ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КПІ імені Ігоря Сікорського”

Кафедра програмного забезпечення комп’ютерних систем

**ЛАБОРАТОРНА РОБОТА №1**

з дисципліни

Програмування

C# .Net. Реалізація основних

принципів ООП мовою C#

Студента 1 курсу ФПМ

гр. КП-23 Верьовочкіна М. С.

Київ – 2023

**Завдання на лабораторну роботу №1.**

**«C# .Net. Реалізація основних принципів ООП мовою C#»**

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 задачі

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.
2. Забезпечити наявність у класах властивостей: складніше, ніж просто get;set;, обгрунтувати доцільність створення властивості.
3. Створити для розроблюваних класів такі конструктори:
   * конструктор за замовчанням;
   * конструктор з параметрами;
   * приватний конструктор;
   * статичний конструктор.

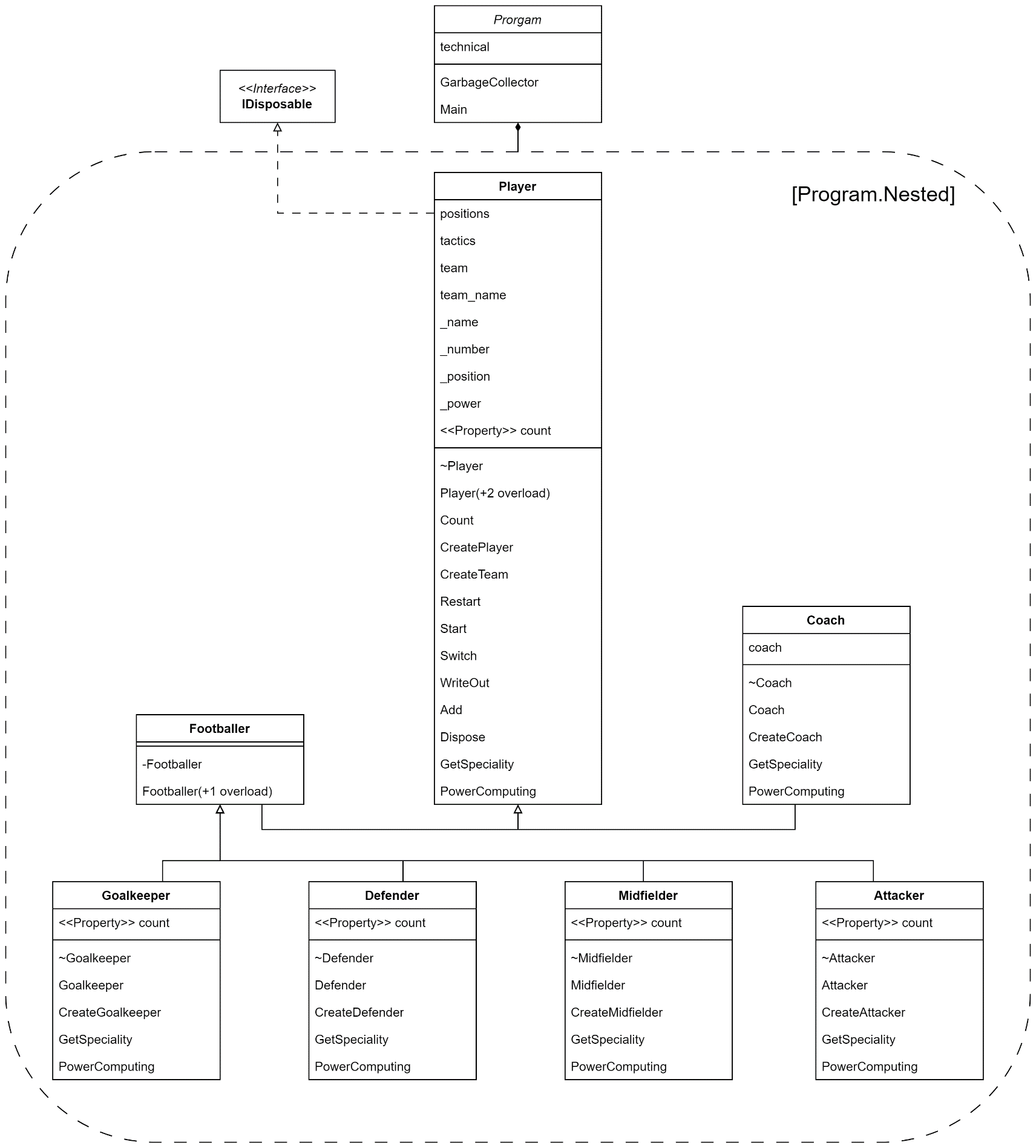
Продемонструвати, яким чином викликаються конструктори базового та дочірнього класів.

