ЛАБОРАТОРНАЯ РАБОТА №1
БАЗОВЫЕ СРЕДСТВА ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ В ЯЗЫКЕ C#

Цель работы: приобрести навыки реализации классов и их основных элементов на языке C#.

Задачи:

1. Изучить синтаксис создания классов в языке C#
2. Научиться реализовывать классы средствами языка C#.

Результатами работы являются:

- Программа, содержащая разработанный класс и набор модульных тестов для демонстрации его работы.
- Подготовленный отчет

КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ

Класс является наиболее распространенной разновидностью ссылочного типа. Самое простое из возможных объявление класса выглядит следующим образом:

```
class YourClassName{}
```

Перед ключевым словом class указываются модификаторы класса, также перед ключевым словом class могут быть указаны атрибуты. Модификаторами невложенных классов являются модификаторы доступа pubic (доступен для всех), internal (доступен в данной сборке), а так же модификаторы abstract (не содержит или содержит не полную реализацию), sealed (нельзя создавать наследников данного класса), static (нельзя создавать экземпляры данного класса, доступны только статичные поля и методы), unsafe (содержит небезопасный код) и partial (содержит часть описания класса, при сборке класс будет создан из нескольких частей описания).

Далее следует название класса: YourClassName, после которого могут быть указаны параметры обобщенных типов, базовый класс и интерфейсы.

Внутри фигурных скобок указываются члены класса (к ним относятся методы, свойства, индексаторы, события, поля, конструкторы, перегруженные операции, вложенные типы и финализатор).

Поля

Поле - это переменная, которая является членом класса или структуры. Например:

```
class Octopus {
    string name;
    public int age = 10;
}
```

С полями разрешено применять следующие модификаторы:

- Статический модификатор static
- Модификаторы доступа:
  - ❖ public: публичный, общедоступный класс или член класса. Такой член класса доступен из любого места в коде, а также из других программ и сборок.
  - ❖ private: закрытый класс или член класса. Представляет полную противоположность модификатору public. Такой закрытый класс или член класса доступен только из кода в том же классе или контексте.
  - ❖ protected: такой член класса доступен из любого места в текущем классе или в производных классах. При этом производные классы могут располагаться в других сборках.
  - ❖ internal: класс и члены класса с подобным модификатором доступны из любого места кода в той же сборке, однако он недоступен для других программ и сборок (как в случае с модификатором public).
  - ❖ protected internal: совмещает функционал двух модификаторов. Классы и члены класса с таким модификатором доступны из текущей сборки и из производных классов.
  - ❖ private protected: такой член класса доступен из любого места в текущем классе или в производных классах, которые определены в той же сборке.
- Модификатор наследования new
- Модификатор небезопасного кода unsafe
- Модификатор доступа только для чтения readonly
- Модификатор многопоточности volatile

Инициализация полей

Инициализация полей является необязательной. Неинициализированное поле получает свое стандартное значение (0, \0, null, false). Инициализаторы полей выполняются перед конструкторами:

```
pubic int age = 10;
```

Методы

Метод выполняет какое-то действие в виде последовательности операторов. Метод может получать входные данные из вызывающего кода посредством указания параметров и возвращать выходные данные обратно вызывающему коду за счет указания возвращаемого типа. Для метода может быть определен возвращаемый тип void, который указывает на то, что метод никакого значения не возвращает. Метод также может возвращать выходные данные вызывающему коду через параметры ref/out.

Сигнатура метода должна быть уникальной в рамках типа. Сигнатура метода включает в себя имя метода и типы параметров (но не содержит имена параметров и возвращаемый тип).

