Міністерство освіти і науки України
Національний технічний університет України «КПІ ім. Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

З лабораторної роботи №5

з курсу

«Основи розробки програмного забезпечення на платформі Microsoft.NET»

Виконав студент *ІП-01 Галько Міла*
(шифр, ПІБ)

Перевірила *Ліщук К.І.*
(ПІБ)

Київ 2022

Комп‘ютерний практикум № 4. Шаблони проектування. Поведінкові шаблони

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 поведінкові патерни. Знати загальну характеристику та призначення кожного з них, особливості реалізації кожного з поведінкових патернів та випадки їх застосування.
2) Реалізувати задачу згідно варіанту, запропонованого нижче. 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

Варіант №1

Реалізувати алгоритм гри судоку. Реалізувати можливість «взяти назад хід».

Програмний код:

Посилання на github: *https://github.com/MilaHalko/C4_.NET/tree/Lab5*

Також код доданий в кінці документу.

Вибір патерну

Через те що стикаємося із проблемою «Можливість взяти хід назад», то очевидним рішенням стане використання патерну «Memento». За допомогою нього отримаємо можливість зберігати стани гри від самого початку до її поточного виду. Отже для реалізації патерну потребуємо клас, дані якого будуть використані для збереження стану – Field(поле судоку). Клас, що зберігає стан – SudokuMemento, та клас із стеком станів – SudokuHistory.

Оскільки стан зберігається при кожному ході клієнта, то можливість створення об’єкту SudokuMemento (збереження стану) краще надавати не клієнтові, а робити цей виклик безпосередньо при внесенні нової цифри у судоку клієнтом без його відома. Отже метод SaveState() (збереження стану гри) є сенс зробити приватним у класі Field. А вже метод “Undo” – користувацький, і повертає стан поля на попередньому ході клієнта.

Опис архітектури:

1. **class Field** – уособлює поле судоку із можливістю взаємодії із ним.
   Атрибути:
   field (базове поле судоку розмірністю 9 на 9),
   H (стек станів, об’єкт SudokuHistory).
   Методи:
   SetValue(int horizontal, int vertical, int value): дозволяє клієнтові виставляти числа на полі судоку за вертикаллю та горизонталлю, перевіряючи те, що число є

однакових чисел). Якщо вставка успішна, додатково виконується збереження стану; якщо неуспішна, то викликається повідомлення про помилку.

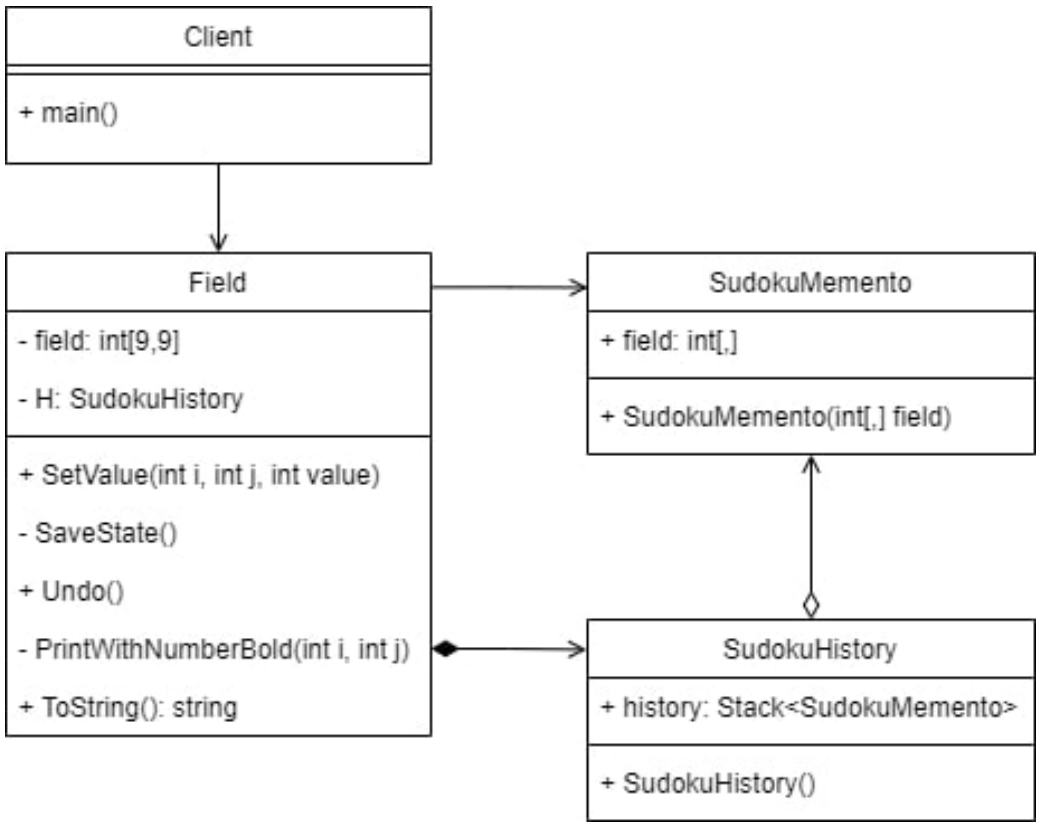
private SaveState() – метод, що додає у стек H новостворений поточний стан гри. Використовується у методі SetValue().

Undo() – метод, що повертає стан поля на попередньому ході.

2. **class SudokuMemento** – клас, що утримує масив-поле класу Field, тим самим зберігаючи стан гри.

3. **class SudokuHistory** – клас, що виконує функцію збереження станів у стеку.

