МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

Лабораторна робота №1

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

на тему: « Узагальнені типи (Generic) з підтримкою подій. Колекції»

Виконав:

студент гр. ІС-11

Поліщук В.

Викладач:

Бардін В.

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

**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 1:**



**Посилання на репозиторій:**

<https://github.com/Vissstor/Lab_1.Net_Stack.git>

**Лістинг коду:**

**StackNode.cs:**

public class StackNode<T>

{

public T Value { get; set; }

public StackNode<T>? NextElement { get; set; }

public StackNode(T value)

{

Value = value;

NextElement = default;

}

}

**MyStackCollection.cs:**

public class MyStackCollection<T> : IEnumerable<T>, ICollection

{

private int \_count;

private StackNode<T>? \_first;

public int Count { get { return \_count; } }

public StackNode<T>? First { get { return \_first; } }

public event EventHandler<StackEventArg<T>>? ItemPushed;

public event EventHandler<StackEventArg<T>>? ItemPopped;

public object SyncRoot => this;

public bool IsSynchronized => false;

public MyStackCollection(IEnumerable<T> collection)

{

if (collection == null) throw new ArgumentNullException(nameof(collection), "Collection is empty");

foreach (var item in collection)

{

Push(item);

}

}

public MyStackCollection()

{

\_count = default;

\_first = default;

}

public void Push(T item)

{

StackNode<T> nodePush = new StackNode<T>(item);

nodePush.NextElement = \_first;

\_first = nodePush;

\_count++;

OnItemPushed(item);

}

public T Pop()

{

if (\_count == 0)

throw new InvalidOperationException("The stack is empty.");

T item = \_first!.Value;

\_first = \_first.NextElement;

\_count--;

OnItemPopped(item);

return item;

}

public T Peek()

{

if (\_count == 0)

throw new InvalidOperationException("The stack is empty.");

return \_first!.Value;

}

public void CopyTo(Array array, int index)

{

if (array == null) throw new ArgumentNullException(nameof(array), "Array is null");

if (index < 0 || index >= array.Length) throw new ArgumentOutOfRangeException(nameof(index));

if (this.Count > array.Length - index) throw new ArgumentException(nameof(this.Count));

StackNode<T>? current = \_first;

int currentIndex = index;

T[]? typedArray = array as T[];

while (current != null)

{

typedArray![currentIndex++] = current.Value;

current = current.NextElement;

}

}

public bool Contains(T item)

{

StackNode<T> stackNode = \_first!;

while (stackNode != null)

{

if (stackNode.Value != null && stackNode.Value!.Equals(item))

return true;

if (item == null && stackNode.Value == null)

return true;

stackNode = stackNode.NextElement!;

}

return false;

}

public bool TryPeek(out T result)

{

if (\_count != 0)

{

result = \_first!.Value;

return true;

}

result = default!;

return false;

}

public bool TryPop(out T result)

{

if (\_count != 0)

{

result = this.Pop();

return true;

}

result = default!;

return false;

}

public void Clear()

{

while (this.Count != 0)

{

this.Pop();

}

\_count = 0;

}

protected virtual void OnItemPushed(T item)

{

ItemPushed?.Invoke(this, new StackEventArg<T>(item, "Значення добавлено"));

}

protected virtual void OnItemPopped(T item)

{

ItemPopped?.Invoke(this, new StackEventArg<T>(item, "Значення видалено"));

}

public IEnumerator<T> GetEnumerator()

{

return new MyEnumerator(this);

/\* StackNode<T> stackNode= \_first;

while (stackNode != null)

{

yield return stackNode.Value;

stackNode = stackNode.NextElement;

}\*/

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

private sealed class MyEnumerator : IEnumerator<T>

{

private StackNode<T>? \_current;

private readonly MyStackCollection<T>? \_stack;

object IEnumerator.Current => Current!;

public MyEnumerator(MyStackCollection<T> stack)

{

\_stack = stack;

if (\_stack is not null && \_stack.Count > 0)

