МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Комп‘ютерний практикум №5

з курсу «Основи розробки програмного забезпечення на платформі Microsoft.NET»

на тему: «Шаблони проектування. Структурні шаблони»

Перевірила:
доцент
Ліщук К. І.

Виконав:
студент 2 курсу
групи ІП-21 ФІОТ
Гриценко А. В.

Київ 2024

Варіант 13

Мета: ознайомитися з основними шаблонами проектування, навчитися застосовувати їх при проектуванні і розробці ПЗ..

Постановка задачі комп‘ютерного практикуму № 5
При виконанні комп‘ютерного практикуму необхідно виконати наступні дії:

1) Вивчити структурні патерни. Знати загальну характеристику та призначення кожного з них, особливості реалізації кожного з породжуючих патернів та випадки їх застосування.

2) Реалізувати задачу згідно варіанту, запропонованого нижче у вигляді консольного застосування на мові C#. Розробити інтерфейси та класи з застосування одного або декількох патернів. Повністю реалізувати методи, пов‘язані з реалізацією обраного патерну.

3) Повністю описати архітектуру проекту (призначення методів та класів), особливості реалізації обраного патерну. Для кожного патерну необхідно вказати основні класи та їх призначення.

4) Навести UML-діаграму класів

5) Звіт повинен містити:
   a. обґрунтування обраного патерну (чому саме він);
   b. опис архітектури проекту (призначення методів та класів);
   c. UML-діаграму класів
   d. особливості реалізації обраного патерну
   e. текст програмного коду
   f. скріншоти результатів виконання.

Варіант індивідуального завдання:

13) Існує модель системи «Рецепт». Модель дозволяє в незмінному вигляді зберігати призначення лікаря і термін дії рецепту. Реалізувати задачу, що дозволяє продовжувати термін дії вже існуючого рецепту.

Для реалізації даного завдання був використаний паттерн Декоратор, оскільки він дозволяє додавати нову функціональність до існуючих об'єктів без зміни їхньої структури. У нашому випадку це означає, що ми можемо додати можливість продовження терміну дії рецепту без зміни існуючого коду моделі "Рецепт". Оригінальний клас "Рецепт" не змінюється, щоб включати нову поведінку. Це означає, що клас залишається відповідальним лише за зберігання даних рецепту, тоді як декоратор відповідає за додаткові функції.

Тому використання паттерна Декоратор для задачі продовження терміну дії рецепту є гарним вибором, оскільки він забезпечує гнучкість, розширюваність та підтримує принцип єдиного обов'язку. Це дозволяє додавати нову функціональність без модифікації існуючої структури об'єкта, що знижує ризики внесення помилок в існуючий код.

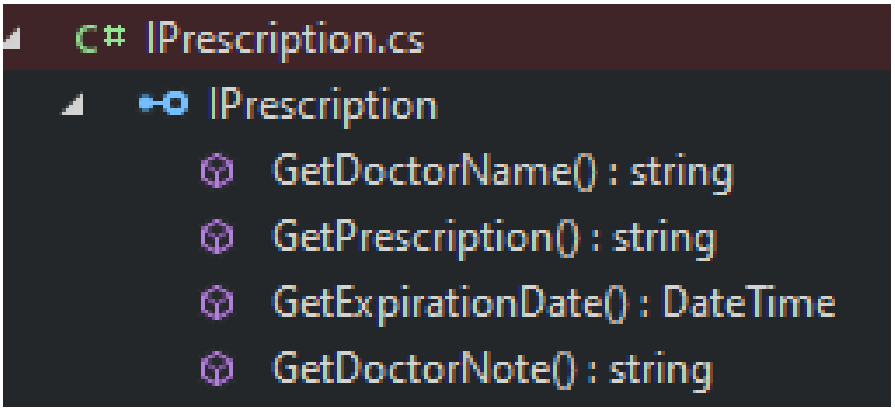

Рисунок 1. Загальний інтерфейс IPrescription (IComponent)

Загальний інтерфейс. Описує контракт для рецептів, включаючи методи для отримання інформації про рецепт.

