

Objectives

- Coverage tools
- Object-oriented Design Principles
 - Design in the Small
 - DRY
 - Single responsibility principle
 - Shy
 - Open-closed principle

Oct 24, 2011

Sprenkle - CSCI209

1

Project 1 Questions?

- Any suggestions of strategy of what works?

Oct 24, 2011

Sprenkle - CSCI209

2

Project 1 Notes

- Test-driven development
 - Incomplete comments, pre-/post conditions
 - Make reasonable assumptions
 - Document assumptions in your test code
 - Write specification that code has to pass
- Systematically develop tests

Oct 24, 2011

Sprenkle - CSCI209

3

Project 1 Notes

- *Independent* test cases
 - Each tests different functionality
 - Should only have one failure
 - Easier to locate the bug
- Handling error cases
 - Sometimes an exception is the expected result

Add an "expected" attribute:

```
@Test(expected=IndexOutOfBoundsException.class)
public void testIndexOutOfBoundsException() {
    ArrayList emptyList = new ArrayList();
    Object o = emptyList.get(0);
}
```

Oct 24, 2011

Sprenkle - CSCI209

4

Project 1 Notes

- Do **not** change the Car class's API or its package
 - Otherwise, won't work with my Car class
- May want to write code for Car class to help you figure out tests

Oct 24, 2011

Sprenkle - CSCI209

5

Project 1 Strategies

- Organizing tests
 - Can have multiple test classes
 - Separate classes by
 - Functionality
 - Fixtures: Preconditions/Object state
 - Same (small) set up required—object(s) in certain states
 - All pass/All Errors
- Name tests clearly and consistently
 - Example: functionality_state_expectedresult

Oct 24, 2011

Sprenkle - CSCI209

6

Review

- How do we know when we've tested enough?
- How can we use coverage criteria?

Oct 24, 2011

Sprenkle - CSCI209

7

True/False Quiz

- A program that passes all test cases in a test suite with 100% path coverage is bug-free.
 - **False.**
 - Examples:
 - The test suite may cover a faulty path with data values that don't expose the fault.
 - Towards Exhaustive Testing
 - Errors of omission
 - Missing a whole if

Oct 24, 2011

Sprenkle - CSCI209

8

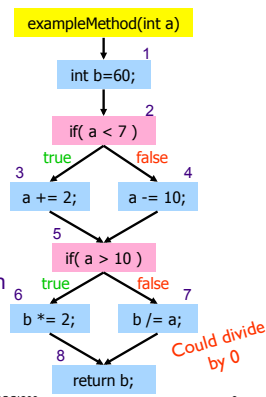
Example

Test Suite:

3-7: a=3
4-6: a=30
3-6: a=6
4-7: a=9

But, error shows up with

3-7: a=0
4-7: a=10



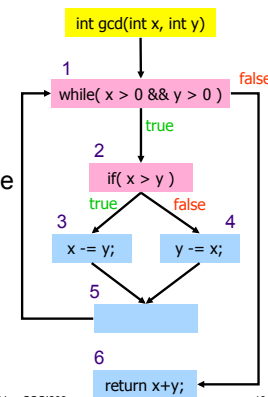
Oct 24, 2011

Sprenkle - CSCI209

9

Error of Omission

- Should verify that x and y are not negative numbers
- Can't cover that code



Oct 24, 2011

Sprenkle - CSCI209

10

True/False Quiz

- When you add test cases to a test suite that covers all statements so that it covers all branches, the new test suite is more likely to be better at exposing faults.
 - **True.**
 - You're adding test cases and covering new paths, which may have faults.

Oct 24, 2011

Sprenkle - CSCI209

11

Which Test Suite Is Better?

