Code Smells

by Roman Hrabovskyi & Andrii Yatsura

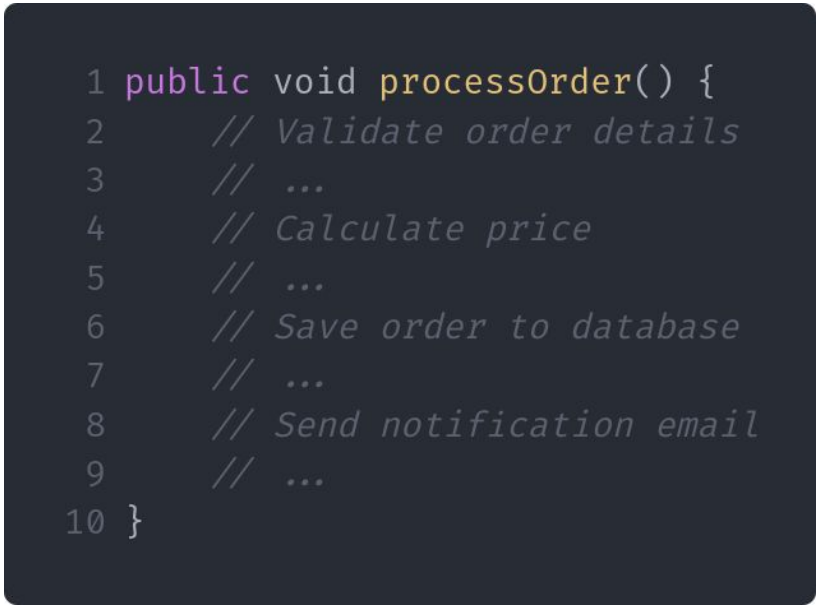
What is *a code smell*?

Code smells defined as patterns indicating deeper issues.
They are not bugs, but precursors to much deeper problems.

Common Code Smells

Long Method

- Difficult to understand
- Contains complex logic
- Hard to test and reuse in other parts of the project



```
public void processOrder() {
    // Validate order details
    // ...
    // Calculate price
    // ...
    // Save order to database
    // ...
    // Send notification email
    // ...
}
```

—

```
public void processOrder() {
    // Validate order details
    // ...
    // Calculate price
    // ...
    // Save order to database
    // ...
    // Send notification email
    // ...
}
```

```
public void processOrder() {
    validateOrderDetails();
    calculatePrice();
    saveOrderToDatabase();
    sendNotificationEmail();
}

private void validateOrderDetails() { /* ... */ }
private void calculatePrice() { /* ... */ }
private void saveOrderToDatabase() { /* ... */ }
private void sendNotificationEmail() { /* ... */ }
```

Refactored

Large Class

- Similar to Long Method
- Class contains too many attributes and methods

```
class User {
    // User login methods
    // User registration methods
    // Profile management methods
    // Notification settings methods
}
```

—

```
class User {
    // User login methods
    // User registration methods
    // Profile management methods
    // Notification settings methods
}
```

```
class User {
    private LoginManager loginManager;
    private RegistrationManager registrationManager;
    private ProfileManager profileManager;
    private NotificationManager notificationManager;
}

class LoginManager { /* ... */ }
class RegistrationManager { /* ... */ }
class ProfileManager { /* ... */ }
class NotificationManager { /* ... */ }
```

Refactored

Primitive Obsession

- Using primitive data types instead of small objects
- Use of constants

```
class Order {
    private String customerName; // Primitive type
    private String address;
    private String phoneNumber;
}
```

—

```
class Order {
    private String customerName; // Primitive type
    private String address;
    private String phoneNumber;
}
```

```
class Order {
    private Customer customer; // Using object types
    private Address address;
    private PhoneNumber phoneNumber;
}

class Customer { /* ... */ }
class Address { /* ... */ }
class PhoneNumber { /* ... */ }
```

Refactored

Feature Envy

- Methods tend to rely on other classes
- High level of coupling
- Hard to maintain

```
class Order {
    private Customer customer;

    public void calculateDiscount() {
        double discount = customer.getDiscountRate() * customer.getOrderHistory().size();
    }
}
```

```
class Order {
    private Customer customer;

    public void calculateDiscount() {
        double discount = customer.getDiscountRate() * customer.getOrderHistory().size();
    }
}
```

```
class Order {
    private Customer customer;

    public void calculateDiscount() {
        double discount = customer.calculateDiscount();
    }
}

class Customer {
    public double calculateDiscount() {
        return getDiscountRate() * getOrderHistory().size();
    }
    // Other methods
}

```

Refactored

Duplicated Code

- Repeated code
- Changes have to be made in a lot of places in code

```
class ReportGenerator {
    public void generateSalesReport() {
        // Prepare data
        // Format report
        // Print report
    }

    public void generateCustomerReport() {
        // Prepare data
        // Format report
        // Print report
    }
}
```

—

```
class ReportGenerator {
    public void generateSalesReport() {
        // Prepare data
        // Format report
        // Print report
    }

    public void generateCustomerReport() {
        // Prepare data
        // Format report
        // Print report
    }
}
```

```
class ReportGenerator {
    public void generateReport(String reportType) {
        prepareData(reportType);
        formatReport(reportType);
        printReport(reportType);
    }

    private void prepareData(String reportType) { /* ... */ }
    private void formatReport(String reportType) { /* ... */ }
    private void printReport(String reportType) { /* ... */ }
}
```

Refactored

Data Clumps

- Poor data organization
- Weak relations between objects

```
class Order {
    private String itemName;
    private int quantity;
    private double price;
}
```

—

```
class Order {
    private String itemName;
    private int quantity;
    private double price;
}
```

```
class Order {
    private OrderItem item;
}

class OrderItem {
    private String name;
    private int quantity;
    private double price;
}
```

Refactored

Identifying and fixing code smells

- Code reviews
- Code refactoring
- Refactoring tools
- Testing and metrics

What is *code refactoring*?

Code refactoring means restructuring code without modifying its behavior. Its aim is improving code quality through reducing complexity, increasing readability, etc.

Common refactoring strategies

- Extract method
- Inline method
- Extract class
- Move/rename method
- Replace conditionals with polymorphism
- Decompose conditional

Thanks for your attention