Об’єктно-орієнтоване програмування **ЛЕКЦІЯ 6**

ПРИЗНАЧЕНИЙ ДЛЯ КОРИСТУВАЧА ІНДЕКСАТОР

Це фактично оператор індекс масиву. Але в C # він не може бути перевантажений як оператор, це і призвело до появи індексаторів. Індексатор зазвичай використовується для тих класів, які являють собою колекції об'єктів. Синтаксис його наступний:
тип this [тип аргумент] {get; set;}. Можна визначити індексатор для читання, для запису і для читання / запису. Індексатор не може бути статичним.

Прийнято в якості індексу використовувати цілочисельний тип, але це не обов'язкова вимога. Це може бути рядок або будь-який інший тип даних.
Створимо простий індексатор.

```
using System.Text;

namespace prim2_lek7
{
   class Myarr
   {
       int [] arr;
      public int Length;

      public Myarr (int Size)
      {
         arr = new int [Size];
         Length = Size;
      }

      // Створюємо найпростіший індексатор
      public int this [int index]
      {
         set
         {
            arr [index] = value;
         }
```

```
        get
        {
           return arr [index];
        }
   }
   }
}
```

виклик індексатора

```
private void button1_Click (object sender, EventArgs e)
      {
         dg.Rows.Clear ();
         Myarr arr1 = new Myarr (Size: 5);
         Random ran = new Random ();

         // Ініціалізувавши кожен індекс екземпляра класу arr1
         for (int i = 0; i <arr1.Length; i ++)
         {
            arr1 [i] = ran.Next (1, 100);
            dg.Rows.Add (arr1 [i] .ToString ());
         }
      }
```

У поточному класі MyArr визначено індексатор, що дозволяє зухвалому коду ідентифікувати піделементи із застосуванням числових значень. Однак треба розуміти, що це не обов'язкова вимога методу-індексатора.

Слід особливо підкреслити, що індексатор зовсім не обов'язково повинен оперувати масивом. Його основне призначення - надати користувачеві функціональні можливості, аналогічні масиву.

На застосування індексаторів накладаються два істотні обмеження. По-перше, значення, що видається индексатором, не можна передавати методу в якості параметра ref або out, оскільки в індексатора не визначене місце в пам'яті для його зберігання. І по-друге, індексатор повинен бути членом свого класу і тому не може бути оголошений як static.

багатовимірні індексатори

Можна також створювати індексатор, який приймає кілька параметрів:

Створимо клас для роботи з двовимірним масивом:

```
class MyArr2
   {

      int [,] arr;
      // Розмірність двомірного масиву
      public int rows, cols;
      public int Length;

      public MyArr2 (int rows, int cols)
      {
         this.rows = rows;
         this.cols = cols;
         arr = new int [this.rows, this.cols];
         Length = rows * cols;
      }

      // Індексатор
      public int this [int index1, int index2]
      {
         get
         {
            return arr [index1, index2];
         }

         set
         {
            arr [index1, index2] = value;
         }
      }
   }
```

виклик індексатора

```
private void button2_Click (object sender, EventArgs e)
```

```
{
    Random ran = new Random ();
    int n = 4, m = 5;
    MyArr2 A = new MyArr2 (n, m);
    DataGridViewTextBoxColumn dgvAge;
    // Створюємо потрібну кількість колонок
    for (int i = 0; i <m; i ++)
    {
        dgvAge = new DataGridViewTextBoxColumn ();
        // Задаємо ширину колонки
        dgvAge.Width = 40;
        // Додали колонку
        dg1.Columns.Add (dgvAge);

    }
    dg1.Rows.Clear ();
    // Вказуємо контролери в який пишемо кількість рядків і стовпців
    dg1.RowCount = n;
    dg1.ColumnCount = m;
    for (int i = 0; i <n; i ++)
        for (int j = 0; j <m; j ++)
        {
            A [i, j] = ran.Next (1, 20);
            dg1.Rows [i] .Cells [j] .Value = A [i, j] .ToString ();
        }

}
```

ref і out

Використання ключового слово ref призводить до передачі аргументу по посиланню, а не за значенням. Ефект передачі по посиланню в тому, що всі зміни, що викликається методу відображаються на значенні змінної аргументу, використовуваної у виклику методу. Наприклад якщо викликає об'єкт передає вираз локальної змінної або вираження доступу до елементу масиву і викликаний метод замінює об'єкт, на який посилається параметр ref, то локальна змінна або елемент масиву волають об'єкта тепер посилатися на новий об'єкт.

Для використання параметра ref і при визначенні методу, і при виклику методу слід явно використовувати ключове слово ref.

```
void changeVar (int v)
{
  v = 1;
}
public static void Main (string [] args)
{
  int myVar = 2;
  changeVar (myVar);
  Console.WriteLine (myVar);
}
```

Виведе 2, хоча ми намагалися поміняти значення в методі.

