Refactoring Fundamentals

Code Smells: Object Orientation Abusers

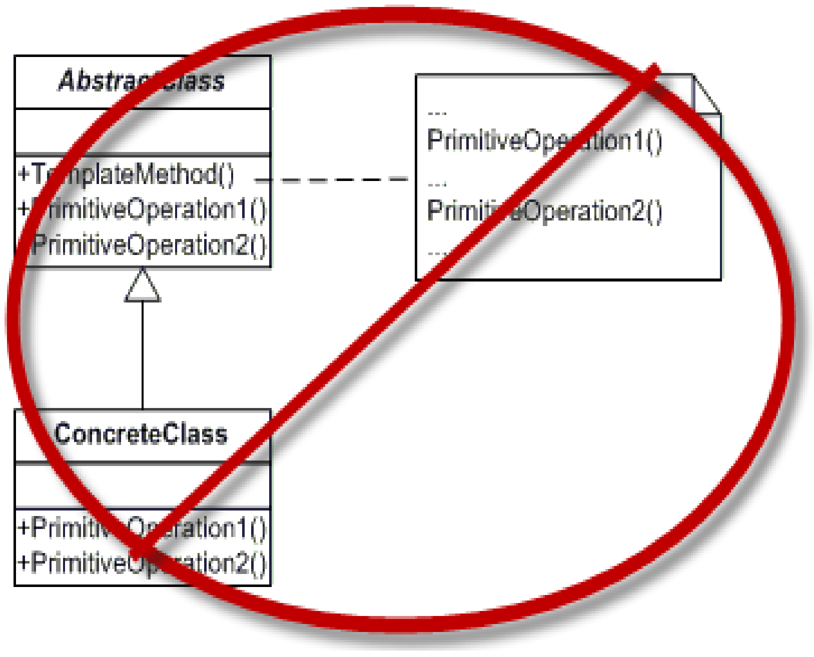
Steve Smith
Ardalis.com
@ardalis

In This Course

- ~~What is Refactoring?~~
- ~~Why do it?~~
- ~~What's the process?~~
- ~~What are some tools that can assist with it?~~
- ~~What is a *Code Smell*?~~
- What are some examples of Code Smells?
- What are some common refactorings?
- How does one apply them correctly?

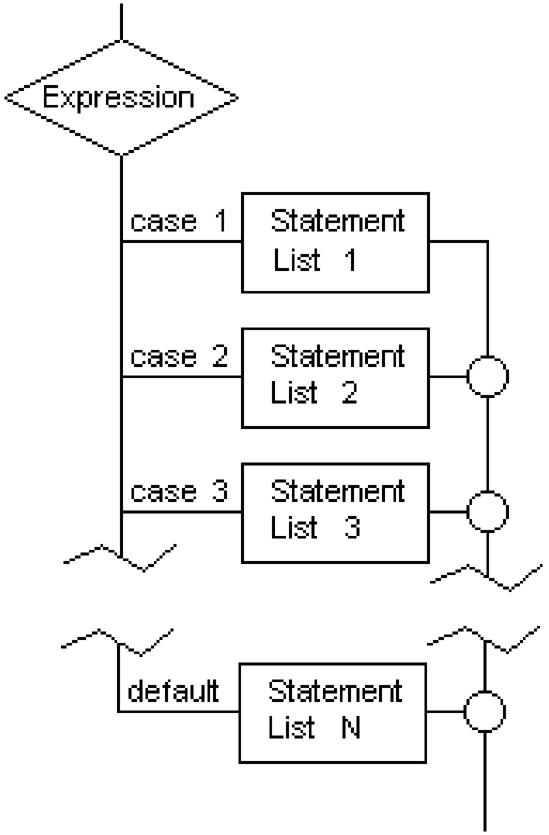
Code Smells: The Object Orientation Abusers

- Coding constructs or techniques that go against the principles and patterns of object oriented design
- Break polymorphism
- Introduce Repetition
- Create Tight Coupling



OO Abusers: Switch Statements

- **See also if-else if-else if-else if…**
- **Main symptom of this smell is duplication**
  - One switch is OK
  - Two or more may warrant attention
- **May indicate lack of encapsulation**
- **Polymorphism can effectively deal with switch statement duplication**
  - Need to move behavior into class with flag



