ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

C#. Індексатори. Деструктори

Лекція 08

План

1. Що таке індексатори та для чого вони використовуються.
2. Порівняння індексаторів з масивами та властивостями.
3. Оголошення індексаторів. Реалізація методів get та set.
4. Використання різних типів індексів. Переваги використання індексаторів.
5. Що таке деструктори та їх роль у керуванні пам'яттю. Зв'язок деструкторів зі збирачем сміття (garbage collector).
6. Приклади

Індексатори

Індексатори – це члени класу, які дозволяють об'єктам класу індексуватися подібно до масивів. Вони забезпечують спосіб доступу до елементів класу за допомогою індексу, як до елементів масиву, використовуючи синтаксис квадратних дужок [].

Індексатори корисні, коли клас інкапсулює колекцію або список елементів, і ви хочете надати зручний спосіб доступу до цих елементів.

Індексатори і масиви

Властивості (properties)– це члени класу, які надають доступ до приватних полів класу. Вони дозволяють контролювати доступ до даних, забезпечуючи інкапсуляцію.

- Auto-implemented properties
- Full properties
- Read-only properties
- Write-only properties

Індексатори (Indexers) – це спеціальні члени класу, які дозволяють отримувати доступ до об'єктів класу, використовуючи синтаксис, схожий на доступ до елементів масиву.

Індексатори і масиви

Властивості – це ідентифікатори визначеного типу, при звертанні до якого викликаються пов’язані методи доступу. Фактично властивість – це природнє розширення поля класу. Синтаксис властивості нічим не відрізняється від поля, але при звертанні до властивості не опрацьовуються дані, а викликаються пов’язані методи.

Індексатори дозволяють звертатися до об'єктів класу як до масивів.

Особливості індексаторів

Дозволяє класам і структурам індексуватися подібно до масивів.
Використовує ключове слово this з квадратними дужками [] для визначення.
Може приймати один або кілька параметрів різних типів.
Не має імені.
В основному використовуються у класах для доступу до закритого поля-масиву

Особливості властивостей

Надає доступ до полів класу або структури через методи get і set.
Має ім'я.
Може приймати лише один параметр (значення для встановлення).
Властивість – це ідентифікатор визначеного типу, при звертанні до якого виконується код.

Індексатори і масиви

Масив – це структура даних, яка зберігає колекцію елементів одного типу в послідовних комірках пам'яті. Доступ до елементів масиву здійснюється за допомогою індексів, які починаються з 0.

Масив - це структура даних, яка зберігає фіксовану кількість елементів одного типу в послідовних комірках пам'яті.
Доступ до елементів масиву здійснюється за допомогою цілочислових індексів, починаючи з 0.
Розмір масиву фіксується при його створенні і не може бути змінений.
Масиви можуть зберігати елементи будь-якого типу, але всі елементи в одному масиві повинні бути одного типу.
Масиви використовуються для зберігання колекцій елементів, коли відома їх кількість і тип.

Індексатори і масиви

Характеристика	Масиви	Індексатори
Визначення	Структура даних, що зберігає колекцію елементів одного типу.	Члени класу, що дозволяють об'єктам індексуватися подібно до масивів.
Реалізація	Вбудований тип даних у C#.	Реалізується користувачем у класі.
Тип даних	Елементи мають однаковий тип.	Елементи можуть мати різні типи.
Розмір	Фіксований розмір при створенні.	Розмір може бути динамічним.
Доступ до елементів	Доступ до елементів здійснюється за числовим індексом.	Доступ до елементів здійснюється за допомогою індексу (будь-якого типу).
Перевірка меж	Перевірка меж виконується під час виконання.	Перевірка меж може бути реалізована користувачем.
Складність	Прості у використанні.	Більш гнучкі та потужні.
Використання	Зберігання колекцій однотипних даних.	Надання доступу до внутрішніх даних класу.

Індексатори і масиви

Індексатори схожі на **масиви** тим, що вони дозволяють отримувати доступ до елементів за індексом. Однак індексатори є членами класу, тоді як масиви є типом даних.

Індексатори схожі на **властивості** тим, що вони використовують методи get і set для доступу до даних. Однак властивості зазвичай використовуються для доступу до окремих значень, тоді як індексатори використовуються для доступу до колекцій.

