Code Smells

Slides 15
CMPT 213

Ugh, I hate reading your code.
I know, I know.

It’s like you ran OCR on a photo of a Scrabble board from a game where JavaScript reserved words counted for triple points.

It looks like someone transcribed a naval weather forecast while woodpeckers hammered their shift keys, then randomly indented it.

It’s like an E E Cummings poem written using only the usernames a website suggests when the one you want is taken.

This looks like the output of a Markov bot that’s been fed bus timetables from a city where the buses crash constantly.
Whatever, it runs fine for now.
So does a burning bus.

https://xkcd.com/1695/
1) What’s wrong with this code?

```java
public class AbsBrakeController {
    private static final double EXTRA_BRAKING = 20;
    private double brakePercentage;

    public AbsBrakeController(double brakePercentage) {
        if (brakePercentage < 0 || brakePercentage > 100) {
            throw new IllegalArgumentException();
        }
        this.brakePercentage = brakePercentage;
    }

    public void brakeHarder() {
        if (brakePercentage < 0 || brakePercentage > 100) {
            throw new IllegalStateException();
        }
        brakePercentage += EXTRA_BRAKING;
        if (brakePercentage > 100) {
            brakePercentage = 100;
        }
    }
}
```
DRY

- [Fowler, Beck 1999]

- DRY: Don’t Repeat Yourself
  - 1 Copy of some code:
  - 2 Copies of some code:
  - 3 Copies of some code:

- What was the problem (code on previous slide)?
  - Duplicate code inside one class.

- Solution
  - REFACTOR:
    Each idea should be found in one place.
2) Refactored; What is the problem still?

```java
public class AbsBrakeController {
    private static final double EXTRA_BRAKING = 20;
    private double brakePercentage;

    public AbsBrakeController(double brakePercentage) {
        if (isBrakePercentageOk(brakePercentage)) {
            throw new IllegalArgumentException();
        }
        this.brakePercentage = brakePercentage;
    }

    public void brakeHarder() {
        if (!isBrakePercentageOk(brakePercentage)) {
            throw new IllegalStateException();
        }
        brakePercentage += EXTRA_BRAKING;
        if (brakePercentage > 100) {
            brakePercentage = 100;
        }
    }

    private boolean isBrakePercentageOk(double brakePercentage) {
        return brakePercentage >= 0 && brakePercentage <= 100;
    }
}
```
DRY Values

- What is still the problem?
  - Duplicate values in code

- Solution
  - REFACTOR:
public class AbsBrakeController {
    private static final double EXTRA_BRAKING = 20;
    private static final double MAX = 100;
    private static final double MIN = 0;
    private double brakePercentage;

    public AbsBrakeController(double brakePercentage) {
        if (isBrakePercentageOk(brakePercentage)) {
            throw new IllegalArgumentException();
        }
        this.brakePercentage = brakePercentage;
    }

    public void brakeHarder() {
        if (!isBrakePercentageOk(brakePercentage)) {
            throw new IllegalStateException();
        }
        brakePercentage += EXTRA_BRAKING;
        if (brakePercentage > MAX) {
            brakePercentage = MAX;
        }
    }

    private boolean isBrakePercentageOk(double brakePercentage) {
        return brakePercentage >= MIN && brakePercentage <= MAX;
    }
}
3) What’s wrong with this code?

```java
public abstract class Shape {
    private char border;

    public void setBorderChar(char ch) {
        border = ch;
    }

    public char getBorderChar() {
        return border;
    }
}

public class Rectangle extends Shape {
    private int x, y, width, height;

    Rectangle(int x, int y, int width, int height) {
    }

    public int getX() {
    }
    public int getY() {
    }
    public int getWidth() {
    }
    public int getHeight() {
    }
}

public class Circle extends Shape {
    private int x, y, radius;

    public Circle(int x, int y, int radius) {
    }

    public int getRadius() {
    }
    public int getX() {
    }
    public int getY() {
    }
}
```
DRY

• What is the problem?
  – ..

