

Objectives

- Code Smells
- Liskov Substitution Principle
- Design Patterns

Nov 5, 2021

Sprenkle - CSCI209

1

1

Before

```
# Check if path exists. if it doesn't, exist create directory
if not os.path.exists(config['output_dir'] +
    config['app_name'] + config['bot_dir'] + 'analysis'):
    os.makedirs(config['output_dir'] +
        config['app_name'] + config['bot_dir'] + 'analysis')
print("Created directory for" + config['output_dir'] +
    config['app_name'] + config['bot_dir'] + 'analysis')
```

Nov 5, 2021

Sprenkle - CSCI209

2

2

After

```
# Check if path exists. if it doesn't, exist create directory
outputDir = config['output_dir'] +
            config['app_name'] + config['bot_dir'] + 'analysis'
if not os.path.exists(outputDir):
    os.makedirs(outputDir)
    print("Created directory for", outputDir)
```

Nov 5, 2021

Sprenkle - CSCI209

3

3

Review

1. What is guaranteed in software development?
 - This informs how we design our code
2. What is the process for writing maintainable code?
 - Define the terms in that process
3. What are some code smells and how do we address them?
 - What is common to how we address code smells?
 - What was the code smell in Roulette's code?
4. What is the open-closed principle?
 - How does it relate to the Roulette code base?

Nov 5, 2021

Sprenkle - CSCI209

4

4

Review: 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

Nov 5, 2021


Sprenkle - CSCI209

5

5

Revised Process to Write Maintainable Code

Apply the design principles, but as your code evolves, you'll see that you didn't always adhere to the principles

1. Identify code smell 
2. **Refactor** code to remove code smell
3. **Test** to confirm code still works!

Nov 5, 2021

Sprenkle - CSCI209

6

6

Review: Code Smells

A hint in the code that something could be designed better

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

Nov 5, 2021

Sprenkle - CSCI209

7

7

Review: 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?
 - Always consider: how is this code likely to change?
- How could we write this in a better way?

8

Review: 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!**
 - There was a draw method specific to each Shape
 - Refactor those methods into Shape child classes

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

Nov 5, 2021

9

9

Review: Open-Closed Principle

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
 - By not changing existing code → we won't create bugs!
- Closed: APIs/interfaces
- Open: add new implementations

Nov 5, 2021

Sprenkle - CSCI209

10

10

Lazy Class

- Problem
 - Class in question doesn't do much
 - Classes cost time and money to maintain and understand
- How could this happen?
 - Refactoring!
 - Planned to be implemented but never happened
- Solution
 - Get rid of class
 - Inline or collapse subclass into parent class

Nov 5, 2021

Sprenkle - CSCI209

11

11

Speculative Generality

- Beware of too much abstraction, allowing for too much flexibility that isn't required
- Solution: Collapse classes

Nov 5, 2021

Sprenkle - CSCI209

12

12

Design By Contract

LISKOV SUBSTITUTION PRINCIPLE

Nov 5, 2021

Sprenkle - CSCI209

13

13

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 code 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 code.*

Nov 5, 2021

Sprenkle - CSCI209

Liskov & Wing, 1994

14

Design by Contract

- By Bertrand Meyer (Open-Closed Principle)
- Methods of classes should 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`

Nov 5, 2021

Sprenkle - CSCI209

15

15

Design by Contract and LSP

- Methods of classes should 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 derived/child classes
 - Preconditions can only be *weakened*
 - Postconditions can only be *strengthened*
 - Derivatives must adhere to base class's constraints

Nov 5, 2021

Sprenkle - CSCI209

16

16

Design by Contract and LSP

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



What if preconditions are stronger?
What if postconditions are weaker?

