.class	2 KB	CLASS File
AbstractList.class	4 KB	CLASS File
AbstractMap$1.class	1 KB	CLASS File
AbstractMap$2.class	1 KB	CLASS File
AbstractMap$3.class	1 KB	CLASS File
AbstractMap$4.class	1 KB	CLASS File
AbstractMap$SimpleEntry.class	2 KB	CLASS File
AbstractMap.class	4 KB	CLASS File
AbstractSequentialList.class	2 KB	CLASS File
AbstractSet.class	2 KB	CLASS File
ArrayList.class	5 KB	CLASS File
Arrays$ArrayList.class	2 KB	CLASS File
Arrays.class	16 KB	CLASS File
BitSet.class	10 KB	CLASS File
Calendar.class	11 KB	CLASS File
Collection.class	1 KB	CLASS File
Collections$1.class	1 KB	CLASS File

171 object(s) (Disk free space: 921 MB) | 355 KB | My Computer

class hw7.Location is
in directory hw7\
in file hw7.jar



Name	Type	Modified	Size	Ratio	Packed	Path
BusRcv.class	CLASS File	8/10/2002 12:25 PM	4,178	45%	2,290	bus\
BusRcv.java	JAVA File	8/10/2002 12:25 PM	4,458	57%	1,923	bus\
BusXmt$1.class	CLASS File	8/9/2002 8:41 AM	149	18%	122	bus\
BusXmt$ScriptEvent.class	CLASS File	8/9/2002 8:41 AM	2,046	55%	926	bus\
BusXmt.class	CLASS File	8/9/2002 8:41 AM	6,950	51%	3,411	bus\
BusXmt.java	JAVA File	8/5/2002 11:13 AM	11,708	67%	3,900	bus\
BusXmt.log	LOG File	8/4/2002 4:21 PM	2,140	85%	320	bus\
BusXmt.properties	PROPERTIES File	8/4/2002 3:10 PM	385	51%	188	bus\
compileAll.bat	MS-DOS Batch File	8/5/2002 11:32 AM	39	0%	40	bus\
LongScript.txt	TXT File	8/4/2002 1:53 PM	5,723	72%	1,594	bus\
NetRead.class	CLASS File	8/10/2002 1:41 PM	2,194	47%	1,159	bus\
NetRead.java	JAVA File	8/10/2002 1:41 PM	2,705	63%	1,004	bus\
NetRead.java~	JAVA~ File	8/10/2002 12:32 PM	2,549	63%	947	bus\
ShortScript.txt	TXT File	8/4/2002 4:18 PM	230	57%	99	bus\
1272480_237600_10560_10560.gif	VuePrint	8/9/2002 4:17 PM	55,009	1%	54,314	hw7\
Locatable.class	CLASS File	8/9/2002 8:40 AM	244	32%	166	hw7\
LocatedVehicle.class	CLASS File	8/9/2002 8:40 AM	1,175	46%	637	hw7\
Location.class	CLASS File	8/10/2002 9:14 PM	4,017	53%	1,882	hw7\
TrafficReceiver.class	CLASS File	8/9/2002 8:40 AM	2,248	47%	1,192	hw7\
TransitBus.class	CLASS File	8/10/2002 9:29 PM	1,059	44%	588	hw7\
TransitBusEvent.class	CLASS File	8/10/2002 9:29 PM	1,536	52%	739	hw7\
TransitBusListener.class	CLASS File	8/9/2002 8:40 AM	202	22%	157	hw7\

Selected 0 files, 0 bytes

Total 174 files, 589KB

import statement

- A class’ full name includes its package.
  - » for example, java.util.ArrayList or java.lang.String
- Often it is more convenient to use the class name without the package, e.g., ArrayList, String
- The **`import`** statement tells the compiler where to find class definitions that don't have a complete package name and aren't in the current package
  - » Classes can be imported individually, or all classes in a package can be imported
  - » java.lang.* is imported automatically by the compiler
  - » is <u>not</u> like #include in C/C++

`import` example

```
import java.util.*
public class Importer {
    public Importer() {
        names = new ArrayList( );
        names.add("Billy");
        names.add("Susan");
    }
```

and

or

```
public class Importer {
    public Importer() {
        names = new java.util.ArrayList( );
        names.add("Billy");
        names.add("Susan");
    }
```

Compiler Error

no import statement

```
public class Importer {
    public Importer() {
        names = new ArrayList( );
        names.add("Billy");
        names.add("Susan");
    }
```

no package name

```
G:\cse142\dev\ex142\lect20\Importer.java:10: cannot resolve symbol
symbol  : class ArrayList
location: class Importer
        names = new ArrayList( );
                    ^
1 error
```

Body of the class file

- If more than one class is defined in the source file, only one of them can be declared public

```
public class PetSet {
   …
}
class Helper {
   …
}
```