1. Використати віртуальні та перевизначені методи.
2. Додати до класів методи, наявність яких дозволятиме управляти знищенням екземплярів цих класів:
   1. реалізувати інтерфейс IDisposable;
   2. створити деструктори;
   3. забезпечити уникнення конфліктів між Dispose та деструктором.
3. 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.

Протокол має містити: титульний аркуш, постановку задачі, UML діаграму класів, фрагменти коду, які демонструють виконання поставлених задач, висновки.

**Всього –** **10 балів**

UML-діаграма розробленої програми:



1. Кілька прикладів полів та методів з різними модифікаторами доступу:
   1. Приватне поле Player.team\_name використовується тільки в методах класу Player:  
      class Player  
      {  
       private static string team\_name;  
       private static void Player.Start(bool coach)  
       {  
       …  
       try { team\_name = Console.ReadLine(); }  
       …  
       }  
      }  
      Console.WriteLine("Team: {0}", team\_name);
   2. Захищене поле Player.\_power використовує як клас Player, так і всі дочірні класи (окрім Footballer):  
      class Player  
      {  
       protected int \_power;  
       protected Player(int power, string position)  
       {  
       …  
       \_power = power;  
       …  
       }  
      }  
      class Coach : Player  
      {  
       protected override int Coach.PowerComputing (string position) => \_power;  
      }  
      class Footballer : Player {}  
      class Defender : Footballer  
      {  
       protected override int Defender.PowerComputing(string position)  
       {  
       …  
       return \_power = 3;  
       …  
       }  
      }
   3. Публічний конструктор Player.Player() використовують зовнішні методи, не пов’язані з класом Player відношенням узагальнення (наслідуванням):  
      class Player  
      {  
       public Player()…  
      }  
      static void GarbageCollector()  
      {  
       …  
       Player p = new Player();  
       …  
      }
2. Класи програми містять кілька властивостей:
   1. Захищена властивість Player.count надає доступ нащадкам класу до приватного поля Player.team.Count. Властивість доступна тільки для читання:  
      class Player  
      {  
       private static List<Player> team = new List<Player>();  
       protected static int count => team.Count;  
      }
   2. Властивість Goalkeeper.count (і подібні) обчислює через функцію батьківського класу шукане значення. Доступна для читання та запису:  
      class Goalkeeper  
      {  
       protected static int count { get { return Count("gk"); } set { } }
3. Для демонстрації роботи конструкторів може бути увімкнений спеціальний параметр technical під час виконання програми:  
   A screenshot of a computer

   Description automatically generatedA computer screen with white text

   Description automatically generated
   1. Конструктор за замовчуванням: public Player()  
       public Footballer()
   2. Конструктор з параметрами: protected Player(int power, string position)  
       protected Footballer(string position) : base(3, position)
   3. Приватний конструктор: private Coach() : base(new Random().Next(0, 2), null)  
       private Goalkeeper() : base("gk")  
       private Defender() : base("df")
   4. Статичний конструктор: static Player()
4. Віртуальні та перевизначені методи:  
   class Player  
   {  
    protected virtual string GetSpeciality()  
    protected virtual int PowerComputing(string position)  
   }  
   class Coach : Player  
   {  
    protected override string GetSpeciality()  
    protected override int PowerComputing(string position) => \_power;  
   }  
   class Footballer : Player  
   class Goalkeeper : Footballer  
   {  
    protected override string GetSpeciality()  
    protected override int PowerComputing(string position)  
   }
5. Класи містять методи для керування знищенням об’єктів.
   1. Батьківський клас Player реалізує інтерфейc IDisposable:  
      class Player : IDisposable  
      {  
       …  
       public void Dispose() { … }  
       …  
      }
   2. Кожен клас, окрім самого Program, має власний деструктор  
      ~Player() { if (technical) Console.WriteLine("\n >> Player destructor\n"); }  
      ~Coach() { if (technical) Console.WriteLine("\n >> Coach destructor\n"); }  
      ~Footballer() { if (technical) Console.WriteLine("\n >> Footballer destructor\n"); }  
      ~Attacker() { if (technical) Console.WriteLine("\n >> Attacker destructor\n"); }

…

* 1. Для уникнення конфліктів між Dispose() та деструктором перший містить метод SuppressFinalize:  
     public void Dispose()  
     {  
      …  
      GC.SuppressFinalize(this);  
     }

1. Метод Program.GarbageCollector() реалізує примусове збирання сміття із застосуванням перелічених методів.