Statement-
adequate
Test Suite

Branch-
adequate
Test Suite

- Branch-adequate suite is not *necessarily* better than Statement-adequate suite
 - Statement-adequate suite could cover buggy paths and include input value tests that Branch-adequate suite doesn't

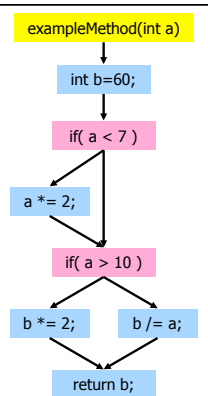
Oct 24, 2011

Sprenkle - CSCI209

12

Example

- TS1 (Statement-Adequate):
 - a=0, 6
- TS2 (Branch-Adequate):
 - a=3, 30
- Statement-adequate will find fault but branch-adequate won't
 - Covers the path that exposes the fault



Oct 24, 2011

Sprenkle - CSCI209

13

Software Testing: When is Enough Enough?

- Need to decide when tested enough
 - Balance goals of releasing application, high quality standards
- Can use program coverage as “stopping” rule
 - Also measure of confidence in test suite
 - Statement, Branch, Path and their tradeoffs
 - Use coverage tools to measure statement, branch coverage
- Still, need to use some other “smarts” besides program coverage for creating test cases

Oct 24, 2011

Sprenkle - CSCI209

14

COVERAGE TOOLS

Oct 24, 2011

Sprenkle - CSCI209

15

Coverage Tools

- Coverage is used in practice
- Don't need to figure out coverage manually
- Available tools to calculate coverage
 - Examples for Java programs: Clover, JCoverage, **Emma**
 - Measure statement, branch/conditional, method coverage

Oct 24, 2011

Sprenkle - CSCI209

16

Eclipse Plugin: EclEmma for Coverage

- Eclipse can be extended through **plugins**
 - Provide additional functionality
- EclEmma Plugin
 - Records executing program's (or JUnit test case's) coverage
 - Displays coverage graphically

Oct 24, 2011

Sprenkle - CSCI209

17

Demonstration



- Execute MediaItemTest with Coverage

Oct 24, 2011

Sprenkle - CSCI209

18

Installing Emma in Eclipse

- Under Help → Install New Software
- Add... a new remote site
 - Name: EcLemma
 - URL: <http://update.eclemma.org/>
- Select to install Emma
 - Go through process
- Restart Eclipse

Oct 24, 2011

Sprenkle - CSCI209

19

OBJECT-ORIENTED DESIGN PRINCIPLES

Oct 24, 2011

Sprenkle - CSCI209

20

Inspiration

"Fifteen years ago companies competed on price. Now it's quality. Tomorrow it's design."
Robert Hayes, Professor of Business Administration,
Harvard Business School, 2005

- *It is tomorrow!*

Oct 24, 2011

Sprenkle - CSCI209

21

Designing Systems

All systems **change**
during their life cycle

- Requirements change
- Misunderstandings in requirements
- Code must be *soft*
 - Flexible
 - Easy to change
 - New or revised circumstances
 - New contexts

Oct 24, 2011

Sprenkle - CSCI209

22

Designing Systems

All systems **change** during their life cycle

- Questions to consider:
 - How can we create designs that are stable in the face of change?
 - How do we know if our designs aren't maintainable?
 - What can we do if our code isn't maintainable?
- Answers will help us
 - Design our own code
 - Understand others' code

Oct 24, 2011

Sprenkle - CSCI209

23

Designing for Change Example

- July 2010, Oracle released Java 6 update 21
 - Generated java.dll replaced
COMPANY_NAME=Sun Microsystems, Inc. with
COMPANY_NAME=Oracle Corporation
- Change caused `OutOfMemoryError` during Eclipse launch
 - Eclipse versions 3.3-3.6 (widespread!)
 - Why? Eclipse uses the name in the DLL in startup (runtime parameters) on Windows
- Temporary Fix: Oracle changed name back
- Requires changes to all Eclipse versions

Source: <http://www.infoq.com/news/2010/07/eclipse-java-6u21>

Best Practices

- (DRY): Don't repeat yourself
- Single Responsibility Principle
- Shy
 - Avoid Coupling
- Tell, Don't Ask
- Open-closed principle
- Avoid code smells

A lot of similar, related fundamental principles

Oct 24, 2011

Sprenkle - CSCI209

25

Don't Repeat Yourself (DRY): Knowledge Representation

Every piece of knowledge must have a single, unambiguous, and authoritative representation within a system

