

Міністерство освіти та науки України
Харківський національний університет радіоелектроніки

Кафедра ПІ

Звіт

З практичної роботи 2

Тема роботи: «Методи рефакторингу коду програмного
забезпечення»

з дисципліни «Аналіз та рефакторинг коду»

Виконав:

ст. гр. ПЗП-22-10

Клецов М.Д.

Перевірив:

ст. викладач Сокорчук І.П.

Харків 2024

1. Мета роботи:

Навчити студентів основним методам рефакторингу коду на основі реальних прикладів з їхніх власних програмних проєктів. Студенти повинні навчитися ідентифікувати проблеми в коді та використовувати відповідні методи рефакторингу для покращення його якості.

2. Завдання

Студент повинен обрати 3 методи рефакторингу з книги Мартін Р. Чистий код: створення і рефакторинг за допомогою AGILE. – ФАБУЛА, 2019. – 416 с. Навести приклади свого особистого коду (з курсових, лабораторних або інших проєктів), який потребує покращення, та продемонструвати застосування обраних методів. 3).

Кожен метод рефакторингу повинен супроводжуватись:

Описом проблеми, яку вирішує даний метод.

Кодом до і після застосування методу рефакторингу.

Поясненням переваг використаного методу.

3. Хід роботи

Було обрано 3 методи рефакторингу. У презентації (Додаток Б) наведено приклади свого особистого коду, який потребує покращення, та застосування обраних методів.

Висновки

У процесі виконання практичної роботи були вивчені три методи рефакторингу з книги Мартіна Роберта "Чистий код", які допомогли покращити якість коду. Для кожного методу було продемонстровано реальні приклади з особистих проєктів, що дозволило зрозуміти, як правильно ідентифікувати проблеми в коді та застосовувати відповідні методи для їх

вирішення. Рефакторинг значно підвищує зрозумілість, ефективність та підтримуваність програмного коду.

В

ДОДАТОК А

Програмний код, використаний як приклад у презентації.

```
function printOwing(invoice) {
  let outstanding = 0;

  console.log("*****");
  console.log("**** Customer Owes ****");
  console.log("*****");

  for (const o of invoice.orders) {
    outstanding += o.amount;
  }

  const today = new Date();
  invoice.dueDate = new Date(today.getFullYear(), today.getMonth(),
today.getDate() + 30);

  console.log(`name: ${invoice.customer}`);
  console.log(`amount: ${outstanding}`);
  console.log(`due: ${invoice.dueDate.toLocaleDateString()}`);
}

let defaultOwner = { firstName: "John", lastName: "Doe" };

const owner = defaultOwner;
console.log(owner.firstName, owner.lastName);

defaultOwner = { firstName: "Jane", lastName: "Smith" };

function calculateArea(radius) {
  return 3.14159 * radius * radius;
}

function calculateDiscount(price) {
  return price * 0.1; // 10% знижка
}
```

ДОДАТОК Б

Презентація на тему «Методи рефакторингу коду програмного забезпечення».

Методи рефакторингу коду програмного забезпечення

КЛЕЦОВ МИКИТА ПЗП І - 22 - 10

Вступ

Що таке рефакторинг?

Це процес покращення існуючого коду без зміни його зовнішньої поведінки.

Чому це важливо?

Підвищення читабельності

Спрощення розширення та підтримки.

Зменшення кількості помилок.

Обрані методи:

1. Extract Method
2. Encapsulate Variable
3. Replace Magic Number with Constant

Метод 1 - Extract Method

- **Проблема:**

Код містить великий блок логіки, який виконує кілька завдань одночасно. Це ускладнює читання, тестування та повторне використання коду.

- Що робить код: Функція `printOwing (invoice)` генерує звіт про заборгованість клієнта. Спочатку вона виводить заголовок, потім обчислює загальну суму боргу клієнта, проходячи всі його замовлення. Далі встановлює дату погашення боргу (на 30 днів від поточної дати) і виводить ім'я клієнта, суму боргу та дату оплати.

Код до рефакторингу

```
function printOwing (invoice) {
  let outstanding = 0;

  console.log("*****");
  console.log("**** Customer Owes ****");
  console.log("*****");

  // Calculate outstanding
  for (const o of invoice.orders) {
    outstanding += o.amount;
  }

  // Record due date
  const today = new Date();
  invoice.dueDate = new Date(today.getFullYear(),
    today.getMonth(), today.getDate() + 30);

  // Print details
  console.log(`name: ${invoice.customer}`);
  console.log(`amount: ${outstanding}`);
  console.log(`due:
    ${invoice.dueDate.toLocaleDateString()}`);
}
```

Код після рефакторингу

- **Переваги :**

- **Зрозумілість :** Розбиття коду на функції робить його читабельнішим.
- **Повторне використання :** Методами можна користуватися в інших місцях програми.
- **Тестування :** Легше перевіряти окремі функції, ніж один великий блок коду.

```
function printOwing (invoice) {
  printBanner ();

  const outstanding =
    calculateOutstanding (invoice);

  recordDueDate (invoice);

  printDetails (invoice, outstanding);
}
```

Метод 2 - Encapsulate Variable

- **Проблема:**

Змінна використовується безпосередньо, що ускладнює контроль її змін і зменшує гнучкість коду.