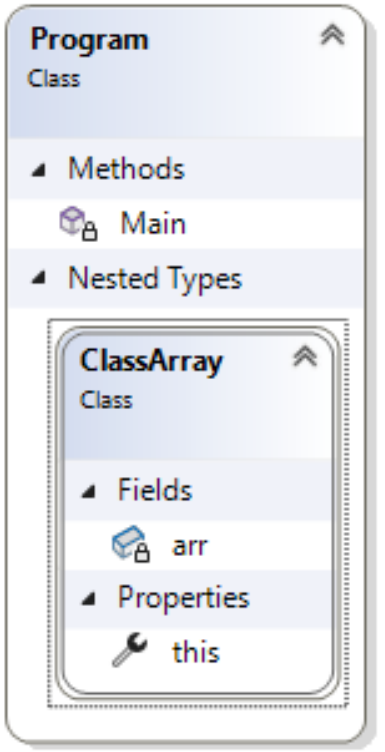
Оголошення індексаторів

Синтаксис:

```
[специфікатор доступу] <тип індексатору> this [<тип  індекс>]
{
    get   { //тіло функції одержання значень за індексами
        return <результат>;
    }
    set    {   //тіло функції встановлення значень за індексами}
}
```

Приклад

```
class ClassArray
{
   private int[] arr = { 1, 2, 3 };

   public int this [int index]
   {
      get
      {
         return arr[index];
      }
      set
      {
         arr[index] = value;
      }
   }
}
```




```
ClassArray classArray = new ClassArray();
hiddenArray[0] = 4;
int k = hiddenArray[1];
Console.WriteLine($"k={k} arr[0]={hiddenArray[0]}");
```

Індексатори

Індексатори зазвичай оголошуються з модифікатором доступу public, щоб забезпечити доступ до них ззовні класу.

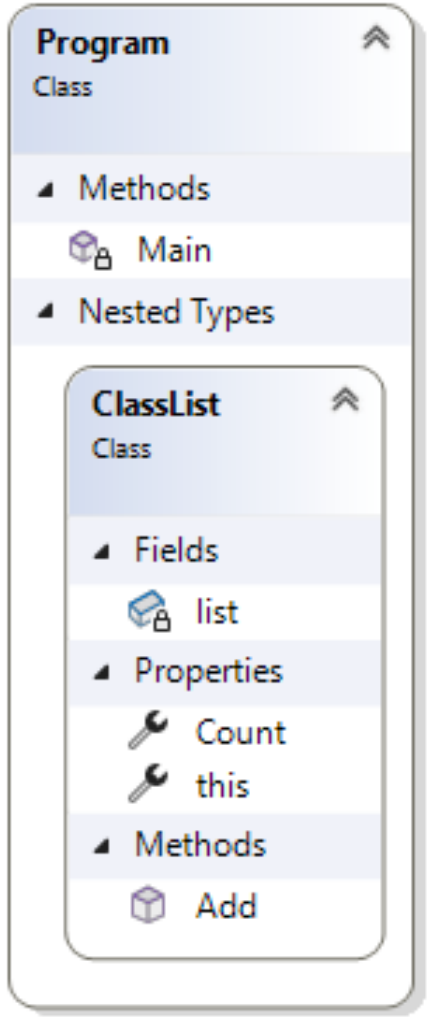
Індексатор завжди повинен мати тип повернення, відмінний від void. Він повертає значення певного типу.

Індексатор може мати як блок get, так і блок set, або тільки один з них:

- Якщо присутній лише блок get, індексатор є "тільки для читання".
- Якщо присутній лише блок set, індексатор є "тільки для запису".
- Обидва блоки не можуть бути відсутні одночасно.

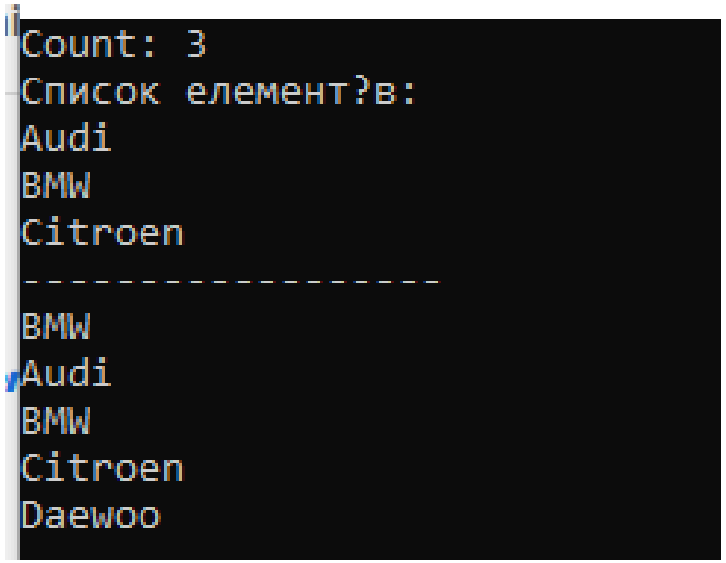
Приклад 1

```
public class ClassList
{
    private List<string> list = new List<string>();

    public string this[int index]
    {
        get { return list[index]; }
        set { list.Add(value); }
    }

    public int Count
    {
        get { return list.Count; }
    }

    public void Add(string item)
    {
        list.Add(item);
    }
}
```