{

\_current = default;

}

}

public T Current

{

get

{

if (\_current == null)

throw new InvalidOperationException();

return \_current.Value;

}

}

public void Dispose()

{

}

public bool MoveNext()

{

if (\_current == null)

{

\_current = \_stack!.First;

}

else

\_current = \_current.NextElement;

return \_current != null;

}

public void Reset()

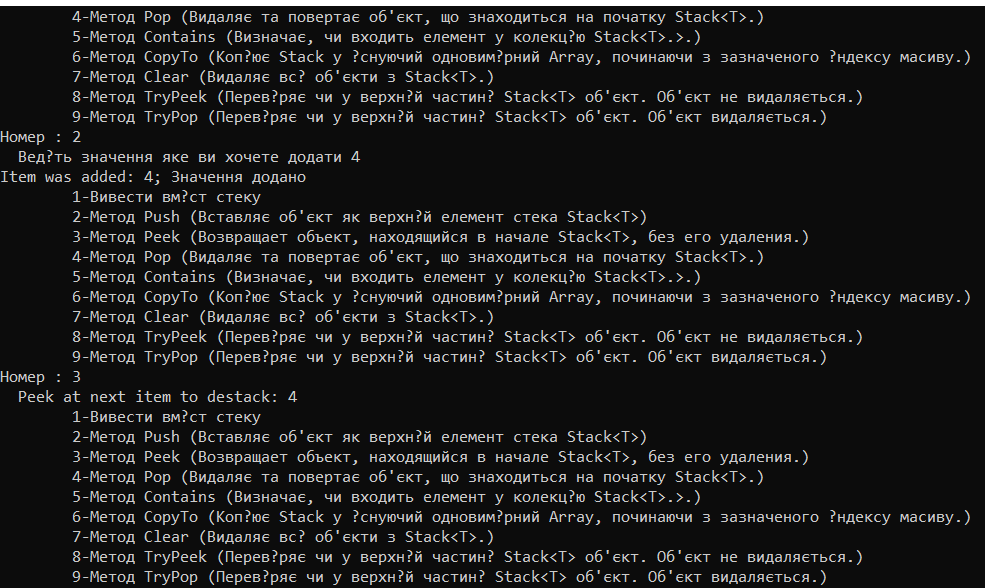
{

\_current = default;

}

}

**Консольна програма:**



**Висновок:** в ході виконання лабораторної роботи я навчився реалізовувати узагальнені типи.

**Питання до роботи:**

1) Дайте визначання колекції. Як колекція пов’язана з інтерфейсами IEnumerable та ICollection?

Колекція (collection) - це абстрактна структура даних, яка дозволяє зберігати та управляти групою об'єктів або значень як один об'єкт. Колекції використовуються для організації та зберігання даних у програмі. Вони можуть приймати різні форми, такі як списки, масиви, словники, стеки, черги і багато інших. Інтерфейси IEnumerable та ICollection в .NET Framework (і подібні інтерфейси в інших мовах програмування) використовуються для роботи з колекціями і надають специфікації для різних операцій, які можна виконати з колекціями.

2) Розкажіть про основні інтерфейси необхідні для функціонування колекцій та LINQ.

LINQ використовує низку стандартних методів розширення (extension methods) для операцій обробки даних над колекціями. Основні інтерфейси для LINQ включають:

IEnumerable<T>: LINQ працює з колекціями, які реалізують IEnumerable<T>, що дозволяє виконувати операції запиту на цих колекціях.

IQueryable<T>: Інтерфейс IQueryable<T> використовується для роботи з джерелами даних, які можуть бути запитані за допомогою LINQ до джерел даних, таких як бази даних.

За допомогою цих інтерфейсів і LINQ можна виконувати операції фільтрації, сортування, групування та проектування над даними, що значно спрощує роботу з колекціями та джерелами даних в C#.

3) Розкажіть про призначення і можливі сценарії застосування інтерфейсу IEnumerator.