- **Intuition:** when need to change representation, make in only one place
- Requires planning
 - What data needed, how represented (e.g., type)

Oct 24, 2011

Sprenkle - CSCI209

26

Single Responsibility Principle

There should never be more than one reason for a class to change

- **Intuition:**
 - Each responsibility is an axis of change
 - More than one reason to change
 - Responsibilities become coupled
 - Changing one may affect the other
 - Code breaks in unexpected ways

Oct 24, 2011

Sprenkle - CSCI209

27

Example

```
interface Network {
    public void connect();
    public void disconnect();
    public void send(String s);
    public String receive();
}
```

- Reasonable interface
- But has two responsibilities
 - Can you group the functionality into two responsibilities?
- Check:
 - Change for different reasons? Called from different parts of program?

Oct 24, 2011

Sprenkle - CSCI209

28

Shy Code

- Won't reveal too much of itself
- Otherwise: get *coupling*
 - Static, dynamic, domain, temporal
- Coupling isn't always bad...

What techniques have we discussed for how to keep our code shy?

Oct 24, 2011

Sprenkle - CSCI209

29

Achieving Shy Code

- Private instance variables
 - Especially mutable fields
- Make classes public only when need to be public
 - i.e., accessible by other classes → part of API
- Getter methods shouldn't return private, mutable state/objects
 - Use clone() before returning

How can you make any field immutable?

Oct 24, 2011

Sprenkle - CSCI209

30

Tell, Don't Ask

- Think of methods as “sending a message”
- Method call: sends a request to do something
 - Don't ask about details
 - Black-box, encapsulation, information hiding

Oct 24, 2011

Sprenkle - CSCI209

31

Static Coupling

- Description: Code requires other code to compile
- Problem if you drag in more than you need
 - Example: poor use of inheritance
 - Brings excess baggage
 - Inheritance is reserved for “is-a” relationships
 - Base class should not include optional behavior
 - Not “uses-a” or “has-a”
- Solution: use *composition* or *delegation* instead

Oct 24, 2011

Sprenkle - CSCI209

32

Dynamic Coupling

- Description: Code uses other code at runtime
 - `getOrder().getCustomer().getAddress().getState()`
- Why a problem: Relies on several objects/ classes and their state
 - If others change, my code has to change
- Solution: Talk *directly* to code

Oct 24, 2011

Sprenkle - CSCI209

33

Domain Coupling

- Description: Business rules, policies are embedded in code
- Why a problem: if change frequently, code has to change frequently
- Solution: put into another place (metadata)
 - Database, property file
 - Process the rules

Oct 24, 2011

Sprenkle - CSCI209

34

Temporal Coupling

- Description: Dependencies on time
 - Order that things occur
 - Occur at a certain time
 - Occur by a certain time
 - Occur at the same time
- Solution: Write *concurrent* code

Oct 24, 2011

Sprenkle - CSCI209

35

Open-Closed Principle

- Bertrand Meyer
 - Author of *Object-Oriented Software Construction*
 - Foundational text of OO programming

Principle: Software entities (classes, modules, methods, etc.) should be **open for extension** but **closed for modification**

- Design modules that *never change* after completely implemented
- If requirements change, extend behavior by adding code
 - Don't change existing code → won't create bugs!

Oct 24, 2011

Sprenkle - CSCI209

36

Attributes of Software that Adhere to OCP

- Open for Extension
 - Behavior of module can be extended
 - Make module behave in new and different ways
- Closed for Modification
 - No one can make changes to module

These attributes seem to be at odds with each other.
How can we resolve them?

Oct 24, 2011

Sprenkle - CSCI209

37

Using Abstraction

- Abstract base classes or interfaces
 - Fixed abstraction → API
 - Cannot be changed
- Derived classes: *possible behaviors*
 - Can always create new child classes of abstract base class

Oct 24, 2011

Sprenkle - CSCI209

38

TODO

- Project 1: Due **Friday**
- Extra credit opportunities

Oct 24, 2011

Sprenkle - CSCI209

39