Приклад 2

```
ClassList objList = new ClassList();

objList.Add("Audi");
objList.Add("BMW");
objList.Add("Citroen");

Console.WriteLine("Count: " + objList.Count);

Console.WriteLine("Список елементів:");
for (int i = 0; i < objList.Count; i++)
   Console.WriteLine(objList[i] + " ");

Console.WriteLine("----------------- ");

objList[1] = "Daewoo";
Console.WriteLine(objList[1]);
for (int i = 0; i < objList.Count; i++)
   Console.WriteLine(objList[i] + " ");
```



Приклад 1

```
public class MyDictionary
{
   private Dictionary<string, string> data = new Dictionary<string, string>();

   public string this[string key]
   {
      get { return data[key]; }
      set { data[key] = value; }
   }

   public IEnumerable<string> Keys
   {
      get { return data.Keys; }
   }
}
```

Приклад 2

```
MyDictionary[] dictionaries = new MyDictionary[3];

for (int i = 0; i < dictionaries.Length; i++)
    dictionaries[i] = new MyDictionary();


dictionaries[0]["kitten"] = "кошеня";
dictionaries[1]["dog"] = "пес";
dictionaries[2]["puppy"] = "щеня";


for (int i = 0; i < dictionaries.Length; i++)
    foreach (string key in dictionaries[i].Keys)
    {
        Console.WriteLine($"  {key}: {dictionaries[i][key]}");
    }
```

```
kitten: кошеня
dog: пес
puppy: щеня
```

Деструктор

Деструктори - це спеціальні методи, які викликаються перед тим, як об'єкт буде знищено збирачем сміття. Їх основне завдання - звільнити ресурси, які були виділені об'єктом (наприклад, файлові дескриптори, з'єднання з базою даних).

Ім'я деструктора має збігатися з ім'ям класу, але перед ним ставиться символ ~.

Деструктори не мають типу повернення і не можуть мати параметрів.

Деструктори викликаються автоматично збирачем сміття, не можна явно викликати деструктор.

Деструктори слід використовувати лише тоді, коли необхідно звільнити некеровані ресурси.
Для керованих ресурсів слід використовувати механізм IDisposable і Dispose().

Деструктор

Синтаксис:

~<ім'я класу>()
{
        // тіло деструктора
 }

```
public class Person
{
    public string Name;
    public int Age;

    public Person()
    {
        Name = "Невідомо";
        Age = 0;
    }
    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }
    ~ Person()
    {
      // Звільнення некерованих ресурсів
    }
}
```

Деструктор

Мова C# володіє механізмом звільнення ресурсів пам’яті, що називається «збирач сміття». Якщо до об’єкта не відбуваються звернення (у програмі не залишилося посилань на об’єкт), цей механізм звільняє пам’ять автоматично.
Збирач сміття викликається періодично в процесі роботи програми.
Деструктор викликається збирачем сміття перед знищенням об’єкта.
Деструктор у класі може бути тільки один.

Уникайте використання деструкторів, якщо це можливо.

Якщо вам потрібно використовувати деструктор, переконайтеся, що він виконує лише необхідні дії для звільнення некерованих ресурсів.

Загальна схема опису класу в C#

```
class <ім'я класу>
{
        [модифікатор] [const] <тип поля> <ім'я поля 1> [=<початкове значення>];
        [модифікатор] [const] <тип поля> <ім'я поля 2> [=<початкове значення>];
……..
        [модифікатор] <ім'я класу> ([<список форм. параметрів>])
        { <тіло конструктора 1>}
        [модифікатор] <ім'я класу> ([<список форм. параметрів>])
        { <тіло конструктора 2>}
……..
        [модифікатор] <тип> this [<тип_індексу> <індекс>] { <тіло індексатору>}
……..
        [модифікатор]<тип результату><ім'я методу>([<список форм. параметрів>])
        {<тіло методу 1>}
        [модифікатор]<тип результату><ім'я методу>([<список форм. параметрів>])
        {<тіло методу 2>}
……..
        ~<Ім'я класу>() {<тіло деструкторв>}
 }
```

Члени класу

Приклад частина 1

```
public class Student
 {
    private string lastName;
    private string group;
    private int[] rating;

    public Student()
    {
       lastName = "Невідомо";
       group = "Невідомо";
       rating = new int[5];
    }

    public Student(string lastName, string group, int[] rating)
    {
       this.lastName = lastName;
       this.group = group;
       this.rating = rating;
    }
```

Приклад частина 2

```
public string LastName
{
    get { return lastName; }
    set { lastName = value; }
}

public string Group
{
    get { return group; }
    set { group = value; }
}

public int this[int index]
{
    get { return rating[index]; }
    set { rating[index] = value; }
}
```

