МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни «Програмування під платформу .NET» для студентів спеціальності «122 Комп‘ютерні науки» освітньої програми «Комп‘ютерні науки»

ЗАТВЕРДЖЕНО
кафедрою системотехніки
Протокол № 4
від “12” жовтня 2017 р.

Харків 2017

Методичні вказівки до лабораторних робіт з дисципліни «Програмування під платформу .NET» для студентів спеціальності «122 Комп‘ютерні науки» освітньої програми «Комп‘ютерні науки» / Упоряд.: Є.В. Губаренко. - Харків: ХНУРЕ, 2017. – 46 с.

Упорядник: Є.В. Губаренко, канд. техн. наук, доц. каф. СТ ХНУРЕ

Рецензент: Ю.В. Міщеряков, канд. техн. наук, доцент, доц. каф. СТ ХНУРЕ.

ЗМІСТ

ЗАГАЛЬНІ ПОЛОЖЕННЯ

C# – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML.

C# створювався паралельно з каркасом Framework .Net і повною мірою враховує всі його можливості – як FCL, так й CLR; повністю об'єктно-орієнтована мова; є потужною об'єктною мовою з можливостями спадкування й універсалізації; потужна бібліотека каркасів підтримує зручність побудови різних типів додатків на C#, дозволяючи легко будувати Web-служби, інші види компонентів, досить просто зберігати й одержувати інформацію з бази даних й інших сховищ даних.

Широке розповсюдження сучасної обчислювальної техніки, її інтенсивне використання в усіх сферах як засобу автоматизації інтелектуальної діяльності людини, надало додатковий імпульс на попит розробників.

Методичні вказівки містять матеріали до лабораторних робіт, які охоплюють всі основні частини мови програмування і дозволяють виконати поставлене перед студентів завдання. Для виконання лабораторної роботи необхідна наявність середовища розробки Visual Studio і ЕОМ. Також у методичних вказівках наведено відомості про організацію самостійної роботи студентів під час підготовки до виконання лабораторних робіт, порядок виконання робіт, оформлення звітів та контрольні запитання та завдання.

1 СТВОРЕННЯ ЕЛЕМЕНТАРНИХ ПРОГРАМНИХ ЗАСОБІВ НА МОВІ C#

1.1 Мета роботи

Навчитися створювати програмні засоби для рішення найпростіших задач з використанням базових команд і операторів мови C#.

1.2 Організація самостійної роботи студентів

Під час підготовки до виконання лабораторної роботи необхідно ознайомитися з базовими поняттями мови C#, структурою програми, базовими типами та операціями.

Типи даних та змінні

В C# існує наступна система типів даних:

1. bool: зберігає значення true або false. Представлений системним типом System.Boolean.
2. byte: зберігає ціле число від 0 до 255 і займає 1 байт. Представлений системним типом System.Byte.
3. sbyte: зберігає ціле число від – 128 до 127 і займає 1 байт. Представлений системним типом System.SByte.
4. short зберігає ціле число від – 32768 до 32767 і займає 2 байта. Представлений системним типом System.Int16.
5. ushort: зберігає ціле число від 0 до 65535 і займає 2 байта. Представлений системним типом System.UInt16.
6. int: зберігає ціле число від – 2147483648 до 2147486647 і займає 4 байта. Представлений системним типом System.Int32.
7. uint: зберігає ціле число від 0 до 4294967295 і займає 4 байта. Представлений системним типом System.UInt32.
8. long: зберігає ціле число від – 9223327036854775808 до 9223327036854775807 і займає 8 байт. Представлений системним типом System.Int64.
9. ulong: зберігає ціле число від 0 до 18446744073709551615 і займає 8 байт. Представлений системним типом System.UInt64.
10. float: зберігає дійсне число і займає 4 байта. Представлений системним типом System.Single.
11. double: зберігає дійсне число і займає 8 байт. Представлений системним типом System.Single.
12. decimal: зберігає десяткове дробове число і займає 16 байт. Представлений системним типом System.Decimal.
13. char: зберігає один символ в кодуванні Unicode і займає 2 байта. Представлений системним типом System.Char.
14. string: зберігає набір символів в кодуванні Unicode. Представлений системним типом System.String.

15. object: може зберігати значення любого типу даних і займає 4 байта на 32-розрядній платформі і 8 байт на 64-розрядній. Представлений системним типом System.Object, який є базовим для всіх інших типів і класів .NET.

Загальний спосіб оголошення змінних:

тип_даних назва_змінної;

Наприклад, int x;. В цьому виразі ми оголошуємо змінну *x* типу *int*, тобто *x* буде зберігати деяке число не більше 4 байт.

В якості імені змінної може бути будь-яка довільна назва, яка задовольняє наступні умови:

– містить не більше 255 символів;

– може містити будь-які цифри, букви і символ підкреслення, при цьому перший символ повинен бути буквою або символом підкреслення;

– не містить знаків пунктуації та пробілів;

– не є ключовим словом мови C#.

Задавши змінну, можна присвоїти їй значення або ініціалізувати її. Варіанти оголошення змінних:

```
bool  isEnabled = true;
int  x;
double  y=3.0;
string  hello="Hello World";
char  c='s';
int  a=4;
int  z=a+5;
```

Для неявної типізації замість назви типу даних використовується ключове слово *var*. Наприклад:

```
var stroka = "Hell to World";
var c = 20;
```

При компіляції компілятор сам визначає тип даних виходячи з присвоєного значення. Такі змінні подібні звичайним, але мають деякі обмеження:

– не можна спочатку оголосити неявно типізовану змінну, а потім її ініціалізувати;

– не можна вказати в якості значення неявно типізованої змінної *null*.

Існує явне і неявне приведення типів даних.

```
byte a = 4;
int b = a + 70;
```

В даному випадку компілятор приводить тип даних *byte* к типу *int*. Такий тип приведення називається неявним. При такому приведенні типів даних, як правило помилок не виникає.

```
int a = 4;
int b = 6;
byte c = (byte)(a+b);
```

При явному приведенні потрібно використовувати операцію проведення (операція «()»), де перед значенням вказується в дужках тип до якого потрібно привести дане значення.

Арифметичні та логічні операції

Операції бувають унарними (виконуються над однім операндом), бінарними (над двома операндами) і тернарними (над трьома операндами). Операндом є змінна або значення (наприклад, число), яка прийма участь в операції. Розглянемо види операцій.

Математичні операції

+ – операція складення двох чисел;

- – операція віднімання двох чисел;

* – операція множення;

/ – операція ділення;

% – отримання остачі від ділення двох чисел;

++ (префіксний інкремент) – *z=++y* спочатку значення змінної *y* збільшується на 1, а потім її значення присвоюється змінній *z*;

++ (постфіксний інкремент) – *z=y++* спочатку значення змінної *y* присвоюється змінній *z*, а потім значення змінної *y* збільшується на 1;

-- (префіксний декремент) – *z=--y* спочатку значення змінної *y* зменшується на 1, а потім її значення присвоюється змінній *z*;

-- (постфіксний декремент) – *z=y--* спочатку значення змінної *y* присвоюється змінній *z*, а потім значення змінної *y* зменшується на 1;

Операції порівняння:

В операціях порівняння порівнюються два операнди і повертається значення типу *bool – true*, якщо вираз вірний і *false*, якщо вираз невірний.

== – порівнює два операнди на рівність: *z=x==y*, *z* дорівнює *true*, якщо *x* дорівнює *y*, інакше – *false*.

!= – *z=x!=y*, *z* дорівнює *true*, якщо *x* не дорівнює *y*, інакше – *false*.

< – *z=x<y*, *z* дорівнює *true*, якщо *x* менше *y*, інакше – *false*.

> – *z=x>y*, *z* дорівнює *true*, якщо *x* більше *y*, інакше – *false*.

<= – *z=x<=y*, *z* дорівнює *true*, якщо *x* менше або дорівнює *y*, інакше – *false*.

>= – *z=x>=y*, *z* дорівнює *true*, якщо *x* більше або дорівнює *y*, інакше – *false*.

Також в C# існують логічні оператори, які повертають значення типу *bool*.

| – *z=x | y*, *z* дорівнює *true*, якщо або *x* або *y*, або і *x*, і *y* дорівнюють *true*, інакше – *false*.

& – *z=x & y*, *z* дорівнює *true*, якщо і *x*, і *y* дорівнюють *true*, інакше – *false*.

! – *z=!y*, *z* дорівнює *true*, якщо *y* дорівнює *false*, інакше – *false*.

^ – *z=x^y*, *z* дорівнює *true*, якщо або *x* або *y* дорівнюють *true*, інакше – *false*.

| | – *z=x| |y*, *z* дорівнює *true*, якщо або *x* або *y*, або і *x*, і *y* дорівнюють *true*, інакше – *false*.

&& – *z=x && y*, *z* дорівнює *true*, якщо і *x*, і *y* дорівнюють *true*, інакше – *false*.

В виразі *z=x | y*; будуть розраховані обидва значення – *x* і *y*. В виразі *z=x| |y*; спочатку буде розраховано значення *x*, і якщо воно дорівнює *true*, то розрахунок значення *y* вже не має сенсу, бо в будь-якому випадку *z* буде дорівнювати *true*. Значення *y* буде розраховувати тільки в тому випадку, якщо *x* дорівнює *false*. Аналогічно для пари & и &&.

