ЛАБОРАТОРНАЯ РАБОТА 13. РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ

1. Цель и содержание

Цель лабораторной работы: изучить принципы работы с типами интерфейсов в C#.

Задачи лабораторной работы:

– научиться объявлять интерфейсы в C#;

– научиться создавать классы, реализующие интерфейсы;

2. Формируемые компетенции

Лабораторная работа направлена на формирование следующих компетенций:

– способность к проектированию базовых и прикладных информационных технологий (ПК-11);

– способность разрабатывать средства реализации информационных технологий (методические, информационные, математические, алгоритмические, технические и программные) (ПК-12).

3. Теоретическая часть

2.1 Наследование интерфейсов. Кроме наследования реализации (implementation inheritance) в языке C# реализован механизм наследования интерфейсов (interface inheritance). Необходимо понимать, что не все объектно-ориентированные языки поддерживают интерфейсы.

Если класс наследует интерфейс, класс как бы берет на себя обязательства реализовать некоторый функционал.

Синтаксис наследования интерфейса не отличается от синтаксиса наследования реализации:

class *КлассРеализующийИнтерфейс* : *Интерфейс*
{
    // *Данные-члены и функции-члены*
}

Отличие от механизма наследования реализации состоит в том, что:

- класс обязательно должен реализовать методы и свойства, продекларированные в интерфейсе (никакой реализации в интерфейсе не существует);

- класс может реализовывать (наследовать) несколько интерфейсов, тогда как базовый класс может быть только один.

2.2 Объявление интерфейсов. Для объявления типа интерфейса используется ключевое слово interface:

```
public interface ICalculate
{
    void Plus(int pPlus);
    void Minus(int pMinus);
}

public interface IVisual
{
    string Name { get; set; }
    void DrawObject();
}
```

В приведенном примере объявлены два интерфейса: ICalculate и IVisual.

Интерфейс ICalculate включает два метода с одним параметром. Из названий видно, что один метод что-то увеличивает, второй – что-то уменьшает на величину параметра.

Второй интерфейс IVisual содержит метод DrawObject без параметров, который призван нарисовать объект, а также свойство Name, которое доступно как для чтения, так и для записи.

Следует понимать, что интерфейсы не предполагают конкретную реализацию методов, а свойства интерфейсов не хранят данных. Интерфейс

должен быть реализован классом, который и определит конкретное наполнение методов и свойств интерфейса.

Для использования объявленных интерфейсов создадим два класса Human (человек) и Car (автомобиль).

```
public class Human
{
    public Human(string pFIO, int pAge)
    {
        FIO = pFIO;
        Age = pAge;
    }

    private string FIO;
    private int Age;
}

public class Car
{
    public Car(string pManufacturer, string pModel, int pVelocity)
    {
        Manufacturer = pManufacturer;
        Model = pModel;
        Velocity = pVelocity;
    }

    private string Manufacturer;
    private string Model;
    private int Velocity;
}
```

В каждом классе определен конструктор, который инициализирует закрытые данные-члены класса.

Требуется для каждого класса реализовать вывод в консоль, а также возможности увеличения скорости автомобиля и возможности изменения возраста у человека.

Этот общий функционал можно реализовать с использованием следующих способов:

1. Добавить все необходимые функции в каждый класс независимо. Этот метод приводит к «разбуханию кода», сложной управляемости кода.

2. Реализовать один класс, например Base (базовый), в котором определить общие методы и свойства (Plus, Minus, DrawObject). Затем использовать класс Base в качестве базового для классов Human и Car, причем в каждом классе переопределить методы базового класса с использованием

ключевого слова override. Этот метод – наследование реализации. Получается иерархия классов, которые по смыслу и наполнению сильно отличаются. Хотя в данном подходе классы и код более упорядочены, все равно возникает избыточность за счет многократного переопределения методов.

3. Определение интерфейсов и реализация возможностей интерфейсов данными классами.

Именно метод 3 будет использован в данной лабораторной работе.

4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

1. Создайте консольное приложение в соответствии с алгоритмом, представленным в лабораторной работе №1.