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

Nov 5, 2021

Sprenkle - CSCI209

17

17

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...
}
  
```

Nov 5, 2021

Sprenkle - CSCI209

18

18

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 );
```

Nov 5, 2021

Sprenkle - CSCI209

19

19

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);
}
```

Nov 5, 2021

Sprenkle - CSCI209

20

20

But What About Users of Classes?

- Consider the test method:

```
public void testMethod( Rectangle r ) {  
    r.setWidth(5);  
    r.setHeight(4);  
    assertEquals(20, r.getWidth()*r.getHeight());  
}
```

- What happens if a `Square` object is passed into method?

Nov 5, 2021

Sprenkle - CSCI209

21

21

The Problem

- A `Square` object is *not* a `Rectangle` object
- Behaviors w.r.t. pre-/post-condition contract are different
- Clients depend on those behaviors

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

Nov 5, 2021

Sprenkle - CSCI209

22

22

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

Nov 5, 2021

Sprenkle - CSCI209

23

23

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)



There is an advanced lab machine named after her.

Nov 5, 2021

Sprenkle - CSCI209

Liskov & Wing, 1994

25

25

& Wing

- Jeannette Wing
 - Executive Vice President of Research at Columbia University
 - Big proponent of computational thinking as assistant director for Computer and Information Science and Engineering at the NSF from 2007 to 2010



Nov 5, 2021

Sprenkle - CSCI209

26

26

How can we create designs that are stable in the face of change?

DESIGN PATTERNS

Nov 5, 2021

Sprenkle - CSCI209

27

27

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 5, 2021

Sprenkle - CSCI209

28

28

Defined Design Patterns

- Software best practices
- Catalogued and discussed in *Design Patterns: Elements of Reusable Object-Oriented Software*
 - Written by the “**Gang of Four**”: Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides
 - Erich Gamma also co-wrote original JUnit framework
 - Didn’t design the patterns; identified them

Nov 5, 2021

Sprenkle - CSCI209

29

29

Understanding Code with Design Patterns

1. Recognize design pattern in code base you're using
2. Understand code design better

Nov 5, 2021

Sprenkle - CSCI209

30

30

Applying Design Patterns

1. Recognize problem as one that can be solved by a design pattern
2. Apply pattern to your problem

Danger: over-applying design patterns
➤ Fall back: Identify and resolve code smells

Nov 5, 2021

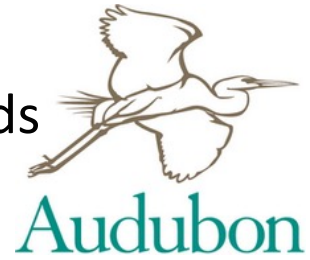
Sprenkle - CSCI209

31

31

Audubon Society calls...

- Need to represent all the different birds
 - Various flying behaviors (some fly, some don't)
 - Make different sounds
 - Examples: Duck, Penguin, Hummingbird, Ostrich, Chicken, Oriole, ...



How can we represent different birds?

Nov 5, 2021

Sprenkle - CSCI209

32

32

Solution Non-Starter: Hierarchy of Classes (under Bird parent class)

- FlyingBird
 - FlyHighBird
 - ScreechingFlyHighBird
 - Eagle
 - ...
 - FlyLowBird
 - SingingFlyLowBird
 - SquawkingFlyLowBird
 - ...
- FlightlessBird
 - SingingFlightlessBird
 - SquawkingFlightlessBird
 - ...

What design principle do these classes violate?

Identify what is likely to change/vary:

- Flying
- Sound

Nov 5, 2021

33

Towards a Solution: Designing Flexible Behaviors

- Include behaviors in abstract Bird class
 - FlyBehavior has performFly() method
 - SoundBehavior has makeSound() method
- Could have setter methods in Bird class to change these
 - Example: bird's wings get clipped

Nov 5, 2021

Sprenkle - CSCI209

34

34

Designing Flexible Behaviors

```
public abstract class Bird {
    protected FlyBehavior flyB;
    protected SoundBehavior soundB;

    public Bird() {
        ...
    }

    public void performSound() {
        soundB.makeSound();
    }

    public void performFly() {
        flyB.performFly();
    }
}
```

Nov 5, 2021

35

35

Designing Flexible Behaviors