- Що робить код: Цей код містить глобальну змінну `defaultOwner`, яка зберігає дані власника за замовчуванням. Потім код отримує доступ до цієї змінної, використовуючи її напряму. Також дозволяється змінювати її значення, що може призвести до небажаних побічних ефектів у програмі.

Код до рефакторингу

```
let defaultOwner = { firstName : "John", lastName : "Doe" };
```

// Використання змінної

```
const owner = defaultOwner ;  
  
console.log( owner.firstName , owner.lastName );
```

// Зміна змінної

```
defaultOwner = { firstName : "Jane", lastName : "Smith" };
```

Код після рефакторингу

- Переваги :

- Контроль доступу: Змінна більше не змінюється напряму .
- Незмінність : Копіюючи дані , ми уникаємо неочікуваних змін .
- Гнучкість : Логіку отримання або зміни значення можна змінювати централізовано .

Код після рефакторингу

```
let defaultOwnerData = { firstName : "John",  
lastName : "Doe" };
```

```
function getDefaultOwner () {  
  
    return { ... defaultOwnerData }; // Повертаємо копію  
  
}
```

```
function setDefaultOwner (newOwner) {  
  
    defaultOwnerData = { ... newOwner }; // Записуємо копію  
  
}
```

Метод 3 - Replace Magic Number with Constant

- **Проблема:**

У коді використовуються "магічні числа", що ускладнює його розуміння та підтримку.

- **Що робить код (до**

рефакторингу): Функція

`calculateArea (radius)` обчислює площу кола, використовуючи значення `3.14159` напряму. Функція

`calculateDiscount (price)` повертає ціну зі знижкою `10%`, але значення `0.1` не пояснюється в коді, що робить його менш зрозумілим.

- Код до рефакторингу

```
function calculateArea (radius) {  
    return 3.14159 * radius * radius;  
}
```

```
function calculateDiscount (price) {  
    return price * 0.1; // 10% знижка  
}
```

Код після рефакторингу

Що покращив рефакторинг:

- **Зрозумілість** : тепер числа мають змістовні імена (`PI`, `DISCOUNT_RATE`), що робить код більш читабельним.
- **Легкість змін** : якщо значення `PI` або `DISCOUNT_RATE` зміниться, його потрібно буде оновити лише в одному місці.
- **Уникнення помилок** : використання констант зменшує ризик випадкової зміни значення або неправильного його використання.

```
const PI = 3.14159;
```

```
const DISCOUNT_RATE = 0.1;
```

```
function calculateArea (radius) {  
    return PI * radius * radius;  
}
```

```
function calculateDiscount (price) {  
    return price * DISCOUNT_RATE;  
}
```


Висновок

- Extract Method (Виділення методу)
- Зменшує складність : Великий метод розбитий на кілька дрібніших , кожен з яких виконує одну конкретну задачу.
- Покращує читабельність : Тепер код легше розуміти , оскільки його структура чітко відображає логіку роботи .
- Сприяє повторному використанню : Виділені методи (printBanner , calculateOutstanding , recordDueDate , printDetails) можна використовувати в інших частинах програми .
- Полегшує тестування : Окремі методи можна тестувати окремо , що спрощує виявлення помилок .
- Encapsulate Variable (Інкапсуляція змінної)
- Забезпечує контроль доступу : Тепер змінна не змінюється безпосередньо , а доступ до неї здійснюється через геттери та сеттери .
- Запобігає випадковим змінам : Користувач отримує копію об'єкта , що виключає ризик випадкових змін вихідних даних .
- Гнучкість і розширюваність : У майбутньому можна легко змінити логіку роботи з даними , не змінюючи код у багатьох місцях .
- Replace Magic Number with Constant (Замінення магічних чисел на константи)
- Покращує читабельність : Тепер числа (3.14159, 0.1) мають змістовні імена (PI, DISCOUNT_RATE), що робить код зрозумілішим .
- Полегшує підтримку : Якщо потрібно змінити значення константи , достатньо зробити це в одному місці , а не у всьому коді .
- Запобігає помилкам : Використання іменованих констант допомагає уникнути неправильного використання чисел.