Section 10!

We're almost done!

Final Review **Course Evals**

CSE 331, 12/7/11 *TA: Krysta Yousoufian*

Announcements

- Final exam: Wed. 12/14, 2:30-4:30, MGH 241
- *Trying* to get A5 back before finals

Course Review

- Disclaimer 1:I don't know any more than you do about the exam!
- Disclaimer 2: I can't promise that this review is completely comprehensive
  - Tried to be thorough, but I'm human
- Unless we say otherwise **(update: confirmed):**
  - Section material is fair game (MVC, etc.)
  - Except enums, switch statements, reflection
  - (The rest ties in with lecture)

Javadoc

- What it's used for
- General tags: @returns, @throws, @param
- CSE331 tags: @requires @modifies, @effects

JUnit Testing

- What to test
  - Each case that tests a significantly different input condition
  - Common cases (e.g. lists: one/two(?)/multiple elements, even/odd size)
  - Special cases (duplicates, sorted list, reverse-sorted list, negative numbers)
  - Edge cases (empty list)
  - Boundary conditions (if different behavior for integers <0 and >0, try -1, 0, 1)
  - Invalid cases (if behavior is documented – does it throw expected exception?)

- http://www.cs.washington.edu/education/courses/cse331/11au/sections/CSE331_section3.pdf
- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect03-testing.pdf

JUnit Testing

- How to write tests
  - Test every method (except constructors – tested implicitly)
  - Test each method in isolation if possible
  - Keep tests short – test only one thing at a time
  - Don't use println()
  - Use asserts
    - Choose the correct assert for the occasion
    - Always assert something
    - Good: assertEquals(10, i)
    - Good: assertTrue(a.isEmpty())
    - Bad: assertTrue(i == 10)
    - Bad: assertTrue(true)

- http://www.cs.washington.edu/education/courses/cse331/11au/sections/CSE331_section4.pdf
- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect03-testing.pdf
- Email I sent out after… A1?

White-box Testing

- What it's important
- What it emphasizes: code coverage
- Categories of code coverage:
  - Statement coverage
  - Edge coverage
  - Path coverage

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect14-testing2.pdf

White-box example

What would statement, edge, path coverage be?

```
public void compute(int x, int y) {
    if (x > 0) {
        x = x-1;
    } else {
        x = x+5;
    }

    if (x > 1) {
        return y;
    } else {
        return x;
    }
}
```

Regression Testing

- What it is / what it's for
- Choosing which tests to run

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect23-regression.pdf

Exceptions, assertions

- Defensive programming
- What to check
  - Preconditions, postconditions, rep invariant, …
- How to check
  - Reasoning to prove correctness
  - assert statements at runtime
- Fail early and friendly
- Exceptions
  - Exceptions vs. preconditions
  - Catching & handling: try-catch-finally
  - Checked vs. unchecked exceptions

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect15-exceptions.pdf

Equality

- Implementing equals(), hashcode()
- Etc.
- If ovverriding equals(), override hashcode()
- Equal objects must have the same hash code

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect04-equality.pdf

Abstract Data Types

- ADT describes what the client sees
  - Abstract fields and operations
- Independent of implementation
  - One ADT, many implementations
- Abstraction function: maps implementation to ADT
  - Show how each ADT field is computed from the class's instance fields

- Ernst's handout: http://www.cs.washington.edu/education/courses/cse331/10sp/conceptual-info/abstraction-functions-and-rep-invariants.html
- Section slides: http://www.cs.washington.edu/education/courses/cse331/11au/sections/CSE331_section8.pdf
- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect05-adt1.pdf
- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect06-adt2.pdf

ADTs: Rep Invaraint

- Specifies conditions for AF to hold
- Refers to state of instance fields
  - `items != null, size > 0`, etc.
- If RI is violated, object is in an invalid state
  - indicates a problem with the implementation
- repOK /checkRep: verifies that RI holds
  - Call at the beginning and end of each method

Example: PriorityQueue

```
public class PriorityQueue {
    // sorted in nondecreasing order
    public List<Integer> vals;

    public PriorityQueue() {
        vals = new
LinkedList<Integer>();
    }

    public Integer poll() {
        if (size == 0)
            return null;
        else
            return vals.remove(0);
    }

    public Integer peek() {
        if (size == 0)
            return null;
        else
            return vals.get(0);
    }

    public boolean size() {
        return vals.size();
    }

    public Integer add(Integer i) {
        // add to list, maintaining
        // nondecreasing order
    }
}
```

Subclassing, Subtyping

- Strong, weak specifications
  - Strong: broader inputs, more specific outputs
  - Weak: more specific inputs, broader outputs
  - Strong spec can be substituted anywhere a weak spec is used
- Subtypes vs. subclasses
- Interfaces, abstract classes

