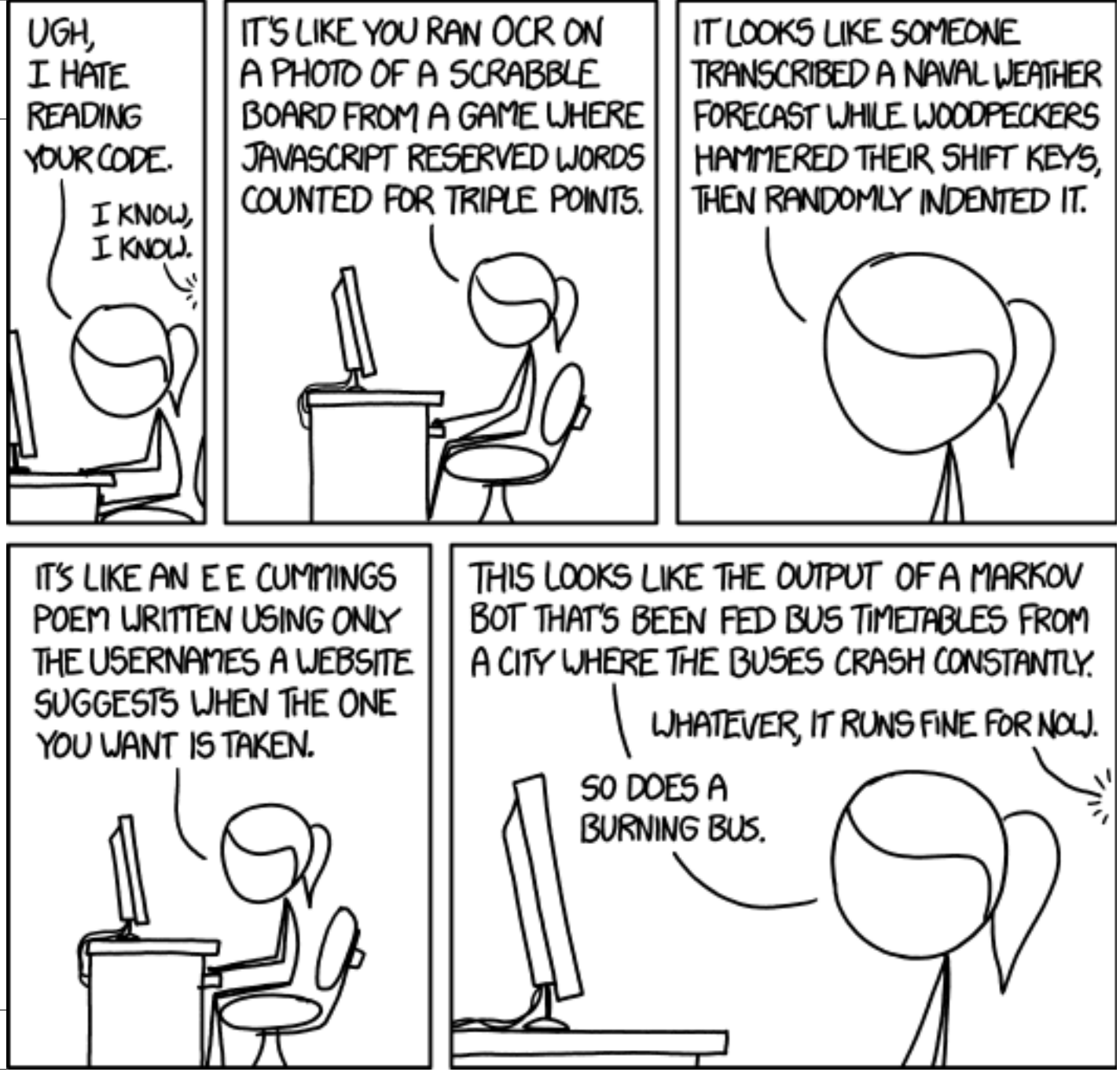
Code Smells

Slides 17
CMPT 213



https://xkcd.com/1695/

1) What's wrong with this code?

standout problem: duplicated code (two same ifs), magic numbers

```
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

- ..“number one in the stink parade is duplicate code”
  [Fowler, Beck 1999]
- DRY: Don’t Repeat Yourself
  - 1 Copy of some code:..great!
  - 2 Copies of some code:.. poor; refactor?
  - 3 Copies of some code:..bad! refactor now!
- What was the problem (code on previous slide)?
  - Duplicate code inside one class.
- Solution
  - REFACTOR:..extractor method
    Each idea should be found in one place.

2) Refactored; What is the problem still?

```
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;
    }
}
```

problem still: magic numbers

DRY Values

- What is still the problem?
  - Duplicate values in code
- Solution
  - REFACTOR:

    ..extract constant

```java
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?

```
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() {...}
}
```

duplicated code

DRY

- What is the problem?
  - ..duplicate code in sibling classes
- Solution
  - REFACTOR:

    .. pull up identical code to base class

Pull-Up x and y

```
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

how does this differ from static factory method?
-> later slides

4) What is wrong with this code?