С методами разрешено применять следующие модификаторы:

- Статический модификатор static
- Модификаторы доступа: public, internal, private, protected
- Модификаторы наследования: new, virtual, abstract, override, sealed
- Модификатор частичного метода partial
- Модификаторы неуправляемого кода: unsafe, extern
- Модификатор асинхронного кода async

Перегрузка методов

Тип может перегружать методы (иметь несколько методов с одним и тем же именем) при условии, что их сигнатуры будут отличаться. Например, все перечисленные далее методы могут сосуществовать внутри одного типа:

```
void Foo (int x) { ... }
void Foo (double x) { ... }
void Foo (int x, float y) { ... }
void Foo ( float x, int y) { ... }
```

Тем не менее, следующие пары методов не могут сосуществовать в рамках одного типа, поскольку возвращаемый тип и модификатор params не входят в состав сигнатуры метода:

```
void Foo (int x) ( ... }
float Foo(int x)(...} // Ошибка на этапе компиляции
void Goo(int[] x){ ... }
void Goo(params int[] x){...} // Ошибка при компиляции
```

Передача по значению или передача по ссылке

Способ передачи параметра - по значению или по ссылке - также является частью сигнатуры. Например, Foo (int) может сосуществовать вместе с Foo (ref int) или Foo (out int). Однако Foo (ref int) и Foo (out int) сосуществовать не могут:

```
void Foo (int x) { ... }
void Foo (ref int x) { ... }
void Foo (out int x) { ... }
```

Конструкторы экземпляров

Конструкторы выполняют код инициализации класса или структуры. Конструктор определяется подобно методу за исключением того, что вместо имени метода и возвращаемого типа указывается имя типа, к которому относится этот конструктор:

```
public class Panda
{
    string name;              // Определение поля
    public Panda (string n)  // Определение конструктора
    {
          name = n;   // Код инициализации (установка поля)
    }
}
…
Panda p = new Panda ("Petey"); // Вызов конструктора
```

Конструкторы экземпляров допускают применение следующих модификаторов:

- Модификаторы доступа: public, internal, private, protected
- Модификаторы неуправляемого кода: unsafe, extern

Перегрузка конструкторов

Класс или структура может перегружать конструкторы. Во избежание дублирования кода один конструктор может вызывать другой конструктор, используя ключевое слово this:

```
using System;
public class Wine
{
    pubic decimal price;
    public int year;
    public Wine(decimal price) { this.price = price; }
    public Wine (decimal price, int year):
          this(price) { this.year = year; }
}
```

Когда один конструктор вызывает другой, вызванный конструктор выполняется первым. Другому конструктору можно передавать выражение, как показано ниже:

```
pubic Wine (decimal price, DateTime year):
    this (price, year.Year) { }
```

В самом выражении использовать, например, ссылку this для вызова метода экземпляра нельзя. (Причина в том, что на этой стадии объект еще не инициализирован конструктором, поэтому вызов любого метода, скорее всего, приведен к сбою.) Тем не менее, можно вызывать статические методы.

Неявные конструкторы без параметров

Компилятор C# автоматически генерирует для класса открытый конструктор без параметров тогда и только тогда, когда в нем не было определено ни одного конструктора. Однако после определения хотя бы одного конструктора конструктор без параметров больше автоматически не генерируется.

Неоткрытые конструкторы

Конструкторы не обязательно должны быть открытыми. Распространенной причиной наличия неоткрытого конструктора является управление созданием экземпляров через вызов статического метода. Статический метод может применяться для возвращения объекта из пула вместо создания нового объекта или для возвращения экземпляра специализированного подкласса, выбираемого на основе входных аргументов:

```
public class Class1
{
    Class1 () // Закрытый конструктор
    public static Class1 Create ( ... )
    {
      //  Выполнение  специальной  логики  для
    возвращения экземпляра Class1
    …
    }
}
```

Ссылка this

Ссылка this указывает на сам экземпляр. В следующем примере метод Marry использует ссылку this для установки поля Mate экземпляра partner:

```
public class Panda
{
   public Panda mate;
   public void Marry (Panda partner)
   {
         Mate = partner;
         partner.Mate = this;
   }
}
```

Ссылка this также разрешает неоднозначность между локальной переменной или параметром и полем. Например:

```
public class Test
{
   string name;
   public Test (string name) { this.name = name; }
}
```

Ссылка this допустима только внутри нестатических членов класса или структуры.