• Solution
  – REFACTOR:
    ..
Pull-Up x and y

```java
public abstract class Shape {
    private char border;
    private int x, y;
    public Shape(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void setBorderChar(char ch) {...}
    public char getBorderChar() {..}
    public int getX() {...}
    public int getY() {...}
}

public class Rectangle extends Shape {
    private int width, height;

    public Rectangle(int x, int y, int width, int height) {
        super(x, y);
        ...
    }

    public int getWidth() {...}
    public int getHeight() {...}
}

public class Circle extends Shape {
    private int radius;

    public Circle(int x, int y, int radius) {
        super(x, y);
        ...
    }

    public int getRadius() {...}
}
```
Template Method
Design Pattern
4) What is wrong with this code?

```java
class IntFileSum {
    int sumUpNumbers(
        File file)
    {
        try (FileReader r =
            new FileReader(file))
        {
            Scanner s = new Scanner(r);

            int sum = 0;
            while (s.hasNextInt()) {
                sum += s.nextInt();
            }
            return sum;
        } catch (IOException e) {
            e.printStackTrace();
        }
        return 0;
    }
}

class IntFileProduct {
    int multiplyUpNumbers(
        File file)
    {
        try (FileReader r =
            new FileReader(file))
        {
            Scanner s = new Scanner(r);

            int product = 1;
            while (s.hasNextInt()) {
                product *= s.nextInt();
            }
            return product;
        } catch (IOException e) {
            e.printStackTrace();
        }
        return 0;
    }
}
```

IntFileSum.java  IntFileProduct.java
What is the problem?
  - ..
    parts of function differs between classes.

Solution
  - If code was identical, just:
    • pull-up to a base class or
    • extract into a function for another class
  - If code differs:
    • REFACTOR:
      ..
Apply Template Method

```java
abstract class IntFileProcessor {
    int processFile(File file) {
        try (FileReader r =
            new FileReader(file)) {
            Scanner s = new Scanner(r);
            int result = getStartVal();
            while (s.hasNextInt()) {
                result = processInt(result, s.nextInt());
            }
            return result;
        } catch (IOException e) {
            e.printStackTrace();
        }
        return 0;
    }

    abstract protected int getStartVal();
    abstract protected int processInt(int cur, int next);
}

class IntFileProcessorSum
    extends IntFileProcessor {
    @Override
    protected int getStartVal() {
        return 0;
    }

    @Override
    protected int processInt(int cur, int next) {
        return cur + next;
    }
}

class IntFileProcessorProduct
    extends IntFileProcessor {
    @Override
    protected int getStartVal() {
        return 1;
    }

    @Override
    protected int processInt(int cur, int next) {
        return cur * next;
    }
}
```
Template Method Design Pattern

- Template Method Design Pattern:
  
  - Template Method:
    Base algorithm implementation

  - Primitive Operation:
    Delegated methods; usually abstract

  - Derived classes override primitive operations to fill in blanks of the template method.
Template Method Design Pattern (UML)

This Example

Generic Pattern
4) What is wrong with this code?

```java
class GenerateStringOfEven {
    public String getNumbers(int max) {
        String answer = "";
        for (int i = 0; i <= max; i++) {
            if (i % 2 == 0) {
                answer += i;
            }
        }
        return answer;
    }
}
class GenerateStringOfOdd {
    public String getNumbers(int max) {
        String answer = "";
        for (int i = 0; i <= max; i++) {
            if (i % 2 == 1) {
                answer += i;
            }
        }
        return answer;
    }
}
class GenerateStringOfAll {
    public String getNumbers(int max) {
        String answer = "";
        for (int i = 0; i <= max; i++) {
            if (true) {
                answer += i;
            }
        }
        return answer;
    }
}
```
abstract class GenerateString {
    protected abstract boolean isInSet(int i);

    public String getNumbers(int max) {
        String answer = "";
        for (int i = 0; i <= max; i++) {
            if (isInSet(i)) {
                answer += i;
            }
        }
        return answer;
    }
}

