Refactoring In Practice

JHUG Meetup, Trasys Greece

Panagiotis Kapralos, Solution Architect@Trasys Greece

March 15, 2017

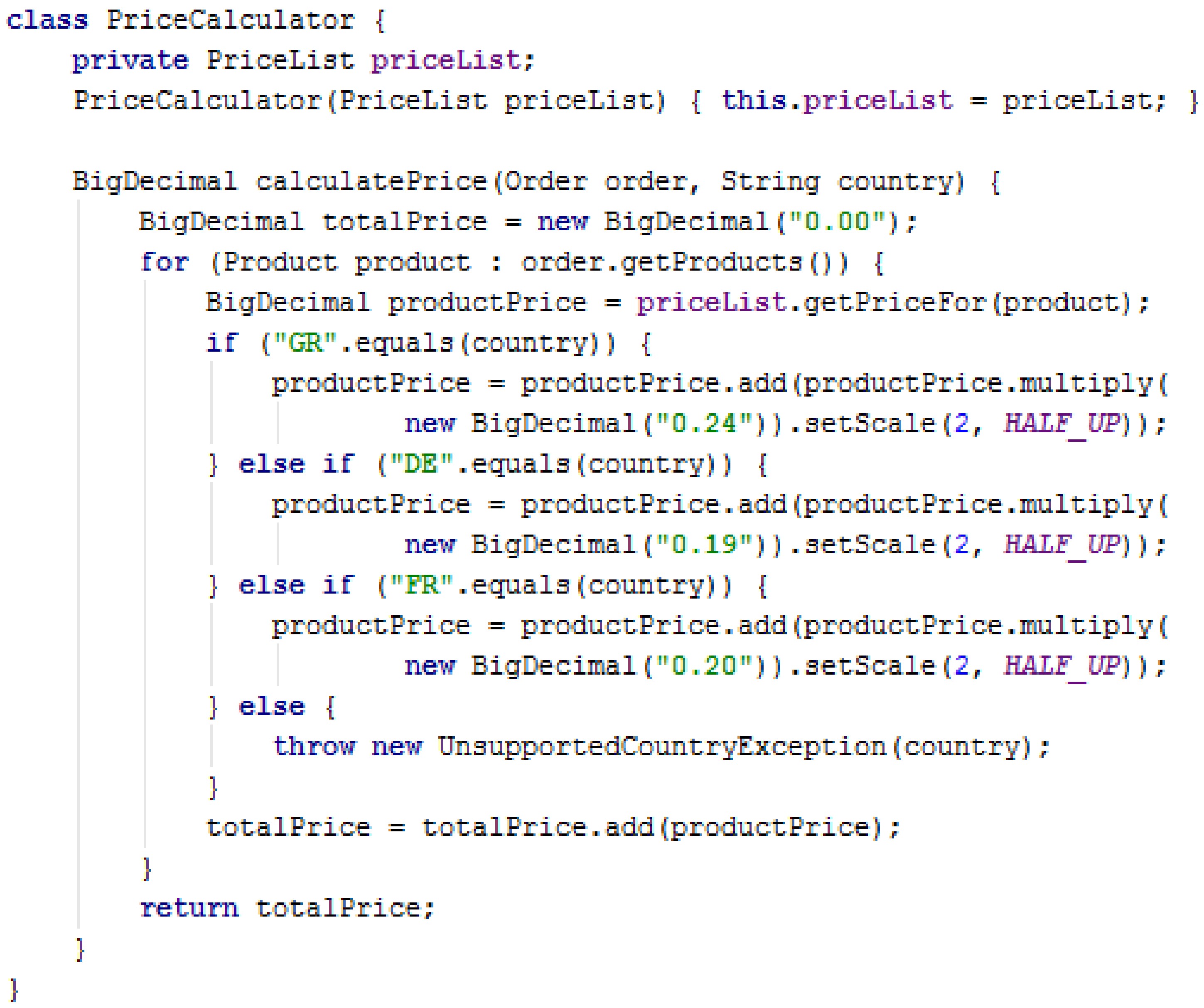
Let’s Demonstrate Refactoring with an Example

The PriceCalculator example

Calculates total price for an order (VAT included)

Each country has different VAT rate

We have been asked to add support for a few more countries



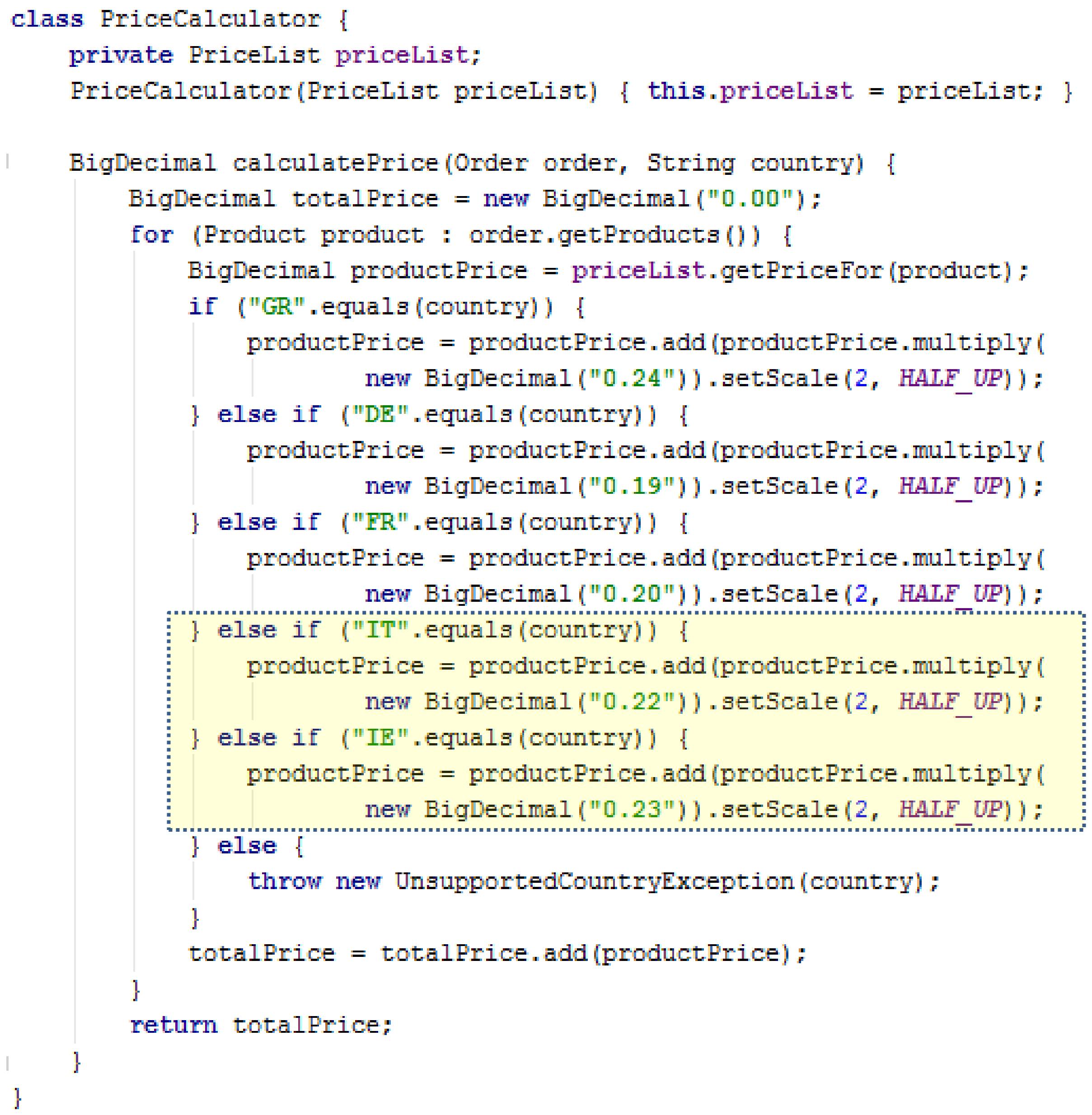
```java
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = priceList.getPriceFor(product);
            if ("GR".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.24")).setScale(2, HALF_UP));
            } else if ("DE".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.19")).setScale(2, HALF_UP));
            } else if ("FR".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.20")).setScale(2, HALF_UP));
            } else {
                throw new UnsupportedCountryException(country);
            }
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }
}
```

So, we could just add a few more else if…

This method is getting too long and complex.

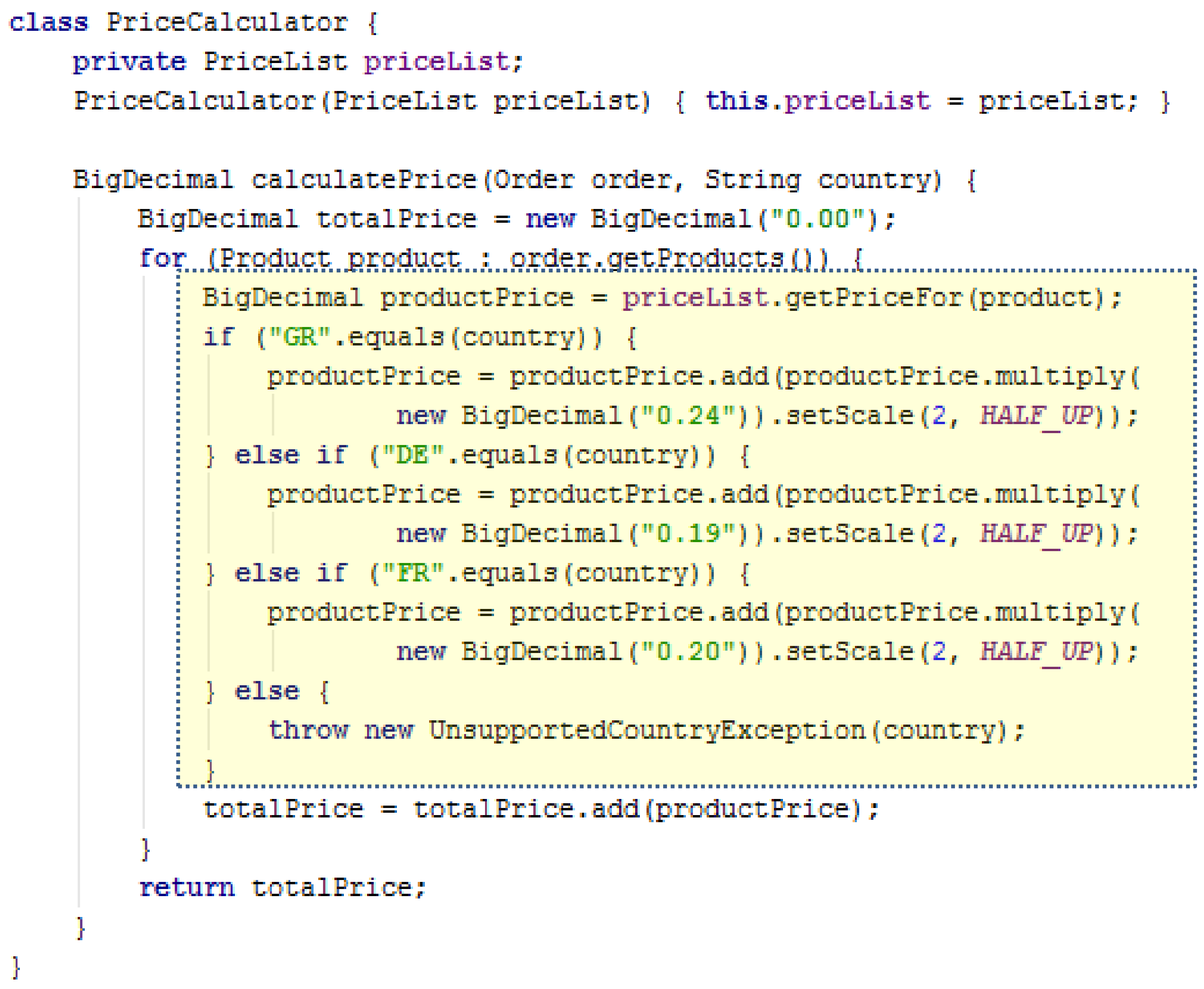
It also has code duplication.

