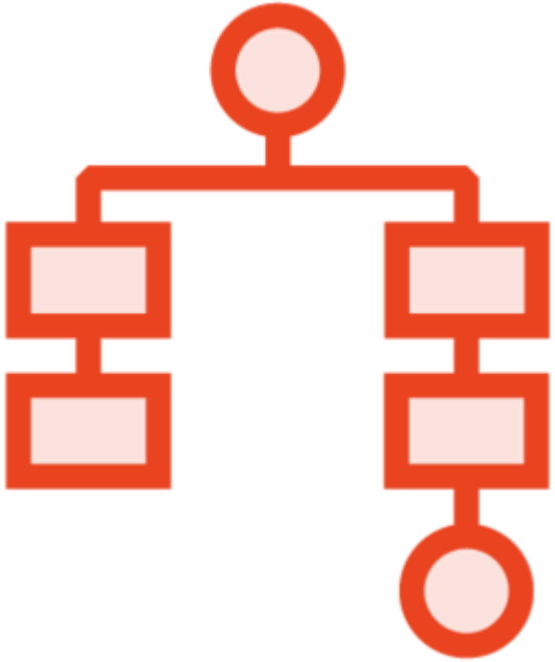
Fixing Object-oriented Abusers

Andrejs Doronins
TEST AUTOMATION ENGINEER

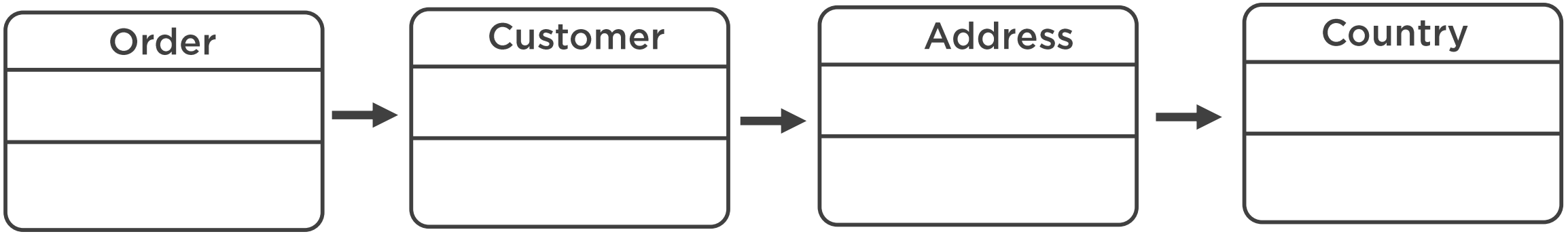
Object-oriented Abusers

Code that doesn’t follow object-oriented programming principles.

Object-oriented Abusers



- Conditional complexity
- Refused bequest
- Temporary field
- Alternative classes with different interfaces


Order
Customer
Address
Country

Conditional Complexity

Complex switch operator or a sequence of if-statements

```
class SomeClass {
    String doSomething(){
        if(someCondition){
            if(otherCondition){
            } else if(){
            }
        } else if(moreConditions){
        } else {
        }
    }
}
```

Complex Conditionals Often Mean:

