Міністерство освіти і науки України
Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

Звіт

з лабораторної роботи №2 з дисципліни
“Програмування 2. Структури даних та алгоритми”

“ДОСЛІДЖЕННЯ ІНДЕКСАТОРІВ ТА ВЛАСТИВОСТЕЙ”

Варіант №13

Виконав студент Флорчук Назарій Петрович

Перевірив викладач Проскура Світлана Леонідівна

Мета роботи:

1. Дослідити індексатори та властивості, набути навички їх створення та використання мовою програмування C#.

Завдання:

1. Вивчити та дослідити опис індексаторів та властивостей мовою програмування C#.
2. Написати програму мовою програмування C#, яка описує клас з індексатором та властивістю згідно з варіантом та тестує роботу індексатора і властивості.

Завдання
Клас є таблицею студентів. Кожен рядок таблиці містить ім’я, прізвище та по батькові студента. Створити одновимірний індексатор, що повертає заданий індексом стовпець таблиці. Параметр індексатора повинен мати тип string. Створити властивість, що контролює доступ до змінної, яка містить кількість студентів у таблиці що мають прізвище “Нечай”.

Код програми (C#):

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

class Student
{
    private string _m_surname;
    private string _m_name;
    private string _m_patronymic;

    public Student(string surname = "", string name = "", string patronymic = "")
    {
        this._m_surname = surname;
        this._m_name = name;
        this._m_patronymic = patronymic;
    }

    /**
     * Return student surname (Property, read only logic)
     */
    public string Surname
    {
        get
        {
            return this._m_surname;
        }
    }

    /**
     * Return student name (Property, read only logic)
     */
    public string Name
    {
        get
        {
            return this._m_name;
        }
    }

    /**
     * Return student patronymic (Property, read only logic)
     */
    public string Patronymic
    {
        get
        {
            return this._m_patronymic;
        }
```

```
    }

    /**
     * Return student full name (Property, read only logic)
     */
    public string FullName
    {
        get
        {
            return this._m_surname + " " + this._m_name + " " + this._m_patronymic;
        }
    }
}

class Column
{
    private int _m_length;
    private string[] _m_column;

    public Column(int length)
    {
        this._m_length = length;
        this._m_column = new string[length];
    }

    /**
     * Add student into table (Indexer, read and write logic)
     */
    public string this[int index]
    {
        get
        {
            return this._m_column[index];
        }
        set
        {
            this._m_column[index] = value;
        }
    }

    /**
     * Get object items length (Property, read only logic)
     */
    public int Length
    {
        get
        {
            return this._m_length;
        }
```

```
    }
}

class StudentsTable
{
    public const string SurnameColumn = "Surname";
    public const string NameColumn = "Name";
    public const string PatronymicColumn = "Patronymic";

    private List<Student> _m_students_table;
    private int _m_nechay_counter;

    public StudentsTable ()
    {
        this._m_students_table = new List<Student>();
        this._m_nechay_counter = 0;
    }

    /**
     * Get table Column object (Indexer, read only logic)
     */
    public Column this[string columnName]
    {
        get
        {
            if (columnName == SurnameColumn || columnName == NameColumn || columnName ==
PatronymicColumn)
            {
                Column column = new Column(this._m_students_table.Count);

                for (int i = 0; i < this._m_students_table.Count; i++)
                {
                    column[i] = (string)
this._m_students_table[i].GetType().GetProperty(columnName).GetValue(this._m_students_table[i]);
                }

                return column;
            }

            throw new System.ArgumentException("Invalid column name.", nameof(columnName));
        }
    }

    /**
     * Set Student object into the table (Indexer, write only logic)
     */
    public Student this[int index]
    {
        set
```

```
    {
        if (value.Surname == "Нечай")
        {
            this._m_nechay_counter++;
        }

        if (this._m_students_table.ElementAtOrDefault(index) != null)
        {
            this._m_students_table[index] = value;
        }
        else
        {
            this._m_students_table.Insert(index, value);
        }
    }
}

/**
 * Return count of students by surname Нечай (Property, read only logic)
 */
public int NechayLength
{
    get
    {
        return this._m_nechay_counter;
    }
}

/**
 * Get object items length (Property, read only logic)
 */
public int Length
{
    get
    {
        return this._m_students_table.Count;
    }
}

/**
 * Print out students to the console
 */
public void Print()
{
    for (int i = 0; i < this._m_students_table.Count; i++)
    {
        System.Console.WriteLine(this._m_students_table[i].FullName);
    }
}
```

```
    ~StudentsTable()
    {
        this._m_students_table.Clear();
    }
}

class Application
{
    static void Main(string[] args)
    {
        StudentsTable studentsTable = new StudentsTable();

        System.Console.WriteLine("____ Initial StudentsTable object:");

        studentsTable.Print();

        System.Console.WriteLine("____ StudentsTable object (with three Student objects):");

        studentsTable[0] = new Student("Кирпичов", "Віктор", "Львович");
        studentsTable[1] = new Student("Патон", "Євген", "Оскарович");
        studentsTable[2] = new Student("Сікорський", "Ігор", "Іванович");

        studentsTable.Print();

        System.Console.WriteLine("____ StudentsTable object (with additional two Student objects):");

        studentsTable[3] = new Student("Нечай", "Михайло", "Потапович");
        studentsTable[4] = new Student("Плотніков", "Володимир", "Олександрович");

        studentsTable.Print();

        System.Console.WriteLine("____ Count of students by surname \"Нечай\":");
        System.Console.WriteLine(studentsTable.NechayLength);