Can we do better?
Let’s go back and try refactoring



```
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = priceList.getPriceFor(product);
            if ("GR".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.24")).setScale(2, HALF_UP));
            } else if ("DE".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.19")).setScale(2, HALF_UP));
            } else if ("FR".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.20")).setScale(2, HALF_UP));
            } else if ("IT".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.22")).setScale(2, HALF_UP));
            } else if ("IE".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.23")).setScale(2, HALF_UP));
            } else {
                throw new UnsupportedCountryException(country);
            }
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }
}
```

Refactoring 1. Extract Method…

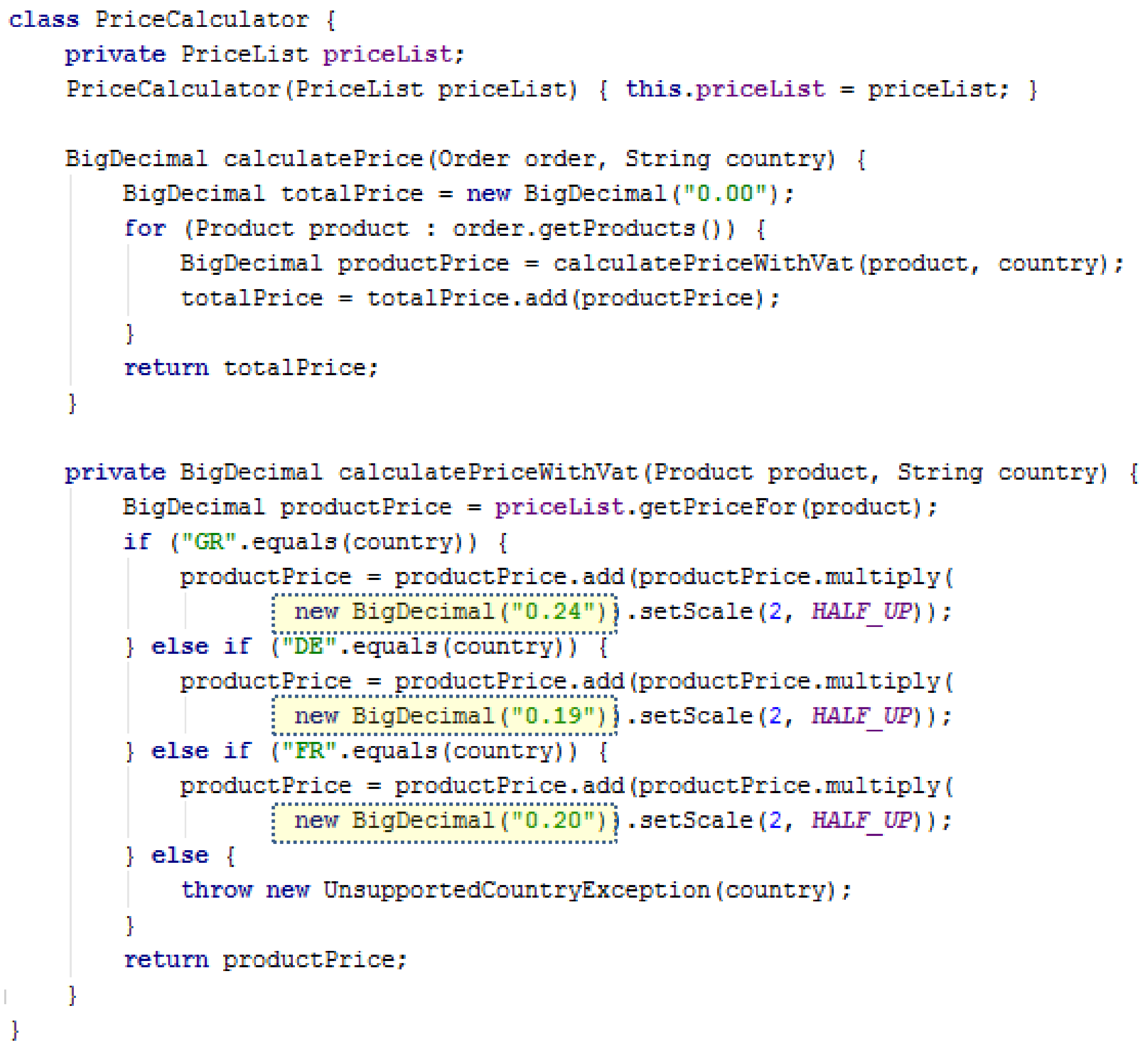


```java
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = priceList.getPriceFor(product);
            if ("GR".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.24")).setScale(2, HALF_UP));
            } else if ("DE".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.19")).setScale(2, HALF_UP));
            } else if ("FR".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.20")).setScale(2, HALF_UP));
            } else {
                throw new UnsupportedCountryException(country);
            }
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }
}
```

Refactoring 1. Extract Method

```
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = calculatePriceWithVat(product, country);
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }

    private BigDecimal calculatePriceWithVat(Product product, String country) {
        BigDecimal productPrice = priceList.getPriceFor(product);
        if ("GR".equals(country)) {
            productPrice = productPrice.add(productPrice.multiply(
                    new BigDecimal("0.24")).setScale(2, HALF_UP));
        } else if ("DE".equals(country)) {
            productPrice = productPrice.add(productPrice.multiply(
                    new BigDecimal("0.19")).setScale(2, HALF_UP));
        } else if ("FR".equals(country)) {
            productPrice = productPrice.add(productPrice.multiply(
                    new BigDecimal("0.20")).setScale(2, HALF_UP));
        } else {
            throw new UnsupportedCountryException(country);
        }
        return productPrice;
    }
}
```

Refactoring 2. Extract variable...

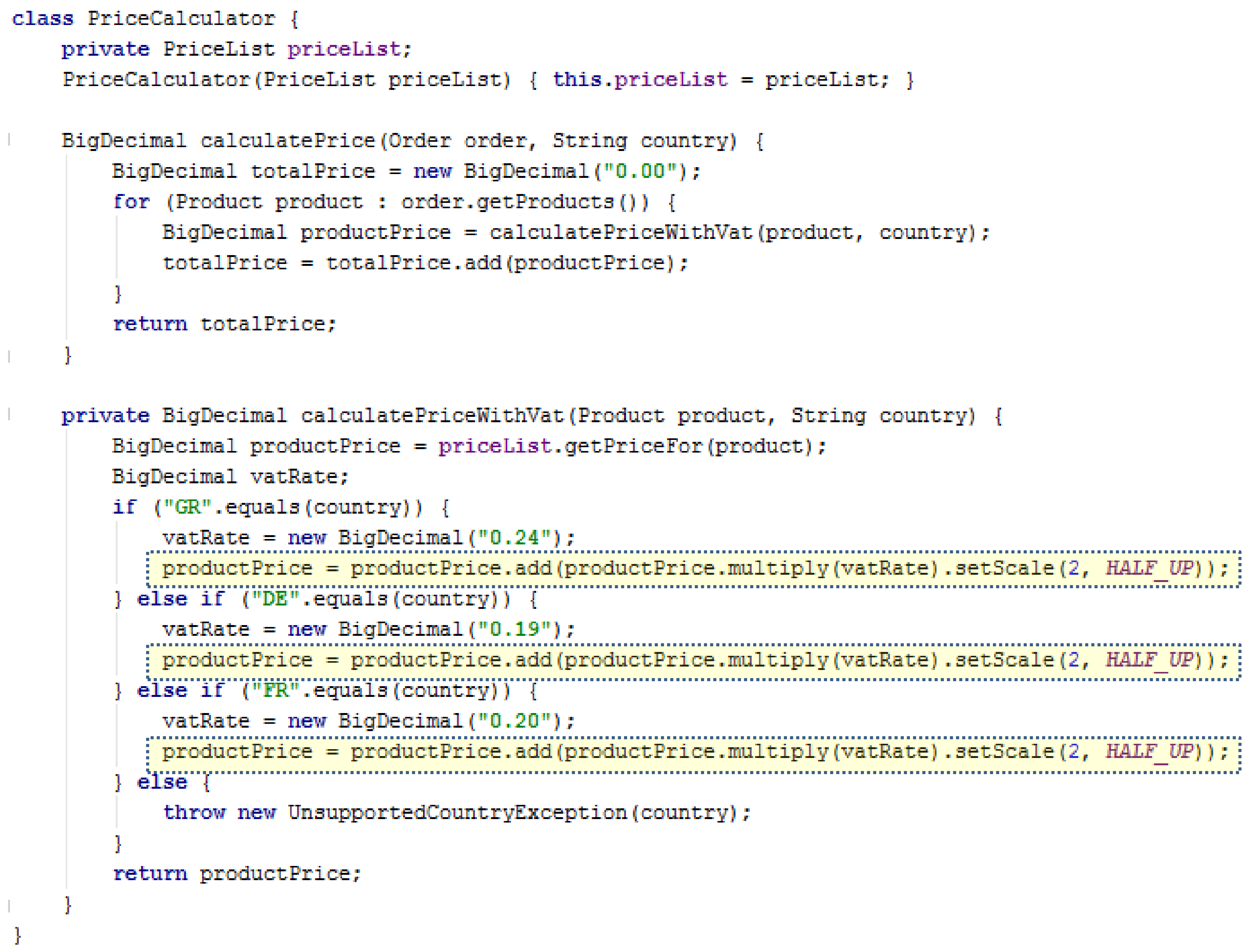


```java
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = calculatePriceWithVat(product, country);
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }

    private BigDecimal calculatePriceWithVat(Product product, String country) {
        BigDecimal productPrice = priceList.getPriceFor(product);
        if ("GR".equals(country)) {
            productPrice = productPrice.add(productPrice.multiply(
                    new BigDecimal("0.24"))).setScale(2, HALF_UP));
        } else if ("DE".equals(country)) {
            productPrice = productPrice.add(productPrice.multiply(
                    new BigDecimal("0.19"))).setScale(2, HALF_UP));
        } else if ("FR".equals(country)) {
            productPrice = productPrice.add(productPrice.multiply(
                    new BigDecimal("0.20"))).setScale(2, HALF_UP));
        } else {
            throw new UnsupportedCountryException(country);
        }
        return productPrice;
    }
}
```