- Missing domain objects
- Not using polymorphism
- Not using inheritance

Conditional Complexity Issues

Starts simple, but gradually harder to understand as logic is added

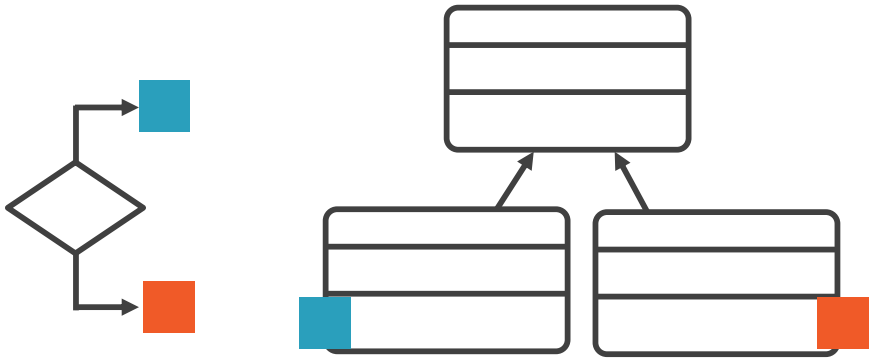
High likelihood of breaking

Breaks the Open/Closed Principle

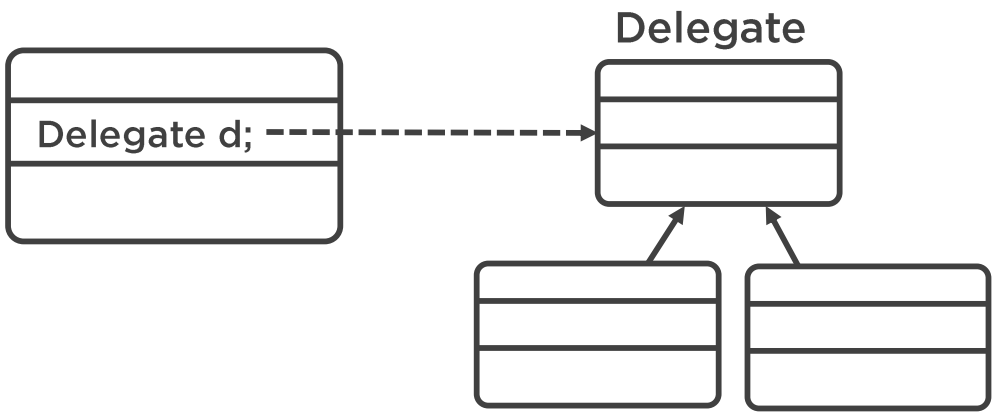
Alcohol Age Restrictions

Country	Age
US	21
EU countries	18
Canada	18 or 19

How to Fix Conditional Complexity



Replace with polymorphism

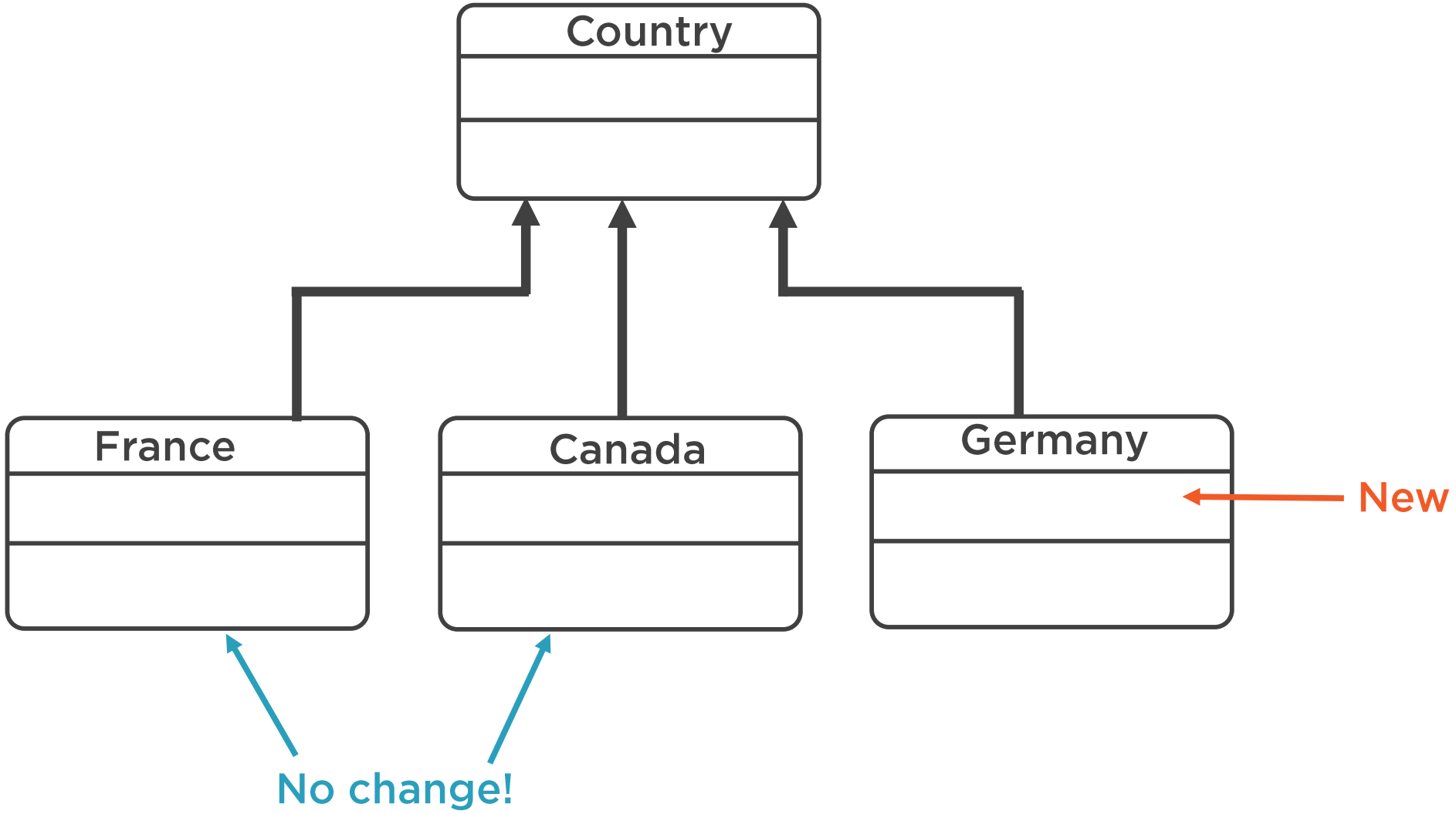


Replace with delegation

Benefits Achieved

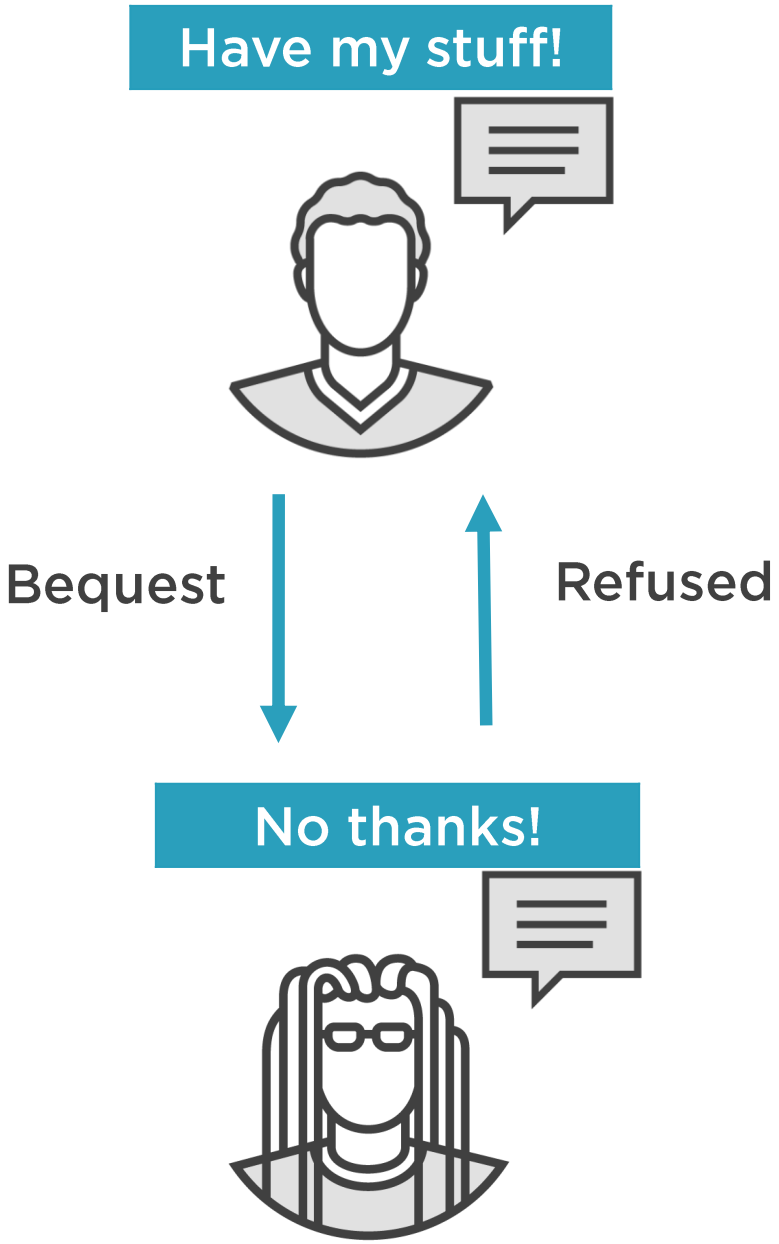