        System.Console.WriteLine("____ StudentsTable object (with additional one Student object):");

        studentsTable[5] = new Student("Нечай", "Ярослав", "Іванович");

        studentsTable.Print();

        System.Console.WriteLine("____ Count of students by surname \"Нечай\":");
        System.Console.WriteLine(studentsTable.NechayLength);

        System.Console.WriteLine("____ Print out StudentsTable object Surname column:");

        Column surnameColumn = studentsTable[StudentsTable.SurnameColumn];

        for (int i = 0; i < surnameColumn.Length; i++)
```

```
        {
            System.Console.WriteLine(surnameColumn[i]);
        }

        System.Console.WriteLine("____ Print out StudentsTable object Name column:");

        Column nameColumn = studentsTable[StudentsTable.NameColumn];

        for (int i = 0; i < nameColumn.Length; i++)
        {
            System.Console.WriteLine(nameColumn[i]);
        }

        System.Console.WriteLine("____ Print out StudentsTable object Patronymic column:");

        Column patronymicColumn = studentsTable[StudentsTable.PatronymicColumn];

        for (int i = 0; i < patronymicColumn.Length; i++)
        {
            System.Console.WriteLine(patronymicColumn[i]);
        }
    }
}
```

```
[redacted]@debian:~/Documents/kpi/basics_of_programming_2/lab_2$ mcs lab_2.cs
[redacted]@debian:~/Documents/kpi/basics_of_programming_2/lab_2$ mono lab_2.exe
____ Initial StudentsTable object:
____ StudentsTable object (with three Student objects):
Кирпичов Віктор Львович
Патон Євген Оскарович
Сікорський Ігор Іванович
____ StudentsTable object (with additional two Student objects):
Кирпичов Віктор Львович
Патон Євген Оскарович
Сікорський Ігор Іванович
Нечай Михайло Потапович
Плотніков Володимир Олександрович
____ Count of students by surname "Нечай":
1
____ StudentsTable object (with additional one Student object):
Кирпичов Віктор Львович
Патон Євген Оскарович
Сікорський Ігор Іванович
Нечай Михайло Потапович
Плотніков Володимир Олександрович
Нечай Ярослав Іванович
____ Count of students by surname "Нечай":
2
____ Print out StudentsTable object Surname column:
Кирпичов
Патон
Сікорський
Нечай
Плотніков
Нечай
____ Print out StudentsTable object Name column:
Віктор
Євген
Ігор
Михайло
Володимир
Ярослав
____ Print out StudentsTable object Patronymic column:
Львович
Оскарович
Іванович
Потапович
Олександрович
Іванович
```

Висновки / Відповіді на контрольні запитання:

1. Що таке індексатор?
   Індексатори — це різновид властивостей, який дозволяє звертатися до полів класу за допомогою індексу.
2. Для чого застосовують індексатор?
   Індексатори в основному використовуються у класах для доступу до закритого поля масиву (одновимірного або багатовимірного). Як і у випадку властивостей, як правило, індексатор оголошують відкритим (зі специфікатором доступу **public**). Тип індексатора не може бути **void**. При описі індексатора може бути відсутньою або частина **get** або частина **set**, але не обидві одночасно. Якщо у описі індексатора присутня тільки частина **get**, то такий індексатор називається *індексатором тільки для читання*. Якщо у описі індексатора присутня тільки частина **set**, то такий індексатор називається *індексатором тільки для запису*. Метод **get** повинен містити оператор **return**. В методі **set** для доступу до значення, яке встановлюється, використовується неявний параметр **value**. У списку індексів вказують через кому опис індексів (тип індексу та його ім’я). Тип індексу та кількість індексів у списку може бути довільною, але найчастіше використовують один індекс цілого типу.
3. Поясніть призначення accessor’ів get та set.
   - Accessor get, метод, виклик якого виконується, коли виконується звернення до властивості чи до індексатора, для отримання його значення.
   - Accessor set, метод, виклик якого виконується, коли виконується звернення до властивості чи до індексатора, для зміни його значення.
4. Що таке властивість?
   Властивість — це різновид метода, який слугує для організації коректного доступу до полів класу.
5. Для чого застосовують властивості?
   Застосовуються для вирішення проблеми коректного використання полів класу. Як правило, властивість оголошують відкритою (зі специфікатором доступу **public**). Якщо властивість використовується для доступу до деякого закритого поля, то тип властивості повинен збігатися із типом цього поля. Тип властивості не може бути **void**. При описі властивості може бути відсутньою або частина **get** або частина **set**, але не обидві одночасно. Якщо у описі властивості присутня тільки частина **get**, то така властивість називається *властивістю тільки для читання*. Якщо у описі властивості присутня тільки частина **set**, то така властивість називається *властивістю тільки для запису*. Метод **get** повинен містити оператор **return**. В методі **set** для доступу до значення, яке встановлюється, використовується неявний параметр **value**.
6. Що таке автоматична властивість? Поясніть її призначення.
   Автоматична властивість оголошується аналогічно класичній, з однією відмінністю — тіло accessor’ів **get** та **set** пусті. Це дозволяє створити неявні поля класу. Які створюються компілятором на етапі компіляції.
7. Чи підлягають перевантаженню індексатори та властивості? Якщо так, то навести приклади.
   Індексатори підлягають перевантаженню. Перевантаження індексаторів — це використання кількох індексаторів, з різними типами індексів або їх кількістю. Приклад наведено у класі StudentsTable.
8. Які типи індексів дозволяють застосовувати індексатори?
   Можна використовувати як цілочисельні, так і рядкові типи даних у якості індексів для індексаторів.