Refactoring 2. Extract variable

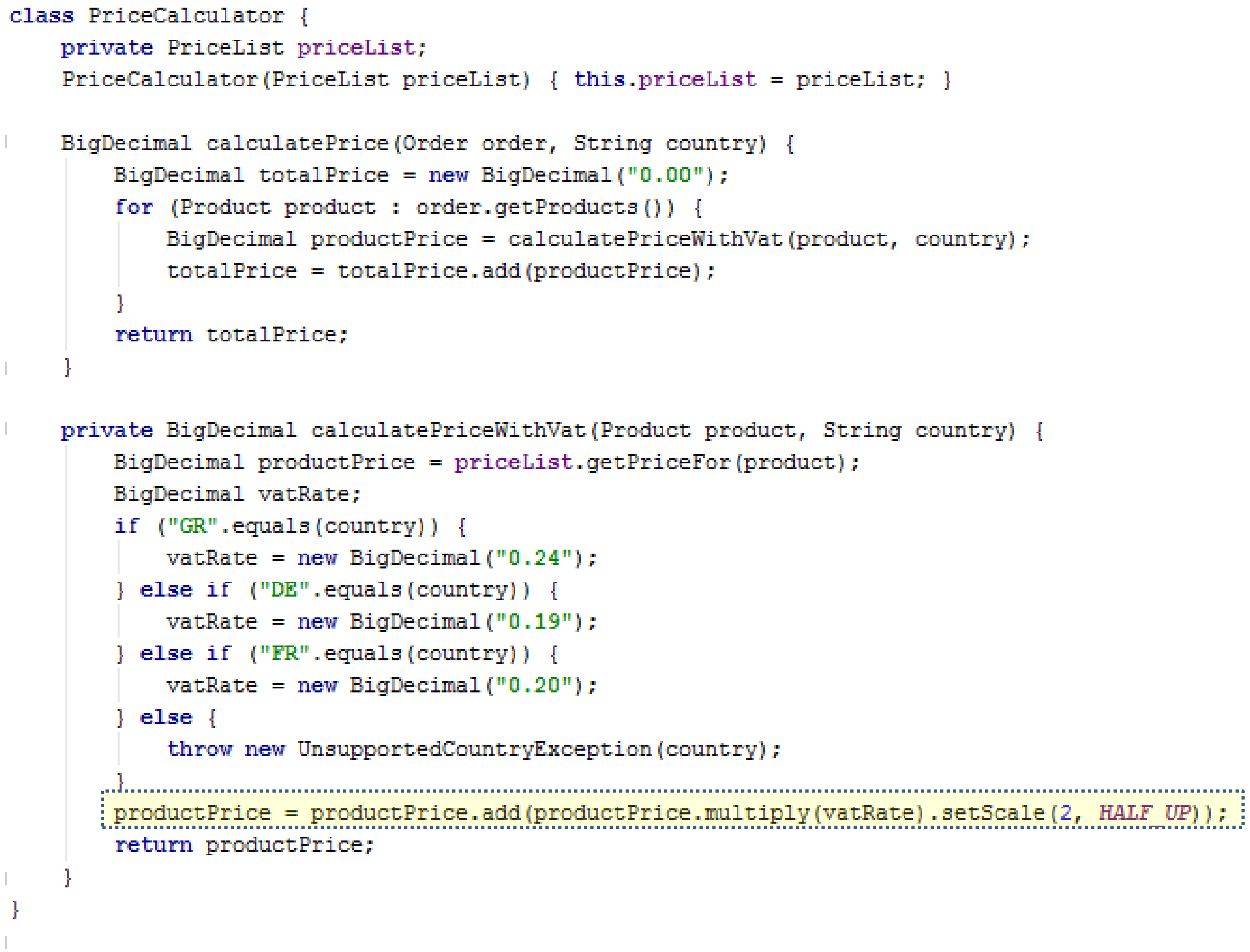
```java
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = calculatePriceWithVat(product, country);
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }

    private BigDecimal calculatePriceWithVat(Product product, String country) {
        BigDecimal productPrice = priceList.getPriceFor(product);
        BigDecimal vatRate;
        if ("GR".equals(country)) {
            vatRate = new BigDecimal("0.24");
            productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        } else if ("DE".equals(country)) {
            vatRate = new BigDecimal("0.19");
            productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        } else if ("FR".equals(country)) {
            vatRate = new BigDecimal("0.20");
            productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        } else {
            throw new UnsupportedCountryException(country);
        }
        return productPrice;
    }
}
```

Refactoring 3. Consolidate duplicate conditional fragments…



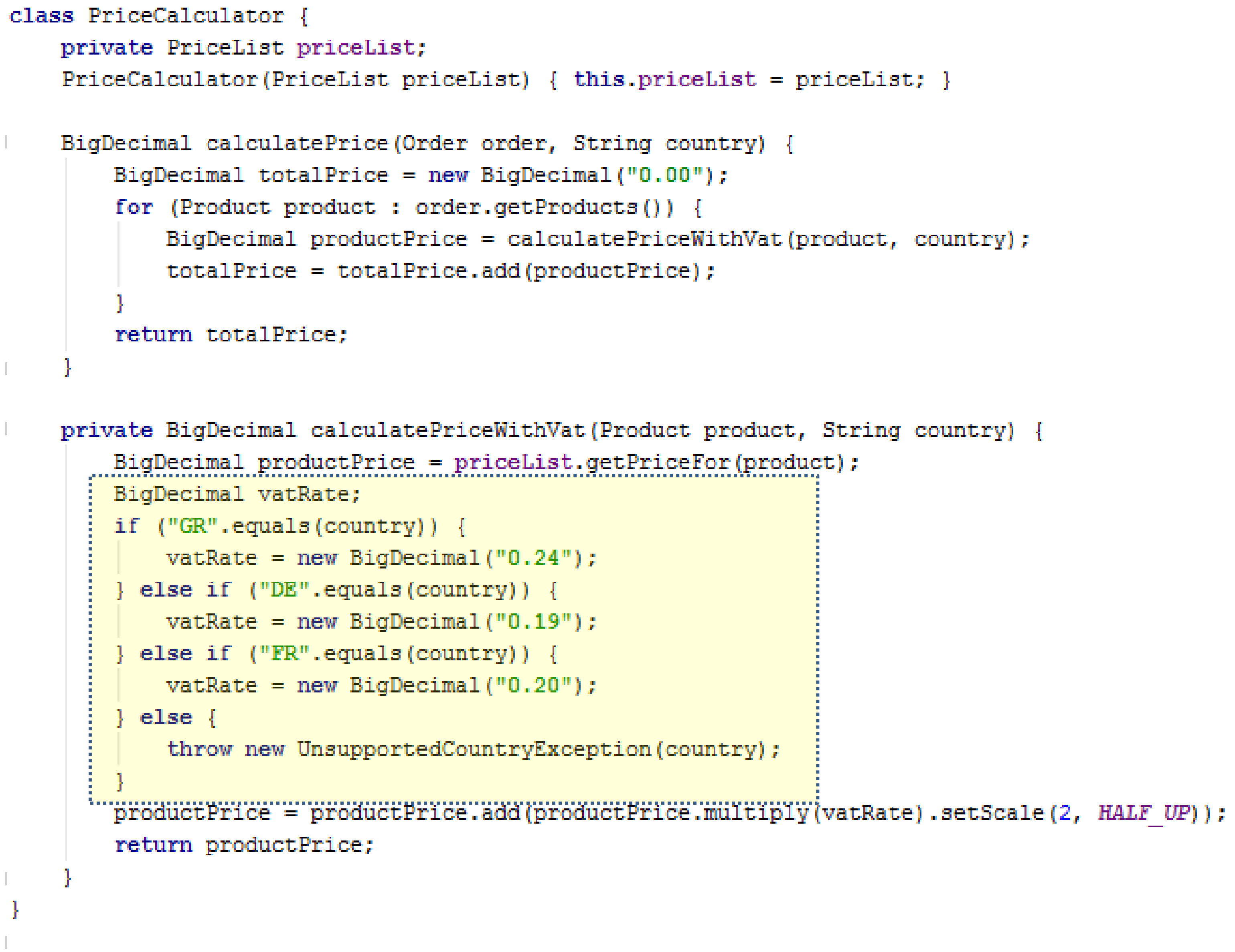
```java
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = calculatePriceWithVat(product, country);
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }

    private BigDecimal calculatePriceWithVat(Product product, String country) {
        BigDecimal productPrice = priceList.getPriceFor(product);
        BigDecimal vatRate;
        if ("GR".equals(country)) {
            vatRate = new BigDecimal("0.24");
            productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        } else if ("DE".equals(country)) {
            vatRate = new BigDecimal("0.19");
            productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        } else if ("FR".equals(country)) {
            vatRate = new BigDecimal("0.20");
            productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        } else {
            throw new UnsupportedCountryException(country);
        }
        return productPrice;
    }
}
```

Refactoring 3. Consolidate duplicate conditional fragments



```java
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = calculatePriceWithVat(product, country);
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }

    private BigDecimal calculatePriceWithVat(Product product, String country) {
        BigDecimal productPrice = priceList.getPriceFor(product);
        BigDecimal vatRate;
        if ("GR".equals(country)) {
            vatRate = new BigDecimal("0.24");
        } else if ("DE".equals(country)) {
            vatRate = new BigDecimal("0.19");
        } else if ("FR".equals(country)) {
            vatRate = new BigDecimal("0.20");
        } else {
            throw new UnsupportedCountryException(country);
        }
        productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        return productPrice;
    }
}
```

Refactoring 4. Extract method…



```java
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = calculatePriceWithVat(product, country);
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }

    private BigDecimal calculatePriceWithVat(Product product, String country) {
        BigDecimal productPrice = priceList.getPriceFor(product);
        BigDecimal vatRate;
        if ("GR".equals(country)) {
            vatRate = new BigDecimal("0.24");
        } else if ("DE".equals(country)) {
            vatRate = new BigDecimal("0.19");
        } else if ("FR".equals(country)) {
            vatRate = new BigDecimal("0.20");
        } else {
            throw new UnsupportedCountryException(country);
        }
        productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        return productPrice;
    }
}
```

