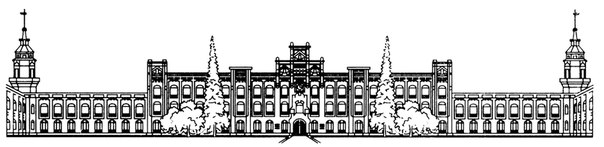
МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

Національний технічний університет України

"Київський політехнічний інститут імені Ігоря Сікорського"



Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Програмування. Частина 2. Основи програмування

Практична робота №7

Тема: Шаблон «Стратегія»

Варіант №11

Виконав: Коваль Богдан

студент групи ІС-31

Перевірив:

Майер І.С.

Київ-2024

**Тема:** Шаблон «Стратегія»

**Хід роботи:**

1. Повторити теоретичні відомості

2. Провести рефакторинг свого коду з лабораторної роботи No6

- використати шаблон «Стратегія»

- дотримуватись принципів SOLID

3. Додати реалізацію ще одного алгоритму сортування на свій вибір

- крім використання стандартних методів реалізуючих сортування

4. Відповісти на контрольні запитання

**Відповіді на контрольні питання**

1. **Патерни проектування** - це типові рішення для типових проблем, що виникають під час розробки програмного забезпечення. Вони є своєрідними шаблонами, які можна використовувати для вирішення архітектурних проблем. Патерни допомагають створювати гнучкі, підтримувані та розширювані системи шляхом надання вже перевірених та ефективних рішень для конкретних проблем.
2. **Шаблон "Стратегія"** використовується для визначення сімейства алгоритмів, інкапсулювання кожного з них та робить їх взаємозамінними. Цей шаблон дозволяє вибирати алгоритм під час виконання програми без зміни клієнтського коду.
3. **SOLID** - це набір принципів об'єктно-орієнтованого програмування, які спрямовані на створення більш гнучких, підтримуваних та розширюваних систем. Ось їх суть:

**Принцип єдиного обов'язку (Single Responsibility Principle)**: Клас повинен мати лише одину причину для зміни.

**Принцип відкритості/закритості (Open/Closed Principle)**: Сутності програми повинні бути відкритими для розширення, але закритими для зміни.

**Принцип підстановки Лісков (Liskov Substitution Principle)**: Об'єкти базового класу повинні бути замінними об'єктами підтипу базового класу без втрати коректності програми.

**Принцип інтерфейсу розділення (Interface Segregation Principle)**: Клієнти не повинні залежати від інтерфейсів, які вони не використовують.

**Принцип зв'язності (Dependency Inversion Principle)**: Класи мають залежати від абстракцій, а не від конкретних реалізацій.

1. **Принцип єдиної відповідальності (Single Responsibility Principle)**: Проблема: Клас виконує більше однієї функції або відповідає за кілька різних аспектів програми.

public class User

{

public string Name { get; set; }

public string Email { get; set; }

public void SaveToDatabase()

{

// Логіка для збереження користувача в базу даних

}

public void SendEmailNotification()

{

// Логіка для відправки email-повідомлення користувачу

}

}

**Принцип відкритості/закритості (Open/Closed Principle)**: Проблема: Клас важко або неможливо розширити без модифікації його вихідного коду.

public class Circle

{

public double Radius { get; set; }

public double Area()

{

return Math.PI \* Radius \* Radius;

}

}

**Принцип заміщення Лісков (Liskov Substitution Principle)**: Проблема: Підкласи не можуть замінити свої базові класи без зміни правильності програми.

public class Rectangle

{

public virtual int Width { get; set; }

public virtual int Height { get; set; }

public int Area()

{

return Width \* Height;

}

}

public class Square : Rectangle

{

public override int Width

{

set { base.Width = base.Height = value; }

}

public override int Height

{

set { base.Width = base.Height = value; }

}

}

**Принцип інверсії залежностей (Dependency Inversion Principle)**: Проблема: Класи залежать від конкретних реалізацій інших класів замість абстракцій.

public class Client

{

private readonly DatabaseConnection \_databaseConnection;

public Client()

{

\_databaseConnection = new DatabaseConnection();

}

public void DoSomething()

{

\_databaseConnection.Connect();

// Виконати дії з базою даних

\_databaseConnection.Disconnect();

}

}

public class DatabaseConnection

{

public void Connect()

{

// Логіка для з'єднання з базою даних

}

public void Disconnect()

{

// Логіка для відключення від бази даних

}

}

1. **Компонент** - це логічно об'єднана частина програмного забезпечення, яка може бути розглянута окремо від інших частин системи. Компоненти можуть бути бібліотеками, модулями, класами або навіть окремими функціями.

**Залежність** - це відношення між двома компонентами, де зміна в одному компоненті може вплинути на інший. Наприклад, якщо компонент A використовує функціональність компонента B, то компонент A залежить від компонента B.

1. Цей принцип стверджує, що компоненти програми повинні явно зазначати свої залежності. Це означає, що залежності не повинні бути прихованими або неявними, а замість цього повинні бути передані як параметри конструктора або методу, або ж зазначені через інші механізми ін'єкції залежностей.
2. Переваги і недоліки явного (explicit) і неявного (implicit) використання залежностей: **Переваги явного використання залежностей:**

* Чіткість і зрозумілість коду: Явне передача залежностей робить код більш зрозумілим і простим для розуміння.
* Легкість тестування: Явне використання залежностей дозволяє легко створювати тести, замінюючи реальні залежності на їхні замінники (моки).
* Зменшення зв'язності: Класи стають менш залежними один від одного, оскільки їхні залежності передаються явно.

**Недоліки неявного використання залежностей**:

* Складність зрозуміння і підтримки коду: Неявне використання залежностей може призвести до складності зрозуміння та відладки коду, оскільки залежності приховані.
* Утруднене тестування: Тести можуть бути складними для написання через неявність залежностей і можуть вимагати великої кількості моків.

1. **Зв'язність (Coupling)**:

Зв'язність визначає ступінь залежності між різними компонентами або класами програми. Висока зв'язність означає, що зміни в одному компоненті можуть вимагати змін в інших компонентах.

**Пов'язаність (Cohesion)**:

Пов'язаність визначає, наскільки сильно пов'язані різні елементи в межах одного компонента або класу. Висока пов'язаність вказує на те, що всі елементи класу пов'язані між собою та виконують одну конкретну задачу або функцію.