Свойства

Снаружи свойства выглядят похожими на поля, но внутренне они содержат логику подобно методам. Например, взглянув на следующий код, невозможно сказать, чем является CurrentPrice - полем или свойством:

```
Stock msft = new Stock () ;
msft.CurrentPrice = 30;
msft.CurrentPrice -= 3;
Console.WriteLine (msft.CurrentPrice);
```

Свойство объявляется подобно полю, но с добавлением блока get/ set. Ниже показано, как реализовать CurrentPrice в виде свойства:

```
public class Stock
{
    private decimal currentPrice;
    public decimal CurrentPrice
    {
          get return currentPrice;
          set currentPrice = value;
    }
}
```

С помощью `get` и `set` обозначаются средства доступа к свойству. Средство доступа get запускается при чтении свойства. Оно должно возвращать значение, имеющее тип как у самого свойства. Средство доступа set выполняется вовремя присваивания свойству значения. Оно принимает неявный параметр по имени value с типом свойства, который обычно присваивается какому-то закрытому полю (в данном случае currentPrice).

Хотя доступ к свойствам осуществляется таким же способом, как и к полям, свойства отличаются тем, что предоставляют программисту полный контроль над получением и установкой их значений. Такой контроль позволяет программисту выбрать любое необходимое внутреннее представление, не демонстрируя внутренние детали пользователю свойства. В приведенном примере метод set мог бы генерировать исключение, если значение value выходит за пределы допустимого диапазона.

Со свойствами разрешено применять следующие модификаторы:

- Статический модификатор static
- Модификаторы доступа: puЬlic, internal, private, protected
- Модификаторы наследования: new, virtual, abstract, override, sealed
- Модификаторы неуправляемого кода: unsafe, extern

Свойства только для чтения и вычисляемые свойства

Свойство будет предназначено только для чтения, если для него указано одно лишь средство доступа get, и только для записи, если

определено одно лишь средство доступа set. Свойства только для записи используются редко.

Свойство обычно имеет отдельное поддерживающее поле, предназначенное для хранения лежащих в основе данных. Тем не менее, свойство может также возвращать значение, вычисленное на базе других данных. Например:

```
decimal currentPrice, sharesOwned;
public decimal Worth
{
    get { return currentPrice * sharesOwned; }
}
```

Автоматические свойства

Наиболее распространенная реализация свойства предусматривает наличие средств доступа get и/или set, которые просто читают и записывают в закрытое поле того же типа, что и свойство. Объявление автоматического свойства указывает компилятору на необходимость предоставления такой реализации.

```
public class Stock
{
    …
    public decimal CurrentPrice { get; set;}
}
```

Компилятор автоматически создает закрытое поддерживающее поле со специальным сгенерированным именем, ссылаться на которое невозможно. Средство доступа set может быть помечено как private или protected, если свойство должно быть доступно другим типам только для чтения. Автоматические свойства появились в версии C# 3.0.

Инициализаторы свойств

Начиная с версии C# 6, к автоматическим свойствам можно добавлять инициализаторы свойств - в точности как к полям:

```
public decimal CurrentPrice { get; set; } = 123;
```

В результате свойство CurrentPrice получает начальное значение 123. Свойства с инициализаторами могут быть предназначенными только для чтения:

```
public int Maximum { get; } = 999;
```

Как и поля, допускающие только чтение, автоматические свойства, предназначенные только для чтения, могут устанавливаться также в конструкторе типа. Это удобно при создании неизменяемых (допускающих только чтение) типов.

Доступность get и set

Средства доступа get и set могут иметь разные уровни доступа. В типичном сценарии использования есть свойство public с модификатором доступа internal или private, указанным для средства доступа set:

```
public class Foo
{
   private decimal x;
     public decimal X
     {
         get return x;
         private set { x = Math.Round (value, 2);
   }
   }
}
```

Обратите внимание, что само свойство объявлено с более либеральным уровнем доступа (public в данном случае), а к средству доступа, которое должно быть менее доступным, добавлен соответствующий модификатор.

Константы

Константа - это статическое поле, значение которого никогда не может изменяться. Константа оценивается статически на этапе компиляции и компилятор литеральным образом подставляет ее значение всегда, когда она используется (довольно похоже на макрос в C++). Константа может относиться к любому из встроенных числовых типов, bool, char, string или перечислению.

