

Objectives

- Wrap up Dependency Inversion Principle
- Design Patterns: Observer, MVC
- Analysis and Design

Nov 16, 2011

Sprenkle - CSCI209

1

Testing Project Notes

- Overall good
- Common specification issues
 - Park when refuel; can't park with go; invalid gears
- Common test issues
 - Incorrect expected results; incorrect states to call methods; missing @Test; missing tests for some methods
- JUnit discussion
 - What does JUnit do for you? Make easier for you?

Nov 16, 2011

Sprenkle - CSCI209

2

Review

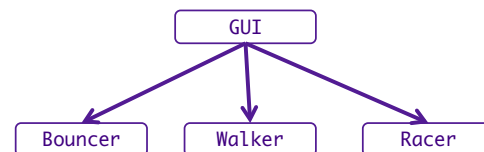
- What is a design pattern?
- What design patterns did we discuss?
 - What design principle(s) does it follow?
- Why do we prefer composition over inheritance?
- What design pattern is used in the screen savers code?
- Any underlying commonalities?

Nov 16, 2011

Sprenkle - CSCI209

3

One Option for Screen Saver Dependencies



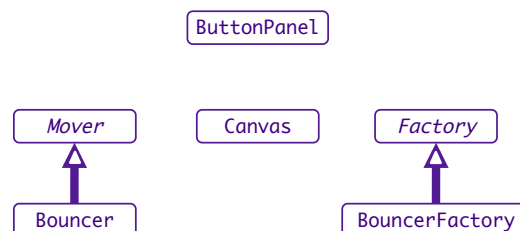
Violates Dependency Inversion Principle:
High-level component is dependent on concrete classes.
If implementations change, GUI may have to change

Nov 16, 2011

Sprenkle - CSCI209

4

Our Screen Saver Dependencies

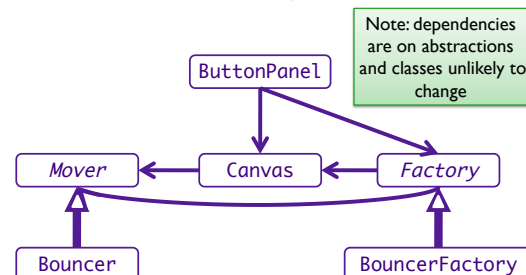


Nov 16, 2011

Sprenkle - CSCI209

5

Our Screen Saver Dependencies



Nov 16, 2011

Sprenkle - CSCI209

6

Guidelines to Follow Dependency Inversion Principle

- No variable should hold a reference to a concrete class
 - Using `new` → holding reference to concrete class
 - Use factory instead
- No class should derive from a concrete class
 - Why? Depends on a concrete class
 - Derive from an interface or abstract class instead
- No method should override an implemented method of its base class
 - Base class wasn't an abstraction
 - Those methods are meant to be shared by subclasses

What's an issue with following all of these guidelines?

Nov 16, 2011

7

Dependency Inversion Principle

**Depend upon
abstractions**

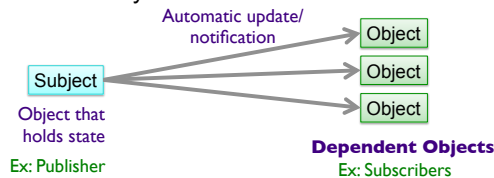
Nov 16, 2011

Sprenkle - CSCI209

8

Design Pattern: Observer

- Defines a 1-to-many dependency between objects
- When one object changes state, all of its dependents are notified and updated automatically



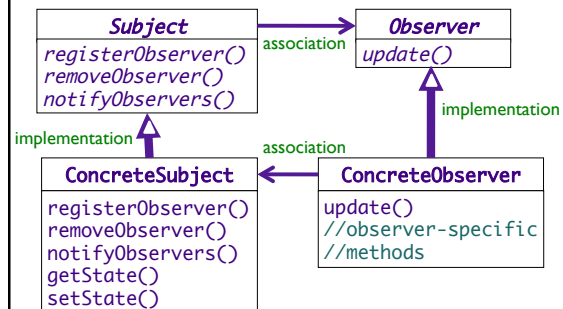
Nov 16, 2011

Sprenkle - CSCI209

9

Observer Pattern

Have we seen this pattern?