Операції присвоювання:

= – присвоювання одного значення другому;

+= – *z*+=*y*; (змінній *z* присвоюється результат додавання *z* і *y*)

-= – *z*-=*y*; (змінній *z* присвоюється результат віднімання *y* із *z*)

*= – *z**=*y*; (змінній *z* присвоюється результат множення *z* і *y*)

/= – *z*/=*y*; (змінній *z* присвоюється результат ділення *z* на *y*)

%= – *z*%=*y*; (змінній *z* присвоюється остача від ділення *z* на *y*)

1.3 Порядок виконання роботи

1. Ознайомитися з теоретичним матеріалом.
2. Виконати індивідуально 12 завдань з пункту 1.4.
3. Оформити звіт
4. Здати практичну частину

1.4 Індивідуальні завдання

Завдання №1

1. Створіть змінну x1 типу int;
2. Проініціалізуйте змінну x1 будь-яким можливим значенням для її типу;
3. Виведіть значення змінної в консоль.

Завдання №2

1. Створіть змінну строкового типу str1;
2. Виведіть в консоль повідомлення: «Ваше ім‘я?»;
3. Зчитайте з консолі текст, який введе користувач, і збережіть в змінну str1.
4. Створіть змінну строкового типу str2 і проініціалізуйте її значенням: «Добрий день» + str1;
5. Виведіть значення змінної str2 в консоль.

Завдання №3

1. Створіть змінну v1 з неявною типізацією і проініціалізуйте її символьним значенням 'v';
2. Змініть значення змінної v1 на будь-яке інше;
3. Виведіть значення змінної v1 в консоль.

Завдання №4

Дано сторону квадрата *x*. Знайдіть його периметр.

Завдання №5

Дано два числа *x* і *y*. Знайти їх середнє арифметичне.

Завдання №6

Дано два концентричних круга з радіусами *R*1 і *R*2. Знайдіть площу цих кругів S1 та S2, а також площу кільця S3. В якості значення π використовуйте 3.14.

Завдання №7

Дано двозначне число. Виведіть в консоль спочатку його ліву цифру, а потім – праву цифру.

Завдання №8

С початку доби пройшло N секунд (N – ціле число). Знайдіть кількість повних годин, які пройшли с початку доби.

Завдання №9

Дано три цілих числа: A, B, C. Перевірити на істину вираз: «Число B знаходиться між числами A і C».

Завдання №10

Дано ціле додатне число. Перевірити на істину вираз «Дане число є непарним трехзначним».

Завдання №11

1. Створіть дві змінні типу int і проініціалізуйте їх будь-якими значеннями;
2. Створіть змінну типу long і збережіть в неї суму двох змінних створених раніше, виведіть результат в консоль.

Завдання №12

1. Створіть дві змінні типу long і проініціалізуйте їх будь-якими значеннями;
2. Створіть змінну типу byte і збережіть в неї добуток двох змінних створених раніше, виведіть результат в консоль.

1.5 Зміст звіту

Звіт має містити:
– мету роботи;
– завдання:
– код програми;
– результат виконання програми;
– висновки.

1.6 Контрольні питання та завдання

1. Які типи даних існують в мові C#?
2. Що таке явне і неявне приведення типів даних?
3. Що таке змінна з неявною типізацією?
4. Назвіть способи приведення строки в число.
5. Назвіть арифметичні операції, які існують в мові C#.
6. Назвіть операції порівняння, які існують в мові C#.

2 ЦИКЛИ. МАСИВИ. МЕТОДИ. СТРУКТУРИ

2.1 Мета роботи

Навчитися використовувати умовні конструкції, масиви і цикли при створенні програмних засобів, які вирішують найпростіші задачі; навчитися використовувати тип даних enum, створювати свої методи і структури засобами мови C#.

2.2 Організація самостійної роботи студентів

Під час підготовки до виконання лабораторної роботи необхідно вивчити роботу з циклами, масивами, створення та використання методів, створення структур.

Масиви

Масив являє собою набір однотипних змінних. Створення масиву схоже с оголошенням змінної: *тип_даних*[] *назва масиву*;. Наприклад:

```
int[] nums = new int[4];
nums[0] = 1;
nums[1] = 2;
nums[2] = 3;
nums[3] = 5;
Console.WriteLine(nums[3]);
```

Спочатку ми створили масив *nums*, який буде зберігати дані типу int. Далі використовуючи операцію *new*, ми виділили пам‘ять для 4 елементів масиву: *new int[4]*. Число 4 ще називається довжиною масиву.

Відлік елементів масиву починається з 0, тому в даному випадку, щоб звернутися к четвертому елементу в масиві треба використовувати вираз *nums*[3].

Існують також альтернативні шляхи ініціалізації масивів:

```
//ці два способи є рівносильними
int[] nums2 = new int[] { 1, 2, 3, 5 };
int[] nums3 = { 1, 2, 3, 5 };
```

В цьому випадку ми одразу вказуємо всі елементи масиву.

Масиви бувають одномірними і багатовимірними. В прикладах вище створювалися одномірні масиви. Приклад створення багатовимірного масиву:

```
int[] nums1 = new int[] { 0, 1, 2, 3, 4, 5 };
int[,] nums2 = { { 0, 1, 2 }, { 3, 4, 5 } };
```

Від багатовимірних масивів слід відрізняти масив масивів або так званий «зубчатий масив»:

```
int[][] nums = new int[3][];
nums[0] = new int[2];
nums[1] = new int[3];
nums[2] = new int[5];
```

Дві групи квадратних дужок вказують, що це масив масивів, тобто такий масив, який в свою чергу містить в собі інші масиви. Причому розмірність кожного з цих масивів може не співпадати.

Деякі методи і властивості масивів:

Властивість Length: дозволяє отримати кількість елементів масиву.

Властивість Rank: дозволяє отримати розмірність масиву.

Метод Array.Reverse: змінює порядок елементів масиву на зворотній.

Метод Array.Sort: сортирує елементи масиву.

Умовні конструкції

Умовні конструкції – один з базових компонентів багатьох мов програмування, які направляють роботу програми за одним із шляхів в залежності від певних умов.

В мові C# використовуються наступні умовні конструкції: if..else і switch..case.

Конструкція if/else перевіряє істинність деякої умови і в залежності від результатів перевірки виконує деякий код:

```
int num1 = 8;
int num2 = 6;
if(num1 > num2)
{
   Console.WriteLine("Число {0} більше числа {1}", num1, num2);
}
else { Console.WriteLine("Число {0} менше числа {1}", num1, num2);}
```

Після ключового слова if ставиться умова. І якщо ця умова виконується, то спрацьовує код, який поміщено в блоці if після фігурних дужок. В якості умов виступають операції порівняння.

В даному випадку перше число більше другого, тому вираз num1>num2 істинний і повертає true, тому управління переходить до строки *Console.WriteLine("Число {0} більше числа {1}", num1, num2);*.

Конструкція switch/case дозволяє обробити відразу декілька умов:

```
Console.WriteLine("Нажмите Y или N");
string selection = Console.ReadLine();
switch (selection)
{
   case "Y":
      Console.WriteLine("Вы нажали букву Y");
      break;
   case "N":
      Console.WriteLine("Вы нажали букву N");
      break;
   default:
      Console.WriteLine("Вы нажали неизвестную букву");
      break;
}
```

Після ключового слова switch в дужках вказується порівнюваний вираз. Значення цього вираз послідовно порівнюється зі значеннями, які розміщені після оператора case. Якщо збіг буде знайдено, то буде виконуватися відповідний блок case. В кінці блоку case ставиться оператор break, щоб запобігти виконанню інших блоків. Щоб опрацювати ситуацію, коли збіг не буде знайдено, треба додати блок default.

Цикли

Цикли також є управляючими конструкціями, які дозволяють в залежності від певних умов виконувати деякі дії декілька разів. В C# існують наступні види циклів: for, foreach, while, do…while.

Цикл for має наступне формальне визначення:

```
for ([ініціалізація лічильника];[умова];[зміна лічильника])
{
        //операції
}
```

Розглянемо стандартний цикл for:

```
for (int i = 0; i < 9; i++)
{
   Console.WriteLine("Квадрат числа {0} равен {1}", i, i * i);
}
```

Перша частина створення циклу – int i =0 створює і ініціює лічильник і. Лічильник необов‘язково повинен бути типу int. Перед виконанням циклу його значення буде дорівнювати 0. Друга частина – умова, при якій буде виконуватися цикл. В даному випадку цикл буде виконуватися до тих пір, поки і не досягне 9. Третя частина – збільшення лічильника на 1. В даному прикладі цикл виконається 9 разів.

Цикл foreach призначений для перебору елементів в контейнерах. Формальне оголошення циклу:

```
foreach (тип_даних назва_змінної in контейнер)
{
   // операції
}
```

Наприклад:

```
int[] array = new int[] { 1, 2, 3, 4, 5 };
foreach (int i in array)
{
   Console.WriteLine(i);
}
```

Цикл for більш гнучкий, ніж foreach. Якщо foreach послідовно бере елементи контейнера і тільки для зчитування, то в циклі for можна перескочити на декілька елементів вперед в залежності від зміни лічильника, а також змінювати елементи.

