Лабораторна робота №10

Визначення особливостей застосування конструкторів

Мета роботи – придбання практичних навичок щодо роботи з конструкторами.

Дана лабораторна робота сприяє напрацюванню наступних **компетентностей** у відповідності до Національної рамки кваліфікації:

Знання:

призначення конструктора і форматів його запису;

особливості застосування оператора new до змінних типу значень;

призначення системи «збору сміття» в C# і формат запису деструктора;

призначення ключового слова this,

Уміння:

для заданої наочної області визначити клас, який інкапсулює елементи-дані, елементи-методи, а також конструктори;

написати програму, яка оброблює і виводить на екран інформацію щодо взаємодії декілька об'єктів, які створюються за допомогою конструкторів.

Комунікації:

обґрунтування рекомендацій команді учасників проекту щодо доцільності застосування відповідних конструкторів при розробці класів;

робота в команді над окремими класами, які містять конструктори.

Автономність і відповідальність:

прийняття рішення щодо доцільності включення в клас певного типу конструктора;

самостійне обґрунтування можливих варіантів реалізацій відповідних конструкторів.

Основні положення

У попередніх лабораторних роботах змінні кожного Building-об'єкта встановлювалися «вручну» за допомогою наступної послідовності інструкцій:

```
house.occupants = 4;
```

```
house.area = 2500;
house.floors = 2;
```

Професіонал ніколи б не використовував подібний підхід. І справа не стільки в тому, що таким чином можна попросту «забути» про один або декількох даних, скільки в тому, що існує набагато більш зручний спосіб це зробити. Цей спосіб - використання конструктора.

Конструктор ініціалізує об'єкт при його створенні. Він має ім'я, що і сам клас, а синтаксично подібний до методу. Однак у визначенні конструкторів не вказується тип значення.

Формат запису конструктора такий:

```
доступ ім'я_класу ()
        {
        // Тіло конструктора
        }
```

Зазвичай конструктор використовується, щоб надати змінним екземпляру, який визначено у класі, початкові значення або виконати початкові дії, необхідні для створення повністю сформованого об'єкта. Крім того, зазвичай в якості елемента доступ використовується модифікатор доступу public, оскільки конструктори, як правило, викликаються поза їх класу.

Всі класи мають конструктори незалежно від того, визначте ви їх чи ні, оскільки C# автоматично надає конструктор за замовчуванням, який ініціалізує всі змінні-елементи, що мають тип значень, нулями, а змінни- елементи посилального типу -- null-значеннями. Але якщо ви визначите власний конструктор, конструктор за замовчуванням більше не використовується.

Приклад 1. Використання простого конструктора.

```
class MyClass
{
   public int x;
   public MyClass( )
   {
      x = 10;
   }
```

```
}
class ConsDemo
{
   public static void Main( )
   {
      MyClass t1 = new MyClass();
      MyClass t2 = new MyClass();
      Console.WriteLine("t1.x = {0}  t2.x = {1}", t1.x, t2.x);
   }
}
```

Результат виконання програми.

```
t1.x = 10  t2.x = 10
```

У цьому прикладі програми конструктор класу MyClass має наступний вигляд:

```
public MyClass ()
{
       x = 10;
}
```

Зверніть увагу на public-визначення конструктора, яке дозволяє викликати його з коду, визначеного поза класом MyClass. Цей конструктор присвоює змінної екземпляру значення 10.

Конструктор MyClass () викликається оператором new при створенні об'єкта класу MyClass.

Наприклад, при виконанні рядка

```
MyClass t1 = new MyClass ();
```

для об'єкта t1 викликається конструктор MyClass (), який присвоює змінної екземпляру t1.x значення 10. Те ж саме справедливо і щодо об'єкта t2, тобто в результаті створення об'єкта t2 значення змінної екземпляру t2.x також стане рівним 10.

Конструктори з параметрами.

У попередньому прикладі використовувався конструктор без параметрів. Але частіше доводиться мати справу з конструкторами, які беруть один або декілька параметрів.

Параметри вносяться в конструктор точно так само, як в метод: для цього достатньо оголосити їх усередині круглих дужок після імені конструктора.