Тепер додамо ref, і отримаємо вже 1.

```
void changeVar (ref int v)
{
  v = 1;
}
```

public static void Main (string [] args) {int myVar = 2; changeVar (ref myVar); Console.WriteLine (myVar);} Ключове слово out використовується для передачі аргументів за посиланням. Воно схоже на ключове слово ref, за винятком того, що ref вимагає ініціалізації змінної перед її передачею. Для роботи з параметром out визначення методу і викликає метод повинні явно використовувати ключове слово out.class OutExample {static void Method (out int i) {i = 44; } Static void Main () {int value; Method (out value); // Value is now 44}} Незважаючи на те що змінні, що передаються в якості аргументів out, можуть не инициализироваться перед передачею, що викликається метод повинен мати значення перед поверненням метода.Об'явленіе методу out використовується тоді, коли необхідно, щоб метод повертав кілька значень. У наступному прикладі використовується ключове слово out для повернення трьох змінних за допомогою одного виклику методу. Зверніть увагу, що третьому аргументу присвоєно значення null. Це дозволяє методам повертати значення на вибір. class OutReturnExample {static void Method (out int i, out string s1, out string s2) {i = 44; s1 = "I've been returned"; s2 = null; } Static void Main () {int value; string str1, str2; Method (out value, out str1, out str2); // Value is now 44 // str1 is now "I've been

returned" // str2 is (still) null; }} Незважаючи на те, що ключові слова ref і out є причиною іншої поведінки середовища виконання, вони не вважаються частиною сигнатури методу під час компіляції. Тому, якщо єдина відмінність між методами полягає в тому, що один метод приймає аргумент ref, а інший - out, вони не можуть бути перевантажені. Наступний приклад коди не компілюватиметься. Однак перевантаження можлива, якщо один метод приймає аргумент ref або out, а інший не приймає ні одного.Прімер 1. Створіть клас з закритим масивом, елементи якого повинні знаходитися в діапазоні [-100, 100]. Крім того, при доступі до елементу перевіряється, чи не вийшла індекс за допустимі межі. Забезпечити обчислення суми негативних елементів, метод обчислення суми позитивних елементів, метод обчислення суми всіх елементів. Використовуйте індексатор.Класссlass Arrays {public bool error = false; // Відкритий ознака помилки int [] a; // Закритий масив int length; // Закрита розмірність public Arrays (int size) // конструктор класу {a = new int [size]; length = size; } Public int Length // властивість - розмірність {get {return length; }} Public int Summa {get {int S = 0; for (int i = 0; i <length; i ++) S = S + a [i]; return S; }} Public int Summa_plus {get {int S = 0; for (int i = 0; i <length; i ++) if (a [i]> = 0) S = S + a [i]; return S; }} Public int Summa_minus {get {int S = 0; for (int i = 0; i <length; i ++) if (a [i] <= 0) S = S + a [i]; return S; }} Public int this [int i] // індексатор {get {if (i> = 0 && i <length) return a [i]; else {error = true; return 0; }} Set {if (i> = 0 && i <length && value> = -100 && value <= 100) a [i] = value; else error = true; }}} Основна програма Приклад 2. Створіть клас з закритим двовимірним масивом, елементи якого повинні знаходитися в діапазоні [-10, 10]. Крім того, при доступі до елементу перевіряється, чи не вийшла індекс за допустимі межі. Забезпечити обчислення суми негативних елементів головної діагоналі і твори позитивних елементів головної діагоналі. Використовуйте індексатор.Класс.class Arrays {public bool error = false; // Відкритий ознака помилки int [] a; // Закритий масив int length; // Закрита розмірність public Arrays (int size) // конструктор класу {a = new int [size]; length = size; } Public int Length // властивість - розмірність {get {return length; }} Public int Summa {get {int S = 0; for (int i = 0; i <length; i ++) S = S + a [i]; return S; }} Public int Summa_plus {get {int S = 0; for (int i = 0; i <length; i ++) if (a [i]> = 0) S = S + a [i]; return S; }} Public int Summa_minus {get {int S = 0; for (int i = 0; i <length; i ++) if (a [i] <= 0) S = S + a [i]; return S; }} Public int this [int i] // індексатор {get {if (i> = 0 && i <length) return a [i]; else {error = true; return 0; }} Set {if (i> = 0 && i <length && value> = -100 && value <= 100) a [i] = value; else error = true; }}} Основна програмаprivate void button1_Click (object sender, EventArgs e) {Random ran = new Random (); int n, m; try {n = Convert.ToInt16 (textBox1.Text); m = Convert.ToInt16 (textBox2.Text); DataGridViewTextBoxColumn dgvAge; // Створюємо

потрібну кількість колонок for (int i = 0; i <m; i ++) {dgvAge = new DataGridViewTextBoxColumn (); // Задаємо ширину колонки dgvAge.Width = 40; // Додали колонку dg.Columns.Add (dgvAge); } Dg.Rows.Clear (); // Вказуємо контролери в який пишемо кількість рядків і стовпців dg.RowCount = n; dg.ColumnCount = m; Arrays2 a = new Arrays2 (n, m); for (int i = 0; i <n; i ++) for (int j = 0; j <m; j ++) {a [i, j] = ran.Next (-10, 10); dg.Rows [i] .Cells [j] .Value = a [i, j] .ToString (); } If (a.error) MessageBox.Show ("Помилка введення даних"); label2.Text = "Похідна позитивних "+ a.Pr.ToString (); label3.Text =" Сума негативних "+ a.Summ.ToString ();} catch {MessageBox.Show (" Помилка введення даних ");}}