В циклі do/while спочатку виконується код циклу, а потім виконується перевірка умови в інструкції while. До тих пір поки умова істинна цикл повторюється. Наприклад:

```
int i = 6;
do
{
   Console.WriteLine(i);
   i--;
}
while (i > 0);
```

В цьому прикладі цикл виконається 6 разів, поки і не дорівнює нулю. Цикл do/while гарантує хоча б одне виконання дій, навіть якщо умова в інструкції while не є істиною.

На відміну від циклу do/while цикл while спочатку перевіряє істинність деякої умови, і якщо умова є істиною, то код циклу виконується:

```
int i = 6;
while (i > 0)
{
   Console.WriteLine(i);
   i--;
}
```

Методи (функції)

Методи представляють собою набір операторів, які виконують певні дії. Загальне оголошення методі виглядає наступним чином:

```
[модифікатори] тип_поверненого_значення назва_методу ([параметри])
{
   // тіло метода
}
```

Модифікатори і параметри не є обов‘язковими. Розглянемо на прикладі методу Main:

```
static void Main(string[] args)
{
   Console.WriteLine("привет мир!");
}
```

Ключове слово static є модифікатором. Далі йде тип поверненого значення. В даному випадку ключове слово void вказує на те, що метод нічого не повертає. Такий метод ще називають процедурою. Потім вказується назва методу – Main і в дужках параметри – string[] args. В фігурних дужках заключено тіло метода – всі дії, які він виконує.

В функції в якості типу поверненого значення замість void використовується будь-який інший тип даних. Функції також обов‘язково повинні містити оператор return, після якого ставиться значення, що повертається. Тип значення, що стоїть після return повинно співпадати з типом вказаним в прототипі функції.

Створивши методи, можна використовувати їх в програмі. Щоб викликати метод, потрібно вказати його ім‘я, а після нього в дужках значення для параметрів.

```
static void Main(string[] args)
{
   string message = Hello(); // виклик першого методу

   Console.WriteLine(message);

   Sum(); // виклик другого методу

   Console.ReadLine();
}
```

```
static string Hello()
{
   return "Hell to World!";
}
static void Sum()
{
   int x = 2;
   int y = 3;
   Console.WriteLine("{0} + {1} = {2}", x, y, x+y);
}
```

Перший метод Hello повертає значення типу string. Тому ми можемо присвоїти значення деякій змінній типу string. Другий метод – процедура Sum – виконує додавання двох чисел і виводить результат на консоль.

Передача параметрів в метод

В мові C# існує два способи передачі параметрів в метод: за значенням і за посиланням. Найбільш простий спосіб передачі параметрів є передача за значенням:

```
static int Sum(int x, int y)
{
   return x + y;
}
```

При визові цього методу в програмі обов‘язково потрібно передати на місце параметрів значення, які відповідають типу параметру:

```
int z = Sum(x, 15);
```

Параметри можуть бути також вихідними. Щоб зробити параметр вихідним, треба перед ним поставити модифікатор out.

```
static void Sum(int x, int y, out int a)
{
   a = x + y;
}
```

В цьому прикладі результат повертається не через оператор return, а через вихідний параметр. Виклик методу в програмі:

```
Sum(x, 15, out z);
```

Слід звернути увагу, що модифікатор out вказується, як при створенні методу, так і при його виклику в методі Main. Також методи, які використовують вихідні параметри обов‘язково повинні присвоїти їм певне значення.

При передачі параметрів за посиланням використовується модифікатор ref:

```
Addition(ref x, y); // виклик методу
```

Як і в випадку з out ключове слово ref використовується як при створенні методу, так і при його виклику. При передачі параметрів за значенням метод отримує не саму змінну, а її копію. При передачі параметру за посиланням метод отримує адресу змінної в пам‘яті. Якщо метод змінює значення параметра, який передається за посиланням, то також змінюється і значення змінної, яка передається на його місце.

C# дозволяє використовувати необов‘язкові параметри. Для таких параметрів необхідно задати значення за замовчуванням. Також слід враховувати,

що після необов‘язкових параметрів всі наступні параметри також повинні бути необов‘язковими.

```
static int OptionalParam(int x, int y, int z=5, int s=4)
{
   return x + y + z + s;
}
```

Так як останні два параметри оголошені як необов‘язкові, то можна опустити один з них, або обидва.

```
static void Main(string[] args)
{
   int rez = OptionalParam(2, 3);

   rez = OptionalParam(2,3,10);
}
```

Використовуючи ключове слово params, можна передати невизначену кількість параметрів.

```
static void Addition(params int[] integers)
{
   int result = 0;
   for (int i = 0; i < integers.Length; i++)
   {
      result += integers[i];
   }
   Console.WriteLine(result);
}
static void Main(string[] args)
{
   Addition(1, 2, 3, 4, 5);
   int[] array = new int[] { 1, 2, 3, 4 };
   Addition(array);
   Addition();
   Console.ReadLine();
}
```

Причому на місце параметру з модифікатором params можна передавати як окреме значення, так і масив значень, або зовсім не передавати параметри. Піссля параметру з модифікатором params не можна вказувати інші параметри.

Перерахування enum

Крім примітивних типів даних в C# є такий тип даних як enum або перерахування. Перерахування являють собою набір констант, зв‘язаних логікою. Створення перерахування відбувається за допомогою оператора enum. Після вказується назва перерахування і тип перерахування (обов‘язково цілочисельний). Якщо тип явно не вказаний, то за замовчуванням використовується тип int. Далі вказується список елементів перерахування через кому.

```
enum Days
{
   Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
}
enum Time : byte
{
```

```
    Morning, Afternoon, Evening, Night
}
```

В даних перерахуваннях кожному елементу присвоюється ціле значення, причому перший елемент має значення 0.

Структури

Крім базових елементарних типів і перерахувань в C# є складений тип даних, який називається структурою. Структури можуть вміщувати в собі звичайні змінні і методи.

Для прикладу створимо структуру Book, в якій будуть зберігатися змінні для назви, автора і року публікації книги. Крім того, структура буде містити метод для виведення інформації про книгу на консоль.

```
struct Book
{
    public string name;
    public string author;
    public int year;

    public void Info()
    {
        Console.WriteLine("Книга '{0}' (автор {1}) була видана в {2} році", name, author, year);
    }
}
```

Крім звичайних методів структура може мати спеціальний метод – конструктор, який виконує деяку початкову ініціалізацію об‘єкту, наприклад, присвоює всім полям деякі значення за замовчуванням. Якщо не використовувати конструктор, то спочатку потрібно буде проініціювати всі поля структури:

```
book1.name = "Война и мир";
book1.author = "Л. Н. Толстой";
book1.year = 1869;
```

Виклик конструктора за замовчуванням дозволяє автоматично проініціювати всі поля структури значеннями за замовчуванням.

```
Book book2=new Book(); // використання конструктора
```

Виклик конструктора має наступний синтаксис: new назва_структури ([список_параметрів]). Конструктор є звичайним методом, тільки не має значення, що повертається, і його назва завжди співпадає з іменем структури.

```
Book book=new Book("Война и мир", "Л. Н. Толстой", 1869);
book.Info();
```

Тепер не потрібно вручну присвоювати полям структури – їх ініціалізацію виконав конструктор.

2.3 Порядок виконання роботи

1. Ознайомитися з теоретичним матеріалом.
2. Виконати індивідуально завдання з пункту 2.4.
3. Оформити звіт
4. Здати практичну частину

2.4 Індивідуальні завдання

Завдання №1

Дано три цілих числа. Знайти кількість додатних і кількість від‘ємних чисел.

Завдання №2

Дано три змінні дійсного типу: A, B, C. Якщо їх значення впорядковані за зростанням (A<B<C) або за спаданням (A>B>C), то збільшить їх в два рази. Інакше замініть значення кожної змінної на протилежне. Виведіть нові значення змінних на консоль.

Завдання №3

Дано ціле число N (N>0). Створіть і виведіть масив розміром N, який вміщує N перших додатних непарних чисел: 1, 3, 5,…

Завдання №4

Створіть двомірний масив 5x5, заповніть будь-яким способом. Виведіть його елементи, які розміщені в стовбцях з непарними номерами (1, 3, 5)

Завдання №5

Дано два цілих числа A і B (A<B). Знайдіть суму квадратів всіх цілих чисел від A до B включно.

Завдання №6

Дано цілі додатні числа A і B (A<B). Виведіть всі цілі числа від A до B включно; при цьому кожне число повинно виводитись стільки разів, скільки є його значення (наприклад, число 3 виводиться 3 рази).

Завдання №7

Дано ціле число N (N>0). Якщо воно є степенем числа 3, то виведіть TRUE, інакше виведіть FALSE.

Завдання №8

Спортсмен-лижник почав тренування, пробігши в перший день 10 км. Кожен наступний день він збільшував довжину пробігу на P процентів від пробігу попереднього дня (P – дійсне, 0<P<50). За даним P визначте, після якого дня сумарний пробіг лижника за всі дні перевищить 200 км, і виведіть найдену кількість днів K (ціле) і сумарний пробіг S (дійсне).