Приклад 2. Використання конструктора з параметрами.

```
using System;
class MyClass
        {
        public int x;
        public MyClass (int i)
                {
                        x = i;
                }
        }
class ParmConsDemo
{
        public static void Main ()
        {
        MyClass t1 = new MyClass (10);
        MyClass t2 = new MyClass (88);
        Console.WriteLine("t1.x = {0}  t2.x = {1}", t1.x, t2.x);
         }
}
```

Результат виконання програми.

```
t1.x = 10  t2.x = 88
```

В конструкторі MyClass () цієї версії програми визначено один параметр з ім'ям «i», який використовується для ініціалізації змінної екземпляру. Таким чином, при виконанні рядку коду

```
MyClass  t1 = new  MyClass  (10);
```

параметру «i» передається значення 10, яке потім присвоюється змінної екземпляру x.

Приклад 3. Додавання конструктора з параметрами в клас Building.

```
using System;
class Building
{
```

```
        public int floors;          // Кількість поверхів
        public int area;            // Загальна площа основи будівлі
        public int occupants;    // Кількість мешканців
        public Building (int f, int a, int o)
                {
                floors = f;
                area = a;
                occupants = 0;
                }
        public int maxOccupant (int minArea)
        {
        return area / minArea;
        }
}
// Використовуємо конструктор Building з парметрами.
class BuildingDemo
{
        public static void Main ()
        {
        Building house = new Building (2, 2500, 4);
        Building office = new Building (3, 4200, 25);
        Console.WriteLine ("Максимальне число осіб для дому, \n" +
        "якщо на кожного повинно доводиться " +
        300 + " квадратних метрів: " +
        house.maxOccupant (300));
        Console.WriteLine ("Максимальне число осіб для офісу, \n" +
        "якщо на кожного повинно доводиться " +
        200 + " квадратних метрів: " +
        office.maxOccupant (200));
        }
}
```

Результат виконання програми.

```
Максимальне число осіб для дому,
якщо на кожного повинно доводиться 300 квадратних метрів: 8
Максимальне число осіб для офісу,
якщо на кожного повинно доводиться 200 квадратних метрів: 21
```

Використання оператора new.

Тепер, коли ви більше знаєте про класи і їх конструктори, можна детальніше ознайомитися з оператором new. Формат його такий:

змінна_типу_класу = new ім'я_класу ();

Тут елемент «змінна_типу_класу» означає ім'я створюваної змінної типу класу.

Під елементом «ім'я_класу» розуміється ім'я реалізованого в об'єкті класу. Ім'я класу разом з наступною за ним парою круглих дужок - це конструктор реалізованого класу.

Якщо в класі конструктор не визначено явним чином, оператор new буде використовувати конструктор за замовчуванням, який надається засобами мови C#.

Таким чином, оператор new можна використовувати для створення об'єкта будь-якого «класового» типу.

Оскільки обсяг пам'яті комп'ютера обмежений, імовірна ситуація, коли оператор new не зможе виділити область, необхідну для створюваного об'єкта, через її відсутність в достатній кількості. У цьому випадку виникне виняткова ситуація відповідного типу.

Застосування оператора new до змінних типу значень.

У C# змінна типу значення містить власне значення. Під час компіляції програми компілятор автоматично виділяє пам'ять для зберігання цього значення. Отже, немає необхідності використовувати оператор new для явного виділення пам'яті.

І навпаки, в змінних посилального типу зберігається посилання на об'єкт, а пам'ять для зберігання цього об'єкта виділяється динамічно, тобто під час виконання програми.

Наприклад: int i = new int ();

У цьому випадку викликається конструктор за замовчуванням для типу int, який ініціалізує змінну нулем.

У загальному випадку виклик оператора new для будь-якого нессилочного типу означає виклик конструктора за замовчуванням для відповідного типу. Але в цьому випадку динамічного виділення пам'яті не відбувається. Більшість програмістів не використовують оператор new з нессилочнимі типами.

Збір «сміття» і використання деструкторів.

При використанні оператора new об'єктам динамічно виділяється пам'ять з пулу вільної пам'яті.

Обсяг буфера динамічно виділеної пам'яті не нескінченний, і рано чи пізно вільна пам'ять може вичерпатися. Отже, результат виконання оператора new може бути невдалим через нестачу вільної пам'яті для створення бажаного об'єкта. Тому одним з ключових компонентів схеми динамічного виділення пам'яті є відновлення вільної пам'яті від невикористовуваних об'єктів, що дозволяє робити її доступною для створення наступних об'єктів.

