CSSE 220 Day 2

API Documentation, Unit Testing And Classes

Your questions about …

- The syllabus
- Java
- etc.

- Could everyone checkout and commit the HW1 project?

Announcements

- Please consider making your picture on ANGEL visible to students in your courses.
  - Home→ Preferences (wrench icon)→ Personal info

More announcements

- Cell Phones
  - please set ringers to silent or quiet.
    - Minimize class disruptions.
    - But sometimes there are emergencies.
- Personal needs
  - If you need to leave class for a drink of water, a trip to the bathroom, or anything like that, you need not ask me. Just try to minimize disruptions.
- Please be here and have your computer up and running by the beginning of class time as best you can.

Bonus points for reporting bugs

- In the textbook
- In any of our materials.
- Use the Bug Report Forum on ANGEL
- More details in the Syllabus

- Check out Piazza

Some major emphases of 220

- *Reinforce from 120:*
  - Procedural programming (functions, conditionals, loops, etc)
  - Using objects
- *Object-Oriented Design*
  - Major emphasis on interfaces
  - GUI programming using Java Swing
  - UML class diagrams
- *Software Engineering concepts*
- *Recursion*
- *Program Efficiency Analysis and big-O notation*
- *Simple sorting and searching algorithms*
  - as examples for the above
- *Data Structures*
  - Abstract data types
  - Specifying and using some standard data structures
  - Implementing simple data structures (lists)

What will I spend my time doing?

- Small programming assignments in class
- Larger programming problems, mostly outside of class
  - Explore the JDK documentation to find the classes and methods that you need
  - Lots of testing and debugging!
  - Reviewing other students' code
- Reading (a lot to read at the beginning; less later)
  - Thinking about exercises in the textbooks
  - Some written exercises, mostly from the textbook
- Discussing the material with other students

Java Documentation

- API Documentation, Docs in Eclipse, Writing your own Docs

Java API Documentation

- What's an API?
  - Application Programming Interface
- The Java API on-line
  - Google for: java api documentation 7
  - Or go to: http://download.oracle.com/javase/7/docs/api/
  - Also hopefully on your computer at C:\Program Files\Java\jdk1.7.0_9\docs\api\index.html

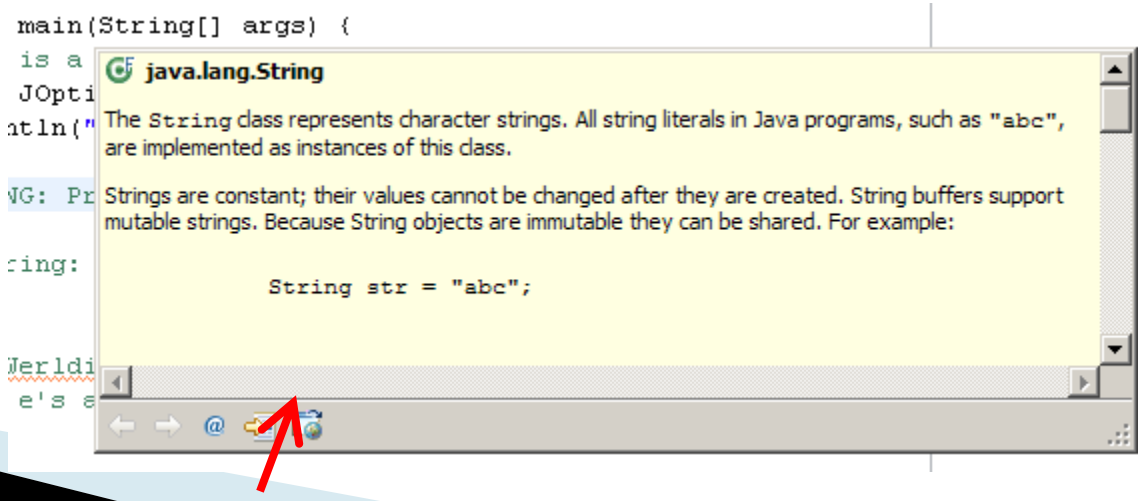
You need the 6 to get the current version of Java

Note: Your version may be something other than 7.0_9. We recommend that you bookmark this page in your browser, so you can refer to it quickly, with or without an internet connection.