```
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;
  }
}
```

IntFileSum.java

```
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;
  }
}
```

IntFileProduct.java

two classes almost the same! only those highlighted lines are different

DRY

- What is the problem?
  - ..classes with nearly identical functions
    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 design pattern

Apply Template Method

```
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;
  }
}
```

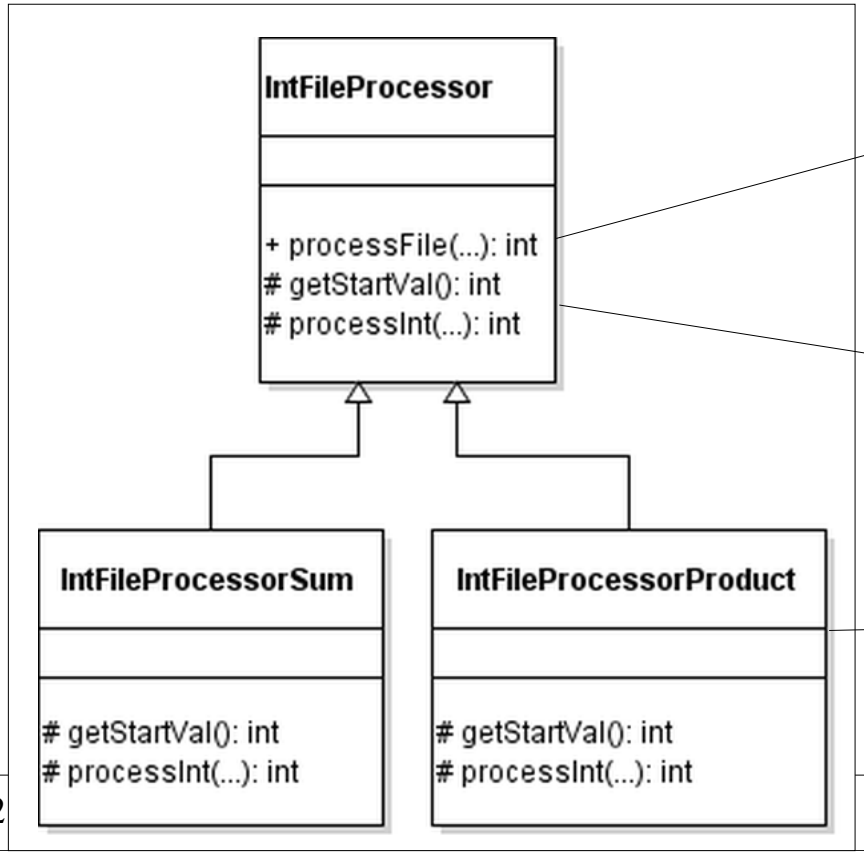
```
class IntFileProcessorProduct
      extends IntFileProcessor
{
  @Override
  protected int getStartVal() {
    return 1;
  }

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

everytime wanna do something new
-> just instantiate a new class

Template Method Design Pattern

- Template Method Design Pattern:
  - ..a template method in base class implements an algorithm and delegates some operation(s) to derived classes



