ЛАБОРАТОРНАЯ РАБОТА № 5

Разработка классов, создание объектов и использование их в программах

Цель работы: научиться разрабатывать и применять алгоритмы и программы с применением пользовательских классов.

Оборудование: персональный компьютер, практикум, тетради для лабораторных работ.

Программное обеспечение: MS Office, MSVisualStudio

Правила по технике безопасности: общие (приложение).

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Класс представляет собой шаблон, по которому определяется форма объекта. В нем указываются данные и код, который будет оперировать этими данными. В C# используется спецификация класса для построения объектов, которые являются экземплярами класса.

Все члены класса: поля, методы, свойства — все они имеют модификаторы доступа. Модификаторы доступа позволяют задать допустимую область видимости для членов класса. То есть контекст, в котором можно употреблять данную переменную или метод. В предыдущей теме мы уже с ними сталкивались, когда объявляли поля класса Book публичными (то есть с модификатором public).

В C# применяются следующие 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.

Бывает такое, что модификатор доступа не явно не указан, но это не значит что его нет. У каждого элемента есть свой модификатор

доступа по умолчанию

Члены проекта	**По умолчанию**	**Какие можно еще**
namespace	public	-
interfase	public	-
enum	public	-
class	internal	public, internal
struct	internal	public, internal
delegate	internal	public, internal
filed	private	all
method	private	all
constructor	private	all

Разберем один из принципов ООП — инкапсуляция. Представим обычный кофейный автомат. Зададим себе вопрос: «Как он работает?», только нужно ответить точно и подробно: откуда вываливается стаканчик, как поддерживается температура внутри, где хранится сахар, молоко и сам кофе, как автомат понимает какой кофе нужно заварить и т.д. Вероятнее всего, ответов на эти все вопросы мы е сможем дать, но на самом деле это и не нужно, так как не совсем они интересуют. Как же так получается, что мы пользуемся всякими сложными механизмами без понимания, как они устроены и на чем основана их работа? Все просто их создатели предоставили простой и удобный интерфейс.

На кофейном автомате — это кнопки на панели. Нажав одну кнопку, мы выбираем объем стакана. Нажав вторую, выбираешь кофе. Третья отвечает за добавление сахара. И это все, что нам нужно сделать. Таким образом неважно, как именно автомат устроен внутри. Главное — он устроен так, что для получения газировки пользователю нужно нажать три а может и две, одну кнопки.

Именно в этом заключается суть **сокрытия**. Все «внутренности» программы скрываются от пользователя. Для него эта информация является лишней, ненужной. Пользователю необходим конечный результат, а не внутренний процесс.

Давайте для примера посмотрим на класс Auto:

```
class Auto {
   void gas() {

        /*внутри автомобиля происходят какие-то сложные
вещи
        в результате которых он едет вперед*/
   }
   void brake() {
```

```
        /*внутри автомобиля происходят какие-то сложные
вещи
        в результате которых он тормозит*/
    }
    static void Main(String[] args)
{
        Auto auto = new Auto();
        //Как все выглядит для пользователя
        //нажал одну педаль - поехал
        auto.gas();
        //нажал другую педаль - затормозил
        auto.brake();
    }
}
```

Вот как выглядит сокрытие реализации в C#-программе. Все как в реальной жизни: пользователю предоставлен интерфейс (методы). Если ему нужно, чтобы автомобиль в программе выполнил действие, достаточно вызвать нужный метод. А уж, что там происходит внутри этих методов — информация лишняя (для пользователя), главное, чтобы все работало как надо.

Здесь мы говорили про сокрытие реализации. Кроме него в C# есть еще сокрытие данных. О нем мы писали в лекции про геттеры и сеттеры, но не будет лишним напомнить.