Switch Statements

```
MethodOne(Class class)
{
   switch (class.TypeId)
    {
       case 1:
        case 2:
        case n:
   }
}

AnotherMethod(Class class)
{
 switch (class.TypeId)
    {
        case 1:
        case 2:
        case n:
   }
}
```

OO Abusers: Temporary Field

- A temporary field is an instance variable that is only set in certain circumstances
- Creates confusion and bugs
- Often used to pass state temporarily between methods, to avoid having to pass many parameters

Common Refactorings

- Introduce Null Object
- Extract Class
  - Parameter Object
  - Method Object

http://commons.wikimedia.org/wiki/File:Temporary_plate.svg

Temporary Field

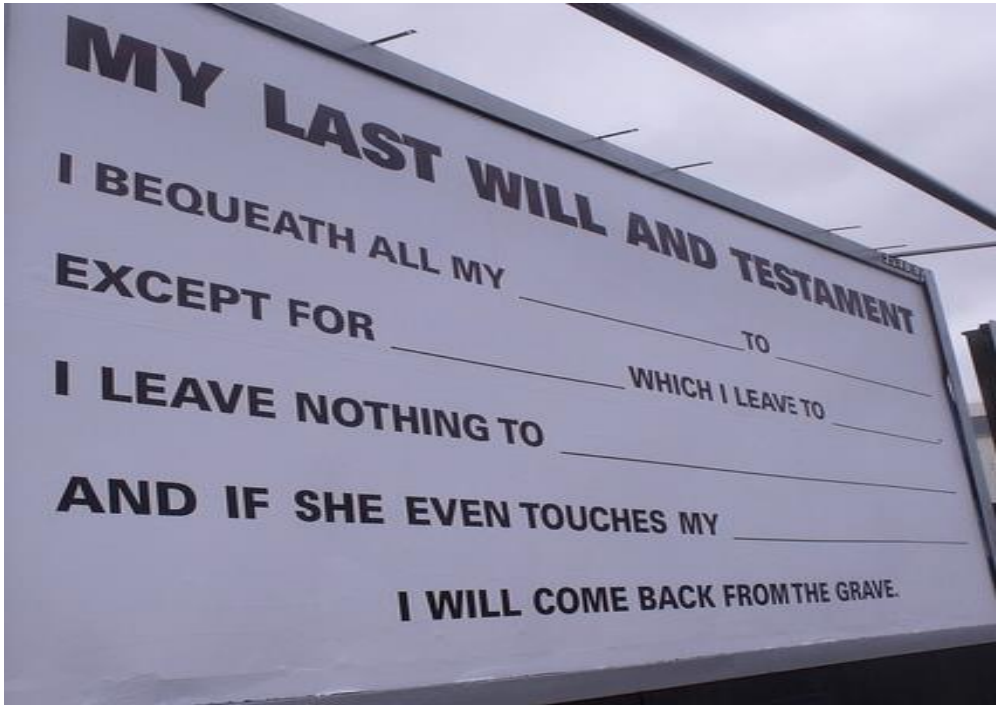
```
class Employee
{
  private decimal _earningsForBonus;

// other fields and methods

  private decimal CalculateBonus()
  {
    return _earningsForBonus * BonusPercentage();
  }

  private void CalculateEarningsForBonus()
  {
      _earningsForBonus = YearToDateEarnings() + OvertimeEarnings() * 2;
  }

}
```

Temporary Field (Refactored)

```
class BonusCalculator
{
  private decimal _earningsForBonus;
  private decimal _bonusPercentage;
  public BonusCalculator(Employee employee)
  {
    CalculateEarningsForBonus(employee.YearToDateEarnings(),
        employee.OvertimeEarnings());
    _bonusPercentage = employee.BonusPercentage();
  }
  public decimal CalculateBonus()
  {
    return _earningsForBonus * _bonusPercentage;
  }
  private void CalculateEarningsForBonus(decimal ytdEarnings, decimal
   ytdOvertimeEarnings)
  {
     _earningsForBonus = ytdEarnings + ytdOvertimeEarnings * 2;
  }
}
```

OO Abusers: Refused Bequest

- Subclasses can use all of their parents' methods and data
- When they refuse or ignore these "gifts," it's a smell
- A faint smell for
  - Implementation details
  - Private data and fields
- A smell worth fixing for
  - Refusing public interface


MY LAST WILL AND TESTAMENT
I BEQUEATH ALL MY ______ TO ______
EXCEPT FOR ______ WHICH I LEAVE TO ______
I LEAVE NOTHING TO ______
AND IF SHE EVEN TOUCHES MY ______
I WILL COME BACK FROM THE GRAVE.

