

CSE 8B—Intro to CS: Java

Winter, 2006

March 16 (Day 20)
Packages
CLASSPATH
Javadoc

Instructor: Neil Rhodes

Packages

Java classes are organized into related groups called *packages*

- `java.io`
 - Contains classes and interfaces like `FileInputStream` and `FileOutputStream`
- `java.lang`
 - Core classes like `String`, `Thread`, `Exception`
- `java.util`
 - Utility classes like `Scanner`
- `javax.swing`

2

To use a class from a package

Either import the class from the package

- `import packagename.Classname`
- Then, just use `Classname` in the rest of your code
- Example:
 - `import java.util.Scanner;`

```
Scanner in = new Scanner(...);
```

Or, specify the package name preceding every use of the class name:

- Example:
 - `java.util.Scanner in = new java.util.Scanner(...)`

Wildcard imports

- `import packagename.*`
- Imports all classes in the specified package
- Doesn't import subpackages of the given packagename

3

Naming packages

Package names should be unique

- Sun reserves package name starting with
 - `java`
 - `javax`
 - `sun`
- Anyone else names their packages using a domain name (in reverse order)
 - `edu.ucsd.cse.mypackage`
 - This way, Sun delegates unique naming.

4

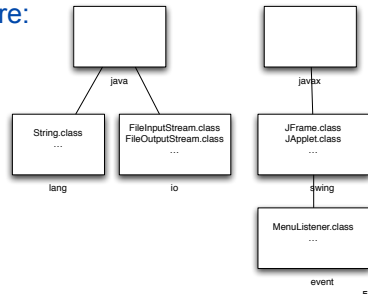
Specifying a package

In your .java file, you use the package command:

```
package packagename;  
import ...  
public class ...
```

You can have many classes in the same package

The files also must be organized in a directory structure that parallels the package structure:



5

Your directory hierarchy

If your Foo and Bar classes will be in a package:

- com.neilrhodes.mypackage

you'd need a directory hierarchy:

- com
- neilrhodes
- mypackage

```
package com.neilrhodes.mypackage;  
  
public class Bar {  
}
```

```
package com.neilrhodes.mypackage;  
  
public class Foo {  
    Bar aBar = new Bar();  
    public static void main(String[] args)  
    {  
        System.out.println("Foo");  
    }  
}
```

6

Compiling

Bar.java will compile

Foo.java won't compile

```
Foo.java:4: cannot find symbol  
symbol : class Bar  
location: class com.neilrhodes.mypackage.Foo  
    Bar aBar;
```

- Problem: the compiler is looking for a file: com/neilrhodes/mypackage/Bar.class
- Solution 1: change directories up 3 levels (into directory containing com) and compile from there:
 - `javac com/neilrhodes/mypackage/Foo.java`
- Solution 2: stay in the mypackage directory and tell the java compiler to start looking for classes three levels above:
 - `javac -classpath ../../.. Foo.java`

7

Running

Try to run Foo from the mypackage directory:

- java Foo
- Fails with:

```
Exception in thread "main" java.lang.NoClassDefFoundError: Foo  
(wrong name: com/neilrhodes/mypackage/Foo)
```

...

Try to run using package name:

- java com.neilrhodes.mypackage.Foo
- Fails with:

```
Exception in thread "main" java.lang.NoClassDefFoundError: com/  
neilrhodes/mypackage/Foo
```

Solution: run from the directory containing com:

- java com.neilrhodes.mypackage.Foo

Or, run from mypackage directory specifying class lookup:

- java -classpath ../../.. com.neilrhodes.mypackage.Foo

8

Using CLASSPATH

Rather than specifying `-classpath` to `java` and `javac`, specify it one time in *environment variable*:

- `setenv CLASSPATH ./users/neilrhod/documents/javacode`

9

JAR files

A Jar file is a container (a zip file, actually), that contains a whole directory hierarchy with `.class` files (and possibly other files like images and sounds)

- Encapsulates entire directory structure
- For example, all junit classes are in `junit.jar`:

```
junit/  
junit/awtui/  
junit/extensions/  
junit/framework/  
junit/runner/  
junit/swingui/  
junit/swingui/icons/  
junit/textui/  
junit3.8.1/  
junit/awtui/AboutDialog$1.class  
junit/awtui/AboutDialog$2.class  
junit/awtui/AboutDialog.class  
junit/awtui/Logo.class  
junit/awtui/ProgressBar.class  
junit/awtui/TestRunner$1.class  
junit/awtui/TestRunner$10.class  
...
```

- Jar files are manipulated with `jar` command-line tool

10

Default package

If your `.java` file doesn't specify a package, it'll be in the *default package*

- That's the same package as any other files with no package specified
- Classes in the same package can access
 - `public` members
 - `protected` members
 - `package` members
 - (no `public/protected/private` specified)

11

Javadoc

Store your documentation in the `.java` file

`javadoc` tool reads the documentation from the `.java` file and produces **HTML** documentation