Завдання №9

Напишіть метод, який запитує ввести два числа і потім повертає суму цих чисел.

Завдання №10

Напишіть метод InvertDigits(K), який змінює порядок слідування цифр цілого додатного числа K на зворотній (K – параметр цілого типу).

Завдання №11

Напишіть метод TrianglePS(a, P, S), який розраховує за стороною *a* рівностороннього трикутника його периметр P = 3 і площа S = *a***a**(√3)/4 (*a* – вхідний параметр, P і S – вихідні параметри; всі параметри являються дійсними).

Завдання №12

Напишіть метод MinMax(*X, Y*), який записує в змінну X мінімальне зі значень X і Y, а в змінну Y – максимальне із цих значень (X і Y – дійсні параметри, які одночасно є вхідними і вихідними).

Завдання №13

Напишіть метод, який приймає будь-яку кількість параметрів цілого типу і повертає їх суму.

Завдання №14

Напишіть рекурсивний метод DigitSum(*K*) цілого типу, який знаходить суму цифр цілого числа K, не користуючись операторами циклу.

Завдання №15

Напишіть перерахування з арифметичними операціями: add, sub, mul, div. Напишіть метод з трьома параметрами:

1 параметр – дійсне число;

2 параметр – дійсне число;

3 параметр – змінна типу перерахування, яке було створено раніше.

В залежності від значення третього параметра, метод виконує конкретну операцію над першим і другим параметром. Після цього повертає результат.

Завдання №16

Створіть структуру, яка зберігає інформацію про товари (назва товару, дата надходження, маса, ціна, назва постачальника, максимальний термін збереження) деякого складу. Структура повинна бути описана в окремому файлі проекту.

Створіть масив з трьох структур, заповніть перші дві структури за допомогою звернення до змінних, третю структуру заповніть за допомогою конструктора. Після цього виведіть всю інформацію на екран.

2.5 Зміст звіту

Звіт має містити:

– мету роботи;

– індивідуально виконані завдання:

– код програми;

– результат виконання програми;

– висновки.

2.6 Контрольні питання та завдання

1. Що таке умовна конструкція? Які види умовних конструкцій існують?
2. Що таке цикл? Описати усі види циклів, які існують в мові C#?
3. Як створити одномірний і двомірний масиви? Чим відрізняється синтаксис оголошення масиву масивів від оголошення багатовимірного масиву?
4. Опишіть загальний синтаксис оголошення масивів?
5. В чому суть модифікатора out?
6. Для чого використовується модифікатор ref?

7. Що таке необов‘язкові параметри?
8. Для чого використовують ключове слово params?
9. Що таке перерахування і для чого вони використовуються?
10. Що таке структура? В яких випадках її використовують?
11. Якими способами можна ініціювати структуру?
12. Чи може структура мати методи?

3 КЛАСИ. ПЕРЕГРУЗКА МЕТОДІВ. НАСЛІДУВАННЯ. ПОЛІМОРФІЗМ

3.1 Мета роботи

Навчитися створювати власні класи і об‘єкти; реалізувати принцип наслідування і перегрузки методів.

3.2 Організація самостійної роботи студентів

Під час підготовки до виконання лабораторної роботи необхідно вивчити роботу з класами, створення об‘єктів, реалізацію принципу наслідування і перегрузку методів.

C# є повноцінною об‘єктно-орієнтованою мовою програмування. Це означає, що програму на C# можна представити у вигляді взаємозв‘язаних взаємодіючих між собою об‘єктів.

Класи. Створення класів

Описанням об‘єкту є клас, а об‘єкт являє собою екземпляр цього класу. Клас створюється за допомогою ключового слова class:

```
class Book
{}
```

Уся функціональність класу представлена його членами – полями (полями називають змінні класу), властивостями, методами, подіями. Крім звичних методів в класах також використовуються і спеціальні методи, які називаються конструкторами. Конструктор викликається при створені нового об‘єкту даного класу. Відмінною рисою конструктора є те, що його назва повинна співпадати з назвою класу.

```
class Book
{
    public string name;
    public string author;
    public int year;

    public Book()
    { }
    public Book(string name, string author, int year)
    {
        this.name = name;
        this.author = author;
        this.year = year;
    }
    public void Info()
    {
        Console.WriteLine("Книга '{0}' (автор {1}) была издана в {2} году", name, author,
year);
    }
}
```

В даному випадку використовується два конструктори. Перший пустий, а другий заповнює поля класу початковими значеннями, які передаються через

параметри. Оскільки імена параметрів і імена полів співпадають, то використовується ключове слово this. Це ключове слово представляє собою посилання на поточний екземпляр класу.

Усі члени класу мають модифікатори доступу. Модифікатори доступу дозволяють задати допустиму область видимості для членів класу. Тобто контекст, в якому можна використовувати дану змінну або метод.

В C# використовуються наступні модифікатори доступу:

public: публічний, загальнодоступний клас або член класу. Такий член класу доступний з будь-якого місця в коді, а також із інших програм і збірок.

private: закритий клас або член класу. Такий закритий клас або член класу доступний тільки із коду в тому же класі або контексті.

protected: такий член класу доступний із будь-якого місця в поточному класі або в похідних класах.

internal: клас і член класу з подібним модифікатором доступні із будь-якого місця коду в поточній зборці, але він не доступний для інших програм і зборок.

protected internal: суміщає функціонал двох модифікаторів. Класи і члени класів з таким модифікатором доступні з поточної зборки і із похідних класів.

Створення полі класу без модифікаторі доступу рівнозначно їх створенню з модифікатором private. Класи створені без модифікатора за замовчуванням мають доступ internal.

Завдяки такій системі модифікаторів доступу можна приховувати деякі моменти реалізації класу від інших частин програми. Таке приховування називається інкапсуляцією.

Перевантаження методів і операторів

Часто виникає необхідність створити один і той же метод, але з різним набором параметрів. Наприклад, маємо наступний код:

```
class State
{
   public string Name { get; set; } // назва
   public int Population { get; set; } // населення
   public double Area { get; set; } // площа
}
```

Створимо метод для нападу на іншу країну – метод Attack. Перша реалізація методу буде приймати в якості параметру об‘єкту State – тобто країна на яку створюється напад:

```
public void Attack(State enemy)
{ }
```

Створимо нову версію даного методу з двома параметрами: країна та кількість військ:

```
public void Attack(State enemy)
{
   // код методу
}
public void Attack(State enemy, int army)
{
   // код методу
}
```

При перевантаженні операторів ми вказуємо модифікатори public static, так як метод, що перевантажується, буде використовуватися для всіх об‘єктів даного класу, далі вказується назва типу, що повертається, і після цього ключове слово operator. Потім назва оператора і параметри.

public static возвращаемый_тип operator оператор(параметры);

Наслідування

Наслідування (inheritance) є одним із ключових моментів ООП. Його сенс полягає в тому, що ми розширюємо функціональність вже існуючих класів за рахунок добавлення нового функціоналу або зміни старого. Наприклад, маємо клас Person, який описує людину.

```
class Person
{
   private string _firstName;
   private string _lastName;

   public string FirstName
   {
      get { return _firstName; }
      set { _firstName = value; }
   }
   public string LastName
   {
      get { return _lastName; }
      set { _lastName = value; }
   }

   public void Display()
   {
      Console.WriteLine(FirstName + " " + LastName);
   }
}
```

Потрібно створити клас, який описує робітника підприємства – Employee. Цей клас буде реалізовувати той же функціонал, що й клас Person, так як робітник – це людина. Раціонально зробити клас Employee похідним (наслідником або підкласом) від класу Person, який в свою чергу називається базовим класом або суперкласом.

```
class Employee : Person
{ }
```

Після двокрапки ми вказуємо базовий клас для даного класу. Для класу Employee базовим є Person і тому клас Employee наслідує всі властивості, методи, поля, які є в класі Person. Тільки конструктор не передається при наслідуванні.

Таким чином, об‘єкт класу Employee також є об‘єктом класу Person

```
static void Main(string[] args)
{
   Person p = new Person { FirstName = "Bill", LastName = "Gates" };
   p.Display();
   p = new Employee { FirstName = "Denis", LastName = "Ritchi" };
   p.Display();
   Console.Read();
```

```
}
```

Всі класи за замовчуванням можуть унаслідуватися, але існують обмеження:

– не підтримується множинне наслідування, клас може унаслідуватися тільки від одного класу.

– при створенні похідного класу потрібно враховувати тип доступу до базового класу – тип доступу до похідного класу повинен бути таким же як і у базового класу, або більш строгим.

– якщо клас був створений з модифікатором sealed, то від цього класу не можна наслідувати і створювати похідні класи.

Похідний клас може мати доступ тільки до тих членів базового класу, які об‘явлені з модифікаторами public, internal, protected або protected internal.

Конструктор не передається похідному класу при наслідуванні. Якщо в базовому класі не створений конструктор за замовчуванням без параметрів, а тільки конструкторі з параметрами, то в похідному класі обов‘язково потрібно викликати один з цих конструкторів через ключове слово base.

```
public Employee(string fName, string lName, string comp)
    : base(fName, lName)
{
  Company = comp;
}
```

Поліморфізм і перевизначення методів

Поліморфізм є третім ключовим аспектом об‘єктно-орієнтованого програмування і несе в собі можливість зміни функціонала, який унаслідувано від базового класу. В базовому класі створюють набір членів, які можуть бути перевизначені в похідному класі. Методи, які є доступними для перевизначення, в базовому класі помічаються модифікатором virtual. Такі методи називаються віртуальними.

