

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №1**

з дисципліни “Програмування 1”

тема “C# .Net. Реалізація основних принципів ООП мовою C#”

|  |  |  |
| --- | --- | --- |
| Виконав  студент 2 курсу  групи КП-02  Чекурда Андрій Віталійович  (*прізвище, ім’я, по батькові*) |  |  |

Київ 2021

**Мета роботи**

Ознайомитися з основами об’єктно-орієнтованого підходу до створення ПЗ у мові С#, створенням класів, об’єктів, механізмами інкапсуляції, наслідування та поліморфізму. Вивчити механізм управління ресурсами, реалізований у .Net.

**Постановка завдання**

Побудувати ієрархію класів, що відтворюватимуть відношення наслідування між об’єктами реального світу (кількість класів >= 5). При цьому:

1.Забезпечити наявність у класах полів та методів з різними модифікаторами доступу, пояснити свій вибір (1 бал).

2.Забезпечити наявність у класах властивостей: складніше, ніж просто get;set;, обґрунтувати доцільність створення властивості (1 бал).

3.Створити для розроблюваних класів такі конструктори (2 бали):

- конструктор за замовчуванням;

- конструктор з параметрами;

- приватний конструктор;

- статичний конструктор.

Продемонструвати, яким чином викликаються конструктори базового та дочірнього класів.

4.Використати віртуальні та перевизначені методи (1 бал).

5.Додати до класів методи, наявність яких дозволятиме управляти знищенням екземплярів цих класів (2 бали):

a.реалізувати інтерфейс IDisposable;

b.створити деструктори;

c.забезпечити уникнення конфліктів між Dispose та деструктором.

6.Забезпечити виклики методів GC таким чином, щоб можна було простежити життєвий цикл об’єктів, що обробляються (зокрема, використати методи Collect, SupressFinalize, ReRegisterForFinalize, GetTotalMemory, GetGeneration, WaitForPendingFinalizers). Створити ситуацію, яка спровокує примусове збирання сміття GC (2 бали).

### **UML діаграма класів**

|  |
| --- |
|  |

**Фрагменти коду для демонстрації виконання задач**

1) Поля та методи з різними модифікаторами доступу:

|  |
| --- |
| protected int timeOnField;  private bool disposed = false;  public static readonly int genTime = 90; |

2) Забезпечити наявність у класах властивостей, складніших, ніж get; set:

|  |
| --- |
| public int Physical  {  get  {  return physical;  }  set  {  if (value < 0 || value > 100)  {  physical = -1;  }  else  {  physical = value;  }  }  } |

3) Створити такі конструктори:

|  |  |
| --- | --- |
| protected() | public(params) |
| protected Footballer()  {  WriteLine("Entering data from base class");  this.name = "FB Player";  this.pace = 65;  this.kicking = 44;  this.physical = 100;  this.timeOnField = 90;  } | public Footballer(int timeOnField, int physical)  {  WriteLine("Public constructor with parameters is called");  this.name = "FB Player1";  this.pace = 87;  this.kicking = 71;  this.physical = physical;  this.timeOnField = timeOnField;  } |

|  |  |
| --- | --- |
| private(params) | static |
| private Forward(double accuracy, Forward forward)  {  WriteLine("Private constructor called");  this.accuracy = accuracy + 5;  this.name = forward.name;  this.pace = forward.pace + 5;  this.physical = forward.physical + 5;  this.timeOnField = forward.timeOnField + 5;  this.kicking = forward.kicking + 5;  this.dribbling = forward.dribbling + 5;  this.finishing = forward.finishing + 5;  this.penalty = forward.penalty + 5;  this.longshot = forward.longshot + 5;  } | static Footballer()  {  WriteLine("static constructor was called");  goaldifference = 0;  } |

4) Використати віртуальні та перевизначені методи:

|  |  |
| --- | --- |
| virtual | override |
| public virtual void DoAction(string action)  {  WriteLine("Some info");  } | public override void DoAction(string action)  {  WriteLine($"What a wonderful {action} from {nameof(Defender)}");  } |

5) Додати до класів методи, наявність яких дозволятиме управляти знищенням екземплярів цих класів:

a.реалізувати інтерфейс IDisposable;

b.створити деструктори;

c.забезпечити уникнення конфліктів між Dispose та деструктором.

|  |
| --- |
| public abstract class Footballer : IDisposable |
| ~Footballer()  {  CleanUp(false);  WriteLine("main destructor called");  } |
| public void Dispose()  {  CleanUp(true);  GC.SuppressFinalize(this);  }  protected virtual void CleanUp(bool disposing)  {  if (!disposed)  {  if (disposing)  {  Console.WriteLine("Releasing managed resources in base class");  this.name = null;  this.pace = 0;  this.physical = 0;  this.kicking = 0;  this.timeOnField = 0;  }  Console.WriteLine("Releasing unmanaged resources in base class");  disposed = true;  }  } |

6) Забезпечити виклики методів GC таким чином, щоб можна було простежити життєвий цикл об’єктів, що обробляються:

|  |
| --- |
| static void ProcessDemonstrateGC()  {  Defender def = new Defender();  WriteLine("---------------------Checking GC Generations--------------------------");  WriteLine($"Generation of {nameof(def)}: {GC.GetGeneration(def)}");  WriteLine($"Total memory: {GC.GetTotalMemory(false)} bytes");  GC.Collect(0);  GC.WaitForPendingFinalizers();  WriteLine();  WriteLine($"Generation of {nameof(def)}: {GC.GetGeneration(def)}");  WriteLine($"Total memory: {GC.GetTotalMemory(false)} bytes");  GC.Collect(2);  WriteLine();  WriteLine($"Generation of {nameof(def)}: {GC.GetGeneration(def)}");  WriteLine($"Total memory: {GC.GetTotalMemory(false)} bytes");  WriteLine("--------------------------Checking over-------------------------------");  WriteLine("-------------------Checking GC of many elements-----------------------");  for(int i = 0; i < 20; i++)  {  Defender def1 = new Defender();  }  Console.WriteLine($"Total memory: {GC.GetTotalMemory(false)} bytes");  Console.WriteLine("Collecting");  GC.Collect();  GC.WaitForPendingFinalizers();  Console.WriteLine($"Total memory: {GC.GetTotalMemory(false)} bytes");  WriteLine("--------------------------Checking over-------------------------------");  } |

**Висновки**

Виконавши лабораторну роботу, я набув вмінь ієрархічно будувати класи та обмежувати доступ до полів задля збереження інформації. Також під час роботи було використано декілька конструкторів різних типів, практика з якими допомогла глибше зрозуміти їх використання. Основна увага була приділена збиранні так званого сміття та знищенню екземплярів класів за допомогою garbage collector та деструкторів, що також дозволило на практиці зрозуміти роботу самого колектора та попрацювати з IDisposable.