Refactoring 4. Extract method

```java
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = calculatePriceWithVat(product, country);
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }

    private BigDecimal calculatePriceWithVat(Product product, String country) {
        BigDecimal productPrice = priceList.getPriceFor(product);
        BigDecimal vatRate = getVatRateFor(country);
        productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        return productPrice;
    }

    private BigDecimal getVatRateFor(String country) {
        BigDecimal vatRate;
        if ("GR".equals(country)) {
            vatRate = new BigDecimal("0.24");
        } else if ("DE".equals(country)) {
            vatRate = new BigDecimal("0.19");
        } else if ("FR".equals(country)) {
            vatRate = new BigDecimal("0.20");
        } else {
            throw new UnsupportedCountryException(country);
        }
        return vatRate;
    }
}
```

Refactoring 5. Substitute algorithm…

```java
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = calculatePriceWithVat(product, country);
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }

    private BigDecimal calculatePriceWithVat(Product product, String country) {
        BigDecimal productPrice = priceList.getPriceFor(product);
        BigDecimal vatRate = getVatRateFor(country);
        productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        return productPrice;
    }

    private BigDecimal getVatRateFor(String country) {
        BigDecimal vatRate;
        if ("GR".equals(country)) {
            vatRate = new BigDecimal("0.24");
        } else if ("DE".equals(country)) {
            vatRate = new BigDecimal("0.19");
        } else if ("FR".equals(country)) {
            vatRate = new BigDecimal("0.20");
        } else {
            throw new UnsupportedCountryException(country);
        }
        return vatRate;
    }
}
```

Refactoring 5. Substitute algorithm

```java
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = calculatePriceWithVat(product, country);
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }

    private BigDecimal calculatePriceWithVat(Product product, String country) {
        BigDecimal productPrice = priceList.getPriceFor(product);
        BigDecimal vatRate = getVatRateFor(country);
        productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        return productPrice;
    }

    private BigDecimal getVatRateFor(String country) {
        Map<String, BigDecimal> vatRates = new HashMap<>();
        vatRates.put("GR", new BigDecimal("0.24"));
        vatRates.put("DE", new BigDecimal("0.19"));
        vatRates.put("FR", new BigDecimal("0.20"));
        if (!vatRates.containsKey(country)) {
            throw new UnsupportedCountryException(country);
        }
        return vatRates.get(country);
    }
}
```

Refactoring 6. Extract Class…

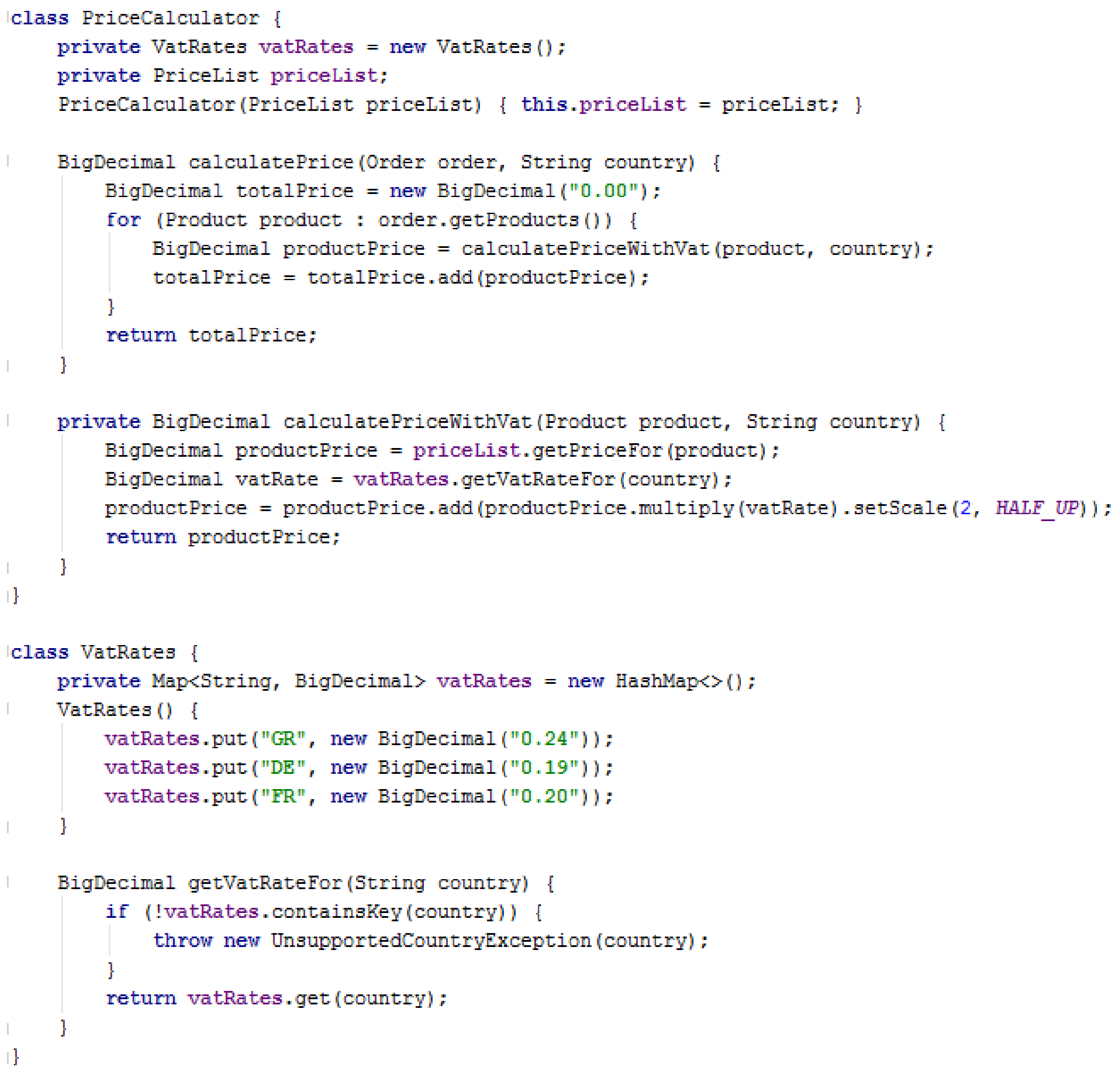
```java
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = calculatePriceWithVat(product, country);
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }

    private BigDecimal calculatePriceWithVat(Product product, String country) {
        BigDecimal productPrice = priceList.getPriceFor(product);
        BigDecimal vatRate = getVatRateFor(country);
        productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        return productPrice;
    }

    private BigDecimal getVatRateFor(String country) {
        Map<String, BigDecimal> vatRates = new HashMap<>();
        vatRates.put("GR", new BigDecimal("0.24"));
        vatRates.put("DE", new BigDecimal("0.19"));
        vatRates.put("FR", new BigDecimal("0.20"));
        if (!vatRates.containsKey(country)) {
            throw new UnsupportedCountryException(country);
        }
        return vatRates.get(country);
    }
}
```

Refactoring 6. Extract Class

```java
class PriceCalculator {
    private VatRates vatRates = new VatRates();
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = calculatePriceWithVat(product, country);
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }

    private BigDecimal calculatePriceWithVat(Product product, String country) {
        BigDecimal productPrice = priceList.getPriceFor(product);
        BigDecimal vatRate = vatRates.getVatRateFor(country);
        productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        return productPrice;
    }
}
```

```java
class VatRates {
    private Map<String, BigDecimal> vatRates = new HashMap<>();
    VatRates() {
        vatRates.put("GR", new BigDecimal("0.24"));
        vatRates.put("DE", new BigDecimal("0.19"));
        vatRates.put("FR", new BigDecimal("0.20"));
    }

    BigDecimal getVatRateFor(String country) {
        if (!vatRates.containsKey(country)) {
            throw new UnsupportedCountryException(country);
        }
        return vatRates.get(country);
    }
}
```

Refactoring 6. Extract Class



```
class PriceCalculator {
    private VatRates vatRates = new VatRates();
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = calculatePriceWithVat(product, country);
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }

    private BigDecimal calculatePriceWithVat(Product product, String country) {
        BigDecimal productPrice = priceList.getPriceFor(product);
        BigDecimal vatRate = vatRates.getVatRateFor(country);
        productPrice = productPrice.add(productPrice.multiply(vatRate).setScale(2, HALF_UP));
        return productPrice;
    }
}

class VatRates {
    private Map<String, BigDecimal> vatRates = new HashMap<>();
    VatRates() {
        vatRates.put("GR", new BigDecimal("0.24"));
        vatRates.put("DE", new BigDecimal("0.19"));
        vatRates.put("FR", new BigDecimal("0.20"));
    }

    BigDecimal getVatRateFor(String country) {
        if (!vatRates.containsKey(country)) {
            throw new UnsupportedCountryException(country);
        }
        return vatRates.get(country);
    }
}
```

No Code Duplication

Separate classes for separate responsibilities

Small classes and methods

Class, method and variable names communicate intention

Add VAT rates for new countries

```
class VatRates {
    private Map<String, BigDecimal> vatRates = new HashMap<>();

    VatRates() {
        vatRates.put("GR", new BigDecimal("0.24"));
        vatRates.put("DE", new BigDecimal("0.19"));
        vatRates.put("FR", new BigDecimal("0.20"));
        vatRates.put("IT", new BigDecimal("0.22"));
        vatRates.put("IE", new BigDecimal("0.23"));
    }

    BigDecimal getVatRateFor(String country) {
        if (!vatRates.containsKey(country)) {
            throw new UnsupportedCountryException(country);
        }
        return vatRates.get(country);
    }
}
```

Adding support for new countries now can be done by changing the VatRates class.

Refactoring is…

- ✓ *… a **disciplined technique** for restructuring an existing body of code, altering its internal structure without changing its external behavior.*

- ✓ *Its heart is a series of **small behavior preserving transformations**. Each transformation (called a "refactoring") does little, but a sequence of transformations can produce a significant restructuring. Since each refactoring is small, it's less likely to go wrong. The system is kept fully working after each small refactoring, reducing the chances that a system can get seriously broken during the restructuring.*

Source: www.refactoring.com

Refactoring Catalog (not exhaustive)

Collapse Hierarchy

Extract Method

Inline Method

Replace Magic Number with Symbolic Constant

Replace Conditional with Polymorphism

Extract Interface

Extract Class

Extract Variable

Encapsulate Field

Move Method

Move Field

Substitute Algorithm

Replace Type Code With Polymorphism

Rename Method

Extract Superclass

Consolidate Duplicate Conditional Fragments

Remove Middle Man

Replace Inheritance with Delegation

Introduce Parameter Object

So, why refactoring?