Each piece of logic encapsulated

Much lower chance of breaking existing code when adding more related logic


Country
France
Canada
Germany
New
No change!

Bequest

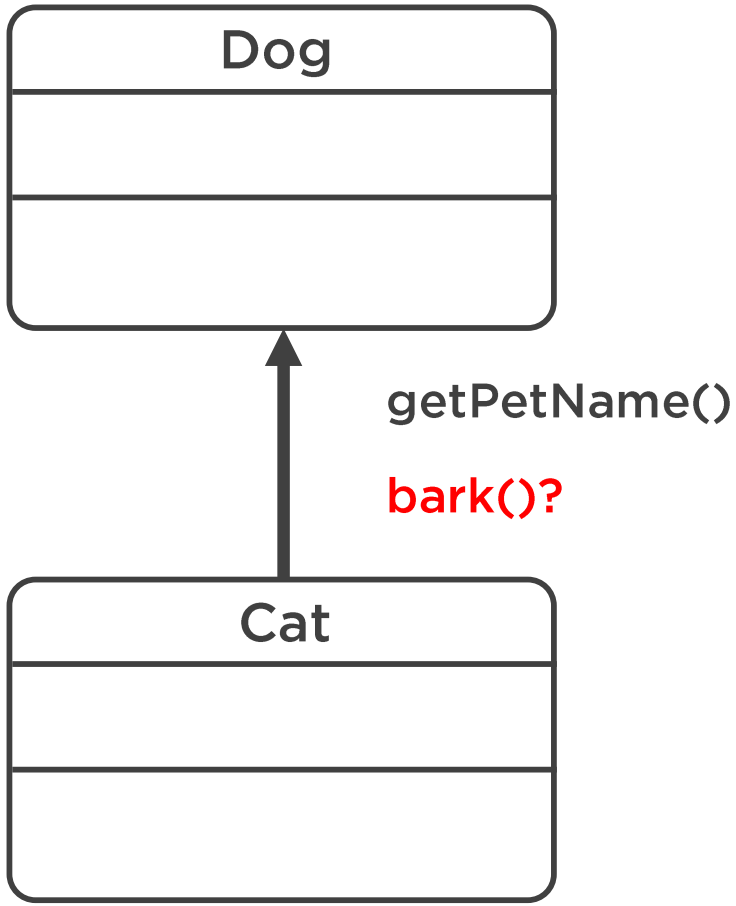
The act of giving or leaving personal property by a will