При створені похідного класу і наслідуванні методів базового класу обирається одна зі стратегій:

1. Звичайне наслідування всіх членів базового класу в похідному.
2. Перевизначення членів базового класу в похідному.
3. Приховання членів базового класу в похідному.

Перша стратегія:

```
class Person
{
  public string FirstName { get; set; }
  public string LastName { get; set; }
  public Person(string lName, string fName)
  {
    FirstName = fName;
    LastName = lName;
  }

  public virtual void Display()
  {
    Console.WriteLine(FirstName + " " + LastName);
```

```
    }
}

class Employee : Person
{
    public string Company { get; set; }
    public Employee(string lName, string fName, string comp)
        :base(fName, lName)
    {
        Company = comp;
    }
}
```

В базовому класі метод Display() об‘явлено з модифікатором virtual, тому даний метод може бути перевизначений, але клас Employee наслідує його як є.

```
    static void Main(string[] args)
    {
        Person p1 = new Person("Bill", "Gates");
        p1.Display(); // виклик методу  Display з класу Person

        Person p2 = new Employee("Tom", "Johns", "UnitBank");
        p2.Display(); // виклик методу Display з класу
        Employee p3 = new Employee("Sam", "Toms", "CreditBank");
        p3.Display(); // виклик методу Display з класу Person
        Console.Read();
    }
}
```

Друга стратегія – перевизначення методів базового класу в похідному:

```
class Employee : Person
{
    public string Company { get; set; }
    public Employee(string lName, string fName, string comp)
        :base(fName, lName)
    {
        Company = comp;
    }
    public override void Display()
    {
        Console.WriteLine(FirstName + " " + LastName + " работает в компании "+
Company);
    }
}
```

В цьому випадку поведінка об‘єкта Employee змінюється:

```
Person p1 = new Person("Bill", "Gates");
p1.Display(); // виклик методу Display із класу Person
Person p2 = new Employee("Tom", "Johns", "UnitBank");
p2.Display(); // виклик методу Display із класу Employee
Employee p3 = new Employee("Sam", "Toms", "CreditBank");
p3.Display(); // виклик методу Display із класу Employee
```

В третій стратегії можна просто створити в похідному класі метод с таким же іменем, без перевизначення за допомогою слова override.

```
class Employee : Person
```

```
{
   public string Company { get; set; }
   public Employee(string lName, string fName, string comp)
         :base(fName, lName)
   {
      Company = comp;
   }
   public new void Display()
   {
      Console.WriteLine(FirstName + " " + LastName + " работает в компании "+
Company);
   }
}
```

В цьому випадку метод Display() в Employee приховує метод Display() із класу Person. Щоб явно приховати метод з базового класу, використовується ключове слово new, але це не обов‘язково, за замовчуванням система робить це неявно.

```
Person p1 = new Person("Bill", "Gates");
p1.Display(); // виклик метода Display з класу Person
Person p2 = new Employee("Tom", "Johns", "UnitBank");
p2.Display(); // вклик метода Display з класу Person
Employee p3 = new Employee("Sam", "Toms", "CreditBank");
p3.Display(); // виклик метода Display з класу Employee
```

Також можна заборонити перевизначення методів і властивостей. В цьому випадку необхідно створити їх з модифікатором sealed.

```
class Employee : Person
{
   public string Company { get; set; }

   public Employee(string lName, string fName, string comp)
         :base(fName, lName)
   {
      Company = comp;
   }
   public override sealed void Display()
   {
      base.Display();
      Console.WriteLine("Место работы : " + Company);
   }
}
```

При створенні методів з модифікатором sealed потрібно враховувати, що sealed використовується в парі з override, тобто тільки в перевизначених методах.

3.3 Порядок виконання роботи

1. Ознайомитися з теоретичним матеріалом.
2. Виконати індивідуально завдання з пункту 3.4.
3. Оформити звіт
4. Здати практичну частину

3.4 Індивідуальні завдання

Завдання №1

Створити клас Телевізор, в якому є поле поточний канал. Передбачте в ньому можливість перемикання каналів (методи класу): наступний канал, попередній канал, перехід до каналу по номеру. Врахуйте, що канал не може мати від‘ємний номер.

Завдання №2

Створити клас Студент, задайте в ньому поля: ім‘я, курс, чи є в нього стипендія. Створіть в класу декілька конструкторів, для можливості задання відразу усіх вказаних параметрів або декількох при створенні об‘єкту класу.

Завдання №3

Створити клас Аудіоплеєр, в якому є поле гучність звуку, для доступу к цьому полю реалізуйте властивість. Гучність може бути в діапазоні від 0 до 100.

Завдання №4

Напишіть програму – емулятор холодильника. Програма повинна дозволяти виконувати через меню в консольному додатку ті або інші дії з холодильником, наприклад, ввімкнути або вимкнути його, і т.п. При цьому після кожної дії користувача, програма повинна відобразити на екрані поточний стан холодильника і меню всіх функцій.

Клас холодильника повинен мати метод, який відображує на поточний стан, а також методи, які дозволяють змінити цей стан. Режими роботи холодильника повинні бути описані за допомогою перерахування (enum). Головний метод програми (Main) повинен виконувати задачу відображення меню і передачу команд користувача об‘єкту класу холодильник.

Завдання №5

Створити клас Круга і реалізуйте наступний функціонал:

1. Перегрузіть конструктор:

– пустий конструктор;

– запитує координати центра круга, його радіус і ініціалізує об‘єкт.

2. Перегрузіть метод отримання довжини круга:

– метод без параметрів повертає довжину круга для поточного об‘єкта;

– метод приймає радіус і повертає довжину круга для заданого радіуса.

3. Перегрузіть метод отримання об‘єкта-круга:

– метод без параметрів повертає копію поточного об‘єкта круга;

– метод приймає координати центра круга, його радіус і повертає об‘єкт круга з заданими параметрами.

4. Метод перевірки попадання точки в круг.

5. Метод перетворення поточного стану об‘єкта в символьну строку.

Завдання №6

Створити клас Геометрична фігура. Створіть в ньому загальні поля/властивості, наприклад, координати центра фігури (за допомогою конструктора повинна бути можливість задати центр).

На основі цього класу створіть два нових – Трикутник і Круг. В цих класах повинні бути свої особливі поля. В обидва класи додайте метод Draw(), в якому існує специфічна логіка малювання фігури.

Завдання №7

Створити клас Квадрат, створіть властивість для зберігання значення стороні і віртуальний метод, який повертає периметр. На основі цього класу створіть клас Куб і перевизначте метод отримання периметра.

Завдання №1

Створити клас Прямокутник з полями координат верхнього лівого (X1, Y1) і нижнього правого (X2, Y2) кутів. Перевизначте в ньому методи ToString, Equals і GetHashCode. Прямокутники рівні, якщо в них однакові координати лівого верхнього і нижнього правого кутів.

3.5 Зміст звіту

Звіт має містити:
– мету роботи;
– індивідуально виконані завдання:
– код програми;
– результат виконання програми;
– висновки.

3.6 Контрольні питання та завдання

1. Що таке клас? Що може містити клас?
2. Як створити об‘єкт класу?
3. Які існують модифікатори доступу до полів класу?
4. Що таке властивість класу?
5. В чому полягає принцип наслідування?
6. Що таке перегрузка методів?
7. Що таке поліморфізм?
8. Які існують три стратегії перевизначення методів?
9. Для чого створюють віртуальні методи?

4 ДЕЛЕГАТИ. ПОДІЇ. ЛЯМБДИ

4.1 Мета роботи

Навчитися створювати класи, які мають делегати, події, анонімні методи і лямбди. Реалізувати прослуховування подій і реагування на події.

4.2 Організація самостійної роботи студентів

Під час підготовки до виконання лабораторної роботи необхідно вивчити делегати, події, анонімні методи і лямбди та ознайомитися з прослуховуванням та реагуванням на події.

Делегати

Крім властивостей та методів класи можуть мати делегати і події. Делегати – це такі об‘єкти, які вказують на інші методи. Тобто делегати є вказівниками на методи. За допомогою делегатів ми можемо викликати певні методи у відповідь на деякі дії, що відбулися.

Методи, на які посилаються делегати, повинні мати такі ж параметри і такий же тип значення, що повертається. Створимо два делегата:

```
delegate int Operation(int x, int y);
delegate void GetMessage();
```

Для об‘явлення делегата використовується ключове слово delegate, після якого йде тип значення, що повертається, назва і параметри. Перший делегат посилається на функцію, яка в якості параметрів приймає два значення типу int і повертає деяке число. Другий делегат посилається на метод без параметрів, який нічого не повертає.

