

CSSE 220 Day 3

Unit Tests and Object References
Implementing Classes in Java, using
Documented Stubs, Test-First Programming

Check out *UnitTesting* and
WordGames from SVN

What Questions Do You Have?

Syllabus

Reading assignments

Homework

Things discussed in class

Anything else

Javadocs: Key Points

- Don't try to memorize the Java libraries
 - Nearly 9000 classes and packages!
 - You'll use a few dozen of them during this course
- Get in the habit of writing the javadocs **before** implementing the methods
 - It will help you **think before doing**, a vital software development skill
 - This is called programming with ***documented stubs***
 - I'll try to model this. If I don't, call me on it!

Test-driven Development,
unit testing and JUnit

WRITING CODE TO TEST YOUR CODE

Unit Testing

- Using code that you write to test other code
 - Focused on testing individual pieces of code (units) in isolation
 - Individual methods
 - Individual classes
- Why would software engineers do unit testing?

Unit Testing With JUnit

- JUnit is a unit testing *framework*
 - A *framework* is a collection of classes to be used in another program.
 - Does much of the work for us!
- JUnit was written by
 - Erich Gamma
 - Kent Beck
- Open-source software
- Now used by **millions** of Java developers

JUnit Example

- `MoveTester` in Big Java shows how to write tests in plain Java
- Look at `JUnitMoveTester` in today's repository
 - Shows the same test in JUnit
 - Let's look at the comments and code together...

Interesting T

Important Slide: Use this as a reference!

- Test “boundary conditions”
 - Intersection points: $-40^{\circ}\text{C} == -40^{\circ}\text{F}$
 - Zero values: $0^{\circ}\text{C} == 32^{\circ}\text{F}$
 - Empty strings
- Test known values: $100^{\circ}\text{C} == 212^{\circ}\text{F}$
 - But not too many
- Tests things that might go wrong
 - Unexpected user input: “zero” when 0 is expected
- Vary things that are “important” to the code
 - String length if method depends on it
 - String case if method manipulates that

Unit test *shout*, *whisper*, and *holleWorld* using “interesting” test cases

EXERCISE

Differences between primitive types and object types in Java

OBJECT REFERENCES

What Do Variables Really Store?

- Variables of **primitive type** store *values*
- Variables of **class type** store *references*

– A reference is like a pointer in C, except

- Java keeps us from screwing up

- No **&** and ***** to worry about
(and the people say, “Amen”)

