

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 №2**

з дисципліни “Програмування 1”

тема “C# .Net. Розширені можливості реалізації ООП у мові C#. Події**.**”

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

Київ 2021

**Мета роботи**

Ознайомитися з такими можливостями мови програмування C# як абстрактні класи, інтерфейси, делегати. Вивчити механізми оброблення подій у C#, а також можливості, які мають методи-розширення.

**Постановка завдання**

Для ієрархії класів, побудованої в лабораторній роботі №1, реалізувати:

1. Множину інтерфейсів. При чому один з класів повинен реалізовувати щонайменше 2 інтерфейси. Також продемонструвати реалізацію explicit implementation інтерфейса, обгрунтувати її використання **(1 бал)**.
2. Абстрактний клас. Забезпечити його наслідування. Наявність в цьому класі абстрактних методів - обов’язкова **(1 бал)**.
3. Механізм «делегат – подія – обробник події» **(2 бали)**.
4. Перетворити код, який забезпечує роботу з подіями та обробниками подій, на код, що використовує (\*) **(2 бали)**:
   1. анонімні методи;
   2. lambda-вирази;
   3. типи Action та Func (кожен з них).

(\*) - допускається реалізація коду однієї події різними способами, необов’язково різних подій.

1. Механізм створення та оброблення власних помилок **(2 бали)**:
   1. створити новий клас виключної ситуації;
   2. створити новий клас аргументів для передачі їх до обробника виключної ситуації;
   3. забезпечити ініціювання створеної виключної ситуації та продемонструвати, як працює обробник даної помилки;
   4. реалізувати різні сценарії оброблення помилки.
2. Метод-розширення будь-якого класу **(1 бал)**.

### **UML діаграма класів**

|  |
| --- |

**Фрагменти коду для демонстрації виконання задач**

1) Множина інтерфейсів та explicit implementation:

| interface IFootballer  {  void DoAction(string action);  string GetInfo();  void PossibleSubstitute();  int GetCost();  }  interface IDefender  {  void PossibleTackle();  void Defend();  }  interface IAttack  {  void PossibleGoal();  }  interface IGoalkeeper  {  void Defend();  }  interface IForward  {  double GetAccuracy();  } |
| --- |
| void IDefender.Defend()  {  WriteLine("I'm defending, as a defender");  }  void IGoalkeeper.Defend()  {  WriteLine("I'm saving a ball, as a goalkeeper");  } |

2) Абстрактний клас:

| public abstract class Footballer : IDisposable, IFootballer  {  protected string name;  protected int pace;  protected int kicking;  protected int physical;  protected int timeOnField;  private bool disposed = false;  public static readonly int genTime = 90;  public static int goaldifference;  static Footballer()  {  WriteLine("static constructor was called");  goaldifference = 0;  }  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;  }  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 virtual void DoAction(string action)  {  WriteLine("Some info");  }  public virtual string GetInfo()  {  return $"Name: {name}, pace: {pace}, kicking: {kicking}, physical: {physical}";  }  public int Physical  {  get  {  return physical;  }  set  {  if(ArgCollection.GetCheckPh(value) == "Checking of name was successful")  {  this.physical = value;  }  else  {  this.physical = 0;  }  WriteLine(ArgCollection.GetCheckPh(value));  }  }  public string Name  {  get  {  return this.name;  }  set  {  if(ArgCollection.GetCheckName(value) == "Checking of name was successful")  {  this.name = value;  }  else  {  this.name = "Undefined";  }  WriteLine(ArgCollection.GetCheckName(value));  }  }  public void PossibleSubstitute()  {  Random random = new Random();  int num = random.Next(1, genTime);  WriteLine($"it is the {num} minute of the match");  if (num > this.timeOnField && this.timeOnField < genTime)  {  WriteLine("This player is tired, he may be substituted");  }  else  {  WriteLine($"I suppose, he can be on the field at least {timeOnField - num} minutes!");  }  }  public abstract int GetCost();  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;  }  }  ~Footballer()  {  CleanUp(false);  WriteLine("main destructor called");  }  } |
| --- |

3-4) Механізм «делегат – подія – обробник події» відразу по стандартах 4 пункту:

| Defender part |
| --- |
| public void TeamWork()  {  WriteLine("Will defender apploud a goalkeeper for his actions?");  if (GiveFiveEvent != null)  {  GiveFiveEvent((Defender)this, saveOfGK);  }  } |
| Goalkeeper part |
| public Func<Defender, double, bool> possibleSave = (def, prOfSave) =>  {  def.GiveFiveEvent -= new DefConnect(applaud);  def.GiveFiveEvent += new DefConnect(applaud);  if (prOfSave > 50)  {  WriteLine("GK: Save was successful");  return true;  }  else  {  WriteLine("GK: WHAT A MISS!!! It's a goal");  goaldifference -= 1;  return false;  }  };  public static Action<Defender, bool> applaud = (def, save) =>  {  if (save)  {  WriteLine("DF: It was a great save! `Giving five`");  }  else  {  WriteLine("DF: Today is not goalkeeper's day");  }  }; |

5) Механізм створення та оброблення власних помилок:

| public static class MyException  {  public static bool CheckParam(int value)  {  if (value <= 100 && value >= 0)  {  return true;  }  throw new Exception("Wrong param(should be from 0 to 100): ");  }  public static bool CheckName(string name)  {  if (char.IsUpper(name[0]))  {  return true;  }  throw new Exception("Wrong name: first letter isn't upper");  }  }  public static class ArgCollection  {  public static string GetCheckPh(int physical)  {  try  {  if (MyException.CheckParam(physical))  {  return $"Checking of {nameof(physical)} was successful";  }  else  {  return "";  }  }  catch (Exception ex)  {  return ex.Message + nameof(physical);  }  }  public static string GetCheckName(string name)  {  try  {  if (MyException.CheckName(name))  {  return "Checking of name was successful";  }  else  {  return "";  }  }  catch (Exception ex)  {  return ex.Message;  }  }  } |
| --- |

6) Метод-розширення будь-якого класу

| public static class MyExtension  {  public static int GetAverageRating(this Defender def)  {  return (def.Defending + def.Jumping + def.Physical) / 3;  }  } |
| --- |

**Висновки**

Виконавши лабораторну роботу, я зрозумів глибше використання інтерфейсів, їх різницю з абстрактними класами. Також були використані вирази lambda та Action, Func, які до цього я не опановував. Основну увагу було приділено подіям та їх обробникам, оскільки цей матеріал був новий, але труднощів особливих не виникло, завершальним етапом став метод - розширення класу, який було цікаво вивчити. Загалом було вивчено базову логіку роботи з делегатами та абстрактними класами, заглиблення в тему з інтерфейсами.