Template Method:
Base algorithm implementation

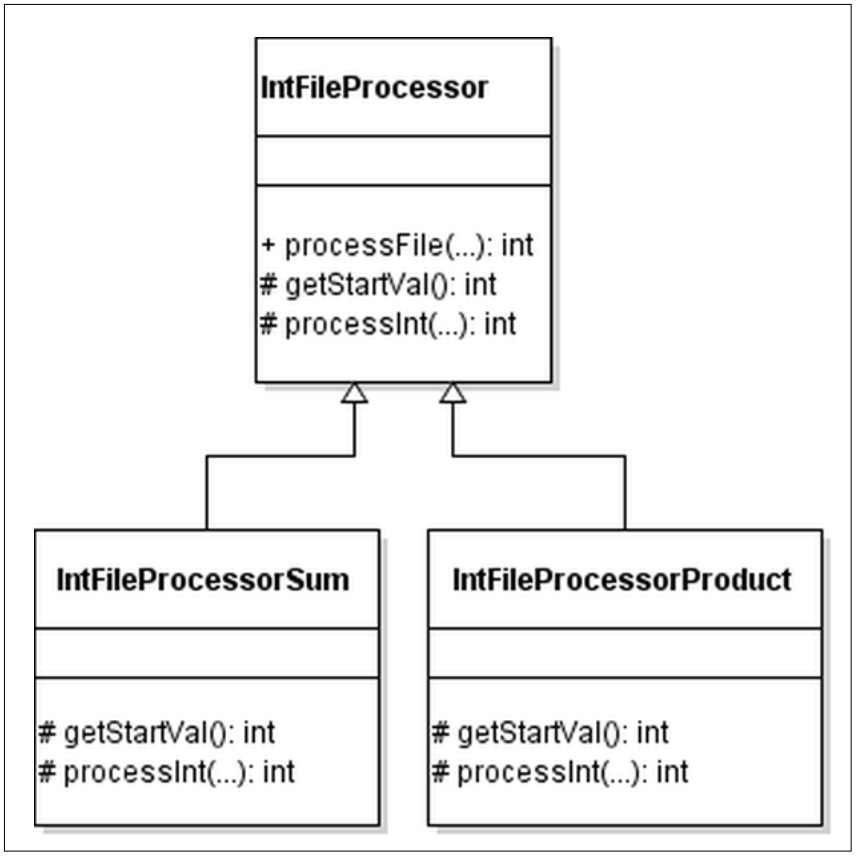
has the

Primitive Operation:
Delegated methods; usually abstract

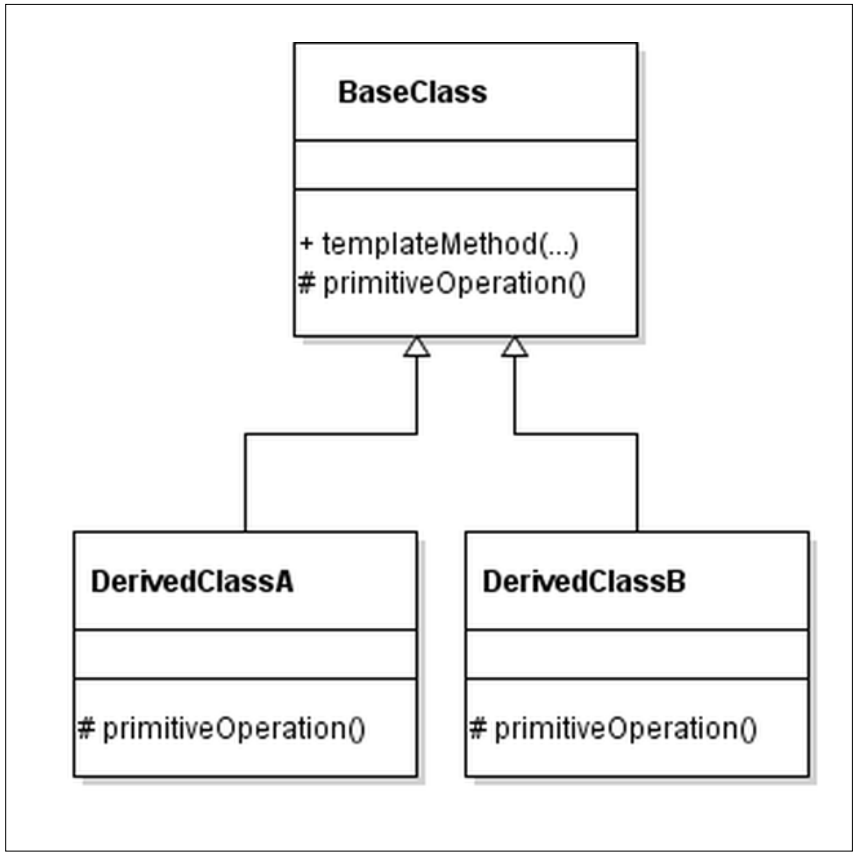
Derived classes override primitive operations to fill in blanks of the template method.

Template Method Design Pattern (UML)

filling in the hole using inheritance



This Example



Generic Pattern

4) What is wrong with this code?

```
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;
  }
}
```


this is the only difference -> code smell: duplicated code
==> use template method

Template Method Solution

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

the order is not matter: abstract protected is ok too!

final

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

any class extends from this class
need to override this function
-> template method is used to force
derived classes to override the function

```
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;
  }
}
```

Client Code Example

- 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

```
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
      .. calls abstract primitive operations
    - Put differences in..derived classes
      (override the primitive operations). to customize algorithm
  - Strategy Pattern says, use me!
    - Put the algorithm in a class which
      ..accepts a reference to an interface
    - Put the differences in classes which
      ..implement the interface

Ex: Template Method

```
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);
}
```

lambda expression wont work here

Ex: Strategy

```
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;
    }
}
```

no abstract, no inheritance, we use implementing interface, use explicit dependency here

explicitly accepting Selector object

dependency injection here with strategy pattern

```
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 =.. compile time decision, cannot have runtime inheritance
  - Great if you.. already have an inheritance hierarchy
  - Ex: Shape's draw() function uses isBorder().
- Strategy pattern customizes the base algorithm through composition.
  - Composition =.. run time decision
  - Flexible for selecting different algorithm during execution.
  - Simpler client code:
    .. no inheritance, use lambda functions if you have one method in the interface

Template Method : algorithm structure remains unchanged but certain steps vary
Strategy : multiple algorithms for a task and want to switch between them easily
-> strategy is more flexible at runtime

Easy Code Smells

Code Smell: Long Method

- Shorter methods are more reusable, flexible, and easy to understand.
- REFACTOR:

  .. extract method
  - 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:

    .. change comments into functions

Code Smell: Needing Comments (cont)

- .. introduce explanatory variable

  Break complex expression down by adding well-named variables.

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

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

Would this be better as:
if(hasRoom && isOkWeight)?

smells:
+the if has lots of ands ors
+not enough on 1 line
-> breaks it!

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