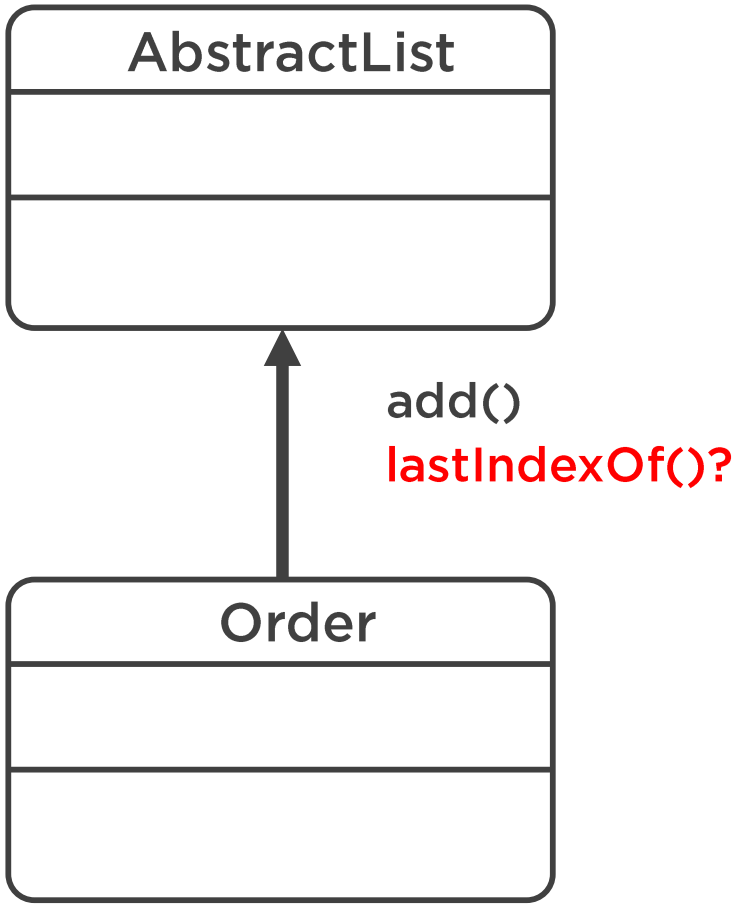

Have my stuff!
Bequest
Refused
No thanks!

Refused Bequest

Subclass inherits fields and methods it doesn’t need

Unwanted Inheritance





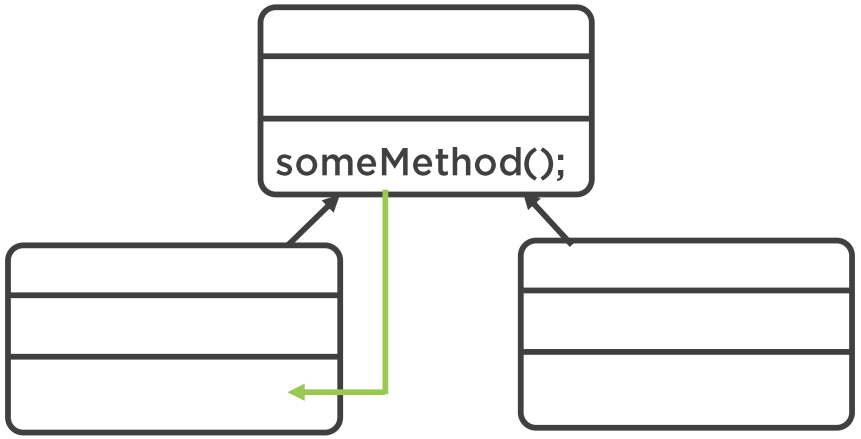
Refused Bequest Issues

Objects inherit behavior that doesn't belong to them – makes coding confusing

Leads to unexpected behavior

How to Fix Refused Bequest

methodA() → methodB()



