REFACTORING METHODS

Методи рефакторингу

Extract Function, Extract Variable, Replace Nested Conditional with Guard Clauses

Виділення функції, виділення змінної, заміна вкладених умов захисними виразами

Виконав ст. гр. ПЗПІ-22-1 Токар Денис

EXTRACT FUNCTION

Виділення функції

Мета: Виділення фрагмента коду в окрему функцію для підвищення читабельності та модульності.

Коли використовувати: Коли блок коду виконує одну логічну операцію, повторюється в різних місцях або занадто великий.

Переваги:

- Збільшує читабельність, роблячи код більш зрозумілим.
- Спрощує тестування, оскільки кожна функція виконує одну конкретну задачу.
- Зменшує дублювання коду.
- Покращує підтримуваність, оскільки зміни можна вносити в одному місці.

ПРИКЛАД EXTRACT FUNCTION

Виділення функції

```
function calculateOrderTotal(order) {
    const subtotal = order.items.reduce((total, item) => total + item.price * item.quantity, 0);
    const tax = subtotal * 0.08;
    const shipping = order.shippingMethod === 'express' ? 10 : 5;
    return subtotal + tax + shipping;
}

// Після рефакторингу
function calculateOrderTotal(order) {
    return calculateSubtotal(order) + calculateTax(order) + calculateShipping(order);
}

function calculateSubtotal(order) {
    // ...
}

function calculateTax(order) {
    // ...
}

function calculateShipping(order) {
    // ...
}
```

EXTRACT VARIABLE

Виділення змінної

Мета: Виділення складного виразу або результату проміжного обчислення в окрему змінну з описовим ім'ям.

Коли використовувати: Коли вираз використовується кілька разів в коді, або коли він занадто складний для розуміння.

Переваги:

• Покращує читабельність, роблячи код більш зрозумілим.

• Спрощує розуміння логіки коду.

• Зменшує ймовірність помилок при введенні виразу.

ПРИКЛАД EXTRACT VARIABLE

Виділення змінної

```
// Перед рефакторингом
if (user.role === 'admin' && user.isActive) {
    // ...
}

// Після рефакторингу
const isAdminAndActive = user.role === 'admin' && user.isActive;
if (isAdminAndActive) {
    // ...
}
```

```
// Перед рефакторингом
if (customer.isGoldMember && order.total > 1000) {
    // ...
}

// Після рефакторингу
const isEligibleForDiscount = customer.isGoldMember && order.total > 1000;
if (isEligibleForDiscount) {
    // ...
}
```

REPLACE NESTED CONDITIONAL WITH GUARD CLAUSES

Заміна вкладених умов захисними виразами

Мета: Заміна глибоко вкладених умовних конструкцій на послідовність простих умов, які перевіряються на початку функції.

Коли використовувати: Коли умовні конструкції стають занадто складними і утрудняють розуміння коду.

Переваги:

- Покращує читабельність, роблячи код більш лінійним.
- Спрощує логіку програми.
- Зменшує кількість вкладеностей.

ПРИКЛАД

REPLACE NESTED CONDITIONAL WITH GUARD CLAUSES

Заміна вкладених умов захисними виразами

```
// Перед рефакторингом
if (user.type === 'customer') {
    if (user.isPremium) {
        // ...
    } else {
        // ...
    }
} else if (user.type === 'admin') {
    // ...
}

// Після рефакторингу
if (user.type !== 'customer') {
    // ...
}

if (user.type === 'customer' && !user.isPremium) {
    // ...
}

// Решта коду для преміум-клієнтів
```

```
// Перед рефакторингом
if (customer.isGold) {
    if (order.total > 1000) {
        // ...
    } else {
        // ...
    }
} else {
    // ...
}

// Після рефакторингу
if (!customer.isGold) {
    // ...
}

if (order.total <= 1000) {
    // ...
}

// Решта коду для золотих клієнтів з великими замовленнями
```

ВИСНОВОК

- Використання наведених методів рефакторингу є важливою практикою в сучасній розробці програмного забезпечення, яка допомагає підтримувати код у хорошому стані та полегшує його подальшу модифікацію.

ДЯКУЮ ЗА УВАГУ!