Щоб скористатися делегатом, потрібно створити його об‘єкт за допомогою конструктора, в який ми передаємо адрес метода, який викликається делегатом. Щоб викликати метод, на який вказує делегат, потрібно використати його метод Invoke. Крім того, делегати можуть виконуватися в асинхронному режимі, при цьому нам не потрібно створювати другий потік, а лише замість метода Invoke використати пару методів BeginInvoke/EndInvoke.

```
class Program
{
   delegate void GetMessage(); // 1. Об‘являємо делегат
   static void Main(string[] args)
   {
      GetMessage del; // 2. Створюємо змінну делегата
      if (DateTime.Now.Hour < 12)
      {
         del = GoodMorning; // 3. Присвоюємо змінній адрес метода
      }
      else
      {
         del = GoodEvening;
      }
      del.Invoke(); // 4. Викликаємо метод
```

```
        Console.ReadLine();
    }
    private static void GoodMorning()
    {
        Console.WriteLine("Good Morning");
    }
    private static void GoodEvening()
    {
        Console.WriteLine("Good Evening");
    }
}
```

За допомогою властивості DateTime.Now.Hour отримуємо поточний час. В залежності від часу в делегат передається адрес певного методу.

```
class Program
{
    delegate int Operation(int x, int y);
    static void Main(string[] args)
    {
        // присвоювання адреси метода через конструктор
        Operation del = new Operation(Add); // делегат вказує на метод Add
        int result = del.Invoke(4,5);
        Console.WriteLine(result);

        del = Multiply; // тепер делегат вказує на метод Multiply
        result = del.Invoke(4, 5);
        Console.WriteLine(result);

        Console.Read();
    }
    private static int Add(int x, int y)
    {
        return x+y;
    }
    private static int Multiply (int x, int y)
    {
        return x * y;
    }
}
```

В цьому випадку делегату присвоюється адрес метода через конструктор. Оскільки метод, як і делегат, має два параметра, то при виклику делегата в методі Invoke передаємо два параметра. Крім того, так як метод повертає значення int, то можна присвоїти результат роботи метода Invoke якій-небудь змінній.

Метод Invoke() при виклику делегата можна опустити і використати скорочену форму:

```
del = Multiply; // тепер делегат вказує на метод Multiply
result = del(4, 5);
```

Події

Події сигналізують систему про те, що відбулася певна дія. Події об‘являються в класі за допомогою ключового слова event, після якого йде назва делегата.

```
// Об‘являємо делегат
public delegate void AccountStateHandler(string message);
// Подія, яка виникла при виведенні грошей
public event AccountStateHandler Withdrowed;
```

Зв‘язок з делегатом означає, що метод, який оброблює дану подію, повинен приймати такі ж параметри, що й делегат, і повертати той же тип, що і делегат.

```
class Account
{
   // Об‘являємо делегат
   public delegate void AccountStateHandler(string message);
   // Подія, яка виникла при виведенні грошей
   public event AccountStateHandler Withdrowed;
   // Подія, яка виникла при додаванні на рахунок
   public event AccountStateHandler Added;
   int _sum; // Змінна для зберігання суми
   int _percentage; // Змінна для зберігання процента
   public Account(int sum, int percentage)
   {
      _sum = sum;
      _percentage = percentage;
   }
   public int CurrentSum
   {
      get { return _sum; }
   }
   public void Put(int sum)
   {
      _sum += sum;
      if (Added != null)
         Added("На рахунок поступило " + sum);
   }
   public void Withdraw(int sum)
   {
      if (sum <= _sum)
      {
         _sum -= sum;
         if (Withdrowed != null)
            Withdrowed("Сума " + sum + " знято з рахунку");
      }
      else
      {
         if (Withdrowed != null)
            Withdrowed("Недостатньо грошей на рахунку");
      }
   }
   public int Percentage
   {
      get { return _percentage; }
   }
}
```

В цьому прикладі створено дві події Withdrowed і Added. Обидві події об‘явлено як екземпляри делегата AccountStateHandler, тому для реагування на ці події потрібно метод, який приймає строку в якості параметра. Потім в методах Put і Withdraw викликаються ці події. Використання подій в основній програмі.

```
class Program
{
   static void Main(string[] args)
   {
      Account account = new Account(200, 6);
      // Додаємо обробник події
      account.Added += Show_Message;
      account.Withdrowed += Show_Message;
      account.Withdraw(100);
      // Удаляем обработчик события
      account.Withdrowed -= Show_Message;
      account.Withdraw(50);
      account.Put(150);
      Console.ReadLine();
   }
   private static void Show_Message(string message)
   {
      Console.WriteLine(message);
   }
}
```

Анонімні методи

Анонімні методи представляють собою скорочений запис методів. Інколи методи потрібні для однієї події і більше ніде не використовуються. Анонімні методи дозоляють вбудувати код там, де він викликається:

```
Account account = new Account(200, 6);
// лолаємо обробник події
account.Added += delegate(object sender, AccountEventArgs e)
{
   Console.WriteLine("Сумма транзакции: {0}", e.sum);
   Console.WriteLine(e.message);
};
```

Вбудовування відбувається за допомогою ключового слова delegate, після якого йде список параметрів і далі сам код анонімного методу. Блок анонімних методів повинен закінчуватися крапкою з комою після закритої фігурної дужки.

Лямбди

Лямбда-вирази представляють спрощений запис анонімних методів. Лямбда-вирази дозволяють створювати лаконічні методи, які можуть повертати деяке значення і які можна передати в якості параметрів в інші методи.

Лямбда-вираз має наступний синтаксис: зліва від лямбди-оператора => визначається список параметрів, а справа блок виразів.

```
class Program
{
   delegate int Square(int x); // об‘являємо делегат, який приймає int и повретає int
   static void Main(string[] args)
   {
```

```
        Square squareInt = i => i * i; // об‘єкту делегата присвоюється лямбда-вираз
        int z = squareInt(6); // використовуємо делегат
        Console.WriteLine(z); // виводимо число 36
        Console.Read();
    }
}
```

Вираз i => i * i представляє собою лямбда-вираз, де i – це параметр, а i*i – вираз. При використанні потрібно враховувати, що кожний параметр в лямбда-виразі неявно перетворюється в відповідний параметр делегата, тому типи параметрів повинні бути однаковими. Крім того, кількість параметрів повинна бути однакова. І тип значення, що повертається, повинен бути таким же, що й в делегата. Коли параметри в лямбда-виразі не потрібні, то замість них використовують пусті дужки.

Також лямбда-вираз необов‘язково повинен приймати блок операторів і виразів. Також він може приймати посилання на метод:

```
class Program
{
    delegate void message(); // делегат без параметрів
    static void Main(string[] args)
    {
        message GetMessage = () => Show_Message();

        GetMessage();
    }
    private static void Show_Message()
    {
        Console.WriteLine("Hello world!");
    }
}
```

Крім того, лямбда-вираз можна передавати в якості параметрів метода.

```
class Program
{
    delegate bool IsEqual(int x);
    static void Main(string[] args)
    {
        int[] integers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        // знайдемо суму чисел більше 5
        int result1 = Sum(integers, x => x > 5);
        Console.WriteLine(result1); // 30
        // знайдемо суму парних чисел
        int result2 = Sum(integers, x => x % 2 == 0);
        Console.WriteLine(result2);  //20
        Console.Read();
    }
    private static int Sum (int[] numbers, IsEqual func)
    {
        int result = 0;
        foreach(int i in numbers)
        {
            if (func(i))
```

```
            result += i;
        }
        return result;
    }
}
```

4.3 Порядок виконання роботи

1. Ознайомитися з теоретичним матеріалом.
2. Виконати індивідуально завдання з пункту 4.4.
3. Оформити звіт
4. Здати практичну частину

4.4 Індивідуальні завдання

Завдання №1

Напишіть «пінг-понг»:

1. Створіть 2 класи Ping і Pong;

2. Один надсилає повідомлення іншому про те, що «виконано пінг», другий – про те, що «виконано понг»;

3. Одна пара об‘єктів грає між собою.

Примітка: для додавання паузи між повідомленнями використовуйте наступний метод:

System.Threading.Thread.Sleep(10000); // Пауза тривалістю 10000 мілісекунд

Завдання №2

Напишіть два класи Мисливець і Заєць.

В класі Заєць є два числових поля, які відповідають поточному місцезнаходженню і є метод, який випадковим чином змінює дані значення (імітація руху зайця). Також в класі Заєць повинна бути подія, підписавшись на яку, можна кожний раз отримати нове місцезнаходження, коли Заєць рухається.

На дану подію необхідно підписати Мисливця і кожний раз, коли Заєць змінює своє місцезнаходження, Мисливець виводить відповідне повідомлення на екран.

Завдання №3

В завданні 2 додатково напишіть на подію Зайця анонімний метод і лямбда-вираз, які виводять відповідне повідомлення на екран.

4.5 Зміст звіту

Звіт має містити:

– мету роботи;

– індивідуально виконані завдання:

– код програми;

– результат виконання програми;

– висновки.

Контрольні питання та завдання

1. Що таке делегат? Який синтаксис його об‘явлення?
2. Для чого використовуються делегати?
3. Що таке подія? В чому полягає різниця між подією і делегатом?
4. Для чого використовують анонімні методи?
5. Що таке лямбда-вираз?
6. Який синтаксис лямбда-виразів? Де зручно використовувати лямбда-вираз?

5 РОБОТА З ФАЙЛАМИ

5.1 Мета роботи

Навчитися зчитувати інформацію з файлу, обробляти її і записувати в інший файл.