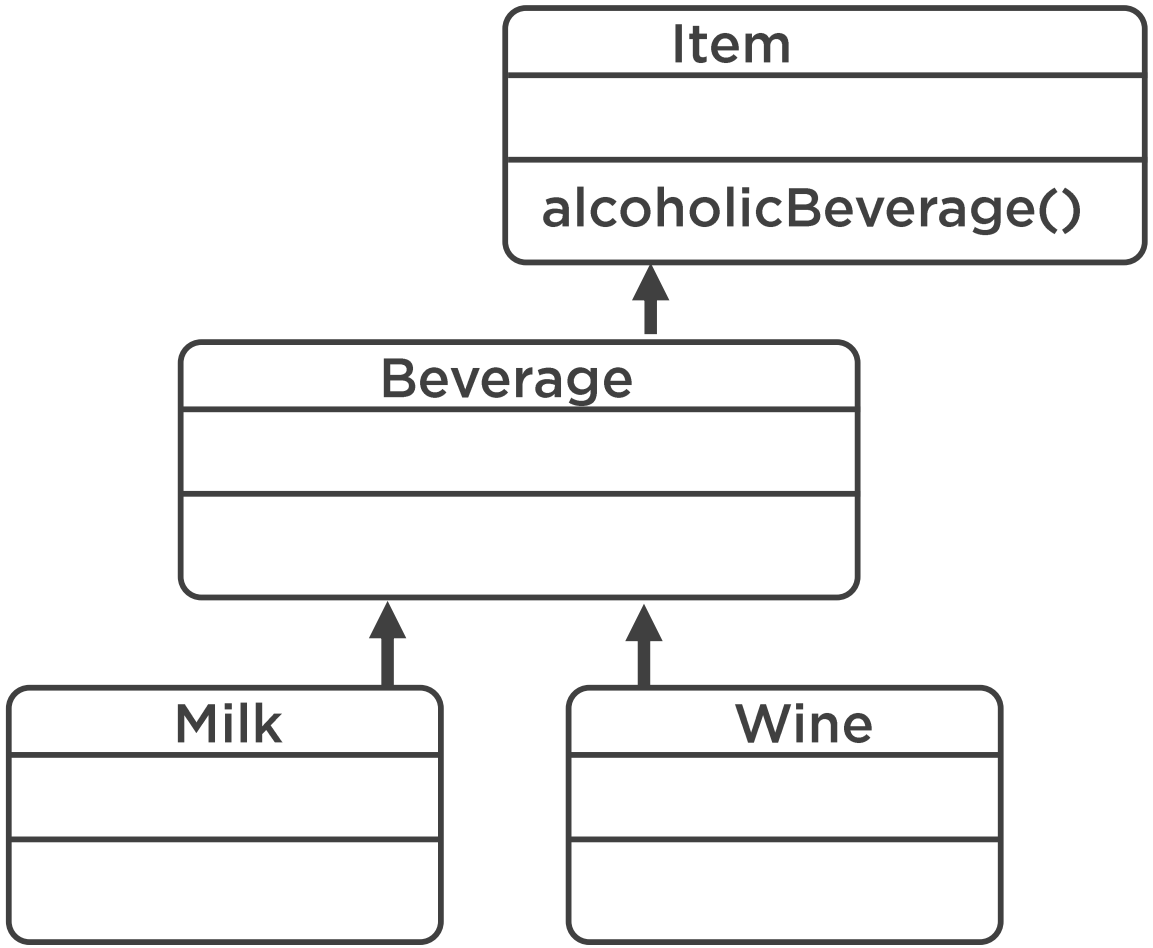
Rename

Rename method or field so that it makes sense for all subclasses

Push Down

Move the method or field down to a subclass, introduce a new one in the hierarchy if necessary

Create Class and Push Down



Favor Composition over Inheritance

In case of Refused Bequest consider redesigning your inheritance tree

Temporary Field

Fields that have values only under certain circumstances and needed by only certain methods. They are empty the rest of the time.

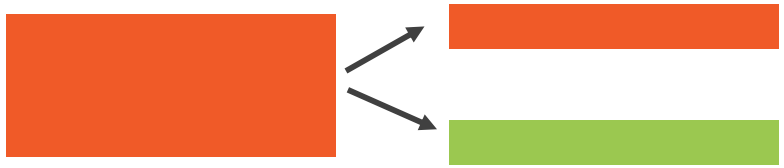
```
class SomeClass {


double doSomething(arg1,arg2 ,arg3,arg4){
    // do something with args
  }
}
```

Temporary Field

Confusing – “why is this field null half of the time” ?