Інтерфейс IEnumerator в C# призначений для перебору (ітерації) елементів колекцій або даних, які підтримують послідовний доступ до своїх елементів. Цей інтерфейс надає стандартний спосіб ітерації над колекціями, незалежно від їх типу або реалізації. Це дозволяє створювати загальні алгоритми обробки даних.

Основне призначення інтерфейсу IEnumerator полягає в тому, щоб дозволити послідовно переглядати елементи колекції без необхідності знати деталі її внутрішньої реалізації. Ось деякі основні методи та їх призначення:

4) Порівняйте інтерфейси IEnumerable та IAsyncEnumerable.

IEnumerable і IAsyncEnumerable - це два різних інтерфейси в C#, які використовуються для ітерації (перебору) послідовностей даних. Основна різниця між ними полягає в тому, що IEnumerable призначений для синхронного доступу до даних, тоді як IAsyncEnumerable призначений для асинхронного доступу до даних.

5) Розкажіть про призначення і обмеження generic типів.

Призначення

Generic типи (загальні типи) в C# дозволяють створювати класи, інтерфейси і методи, які можуть працювати з різними типами даних без визначення конкретних типів під час компіляції. Головна мета generic типів - забезпечити велику перевикористовуваність коду і вдосконалити безпеку типів.

Обмеження

При роботі з generic типами можна використовувати обмеження, які накладають певні правила на типи, які можуть бути використані для параметрів типу. Ці обмеження дозволяють покращити безпеку типів та забезпечити правильну роботу коду. Ось деякі з найбільш поширених обмежень:

where T : class: Обмеження class означає, що параметр типу T повинен бути посиланням на посиланням на посиланням (reference type), тобто класом або іншим посиланням на тип (включаючи інтерфейси).

where T : struct: Обмеження struct означає, що параметр типу T повинен бути значимим типом (value type), тобто структурою.

where T : new(): Обмеження new() означає, що параметр типу T повинен мати доступний конструктор без параметрів (пустий конструктор).

where T : struct, new(): Можна комбінувати різні обмеження для визначення більш конкретних правил для параметра типу T.

6) Поясніть призначення оператору default та його обмеження.

Опетор default відповідає за те, щоб повернути значення за замовчуванням залежно від типу. Наприклад, int – 0, string – null, тощо. Для того, щоб використовувати default, тип повинен мати значення за замовчуванням, тобто клас, інтерфейс не зможуть бути використані для цього. (Якщо вони не позначені як Nullable)

7) Розкажіть про лямбда вирази. Наведіть приклад використання лямбда-виразу.

Лямбда-вирази в C# - це короткий і зручний спосіб створювати анонімні методи або делегати без визначення окремих методів. Вони дозволяють вам передавати фрагменти коду як аргументи до методів, реалізувати функціональність на льоту і полегшують роботу з колекціями та LINQ

8) Розкрийте різниці між expression & statement лямбдами.

Основна різниця між expression і statement lambda полягає в їх призначенні та тому, як вони поводяться щодо повернення значень. Expression lambda використовується, коли потрібно обчислити та повернути значення, тоді як statement lambda використовується для виконання послідовності дій без необхідності повертати значення.

9) Розкажіть про події, і як вони реалізовані в С#.

Подія – це механізм, при якому всі об’єкти можуть підписатись, отримати сповіщення та зреагувати на нього. Події реалізують патерн Observer. Визначення події базується на основі делегату.

10) Поясніть, яким чином виконується підписання на події та скасування підписки.

Для підписки чи скасування події потрібен екземпляр делегату, який буде відповідати сигнатурі події.

Підписка на подію відбувається за допомогою оператора +=, коли скасування підписки за допомогою -=.

11) Наведіть склад класу делегату та поясніть, чим забезпечується контроль типів в делегатах.

Делегат – це тип, який представляє собою посилання на метод. Вони зберігають та викликають методи. Контроль типів забезпечується сигнатурою делегату, тому сигнатура методу має відповідати сигнатурі делегату.