Nov 16, 2011

Sprenkle - CSCI209

10

Design Principle: Loose Coupling

- A principle behind Observer pattern

Goal: loosely coupled designs
between objects that interact

- Loosely coupled objects can interact but have very little knowledge of each other
 - Minimize dependency between objects
 - More flexible systems
 - Handle change

Nov 16, 2011

Sprenkle - CSCI209

11

Model - Viewer - Controller (MVC)

- A common **design pattern** for GUIs
- Separate
 - Model: application data
 - View: graphical representation
 - Controller: input processing



Nov 16, 2011

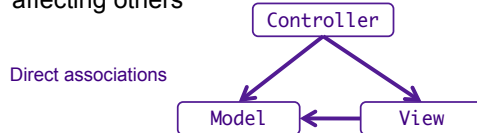
Sprenkle - CSCI209

12

Model-Viewer-Controller



- Can have multiple viewers and controllers
- Goal: modify one component without affecting others



Nov 16, 2011

Sprenkle - CSCI209

13

Model



- Code that carries out some task
- Nothing about how view presented to user
- Purely **functional**
- Must be able to register views and notify views of changes

Nov 16, 2011

Sprenkle - CSCI209

14

Multiple Views

- Provides GUI interface components of model
 - Look & Feel of the application
- User manipulates view
 - Informs **controller** of change
- Example of multiple views: spreadsheet data
 - Rows/columns in spreadsheet
 - Pie chart, bar chart, ...



Nov 16, 2011

Sprenkle - CSCI209

15

Controller(s)



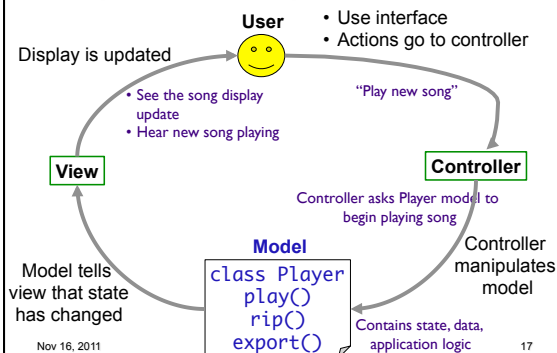
- Takes user input and figures out what it means to the model
 - Makes decisions about behavior of model based on UI
- Update **model** as user interacts with **view**
 - Calls model's mutator methods
- Views are associated with controllers

Nov 16, 2011

Sprenkle - CSCI209

16

Example: Music Player



Nov 16, 2011

17

MVC: Combination of Design Patterns

Nov 16, 2011

Sprenkle - CSCI209

18

MVC: Combination of Design Patterns

- Observer
 - Views, Controller notified of Model's state changes
- Strategy
 - View can plug in different controllers
 - Different views of the same model
- Composite
 - View is a composite of GUI components
 - Top-level component learns about model update, updates components
 - A container computes its preferred size by combining all the preferred sizes of its components

Nov 16, 2011

Sprenkle - CSCI209

19

Code Analysis

- Consider GUIs we've seen
 - Which use the MVC pattern?
 - Identify M, V, and C in applicable GUIs

Nov 16, 2011

Sprenkle - CSCI209

20

ANALYSIS & DESIGN: FORMALIZED

Nov 16, 2011

Sprenkle - CSCI209

21

Design Heuristics

- Model real world whenever possible
- Avoid all-powerful (omnipotent) classes
- Distribute system intelligence among classes evenly
 - Top-level classes should share work uniformly
 - More easily understood system
 - More easily communicated design
- Minimize # of messages between class and helper
 - Reduce coupling btw class and helper

Nov 16, 2011

Sprenkle - CSCI209

22

Analysis Phase

- Create an abstract model in client's vocabulary
- Strategy:
 1. Identify classes that model (shape) system as set of abstractions
 2. Determine each class's purpose, or main responsibility
 - member functions
 - data members
 3. Determine helper classes for each
 - Help complete responsibilities


 "Dohickey"

Nov 16, 2011

Sprenkle - CSCI209

23

Analysis Phase Discussion

- Expect to **iterate**
 - Won't find all classes at first
 - Especially helpers
 - Won't know all responsibilities
- Uncertainty in problem statement
 - May be concerns that need to be settled
 - Try to understand requested software system at level of those requesting software
- Rarely one true correct best design



Nov 16, 2011

Sprenkle - CSCI209

24

Identification of Classes

- Potentially model the system
- Usually **nouns** from problem description or from domain knowledge
- Model real world whenever possible
 - More understandable software
 - Helps during maintenance when someone unfamiliar with system must update/fix code

Nov 16, 2011

Sprenkle - CSCI209

25

Identifying Responsibilities

- Responsibilities convey purpose of class, its role in system
- Questions to Ask:
 - What are the other responsibilities needed to model the solution?
 - Which class should take on this particular responsibility?
 - What classes help another class fulfill its responsibility?

Nov 16, 2011

Sprenkle - CSCI209

26

Have You Modeled Everything?