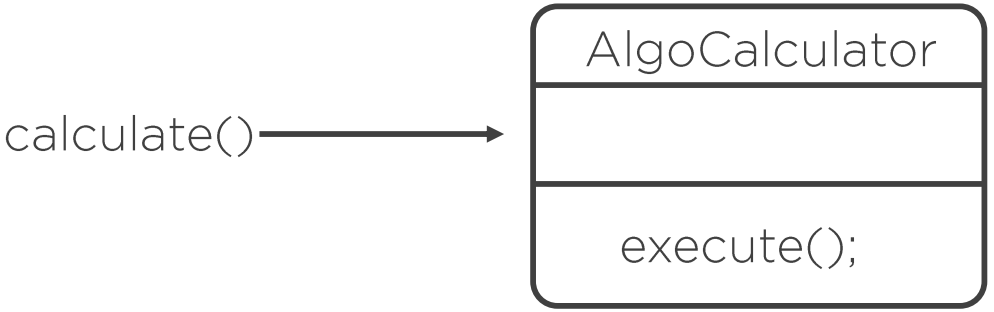
Temporary Fields indicate low class cohesion

How to Fix Temporary Field



Extract Class

(Split into several smaller classes)



Replace Method with Method Object

(A combination of Extract Class and Move Method)

```
class Order {
  double calculate(a1, a2, a3, a4){
      // impl
    }
}
```

```
class Order {
    double calculate(){
    return new AlgoCalculator(this).execute();
    }
}
class AlgoCalculator {
    a1, a2, a3, a4;
    public AlgoCalculator(Order o) { // ... }
    public double execute(){ // ... }
}
```

