

Objectives

- Coverage tools
- Object-oriented Design Principles
 - Design in the Small

Oct 22, 2008

Sprenkle - CS209

1

Project 1 Questions?

Oct 22, 2008

Sprenkle - CS209

2

Project 1 Notes

- Test-driven development
 - Incomplete comments, pre-/post conditions
 - Make reasonable assumptions
 - Document assumptions in your test code
 - Write the specification that code has to pass
- Organizing tests
 - Can have multiple test classes
 - Organize by fixture, functionality, all pass, all errors

Oct 22, 2008

Sprenkle - CS209

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 22, 2008

Sprenkle - CS209

4

Review

- How do we know when we've tested enough?

Oct 22, 2008

Sprenkle - CS209

5

Coverage Tools

- Coverage is used in practice
- You 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 22, 2008

Sprenkle - CS209

6

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 22, 2008

Sprenkle - CS209

7

Demonstration



- Execute MedialtemTest with Coverage
 - Note: removed BookOnTape test

Oct 22, 2008

Sprenkle - CS209

8

Installing Emma in Eclipse

- Under Help → Software Updates → Find and Install
- Search for new features to install
- Create a New Remote Site
 - Name: EclEmma
 - URL: <http://update.eclEmma.org/>
- Finish
- Select to install Emma
 - Go through process
- Restart Eclipse

Oct 22, 2008

Sprenkle - CS209

9

OBJECT-ORIENTED DESIGN PRINCIPLES

Oct 22, 2008

Sprenkle - CS209

10

Designing Systems

- All systems change during their life cycle
 - Changes in requirements
 - Misunderstandings in requirements
- Code must be *soft*
 - Flexible
 - Easy to change
 - New or revised circumstances
 - New contexts

Oct 22, 2008

Sprenkle - CS209

11

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 22, 2008

Sprenkle - CS209

12

Best Practices

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

A lot of similar, related fundamental principles

Oct 22, 2008

Sprenkle - CS209

13

DRY: Knowledge Representation

- Intuition: when need to change code, make in only one place

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

- Requires planning
 - What data needed, how represented (e.g., type)

Oct 22, 2008

Sprenkle - CS209

14

Shy Code

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

Oct 22, 2008

Sprenkle - CS209

15

Static Coupling

- Code requires other code to compile
 - Not really a bad thing
 - BUT don't 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"

Oct 22, 2008

Sprenkle - CS209

16

Dynamic Coupling

- Code uses other code at runtime
 - `getOrder().getCustomer().getAddress().getState()`
 - Relies on several objects/classes and their state
- Talk directly to code

Oct 22, 2008

Sprenkle - CS209

17

Domain Coupling

- Business rules, policies are embedded in code
 - Problem if change frequently
 - Code will have to change frequently
- Put into another place (metadata)
 - Database, property file
 - Process the rules

Oct 22, 2008

Sprenkle - CS209

18

Temporal Coupling

- Order that things occur
- Occur at a certain time
- Occur by a certain time
- Occur at the same time

➡ Write *concurrent* code

Oct 22, 2008

Sprenkle - CS209

19

Tell, Don't Ask

- Think of methods as “sending a message”
 - Method call: *sends a request to do something*
 - Don't ask about details
 - Return: *answer*

Oct 22, 2008

Sprenkle - CS209

20

Open-Closed Principle

- Bertrand Meyer
 - Author of *Object-Oriented Software Construction*
 - Foundational text of OO programming
- **Principle:** Software entities (classes, modules, functions, 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
 - Not changing existing code

Oct 22, 2008

Sprenkle - CS209

21

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 22, 2008

Sprenkle - CS209

22

Using Abstraction

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

Oct 22, 2008

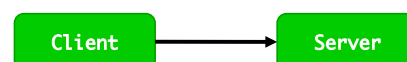
Sprenkle - CS209

23

Not Open-Closed Principle

- Client uses Server class

```
public class Client {
    public void method(Server x) {
        ...
    }
}
```



Oct 22, 2008

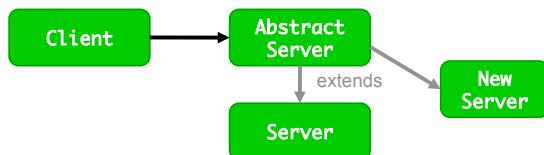
Sprenkle - CS209

24

Open-Closed Principle

- Client uses AbstractServer class

```
public class Client {
    public void method(AbstractServer x) {
        ...
    }
}
```



Oct 22, 2008

Sprenkle - CS209

25

Strategic Closure

- No significant program can be completely closed
- Must choose kinds of changes to close
 - Requires knowledge of users, probability of changes
 - Most probable changes should be closed

Oct 22, 2008

Sprenkle - CS209

26

Heuristics and Conventions

- Member variables are private
 - A function that depends on a variable cannot be closed to changes to that variable
 - The class itself can't be closed to it
 - All other classes should be
- No global variables
 - Every module that depends on global variable cannot be closed to changes to that variable
 - What happens if someone uses variable in unexpected way?
 - Counter examples: `System.out`, `System.in`

➡ Apply abstraction to parts you think are going to change

Oct 22, 2008

27

Code Smells

A hint in the code that something could be designed better

- Duplicated code
- Long method
- Large class
- Long parameter list
- Very similar subclasses
- Too many public variables
- Empty catch clauses
- Switch statements/long if statements
- Shotgun surgery
- Literals
- Global variables
- Side effects
- Using `instanceof`

Oct 22, 2008

Sprenkle - CS209

28

Duplicated Code

- What's the problem with duplicated code?
- Why do we like it?
 - What made us write the duplicated code?

Oct 22, 2008

Sprenkle - CS209

29

Duplicated Code

- Example: same expression in 2 methods of the same class
 - Extract method
 - Call method from those two places
 - `MediaItem` class example in Eclipse
- Example: duplicated code in 2 sibling subclasses
 - Extract method, put into parent class
 - If similar but not duplicate, extract the duplicate code (or parameterize)

Oct 22, 2008

Sprenkle - CS209

30