1. `int x = 10;`

2. `int y = 20;`

3. `Rectangle box = new Rectangle(x, y, 5, 5);`

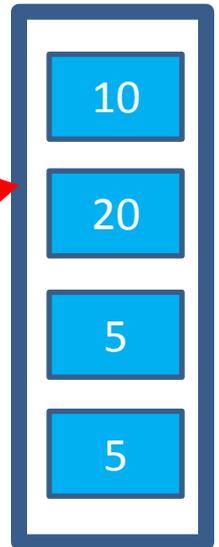
box

x

y

10

20



Assignment Copies Values

- Actual value for number types
- Reference value for object types

– The actual **object is not copied**

– The **reference value** (“the pointer”) is copied

1. `int x = 10;`



2. `int y = x;`

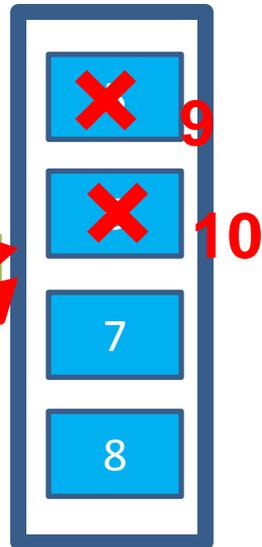
3. `y = 20;`



`box`



`box2`



4. `Rectangle box = new Rectangle(5, 6, 7, 8);`

5. `Rectangle box2 = box;`

6. `box2.translate(4, 4);`

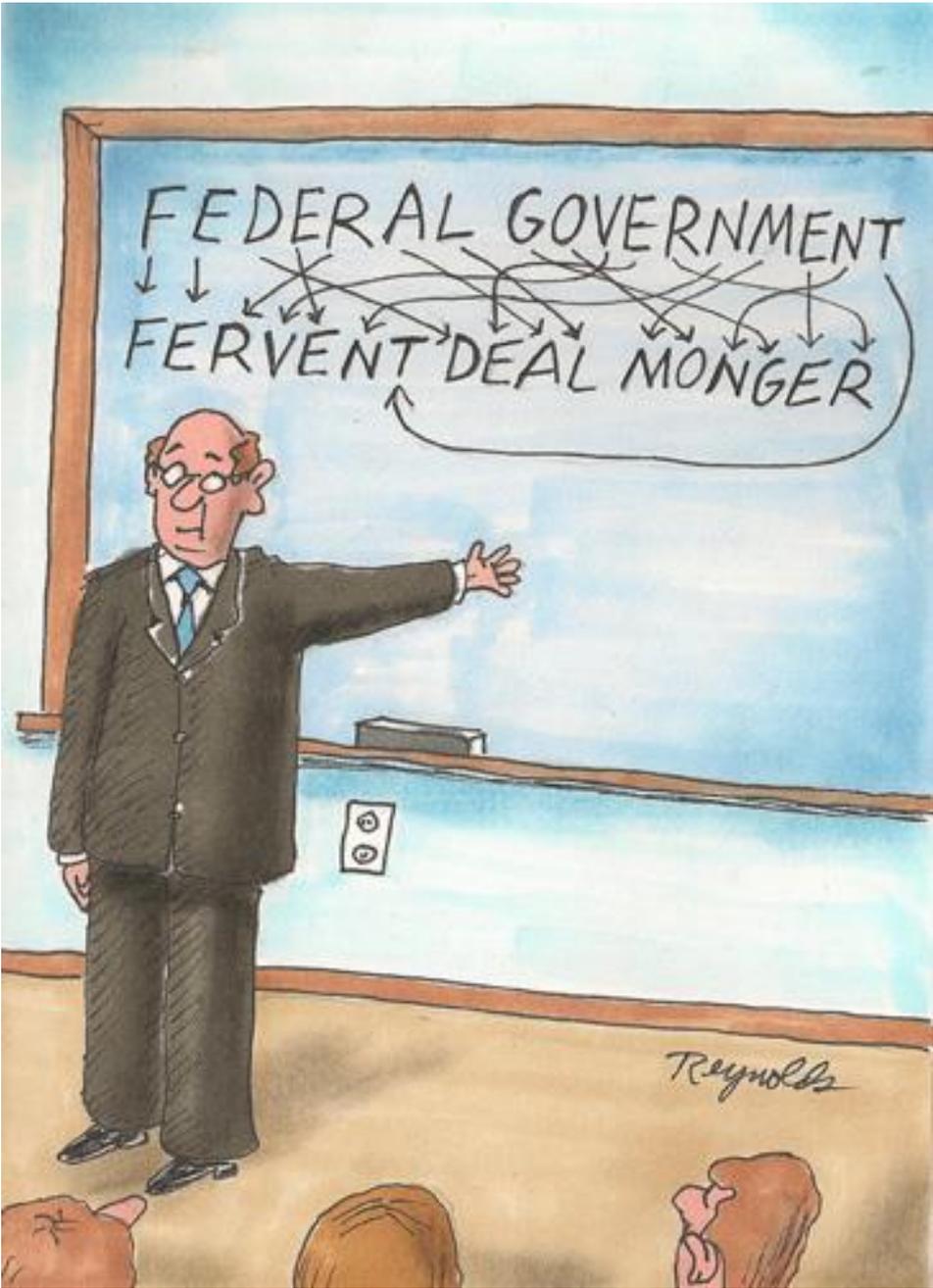
ENCAPSULATION

Separating implementation details from how an object is used

Encapsulation in Object-Oriented Software

- *Encapsulation*—separating implementation details from how an object is used
 - Client code sees a *black box* with a known *interface*

	Functions	Objects
Black box exposes	Function signature	Constructor and method signatures
Encapsulated inside the box	Operation implementation	<u>Data storage</u> and <u>operation implementation</u>



How To: Implement a Class

- 1. Create the** (initially empty) **class**
 - File ⇒ New ⇒ Class
2. Write ***documented stubs*** for the public interface of the class
- 3. Implement the class:**
 - Determine and implement instance fields
 - Implement constructors and methods, adding private methods and additional instance fields as needed
- 4. Test the class**

3. Test and implement each constructor and method

- Write the test cases BEFORE implementing the constructor/method

WordGames Shouter

LIVE CODING

Censor

- **Censor**: given a string *inputString*, produces a new string by replacing each occurrence of **charToCensor** with a “*” (an asterisk).
- How do you deal with **charToCensor** ?
 - Can it be a parameter of *transform*?
 - No, that violates the *specification*
 - Can it be a local variable of *transform*?
 - No, it needs to live for the entire lifetime of the Censor.
 - What’s left?
 - Answer: It is a *field* ! (What is a sensible name for the field?)
- How do you initialize the field for **charToCensor** ?
 - Answer: by using Censor’s constructors!

WordGames Censor

LIVE CODING

WRAP UP QUIZ AND TURN IT IN

Continue with homework if time permits