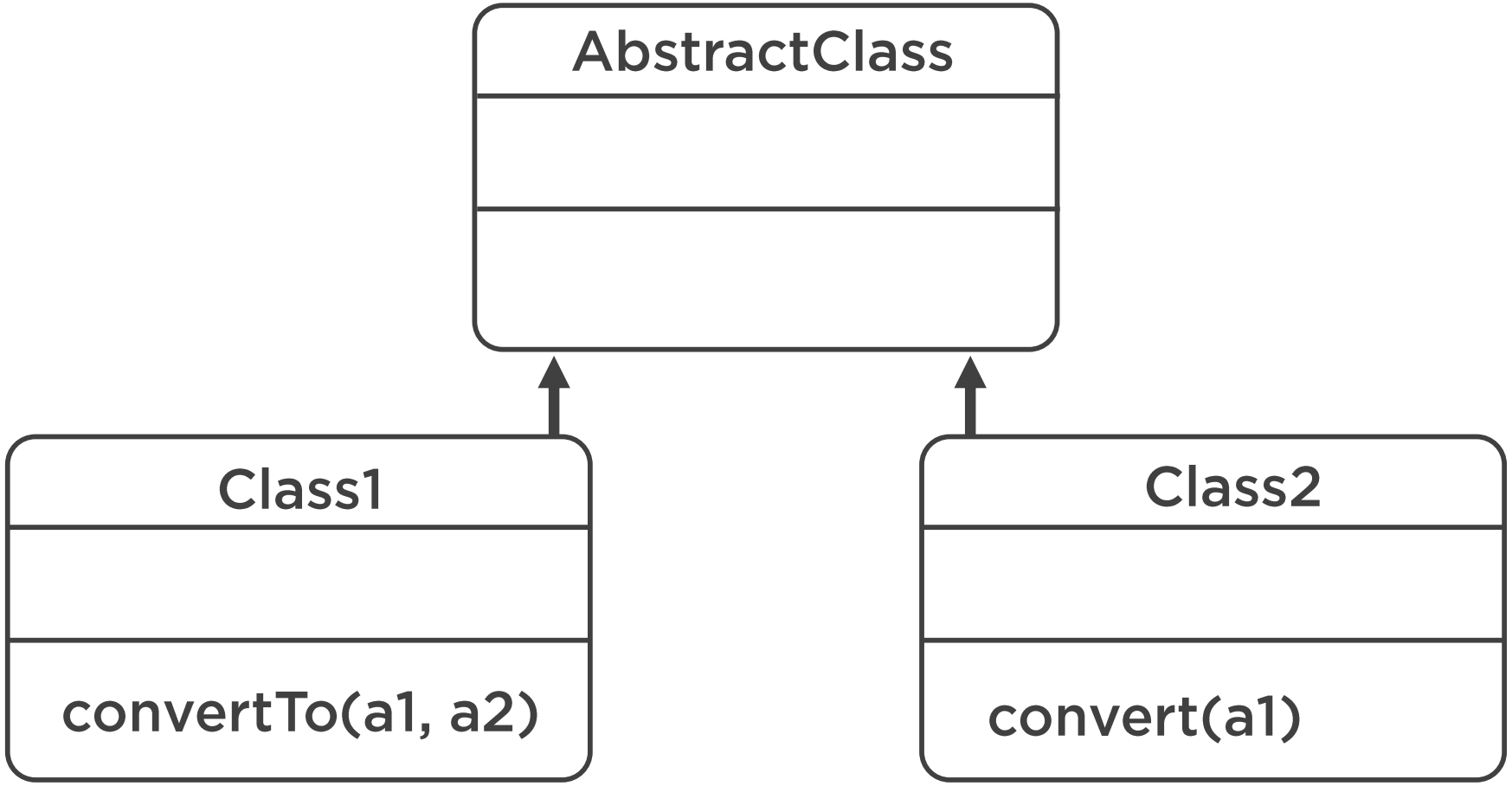
Benefits Achieved

Better code clarity

Alternative Classes with Different Interfaces

Two or more methods exist across multiple classes that do the same thing.

Alternative Classes with Similar Methods



New Requirements

Display the price in dollars AND in the customer's local currency if they are not based in the US

What about conditional complexity and primitive obsession?

Different Interfaces Issues

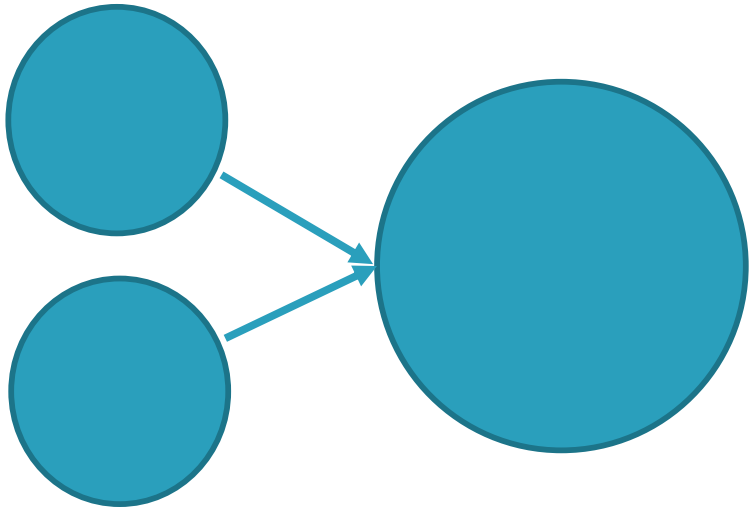
Not DRY – code is duplicated with just minor variations

Can cause problems if one place is updated, but not the other

“[Alternative Classes with Different Interfaces] code smell exposes a lack of a common interface for closely related classes, so it can also be considered a certain type of inheritance misuse”

Mäntylä, M. V. and Lassenius, C.

How to Fix Different Interfaces



Combine into one

Change until both methods are identical and leave just one

Benefits Achieved

DRY code

Single point of change

When to Ignore

When classes with different interfaces are in separate libraries you can't modify

Summary

- Conditional Complexity should be replaced with Polymorphism or Delegation
- Subclasses should inherit only what they need
- Avoid Temporary Fields
- Keep a single solution for one task