```
public class Duck extends Bird {
    //Recall: protected FlyBehavior flyB;
    //Recall: protected SoundBehavior soundB;

    public Duck() {

    }
    ...
}
```

← What do we need to do in here?

Nov 5, 2021

Sprenkle - CSCI209

36

36

Designing Flexible Behaviors

```
public class Duck extends Bird {
    //Recall: protected FlyBehavior flyB;
    //Recall: protected SoundBehavior soundB;

    public Duck() {
        flyB = new FlyHighBehavior();
        soundB = new QuackBehavior();
    }
}
```

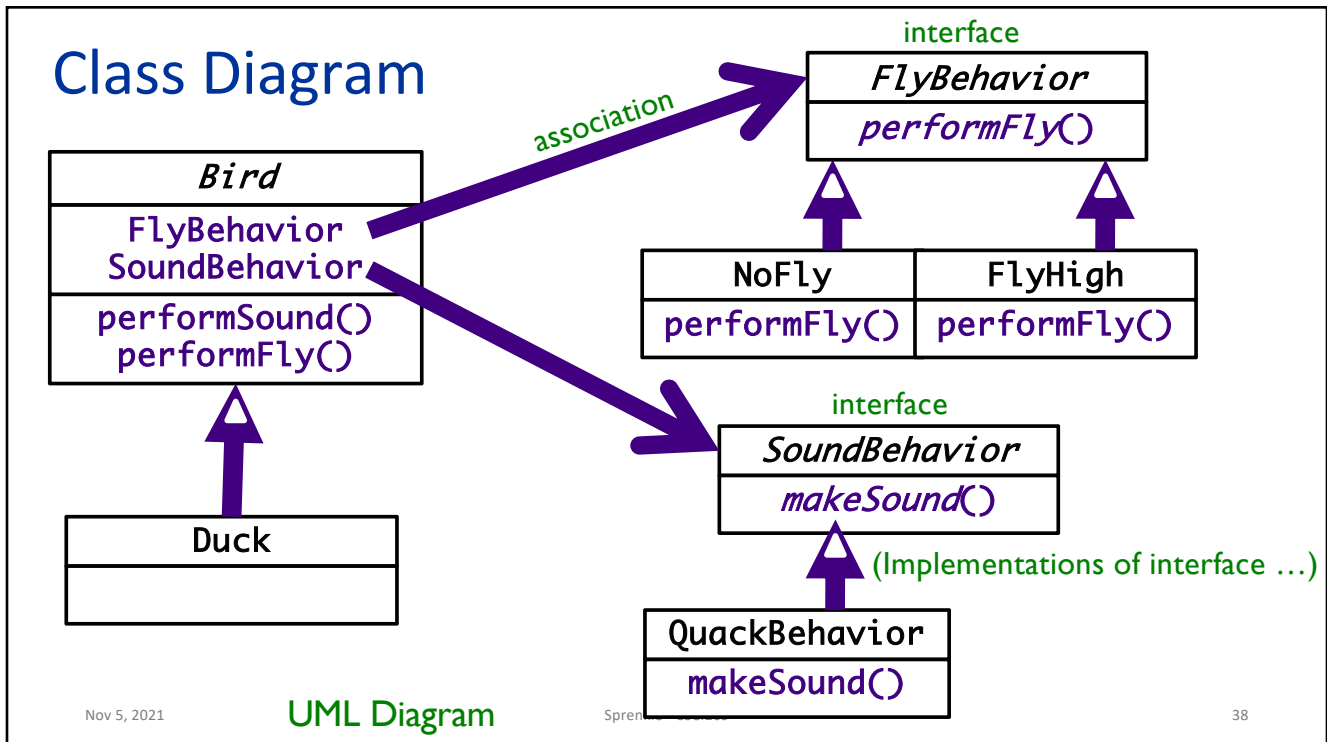
Do we need to do anything else to *this* class, with respect to fly and sound behavior?

