Code refactoring

Maciej Koziara

Refactoring

A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.

Martin Fowler

Why change something that works?

// Replace old, hard to understand code with new one
// We know more than at the beginning of the project
// Move solutions to newer approaches / technologies
// Easier and faster application development
// Cheaper maintenance
// Way to pay the technical dept

Technical debt

Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. The danger occurs when the debt is not repaid.

Ward Cunningham

How to loan technical debt?

// Taking shortcuts
// Ignoring old code
// Premature optimization

Premature optimization is the root of all evil.

1. Avoid over-engineering (YAGNI - You Ain't Gonna Need It)
2. Avoid Gold-Plating your code

Refactoring process

1. Understand code
2. Verify existing behaviour
3. Write tests
4. Apply refactoring
5. Verify behaviour hasn’t changed

Boy scout rule

Code smells

Code that indicates some bigger problem or antipattern usage.

1. How to identify?
2. How to fix?
3. Why we want to refactor?

Code smell - Long Methods

1. Every method containing more than 20 lines should pay our attention
2. Extract to smaller methods
3. Usually long methods break Single Responsibility principle, are harder to read and reason about

Intellij Tip: Ctrl + Alt + M - extract method

Code smell - Long Methods

```
public static void main(String... args) {
    System.out.println("IIIIIIIII                ffffffffffffff
    System.out.println("I::::::::I              f::::::::::::::f
    System.out.println("I::::::::I             f::::::::::::::::f
    System.out.println("II::::::II             f::::::fffffff:::::f
    System.out.println("I::::Innnn  nnnnnnnn     f:::::f       ffffffoooo
    System.out.println("I::::In:::nn::::::::nn   f:::::f           oo:::::
    System.out.println("I::::In::::::::::::::nn f:::::::ffffff    o::::::
    System.out.println("I::::Inn:::::::::::::::nf::::::::::::f    o:::::oo
    System.out.println("I::::I  n:::::nnnn:::::nf::::::::::::f    o::::o
    System.out.println("I::::I  n::::n    n::::nf:::::::ffffff    o::::o
    System.out.println("I::::I  n::::n    n::::n f:::::f          o::::o
    System.out.println("I::::I  n::::n    n::::n f:::::f          o::::o
    System.out.println("II::::::IIn::::n    n::::nf:::::::f         o::::
    System.out.println("I::::::::In::::n    n::::nf:::::::f         o::::
    System.out.println("I::::::::In::::n    n::::nf:::::::f          oo:::
    System.out.println("IIIIIIIIInnnnnn    nnnnnnfffffffff            ooo

    System.out.println("Current date is " + LocalDateTime.now());

    System.out.println();
    System.out.println();
    System.out.println();
    System.out.println();

    System.out.println("1. Check available courses");
    System.out.println("2. Enroll to course");
    System.out.println("3. Check my course status");
    System.out.println("4. Exit");

    System.out.println();
    System.out.println();
    System.out.println();
    System.out.println();
```



```
public static void main(String... args) {
    printBanner();
    printCurrentDate();
    printEmptySection();
    printMenu();
    printEmptySection();
    printCurrentDate();
}
```

Code smell - Primitive Obsession

1. Representing domain objects by primitive types, constants usage instead of enums
2. Logically group primitives into their own class
3. Better understandability and organization of code

Intellij Tip: Ctrl + Alt + c - extract constance

Code smell - Primitive Obsession

```
private static void printReport(String city, long population) {
    System.out.println("Population of city " + city + " is " + population);
}
```

```
private static void printReport(City city) {
    System.out.println(String.format(REPORT, city.getName(), city.getPopulation(), city.getNumberOfCinemas()));
}
```

Code smell - Nulls overuse

1. Application flow based on null values returned from methods
2. Use optionals
3. Gives clean intentions to other programmers about possible method outcome

Intellij Tip: Ctrl + Alt + v - extract variable

Code smell - Nulls overuse

```
private static void printAssignedDoctor(Patient patient) {
    if (patient.getDoctor() != null) {
        System.out.println(String.format("Patient %s is assi
    } else {
        System.out.println(String.format("Patient %s does no
    }
}
```

```
private static void printAssignedDoctor(Patient patient) {
    Optional<Doctor> doctor = patient.getDoctor();
    if (doctor.isPresent()) {
        System.out.println(String.format("Patient %s is assi
    } else {
        System.out.println(String.format("Patient %s does nc
    }
}
```

Code smell - comments

1. Comments (not JavaDoc) in code
2. Refactor code so comments will be not necessary to understand it
3. Code becomes more intuitive and obvious.