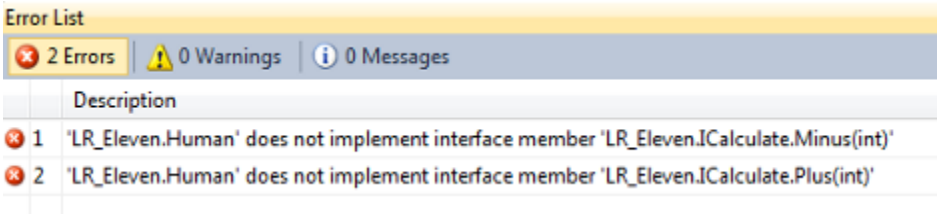
2. Определите в приложении классы Human и Car, а также интерфейсы ICalculate и IVisual, представленные в разделе «Теоретическое обоснование» данной лабораторной работы.

3. Реализуем механизм наследования интерфейса ICalculate классом Human. Для этого выполним следующие действия:

3.1. В определении класса укажем, что класс Human наследует интерфейс ICalculate.

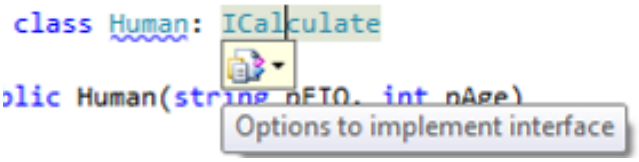
```
public class Human: ICalculate
```

3.2. Обратите внимание, что после этого попытка перекомпилировать проект приведет к ошибкам:

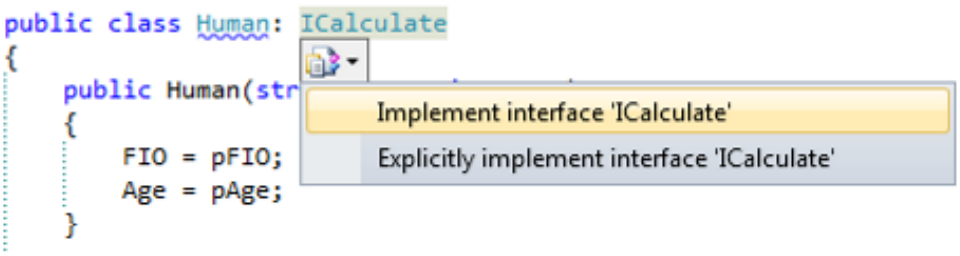


То есть компилятор сообщает, что интерфейс наследуется классом, но методы, заявленные в интерфейсе, классом не реализованы. Реализуем их.

3.3. Наведите курсор мыши на интерактивное подчеркивание и появится выпадающий список:



Раскройте его и выберите команду «Implement interface» (реализовать интерфейс).



3.4. В классе Human появятся два новых метода, как и было объявлено в интерфейсе ICalculate. Определение класса примет вид:

```
public class Human: ICalculate
{
    public Human(string pFIO, int pAge)
    {
        FIO = pFIO;
        Age = pAge;
    }

    private string FIO;
    private int Age;

    public void Plus(int pPlus)
    {
        throw new NotImplementedException();
    }

    public void Minus(int pMinus)
    {
        throw new NotImplementedException();
    }
}
```

Обратите внимание, что в каждый сгенерированный метод среда разработки добавила код вызова исключения, то есть проект скомпилируется без ошибок но в процессе выполнения, при попытке использовать методы Plus или Minus программа завершится с ошибками. Это сделано для того, чтобы программист не забыл реализовать данные методы интерфейсов.

В процессе выполнения пп. 3.1 – 3.3 были использованы возможности Visual Studio по автоматизации реализации интерфейса. Очевидно, что интерфейс можно было реализовать, самостоятельно написав данный код.

– использование вызова базового конструктора;

– использование вызова любого базового метода (отличного от конструктора).

4. Определим окончательную реализацию для методов Plus и Minus:

```
public void Plus(int pPlus)
{
    Age += pPlus;
}

public void Minus(int pMinus)
{
    Age -= pMinus;
}
```

5. Аналогичным образом реализуем наследование интерфейса IVisual для класса Human. Окончательно для класса Human получим:

```
public class Human: ICalculate, IVisual
{
    public Human(string pFIO, int pAge)
    {
        FIO = pFIO;
        Age = pAge;
    }

    private string FIO;
    private int Age;

    public void Plus(int pPlus)
    {
        Age += pPlus;
    }

    public void Minus(int pMinus)
    {
        Age -= pMinus;
    }

    public string Name
    {
        get
        {
            return FIO + " : " +Age.ToString();
        }
        set
        {
            FIO = value;
        }
    }

    public void DrawObject()
    {
        Console.WriteLine
            (
                "    o    \n" +
                "  -----  \n" +
                "    |    \n" +
                "   / \\   \n" +
                "  /   \\  \n"
            );
        Console.WriteLine(Name);
    }
}
```