5.2 Організація самостійної роботи студентів

Під час підготовки до виконання лабораторної роботи необхідно вивчити роботу з файлами. Навчитися зчитувати інформацію з текстового файлу і записувати в файл Microsoft Word.

Зчитування і запис текстових файлів

Клас StreamReader дозволяє зчитувати текст або окремі строки тексту з текстового файлу. Серед його методів можна виділити наступні:

Close: закриває файл, який зчитується, і звільнює ресурси.

Peek: повертає наступний доступний символ, якщо символів більш нема, то повертає -1.

Read: зчитує і повертає наступний символ в числовому виді. Має перегружену версію: Read(char[] array, int index, int count), де array – масив, куди зчитуються символи, index – індекс в масиві array, починаючи з якого записуються символи, які зчитуються, і count – максимальна кількість зчитуваних символів.

ReadLine: зчитує одну строку в файлі.

ReadToEnd: зчитує весь текст з файлу.

```
string path= @"C:\SomeDir\hta.txt";
try
{
   Console.WriteLine("******зчитуємо весь файл********");
   using (StreamReader sr = new StreamReader(path))
   {
      Console.WriteLine(sr.ReadToEnd());
   }
   Console.WriteLine();
   Console.WriteLine("******зчитуємо по-строково********");
   using (StreamReader sr = new StreamReader(path, System.Text.Encoding.Default))
   {
      string line;
      while ((line = sr.ReadLine()) != null)
      {
         Console.WriteLine(line);
      }
   }
   Console.WriteLine();
   Console.WriteLine("******зчитуємо блоками********");
   using (StreamReader sr = new StreamReader(path, System.Text.Encoding.Default))
   {
      char[] array = new char[4];
```

```
            // считываем 4 символа
            sr.Read(array, 0, 4);
            Console.WriteLine(array);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
```

Для запису в текстовий файл використовують клас StreamWriter. Свою функціональність він реалізує через наступні методи:

Close: закриває файл, який записується і звільнює ресурси.

Flush:записує в файл залишені в буфері дані і очищує буфер.

Write: записує в файл дані найпростіших типів.

WriteLine: записує дані в файл і після запису додає символ закінчення строки.

```
    string readPath= @"C:\SomeDir\hta.txt";
    string writePath = @"C:\SomeDir\ath.txt";
    string text = "";
    try
    {
        using (StreamReader sr = new StreamReader(readPath, System.Text.Encoding.Default))
        {
            text=sr.ReadToEnd();
        }
        using (StreamWriter sw = new StreamWriter(writePath, false, System.Text.Encoding.Default))
        {
            sw.WriteLine(text);
        }
        using (StreamWriter sw = new StreamWriter(writePath, true, System.Text.Encoding.Default))
        {
            sw.WriteLine("Дозапись");
            sw.Write(4.5);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
```

Робота з файлом Microsoft World

Спочатку в розділі References додаємо посилання Microsoft.Office.Interop.Word. Потім додаємо наступні using:

```
    using Word = Microsoft.Office.Interop.Word;
    using System.Reflection;
```

Для роботи з інтерфейсом Word необхідно створити об‘єкти:

```
    Word._Application application;
    Word._Document document;
```

Майже всі функції Word потребують об‘єктних параметрів, навіть якщо це пусті строки і логічні значення.

```
Object missingObj = System.Reflection.Missing.Value;
Object trueObj = true;
Object falseObj = false;
```

Щоб запустити Word і відкрити в ньому шаблон, потрібно наступний код:

```
//створюємо об‘єкт додатку word
application = new Word.Application();
// створюємо шлях до файлу
Object templatePathObj = "шлях до файлу шаблону";;
try
{
   document = application.Documents.Add(ref templatePathObj, ref missingObj, ref missingObj, ref missingObj);
}
catch (Exception error)
{
   document.Close(ref falseObj, ref missingObj, ref missingObj);
   application.Quit(ref missingObj, ref missingObj, ref missingObj);
   document = null;
   application = null;
   throw error;
}
_application.Visible = true;
```

Важливо два моменти:

1. Ми створюємо некерований ресурс, який не забере збірник мусору – окремий процес в пам‘яті з додатком Word. Якщо не закрити його і не вивести на екран, то він так і залишиться висіти.

2. За замовчуванням Word запускається невидимо, на екран виводить сам програміст.

5.3 Порядок виконання роботи

1. Ознайомитися з теоретичним матеріалом.
2. Виконати завдання з пункту 5.4.
3. Оформити звіт
4. Здати практичну частину

5.4 Індивідуальні завдання

Дано вхідний файл з завданнями. Приклад вхідного файлу:

1. Розв‘язати рівняння $\left(\frac{5}{2}\right)^{3x+1} = \left(\frac{2}{5}\right)^{-4}$.

а) 5; б) 1; в) -5; г) -1; д) 6 .

2. Розв‘язати рівняння $\log_2(2x-3) = 1$.

а) -7; б) –1,75; в) -6; г) 4; д) 2,5 .

3. Знайти середнє арифметичне дійсних коренів рівняння $3^{2x} - 5 \cdot 3^{x} - 36 = 0$.

а) 1; б) 2; в) 3; г) 4; д) 1,5 .

4. Розв‘язати нерівність $\log_x \frac{75-5x}{x+5} > 1$.

Відповідь:_____

Сформувати файл Word за зразком:

7. На скільки $\frac{2}{3}$ числа 450 більше за 0,15 числа 480?

А	Б	В	Г	Д
118	150	228	300	372

8. Укажіть правильну нерівність.

А	Б	В	Г	Д
$\frac{3}{8} > \frac{5}{8}$	$\frac{7}{2} < \frac{7}{3}$	$\frac{8}{9} > \frac{9}{8}$	$\frac{5}{6} > \frac{4}{5}$	$\frac{19}{21} < \frac{6}{7}$

9. Визначте кількість усіх дробів із знаменником 28, які більші за $\frac{4}{7}$, але менші за $\frac{3}{4}$.

А	Б	В	Г	Д
шість	чотири	три	два	один

5.5 Зміст звіту

Звіт має містити:
– мету роботи;
– індивідуальне завдання:
– код програми;
– результат виконання програми;
– висновки.

5.6 Контрольні питання та завдання

1. Який клас відповідає за зчитування файлу? Назвіть його основні методи.
2. Який клас відповідає за запис у файл? Назвіть його основні методи.
3. Які об‘єкти використовуються для роботи з документом Word?
4. Про що важливо пам‘ятати при роботі з документом Word?

6 БАГАТОПОТОЧНІСТЬ ТА ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ

6.1 Мета роботи

Навчитися створювати програми з використанням багатопоточності. Отримання навичок організації паралельних обчислень.

6.2 Організація самостійної роботи студентів

Під час підготовки до виконання лабораторної роботи необхідно вивчити основні аспекти багатопоточності і організацію паралельних обчислень.

Основний функціонал для використання потоків розміщено в просторі імен System.Therding. В ньому визначено клас, що представляє окремий потік – клас Thread.

Клас Thread визначає ряд методів і властивостей, які дозволяють керувати потоком і отримувати інформацію про нього. Основні властивості класу:

Статична властивість CurrentContext дозволяє отримати контекст, в якому виконується потік.

Статична властивість CurrentThread повертає посилання на виконуваний потік.

Властивість IsAlive вказує, чи працює потік в поточний момент.

Властивість IsBackground вказує, чи є потік фоновим.

Властивість Name містить ім'я потоку.

Властивість Priority зберігає пріоритет потоку – значення перерахування ThreadPriority.

Властивість ThreadState повертає стан потоку – одне із значень перерахування ThreadState.

Деякі методи класу Thread:

Статичний метод Sleep зупиняє потік на певну кількість мілісекунд.

Метод Abort повідомляє середовище CLR про те, що треба припинити потік, проте припинення роботи потоку відбувається не відразу, а тільки тоді, коли це стає можливо. Для перевірки завершеності потоку слід опитувати його властивість ThreadState.

Метод Interrupt перериває потік на деякий час.

Метод Join блокує виконання потоку, який його викликав, до тих пір, поки не завершиться потік, для якого був викликаний даний метод.

Метод Resume відновлює роботу раніше призупиненого потоку.

Метод Start запускає потік.

Метод Suspend припиняє потік.

Статуси потоку містяться в перерахуванні ThreadState:

Aborted: потік зупинений, але поки ще остаточно не завершено.

AbortRequested: для потоку викликаний метод Abort, але зупинка потоку ще не відбулася.

Background: потік виконується у фоновому режимі.

Running: потік запущений і працює (не призупинено).

Stopped: потік завершено.

StopRequested: потік отримав запит на зупинку.

Suspended: потік призупинено.

SuspendRequested: потік отримав запит на призупинення.

Unstarted: потік ще не був запущений.

WaitSleepJoin: потік заблокований в результаті дії методів Sleep або Join.

У процесі роботи потоку його статус багаторазово може змінитися під дією методів. Так, на самому початку ще до застосування методу Start його статус має значення Unstarted. Запустивши потік, ми змінимо його статус на Running. Викликавши метод Sleep, статус зміниться на WaitSleepJoin. А застосовуючи метод Abort, ми тим самим переведемо потік в стан AbortRequested, а потім Aborted, після чого потік остаточно завершиться.