Константа объявляется с помощью ключевого слова const и должна быть инициализирована каким-нибудь значением. Например:

```
public class Test
{
   public const string MESSAGE = "Hello World";
}
```

Константа намного более ограничена, чем поле static readonly - как в типах, которые можно применять, так и в семантике инициализации поля. Константа также отличается от поля static readonly тем, что ее оценка происходит на этапе компиляции. Например:

```
public static double Circumference (double radius)
{
   return 2 * System.Math.PI * radius;
}
```

компилируется в:

```
public static double Circumference (double radius)
{
   return 6.2831853071795862 * radius;
}
```

Для PI имеет смысл быть константой, поскольку ее значение может никогда не меняться. В противоположность этому поле static readonly может иметь разные значения в зависимости от приложения.

Константы могут также объявляться локально внутри метода. Например:

```
static void Main ()
{
    const double twoPI = 2 * System.Math.PI;
    …
}
```

Нелокальные константы допускают использование следующих модификаторов:

– Модификаторы доступа: public, internal, private, protected
– Модификатор наследования new

Модульные тесты в C#

Чтобы выполнить unit-тестирование в C#, необходимо в рамках того же самого решения создать ещё один проект соответствующего типа.

Правой кнопкой щёлкните по решению, выберите “Добавить” и затем “Создать проект…” (Рис. 1).

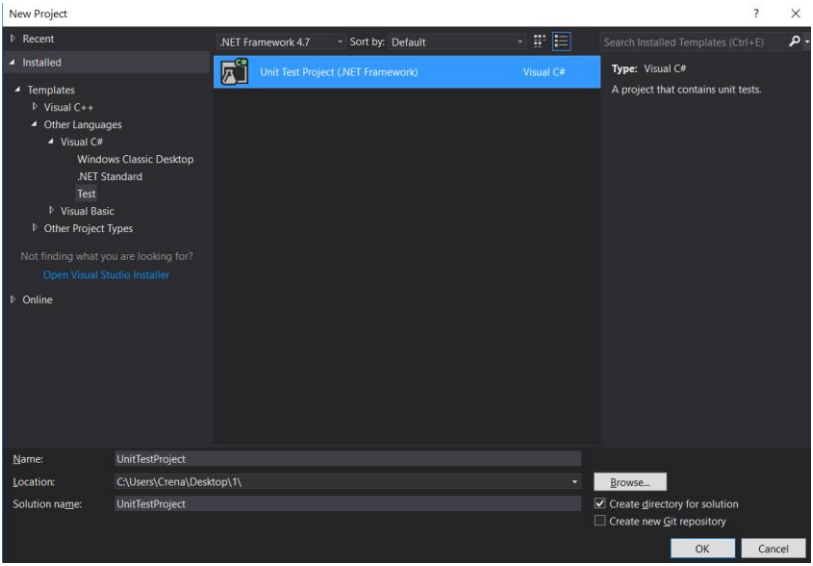

Рис. 1.1. Создание проекта для модульных тестов

Если данного типа проектов нет в списке, нужно добавить модуль тестирования через инсталятор VS.

Сгенерированный код имеет вид:

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace UnitTestProject
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

Директива [TestMethod] обозначает, что далее идёт метод, содержащий модульный (unit) тест. А [TestClass] в свою очередь говорит о том, что далее идёт класс, содержащий методы, в которых присутствуют unit-тесты.

В References проекта необходимо добавить ссылку на проект, код которого будет тестироваться.

Тестирующий метод обычно содержит три необходимых компонента:

1. Исходные данные: входные значения и ожидаемый результат;

2. Код, вычисляющий значение с помощью тестируемого метода;

3. Код, сравнивающий ожидаемый результат с полученным. Для сравнения ожидаемого результата с полученным используется метод AreEqual класса Assert.