6. Реализуем интерфейсы ICalculate и IVisual для класса Car:

```
public class Car: IVisual, ICalculate
{
    public Car(string pManufacturer, string pModel, int pVelocity)
    {
        Manufacturer = pManufacturer;
        Model = pModel;
        Velocity = pVelocity;
    }

    private string Manufacturer;
    private string Model;
    private int Velocity;

    public void Plus(int pPlus)
    {
        Velocity += pPlus;
    }

    public void Minus(int pMinus)
    {
        Velocity += pMinus;
    }

    public string Name
    {
        get
        {
            return Manufacturer + " - " + Model +
                " : " + Velocity.ToString() + "km/h";
        }
        set
        {
            Model = value;
        }
    }

    public void DrawObject()
    {
        Console.WriteLine
            (
                "     -----------\n" +
                "____/           \\_____\n" +
                "|                      |\n" +
                "---(@)------------(@)--- \n"
            );
        Console.WriteLine(Name);
    }
}
```

7. Когда все классы и интерфейсы определены и реализованы можно их использовать. Продемонстрируем использование типов данных созданием соответствующих объектов в функции main.

```
static void Main(string[] args)
{
    Console.BackgroundColor = ConsoleColor.White;
    Console.ForegroundColor = ConsoleColor.Blue;
    Console.Clear();

    Console.Title = "Лабораторная работа №11";

    Human h = new Human("Иванов Иван Иванович", 50);
    h.Plus(5);
    h.Minus(1);
    h.DrawObject();

    Console.WriteLine("\n\n\n");

    Car car = new Car("Hyundai", "ix35", 120);
    car.Plus(25);
    car.Minus(11);
    car.DrawObject();

    Console.ReadKey();
}
```

8. Вид окна разработанного приложения представлен на рис. 18.1:

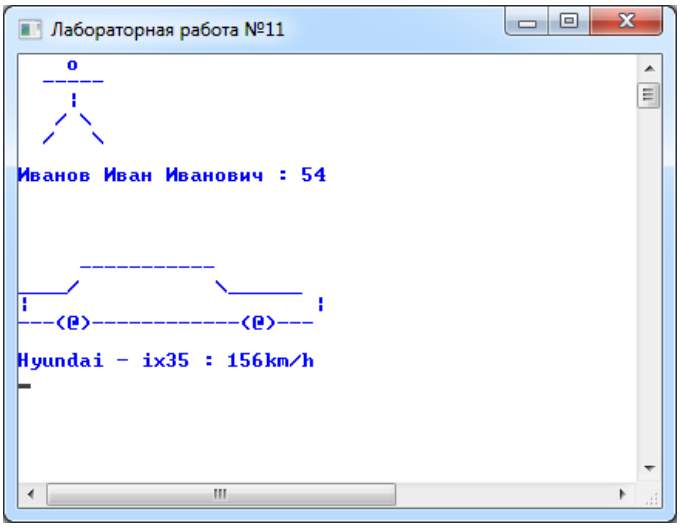


Рисунок 18.1 – Консольное приложение на основе интерфейсных типов.

Диаграмма типов данных, созданная редактором Visual Studio, для разработанного приложения, показана на рис. 18.2 (сравните ее с диаграммой на рис. 17.2 в лабораторной работе №17):

