Лабораторная работа №2

Работа с классами C#

1) Работа с готовыми классами

Язык C# является объектно-ориентированным языком программирования (ООП). В основе концепции ООП лежит понятие *класса* – логической структуры, позволяющей создавать свои собственные пользовательские типы путем группирования переменных других типов, методов и событий. *Экземпляр класса* (объект) – это переменная, реализующая класс, то есть класс описывает свойства и методы, которые будут доступны у объекта, построенного по описанию, заложенному в классе. Существуют также исключения в виде *статических классов* - в памяти имеется только одна копия объекта, и клиентский код может получить к ней доступ только посредством самого класса, а не переменной экземпляра.

Объекты можно создавать с помощью ключевого слова *new*, за которым следует имя класса, на котором будет основан объект:

```
Customer object1 = new Customer();
```

Здесь *Customer* – имя класса, *object*1 – экземпляр класса *Customer*. Отметим, что после ключевого слова *new* имеется запись *Customer*(), которая означает вызов конструктора класса *Customer* без параметров. Также отметим, что *object*1 является ссылкой на созданный объект, но не содержит его данные. Фактически, можно создать ссылку на объект без создания самого объекта:

```
Customer object2;
```

Создание таких ссылок, которые не указывают на объект, не рекомендуется (за исключением ссылок на статические классы), так как попытка доступа к объекту по такой ссылке приведет к сбою во время выполнения. Однако такую ссылку можно сделать указывающей на объект, создав новый объект или назначив ее существующему объекту:

```
Customer object3 = new Customer();
```

```
Customer object4 = object3;
```

В данном коде создаются две ссылки на объекты, которые указывают на один объект. Поэтому любые изменения объекта, выполненные посредством *object*3, будут видны при последующем использовании *object*4. Поскольку на объекты, основанные на классах, указывают ссылки, классы называют ссылочными типами.

2) Создание пользовательских классов. Члены класса. Модификаторы доступа.

Структуру классов отражают их *члены*, представляющие их данные и поведение:

1) *Поля* являются переменными, объявленными в области класса. Поле может иметь встроенный числовой тип или быть экземпляром другого класса. Обычно поля являются закрытыми для доступа и хранят данные, которые используются несколькими *методами к*ласса.

2) *Методы* определяют действия, которые может выполнить класс. Методы могут получать параметры, предоставляющие входные данные, и возвращать выходные данные посредством параметров. Также методы могут возвращать значения напрямую, без использования параметров.

3) *Свойства* – способ доступа к внутреннему состоянию объекта, имитирующий переменную некоторого типа. Доступ к ним осуществляется так же, как если бы они были полями этого класса. Свойство может защитить поле класса от изменений (независимо от объекта).

4) *Константы* — это поля или свойства, значения которых устанавливаются во время компиляции и не изменяются.

5) *События* предоставляют другим объектам уведомления о различных случаях, таких как нажатие кнопки или успешное выполнение метода.

6) *Перегруженные операторы* рассматриваются как члены класса. При перегрузке оператора его следует определить как открытый статический метод

в классе.

7) *Конструкторы* – это методы классов, вызываемые при создании объекта заданного типа. Зачастую они используются для инициализации данных объекта.

8) *Деструкторы* очень редко используются в C#. Они являются методами, вызываемыми средой выполнения, когда объект нужно удалить из памяти. Деструкторы обычно применяются для правильной обработки ресурсов, которые должны быть высвобождены.

Модификаторы доступа – это ключевые слова, задающие доступность члена или типа. Выделим 3 наиболее используемых модификатора:

1) **public** : Неограниченный доступ.

2) **protected** : Доступ ограничен содержащим классом или типами, которые являются производными от содержащего класса.

3) **private** : Доступ ограничен содержащим типом.

