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”

тема "C# .Net. Розширені можливості реалізації ООП у мові C#. Події."

Виконав
студент 2 курсу
групи КП-01
Беліцький Олександр Сергійович
(*прізвище, ім’я, по батькові*)

Перевірив
“____” “_____” 20___ р.
викладач
Заболотня Тетяна Миколаївна
(*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.

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

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

4. Перетворити код, який забезпечує роботу з подіями та обробниками подій, на код, що використовує (*):
    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:
    - створити новий клас виключної ситуації;
    - створити новий клас аргументів для передачі їх до обробника виключної ситуації;
    - забезпечити ініціювання створеної виключної ситуації та продемонструвати, як працює обробник даної помилки;
    - реалізувати різні сценарії оброблення помилки.

6. Метод-розширення будь-якого класу.

Аналіз вимог і проектування

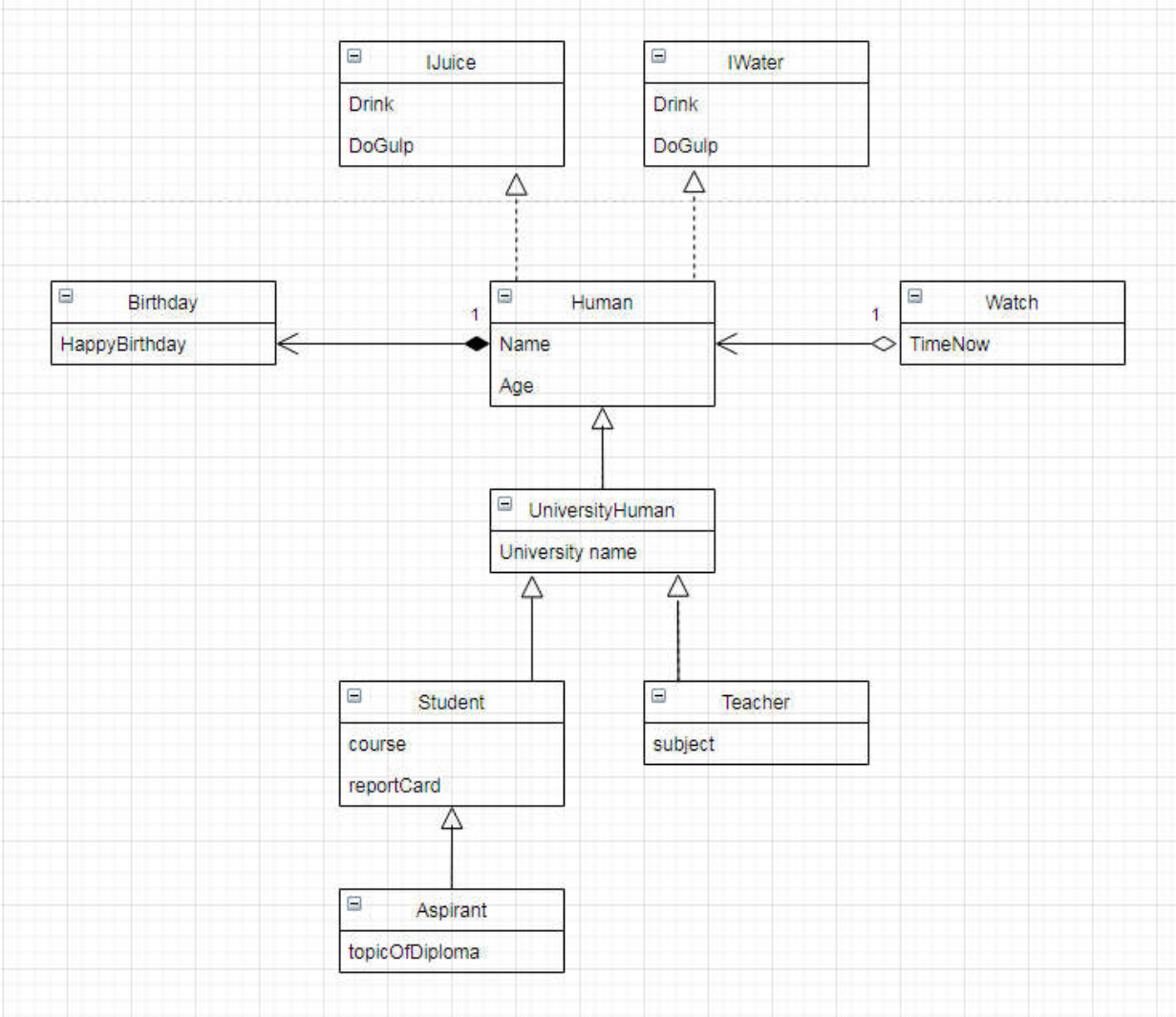

рис.1 UML-діаграма класів

Зв'язки:

День народження - Людина - композиція, бо День народження без людини існувати не може.

День народження - "Годинник" - агрегація, бо час існує окремо від людини і одночасно з нею.

Людина - Сік/Вода - імплементація, Людина реалізує інтерфейси Соку і Води.

Людина - Людина з університету - наслідування, бо людей багато і піпгрупа людей, що причетні до університету.

Людина з університету - Студент - Аспірант - наслідування(по аналогії з попереднім).

Людина з університету - Вчитель - наслідування(по аналогії з попереднім).

Приклади результатів

1. *Створити множину інтерфейсів. При чому один з класів повинен реалізовувати щонайменше 2 інтерфейси. Також продемонструвати реалізацію explicit implementation інтерфейса, обгрунтувати її використання.*

У даній роботі були створені два інтерфейси Соку і Води. Людина може пити ці напої по-різному, тому були використані саме інтерфейси.

```
public interface IJuice
{
    void Drink();
    void DoGulp();
}
public interface IWater
{
    void Drink();
    void DoGulp();