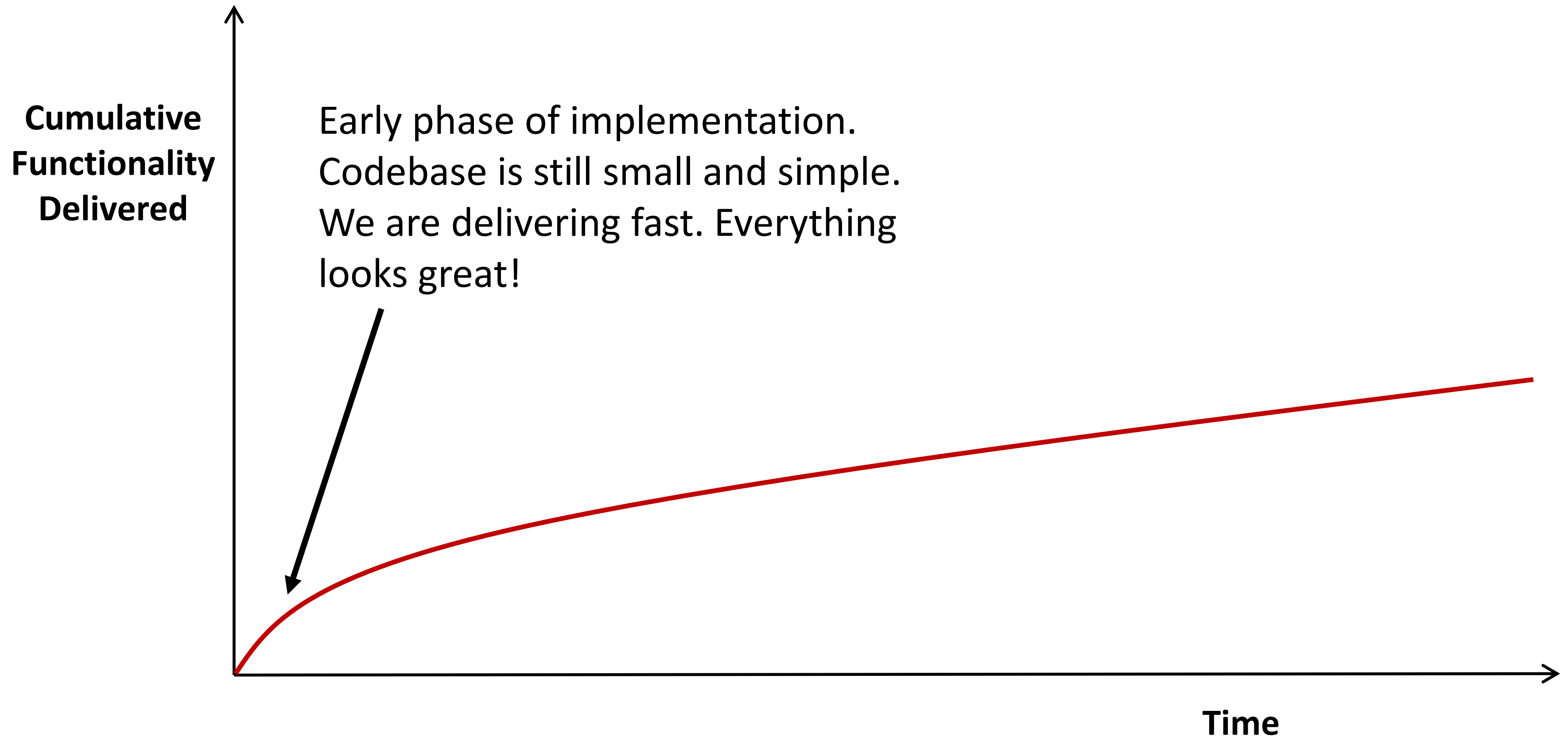
- ❓ The code is working. What’s the point changing it?
- ❓ After all, we are paid to add valuable features, not changing the code internals.
- ❓ So what is this all about? Is refactoring just making our code look “pretty”?

So, why refactoring?

- The code is working. What's the point changing it?
- After all, we are paid to add valuable features, not changing the code internals.
- So what is this all about? Is refactoring just making our code look "pretty"?

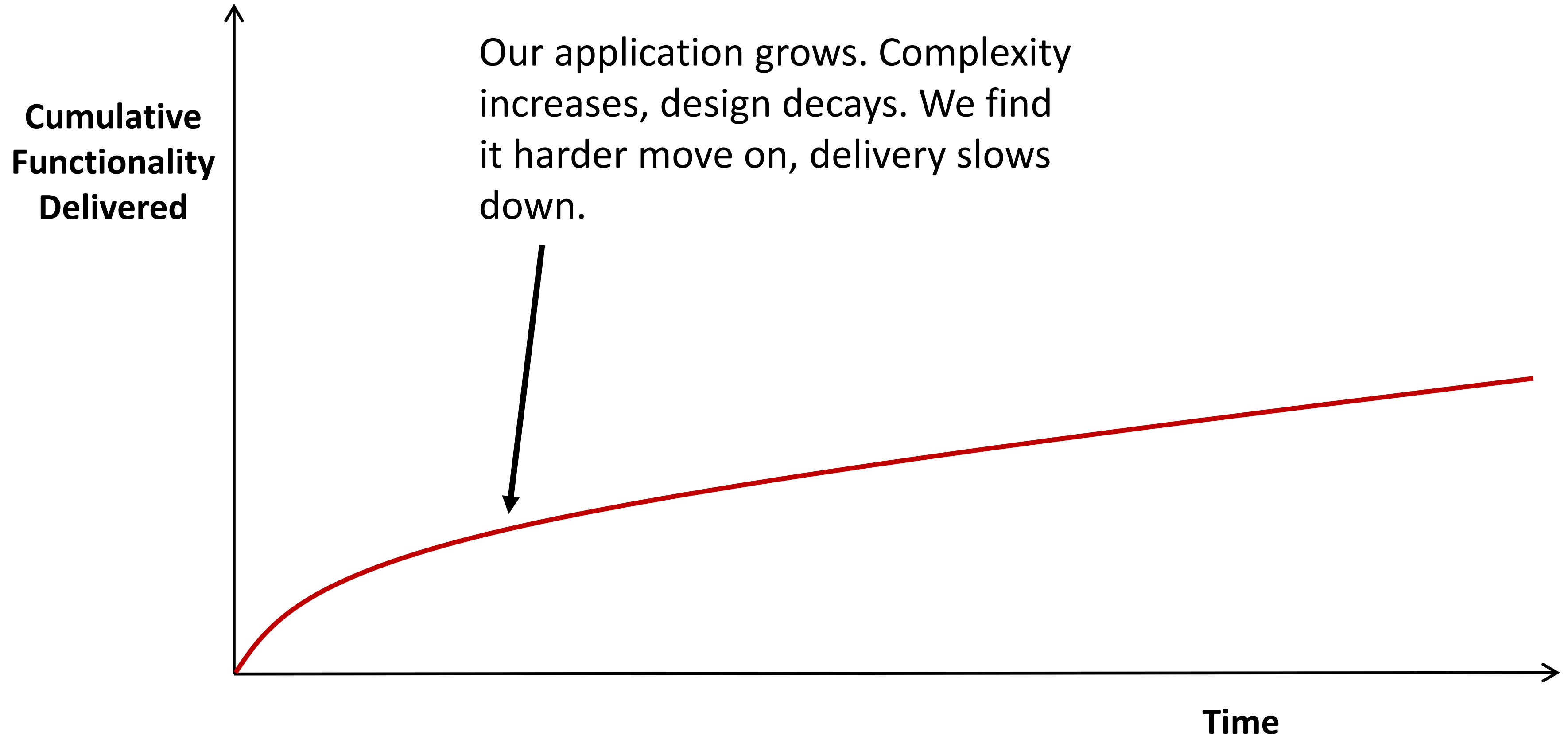
The real justification of refactoring is **Economics**. Let' see why…

Economics of Refactoring

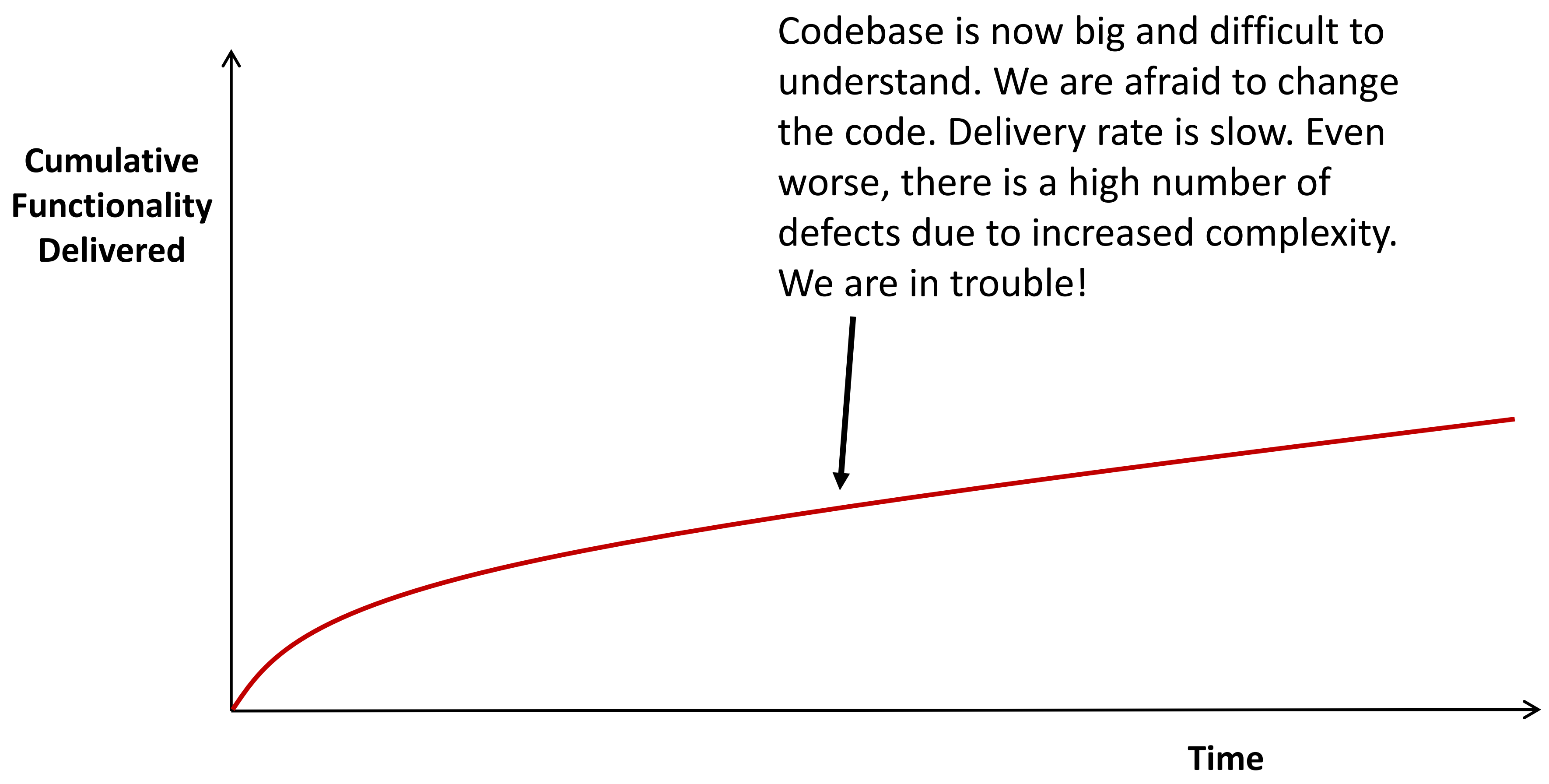


Early phase of implementation. Codebase is still small and simple. We are delivering fast. Everything looks great!

