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. Об'єктно-орієнтоване програмування та шаблони проєктування”
тема “C# .Net. Розширені можливості реалізації ООП у мові C#. Події.”

Виконала
студентка II курсу
групи КП-02
Красношапка Анастасія Андріївна

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

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 задачі

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):
   a) анонімні методи;
   b) lambda-вирази;
   c) типи Action та Func (кожен з них).

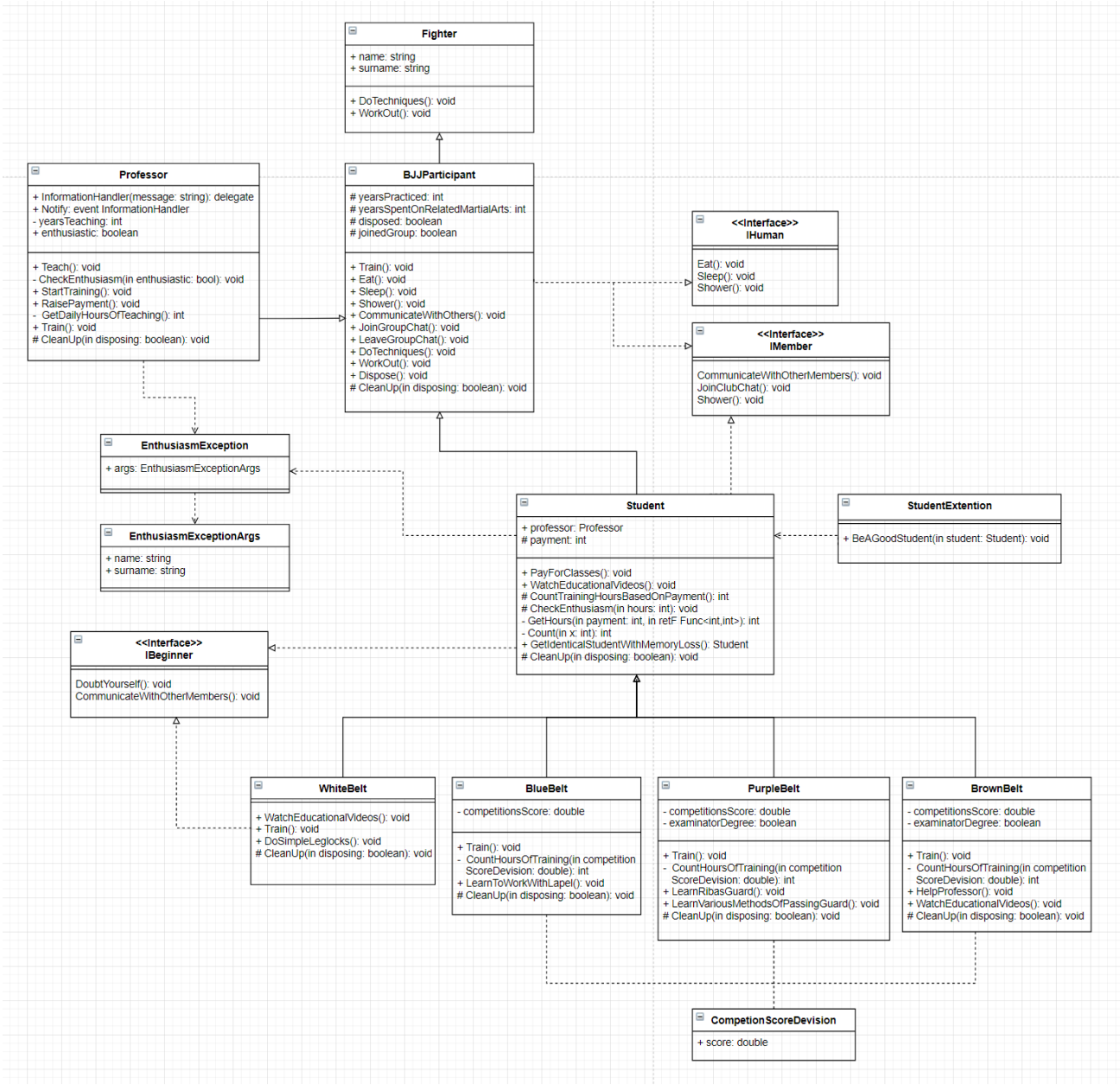
   (*) - допускається реалізація коду однієї події різними способами, необов’язково різних подій.

5. Механізм створення та оброблення власних помилок (2 бали):
   a) створити новий клас виключної ситуації;
   b) створити новий клас аргументів для передачі їх до обробника виключної ситуації;

c) забезпечити ініціювання створеної виключної ситуації та продемонструвати, як працює обробник даної помилки;
d) реалізувати різні сценарії оброблення помилки.

6. Метод-розширення будь-якого класу (1 бал).

UML діаграма класів



Виконання поставлених задач

1. Множину інтерфейсів. При чому один з класів повинен реалізовувати щонайменше 2 інтерфейси.

```
3 references
public interface IBeginner
{
    0 references
    void DoubtYourself();
    0 references
    public void CommunicateWithOtherMembers();
}
```

```
3 references
public interface IMember
{
    0 references
    void CommunicateWithOtherMembers();
    0 references
    void JoinGroupChat();
    0 references
    void LeaveGroupChat();
}
```

```
1 reference
public interface IHuman
{
    0 references
    void Eat();
    0 references
    void Sleep();
    0 references
    void Shower();
}
```

```
2 references
public class BJJPracticioner : Fighter, IDisposable, IMember, IHuman
{
```

Також продемонструвати реалізацію explicit implementation інтерфейса.

```
0 references
void IBeginner.CommunicateWithOtherMembers()
{
    Console.WriteLine("Communicates with other members to learn from their experience.");
}

0 references
void IMember.CommunicateWithOtherMembers()
{
    Console.WriteLine("Communicates with other members for fun.");
}
```

