Satisfying the Open Closed Principle

Never write an IF again

…unless you really need to

…but seriously, don't

Who Am I?

- Dustin Williams
- dwilliams@manifestcorp.com
- Consultant @ Manifest Solutions
- Instructor @ Columbus State
- @williadd

Unrelated Fact

StatusResultMatchers.isIAmATeapot

public ResultMatcher isIAmATeapot()

Assert the response status code is HttpStatus.I_AM_A_TEAPOT (418).

418 I'm a teapot (RFC 2324)

This code was defined in 1998 as one of the traditional IETF April Fools' jokes, in RFC 2324, Hyper Text Coffee Pot Control Protocol, and is not expected to be implemented by actual HTTP servers. The RFC specifies this code should be returned by tea pots requested to brew coffee.

Why this topic?

- Java CoE @ Disney, 2009
  - Just learning to review code objectively
  - Couldn't articulate the WHY
  - First heard of SOLID
- Delivery Oversight @ Manifest Solutions
  - Consultants lowering their standards
  - Not sure what to look for when refactoring

What's in it for you?

Observing the Open/Closed principle helps eliminate risk and contain costs

(LOUD APPLAUSE FROM PROJECT MANAGERS)

You can avoid

- Changes to code that is "done"
- Constructs that increase cyclomatic complexity
- Increasing costs for the next new feature

It was a dark and stormy night…

In a small dark room, alone, possibly scared:

"I think all I need to do is change this section of code. Adding another else should be all I need. I think I'll go to bed."

The next day:

"HEY, you broke the build!!!"

DUH-DUH-DUH

You probably just violated the open-closed principle

- Single Responsibility
- **Open Closed**
- Liskov Substitution
- Interface Segregation
- Dependency Injection

Definitions

- From Bertrand Meyer in the late 1980s: A module is open if it is still available for extension, closed if it is available for use by other modules.
- Solutions rely on implementation inheritance.

Definitions

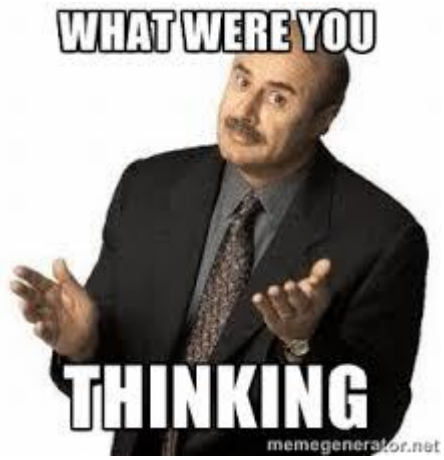
- Concept updated in the mid 1990s
- Evolved to refer to abstract base classes, interface specifications, and encapsulation
- Newer, generally accepted definition: A software entity should be open for extension, but closed for modification

Why is it Important?

- Helps maintain a constant, rather than increasing, cost of ownership
- Feel more secure that a change will not have unintended side effects
- Helps us adhere to the other SOLID principles
- Suggests that an application and its features can be bundled separately

What Does the World Look Like?

- Case statements everywhere
- Ifs on every line
- Constantly retesting the same portions of code because they are always changing
- Proliferation of methods to act on specific types / private methods
- Control parameters
- Cats and dogs living together, mass hysteria


WHAT WERE YOU
THINKING
memegenerator.net

Violation – Proliferation of IF

```
public void handle(int statusCode) {
    if(400 == statusCode) {
    ...
    } else if(404 == statusCode) {
    ...
    }
    ...
    } else {
    ...
    }
}
```

Violation – Orchestration

```
public void workflow() {
    actionA();
    actionB();
    for(int i = 0; i < 10; i++) {
        actionC();
    }
    actionD();
}
```

Violation – Business Rules

```
public double applyTax(org, loc, total) {
    if(org.isTaxExempt())
        return 0d;
    } else{
        if(loc.hasLocalTax()){
        ...
        }else if(loc.hasStateTax()){
        ...
        }
    }
}
```

Violation – Control Parameter

```
public void someMethod(boolean someDecision) {
    if(someDecision) {
        ...
    } else {
        ...
    }
}
```

There's No Hope

- If failure is that easy, what chance do we have?
- Apply what we know
  - Composition v. Inheritance
  - Red Green Refactor
  - Change is constant

Compliance – Dependency Injection

```
public class SomeClass {
    private Log logger = new ConsoleLogger();

    public void setLogger(Log logger) {
        this.logger = logger;
    }
}
```

