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. Об'єктно-орієнтоване програмування та шаблони проєктування”
тема “C# .Net. Реалізація основних принципів ООП мовою C#”

Виконала
студентка II курсу
групи КП-02
Красношапка Анастасія Андріївна

Перевірила
“____” “_____” 20___ р.
викладач
Заболотня Тетяна Миколаївна

Київ 2021

Мета роботи

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

Постановка задачі

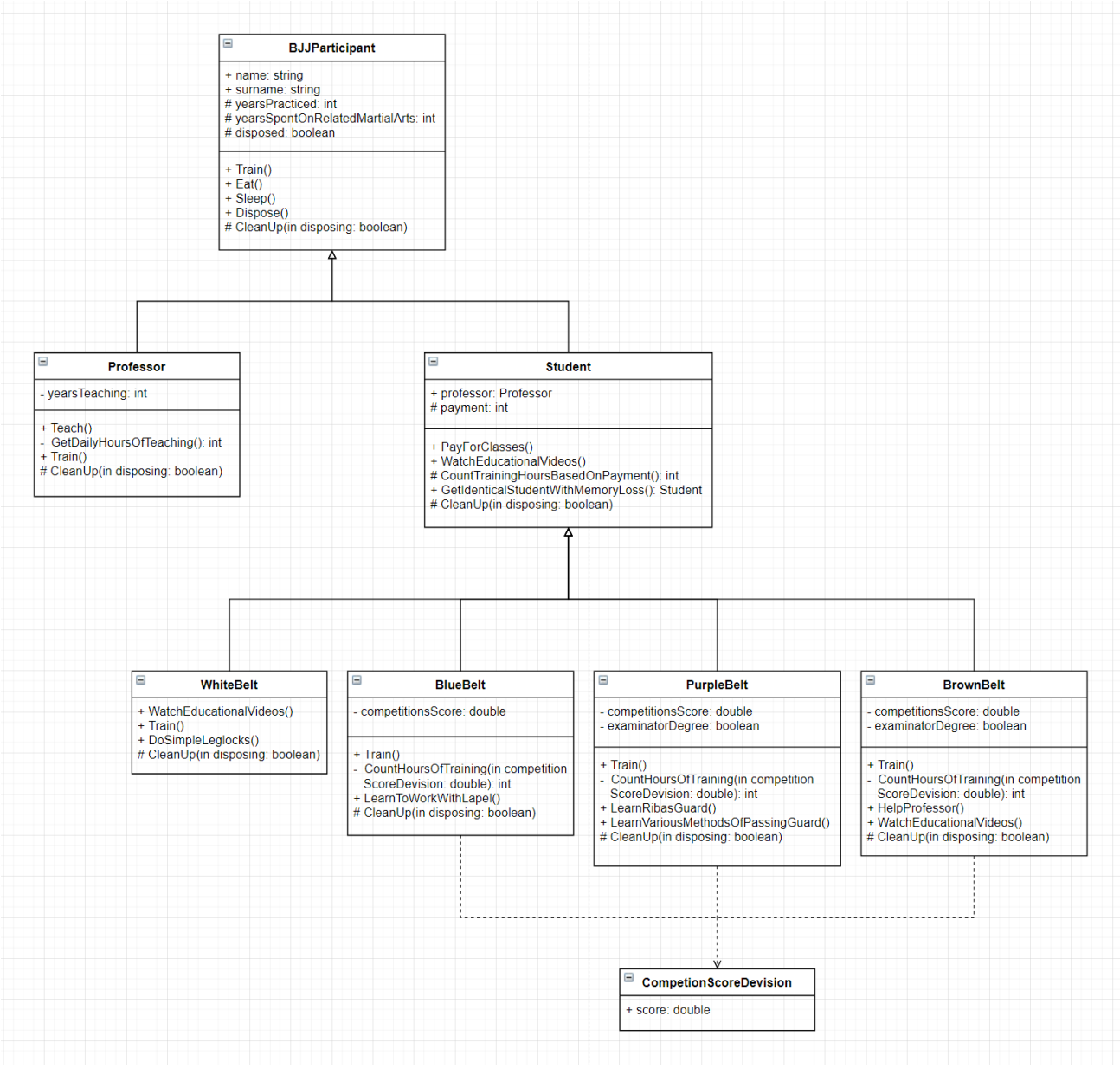
Побудувати ієрархію класів, що відтворюватимуть відношення наслідування між об’єктами реального світу (кількість класів >= 5). При цьому:

1. Забезпечити наявність у класах полів та методів з різними модифікаторами доступу, пояснити свій вибір **(1 бал)**.
2. Забезпечити наявність у класах властивостей: складніше, ніж просто get;set;, обгрунтувати доцільність створення властивості **(1 бал)**.
3. Створити для розроблюваних класів такі конструктори **(2 бали)**:
    - конструктор за замовчанням;
    - 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)**:
    - реалізувати інтерфейс IDisposable;
    - створити деструктори;
    - 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



Виконання поставлених задач

1. Забезпечити наявність у класах полів та методів з різними модифікаторами доступу.

```
14 references
public Professor professor;
17 references
protected int payment;
```

```
12 references
private double competitionsScore;
5 references
private bool examinatorDegree;
```

```
1 reference
public override void Train()
{
    int hours = CountHoursOfTraining(CompetitionScoreDevision.score);
    Console.WriteLine($"Trains for {hours} hours a week.");
}
```

```
1 reference
private int CountHoursOfTraining(double competitionScoreDevision)
{
    int normalHours = this.CountTrainingHoursBasedOnPayment();
    if(this.competitionsScore < competitionScoreDevision)
    {
        return (int)(normalHours*1.2);
    }
    return normalHours;
}
```

```
4 references
protected int CountTrainingHoursBasedOnPayment()
{
    int hours = (int)Math.Floor(this.payment / 100.0);
    return hours;
}
```