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect07-subtyping.pdf

Modular design

- What it is, why it's important
- Principles:
  - Composable, decomposable
  - Understandable
  - Continuity
  - Isolation
- Maximize cohesion:
- Minimize coupling:
- Avoid "god classes"

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect08-designPrinciples.pdf

Modular design

- What it is, why it's important
- Principles:
  - Composable, decomposable
  - Understandable
  - Continuity
  - Isolation
- Maximize cohesion: group related data, behavior
- Minimize coupling: reduce dependencies between classes
- Avoid "god classes"

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect08-designPrinciples.pdf

Style

- Lots of stuff

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect09-style.pdf

Design Patterns

- Recipes for good code organization
- Creational
  - Singleton, factory, prototype, interning, flyweight
- Structural (wrappers)
  - Adapter, decorator, proxy
- Behavorial
  - Visitor
- An aside: inversion of control

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect10-designPatterns.pdf
- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect11-designPatterns2.pdf
- http://www.cs.washington.edu/education/courses/cse331/11au/sections/CSE331_section5.pdf
- http://www.cs.washington.edu/education/courses/cse331/11au/sections/CSE331_section4.pdf

Design Patterns: MVC

Model

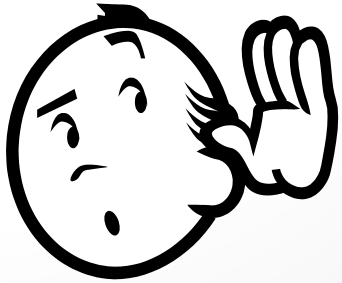
talks to data source to retrieve and store data

View

asks model for data and presents it in a user-friendly format

Controller

listens for the user to change data or state in the UI, notifying the model or view accordingly



GUI

- How GUI is built
  - JFrame, components
  - Layout managers
- How GUI handles user actions
  - ActionListeners
- Model/view separation

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect12-basicGUI.pdf
- Slides and examples from spring (used in section)

Generics

- Why to use
- How to use
- Generic classes, generic methods
- Wildcards
- Bounded types: extends, super

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect16-generics.pdf
- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect16-generics.pdf (includes demos)

Bugs

- Try to prevent them
  - Impossible by design
  - Don't write bugs!(simplicity helps)
  - Immediate visibility: catch errors early and close to source
    - Code shouldn't hide errors and keep going
- Regression testing
- Testing vs. debugging (how different?)

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect17-debugging.pdf

Debugging

- Systematic process

1. Determine repro steps
   - Simplify input, steps to narrow down the cause
   - Find similar cases where it does and doesn't repro
   - Consistent repro not always possible ("Heisenbugs")
2. Narrow down location and cause
3. Fix the bug
4. Add a regression test

- Log files can help
- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect17-debugging.pdf

Reasoning about code

- Determine what is guaranteed to be true during execution
- Hoare triples: P, code, Q => boolean
- Look for weakest (most general) precondition, strongest (most specific) postcondition
- Loop invariant

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect18-reasoning1.pdf

Reasoning about code

- Prove correctness of ADT
  - Rep invariant satisfied
  - Spec satisfied
  - Client code behaves correctly
- Proof by induction
  - Base case: invariant satisfied by constructor
  - Inductive step: if invariant holds going into each method, it holds going out

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect19-reasoning2.pdf

Usability / UI Design

- Iterative process: design-implement-evaluate-repeat
- What defines a good UI
- Design principles
- Paper prototyping
- User testing

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect20-usability.pdf

UML Diagrams

- Why to use
  - Captures program organization, control flow
  - Hard to do with a programming language
  - Helps with design before coding
- Class diagrams
- Object diagrams
- Sequence diagrams
- Example: TicTacToe w/ Violet (http://sourceforge.net/projects/violet/)

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect21-uml.pdf

Symbolic Execution

- Abstractly reason about path execution
- Use abstract values for variables
- Concolic testing: combine symbolic execution and concrete testing
  - Increase code coverage

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect24-symbolic.pdf

Version Control

- Subversion
- What it is
- Why it's important
- How to use it (high-level functions)
  - Pushing your changes
  - Getting new changes
  - Resolving conflicts
  - Etc…

- http://www.cs.washington.edu/education/courses/cse331/11au/sections/CSE331_section3.pdf

Version Control II

- Centralized vs. decentralized

- http://www.cs.washington.edu/education/courses/cse331/11au/lectures/lect25-vcsResearch.pdf

Course Evals

- I'm TA'ing 331 next quarter
- We don't get evals back til mid next quarter
- We want feedback before then
- Face-to-face
  - Fair game: course material, assignments, section, me… anything relevant to next quarter
  - What worked well?
  - What didn't work?
  - What would you change?
- Anonymous
  - Form on course website
  - For anything you're not comfortable sharing in person