Рассмотрим создание пользовательского класса «Календарь», чтобы пояснить введенные выше определения:

```
public class CalendarEntry
{
    private DateTime date; //поле date типа DateTime, использование ограничено
    public string day; //открытое поле day типа String (строка). Доступ неограничен

    public DateTime Date // открытое свойство типа DateTime
    {
        get // реализует механизм получения значения свойства
        {
            return this.date; // вернуть значение закрытого поля date
        }
        set // реализует механизм установки значения свойства
        {
            // Здесь можно проверить диапазон допустимых значений
            if (value.Year > 1900 && value.Year <= DateTime.Today.Year)
            {
                this.date = value;
            }
        }
```

```
    }
    // Открытый метод для установки даты
    // Формат вызова: instance.SetDate("1975, 6, 30");
    public void SetDate(string dateString)
    {
        DateTime dt = Convert.ToDateTime(dateString);

        // Здесь можно определить диапазон допустимых значений
        if (dt.Year > 1900 && dt.Year <= DateTime.Today.Year)
        {
            date = dt;
        }
    }
    // Открытый метод для получения временного промежутка
    public TimeSpan GetTimeSpan(string dateString)
    {
        DateTime dt = Convert.ToDateTime(dateString);

        if (dt != null && dt.Ticks < date.Ticks)
        {
            return date - dt;
        }
    }
}
```

В кассе объявлены два поля – *date* и *day*, первое закрыто для доступа извне класса (значение поля нельзя изменить из любого другого класса, правильное использование поля), второе – открыто (неправильное использование поля). Класс содержит два метода – *SetDate*, принимающий в качестве аргумента текстовую строку, содержащую дату в формате “гггг, мм, дд” и не имеющий возвращаемого значения, второй метод используется для получения временного промежутка относительно аргумента текстового типа *dateString*. Обратите внимание, что поля обычно используются для хранения состояния класса, поэтому использование модификаторов *public* с полями не рекомендуется, так как при установке значений таких полей не производится проверка и может возникнуть исключительные ситуации.

Язык C# имеет мощные средства для обеспечения доступа к закрытым полям с помощью свойств. Например, в классе *CalendarEntry* свойство *Date*

позволяет установить или получить значение даты. Обратите внимание на ключевые слова *get*, *set*, *value* и *this*. Метод доступа свойства *get* используется для возврата значения свойства, а метод доступа *set* используется для назначения нового значения. С помощью ключевого слова *value* можно получить доступ к новому значению свойства. Ключевое слово *this* означает, что доступ производится к члену текущего класса. Отметим, что использование ключевого слова *this* обязательно, только если имя поля класса совпадает с именем аргумента метода.

Пример использования класса:

```
CalendarEntry ce = new CalendarEntry(); //Создание экземпляра класса
ce.day = "Вторник"; // Установка значения открытого поля
ce.Date = Convert.ToDateTime("1987, 7, 5"); // Установка даты через свойство
TimeSpan time = ce.GetTimeSpan("2013,9,17"); //Получение временного промежутка
```

В приведенном выше примере используется конструктор по умолчанию. Программист может создавать конструкторы, имеющие входные параметры. Для этого в классе нужно реализовать метод без возвращаемого значения и с подходящими аргументами, например:

```
public class CalendarEntry
{
    public string day;
    public CalendarEntry(DateTime date)
    {
        this.date = date;
    }
}
```

Здесь также показан пример использования ключевого слова *this*. Создание экземпляра класса в этом случае может выглядеть следующим образом:

```
CalendarEntry ce = new CalendarEntry(Convert.ToDateTime("1987, 7, 5"));
```

Константы в классе объявляются аналогично объявлению полей, но с указанием ключевого слова *const*:

```
public const float PI = 3.1415f;
```

В приведенных выше примерах часто использовалась запись вида:

```
Convert.ToDateTime("1987, 7, 5");
```

В этой записи у *статического* класса *Convert* вызывается *статический* метод *ToDateTime*(), который переводит строковое представление числа к типу *DateTime*. То есть, для использования класса *Convert* не нужно создавать экземпляр.

3) Перегрузка операторов

Как и C++, язык C# позволяет выполнять перегрузку операторов для их использования в собственных классах. Это позволяет добиться естественного вида определяемого пользователем типа данных и использовать его в качестве основного типа данных. Например, можно создать новый тип данных с именем *ComplexNumber*, представляющий комплексное число, и определить методы выполнения математических операций над такими числами с использованием стандартных арифметических операторов, например оператора + для сложения двух комплексных чисел.