Пример теста:

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using MathTaskClassLibrary;

namespace MathTaskClassLibraryTests
{
    [TestClass]
    public class GeometryTests
    {
        [TestMethod]
        public void RectangleArea_3and5_15returned()
        {
            // исходные данные
            int a = 3;
            int b = 5;
            int expected = 15;

            // тестируемый метод
            Geometry g = new Geometry();
            int actual = g.RectangleArea(a, b);

            // сравнение результатов
            Assert.AreEqual(expected, actual);
        }
    }
}
```

Чтобы просмотреть все тесты, доступные для выполнения, необходимо открыть окно “Обозреватель тестов”. Для этого в меню Visual Studio щёлкните на кнопку “ТЕСТ”, выберите “Окна”, а затем нажмите на пункт “Обозреватель тестов” (Рис. 1.2.)

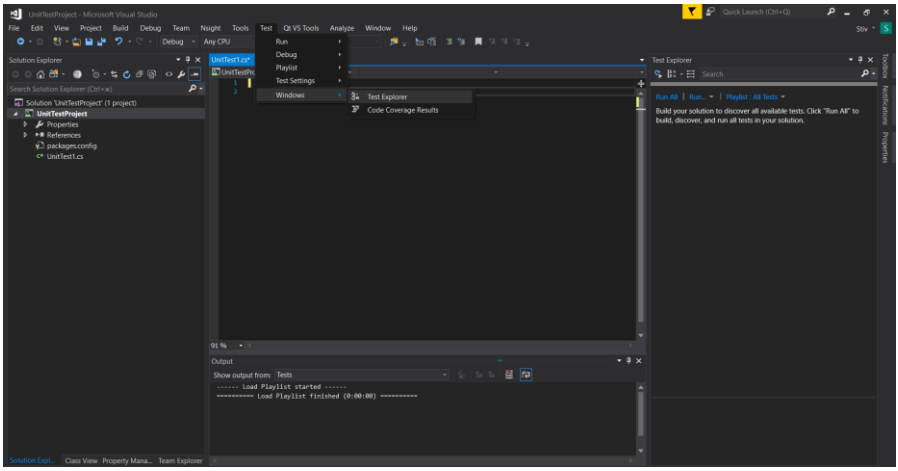

Рис. 1.2. Окно обозревателя тестов

После сборки проекта в окне обозревателя тестов отобразится список тестов, нажав кнопку «Выполнить все тесты» в этом окне отобразится результат выполнения тестов и затраченное время.

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Реализовать класс в соответствии с вариантом задания (см. таблицу ниже). Класс должен содержать:

- закрытые неизменяемые поля, хранящие состояние класса;
- методы для выполнения операций над объектами класса. Эти методы должны применять операцию к текущему объекту (this) и объекту, переданному в качестве аргумента метода. Для представления результата создается новый объект, который и возвращается из метода. При этом ни текущий объект (this), ни объект-аргумент метода не изменяются;
- свойства, возвращающие атрибуты абстракции, представленной классом;
- закрытый конструктор, принимающий аргументы — значения полей;
- статические методы конструирования.

ТРЕБОВАНИЯ К РЕАЛИЗАЦИИ

При выполнении лабораторной работы необходимо использовать стандарт кодирования языка C#. Экземпляры разработанного класса должны быть неизменяемыми. Для разработанного класса создать набор модульных тестов, проверяющих корректность кода, а также демонстрирующих полученный API.

ВАРИАНТЫ ЗАДАНИЙ

Вариант	Класс и его элементы
1	Класс: Комплексное число Состояние (поля): действительная и мнимая части Методы конструирования: создание комплексного числа в алгебраической форме, создание комплексного числа в тригонометрической форме Свойства: действительная часть, мнимая часть, модуль, аргумент Операции: сложение и вычитание
2	Класс: Комплексное число Состояние (поля): модуль и аргумент Методы конструирования: создание комплексного числа в алгебраической форме, создание комплексного числа в тригонометрической форме Свойства: действительная часть, мнимая часть, модуль, аргумент Операции: умножение и деление
3	Класс: Цвет (модель RGB) Состояние (поля): красная, зеленая и синяя составляющие (значения — вещественные числа из промежутка [0; 1]) Методы конструирования: создание цвета в модели RGB, создание цвета в модели CMY Свойства: красная, зеленая, синяя, cyan, magenta, yellow составляющие Операции: сложение и вычитание Примечание: красная, зеленая и синяя составляющие цвета, заданного в модели RGB, и cyan, magenta, yellow составляющие этого же цвета в модели CMY связаны соотношениями: cyan = 1 – red; magenta = 1 – green; yellow = 1 – blue.