Common Refactorings

- Push Down Method
- Push Down Field
- Replace Inheritance with Delegation

http://flic.kr/p/bwKi8U

OO Abusers: Alternative Classes with Different Interfaces

- **Individual methods that do the same thing should use the same interface**

- **Avoid:**
  - ClassOne.Add()
  - ClassTwo.Insert()
  - ClassThree.MakeAnother()

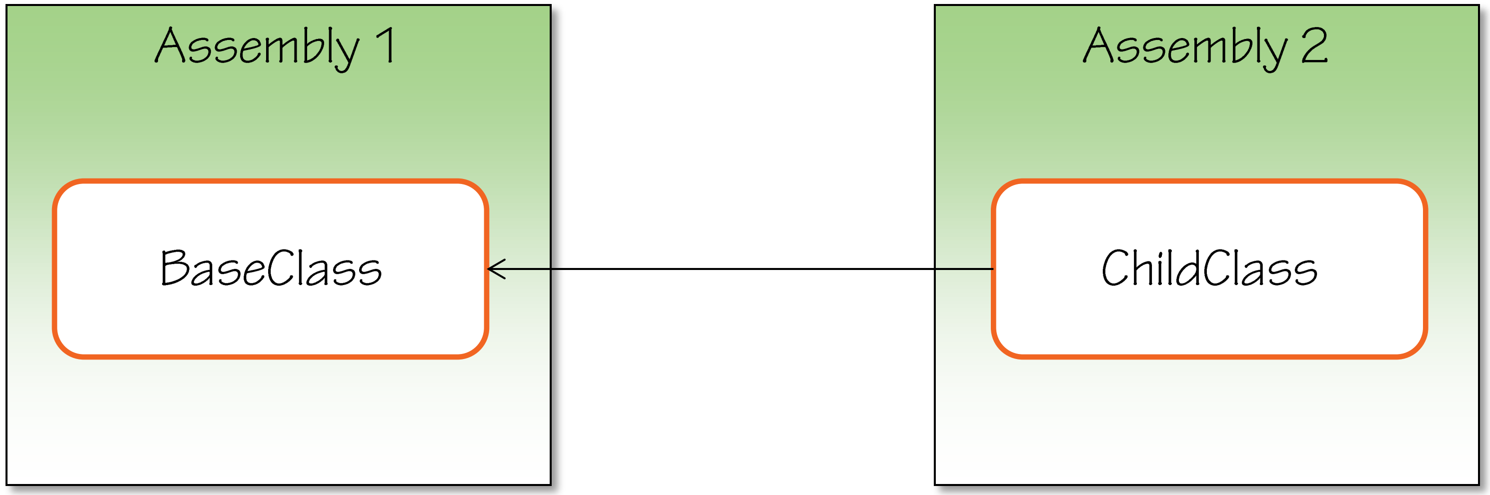
Refactor

- **Rename Method**
- **Move Method**
- **Extract Superclass**

http://commons.wikimedia.org/wiki/File:USB_types_2.jpg

OO Abusers: Base Class Depends on Subclass

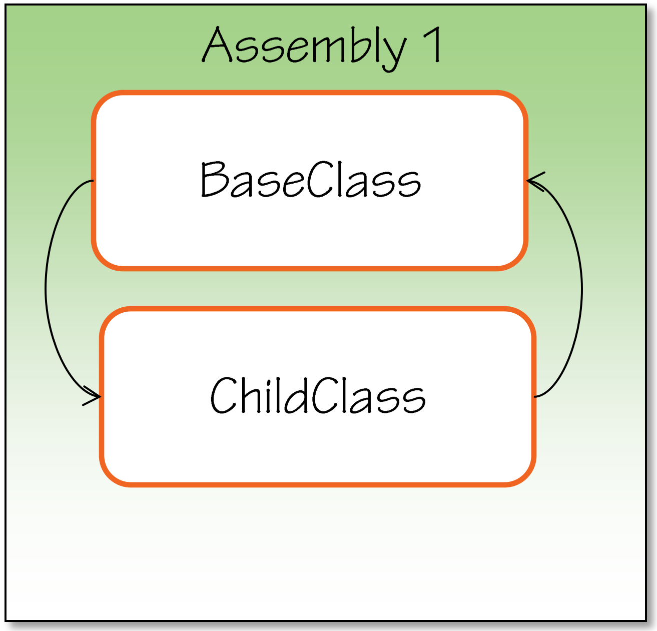