Перегрузка оператора + выглядит следующим образом:

```
// Overloading '+' operator:
public static Customer operator+(Customer a, Customer b)
{
    return new Customer(…);
}
```

Обратите внимание, что оператор является членом класса, а не экземпляра класса (модификатор *static*), а также что результатом выполнения является новый экземпляр класса *Customer*. Аналогично перегружаются операторы -, /,*.

4) Пространства имен

В программировании на C# пространства имен используются с полной нагрузкой по двум направлениям. Во-первых, платформа .NET Framework использует пространства имен для организации большинства классов. Это

выполняется следующим образом.

```
System.Console.WriteLine("Hello World!");
```

System – это пространство имен, а *Console* – класс в нем. Использование ключевого слова *using* может отменить необходимость полного имени, как показано в следующем примере.

```
using System;
```

Во-вторых, объявление собственного пространства имен поможет в управлении областью действия имен классов и методов в крупных программных проектах. Для объявления пространства имен воспользуйтесь ключевым словом namespace, как показано в следующем примере:

```
namespace SampleNamespace
{
    class SampleClass
    {
        public void SampleMethod()
        {
            System.Console.WriteLine("SampleMethod inside SampleNamespace");
        }
    }
}
```

5) Создание нового класса

Для добавления нового класса к существующему проекту выберете пункт меню «Проект->Добавить класс», после этого появится диалоговое окно (рисунок 1):