Economics of Refactoring

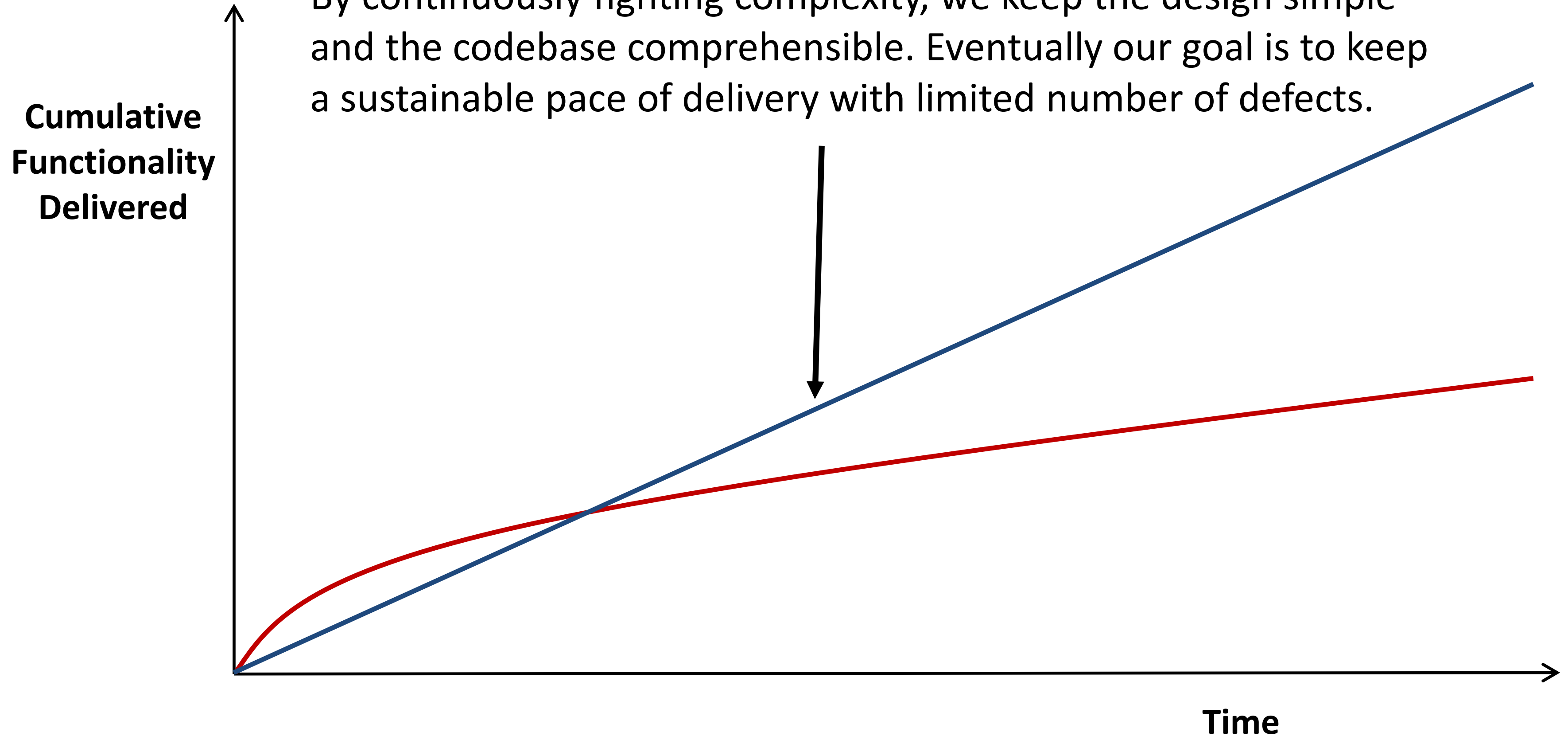


Economics of Refactoring

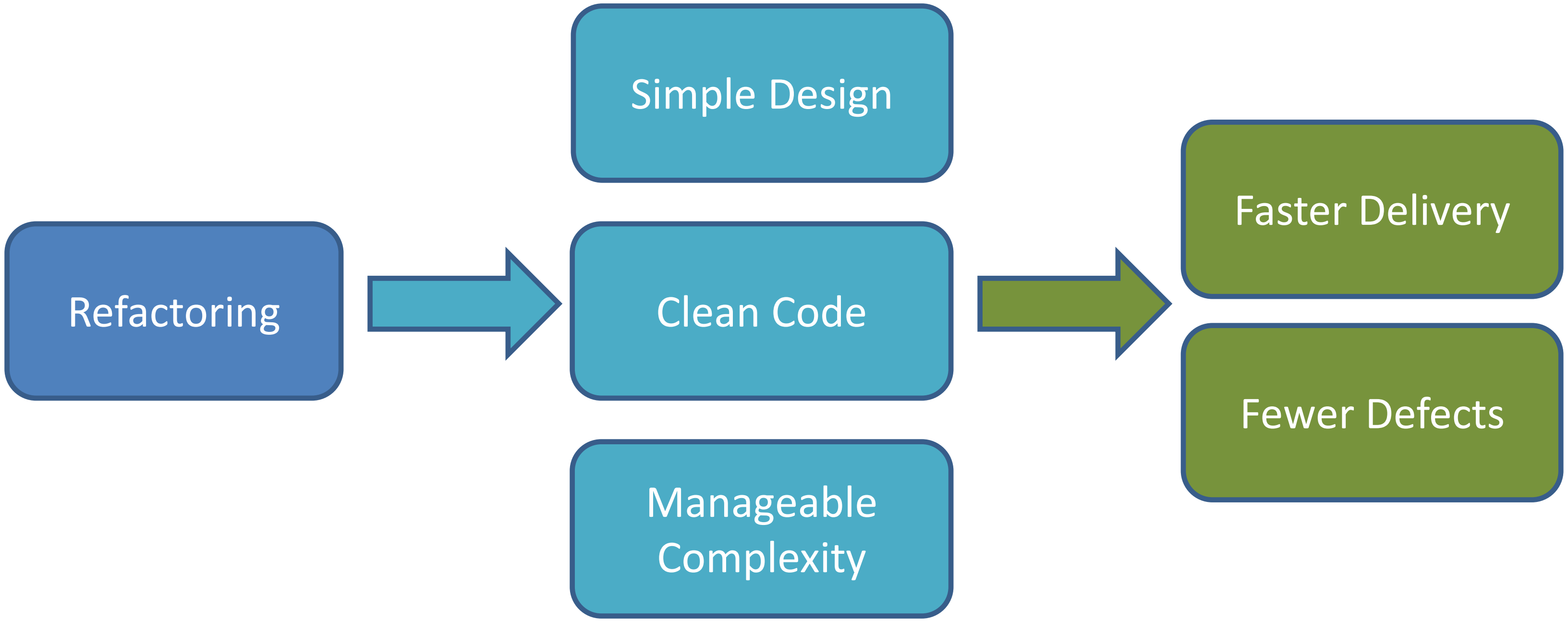


Economics of Refactoring

We need refactoring so that we can achieve something like this. By continuously fighting complexity, we keep the design simple and the codebase comprehensible. Eventually our goal is to keep a sustainable pace of delivery with limited number of defects.



Justification of Refactoring



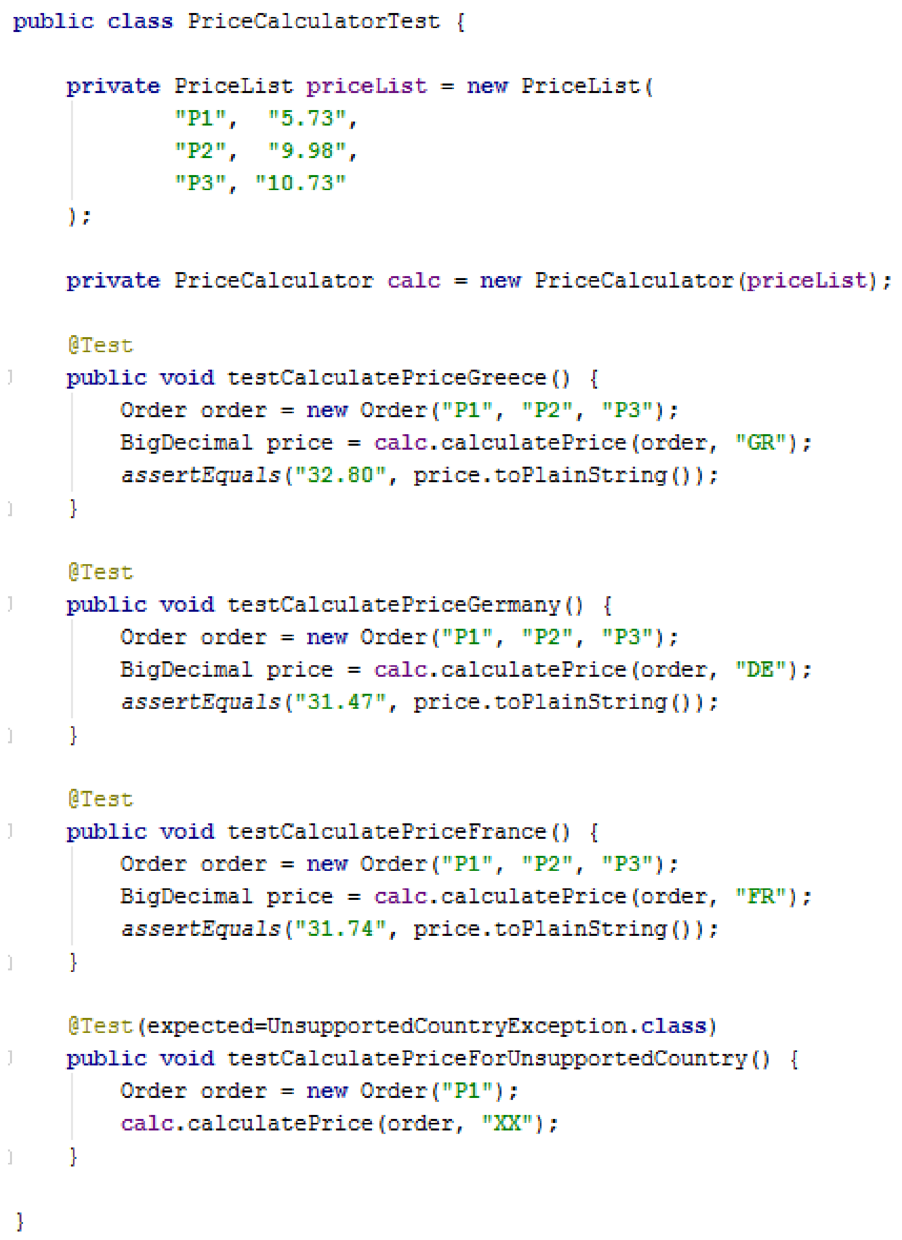
Practical Advices on Refactoring

1. Verify Refactoring with Automated Tests

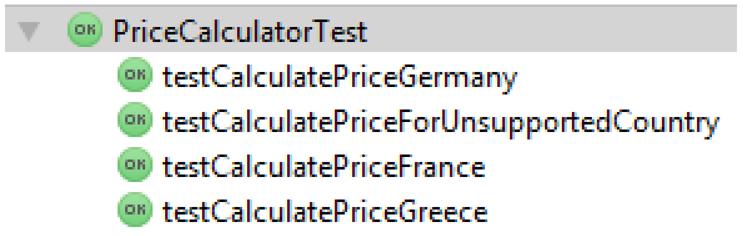
Always consider the possibility to break code while refactoring