class GenerateStringOfEven extends GenerateString {
    @Override
    protected boolean isInSet(int i) {
        return i % 2 == 0;
    }
}

class GenerateStringOfOdd extends GenerateString {
    @Override
    protected boolean isInSet(int i) {
        return i % 2 == 1;
    }
}

class GenerateStringOfAll extends GenerateString {
    @Override
    protected boolean isInSet(int i) {
        return true;
    }
}
• Use the GenerateString base class to print out all numbers between 0 and 100 which are multiples of 5
  – Create an anonymous class inside your function

```java
abstract class GenerateString {
    protected abstract boolean isInSet(int i);
    public String getNumbers(int max) {...}
}

void clientCode() {
    GenerateString gen = new GenerateString() {
        @Override
        protected boolean isInSet(int i) {
            return i % 5 == 0;
        }
    };

    System.out.println(gen.getNumbers(100));
}
```
The Great Pattern Smack-down!

Template Method
vs
Strategy Pattern
Solving Similar Problems

• You have an algorithm which is use in multiple places with only minor differences.

• Solution Contenders
  – Template Method Pattern says, use me!
    • Put the algorithm in the base class which ..
    • Put differences in.. (override the primitive operations).
  – Strategy Pattern says, use me!
    • Put the algorithm in a class which ..
    • Put the differences in classes which ..
Ex: Template Method

```java
abstract class GenerateString1 {
    protected abstract boolean isInSet(int i);

    public String getNumbers(int max) {
        String answer = "";
        for (int i = 0; i <= max; i++) {
            if (isInSet(i)) {
                answer += i;
            }
        }
        return answer;
    }
}

void clientCode() {
    GenerateString1 gen = new GenerateString1() {
        @Override
        protected boolean isInSet(int i) {
            return i % 5 == 0;
        }
    };
    String result = gen.getNumbers(100);
}
```
Ex: Strategy

```java
interface Selector {
    boolean isInSet(int i);
}

class GenerateString2 {
    public String getNumbers(int max, Selector selector) {
        String answer = "";
        for (int i = 0; i <= max; i++) {
            if (selector.isInSet(i)) {
                answer += i;
            }
        }
        return answer;
    }
}

void clientCode() {
    GenerateString2 gen = new GenerateString2();
    Selector sel = i -> i % 5 == 0;
    String result = gen.getNumbers(100, sel);
}
```
Comparison

• Template method pattern customizes the base algorithm through inheritance.
  - Inheritance = ..
  - Great if you..
  - Ex: Shape’s draw() function uses isBorder().

• Strategy pattern customized the base algorithm through composition.
  - Composition = ..
  - Flexible for selecting different algorithm during execution.
  - Simpler client code:
    ..
Easy Code Smells
Code Smell: Long Method

• Shorter methods are more reusable, flexible, and easy to understand.

• REFACTOR:
  – Raise the level of abstraction
  – Make code shorter and easier to read.
Code Smell: Needing Comments

- Comments are deodorant: If code is unclear and needs comments to explain, it means the code smells.
- Refactor to clean up code!
  - Extract Method:
    ..
Code Smell: Needing Comments (cont)

- Break complex expression down by adding well-named variables.

```java
int numPages = 0, binderSize = 10, weight = 0;
if (numPages < binderSize && weight >= 10 && weight <= 100) {
    ...
}
```

```java
boolean isFull = numPages >= binderSize;
boolean isLight = weight < 10;
boolean isHeavy = weight > 100;
if (!isFull && !isLight && !isHeavy) {
    ...
}
```
Example

- `smells.shapes.TextBox`
  - `draw1()`: Initial algorithm
  - `draw2()`: Refactored algorithm
  - `draw3()`: Refactored with Extract class
Summary

- DRY: Don’t Repeat Yourself
  - Extract Method
  - Extract Constant
  - Pull-up to base class

- Template Method Pattern
  - Base class has algorithm & calls abstract methods
  - Derived classes override abstract methods

- Long Method: Extract method

- Needing comments:
  - Extract method
  - Introduce Explanatory Variable