

Objectives

- Wrap-up Open-closed principle
- Code Smells
- Refactoring

Oct 26, 2011

Sprenkle - CSCI209

1

Who is John McCarthy?

Oct 26, 2011

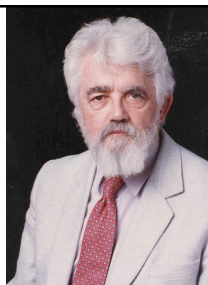
Sprenkle - CSCI209

2

John McCarthy

- Father of Lisp and AI
- 1971 Turing Award winner

"He who refuses to do arithmetic is doomed to talk nonsense."



"Program designers have a tendency to think of the users as idiots who need to be controlled. They should rather think of their program as a servant, whose master, the user, should be able to control it. If designers and programmers think about the apparent mental qualities that their programs will have, they'll create programs that are easier and pleasanter — more humane — to deal with."

What is the output?

```
System.out.println("The answer is " +
    100L + 3.3f);
```

- a) The answer is 1003.3
- b) The answer is 103.3
- c) The answer is 100
- d) Error (no output).

Oct 26, 2011

Sprenkle - CSCI209

4

Project 1

- In tests, can (probably should) have multiple assert statements
 - Method exits (fails) if assert is false
 - Continues executing remaining asserts
- Reminder: don't change the API, package of Car class
- Any more questions? Strategy suggestions?

Oct 26, 2011

Sprenkle - CSCI209

5

Review

- Why is coverage not enough?
 - What can we do to improve testing?
- What is guaranteed in software development?
- What are some principles of design in Object-oriented Programming to address the challenge posed by that guarantee?

Oct 26, 2011

Sprenkle - CSCI209

6

Review: Best Practices

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

Oct 26, 2011

Sprenkle - CSCI209

7

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 26, 2011

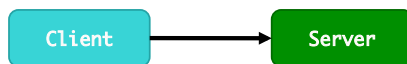
Sprenkle - CSCI209

8

Not Open-Closed Principle

- Client uses Server class

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



Oct 26, 2011

Sprenkle - CSCI209

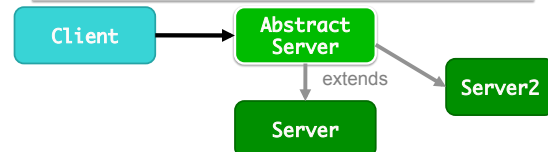
9

Open-Closed Principle

Or ServerInterface

- Client uses AbstractServer class

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



Oct 26, 2011

Sprenkle - CSCI209

10

Strategic Closure

- No significant program can be completely closed
- Must choose kinds of changes to close
 - Requires knowledge of users, probability of changes
- **Goal: Most probable changes should adhere to open-closed principle**

Oct 26, 2011

Sprenkle - CSCI209

11

Heuristics and Conventions

- Member variables are private
 - A method 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 26, 2011

12

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 26, 2011

Sprenkle - CSCI209

13

Duplicated Code

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

What can we do when we have duplicated code?
(How can we get rid of the duplicate code?)

Oct 26, 2011

Sprenkle - CSCI209

14

Duplicated Code

- What's the problem with duplicated code?
 - If code changes, need to change in every location
 - Duplicate effort to test code to make sure it works
 - More statements for test suite to test!

Oct 26, 2011

Sprenkle - CSCI209

15

Example Code

```
public int indexOfPresentItem(String title) {
    int cur = 0;
    for (MediaItem i : this.collection) {
        if (i.getTitle().equals(title) && i.isPresent())
            return cur;
        cur++;
    }
    return -1;
}

public int indexOfAbsentItem(String title) {
    int cur = 0;
    for (MediaItem i : this.collection) {
        if (i.getTitle().equals(title) && !i.isPresent())
            return cur;
        cur++;
    }
    return -1;
}
```

Parameterize

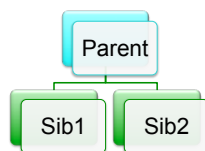
Oct 26, 2011

Sprenkle - CSCI209

16

Duplicated Code

- Example: same expression in at least one method of a class
 - Solution: Extract method
 - Call method from those two places
- Example: duplicated code in 2 sibling child classes



Oct 26, 2011

Sprenkle - CSCI209

17

Duplicated Code

- Example: duplicated code in 2 sibling child classes
 - Extract method, put into parent class
 - Eclipse: extract method, pull up
 - If similar but not duplicate, extract the duplicate code or parameterize
- Example: duplicated code in unrelated classes

Oct 26, 2011

Sprenkle - CSCI209

18

Duplicated Code

- Example: duplicated code in unrelated classes
 - Ask: where does method belong?
 - One solution:
 - Extract class
 - Use new class in classes
 - Another solution:
 - Keep in one class
 - Other class calls that method

Oct 26, 2011

Sprenkle - CSCI209

19

Refactoring: Solution to Code Smells

Refactoring: Updating a program to improve its design and maintainability without changing its current functionality significantly

- Example
 - Creating a single method that replaces 2 or more sections of similar code
 - Reduces redundant code
 - Makes code easier to debug, test

After refactoring your code, what should you do next?

Oct 26, 2011

Sprenkle - CSCI209

20

Long Methods

- What's the problem with long methods?
- What made us write them?
- How can we fix them?
- What is an issue with lots of short methods?

Oct 26, 2011

Sprenkle - CSCI209

21

Long Methods: Issues and Solutions

- Issues:
 - Hard to understand (see) what method does
 - Smaller methods have reader overhead
 - Look at code for called methods
 - But, should use descriptive names
 - In Eclipse, use F3 to jump to a method's definition
- Solutions:
 - Find lines of code that go together (may be identified by a comment) and extract method

Oct 26, 2011

Sprenkle - CSCI209

22

Large Class

- What's the problem?

Oct 26, 2011

Sprenkle - CSCI209

23

Large Class

- Issue: Too many instance variables → trying to do too much
 - Violates **Single Responsibility Principle**
- Solutions: Possible example: `UserInterface`
 - Bundle groups of variables together into another class
 - Look for common prefixes or suffixes
 - If includes optional instance variables (only sometimes used), create child classes
 - Look at how users use the class for ideas of how to break it up

Eclipse: Refactor → Extract Class or Extract Superclass

Oct 26, 2011

24

Long Parameter List

- More difficult to use (do I have everything?)
 - Example: `MediaItem`, subclass constructors
- If method signature changes, have a lot of places to change
- Solutions: Use objects
 - Instead of separate parameters for an object's data
 - Group parameters together

Eclipse: Refactor → Introduce Parameter Object
OR Refactor → Change Method Signature

Oct 26, 2011

Sprenkle - CSCI209

25

- What does this code mean?

```
// generates a random int so the jewel moves at
// random intervals
if (numIters == random.nextInt(100) + 85) {
    treasure.move(this);
    numIters = 0;
}
```

Oct 26, 2011

Sprenkle - CSCI209

26

Literals or Magic Numbers

- If a number has a special meaning, make it a constant
 - Distinguish between 0 and `NO_VALUE_ASSIGNED`
 - If value changes (-1 instead of 0), only one place to change

Eclipse: Refactor → Extract Constant

Oct 26, 2011

Sprenkle - CSCI209

27

TO DO

- Project 1 due Friday

Oct 26, 2011

Sprenkle - CSCI209

28