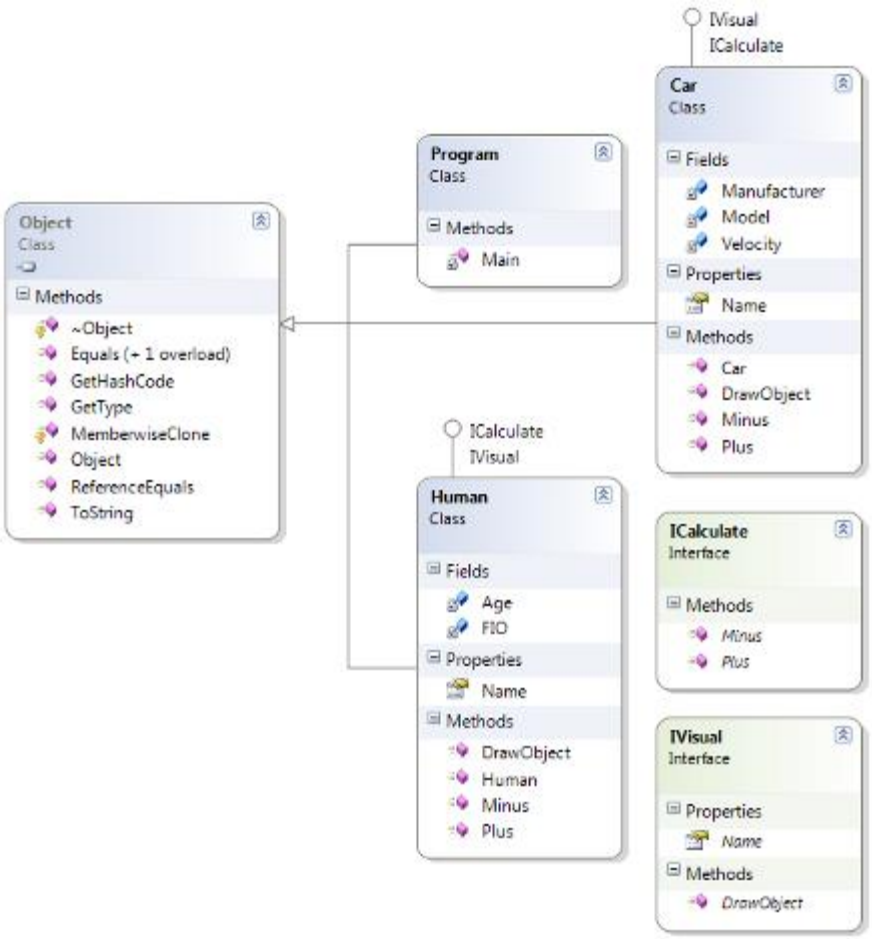


Рисунок 18.2 – Диаграмма классов приложения.

Индивидуальное задание.

Спроектируйте классы, наполните их требуемой функциональностью, определите интерфейс, продемонстрируйте использование объектов класса и методов интерфейса. Постройте диаграмму классов своего приложения средствами Visual Studio.

В качестве классов можно использовать иерархию классов, разработанную в лабораторной работе №17. Интерфейс спроектируйте и реализуйте самостоятельно.

7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Что такое наследование реализации? Как описать синтаксически наследование реализации?

2. Для чего используется ключевое слово base?

3. Можно ли переопределить метод класса? Свойства класса? Данные класса?

4. Что такое наследование интерфейса? Укажите основные отличия от наследования реализации.

5. Для чего используется ключевое слово virtual? Для чего используется ключевое слово override?

6. Может ли один класс наследовать несколько классов? Несколько интерфейсов?

7. Внимательно изучите код примеров данной лабораторной работы и подумайте, каким образом специфицируются методы интерфейса public, private или protected? В чем причина применения именно такого модификатора доступа?

9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [6].

ЛАБОРАТОРНАЯ РАБОТА 14. ФАЙЛОВЫЙ ВВОД-ВЫВОД. РАБОТА С КАТАЛОГАМИ. РАБОТА С ФАЙЛАМИ.

1. Цель и содержание

Цель лабораторной работы: научиться использовать механизмы файлового ввода-вывода.

Задачи лабораторной работы:

– научиться применять классы для работы с файлами;

– научиться применять классы для работы с каталогами;

– научиться использовать потоки ввода-вывода.

2. Формируемые компетенции

Лабораторная работа направлена на формирование следующих компетенций:

– способность к проектированию базовых и прикладных информационных технологий (ПК-11);

– способность разрабатывать средства реализации информационных технологий (методические, информационные, математические, алгоритмические, технические и программные) (ПК-12).

3. Теоретическая часть

3.1 Классы .NET Framework для реализации операций ввода-вывода.

Весь ввод и вывод в .NET Framework подразумевает использование потоков. Поток – это абстрактное представление последовательного устройства. Последовательное устройство – это нечто такое, что хранит данные в линейной структуре и точно таким же образом обеспечивает доступ к ним: считывает или записывает по одному байту за одну единицу времени.