- Proper use of inheritance allows child classes to be packaged independently from base classes



- Child classes are always dependent on base classes

OO Abusers: Base Class Depends on Subclass

- Having dependencies on child classes within base classes forces all such classes to be deployed together



Refactor

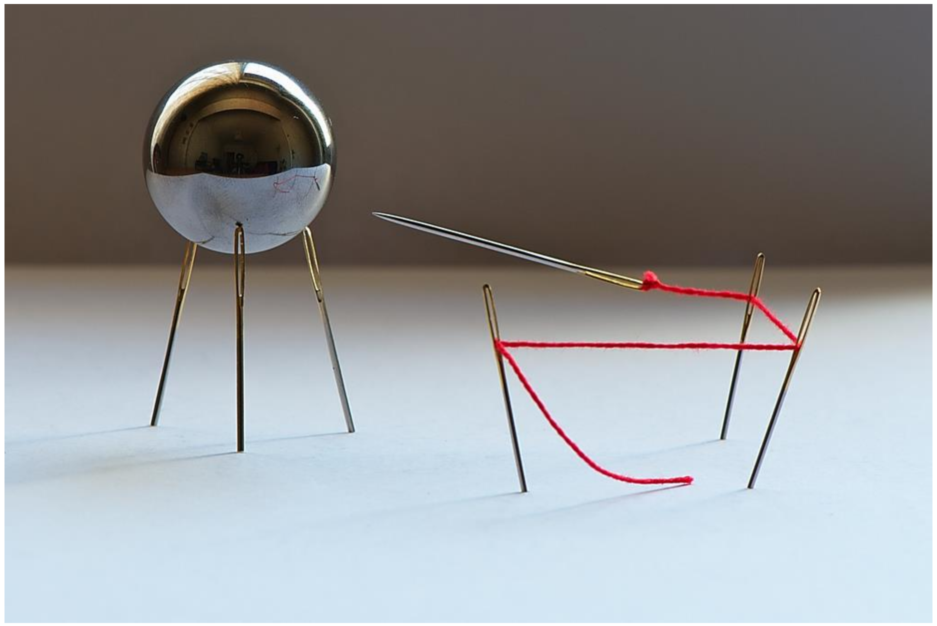
- Push method down
- Implement Factory
- Use reflection

OO Abusers: Inappropriate Static

- **Inheritance provides huge value in OO systems**

- **Reserve static for:**
  - Stateless operations
  - Behavior will never change

