Методи рефакторингу коду програмного забезпечення

Рефакторинг — це процес перетворення внутрішньої структури програмного забезпечення без зміни зовнішньої поведінки. Метою рефакторингу є покращення якості коду, зростання його читабельності, гнучкості та продуктивності.

Головний принцип рефакторингу — залишити зовнішню поведінку програми (функціональність) незмінною, зосередившись на внутрішній структурі.

Підготував ст. гр. ПЗПІ-22-4
Попов Богдан Сергійович

Заміна вкладених умовних операторів на захисні умовні блоки

Вкладені умовні оператори

Вкладені умовні оператори ускладнюють читання та розуміння коду. Їхнє читання може бути стомлюючим, а внесення змін - ризикованим.

Захисні умовні блоки

Захисні умовні блоки дозволяють вийти з функції, якщо умова не виконується, зробивши код простішим та легшим для розуміння.

Код до рефакторингу

```
function calculateDiscount(user) {
    if (user) {
        if (user.isPremium) {
            if (user.purchaseAmount > 1000) {
                return user.purchaseAmount * 0.2;
            }
        }
    }
    return 0;
}
```

Код після рефакторингу

```
function calculateDiscount(user) {
    if (!user) return 0;
    if (!user.isPremium) return 0;
    if (user.purchaseAmount <= 1000) return 0;
    return user.purchaseAmount * 0.2;
}
```

Інкапсуляція колекції

Відкрита колекція

Відкрита колекція може бути модифікована будь-якою частиною коду, що може призвести до непередбачуваних побічних ефектів.

Інкапсуляція

Інкапсуляція колекції приховує її внутрішню реалізацію, захищаючи її від змін та забезпечуючи централізований доступ до даних.

Код до рефакторингу

```
class Classroom {
    constructor() {
        this.students = [];
    }
}

const classroom = new Classroom();
classroom.students.push("John");
classroom.students.push("Jane");
```

Код після рефакторингу

```
class Classroom {
    constructor() {
        this._students = [];
    }

    addStudent(student) {
        this._students.push(student);
    }

    getStudents() {
        return [...this._students];
    }
}

const classroom = new Classroom();
classroom.addStudent("John");
classroom.addStudent("Jane");
```

Заміна умовної логіки поліморфізмом

Умовна логіка

Умовна логіка робить код громіздким та складним для підтримки. Додавання нових типів може вимагати модифікації умовних операторів.

Поліморфізм

Поліморфізм дозволяє використовувати різні типи об'єктів з одним і тим же інтерфейсом, що робить код більш гнучким та масштабованим.

Код до рефакторингу

```
function getShippingCost(order) {
    switch (order.type) {
        case "standard":
            return 5;
        case "express":
            return 10;
        case "overnight":
            return 20;
        default:
            throw new Error("Unknown order type");
    }
}
```

Код після рефакторингу

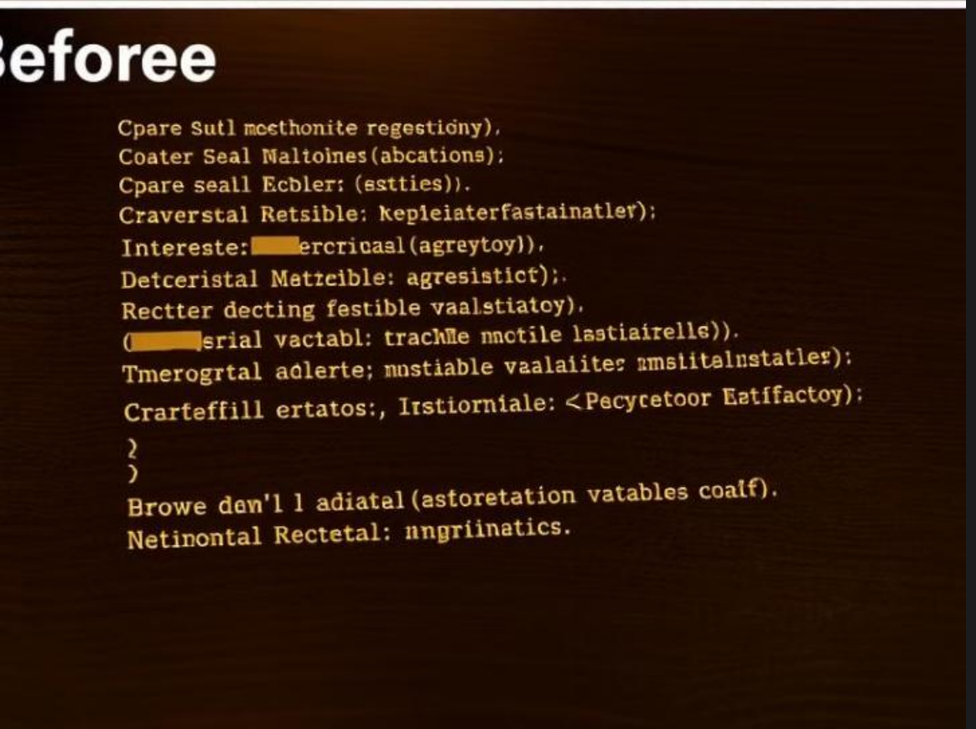
```
class Order {
    getShippingCost() {
        throw new Error("Must be implemented in subclass");
    }
}

class StandardOrder extends Order {
    getShippingCost() {
        return 5;
    }
}

class ExpressOrder extends Order {
    getShippingCost() {
        return 10;
    }
}

class OvernightOrder extends Order {
    getShippingCost() {
        return 20;
    }
}

const order = new ExpressOrder();
console.log(order.getShippingCost()); // 10
```