Например, у нас есть класс Cat:

```
class Kot
{
    public string name; //кличка кота
    public int age; //возраст кота
    public int weight; //вес кота

    public Kot(string name, int age, string weight)
    {
        this.name = name;
        this.age = age;
        this.weight = weight;
    }
    public Kot() { }

    public void say()
    {
        Console.WriteLine("кот говорит Мяу!!!");
    }
}
```

Этот класс на самом деле написан неверно. Проблема в том, что его данные (поля) открыты для всех, и другой программист легко может создать в программе безымянного кота с весом 0 и возрастом -100 лет:

```
      static void Main(string[] args)
         {
             Kot cat = new Kot();
             cat.age = -100;
             cat.name = "";
             cat.weight = 0;
             cat.name = "dfef";
             Console.Write("Кот по имени {0} мяукает
так", cate.name);
             cat.say ();
         }
```

В такой ситуации можно пристально следить за тем, не создает ли кто-то из твоих коллег объектов с неправильным состоянием, но гораздо лучше было бы исключить саму возможность создавать такие «неправильные объекты».

С сокрытием данных нам помогают:

- модификаторы доступа (private, protected, internal);
- геттеры и сеттеры.

Туда можем, например, заложить проверку, не пытается ли кто-то присвоить коту отрицательное число в качестве возраста.

Использование инкапсуляции дает нам несколько важных преимуществ:

- Контроль за корректным состоянием объекта. Примеры этому были выше: благодаря сеттеру и модификатору private, мы обезопасили нашу программу от котов с весом 0.
- Удобство для пользователя за счет интерфейса. Мы оставляем «снаружи» для доступа пользователя только методы. Ему достаточно вызвать их, чтобы получить результат, и совсем не нужно вникать в детали их работы.
- Изменения в коде не отражаются на пользователях. Все изменения мы проводим внутри методов. На пользователя это не повлияет: он как писал auto.gas() для газа машины, так и будет писать. А то, что мы поменяли что-то в работе метода gas() для него останется незаметным: он, как и раньше, просто будет получать нужный результат.

Что же сделать, чтобы наш класс был написан правильно? Нам нужно защитить данные от некорректного вмешательства извне:

Во-первых, все переменные экземпляра (поля) необходимо помечать модификатором private. Private — самый строгий модификатор доступа в ООП.

```
class Kot
    {
        private string name; //кличка кота
        int age; //возраст кота
        private int weight; //вес кота
        public Kot(string name, int age, int weight)
        {
            this.name = name;
            this.age = age;
            this.weight = weight;
        }
        public Kot() { }
        public void say()
        {
            Console.WriteLine($"кот по кличке {name}, возраст {age} говорит
Мяу!!!");
    }
    class Program
        {
            static void Main(string[] args)
            {
                Kot cat = new Kot();
                cat.Age = 15;//Будет ошибка, так как поле невидимое
                cat.Name = "Tom";;//Будет ошибка, так как поле
    невидимое
                cat.Age = 10; ;//Будет ошибка, так как поле
    невидимое

                cat.say();

            }
        }
```

Компилятор это видит и сразу выдает ошибку.

Теперь поля вроде как защищены. Но получается, что доступ к ним закрыт «намертво»: в программе нельзя даже получить вес существующей кошки, если это понадобится. Это тоже не вариант: в таком виде наш класс практически нельзя использовать. Это можно исправит при помощи создания свойств, и описать у них аксессоры: get, set. Название происходит от английского «get» — «получать» (т.е. «метод для получения значения поля») и set — «устанавливать» (т.е. «метод для установки значения поля»).

Давай посмотрим, как они выглядят на примере нашего класса Cat

```
class Kot
    {
        private string name; //кличка кота
        int age; //возраст кота
        private int weight; //вес кота
        public string Name {
            get { return name; }
            set { name = value; }
        }
```

```
        public int Age {
            set
            {
                if (value < 0 || value > 15)
                {
                    Console.WriteLine($"{value}, Не может быть таким возраст,
поэтому присвоем 1");
                    age = 1;
                }
                else
                    age = value;
            }
        }
        public int Weight { get; set; }

        public Kot(string name, int age, int weight)
        {
            this.name = name;
            this.age = age;
            this.weight = weight;
        }
        public Kot() { }
        public void say()
        {
            Console.WriteLine($"кот по кличке {name}, возраст {age} говорит
Мяу!!!");
    }
    class Program
        {
            static void Main(string[] args)
            {
                Kot cat = new Kot();
                cat.Age = 15;
                cat.Name = "Tom";
                cat.Age = 10;

                cat.say();

            }
        }
```

```
Консоль отладки Microsoft Visual Studio
кот по кличке Tom, возраст 10 говорит Мяу!!!
```