Q1

Java Documentation in Eclipse

- Setting up Java API documentation in Eclipse
  - Should be done already,
  - If the next steps don't work for you, instructions are in today's homework
- Using the API documentation in Eclipse
  - Hover text
  - Open external documentation (Shift-F2)



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!

Q2

Writing Code to Test Your Code

- Test-driven Development, unit testing and JUnit

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?

Q3

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

Q4

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 Tests

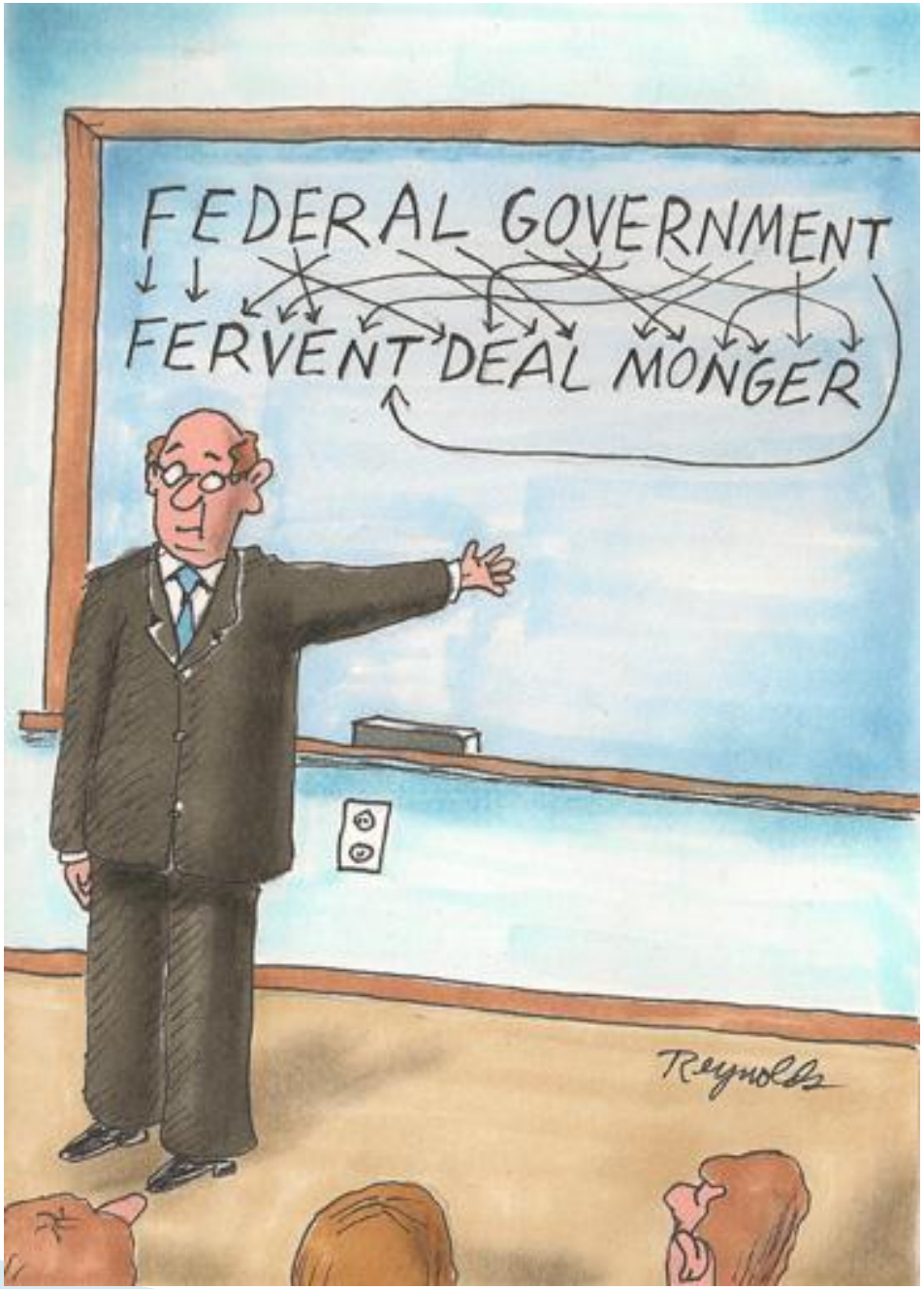
Important Slide: Use this as a reference!