Code smell - comments

```
// check if isbn contains only digits
if (!cleanIsbn.matches( regex: "[0-9]+")) {
    return false;
}
```

```
if (doesNotContainOnlyDigits(cleanIsbn)) {
    return false;
}
```

Code smell - Bad naming

1. Meaningless names for methods, variables etc.
2. Rename and give names more meaning
3. Code is easier to understand

```
int foo = sum(basket.items);
```

```
int totalPrice = countTotalPrice(basket.getItems());
```

Intellij Tip: Shift + F6 - rename

Code smell - Public fields

1. Classes with direct access to their internal values
2. Access fields via accessors methods
3. Better encapsulation

Code smell - Public fields

```
public class Basket {

    public List<Item> items;

    public Basket(List<Item> items) { this.items = items; }
}
```

```
public class Basket {

    private final List<Item> items;

    public Basket(List<Item> items) {
        this.items = items;
    }

    public List<Item> getItems() {
        return items;
    }
}
```

```
basket.items.size()
```

```
basket.getItems().size()
```

Code smell - Temp variables

1. Variable that exists only to store temporary result
2. Use streams with reduction or recursion
3. Code becomes more intuitive and obvious

Code smell - Temp variables

```
double discounted = foo;
if (foo > 200 || basket.items.size() > 5) {
    discounted = foo * 0.8;
}

return discounted;
```



```
if (shouldBeDiscounted(basket, totalPrice)) {
    return countDiscountedPrice(totalPrice);
}

return totalPrice;
```

Code smell - Complex if condition

1. Condition too complex to understand at first glance
2. Move to method with name explaining condition
3. Application flow becomes easier to understand

```
if (foo > 200 || basket.items.size() > 5) {
    discounted = foo * 0.8;
}
```

```
if (shouldBeDiscounted(basket, totalPrice)) {
    return countDiscountedPrice(totalPrice);
}
```

```
private boolean shouldBeDiscounted(Basket basket, int totalPrice) {
    return totalPrice > 200 || basket.getItems().size() > 5;
}
```

Code smell - ignoring language features

1. Usage of old approaches in new code
2. Change code so it will be using new features

Code smell - ignoring language features

```java
private int sum(List<Item> items) {
    int s = 0;

    for (Item i: items) {
        if (!i.isForFree) {
            s += i.price;
        }
    }

    return s;
}
```

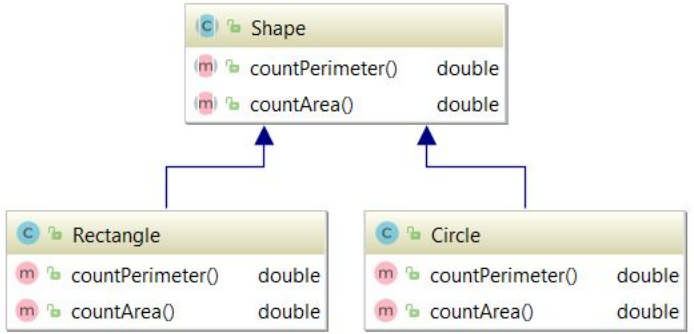
```java
private int countTotalPrice(List<Item> items) {
    return items.stream()
            .filter(i -> !i.isForFree())
            .mapToInt(Item::getPrice)
            .sum();
}
```

Code smell - Switches

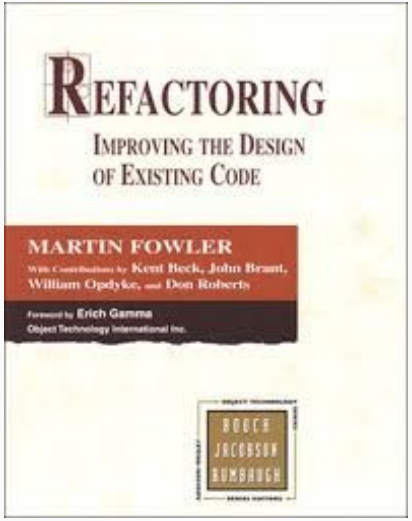
1. Use switches instead of OOP design
2. Move from switch usage to one abstract class and several implementations
3. Code become more concise, it’s easier to add new class implementations than switch cases

Code smell - Switches

```
public double calculateArea(Shape shape) {
    switch (shape.getType()) {
        case CIRCLE:
            return shape.getRadius() * shape.getRadius() * Math.PI;
        case RECTANGLE:
            return shape.getWidth() * shape.getHeight();
        default:
            throw new RuntimeException("Unsupported shape: " + shape.getType());
    }
}
```



Worth to check

		
Robert C. Martin Clean Code	**Martin Fowler** Refactoring	https://refactoring.guru/