- Strategy: Role playing
- Act as different classes: can you do everything you want in various scenarios?
 - Fill in missing classes, responsibilities
 - Methods: parameters, what returned
 - Restructure as necessary
 - No code yet so not actually refactoring
- Example **use cases**/scenarios:
 - User borrows a video and returns it two days late
 - User tries to borrow book that is already checked out

Nov 16, 2011

Sprenkle - CSCI209

27

Discussion

- What else can use cases be used for?

Nov 16, 2011

Sprenkle - CSCI209

28

Discussion

- What else can use cases be used for?
 - Test Cases

Nov 16, 2011

Sprenkle - CSCI209

29

TEAM FINAL PROJECT

Nov 16, 2011

Sprenkle - CSCI209

30

Project Deliverables Timeline

Deliverable	Who	Weight	Due Date
Preparation	Individual	8%	Mon, 11/28
Preliminary Implementation	Team	37%	Mon, 12/5
Final Implementation	Team	40%	You decide → latest 12/16
Analysis	Individual	15%	12/16

Nov 16, 2011

Sprenkle - CSCI209

31

Teams

- Sophomores: Suraj, Garrett, Richard, Anton
- Juniors: Amy, Hank, Phil, Shannon
- Seniors: Andrew, Anna, Mike

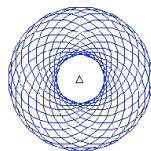
Nov 16, 2011

Sprenkle - CSCI209

32

SLogo Project Overview

- Goal: Create an IDE for simplified version of Logo
- Logo: programming language designed to teach children to program
 - Low floor, high ceiling



Nov 16, 2011

Sprenkle - CSCI209

33

SLogo Functionality Overview

- User enters SLogo commands
 - Commands defined by specification
- Interpreted and batch modes for entering commands
 - User can save files of commands
- Have turtle execute the commands
 - Or descriptive error messages
- Many possible extensions

Nov 16, 2011

Sprenkle - CSCI209

34

High-Level Design Brainstorm

- What are the pieces that you'll need to implement for this project?

Nov 16, 2011

Sprenkle - CSCI209

35

Programming Language Syntax

- What does an identifier look like in Java?
- What does an assignment statement look like in Java?
- What can be on the left hand side?
- What can be on the right hand side?
- What does a multiplication look like?
- How do we evaluate arithmetic expressions?

Nov 16, 2011

Sprenkle - CSCI209

36

Programming Language Design

- Must be unambiguous
 - Programming Language defines a syntax and semantics
- Interpreting programming languages
 - Parse program into tokens
 - Example: $x = 4 * 3;$ →

`<id> <assignment> <num> <mult> <num> <endofstmt>`

- Verify that tokens are in a valid form
- Generate executable code

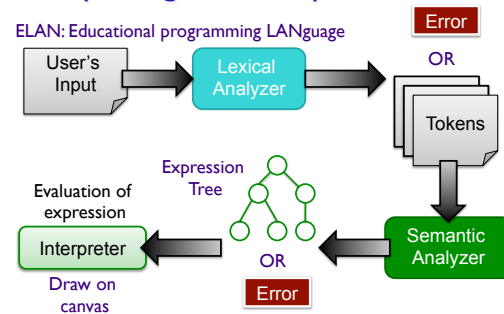
Nov 16, 2011

Sprenkle - CSCI209

37

Interpreting User's Input

ELAN: Educational programming LANguage

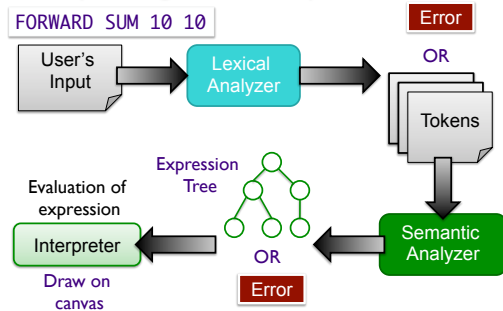


Nov 16, 2011

Sprenkle - CSCI209

38

Interpreting User's Input



Nov 16, 2011

Sprenkle - CSCI209

39

What We Need to Do/Represent

- Lexical Analysis
- Semantic Analysis
- Evaluation

Nov 16, 2011

Sprenkle - CSCI209

40

What We Need to Do/Represent

- Lexical Analysis
 - Recognize/create tokens
 - Report errors in creating tokens
- Semantic Analysis
 - Convert infix tokens into postfix
 - Report errors
 - Parse tokens into *expressions*
 - Report errors
- Evaluation
 - Evaluate expressions with respect to turtle/model

Nov 16, 2011

Sprenkle - CSCI209

41

TODO

- Assignment 11 due Friday
- Project Analysis due Monday after Thanksgiving

Nov 16, 2011

Sprenkle - CSCI209

42