- Test “boundary conditions”
  - Intersection points: -40℃ == -40℉
  - Zero values: 0℃ == 32℉
  - Empty strings
- Test known values: 100℃ == 212℉
  - 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

Exercise

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

Interlude

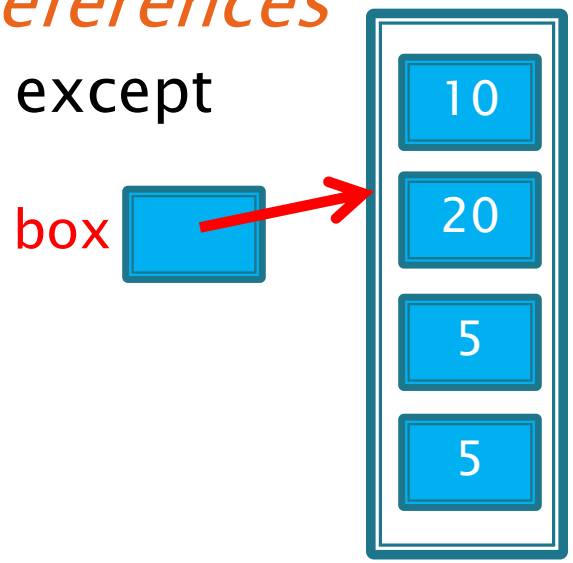


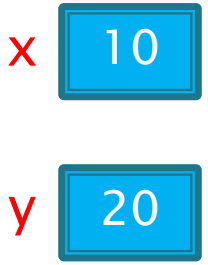
Object References

- Differences between primitive types and object types in Java

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")
- Consider:





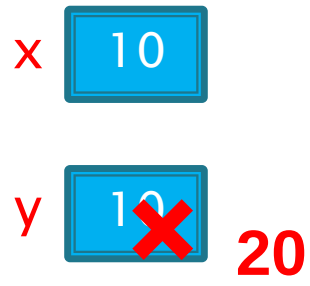

```
1. int x = 10;
2. int y = 20;
3. Rectangle box = new Rectangle(x, y, 5, 5);
```

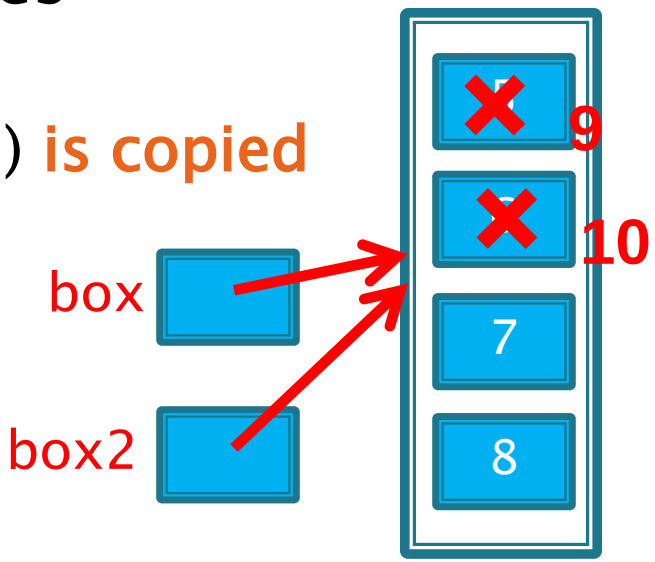
Q5

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**
- Consider:

```
1. int x = 10;
2. int y = x;
3. y = 20;
```






```
4. Rectangle box = new Rectangle(5, 6, 7, 8);
5. Rectangle box2 = box;
6. box2.translate(4, 4);
```

Q6 - Q7

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*
  - Implementation can change without changing client

	Functions	Objects
Black box exposes	Function signature	Constructor and method signatures
Encapsulated inside the box	Operation implementation	**Data storage** and **operation implementation**

Q8 – Q9

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

Live Coding

- WordGames Shouter

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!

Live Coding

- WordGames Censor

Exercise

- Finish quiz and pass it in
  Continue working on homework

Q10 – 11