}
```

Реалізація інтерфейсів у класі Human.

```
public void Drink()
    {
        Console.WriteLine("I drank something");
    }

    void IJuice.DoGulp()
    {
        if (highSugar)
        {
            Sip?.Invoke($"{this.name} has high level of sugar. Please, take water");
        }
        else
        {
            Sip?.Invoke($"{this.name} took a sip of juice");
        }
    }

    void IWater.DoGulp()
    {
        Sip?.Invoke($"{this.name} took a sip of water");
    }
```

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

В даній лабораторній було змінено статус класу UniversityPerson - він став абстрактним та метод *GoToClass()* став абстрактним. Тому метод був переписаний у класах, які налідують його: *Student* та *Teacher.*

Клас UniversityPerson:

```
using System;

public abstract class UniversityPerson : Human
{
    protected string UniversityName;

    public UniversityPerson(string name, int age, string UniversityName) : base(name,
age)
    {
        this.UniversityName = UniversityName;
    }

    public UniversityPerson(string name, int age, bool highSugar) : base(name, age,
highSugar)
    {
        this.UniversityName = "unknown";
    }

    public abstract void GoToClass();
}
```

Переписаний метод *GoToClass()* для:

- *Student*

```
public override void GoToClass()
    {
        WriteLine("I'm going to classroom");
        WriteLine("I'm studying here");
    }
```

- *Teacher*

```
public override void GoToClass()
    {
        Console.WriteLine("I'm going to classroom");
        Console.WriteLine("I'm teaching here");
    }
```

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

Для забезпечення даного механізму був створений делегат ToGulp(зробити ковток) та відповідна подія. Вони розташовуються в класі *Human*. Для здійснення складнішого механізму також додано рівень цукру в крові, що показує, чи може людина пити Сік.

```
public delegate void ToGulp(string drink);
public event ToGulp Sip;

protected bool highSugar;
```

Безпосередня реалізація події знаходиться в методах, що імплементують інтерфейси Соку і Води(див. пункт1).