У багатьох мовах програмування звільнення раніше виділеної пам'яті виконується вручну. Однак в C# ця проблема вирішується по-іншому, а саме з використанням системи збору сміття.

Система збору сміття C# автоматично повертає пам'ять для повторного використання, діючи непомітно і без втручання програміста. Її робота полягає в наступному.

Якщо не існує жодного посилання на об'єкт, то передбачається, що цей об'єкт більше не потрібен, і займана ним пам'ять звільняється.

Цю (відновлену) пам'ять знову можна використовувати для розміщення інших об'єктів.

Система збору сміття діє тільки спорадично під час виконання окремої програми. Ця система може і не діяти: вона не «включається» лише тому, що існує один або декілька об'єктів, які більше не використовуються в програмі. Оскільки на збір сміття потрібен певний час, динамічна система C# активізує цей процес тільки по необхідності або в спеціальних випадках. Таким чином, ви навіть не будете знати, коли відбувається збір сміття, а коли - ні.

Деструктори.

Засоби мови C# дозволяють визначити метод, який повинен викликатися безпосередньо перед тим, як об'єкт буде остаточно зруйнований системою збору сміття. Цей метод називається деструктором, і його можна використовувати для забезпечення гарантії «чистоти» ліквідації об'єкту. Наприклад, ви могли б використовувати деструктор для гарантованого закриття файлу, відкритого деяким об'єктом.

Формат запису деструктора такий:

```
~ ім'я_класу ()
{
        // Код деструктора
}
```

Елемент «ім'я_класу» тут означає ім'я класу. Таким чином, деструктор оголошується подібно конструктору за винятком того, що його імені передує символ «тильда» (~). Подібно конструктору, деструктор не повертає значення.

Щоб додати деструктор в клас, досить включити його як елемент. Він викликається в момент, що передує процесу утилізації об'єкта. У тілі деструктора ви вказуєте дії, які, на вашу думку, повинні бути виконані перед руйнуванням об'єкта.

Важливо розуміти, що деструктор викликається тільки перед початком роботи системи збору сміття і не викликається, наприклад, коли об'єкт виходить за межі області видимості. Це означає, що ви не можете точно знати, коли буде виконаний деструктор. Однак точно відомо, що всі деструктори будуть викликані перед завершенням програми.

Використання деструктора демонструється в наступній програмі, яка створює і руйнує велику кількість об'єктів. У певний момент виконання цього процесу буде активізований збір сміття, а значить, викликані деструктори об'єктів, що руйнуються.

Приклад 4. Демонстрація використання деструктора.

```
using System;
class Destruct
{
   public int x;

   public Destruct(int i)
   {
      x = i;
   }
   // Викликається при утилізації об'єкта.
   ~Destruct()
   {
      Console.WriteLine("деструктуризація  " + x);
```

```
    }
    // Метод створює об'єкт, який негайно руйнується.
    public void generator(int i)
    {
        Destruct obiekt = new Destruct(i);
    }
}
class DestructDemo
{
    public static void Main()
    {
        int count;
        Destruct ob = new Destruct(0);
        /*Тепер згенеруємо велике число об'єктів. У якийсь момент часу почнеться збір сміття. Зауваження: можливо, для активізації цього процесу вам прийдетеся збільшити кількість генеруються об'єктів.
        */
        for (count = 1; count < 5; count++)
            ob.generator(count);
        Console.WriteLine("Готово");
    }
}
```

Результат виконання програми.

```
Готово
деструктуризація  4
деструктуризація  0
деструктуризація  3
деструктуризація  2
деструктуризація  1
```

Ключове слово this/

При виклику методу йому автоматично передається неявно заданий аргумент, який являє собою посилання на зухвалий об'єкт (тобто об'єкт, для якого викликається метод). Це посилання і називається ключовим словом this.

Щоб зрозуміти сенс посилання this, розглянемо спочатку програму, що створює клас Rect, який інкапсулює значення ширини і

висоти прямокутника і включає метод area (), що обчислює площу прямокутника.