- source file must have same name as name of public class

  `public class PetSet {…}` must be in PetSet.java

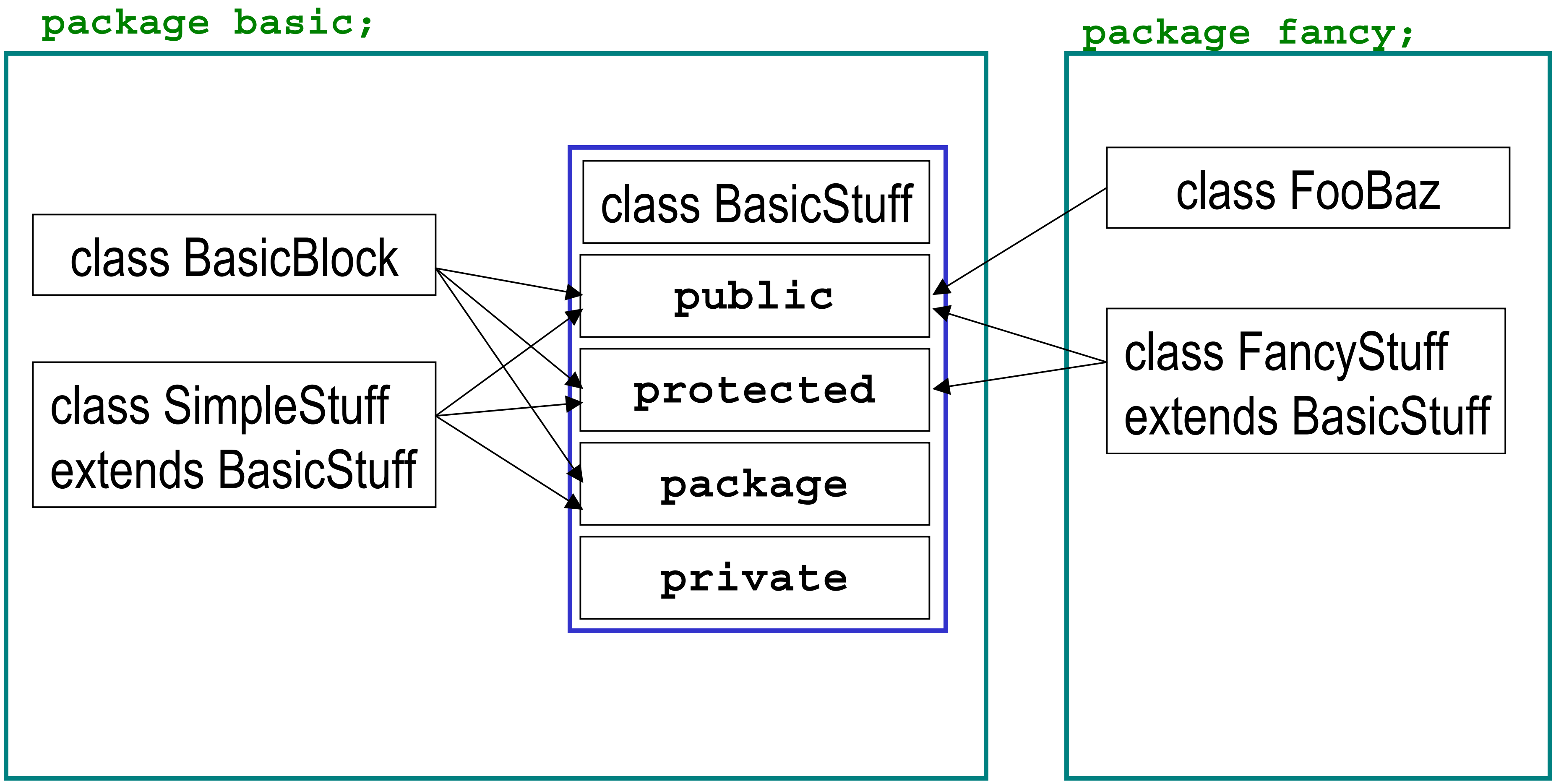
Encapsulation

- Encapsulation is the way we, as software architects, control the way users of our classes interact with those classes
- Java has features built in to the language that allow us to hide implementation details
  - » Public aspects of the implementation are a commitment for life (method names, variables)
  - » Hidden parts of an implementation can be changed without affecting users

Java syntax for encapsulation

- There are four levels of access to class members
  - » public: member visible to any class anywhere
  - » protected: member visible to classes in same package, plus subclasses
  - » package: member visible to classes in same package
  - » private: member visible only within the class
- Keywords match the names above, except package access, which uses no keyword

Visibility across package boundaries



Access control keywords

```
package uw.java.course;

public class Test {
   public Test() {…}
   public void publicMethod() {…}
   protected int protectedInt;
   String packageString;
   private double privateDouble;
}
```

Guidelines

- Use public for most constructors and those methods that you want others to know about
- Use private for internal "helper" methods
- Use private for instance variables
  - » Only in rarest cases should variables be made public because you may well want to change their implementation
- Use protected and package (default) only in very specific cases where needed