Лабораторна робота №7. Основи роботи з класами в C#

Мета роботи: Ознайомитися з поняттям класів у C#, їх створенням, властивостями, методами, конструкторами та об’єктами. Навчитися використовувати класи для моделювання об’єктів, організації даних і реалізації логіки в програмі.

Теоретичні відомості

Клас у C# — це користувацький тип даних, який використовується для моделювання реальних об’єктів і явищ. Він дозволяє об’єднувати дані (поля) і поведінку (методи) в одну логічну одиницю. Класи є основою об’єктно-орієнтованого програмування (ООП) і підтримують такі ключові принципи, як інкапсуляція, успадкування та поліморфізм. Клас оголошується за допомогою ключового слова class. Він може містити:

- Поля — змінні, які зберігають дані.
- Властивості — спеціальні методи для доступу до полів.
- Методи — функції для виконання певних операцій.
- Конструктори — спеціальні методи для ініціалізації об’єкта.

```
class Student
{
    // Поля
    private string name;
    private int age;

    // Властивості
    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    public int Age
    {
        get { return age; }
        set
        {
            if (value > 0)
                age = value;
            else
                throw new ArgumentException("Вік має бути позитивним.");
        }
    }

    // Конструктор
    public Student(string name, int age)
    {
        Name = name;
        Age = age;
    }

    // Метод
    public void DisplayInfo()
    {
```

```
        Console.WriteLine($"Ім'я: {Name}, Вік: {Age}");
    }
}
```

Об’єкт — це екземпляр класу. Він створюється за допомогою ключового слова new, яке викликає конструктор класу.

```
Student student = new Student("Іван", 20);
student.DisplayInfo(); // Виведе: Ім'я: Іван, Вік: 20
```

Інкапсуляція — це принцип, який забезпечує приховування внутрішньої реалізації класу від зовнішнього світу. Для контролю доступу до членів класу використовуються модифікатори доступу, такі як public та private. public дозволяє доступ з будь-якої частини програми, тоді як private обмежує доступ лише межами класу. Це важливий аспект інкапсуляції, який допомагає приховати внутрішню реалізацію класу від зовнішнього світу.

```
public class Student
{
    // Поле, доступне лише всередині класу
    private string name;

    // Поле, доступне з будь-якого місця програми
    public int Age;

    // Метод, доступний лише всередині класу
    private void PrintPrivateMessage()
    {
        Console.WriteLine("Це приватне повідомлення.");
    }

    // Метод, доступний з будь-якого місця програми
    public void PrintPublicMessage()
    {
        Console.WriteLine($"Студент: {name}, Вік: {Age}");
    }

    // Публічний метод для встановлення значення приватного поля
    public void SetName(string studentName)
    {
        name = studentName;
    }
}

// Використання
Student student = new Student();
student.Age = 20;               // Можна змінювати, оскільки поле публічне
student.SetName("Іван");        // Використання методу для встановлення приватного
значення
student.PrintPublicMessage();   // Виводить: Студент: Іван, Вік: 20
// student.name = "Іван";       // Помилка, оскільки поле приватне
// student.PrintPrivateMessage(); // Помилка, оскільки метод приватний
```

Окрім того, класи дозволяють використовувати властивості для спрощення доступу до приватних полів. Властивості поєднують гнучкість методів із синтаксисом полів. Наприклад, автоматичні властивості значно скорочують код для оголошення змінних, які мають геттери і сеттери.

```
public class Student
{
    private string name;
```

```
    private int age;

    // Властивість для доступу до імені
    public string Name
    {
        get { return name; }               // Повертає значення
        set { name = value; }              // Приймає значення
    }

    // Властивість із перевіркою в сетері
    public int Age
    {
        get { return age; }                // Повертає значення
        set
        {
            if (value > 0)                 // Перевірка перед встановленням
            {
                age = value;
            }
            else
            {
                Console.WriteLine("Вік не може бути менше 0!");
            }
        }
    }
}

// Використання
Student student = new Student();

// Встановлення значень через властивості
student.Name = "Марія";
student.Age = 18;

// Отримання значень через властивості
Console.WriteLine($"Ім'я: {student.Name}, Вік: {student.Age}");

// Спроба встановити недопустиме значення
student.Age = -5; // Виведе: "Вік не може бути менше 0!"
```

Статичні поля, властивості та методи належать класу, а не його екземпляру. Їх можна викликати без створення об’єкта. Приклад статичного методу:

```
class MathHelper
{
    public static int Add(int a, int b)
    {
        return a + b;
    }
}

// Виклик без створення об'єкта
int result = MathHelper.Add(5, 3);
Console.WriteLine(result); // 8
```

Зміст роботи

Згідно з обраним у лабораторній роботі 5 варіантом теми, необхідно виконати наступну дії. Всі дії відбуваються в репозиторії OOPWPFProject.

Завдання 1: Злити гілку feature/ connect-struct в основну гілку (master або main)

1.1 Створити нові гілку

```
git checkout -b feature/add-class
```

Завдання 2: Створення класу. Згідно з обраною темою описати поля класу. Необов'язкові поля зробити nullable.

Завдання 3: Додати до класу конструктори:

3.1 Конструктор за замовчуванням (без параметрів).

3.2 Конструктор із параметрами для ініціалізації всіх обов'язкових полів.

3.3 Реалізувати перевірку значень, щоб уникнути некоректного введення даних.

Завдання 4: Додати до класу методи:

4.1 DisplayInfo() — метод для форматованого виведення інформації про об'єкт.

4.2 Clone() — метод для створення копії об'єкта.

Завдання 5: Створити список об'єктів класу. Використовувати ObservableCollection для зберігання об'єктів класу.

Завдання 6: Реалізувати метод AddToList(), який приймає дані з форми (наприклад, через параметри) та додає новий об'єкт класу до списку. Викликати цей метод у відповідному обробнику події кнопки "Додати".

Завдання 7: Підключення списку класів до таблиці:

7.1 Забезпечити прив'язку списку до компонента DataGrid (або іншого табличного компонента).

7.2 Оновлення таблиці має відбуватися автоматично при зміні списку.

Завдання 8: Реалізувати обробник події для кнопки "Видалити". Видаляти виділений елемент із таблиці та списку.

Завдання 9: Реалізація сортування:

9.1 Додати можливість сортувати список об'єктів за кожним із полів.

9.2 Використовувати ComboBox для вибору параметра сортування (наприклад, за іменем, групою, датою народження).

9.3 Реалізувати сортування за допомогою методу OrderBy або OrderByDescending.

Завдання 10: Створити коміт та завантажити зміни в віддалений репозиторій.

Контрольні запитання

1. Що таке клас у C#, і як він відрізняється від структури?

2. Як оголошується клас у C#, і які модифікатори доступу можна використовувати для класів?
3. Що таке об’єкт класу, і як він створюється?
4. Яка різниця між полями, властивостями та методами у класах?
5. Що таке конструктор у C#? Які типи конструкторів підтримуються у класах і структурах?
6. Як оголосити метод у класі, і які правила іменування слід дотримуватись?
7. Як властивості (get/set) забезпечують доступ до приватних полів у класах і структурах?
8. Як передати параметри в метод класу? Що таке параметри за замовчуванням?
9. Чим відрізняється статичний клас від звичайного класу, і коли доцільно використовувати статичні класи?
10. Що таке статичні поля у класі, і як вони відрізняються від екземплярних полів?