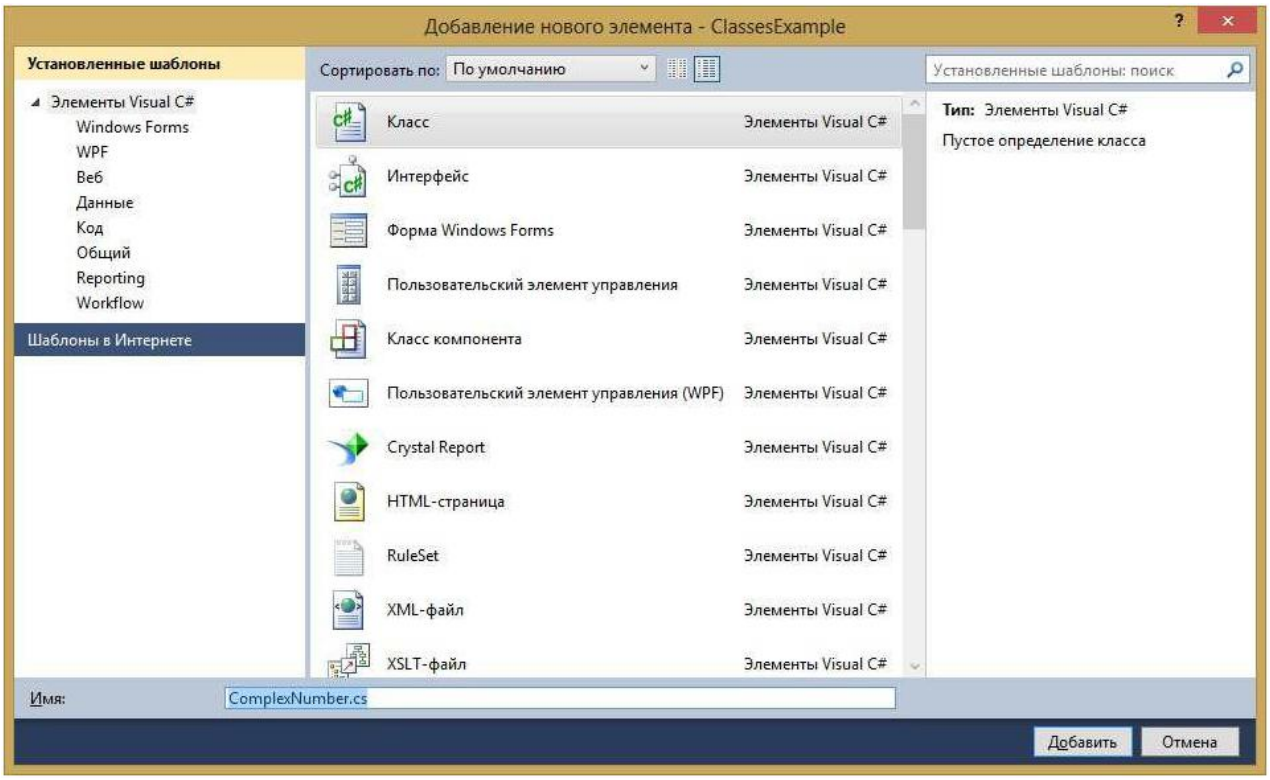

Рисунок 1 – Добавление нового класса

В списке необходимо выбрать пункт «Класс», а в поле имя написать имя класса, например ComplexNumber, и нажать кнопку «Добавить». Среда разработки автоматически сгенерирует код, в котором новый класс автоматически добавится в пространство имен по умолчанию:

```
using System;
using System.Collections.Generic; using System.Linq;
using System.Text;

namespace ClassesExample
{
    class ComplexNumber
    {
    }
}
```

Для просмотра списка классов проекта можно воспользоваться обозревателем решений.

6) Задания к лабораторной работе №2

1) Создать класс «Время», позволяющий суммировать и вычислять разность двух промежутков времени, а также имеющий методы или свойства получения полного количества секунд, минут и часов.

2) Создать класс «Комплексное число», позволяющий вычислять сумму, разность и произведение комплексных чисел, и имеющий свойства – мнимая часть, действительная часть, модуль и аргумент.

3) Создать класс «Вектор в трехмерном пространстве» позволяющий вычислять сумму векторов, их разность и произведение вектора на скаляр, а также имеющий методы или свойства получения модуля, и проекций вектора на оси декартовых и сферических координат.

4) Создать класс «Матрица 3x3» позволяющий вычислять сумму и произведение матриц, а также методы или свойства получения определителя матрицы и суммы элементов на главной и побочной диагонали.

5) Создать класс «Вектор в трехмерном пространстве» позволяющий вычислять сумму векторов, их разность и произведение вектора на скаляр, а также имеющий методы или свойства получения модуля, и проекций вектора на оси декартовых и цилиндрических координат.

6) Создать класс многочлен (порядок меньше 100-го) позволяющий задавать значения коэффициентов, вычислять значение многочлена для произвольного аргумента, а также реализовать операции сложения, вычитания и умножения многочленов.

7) Создать класс "Треугольник". Треугольник определяется координатами трёх вершин на плоскости (x1, y1, x2, y2, x3, y3). Реализовать в нём метод(i) для получения длины i-ой стороны. В классе должны быть реализованы следующие свойства: для получения и изменения координат вершин, для получения периметра треугольника, а так же bool свойства isRight (прямоугольный), isIsosceles (равнобедренный), isEquilateral (равносторонний) говорящие о том, является ли треугольник прямоугольным, равнобедренным

или равносторонним.

8) Создать класс треугольник. Треугольник определяется длинами трёх сторон (a, b, c). Реализовать метод для определения угла противолежащего одной из сторон треугольника (метод в качестве аргумента принимает строку содержащую букву (название стороны) "a", "b" или "c" и возвращает угол в градусах). В классе должны быть реализованы следующие свойства: для получения и изменения длин сторон, для получения периметра и для получения площади по формуле Герона.

7) Контрольные вопросы

1) Что такое класс и экземпляр класса?

2) Что такое статический класс?

3) Что такое поля класса? Каковы особенности предоставления к ним доступа и их именования?

4) Чем принципиально отличаются свойства класса от полей?

5) Что такое метод класса?

6) Какой синтаксис у конструкторов по-умолчанию и с параметрами?

7) Для чего используются перегруженные операторы?

8) Какие модификаторы доступа используются в C#?

9) Опишите особенности применения пространств имён в C#.

10) Для чего используется ключевое слово this?