Методи:

- GetDoctorName(): повертає ім'я лікаря.
- GetPrescription(): повертає інформацію про рецепт.
- GetExpirationDate(): повертає дату закінчення дії рецепту.
- GetDoctorNote(): повертає примітку від лікаря.

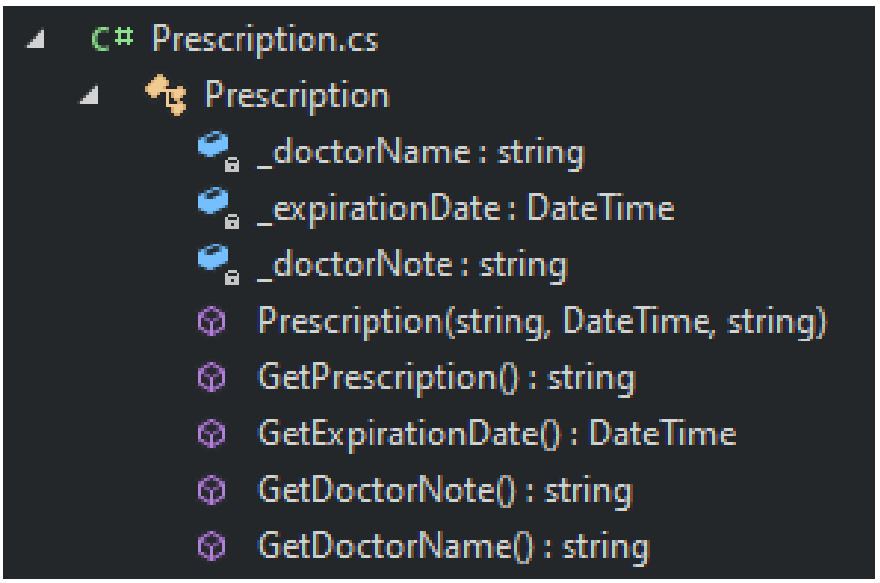

Рисунок 2. Реалізація інтерфейсу

Реалізує інтерфейс IPrescription. Призначений для зберігання інформації про рецепт, включаючи ім'я лікаря, дату закінчення дії та примітку від лікаря.

Конструктор:

- Prescription(string doctorName, DateTime expirationDate, string doctorNote): ініціалізує поля класу.

Методи:

- Реалізує методи інтерфейсу IPrescription.

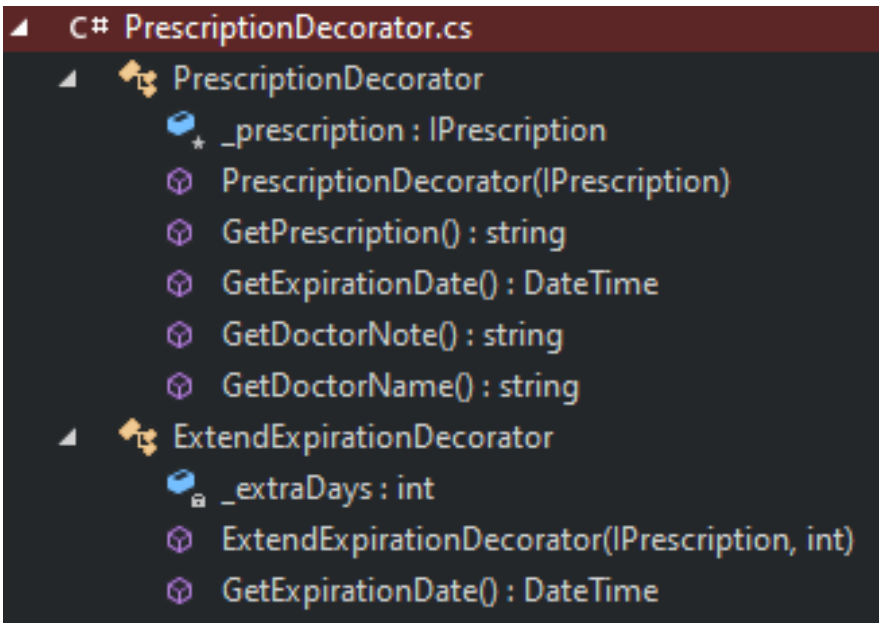

Рисунок 3. Базовий клас для декораторів

Реалізує інтерфейс IPrescription і служить базовим класом для декораторів. Має захищене поле _prescription, яке зберігає оригінальний об'єкт рецепту.