2. Забезпечити наявність у класах властивостей: складніше, ніж просто get;set.

```
0 references
public double CompetitionsScore
{
    get
    {
        return this.competitionsScore;
    }
    set
    {
        if((this.competitionsScore != 0 && this.competitionsScore != 1 && value > 0 && value < 1)
            || (this.competitionsScore == 0 && value >= 0 && value < 1)
            || (this.competitionsScore == 1 && value > 0 && value <= 1))
        {
            this.competitionsScore = value;
        }
    }
}
```

```
2 references
public int YearsTeaching
{
    get
    {
        return this.yearsTeaching;
    }
    set
    {
        if(value >= this.yearsPracticed && value >= this.YearsTeaching)
        {
            this.yearsTeaching = value;
        }
    }
}
```

3. Створити для розроблюваних класів такі конструктори:

- конструктор за замовчанням

```
0 references
public Professor()
{
    this.name = "John";
    this.surname = "Dow";
    this.yearsPracticed = 0;
    this.yearsSpentOnRelatedMartialArts = 0;
    this.yearsTeaching = 0;
    Console.WriteLine("[Professor constructor]");
}
```

- конструктор з параметрами

```
0 references
public Professor(string name, string surname, int yearsPracticed, int yearsTeaching, int yearsSpentOnRelatedMartialArts = 0)
{
    this.name = name;
    this.surname = surname;
    this.yearsPracticed = yearsPracticed;
    this.yearsSpentOnRelatedMartialArts = yearsSpentOnRelatedMartialArts;
    if(yearsPracticed < YearsTeaching)
    {
        this.yearsTeaching = yearsPracticed;
    }
    else
    {
        this.yearsTeaching = yearsTeaching;
    }
    Console.WriteLine("[Professor constructor]");
}
```

- приватний конструктор

```
1 reference
private Student(string name, string surname, Professor professor)
{
    this.name = name;
    this.surname = surname;
    this.yearsPracticed = 0;
    this.yearsSpentOnRelatedMartialArts = 0;
    this.professor = professor;
    this.payment = 0;
    Console.WriteLine("[Student constructor]");
}
```

- статичний конструктор

```
static CompetitionScoreDevision()
{
    // If the program was more advanced, constructor would download current statistic information
    // and calculate the score based on this information.

    score = Math.PI / 10;
}
```

Продемонструвати, яким чином викликаються конструктори базового та дочірнього класів.

```
[BJJPractitioner constructor]
[Student constructor]
[WhiteBelt constructor]
[New White Belt object has been created.]
```

4. Використати віртуальні та перевизначені методи.

```
1 reference
public virtual void Train()
{
    Console.WriteLine("Trains for 6 hours a week.");
}

1 reference
public override void Train()
{
    int hours = CountHoursOfTraining(CompetitionScoreDevision.score);
    Console.WriteLine($"Trains for {hours} hours a week.");
}
```

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

- реалізувати інтерфейс IDisposable

```
public class BJJPracticioner : IDisposable

public void Dispose()
{
    this.CleanUp(true);
    Console.WriteLine("[Disposing]");
    Console.WriteLine("[Object is clearnd up.]");
    GC.SuppressFinalize(this);
    Console.WriteLine("[Finalization suppressed]");
}
```

- створити деструктори

```
0 references
~BJJPracticioner()
{
    this.CleanUp(false);
    Console.WriteLine("[Finalization]");
}
```

- забезпечити уникнення конфліктів між Dispose та деструктором

```
protected bool disposed = false;
```

```
2 references
protected virtual void CleanUp(bool disposing)
{
    if(!this.disposed)
    {
        if(disposing)
        {
            // clean managed resourses
            this.name = null;
            this.surname = null;
            this.yearsPracticed = 0;
            this.yearsSpentOnRelatedMartialArts = 0;
        }
        // if there were any unmanaged resourses, they would be cleaned up here
        this.disposed = true;
    }
}
```

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).

Життєвий цикл спеціально створеного об’єкта:

```
[BJJPractitioner constructor]
[Student constructor]
[WhiteBelt constructor]
[New White Belt object has been created.]
Trains for 0 hours a week.
Eats 3 times a day and snacks after trainings.
[White Belt's generation is 0.]
[Disposing]
[Object is clearnd up.]
[Finalization suppressed]
```

Використана пам’ять до створення об’єктів, після їх створення та після вивільнення пам’яті garbage collector-ом:

```
Number of bytes before creating 50 000 objects: 77408.
Number of bytes after creating 50 000 objects: 1285176.
Number of bytes after creating 10 000 objects and activating GC: 75552.
```

Використані методи:

```
int gen = GC.GetGeneration(practitioner);
```

```
GC.Collect();
GC.WaitForPendingFinalizers();

long bytesAfter = GC.GetTotalMemory(true);

GC.ReRegisterForFinalize(practitioner);

GC.SuppressFinalize(this);
```

Створити ситуацію, яка спровокує примусове збирання сміття GC.

```
// For activating garbage collector.
for(int i = 0; i < 50_000; i++)
{
    Object obj = new Object();
}
```

Висновки

Під час лабораторної роботи я краще зрозуміла особливості використання різних модифікаторів доступу, повторила наслідування класів. Навчилися використовувати приватний та статичний конструктори. А також, зрозуміла різницю між Dispose та Finalize, прочитала офіційну документацію по ним та зрозуміла, як їх використовувати.