Приклад 5.

```
using System;
class Rect
{
   public int width;
   public int height;
   public Rect(int w, int h)
   {
      width = w;
      height = h;
   }
   public int area()
   {
      return width * height;
   }
}
class UseRect
{
   public static void Main()
   {
      Rect r1 = new Rect(4, 5);
      Rect r2 = new Rect(7, 9);
      Console.WriteLine("Площа прямокутника r1:  " + r1.area());
      Console.WriteLine("Площа прямокутника r2:  " + r2.area());
   }
}
```

Результат виконання програми.

```
Площа прямокутника r1:  20
Площа прямокутника r2:  63
```

Як вам уже відомо, всередині методу можна отримати прямий доступ до інших членів класу, тобто без зазначення імені об'єкта або класу. Таким чином, усередині методу area () інструкція

```
return width * height;
```

означає, що буде виконана операція множення змінних width і height, які пов'язані із зухвалим об'єктом, і метод поверне їх здобуток. Але та ж сама інструкція може бути переписана наступним чином:

```
return this.width * this.height;
```

Тут слово this посилається на об'єкт, для якого викликається метод area ().

Отже, вираз this.width посилається на копію змінної width цього об'єкта, а вираз this.height - на копію змінної height того ж об'єкта.

Приклад 5. Застосування повного класу Rect, який написано з використанням посилання this:

```
using System;
class Rect
{
   public int width;
   public int height;
   public Rect(int w, int h)
   {
      this.width = w;
      this.height = h;
   }
   public int area()
   {
      return this.width * this.height;
   }
}
class UseRect
{
   public static void Main()
   {
      Rect r1 = new Rect(4, 5);
      Rect r2 = new Rect(7, 9);
      Console.WriteLine("Площа прямокутника r1:  " + r1.area());
      Console.WriteLine("Площа прямокутника r2:  " + r2.area());
```

```
    }
}
```

Результат виконання програми.

```
Площа прямокутника r1:  20
Площа прямокутника r2:  63
```

Насправді жоден C# - програміст не використовує посилання this так, як показано в цій програмі, оскільки це не дає ніякого виграшу, та й стандартна форма виглядає простіше. Однак з this можна іноді отримати користь. Наприклад, синтаксис C# допускає, щоб ім'я параметра або локальної змінної збігалося з ім'ям змінної екземпляру. У цьому випадку локальне ім'я буде приховувати змінну екземпляру. І тоді доступ до прихованої змінної екземпляру можна отримати за допомогою посилання this.

Наприклад, наступний фрагмент коду (хоча його стиль написання не рекомендується до застосування) являє собою синтаксично допустимий спосіб визначення конструктора Rect ().

```
public Rect (int width, int height)
{
        this.width = width;
        this.height = height;
}
```

У цій версії конструктора імена параметрів збігаються з іменами змінних екземпляра, в результаті чого за першими ховаються другі, а ключове слово this якраз і використовується для доступу до прихованих змінних екземпляра.

Як буде показано в наступних лабораторних роботах ключове слово this широко використовується при опису графічних додатків.

Порядок виконання лабораторної роботи

Загальна частина.

1. Набрати, відкомпілювати і запустити на виконання прикладі програм, які були наведені в розділі «Основні положення» даної лабораторної роботи.

2. Проекспериментуйте з програмами:

Зверніть увагу на особливості визначення і виклику конструкторів, а також на спосіб передачі в них параметрів.

Індивідуальна частина.

Модифікувати раніше розроблену індивідуальну програму для попередньої лабораторної роботи №2 (її структура аналогічна програмі, наведеною в прикладі 4 (див. лабораторну роботу №2).

Для чого слід додати в раніш налагоджену програму параметризований конструктор, який при створенні об'єкта автоматично повинен ініціалізувати відповідні поля (тобто змінні екземпляра). В результаті має бути отримана програма подібна прикладу 3 для поточної роботи.

Зміст звіту

1. Титульний лист.
2. Цілі лабораторного заняття і вказівка, які навички та вміння передбачається отримати в результаті його виконання.
3. Тексти налагоджених програм загальної частини лабораторного заняття з необхідними коментарями і результатом виконання.
4. Текст налагодженої програми з результатом виконання контрольних прикладів індивідуального завдання.
6. Висновки.

Контрольні питання

1. Чим відрізняються методи від конструкторів?
2. Коли доцільно використовувати конструктори?
3. Які особливості застосування оператора new до змінних типу значень?
4. Опішить призначення системи «збору сміття» в C# і формат запису деструктора;
5. Коли доцільно застосовувати ключове слова this?