Конструктор:

- PrescriptionDecorator(IPrescription prescription): ініціалізує поле _prescription.

Методи:

- GetPrescription(): повертає результат виклику відповідного методу оригінального рецепту.
- GetExpirationDate(): повертає результат виклику відповідного методу оригінального рецепту.
- GetDoctorNote(): повертає результат виклику відповідного методу оригінального рецепту.
- GetDoctorName(): повертає результат виклику відповідного методу оригінального рецепту.

Клас ExtendExpirationDecorator: наслідує клас PrescriptionDecorator. Призначений для розширення терміну дії рецепту.

Конструктор:

- ExtendExpirationDecorator(IPrescription prescription, int extraDays): ініціалізує базовий клас та додає кількість додаткових днів.

Методи:

- GetExpirationDate(): повертає дату закінчення дії рецепту, збільшену на додаткові дні.

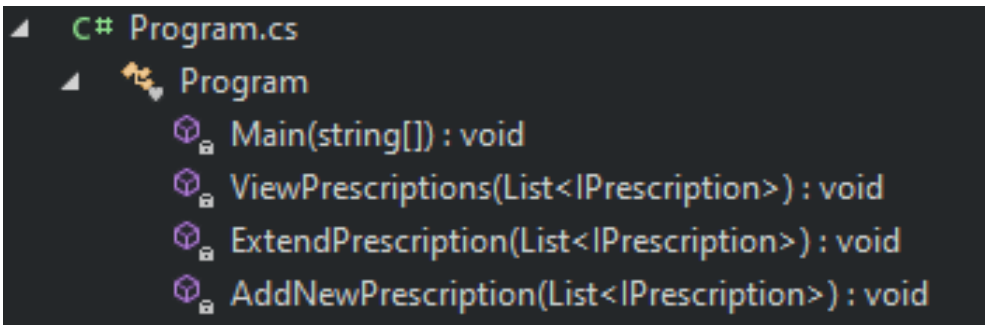

Рисунок 4. Основний клас Program

Методи:

- ViewPrescriptions(List<IPrescription> prescriptions): виводить список рецептів на екран.
- ExtendPrescription(List<IPrescription> prescriptions): дозволяє користувачеві продовжити термін дії обраного рецепту.
- AddNewPrescription(List<IPrescription> prescriptions): дозволяє користувачеві додати новий рецепт до списку.

Тобто класи розподілені таким чином:

- Component: інтерфейс IPrescription
- ConcreteComponent: клас Prescription
- Decorator: абстрактний клас PrescriptionDecorator
- ConcreteDecoratorA/B: клас ExtendExpirationDecorator

Особливості реалізації обраного патерну "Декоратор"

Паттерн Декоратор дозволяє розширювати функціональність динамічно, без зміни вихідного коду класу. Наприклад, клас ExtendedPrescription додає функціональність продовження терміну дії рецепту без зміни базового класу Prescription.

Можна додавати кілька нових обов'язків відразу. Кілька декораторів можуть бути застосовані до одного об'єкта, кожен з яких додає свою функціональність.

Замість створення складного об'єкта з безліччю методів і атрибутів, декоратор розбиває функціональність на дрібні, незалежні об'єкти.Це сприяє кращій модульності та полегшує тестування та підтримку. Кожен декоратор відповідає за одну конкретну функціональність.

Але в той же час, велика кількість вкладених декораторів може ускладнити розуміння того, який декоратор що додає і як вони взаємодіють один з одним. Наприклад, у нашій програмі, якщо до рецепту додається багато декораторів, може бути складно відстежити послідовність їх застосування та кінцевий результат.