4	Класс: Цвет (модель CMY) Состояние (поля): cyan, magenta и yellow составляющие (значения — вещественные числа из промежутка [0; 1]) Методы конструирования: создание цвета в модели RGB, создание цвета в модели CMY Свойства: красная, зеленая, синяя, cyan, magenta, yellow составляющие Операции: сложение и вычитание Примечание: красная, зеленая и синяя составляющие цвета, заданного в модели RGB, и cyan,magenta, yellow составляющие этого же цвета в модели CMY связаны соотношениями: cyan = 1 – red; magenta = 1 – green; yellow = 1 – blue.

5	Класс: Промежуток времени Состояние (поля): количество «тиков» системного таймера Методы конструирования: создание промежутка времени, заданного в секундах; создание промежутка времени, заданного в «тиках» системного таймера Свойства: количество «тиков»; общее количество прошедших секунд; общее количество прошедших минут Операции: сложение и вычитание Примечание: считать, что за секунду происходит 18,2 «тиков» системного таймера
6	Класс: Промежуток времени Состояние (поля): количество секунд Методы конструирования: создание промежутка времени, заданного в минутах; создание промежутка времени, заданного в «тиках» системного таймера Свойства: количество «тиков»; общее количество прошедших секунд; общее количество прошедших минут Операции: сложение и вычитание Примечание: считать, что за секунду происходит 18,2 «тиков» системного таймера
7	Класс: Масса Состояние (поля): величина массы в граммах Методы конструирования: создание величины массы, заданной в граммах; в фунтах Свойства: масса в граммах; в фунтах; в унциях Операции: сложение и вычитание Примечание: 1 фунт = 453,59237 г; 1 унция = 1/16 фунта = 28,349523125 г
8	Класс: Масса Состояние (поля): величина массы в фунтах Методы конструирования: создание величины массы, заданной в унциях; в фунтах Свойства: масса в граммах; в фунтах; в унциях Операции: сложение и вычитание Примечание: 1 фунт = 453,59237 г; 1 унция = 1/16 фунта = 28,349523125 г

9	Класс: Скорость Состояние (поля): величина скорости в метрах в секунду Методы конструирования: создание величины скорости в метрах в секунду; в километрах в час Свойства: скорость в метрах в секунду; в километрах в час; в милях в час Операции: сложение и вычитание Примечание: 1 миля/ч = 1,609 км/ч
10	Класс: Скорость Состояние (поля): величина скорости в километрах в час Методы конструирования: создание величины скорости в милях в час; в километрах в час Свойства: скорость в метрах в секунду; в километрах в час; в милях в час Операции: сложение и вычитание Примечание: 1 миля/ч = 1,609 км/ч

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Приведите синтаксис определения класса на языке C# (поля, методы, конструкторы, свойства).
2. Перечислите области видимости для класса и для его элементов в C#.
3. Объясните по какому принципу следует выбирать область видимости?
4. Сформулируйте определение свойства (property) класса в C#? Приведите синтаксис определения свойства.
5. Перечислите и раскройте основные цели перегрузки (overloading) методов? В каких случаях она должна использоваться?
6. Раскройте значение термина статического элемента (поле или метод) класса.
7. Изложите назначение конструктора.
8. Что такое конструктор по умолчанию? Приведите общий синтаксис определения конструктора. Допускается ли перегрузка конструктора?

ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

На выполнение лабораторной работы отводится 8 часов: 7 часов на выполнение работы, 1 час на подготовку и защиту отчета по лабораторной работе.

Номер варианта студенту выдается преподавателем.

Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)): титульный лист, формулировка задания (вариант), этапы выполнения работы (со скриншотами), результаты выполнения работы. выводы.