

## Objectives

- Liskov Substitution Principle
- Good enough design
- Refactoring for Extensibility

Nov 2, 2011

Sprenkle - CSCI209

1

## Reflection on Assignment 9

- How did you figure out what the code did?
- How did you make design decisions?
- Were there any particularly difficult design decisions?
  - What were the tradeoffs?
- Goal was readability; were there other issues that you ran into/designed for?

Nov 2, 2011

Sprenkle - CSCI209

2

## Assignment 9 Lessons

- Code should be soft
  - Eclipse makes code easier to change
    - The Refactor menu is a great resource
- Keep asking yourself
  - Is this understandable?
    - Will other people know what this code means?
      - Will I remember what the code means?
      - Maintaining code and bug fixes happen much more than writing new code
  - Does this code have a funny smell?
    - Literals, long methods, large classes, ...

Nov 2, 2011

Sprenkle - CSCI209

3

## Assignment 9 Lessons

- If I'm having trouble writing test cases, that may mean I need to change my code's design
  - Ex: Smaller parts

Nov 2, 2011

Sprenkle - CSCI209

4

## Review

- What are some metrics of code design?
  - How can we use the metric?
  - What is the intuition behind the metric?

Nov 2, 2011

Sprenkle - CSCI209

5

## LISKOV SUBSTITUTION PRINCIPLE

Oct 31, 2011

Sprenkle - CSCI209

6

## Liskov Substitution Principle (LSP)

- Named after Barbara Liskov
  - MIT Professor of Engineering
  - 2008 ACM Turing Award
  - Contributions to programming languages, pervasive computing
  - Trivia: first woman in the United States to receive a Ph.D. from a computer science department (Stanford, 1968)



Oct 31, 2011

Sprenkle - CSCI209

7

## Liskov Substitution Principle (LSP)

- The substitution principle:

If for each object  $o_1$  of type  $S$  there is an object  $o_2$  of type  $T$  such that for all programs  $P$  defined in terms of  $T$ , the behavior of  $P$  is unchanged when  $o_1$  is substituted for  $o_2$ , then  $S$  is a subtype of  $T$ .

- In other words...

If a module is using a base class, then it should be able to replace the base class with a derived class *without affecting the functioning of the module.*

Oct 31, 2011

Sprenkle - CSCI209

Wing &amp; Liskov, 1994

## Code Smell: Using instanceof

```
public void drawShape( Shape shape ) {
    if ( shape instanceof Square ) {
        drawSquare(shape);
    }
    else if( shape instanceof Circle ) {
        drawCircle(shape);
    }
}
```

- Why isn't this good code?
- How could we write this in a better way?

Oct 31, 2011

Sprenkle - CSCI209

9

## Code Smell: Using instanceof

- Previous example: had to know all of the Shape classes
  - Update whenever a Shape is added or removed
- Better code: **Polymorphic!**

```
public void drawShape( Shape shape ) {
    shape.draw();
}
```

Oct 31, 2011

Sprenkle - CSCI209

10

## Design by Contract

- Methods of classes declare preconditions and postconditions
  - Preconditions must be met for method to execute
  - After executing, postconditions must be true
    - Example for Rectangle's setWidth:
      - myWidth == newWidth && myHeight == oldHeight

Oct 31, 2011

Sprenkle - CSCI209

11

## Design by Contract and LSP

- Methods of classes declare preconditions and postconditions
  - Preconditions must be met for method to execute
  - After executing, postconditions must be true
    - Example for Rectangle's setWidth:
      - myWidth == newWidth && myHeight == oldHeight
- For derivatives
  - Preconditions can only be weakened
  - Postconditions can only be strengthened
  - Derivatives must adhere to constraints for base class

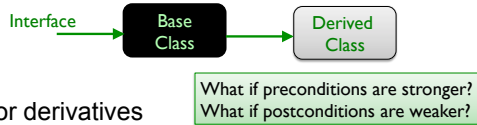
Oct 31, 2011

Sprenkle - CSCI209

12

## Design by Contract and LSP

- Recall: User interacts with interface, e.g., the base class



- For derivatives
  - Preconditions can only be weakened
  - Postconditions can only be strengthened
  - Derivatives must adhere to constraints for base class

Oct 31, 2011

Sprenkle - CSCI209

13

## Summary of LSP

- Liskov Substitution Principle (a.k.a. design by contract) is an important feature of programs that conform to the Open-Closed Principle
  - Derived types must be completely substitutable for their base types
  - Derived types can then be modified without consequence

Oct 31, 2011

Sprenkle - CSCI209

14

## Rectangle Class

```

public class Rectangle {
    private int myHeight;
    private int myWidth;

    public void setWidth( int w ) {
        myWidth = w;
    }

    public void setHeight( int h ) {
        myHeight = h;
    }

    // getters...
}
    
```