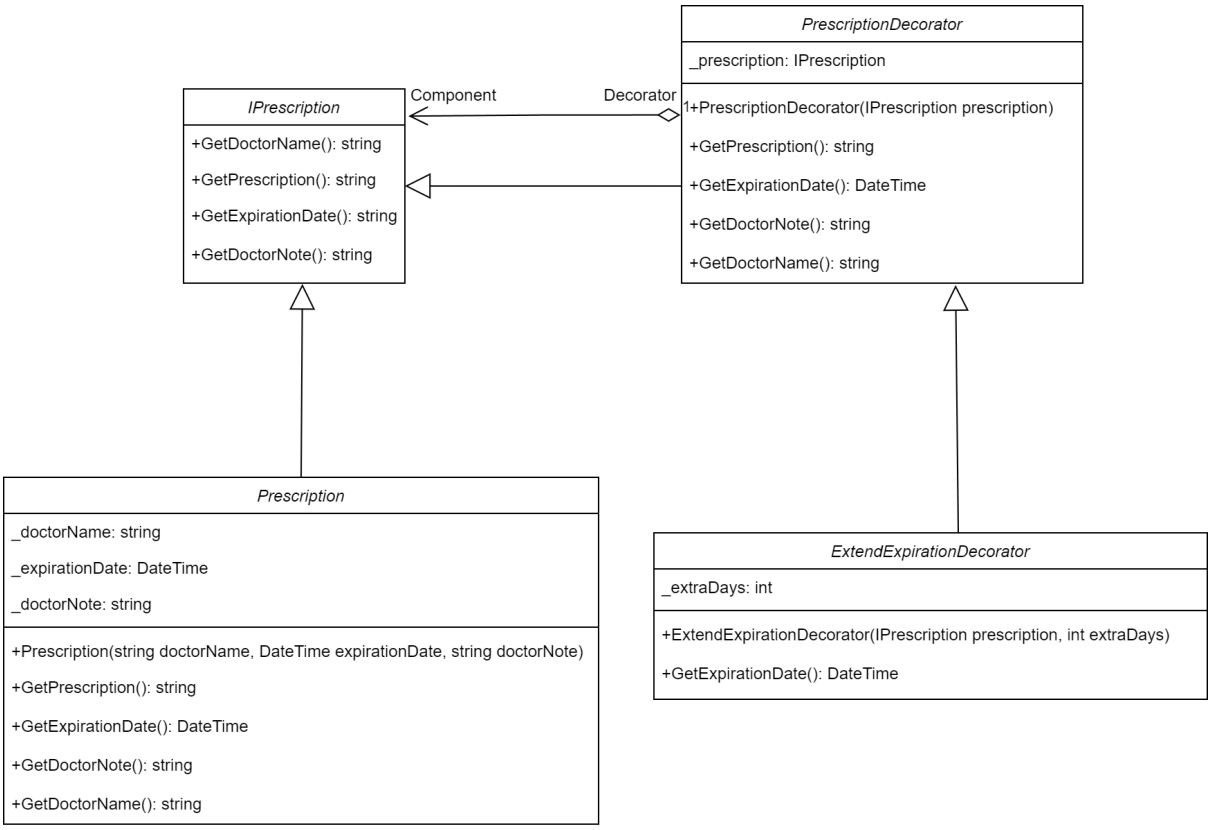

Рисунок 5. UML-діаграма класів

Скріншоти результатів виконання

```
Menu:
1. View prescriptions
2. Extend prescription expiration
3. Add new prescription
4. Exit
Select an option: 1

Available prescriptions:
1. Prescription by Dr. Smith          | Expiration date: 20.06.2024 13:19:54        | Note: Take one pill daily
2. Prescription by Dr. Johnson        | Expiration date: 05.07.2024 13:19:54        | Note: Apply ointment twice daily
3. Prescription by Dr. William        | Expiration date: 20.07.2024 13:19:54        | Note: Rest and drink plenty of fluids
4. Prescription by Dr. Brown          | Expiration date: 15.06.2024 13:19:54        | Note: Inhale medication once a day
5. Prescription by Dr. Jones          | Expiration date: 30.06.2024 13:19:54        | Note: Take two tablets after meals
```

```
Menu:
1. View prescriptions
2. Extend prescription expiration
3. Add new prescription
4. Exit
Select an option: 2

Enter the number of the prescription you want to extend:
1
Enter the number of days to extend the expiration:
10

-----------------------------------------
Previous expiration date: 20.06.2024 13:19:54
New expiration date: 30.06.2024 13:19:54
Doctor name: Smith
Doctor's note: Take one pill daily
-----------------------------------------
```

```
Menu:
1. View prescriptions
2. Extend prescription expiration
3. Add new prescription
4. Exit
Select an option: 3

Enter the doctor's name: Artem
Enter the expiration date and time (yyyy-MM-dd HH:mm): 2024-10-10 16:20
Enter the doctor's note: Take one pill
New prescription added successfully.

Menu:
1. View prescriptions
2. Extend prescription expiration
3. Add new prescription
4. Exit
Select an option: 1

Available prescriptions:
1. Prescription by Dr. Smith          | Expiration date: 30.06.2024 13:19:54        | Note: Take one pill daily
2. Prescription by Dr. Johnson        | Expiration date: 05.07.2024 13:19:54        | Note: Apply ointment twice daily
3. Prescription by Dr. William        | Expiration date: 20.07.2024 13:19:54        | Note: Rest and drink plenty of fluids
4. Prescription by Dr. Brown          | Expiration date: 15.06.2024 13:19:54        | Note: Inhale medication once a day
5. Prescription by Dr. Jones          | Expiration date: 30.06.2024 13:19:54        | Note: Take two tablets after meals
6. Prescription by Dr. Artem          | Expiration date: 10.10.2024 16:20:00        | Note: Take one pill
```

Рисунок 6. Результати роботи програми

Висновок