Пріоритети потік розташовуються в перерахуванні ThreadPriority:

Lowest

BelowNormal

Normal

AboveNormal

Highest

За замовчуванням потоку задається значення Normal. Однак ми можемо змінити пріоритет в процесі роботи програми. Наприклад, підвищити важливість потоку, встановивши пріоритет Highest. Середовище CLR буде читати та аналізувати значення пріоритету та на їх підставі виділяти даному потоку ту чи іншу кількість часу.

Монітори

Для синхронізації потоків ми можемо використовувати монітори, представлені класом System.Threading.Monitor.

```
class Program
{
   static int x=0;
   static object locker = new object();
   static void Main(string[] args)
   {
      for (int i = 0; i < 5; i++)
      {
         Thread myThread = new Thread(Count);
         myThread.Name = "Поток " + i.ToString();
         myThread.Start();
      }

      Console.ReadLine();
   }
   public static void Count()
   {
      Monitor.Enter(locker);
      try
      {
         x = 1;
         for (int i = 1; i < 9; i++)
```

```
            {
                Console.WriteLine("{0}: {1}", Thread.CurrentThread.Name, x);
                x++;
                Thread.Sleep(100);
            }
        }
        finally
        {
            Monitor.Exit(locker);
        }
    }
}
```

Клас AutoResetEvent

Клас AutoResetEvent також служить цілям синхронізації потоків. Цей клас є обгорткою над об'єктом ОС Windows "подія" і дозволяє переключити даний об'єкт-події із сигнального в несигнальний стан.

```
class Program
{
    static AutoResetEvent waitHandler = new AutoResetEvent(true);
    static int x=0;

    static void Main(string[] args)
    {
        for (int i = 0; i < 5; i++)
        {
            Thread myThread = new Thread(Count);
            myThread.Name = "Потік " + i.ToString();
            myThread.Start();
        }
        Console.ReadLine();
    }
    public static void Count()
    {
        waitHandler.WaitOne();
        x = 1;
        for (int i = 1; i < 9; i++)
        {
            Console.WriteLine("{0}: {1}", Thread.CurrentThread.Name, x);
            x++;
            Thread.Sleep(100);
        }
        waitHandler.Set();
    }
}
```

М'ютекси

Ще один інструмент управління синхронізацією потоків являє клас Mutex, також знаходиться в просторі імен System.Threading. Даний клас є класом-оболонкою над відповідним об'єктом ОС Windows "м'ютекс".

```
class Program
{
    static Mutex mutexObj = new Mutex();
```

```
    static int x=0;

    static void Main(string[] args)
    {
        for (int i = 0; i < 5; i++)
        {
            Thread myThread = new Thread(Count);
            myThread.Name = "Потоκ " + i.ToString();
            myThread.Start();
        }

        Console.ReadLine();
    }
    public static void Count()
    {
        mutexObj.WaitOne();
        x = 1;
        for (int i = 1; i < 9; i++)
        {
            Console.WriteLine("{0}: {1}", Thread.CurrentThread.Name, x);
            x++;
            Thread.Sleep(100);
        }
        mutexObj.ReleaseMutex();
    }
}
```

Спочатку створюємо об'єкт м'ютексу: Mutex mutexObj = new Mutex().

Основну роботу по синхронізації виконують методи WaitOne() і ReleaseMutex(). Метод mutexObj.WaitOne() призупиняє виконання потоку до тих пір, поки не буде отриманий м'ютекс mutexObj.

Після виконання всіх дій, коли м'ютекс більше не потрібен, потік звільняє його з допомогою методу mutexObj.ReleaseMutex()

Таким чином, коли виконання дійде до виклику mutexObj.WaitOne(), потік буде чекати, поки не звільниться м'ютекс. І після його отримання продовжить виконувати свою роботу.

Семафори

Ще один інструмент, який пропонує нам платформа .NET для управління синхронізацією, представляють семафори. Семафори дозволяють обмежити доступ певною кількістю об'єктів.

Наприклад, у нас така задача: є деяке число читачів, які приходять в бібліотеку три рази в день і що там читають. І нехай у нас буде обмеження, що одноразово в бібліотеці не може перебувати більше трьох читачів. Дану задачу дуже легко вирішити за допомогою семафорів:

```
using System;
using System.Threading;

namespace SemaphoreApp
{
    class Program
```

```
{
    static void Main(string[] args)
    {
        for (int i = 1; i < 6; i++)
        {
            Reader reader = new Reader(i);
        }

        Console.ReadLine();
    }
}

class Reader
{
    // создаем семафор
    static Semaphore sem = new Semaphore(3, 3);
    Thread myThread;
    int count = 3;// лічильник читачів

    public Reader(int i)
    {
        myThread = new Thread(Read);
        myThread.Name = "Читач " + i.ToString();
        myThread.Start();
    }

    public void Read()
    {
        while (count > 0)
        {
            sem.WaitOne();
            Console.WriteLine("{0} входить в бібліотеку", Thread.CurrentThread.Name);
            Console.WriteLine("{0} читає", Thread.CurrentThread.Name);
            Thread.Sleep(1000);
            Console.WriteLine("{0} покидає бібліотеку", Thread.CurrentThread.Name);
            sem.Release();
            count--;
            Thread.Sleep(1000);
        }
    }
}
```

У даній програмі читач представлений класом Reader. Він інкапсулює всю функціональність, пов'язану з потоками, через змінну Thread myThread.

Для створення семафора використовується клас Semaphore: static Semaphore sem = new Semaphore(3, 3);. Його конструктор приймає два параметри: перший вказує, якому числу об'єктів спочатку буде доступний семафор, а другий параметр вказує, якою максимальне число об'єктів буде використовувати даний семафор. В даному випадку у нас тільки три читача можуть одночасно перебувати в бібліотеці, тому максимальне число дорівнює 3.

Основний функціонал зосереджений в методі Read, який і виконується в потоці. На початку для очікування отримання семафора використовується метод sem.WaitOne(). Після того, як в семафорі звільниться місце, даний потік заповнює вільне місце і починає виконувати всі подальші дії. Після закінчення читання ми вивільняємо семафор з допомогою методу sem.Release(). Після цього в семафорі звільняється одне місце, яке заповнює інший потік.

6.3 Порядок виконання роботи

1. Ознайомитися з теоретичним матеріалом.
2. Виконати індивідуально завдання з пункту 4.4.
3. Оформити звіт
4. Здати практичну частину

6.4 Індивідуальні завдання

1. Задано файл з деякою кількістю текстових рядків. Необхідно реалізувати пошук заданого рядка у файлі. В одному потоці виконується обробка одного рядка із файлу.

2. Створити програму множення матриці на вектор. В одному потоці виконується обробка однієї строки матриці.

3. Реалізувати протистояння декількох команд. Кожна команда збільшується на випадкову кількість бійців і вбиває випадкову кількість бійців противника. В одному потоці реалізується боротьба однієї команди.

4. Створити гру, де будуть 2-3 барана та вовк. Якщо координат вовка співпадають з координатами барана, то баран зникає. Якщо координати баранів співпадають з‘являється новий баран. Всі рухаються хаотично.

5. Створити два потоки. Перший з них рахує числа Фобіначчі, другий – прості числа. Результат роботи кожного потоку зберігається в окремий файл. Після зупинки потоку програма проводить аналіз файлів, виводить їх на екран, а також показує кількість знайдених чисел Фобіначчі і простих чисел.

6. Створити два потоки. Перший поток записує в файл випадкові данні. Другий читає дані з цього файлу та виводить їх на екран.

7. Створити програму, в окремому потоці рахуючи значення W і неперервно оновлювати його в консолі. Для розрахунку числа використовувати наступну формулу W = 1 + sin(x) – 2*cos(x) + 4*sin2(x) – 8*cos2(x) +..

6.5 Зміст звіту

Звіт має містити:
– мету роботи;
– індивідуально виконані завдання:
– код програми;
– результат виконання програми;
– висновки.

Контрольні питання та завдання

1. Що таке поток?
2. Для чого використовується багатопоточність?
3. В чому полягає перевага багатопоточності?
4. Як відбувається синхронізація потоков?
5. Що таке монітори, м‘ютекси, семафори? Для чого їх використовують?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Джозеф Албахарі, C# 6.0 Справочник./Джозеф Албахарі, Бен Албахарі. – Вильямс, 2016. – 1040. – 978-5-8459-2087-4, 978-1-491-92706-9

2. Павловская Т.А., C#. Программирование на языке высокого уровня: Учебник для вузов. – СПб.: Питер, 2009. – 432 с.

3. Пахомов Б.И., C# для начинающих. – СПб: БХВ-Питербург, 2014. – 432 с.

4. Троелсен, Эндрю Язык программирования C# 5.0 и платформа .NET 4.5 / Эндрю Троелсен. – М.: Вильямс, 2015. - 633 с.

5. Шилдт, Г. Полный справочник по C# : пер.с англ. / Г.Шилдт. – М: Издательский дом "Вильямс", 2006. - 752 с.

6. Скит, Джон C# для профессионалов. Тонкости программирования / Джон Скит. – М.: Вильямс, 2014. – 608 с.