

## Objectives

- Wrap up Design Patterns
- Designing APIs

Nov 30, 2011

Sprenkle - CSCI209

1

## SLogo Notes

- No final exam: final project is 20% of your grade
- FORWARD: in current direction
  - Not the same as North
- Remember: Use use cases to see what you're missing
- Can use the WWW to search for help (within reason)
  - OK: "Handling files in Java"
  - Not OK: "Implementing Turtle Graphics in Java"
  - Cite your sources

Nov 30, 2011

Sprenkle - CSCI209

2

## SLogo Notes

- Common extensions:
  - Undo/redo
  - Tail modifications
  - Custom turtle images
  - Showing variable values
- Post-mortem
  - Text file or PDF—**not** a ODT, DOC, or DOCX file

Nov 30, 2011

Sprenkle - CSCI209

3

## Review

- What is a design pattern?
- What design patterns have we discussed?
  - What design principle(s) does it follow?

Nov 30, 2011

Sprenkle - CSCI209

4

## Review: Design Pattern

General reusable solution to a commonly occurring problem in software design

- Not a finished design that can be transformed directly into code
- Description or *template* for how to solve a problem that can be used in many different situations
  - "Experience reuse", rather than code reuse

Nov 30, 2011

Sprenkle - CSCI209

5

## Summary of Design Patterns

Pattern	When to Use
Delegation	
Strategy	
Factory	
Observer	
Decorator	

## Summary of Design Patterns

Pattern	When to Use
Delegation	Have something that could be a separate responsibility/class
Strategy	Have a variety of algorithms that can be encapsulated to solve a given problem
Factory	To generate objects without knowing their concrete class
Observer	Have objects that need to be notified of changes to other objects
Decorator	Want to add (possibly many additive) behaviors

Not all of the design patterns

Nov 30, 2011

Sprenkle - CSCI209

7

## What's Your Coffee Drink?

- How can we represent the various beverages?
- What are the possible implementation issues?

Nov 30, 2011

Sprenkle - CSCI209

8

## What's Your Coffee Drink?

Beverage
description milk soy flavoring whippedcream
getDescription() cost() hasMilk() setMilk() ...

How many additional methods will we need to add to create a comprehensive beverage object?

How will we compute cost?

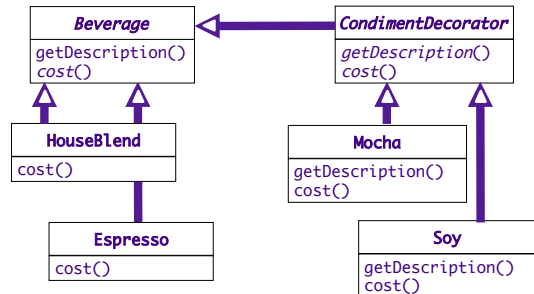
What happens when a new beverage feature is added?

Nov 30, 2011

Sprenkle - CSCI209

9

## One Solution: Decorator



Nov 30, 2011

UML Diagram

Sprenkle - CSCI209

10

## Mocha's Implementation

```

public class Mocha extends CondimentDecorator {
    private Beverage beverage;

    public Mocha(Beverage beverage) {
        this.beverage = beverage;
    }

    public String getDescription() {
        return beverage.getDescription() + ", Mocha";
    }

    public double cost() {
        return .20 + beverage.cost();
    }
}

```

What design patterns are used within this class?  
How would we use this class?  
How would we create other beverages?

Nov 30, 2011

## Mocha's Implementation

```

public class Mocha extends CondimentDecorator {
    private Beverage beverage;

    public Mocha(Beverage beverage) {
        this.beverage = beverage;
    }

    public String getDescription() {
        return beverage.getDescription() + ", Mocha";
    }

    public double cost() {
        return .20 + beverage.cost();
    }
}

```

Handles part it knows about,  
Delegates rest to Beverage

Generalize: when to use the Decorator pattern,  
tradeoffs of this design pattern

Nov 30, 2011

## Design Pattern: **Decorator**

- Used to add behavior to an object dynamically
  - Typically added by doing computation before or after an existing method in the object
- Benefits:
  - Alternative to inheritance
  - Can add any number of decorators
- Possible drawback:
  - Could add many small classes → less than straightforward for others to understand

Nov 30, 2011

Have we seen decorators used in practice?

13

## Change in Requirements

- Beverage class has two new methods: `setSize(...)` and `getSize()`
- Condiments should be charged according to size
  - Example: Soy costs 10¢, 15¢ and 20¢ respectively for small, medium, and large

How would you alter the decorator classes to handle this change in requirements?

Nov 30, 2011

Sprenkle - CSCI209

14

## Handling Change in Requirements

```
public double cost() {
    double cost = beverage.cost();

    if (getSize() == Beverage.SMALL) {
        cost += .10;
    } else if (getSize() == Beverage.MEDIUM) {
        cost += .15;
    } else if (getSize() == Beverage.LARGE) {
        cost += .20;
    }
    return cost;
}
```

Nov 30, 2011

Sprenkle - CSCI209

15

## Friday: Guest Speaker David Shepherd

- Designer of productivity tools at ABB
- Talk: "From Student to Professional: Key Software Development Skills Not Taught in (Most) Computer Science Curricula"
  - Purpose: To introduce students to a professional developer's toolset and highlight commonly missing skills
- Read two articles on course schedule page

Nov 30, 2011

Sprenkle - CSCI209

16

## Friday: Guest Speaker David Shepherd

- On Sakai forum for class, answer questions about the talk
  - What were the three most important points in his talk?
  - What was the most surprising thing he mentioned about the differences between school and production development?
  - What specific tool/technique mentioned today were you least familiar with prior to the talk? Do you think you will use this tool? Why or why not?
  - What are the costs of learning a tool vs the productivity gains? How can productivity tool developers decrease costs while increasing productivity gains?
  - In your opinion, what method for gaining experience is the easiest and why?
- Due on Monday

Nov 30, 2011

Sprenkle - CSCI209

17

## Looking Ahead

- David Shepherd – Friday
  - Read articles found on course schedule page
- Monday: 1<sup>st</sup> SLogo prototype due

Nov 30, 2011

Sprenkle - CSCI209

18