У даній роботі ми ознайомилися з шаблоном проектування "Декоратор" та його застосуванням у контексті системи «Рецепт». Основна мета полягала в тому, щоб навчитися застосовувати цей патерн для розширення функціональності об'єктів без зміни їхньої структури. Модель системи «Рецепт» дозволяє зберігати призначення лікаря та термін дії рецепту в незмінному вигляді. Використання патерну Декоратор надало можливість динамічно змінювати термін дії рецепту, додаючи нову поведінку до вже існуючих об'єктів.

Програмний код

IPrescription.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace laba5
{
   public interface IPrescription
   {
      string GetDoctorName();
      string GetPrescription();
      DateTime GetExpirationDate();
      string GetDoctorNote();
   }
}
```

Prescription.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace laba5
{
   public class Prescription : IPrescription
   {
      private string _doctorName;
```

```
        private DateTime _expirationDate;
        private string _doctorNote;

        public Prescription(string doctorName, DateTime expirationDate, string
doctorNote)
        {
            _doctorName = doctorName;
            _expirationDate = expirationDate;
            _doctorNote = doctorNote;
        }

        public string GetPrescription()
        {
            return $"Prescription by Dr. {_doctorName}";
        }

        public DateTime GetExpirationDate()
        {
            return _expirationDate;
        }

        public string GetDoctorNote()
        {
            return _doctorNote;
        }

        public string GetDoctorName()
        {
            return _doctorName;
        }
    }
}
```

PrescriptionDecorator.cs:

```
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace laba5
{
    public abstract class PrescriptionDecorator : IPrescription
    {
        protected IPrescription _prescription;

        public PrescriptionDecorator(IPrescription prescription)
        {
            _prescription = prescription;
        }

        public virtual string GetPrescription()
        {
            return _prescription.GetPrescription();
        }

        public virtual DateTime GetExpirationDate()
        {
            return _prescription.GetExpirationDate();
        }

        public virtual string GetDoctorNote()
        {
            return _prescription.GetDoctorNote();
        }

        public virtual string GetDoctorName()
        {
            return _prescription.GetDoctorName();
        }
    }
```

```
    public class ExtendExpirationDecorator : PrescriptionDecorator
    {
        private int _extraDays;

        public ExtendExpirationDecorator(IPrescription prescription, int
extraDays)
            : base(prescription)
        {
            _extraDays = extraDays;
        }

        public override DateTime GetExpirationDate()
        {
            return _prescription.GetExpirationDate().AddDays(_extraDays);
        }
    }
}
```