- ✓ Always check that the code is covered by tests before refactoring
- ✓ If not, write some tests first and then refactor
- ✓ Refactor only when tests are **GREEN**. If code is broken, fix it and then refactor.
- ✓ Sometimes it is not practical or feasible to add tests. In this case, consider not to refactor.

The tests of PriceCalculator example



```java
public class PriceCalculatorTest {

    private PriceList priceList = new PriceList(
            "P1",  "5.73",
            "P2",  "9.98",
            "P3", "10.73"
    );

    private PriceCalculator calc = new PriceCalculator(priceList);

    @Test
    public void testCalculatePriceGreece() {
        Order order = new Order("P1", "P2", "P3");
        BigDecimal price = calc.calculatePrice(order, "GR");
        assertEquals("32.80", price.toPlainString());
    }

    @Test
    public void testCalculatePriceGermany() {
        Order order = new Order("P1", "P2", "P3");
        BigDecimal price = calc.calculatePrice(order, "DE");
        assertEquals("31.47", price.toPlainString());
    }

    @Test
    public void testCalculatePriceFrance() {
        Order order = new Order("P1", "P2", "P3");
        BigDecimal price = calc.calculatePrice(order, "FR");
        assertEquals("31.74", price.toPlainString());
    }

    @Test(expected=UnsupportedCountryException.class)
    public void testCalculatePriceForUnsupportedCountry() {
        Order order = new Order("P1");
        calc.calculatePrice(order, "XX");
    }

}
```



```
▼ OK PriceCalculatorTest
     OK testCalculatePriceGermany
     OK testCalculatePriceForUnsupportedCountry
     OK testCalculatePriceFrance
     OK testCalculatePriceGreece
```

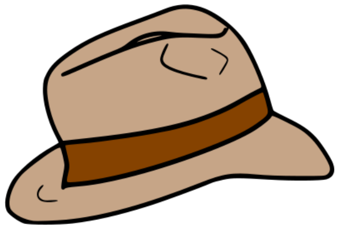
2. Separate refactoring from adding function

Refactoring and adding function at the same time may distract you

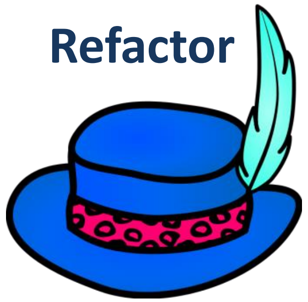
The **“Two Hats”** metaphor

- ✓ When programming, operate in one of two modes: either **refactoring** or **adding function**. You may switch hats frequently, but never wear both!
- ✓ Separating refactoring from adding function keeps your work **disciplined** and **focused**.

Add Function



Refactor



3. Favor opportunistic over planned refactoring

Organizing refactoring in planned tasks or “refactoring sprints” is hard to sell and interrupts the flow of delivery. Refactoring tasks usually get de-prioritized over feature tasks.

Opportunistic Refactoring

While working on your “normal” tasks, pickup the opportunity to refactor

- ✓ to make the task easier to implement (preparatory refactoring)
- ✓ to improve messy code (litter-pickup refactoring)
- ✓ to make code easier to understand (comprehension refactoring).

Planned refactoring is still useful in cases of restructure that needs dedicated effort. That’s okay, as long as planned refactoring is the exception and not the rule.

4. Balance refactoring with feature delivery

You start refactoring... then you see another piece of code that needs improvement… and another… you end up getting lost in refactoring and not delivering features!

The boy-scout rule: *always leave the code behind in a better state than you found it.*

Robert C. Martin (Uncle Bob)

✓ **Small improvements are okay**; it is not necessary to refactor everything in one go.

✓ **Be pragmatic**. Do not aim for "perfect" code, but for code that is good enough so that it is easily maintained and understood.

5. Let the code smells drive refactoring

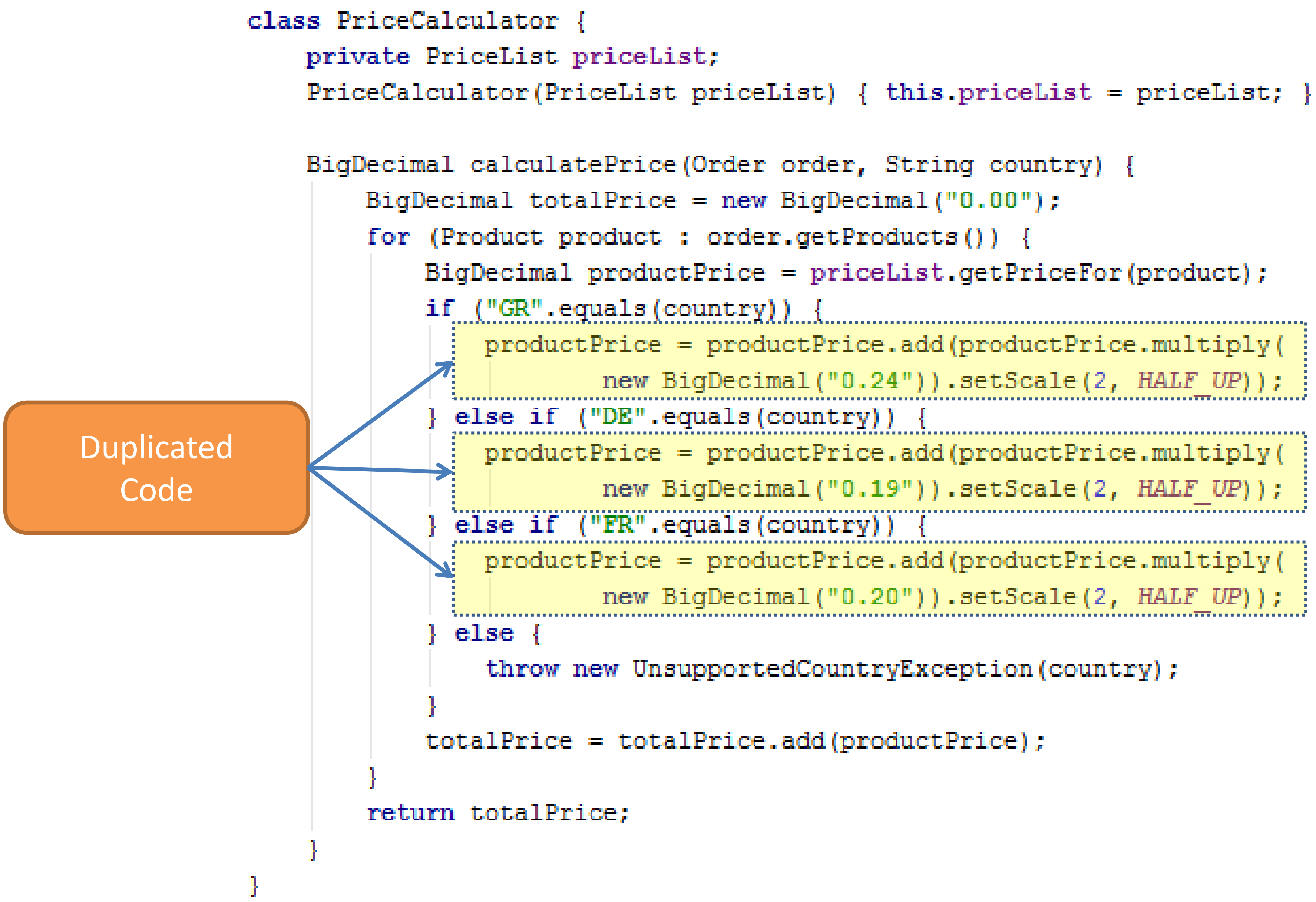
<u>Code Smell</u>

- ✓ A symptom in the code that is quick to spot and possibly indicates a deeper problem
- ✓ Can be used as heuristics to indicate when to refactor and what refactoring to apply
- ✓ Code smell is a driver for refactoring

Code Smells Catalog

- Long Method
- Dead Code
- Duplicated Code
- Large Class
- Long Parameter List
- Divergent Change
- Shotgun Surgery
- Feature Envy
- Data Clumps
- Switch Statements
- Comments
- Message Chains
- Data Class
- Temporary Field
- Middle Man
- Uncommunicative Name
- Primitive Obsession
- Inappropriate Intimacy
- Lazy Class
- Incomplete Library Class
- Speculative Generality
- Parallel Hierarchies
- Alternative Classes with Different Interfaces
- Refused Bequest