Nov 5, 2021

Sprenkle - CSCI209

37

37



38

Unified Modeling Language (UML)

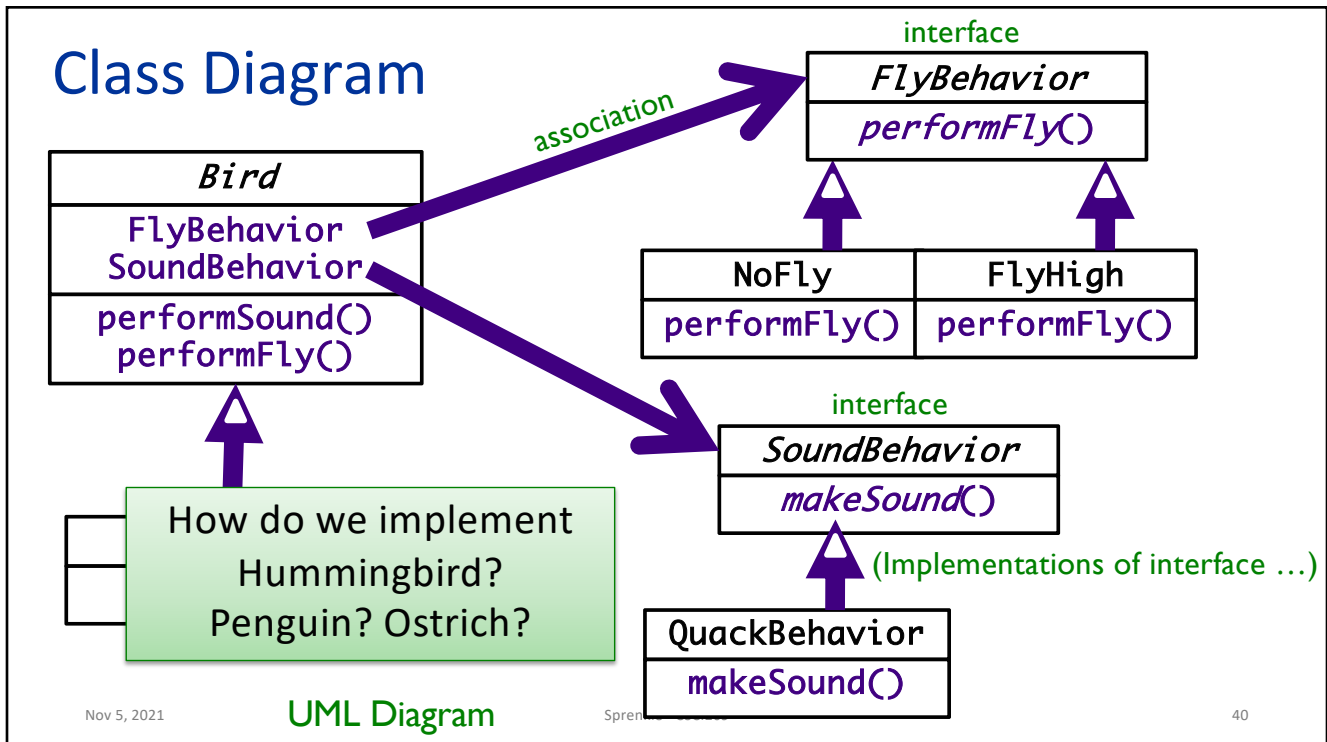
- Standardized general-purpose modeling language
 - Graphical language for visualizing, specifying and constructing the artifacts of a software system
- Includes a set of graphical notation techniques to create abstract models of specific systems
- Used in designing a large system
 - Focus on big picture, not the code

Nov 5, 2021

Sprenkle - CSCI209

39

39



40

Looking Ahead

- Assignment 7: Due Thursday
- Exam: next Fri - Sun

Nov 5, 2021

Sprenkle - CSCI209

41

41