Покращення читабельності та підтримки коду

1 Зменшення складності

Рефакторинг спрощує код, роблячи його легшим для розуміння та модифікації.

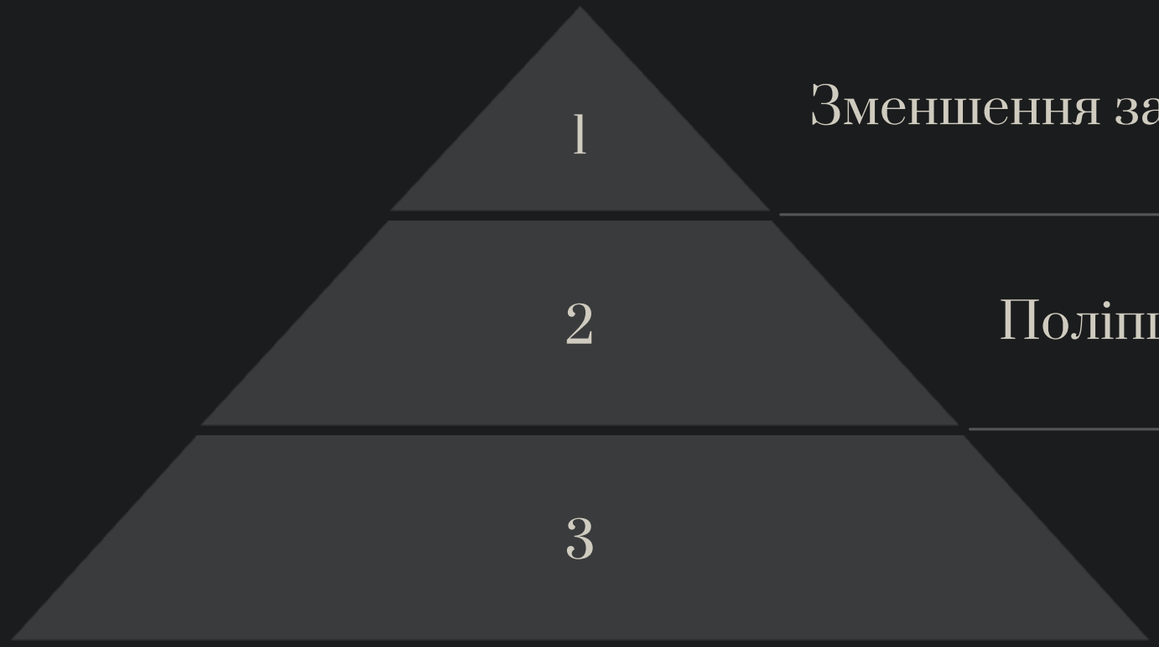
2 Зменшення кількості помилок

Простий код зменшує ймовірність помилок та полегшує їх виправлення.

3 Зменшення технічного боргу

Рефакторинг коду покращує його якість, знижуючи технічний борг, накопичений з часом.

Підвищення гнучкості та масштабованості додатку



Зменшення залежностей

Поліпшення повторного використання коду

Збільшення гнучкості та масштабованості

Переваги рефакторингу коду

1

Зменшення часу розробки

Рефакторинг робить код легшим для модифікації, скорочуючи час розробки та впровадження нових функцій.

2

Зменшення вартості розробки

Простий код легше підтримувати та вносити зміни, що зменшує загальну вартість розробки.

3

Зменшення ризику помилок

Простий код зменшує ймовірність помилок, покращуючи стабільність і надійність програми.

Підсумки та ключові висновки

Рефакторинг — це важливий процес, який покращує якість коду та підвищує гнучкість та масштабованість програмного забезпечення. Він дає змогу створити більш надійний, підтримуваний та зрозумілий код, що веде до зменшення ризиків та скорочення часу розробки.

Список використаних джерел

- Фаулер Мартін. «Refactoring: Improving the Design of Existing Code»
- Фаулер Мартін. «Refactoring: Improving the Design of Existing Code» (2-е видання)
- Мартін Роберт. «Clean Code: A Handbook of Agile Software Craftsmanship»
- Хант Ендрю та Томас Девід. «The Pragmatic Programmer»