Oct 31,

15

## Square Class

- A square is a rectangle
  - But a rectangle is not a square
- In the interest of code reuse
 

```
public class Square extends Rectangle
```
- Any problems with this implementation?
  - Inherits:

```

private int myHeight;
private int myWidth;
public void setWidth( int w );
public void setHeight( int h );
    
```

Oct 31, 2011

Sprenkle - CSCI209

16

## To Keep Square Consistent...

```

public void setWidth( int w ) {
    super.setWidth(w);
    super.setHeight(w);
}

public void setHeight( int h ) {
    super.setWidth(h);
    super.setHeight(h);
}
    
```

Oct 31, 2011

Sprenkle - CSCI209

17

## But What About Users of Classes?

- Consider the function:
 

```
public void testMethod( Rectangle r ) {
    r.setWidth(5);
    r.setHeight(4);
    assertEquals(20, r.getWidth()*r.getHeight());
}
```
- What happens if method is called with a Square object?

Oct 31, 2011

Sprenkle - CSCI209

18

## The Problem

- A Square object is *not* a Rectangle object
- Behaviors are different
- Clients depend on behaviors

**Lesson:** All derivatives of class **must** have the same **behavior**

Oct 31, 2011

Sprenkle - CSCI209

19

## Discussion of Abstraction

- What does abstraction allow?
- Are there any limitations to abstraction?

Oct 31, 2011

Sprenkle - CSCI209

20

## Summary of Designing for Change

Use **abstraction** for code that is *likely to change*

- Can depend on code that is *stable* and unlikely to change
  - Example of stable code: `System.out`

Oct 31, 2011

Sprenkle - CSCI209

21

## Refactoring Summary

- Write code and then **rewrite** code
  - Eye toward extensibility, flexibility, maintainability, and readability
  - Maintain correctness
- Reading/understanding other people's code can be difficult
  - Make your code readable, understandable
- Probably impossible to design/write "correctly" the first time
  - A lot harder to get the logic right, make sure you're not creating bugs, know/check the right answer...
  - Could cause yourself headaches coding this way first

Oct 31, 2011

Sprenkle - CSCI209

22

## Good-Enough Design Discussion

### Perfect Design

- ✓ Follows all design principles
- OCP, Single Responsibility, no code smells, ...
- May not be possible
- Infinite refactoring, development
- Code never released

### Good-enough Design

- Not everyone agrees on design
- Maintenance requires changes to a few places
- ✓ Code gets released to customers

Similar tradeoffs in testing

Oct 31, 2011

Sprenkle - CSCI209

23

## PMD Reports

- Java source code analyzer
- Looks for possible bugs, poor coding practices
  - Duplicate code
  - Dead code
  - Empty if/while/catch blocks
  - Suboptimal code (e.g., Strings, StringBuffer)
  - And more!
- Eclipse Plugin:
  - Update site: <http://pmd.sourceforge.net/eclipse>

Oct 31, 2011

Sprenkle - CSCI209

See Also: FindBugs

24

## REFACTORING FOR EXTENSIBILITY

Oct 31, 2011

Sprenkle - CSCI209

25

## Simulating a Roulette Game

- See handout

Oct 31, 2011

Sprenkle - CSCI209

26

## Get a Solution

- Import Existing Project
  - /home/courses/cs209/handouts/roulette.tar

Oct 31, 2011

Sprenkle - CSCI209

27

## Understanding Code

- Execute the code
  - What is the main driver for this project?
- What are each class's responsibilities?
- What does `test.RouletteTestSuite` do?

Oct 31, 2011

Sprenkle - CSCI209

28

## Bug in the Code

- Determining if Odd/Even Bet was won is incorrect

Oct 31, 2011

Sprenkle - CSCI209

29

## Understanding Code

- Focus: how **open** is the code to adding **new** kinds of **bets** and how **closed** it is to **modification**?
  - How many classes know about the `Bet` class?
  - What code would need to be added to `Game` to allow the user to make another kind of bet that paid one to one odds and was based on whether the number spun was high (between 19 and 36) or low (between 1 and 18)?

Oct 31, 2011

Sprenkle - CSCI209

30

## Looking Ahead

- Friday:
  - Extra Credit Opportunity: 12:30 p.m., Women's Resource Room
  - 3:30 p.m., Noah Egorin
- Next Wednesday: Roulette Refactoring due
- Next Friday: 2<sup>nd</sup> Exam
  - Focus: Python vs. Java, collections, testing, coverage, design principles (tradeoffs), GUIs
  - Terminology