Program.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace laba5
{
    class Program
    {
        static void Main(string[] args)
        {
            List<IPrescription> prescriptions = new List<IPrescription>
        {
```

```
        new Prescription("Smith", DateTime.Now.AddDays(30), "Take one pill daily"),
        new Prescription("Johnson", DateTime.Now.AddDays(45), "Apply ointment twice daily"),
        new Prescription("William", DateTime.Now.AddDays(60), "Rest and drink plenty of fluids"),
        new Prescription("Brown", DateTime.Now.AddDays(25), "Inhale medication once a day"),
        new Prescription("Jones", DateTime.Now.AddDays(40), "Take two tablets after meals"),
    };

    bool exit = false;
    while (!exit)
    {
        Console.WriteLine("\nMenu:");
        Console.WriteLine("1. View prescriptions");
        Console.WriteLine("2. Extend prescription expiration");
        Console.WriteLine("3. Add new prescription");
        Console.WriteLine("4. Exit");
        Console.Write("Select an option: ");

        string input = Console.ReadLine();
        switch (input)
        {
            case "1":
                ViewPrescriptions(prescriptions);
                break;
            case "2":
                ExtendPrescription(prescriptions);
                break;
            case "3":
                AddNewPrescription(prescriptions);
                break;
            case "4":
                exit = true;
                break;
```

```
                default:
                    Console.WriteLine("Invalid option. Please try again.");
                    break;
            }
        }
    }

    static void ViewPrescriptions(List<IPrescription> prescriptions)
    {
        Console.WriteLine("\nAvailable prescriptions:");
        for (int i = 0; i < prescriptions.Count; i++)
        {
            Console.WriteLine($"{i + 1}. {prescriptions[i].GetPrescription()} \t\t| Expiration date: {prescriptions[i].GetExpirationDate()} \t\t| Note: {prescriptions[i].GetDoctorNote()}");
        }
    }

    static void ExtendPrescription(List<IPrescription> prescriptions)
    {
        Console.WriteLine("\nEnter the number of the prescription you want to extend: ");
        if (int.TryParse(Console.ReadLine(), out int prescriptionNumber) && prescriptionNumber > 0 && prescriptionNumber <= prescriptions.Count)
        {
            Console.WriteLine("Enter the number of days to extend the expiration:");
            if (int.TryParse(Console.ReadLine(), out int extraDays))
            {
                IPrescription selectedPrescription = prescriptions[prescriptionNumber - 1];
                IPrescription extendedPrescription = new ExtendExpirationDecorator(selectedPrescription, extraDays);

                prescriptions[prescriptionNumber - 1] = extendedPrescription; // Update the list with the extended prescription
```

```
                Console.WriteLine("\n----------------------------------------");
                Console.WriteLine($"Previous expiration date:
{selectedPrescription.GetExpirationDate()}");
                Console.WriteLine($"New expiration date:
{extendedPrescription.GetExpirationDate()}");
                Console.WriteLine($"Doctor name:
{selectedPrescription.GetDoctorName()}");
                Console.WriteLine($"Doctor's note:
{extendedPrescription.GetDoctorNote()}");
                Console.WriteLine("----------------------------------------");
            }
            else
            {
                Console.WriteLine("Invalid number of days.");
            }
        }
        else
        {
            Console.WriteLine("Invalid prescription number.");
        }
    }

    static void AddNewPrescription(List<IPrescription> prescriptions)
    {
        Console.Write("\nEnter the doctor's name: ");
        string doctorName = Console.ReadLine();

        Console.Write("Enter the expiration date and time (yyyy-MM-dd
HH:mm): ");
        if (DateTime.TryParseExact(Console.ReadLine(), "yyyy-MM-dd
HH:mm", null, System.Globalization.DateTimeStyles.None, out DateTime
expirationDate))
        {
            Console.Write("Enter the doctor's note: ");
            string doctorNote = Console.ReadLine();
```

```
            IPrescription newPrescription = new Prescription(doctorName,
expirationDate, doctorNote);
            prescriptions.Add(newPrescription);

            Console.WriteLine("New prescription added successfully.");
        }
        else
        {
            Console.WriteLine("Invalid date and time format.");
        }
    }
  }
}
```