- **Examples:**
  - Simple math operations
  - Global constants

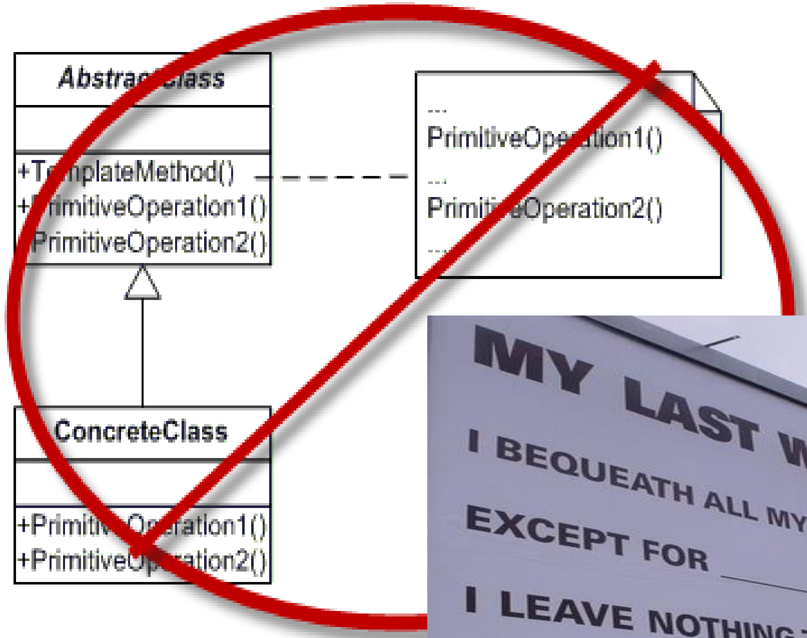


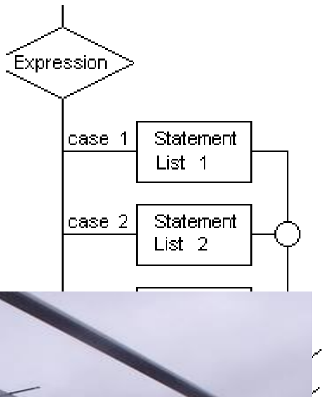
Refactor

- **Move method**
- **Replace static variable with parameter**

http://flic.kr/p/6HAzD1

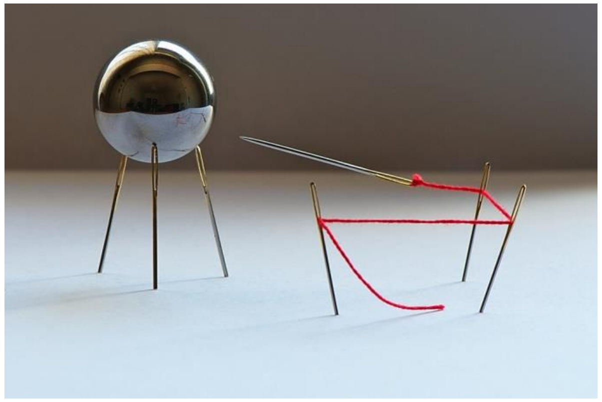
Summary

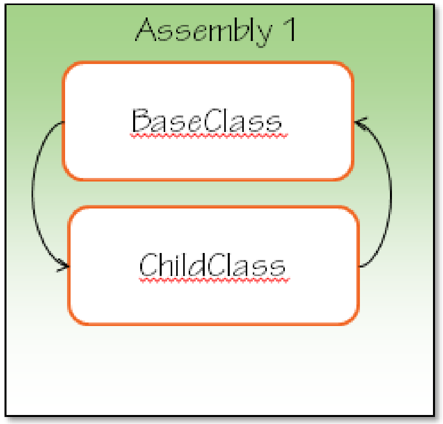

AbstractClass
+TemplateMethod()
+PrimitiveOperation1()
+PrimitiveOperation2()
...
PrimitiveOperation1()
...
PrimitiveOperation2()
...
ConcreteClass
+PrimitiveOperation1()
+PrimitiveOperation2()


Expression
case 1
Statement List 1
case 2
Statement List 2

TEMPORARY

MY LAST WILL AND TESTAMENT
I BEQUEATH ALL MY ____ TO ____
EXCEPT FOR ____ WHICH I LEAVE TO ____
I LEAVE NOTHING TO ____
AND IF SHE EVEN TOUCHES MY ____
I WILL COME BACK FROM THE GRAVE.




Assembly 1
BaseClass
ChildClass

References

Related Pluralsight Courses

SOLID Principles of Object Oriented Design http://bit.ly/rKbR9a

Design Patterns Library http://bit.ly/SJmAX1

Books

Code Complete http://amzn.to/Vq5YLv

Clean Code http://amzn.to/YjUDI0

Thanks!

Steve Smith

Ardalis.com

Twitter: @ardalis

pluralsight
hardcore developer training

Guest Opinion: Scott Hanselman