**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»**

**Зображення, що містить будівля, ескіз, панорама, чорно-білий

Автоматично згенерований опис**

**Кафедра інформаційних систем та технологій**

Лабораторна робота №1

з дисципліни «Розробка програмного забезпечення на платформі .Net»

на тему:

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

|  |  |  |
| --- | --- | --- |
| Викладачк:  Бардін В. |  | Виконала:                                               Студентка групи ІС-12  Гусєва Тетяна |

Київ – 2023

**Мета лабораторної роботи** – 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:

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

**Реалізація:**

Дек являє собою двосторонню чергу, добавляти та видаляти елементи можна з початку та з кінця, тому за основу реалізації беремо двозв’язний список. Реалізуємо кожен елемент таким класом:

class DequeElement<T>

{

public DequeElement(T data)

{

Data = data;

}

public T Data { get; set; }

public DequeElement<T> Previous { get; set; }

public DequeElement<T> Next { get; set; }

}

Маємо два методи для додавання елемента.

Щоб добавити елемент в початок черги перевстановлюємо посиляння на змінну head.

public void AddFirst(T data)

{

DequeElement<T> node = new DequeElement<T>(data);

DequeElement<T> temp = head;

node.Next = temp;

head = node;

if (count == 0)

tail = head;

else

temp.Previous = node;

count++;

}

Аналогічно, щоб добавити елемент в кінець черги перевстановлюємо змінну tail.

public void AddLast(T data)

{

DequeElement<T> node = new DequeElement<T>(data);

if (head == null)

head = node;

else

{

tail.Next = node;

node.Previous = tail;

}

tail = node;

count++;

}

Маємо два методи для видалення елемента.

Для видалення елементу з початку деку потрібно перевстановити посилання на перший елемент.

public T RemoveFirst()

{

if (count == 0)

throw new InvalidOperationException();

T output = head.Data;

if (count == 1)

{

head = tail = null;

}

else

{

head = head.Next;

head.Previous = null;

}

count--;

return output;

}

Для видалення останнього елементу перевстановлюємо посилання із змінної tail на передостанній елемент.

public T RemoveLast()

{

if (count == 0)

throw new InvalidOperationException();

T output = tail.Data;

if (count == 1)

{

head = tail = null;

}

else

{

tail = tail.Previous;

tail.Next = null;

}

count--;

return output;

}

**Результат:**

Зображення, що містить текст, знімок екрана, програмне забезпечення

Автоматично згенерований опис

**Відповіді на теоретичні питання.**

1. Дайте визначання колекції. Як колекція пов’язана з інтерфейсами

IEnumerable та ICollection?

Колекція - це клас, який використовується для представлення набору подібних елементів даних у вигляді єдиної одиниці. Ці одиничні класи використовуються для групування та управління пов'язаними об'єктами.

Колекція може бути пов'язана з інтерфейсом IEnumerable, якщо вона дозволяє перебирати свої елементи, або з інтерфейсом ICollection, якщо вона дозволяє взаємодіяти з елементами колекції (додавання, видалення, отримання кількості). Реалізація інтерфейсу ICollection передбачає реалізацію методів для роботи з елементами колекції, тоді як IEnumerable вимагає реалізації лише одного методу - GetEnumerator().

2. Розкажіть про основні інтерфейси необхідні для функціонування

колекцій та LINQ.

IEnumerable (System.Collections.IEnumerable):

Основний інтерфейс для перебору елементів в колекціях. Визначає метод GetEnumerator(), який повертає об'єкт, який реалізує інтерфейс IEnumerator. Дозволяє використовувати цикли foreach для ітерації через елементи колекції.

IEnumerator (System.Collections.IEnumerator):

Визначає методи для поетапного перебору елементів колекції.Містить методи MoveNext() для переходу до наступного елемента, Reset() для скидання ітератора до початку і властивість Current, що повертає поточний елемент у колекції.

ICollection (System.Collections.ICollection):

Наслідує IEnumerable і додає функціональність, пов'язану з кількістю та додаванням / видаленням елементів в колекції. Містить методи Add, Remove, Clear і властивість Count.

IList (System.Collections.IList):

Розширює ICollection і додає функціональність індексованого доступу до елементів колекції. Дозволяє отримувати доступ до елементів за допомогою індексів (наприклад, myList[0]). Містить методи для роботи з індексами, такі як Insert, RemoveAt, IndexOf.

IEnumerable<T> (System.Collections.Generic.IEnumerable<T>):

Загальний варіант інтерфейсу IEnumerable для типізованих колекцій. Дозволяє ітерувати через колекції, де тип елементів відомий.

