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

**ЗВІТ**

**ДО ЛАБОРАТОРНОЇ РОБОТИ №1**

з дисципліни

«Сучасні технології розробки WEB-застосувань на платформі Microsoft.NET»

студента III курсу ФІОТ групи ІК-12

Басюка Валентина

Перевірив:
Бардін Владислав

Київ 2023

# ЛАБОРАТОРНА РОБОТА №1

Узагальнені типи (Generic) з підтримкою подій. Колекції

**Мета:** Навчитися проектувати та реалізовувати узагальнені типи, а також типи з підтримкою подій.

**Варіант №1**

**Завдання:**

1. Розробити клас власної узагальненої колекції, використовуючи стандартні інтерфейси колекцій із бібліотек System.Collections та System.Collections.Generic. Стандартні колекції при розробці власної не застосовувати. Для колекції передбачити методи внесення даних будь-якого типу, видалення, пошуку та ін. (відповідно до типу колекції).

2. Додати до класу власної узагальненої колекції підтримку подій та обробку виключних ситуацій.

3. Опис класу колекції та всіх необхідних для роботи з колекцією типів зберегти у динамічній бібліотеці.

4. Створити консольний додаток, в якому продемонструвати використання розробленої власної колекції, підписку на події колекції.

| Варіант | Опис узагальненої колекції | Функціонал | Реалізація |
|---|---|---|---|
| 1 | Стек | Див. Stack<T> | Збереження даних за допомогою динамічно зв’язаного списку |

# Лістинг програмного коду

```
using System.Collections;
using System.Text;

namespace MyCollectionLibrary;

public class MyStack<T> : IEnumerable<T>
{
    private class Item
    {
        public T? Data { get; set; }
        public Item? Next { get; set; }
    }

    private Item? _top;

    public event EventHandler<T>? OnItemAdded;
    public event EventHandler<T>? OnItemRemoved;
    public event EventHandler? OnClear;

    public void Push(T item)
    {
        var newItem = new Item { Data = item, Next = _top };
        _top = newItem;
        OnItemAdded?.Invoke(this, item);
    }

    public T Pop()
    {
        if (IsEmpty())
            throw new InvalidOperationException("Stack is empty.");

        T data = _top.Data;
        _top = _top.Next;
        OnItemRemoved?.Invoke(this, data);
        return data;
    }

    public T Peek()
    {
        if (IsEmpty())
            throw new InvalidOperationException("Stack is empty.");

        return _top.Data;
    }

    public bool IsEmpty()
    {
        return _top == null;
    }

    public void Clear()
    {
        _top = null;
        OnClear?.Invoke(this, EventArgs.Empty);
    }

    public bool Contains(T item)
    {
        Item? current = _top;
        while (current != null)
        {
            if (EqualityComparer<T>.Default.Equals(current.Data, item))
                return true;
```

```
            current = current.Next;
        }

        return false;
    }

    public override string ToString()
    {
        var sb = new StringBuilder();
        foreach (var item in this)
        {
            sb.AppendLine(item?.ToString());
        }
        return sb.ToString();
    }

    // Implementation of the IEnumerable<T> interface
    public IEnumerator<T> GetEnumerator()
    {
        return new MyEnumerator(this);
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }

    // Implementation of the own IEnumerator<T> interface
    private class MyEnumerator : IEnumerator<T>
    {
        private Item? _current;
        private readonly MyStack<T> _stack;

        public MyEnumerator(MyStack<T> stack)
        {
            _stack = stack;
            _current = null;
        }

        public bool IsHasNext()
        {
            return _current != null;
        }

        public T Current
        {
            get
            {
                if (!IsHasNext())
                    throw new InvalidOperationException("Enumerator is not
started.");

                return _current.Data;
            }
        }

        object IEnumerator.Current => _current.Data;

        public void Dispose()
        {
            // Don't need to perform additional actions when freeing resources.
        }
```

```
        public bool MoveNext()
        {
            if (!IsHasNext())
                _current = _stack._top;
            else
                _current = _current.Next;

            return IsHasNext();
        }

        public void Reset()
        {
            _current = null;
        }
    }
}
```