Частина коду із основної програми, щоб показати обробник події та підписання події на обробник.

```
static void Main(string[] args)
    {
     Human gerd = new Human("Gerd", 44, true);
     IWater iW = gerd;
     IJuice iJ = gerd;

     gerd.Sip += DisplayMessage;

     iW.DoGulp();
     iJ.DoGulp();
          }

private static void DisplayMessage(string message)
    {
        Console.WriteLine(message);
    }
```

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

   a. *анонімні методи;*

```
Student nick = new Student("Nick", 22, false);

            nick.Sip += delegate (string message)
            {
                Console.WriteLine(message);
            };
            iW = nick;
            iJ = nick;

            iW.DoGulp();
            iJ.DoGulp();
```

b. lambda-вирази;

```
Student bob = new Student("Bob", 33, true);

            bob.Sip += message => Console.WriteLine(message);
            iW = bob;
            iJ = bob;

            iW.DoGulp();
            iJ.DoGulp();
```

c. типи Action та Func (кожен з них).

```
Student bob = new Student("Bob", 33, true);

            Action<int, int> DoMath;
            DoMath = bob.Add;
            Operation(36, 22, DoMath);
            DoMath = bob.Substract;
            Operation(34, 22, DoMath);

            Console.WriteLine();

            Func<int, int, int> Multiply = bob.Multiply;
            int multiResult = GetMultiply(4, 5, Multiply);
```

У класі *Student* було створено методи додавання та віднімання, що забезпечують роботу делегата *Action* і метод множення, що забезпечує роботу делегата *Func*.

```
public void Add(int x1, int x2)
    {
        Console.WriteLine("Summation result: " + (x1 + x2));
    }

    public void Substract(int x1, int x2)
    {
        Console.WriteLine("Subtraction result: " + (x1 - x2));
    }

    public int Multiply(int x1, int x2)
    {
        return x1 * x2;
    }
```

5. *Механізм створення та оброблення власних помилок:*

   - *створити новий клас виключної ситуації;*
   - *створити новий клас аргументів для передачі їх до обробника виключної ситуації;*
   - *забезпечити ініціювання створеної виключної ситуації та продемонструвати, як працює обробник даної помилки;*
   - *реалізувати різні сценарії оброблення помилки.Додати до класів методи, наявність яких дозволятиме управляти знищенням екземплярів цих класів:*

Все зазначене забезпечено у наступному фрагменті коду:

```
class HumanException : Exception
{
    public Human args;
    public HumanException(Human args) : base()
    {
        this.args = args;
    }

    public override string Message => $"Error: {args.Name} has high level of sugar.
{args.Name} can't drink juice.";
}
```

6. *Метод-розширення будь-якого класу.*

   Зазначене забезпечено у наступному фрагменті коду:

```
using System;

public static class TeacherExtension
{
    public static void GivePresent(this Teacher teacher, string present)
    {
        teacher.Present = present;
    }
}
```

Висновки

Виконавши дану лабораторну роботу було підкоректовано знання з основ об'єктно-орієнтовного програмування. Були набути навички з роботи із абстрактними класами.

Також ознайомився з explicit implementation інтерфейсів. якщо клас реалізує два інтерфейси, які містять член з однаковою сигнатурою, то реалізація цього члена в класі змусить обидва інтерфейси використовувати цей член як свою реалізацію. Щоб викликати іншу реалізацію залежно від того, який інтерфейс використовується, потрібно явно реалізувати член інтерфейсу. Явна реалізація інтерфейсу — це член класу, який викликається лише через вказаний інтерфейс.

Ознайомився з делегатами, подіями і їх супроводом: анонімними методами, лямбда-виразами та окремими делегатами Action та Func. Делегати репрезентують такі об'єкти, які вказують на методи. Тобто делегати – це покажчики на методи і за допомогою делегатів ми можемо викликати ці методи. Події сигналізують системі про те, що сталася певна дія. І якщо нам треба відстежити ці дії, то ми можемо застосовувати події.

Компіляція всього коду відбувалася за допомогою утиліти dotnet.