ICollection<T> (System.Collections.Generic.ICollection<T>):

Загальний варіант інтерфейсу ICollection для типізованих колекцій. Додає функціональність для роботи з кількістю та додаванням / видаленням типізованих елементів.

3. Розкажіть про призначення і можливі сценарії застосування

інтерфейсу IEnumerator.

IEnumerator - це інтерфейс, який визначає методи та властивості для поетапного перебору елементів колекції. Він має два основних методи: MoveNext() для переходу до наступного елемента і Reset() для скидання ітератора до початку. Також в інтерфейсі IEnumerator є властивість Current, яка повертає поточний елемент колекції. Цей інтерфейс дозволяє зручно перебирати елементи колекцій без знання їхньої конкретної реалізації.

4. Порівняйте інтерфейси IEnumerable та IAsyncEnumerable.

IEnumerable:

IEnumerable призначений для синхронного перебору даних. Всі операції, пов'язані з цим інтерфейсом, виконуються в поточному (основному) потоці виконання. Ітератори IEnumerable блокують викликаючий потік, коли вони чекають на наступний елемент, що може впливати на продуктивність та ресурси. Він використовується, коли ви працюєте з синхронними даними і не потребуєте асинхронності.

IAsyncEnumerable:

IAsyncEnumerable призначений для асинхронного перебору даних. Це корисно, коли ви маєте справу з асинхронними даними або потребуєте уникнути блокування головного потоку. Ітератори IAsyncEnumerable не блокують викликаючий потік, коли вони чекають на наступний елемент. Вони дозволяють головному потоку виконувати іншу роботу під час очікування на дані. Використовується в асинхронних сценаріях, де важлива продуктивність та відсутність блокування.

5. Розкажіть про призначення і обмеження generic типів.

Призначення generic типів: перевикористання коду, безпека типів та змення приведення типів, підвищення читабельності.

Обмеження generic типів: потрібно визначати типові параметри <T>, можливість обмежувати типовий параметр, об’ємніший обсяг коду.

6. Поясніть призначення оператору default та його обмеження.

Призначення оператору default: отримання значення за замовчуванням, ініціалізація змінних, використання в generic.

Обмеження оператору default: може бути використаний не для всіх типів даних.

7. Розкажіть про лямбда вирази. Наведіть приклад використання

лямбда-виразу.

Лямбда-вирази - це короткий і зручний спосіб визначити безіменні функції або делегати в мовах програмування, які підтримують функціональну програмування. Лямбда-вирази дозволяють передавати функції як аргументи, повертати їх як результати, а також створювати функції без необхідності оголошення окремих методів чи класів.

Func<int, int> square = x => x \* x;

int result = square(5);

Console.WriteLine(result);

Func<int, int> - це тип делегата, який приймає один аргумент типу int і повертає результат такого ж типу. Лямбда-вираз (x => x \* x) визначає анонімну функцію, яка обчислює квадрат переданого їй числа. За допомогою цього лямбда-виразу ми створюємо функцію square, яку можна викликати для обчислення квадрату числа.

8. Розкрийте різниці між expression & statement лямбдами.

Expression лямбда завжди повертає значення, часто використовуються для визначення коротких, одноразових функцій, таких як обчислення виразів у LINQ запитах чи проектування результатів.

Statement лямбда використовується для виконання операцій та не повертає значення напряму, використовуються тоді, коли потрібно виконати послідовність операцій, такі як виведення на консоль чи зміна стану об'єкта, і не потрібно повертати значення.

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

Події дозволяють об'єктам взаємодіяти один з одним і надсилати сповіщення про події, які відбуваються в системі. Події в C# дозволяють реалізовувати підписку та сповіщення між об'єктами, що забезпечує багато можливостей для реалізації шаблонів спостерігача, обробки подій та інших механізмів реакції на зміни в програмному середовищі.

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

підписки.

Підписання на подію додається до делегату обробник події за допомогою оператора +=

myObject.MyEvent += MyEventHandler;

Скасування підписки на подію виконується за допомогою оператора -=

myObject.MyEvent -= MyEventHandler;

11. Наведіть склад класу делегату та поясніть, чим забезпечується

контроль типів в делегатах.

Делегати використовуються для створення змінних, які вказують на методи. Вони дозволяють передавати методи як параметри, зберігати їх в колекціях, підписувати та відписувати від подій і так далі. Контроль типів в делегатах забезпечується шляхом визначення і оголошення типу делегата.

Склад класу делегата включає наступні компоненти: зворотний тип, ім’я делегата, список параметрів.

delegate int MathOperation(int x, int y);