2. Абстрактний клас. Забезпечити його наслідування. Наявність в цьому класі абстрактних методів - обов’язкова.

```
1 reference
public abstract class Fighter
{
    29 references
    public string name;
    29 references
    public string surname;

    0 references
    public abstract void DoTechniques();
    0 references
    public abstract void WorkOut();
}
```

```
2 references
public class BJJPracticioner : Fighter, IDisposable, IMember, IHuman
{
```

3. Механізм «делегат – подія – обробник події».

```
1 reference
public delegate void InformationHandler(string message);
4 references
public event InformationHandler Notify;

1 reference
public void StartTraining()
{
    Console.WriteLine("Starts training.");
    Notify?.Invoke("Training has been started.");
}

1 reference
public void RaisePayment()
{
    Console.WriteLine("Raises payment.");
    Notify?.Invoke("Payment has been raised.");
}
```

```
professor.Notify += delegate(string mes)
{
    Console.WriteLine("\r\n***\r\nNotification!\r\n" + mes + "\r\n***\r\n");
};

List<string> notifications = new List<string>();
professor.Notify += mes => notifications.Add(mes);
```

4. Перетворити код, який забезпечує роботу з подіями та обробниками подій, на код, що використовує (*):

a) анонімні методи;

```
professor.Notify += delegate(string mes)
{
    Console.WriteLine("\r\n***\r\nNotification!\r\n" + mes + "\r\n***\r\n");
};
```

b) lambda-вирази;

```
professor.Notify += mes => notifications.Add(mes);
```

c) типи Action та Func

```
Action<Student, Professor> action = ChangeProf;
ChangeRelationship(student2, professor, action);

protected int CountTrainingHoursBasedOnPayment()
{
    Func<int, int> countFunc = Count;
    int hours = this.GetHours(this.payment, countFunc);
    return hours;
}
```

5. Механізм створення та оброблення власних помилок:

a) створити новий клас виключної ситуації;

```
[System.Serializable]
7 references
public class EnthusiasmException : System.Exception
{
    11 references
    public EnthusiasmExceptionArgs args;

    0 references
    public EnthusiasmException() { }
    0 references
    public EnthusiasmException(string message) : base(message) { }
    2 references
    public EnthusiasmException(string message, EnthusiasmExceptionArgs args) : base(message)
    {
        this.args = args;
    }
    0 references
    public EnthusiasmException(string message, System.Exception inner) : base(message, inner) { }
    0 references
    protected EnthusiasmException(
        System.Runtime.Serialization.SerializationInfo info,
        System.Runtime.Serialization.StreamingContext context) : base(info, context) { }
}
```

b) створити новий клас аргументів для передачі їх до обробника виключної ситуації;

```
4 references
public class EnthusiasmExceptionArgs
{
    6 references
    public string name;
    6 references
    public string surname;

    2 references
    public EnthusiasmExceptionArgs(string name, string surname)
    {
        this.name = name;
        this.surname = surname;
    }
}
```

c) забезпечити ініціювання створеної виключної ситуації та продемонструвати, як працює обробник даної помилки;

```
Exception caught: lab2.EnthusiasmException: Cannot teach classes without love for the martial art.
   at lab2.Professor.CheckEnthusiasm(Boolean enthusiastic) in D:\KPI\SEMESTER 3\Progbase\lab2\Profes
sor.cs:line 81
   at lab2.Professor.Teach() in D:\KPI\SEMESTER 3\Progbase\lab2\Professor.cs:line 66               sor.cs:line 81
Professor John Dow is sent om motivational weekend.

Trains for 0 hours a week.
Exception caught: lab2.EnthusiasmException: Lacks enthusiasm, visits no trainings.
   at lab2.Student.CheckEnthusiasm(Int32 hoursTraining) in D:\KPI\SEMESTER 3\Progbase\lab2\Student.c
s:line 86                                                                                          s:line 86
   at lab2.BlueBelt.Train() in D:\KPI\SEMESTER 3\Progbase\lab2\BlueBelt.cs:line 65
Student John Dow is sent om motivational weekend.
```

d) реалізувати різні сценарії оброблення помилки.

```
1 reference
public void Teach()
{
    try
    {
        CheckEnthusiasm(this.enthusiasticAboutBJJ);
        int hours = this.GetDailyHoursOfTeaching();
        Console.WriteLine($"Teaches {hours} hours a day.");
    }
    catch(EnthusiasmException e)
    {
        Console.WriteLine("Exception caught: {0}", e);
        Console.WriteLine($"Professor {e.args.name} {e.args.surname} is sent om motivational weekend.");
    }
}
```

```
2 references
public override void Train()
{
    int hours = this.CountTrainingHoursBasedOnPayment();
    Console.WriteLine($"Trains for {hours} hours a week.");
    try
    {
        CheckEnthusiasm(hours);
    }
    catch(EnthusiasmException e)
    {
        Console.WriteLine("Exception caught: {0}", e);
        Console.WriteLine($"Student {e.args.name} {e.args.surname} is sent om motivational weekend.");
    }
}
```

6. Метод-розширення будь-якого класу.

```
1 reference
public static class StudentExtension
{
    1 reference
    public static void BeAGoodStudent(Student student)
    {
        System.Console.WriteLine($"Student {student.name} {student.surname} is a good student.");
    }
}
```

Висновки

Під час лабораторної роботи я зрозуміла особливості використання делегатів, абстрактних класів. Навчилися використовувати події, анонімні методи, lambda-вирази, Func та Action. Навчилася створювати власні типи Exception та повторила оброблення помилок. А також навчилася створювати методи-розширення для недоступних класів.