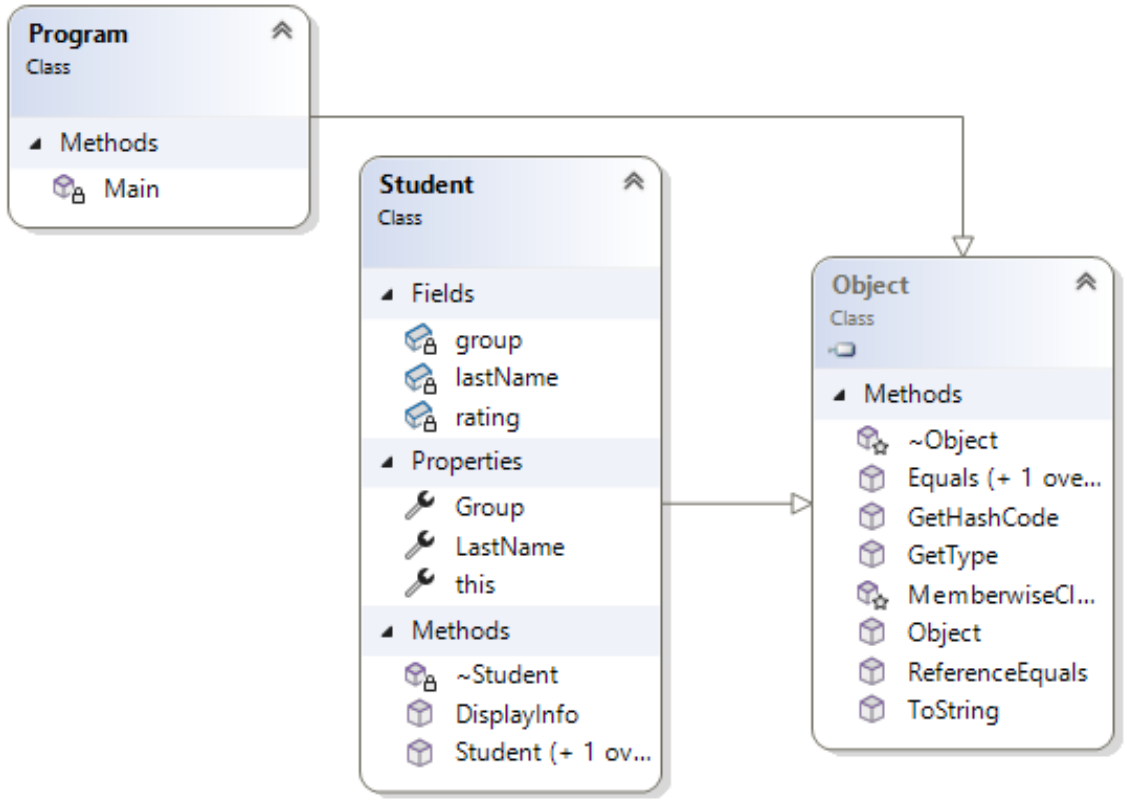
Приклад частина 3

```
public void DisplayInfo()
{
    Console.WriteLine($"Прізвище: {lastName}");
    Console.WriteLine($"Група: {group}");
    Console.Write("Оцінки: ");
    for (int i = 0; i < rating.Length; i++)
        Console.Write($"{rating[i]} ");
    Console.WriteLine();
}

~Student()
{
    // Додаткова логіка очищення ресурсів (якщо потрібно)
    Console.WriteLine($"Об'єкт Student {lastName} знищено.");
}
```

Приклад частина 4

```
Console.OutputEncoding = Encoding.Unicode;
Console.InputEncoding = Encoding.Unicode;

Student[] students = new Student[3];

students[0] = new Student("Левківський", "КН-24-1", new int[] { 5, 4, 5, 3, 4 });
students[1] = new Student("Марчук", "КН-24-1", new int[] { 4, 4, 5, 5, 5 });
students[2] = new Student("Кравченко", "КН-24-1", new int[] { 5, 4, 4, 4, 5 });

foreach (Student student in students)
    student.DisplayInfo();

students[2][2] = 5;

Console.WriteLine("\nОновлена інформація:");
students[2].DisplayInfo();
```

Приклад частина 5




```
Microsoft Visual Studio Debug Console
Прізвище: Левківський
Група: КН-24-1
Оцінки: 5 4 5 3 4
Прізвище: Марчук
Група: КН-24-1
Оцінки: 4 4 5 5 5
Прізвище: Кравченко
Група: КН-24-1
Оцінки: 5 4 4 4 5

Оновлена інформація:
Прізвище: Кравченко
Група: КН-24-1
Оцінки: 5 4 5 4 5
Об'єкт Student Кравченко знищено.
Об'єкт Student Марчук знищено.
Об'єкт Student Левківський знищено.
```