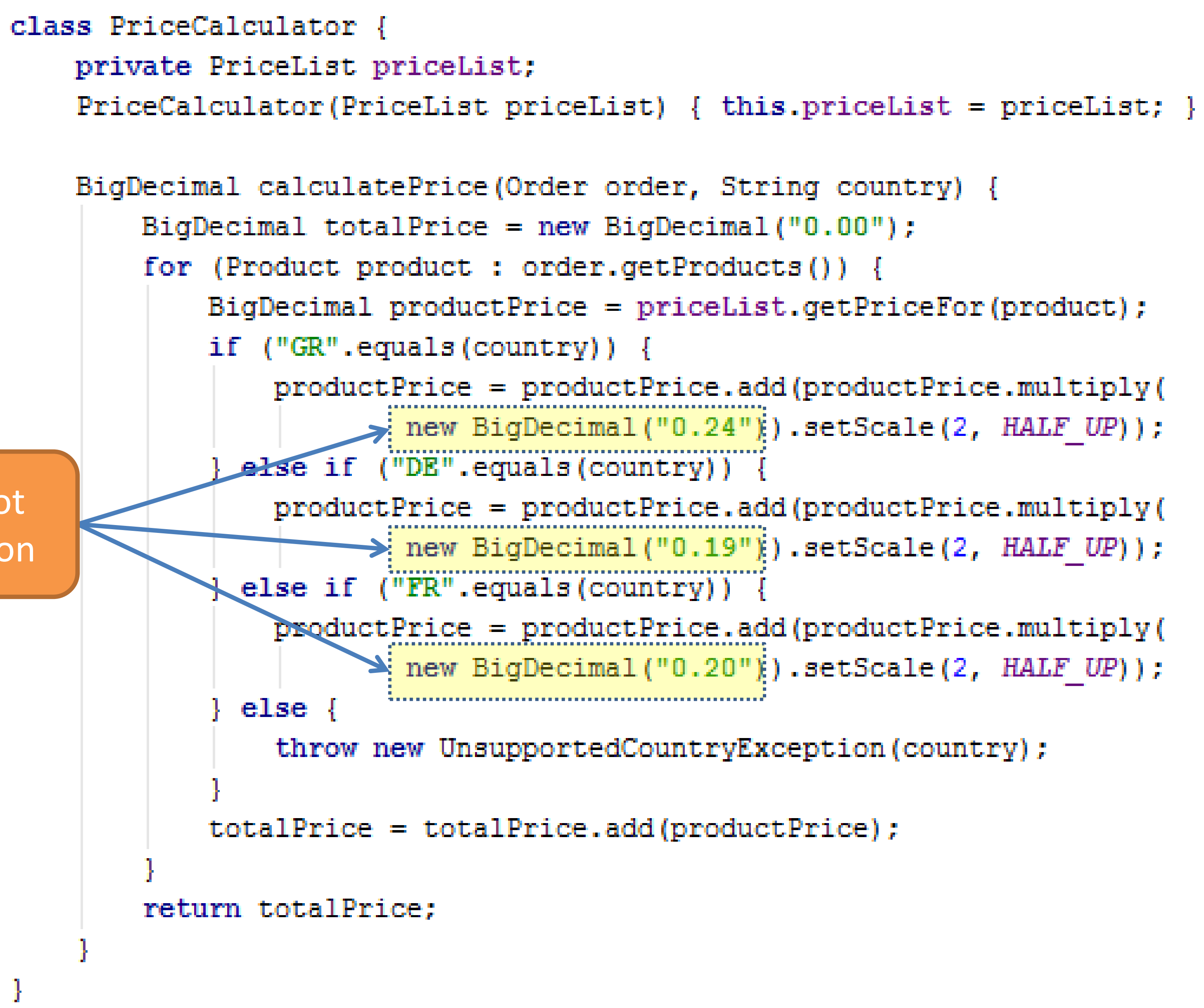
Code smells in the PriceCalculator example



```java
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = priceList.getPriceFor(product);
            if ("GR".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.24")).setScale(2, HALF_UP));
            } else if ("DE".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.19")).setScale(2, HALF_UP));
            } else if ("FR".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.20")).setScale(2, HALF_UP));
            } else {
                throw new UnsupportedCountryException(country);
            }
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }
}
```

Code smells in the PriceCalculator example

Code does not reveal intention



```java
class PriceCalculator {
    private PriceList priceList;
    PriceCalculator(PriceList priceList) { this.priceList = priceList; }

    BigDecimal calculatePrice(Order order, String country) {
        BigDecimal totalPrice = new BigDecimal("0.00");
        for (Product product : order.getProducts()) {
            BigDecimal productPrice = priceList.getPriceFor(product);
            if ("GR".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.24")).setScale(2, HALF_UP));
            } else if ("DE".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.19")).setScale(2, HALF_UP));
            } else if ("FR".equals(country)) {
                productPrice = productPrice.add(productPrice.multiply(
                        new BigDecimal("0.20")).setScale(2, HALF_UP));
            } else {
                throw new UnsupportedCountryException(country);
            }
            totalPrice = totalPrice.add(productPrice);
        }
        return totalPrice;
    }
}
```

6. Learn & use the refactoring capabilities of your IDE

Modern IDEs support refactoring. Familiarize yourself with the refactoring capabilities of your favorite IDE.

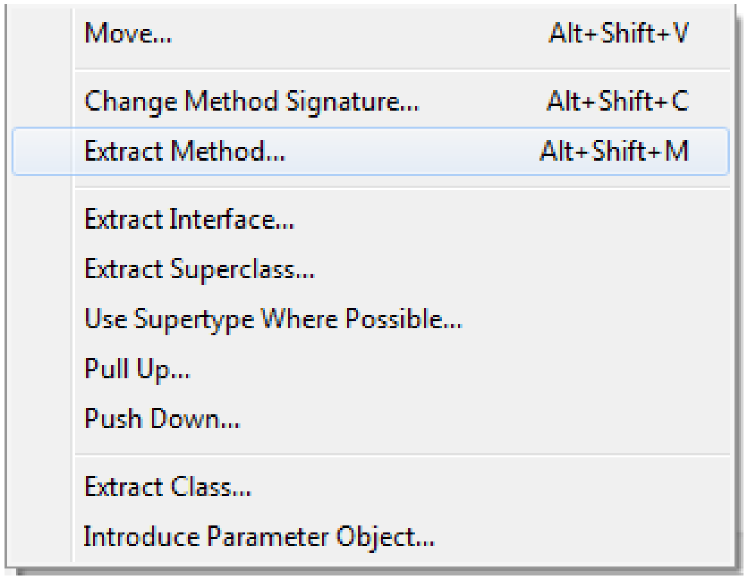
Benefits of using IDE supported refactoring:

✓ **Safe**: IDE performs cross checks and protects you from common mistakes

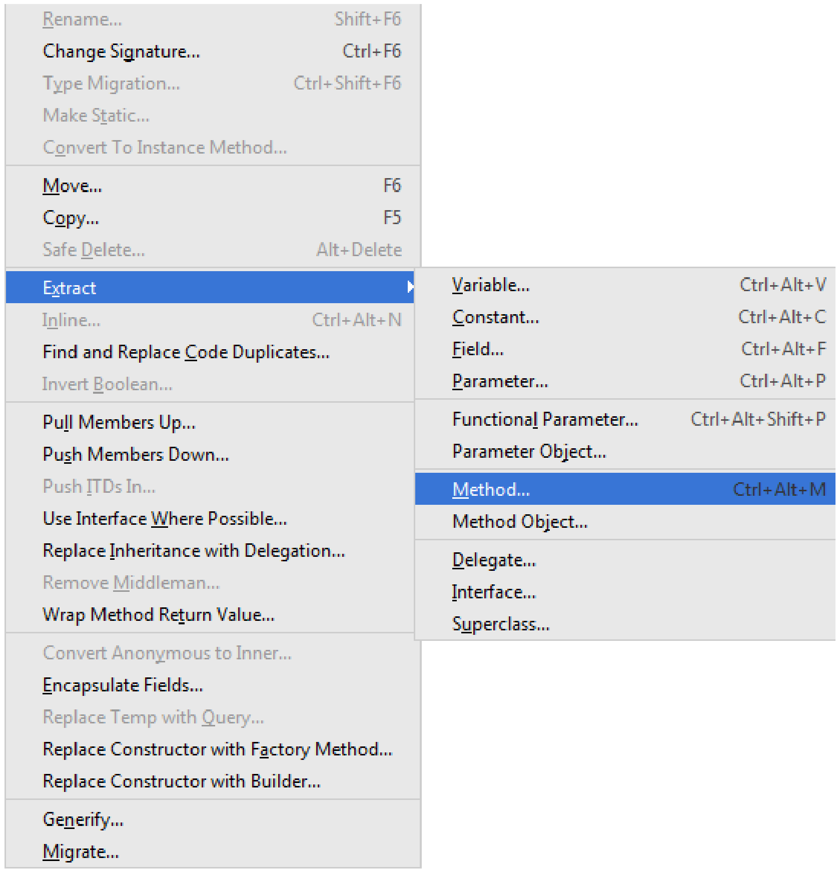
✓ **Fast**: Usually faster than manual refactoring

IDE refactoring menu examples

eclipse


Move... Alt+Shift+V
Change Method Signature... Alt+Shift+C
Extract Method... Alt+Shift+M
Extract Interface...
Extract Superclass...
Use Supertype Where Possible...
Pull Up...
Push Down...
Extract Class...
Introduce Parameter Object...

IntelliJ IDEA


Rename... Shift+F6
Change Signature... Ctrl+F6
Type Migration... Ctrl+Shift+F6
Make Static...
Convert To Instance Method...
Move... F6
Copy... F5
Safe Delete... Alt+Delete
Extract
Inline... Ctrl+Alt+N
Find and Replace Code Duplicates...
Invert Boolean...
Pull Members Up...
Push Members Down...
Push ITDs In...
Use Interface Where Possible...
Replace Inheritance with Delegation...
Remove Middleman...
Wrap Method Return Value...
Convert Anonymous to Inner...
Encapsulate Fields...
Replace Temp with Query...
Replace Constructor with Factory Method...
Replace Constructor with Builder...
Generify...
Migrate...
Variable... Ctrl+Alt+V
Constant... Ctrl+Alt+C
Field... Ctrl+Alt+F
Parameter... Ctrl+Alt+P
Functional Parameter... Ctrl+Alt+Shift+P
Parameter Object...
Method... Ctrl+Alt+M
Method Object...
Delegate...
Interface...
Superclass...

7. Know when you should not refactor

<u>Cases not suitable for refactoring</u>

- ✓ A product that is near end-of-life
- ✓ Code that is not going to be maintained, for example a tool that migrates data and will run only once

Remember the **economic justification**. In the cases above, refactoring is **an investment that will never pay off!**

Sources & Further Reading

- **Refactoring, improving the design of existing code by Martin Fowler**, ISBN 013306526X
- **Refactoring.com:** https://www.refactoring.com/
- **Martin Fowler blog:** https://martinfowler.com/tags/refactoring.html
- **Ron Jeffries, Refactoring – Not on the backlog!:** http://ronjeffries.com/xprog/articles/refactoring-not-on-the-backlog/
- **Martin Fowler, Workflows of Refactoring:** https://martinfowler.com/articles/workflowsOfRefactoring/

Refactoring Summary

<u>What?</u>

- ✓ Disciplined Technique for restructuring code
- ✓ Preserving external behavior
- ✓ Done in small steps (refactorings)

<u>Why?</u>

- ✓ Sustainable Delivery Pace
- ✓ Refactoring → Clean Code → Faster Delivery

Refactoring Summary

<u>How?</u>

✓ Always together with automated tests

✓ Work in “refactoring mode” (separate from “add function”)

✓ Mostly in opportunistic fashion

✓ Balance with feature delivery

✓ Driven by code smells

✓ Use your IDE capabilities

✓ Don’t do it if it’s not going to pay off

Questions & Discussion