UML діаграма



Вміст файлу Program.cs

```
using SudokuMementoProg;

namespace sudoku
{
	class Program
	{
		static void Main(string[] args)
		{
			Field sudoku = new Field();
			Console.WriteLine("*** SUDOKU GAME STARTED:");
			Console.WriteLine(sudoku);
			sudoku.Undo();
			sudoku.SetValue(0, 0, 1); // +
			sudoku.SetValue(0, 4, 9); // wrong
			sudoku.SetValue(0, 4, 7); // +
			sudoku.SetValue(0, 5, 10);// wrong
			sudoku.Undo();            // -
			sudoku.SetValue(1, 3, 1); // +
			sudoku.SetValue(5, 4, 8); // wrong (square)
			sudoku.Undo();             // -
			sudoku.Undo();            // -
			sudoku.Undo();            // -
		}
	}
}
```

Вміст файлу Field.cs

```
namespace SudokuMementoProg
{
	internal class Field
	{
		private int[,] field =
		{
			{0, 8, 9, 0, 0, 0, 3, 0, 0 },
			{0, 0, 0, 0, 3, 6, 0, 2, 9 },
			{0, 0, 0, 0, 0, 2, 5, 0, 0 },
			{0, 7, 3, 0, 0, 0, 4, 8, 2 },
			{0, 0, 0, 5, 0, 8, 0, 0, 0 },
			{0, 0, 0, 4, 0, 0, 0, 0, 7 },
			{7, 9, 0, 3, 6, 0, 0, 1, 8 },
			{0, 0, 0, 0, 5, 0, 0, 0, 6 },
			{3, 0, 2, 0, 9, 0, 0, 0, 5 }
		};
		private SudokuHistory H = new SudokuHistory();
		public void SetValue(int horizontal, int vertical, int value)
		{
			Console.WriteLine($"\n*** SET [{horizontal}, {vertical}] = {value}");
			bool correct = true;
			if (value < 1 || value > 9) { correct = false;}
			if (correct)
			{
				for (int index = 0; index < 9; index++)
				{
					if (value == field[index, vertical]) { correct = false; }
					if (value == field[horizontal, index]) {correct = false;}
				}
			}
			if (correct)
			{
				int sect_i = (horizontal / 3) * 3;
				int sect_j = (vertical / 3) * 3;
				for (int i = sect_i; i < sect_i + 3; i++)
				{
					for (int j = sect_j; j < sect_j + 3; j++)
					{
						if (value == field[i, j])
						{
							correct = false;
							break;
						}
					}
					if(!correct)
					{
						break;
					}
```

```
                    }
                }

                if (correct)
                {
                    Console.WriteLine("Saving process:");
                    this.SaveState();
                    Console.WriteLine("State is pushed! Old Sudoku:");
                    this.PrintWithNumberBold(horizontal, vertical);
                    field[horizontal, vertical] = value;
                    Console.WriteLine("New Sudoku:");
                    this.PrintWithNumberBold(horizontal, vertical);
                }
                else
                {
                    Console.WriteLine("New value is not possible!");
                    this.PrintWithNumberBold(horizontal, vertical);
                }
            }

            private void SaveState()
            {
                H.history.Push(new SudokuMemento(field));
            }

            public void Undo()
            {
                Console.WriteLine("\n*** UNDO STEP:");
                if (H.history.Count() > 0)
                {
                    Console.WriteLine("Undo Process. Current Sudoku:");
                    Console.WriteLine(this);

                    SudokuMemento memento = H.history.Pop();
                    int i_ = -1, j_ = -1;
                    bool changed = false;
                    for (int i = 0; i < 9; i++)
                    {
                        for (int j = 0; j < 9; j++)
                        {
                            if (this.field[i, j] != memento.field[i, j])
                            {
                                i_ = i;
                                j_ = j;
                                this.field[i, j] = memento.field[i, j];

                                changed = true;
                                break;
                            }
                        }
                        if (changed)
                        {
                            break;
                        }
                    }
                    Console.WriteLine("Undo is done! Previous Sudoku:");
                    this.PrintWithNumberBold(i_, j_);
                }
                else
                {
                    Console.WriteLine("Stack is empty -> Undo is impossible!");
                }
            }

            public override string ToString()
            {
                string fieldStr = "";
                for (int i = 0; i < 9; i++)
                {
                    for (int j = 0; j < 9; j++)
                    {
                        fieldStr += field[i, j].ToString() + " ";
                    }
                    fieldStr += "\n";
                }
                return fieldStr;
            }
```

```
                private void PrintWithNumberBold(int horizontal, int vertical)
                {
                        for (int i = 0; i < 9; i++)
                        {
                                for (int j = 0; j < 9; j++)
                                {
                                        if (horizontal == i && vertical == j)
                                        {
                                                Console.ForegroundColor = ConsoleColor.Green;
                                                Console.Write(field[i, j].ToString() + " ");
                                                Console.ResetColor();
                                        }
                                        else
                                        {
                                                Console.Write(field[i, j].ToString() + " ");
                                        }
                                }
                                Console.WriteLine();
                        }
                        Console.WriteLine();
                }
        }
}
```

Вміст файлу SudokuMemento.cs

```
namespace SudokuMementoProg
{
        internal class SudokuMemento
        {
                public int[,] field { get; private set; }
                public SudokuMemento(int[,] field)
                {
                        this.field = new int[9, 9];
                        for (int i = 0; i < 9; i++)
                        {
                                for (int j = 0; j < 9; j++)
                                {
                                        this.field[i, j] = field[i, j];
                                }
                        }
                }
        }
}
```

Вміст файлу SudokuHistory.cs

```
namespace SudokuMementoProg
{
    internal class SudokuHistory
        {
                public Stack<SudokuMemento> history { get; private set; }
                public SudokuHistory()
                {
                        history = new Stack<SudokuMemento>();
                }

        }
}
```