- Documentation is block comments (with two asterisks in opening comment):

```
/**  
 *  
 * ...  
 */
```

- Documentation is *tagged*:

- `@author name`
- `@version text`
- `@param parameter-name description`
- `@return description`
- `@throws full-exception-classname description`
- `@see package-name`
- `@see <URL>`

12

Java file

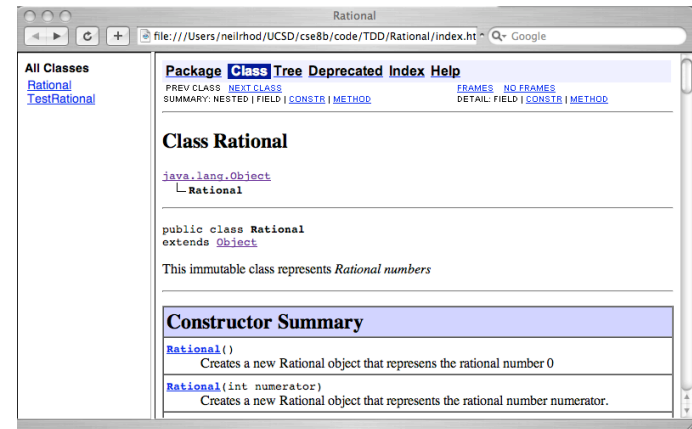
```
/**
 * This immutable class represents <i>Rational
 numbers</i>
 *
 * @author Neil Rhodes
 * @version 1.0
 */
public class Rational {
    private int num;
    private int denom;

    /**
     * Creates a new Rational object that represents
     the rational number 0
     */
    public Rational() {
        num = 0;
        denom = 1;
    }

    /**
     * Creates a new Rational object that
     represents the rational number
     * numerator.
     * @param numerator The numerator of the
     rational number
     */
    public Rational(int numerator) {
        ...
    }
}
```

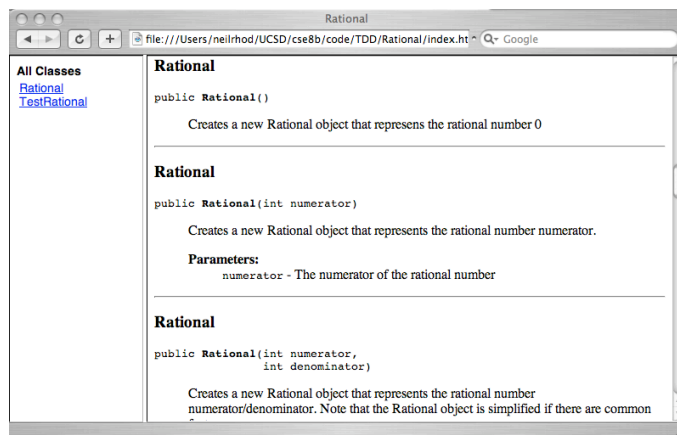
13

Output



14

More output



15

To run

Use

- `javadoc file1.java file2.java`

Problem:

- No links to java classes (like Object).

Solution, use -link option

- `javadoc -link http://java.sun.com/j2se/1.5.0/docs/api/ *.java`

16

Refactoring

Short list of common refactorings

- Rename method
- Replace Magic Number with Symbolic Constant
- Decompose Conditional
- Consolidate Duplicate Conditional Fragments
- Replace Conditional with Polymorphism

This list from *Refactoring* by Fowler. Small sampling of many provided

17

Rename method

Situation:

- The name of a method does not reveal its purpose

Solution:

- Change the name of the method.

18

Replace magic number with symbolic constant

Situation:

- You have a literal number with a particular meaning

Solution:

- Create a constant, name it after the meaning, and replace the number with it.

19

Decompose Conditional

Situation:

- You have a complicated conditional (if-then-else) statement.

Solution:

- Extract methods from the condition, then part, and else parts.

20

Consolidate duplicate conditional fragments

Situation:

- The same fragment of code is in all branches of a conditional expression

Solution:

- Move it outside of the expression.

21

Replace Conditional with Polymorphism

Situation:

- You have a conditional that chooses different behavior depending on the type of the object

Solution

- Move each leg of the conditional to an overriding method in a subclass. Make the original method abstract

22

Good books to read

Code Complete by Steve McConnell

- How to:
 - comment
 - name functions and variables
 - debug
 - design
- Great deal: 960 pages for only \$32!
- “The bible of software construction”
- I can’t recommend it highly enough
 - I should have required it for this course



23

Good books to read

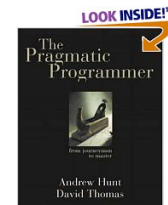
The Pragmatic Programmer: from Journeyman to Master

- by Andrew Hunt and David Thomas

Will be more useful as you gain more experience

Also a must-have

- maybe not immediately, but soon



24

Good books to read

Refactoring: Improving the design of existing code

- By Martin Fowler

Step-by-step instructions on making code better