Compliance - Template

```
public abstract class SomeClass {
    protected abstract void specializedAction();

    public void genericAction() {
        ...
        specializedAction();
        ...
    }
}

public class SpecialCase extends SomeClass {
    protected void specializedAction() {
        ...
    }
}
```

Compliance - Strategy

```
interface FeeStrategy {
    boolean canHandle(Trip trip);
    double calculate(Trip trip);
}

class AirportToDowntownFee implements {
    public boolean canHandle(Trip trip) {
        return trip.getorig().equals("ATL") && trip.getDest().equals("downtown")
    }
    public double calculate(Trip trip) { return 30d; }
}

class FeeCalculator {
    private List<FeeStrategy> feeStrategies;

    public double calculate(Trip trip) {
        FeeStrategy feeStrategy;
        for(FeeStrategy f : feeStrategies) {
            if (f.canHandle(trip)) {
                feeStrategy = f;
                break;
            }
        }
        return feeStrategy.calculate(trip);
    }
}
```

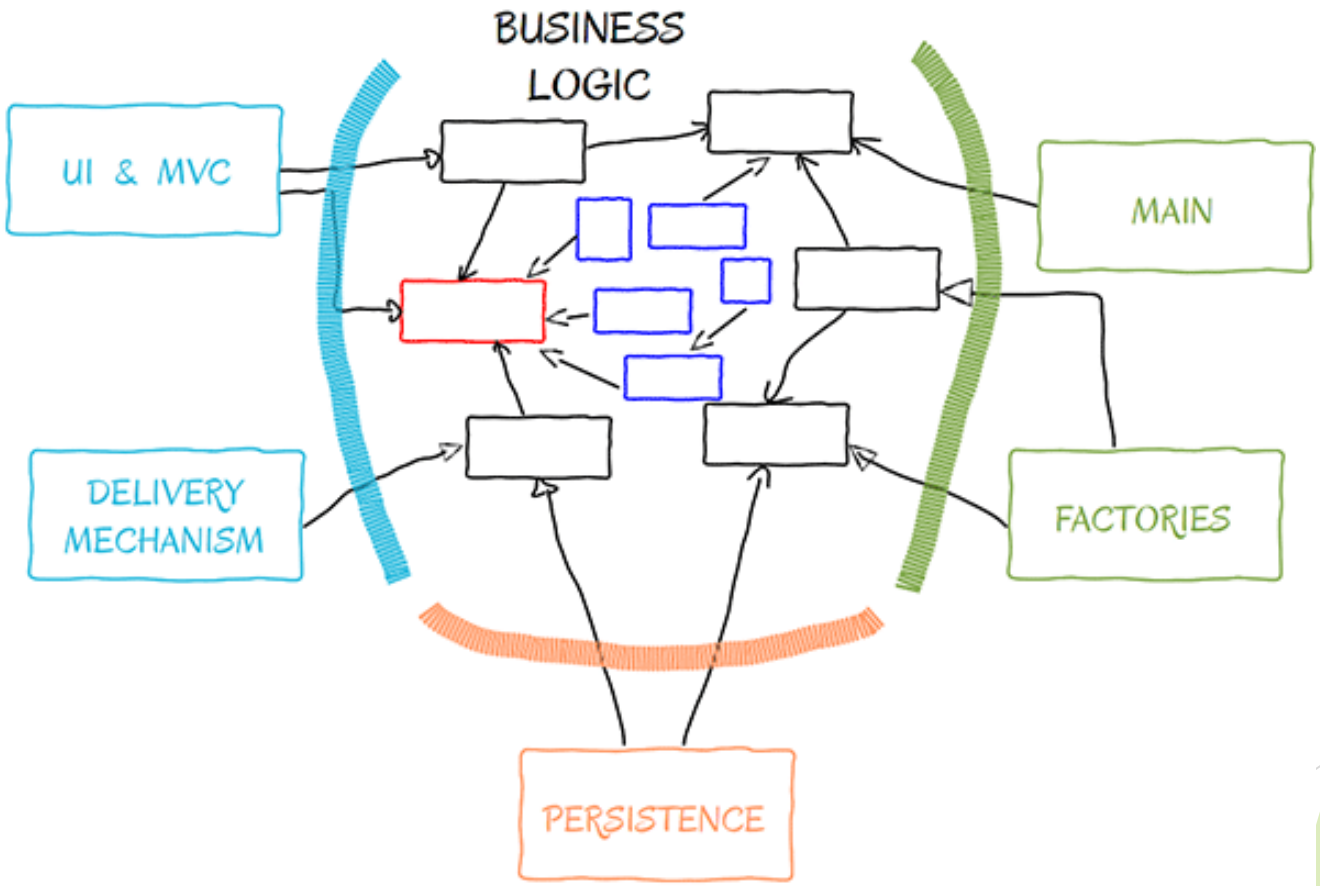
Compliance - Other

- Command
  - Knows the receiver and invokes an action on the receiver
  - More complex than the Strategy pattern
- Visitor
  - Valuable for traversing systems with many types of content
  - Uses double dispatch

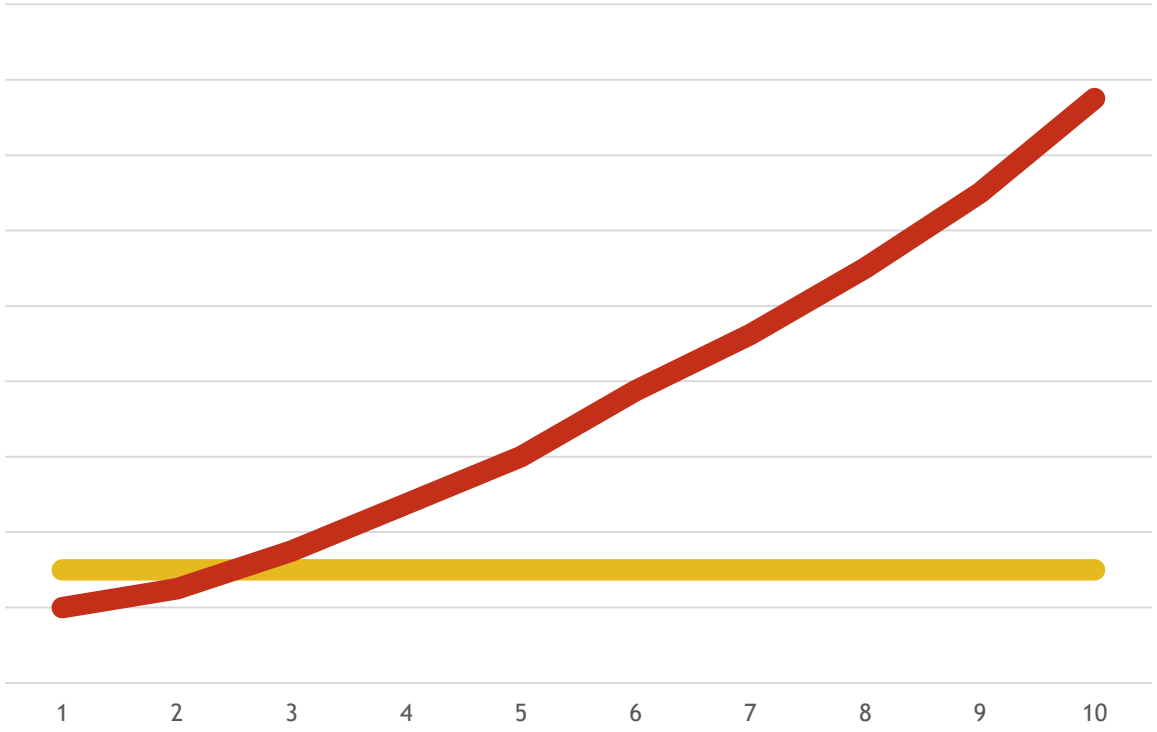
CODE

Real World Examples

Fewer Changes, Less Risk



Lower Cost of Change



What Can You Do?

- Learn to identify violations that exist, as they are occurring, and before they occur
- Make fixing violations easier with DI and SR
- Categorize violations, refactoring effort can be quantified and justified
- Don’t let perfection prevent you from trying

Credits

- Wikipedia
- http://code.tutsplus.com/tutorials/solid-part-2-the-openclosed-principle--net-36600
- https://8thlight.com/blog/uncle-bob/2014/05/12/TheOpenClosedPrinciple.html
- http://www.codeproject.com/Articles/613119/SOLID-Principles-The-Open-Closed-Principle-What-Wh
- https://lostechies.com/gabrielschenker/2009/02/13/the-open-closed-principle/