А если попробовать задать значение возраста отрицательное значение

```
    class Program
        {
            static void Main(string[] args)
            {
                Kot cat = new Kot();
                cat.Age = 15;
                cat.Name =-1;

                cat.say();

            }
        }
```

```
Консоль отладки Microsoft Visual Studio
-1, Не может быть таким возраст, поэтому присвоем 1
кот по кличке Tom, возраст 1 говорит Мяу!!!
```

Практическая часть

Написать класс, согласно индивидуальному заданию, описать все необходимые поля, проинкапсулировать их, описать явные get, set.

1	Описать класс, реализующий десятичный счетчик, который может увеличивать или уменьшать свое значение на единицу в заданном диапазоне. Предусмотреть инициализацию счетчика значениями по умолчанию и произвольными значениями. Счетчик имеет два метода: увеличения и уменьшения, — и свойство, позволяющее получить его текущее состояние. Написать программу, демонстрирующую все возможности класса
2	Описать класс «домашняя библиотека». Предусмотреть возможность работы с произвольным числом книг, поиска книги по какому-либо признаку (например, по автору или по году издания), добавления книг в библиотеку, удаления книг из нее, сортировки книг по разным полям
3	Составить описание класса для представления времени. Предусмотреть возможности установки времени и изменения его отдельных полей (час, минута, секунда) с проверкой допустимости вводимых значений. В случае недопустимых значений полей выводится строка ошибки, и дается возможность еще раз ввести. Создать методы изменения времени на заданное количество часов, минут и секунд, на просто часов, на просто минут, на часов и минут вместе.
4	Создать класс для хранения комплексных чисел. Реализовать операции над комплексными числами: сложение, вычитание, умножение, деление, сопряжение, возведение в степень, извлечение корня. Предусмотреть возможность изменения формы записи комплексного числа: алгебраическая форма, тригонометрическая форма, экспоненциальная форма
5	Составить описание класса для вектора, заданного координатами его концов в трехмерном пространстве. Обеспечить операции сложения и вычитания векторов с получением нового вектора (суммы или разности), вычисления скалярного произведения двух векторов, длины вектора, косинуса угла между векторами.
6	Описать класс, представляющий треугольник. Предусмотреть методы для создания объектов, вычисления площади, периметра и точки пересечения медиан. Описать свойства для получения состояния объекта.
7	Счета. Клиент может иметь несколько счетов в банке. Учитывать возможность блокировки/разблокировки счета. Реализовать поиск и сортировку счетов. Вычисление общей суммы по счетам. Вычисление суммы по всем счетам, имеющим положительный и отрицательный балансы отдельно.

8	Описать класс «Абонент». Класс должен включать в себя следующие поля (свойства): • Фамилия • Имя • Отчество • Прозвище • Группа (Семья, Друзья, Коллеги, VIP, Бизнес…) • Дата рождения • Адрес • Телефон • Электронный адрес • … Определить конструктор без параметров, конструкторы с параметрами. Продумать, какие свойства должны быть доступны только для просмотра, а какие – и для изменения значения. Переопределить стандартный метод ToString() для перевода информации об абоненте в строковый формат. Определить методы для сравнения двух абонентов по указанному свойству.
9	Описать класс «Студент». Класс должен включать в себя следующие поля (свойства): • Фамилия • Имя • Отчество • Дата рождения • Адрес • Телефон • Электронный адрес • Курс • Группа • Номер зачетной книжки • … Определить конструктор без параметров, конструкторы с параметрами. Продумать, какие свойства должны быть доступны только для просмотра, а какие – и для изменения значения. Переопределить стандартный метод ToString() для перевода информации о студенте в строковый формат. Определить методы для сравнения двух студентов по указанному свойству.
10	Описать класс «Книга». Класс должен включать в себя следующие поля (свойства): • Фамилия автора • Имя автора • Отчество автора • Название книги • Код • Год издания • Количество страниц • … Определить конструктор без параметров, конструкторы с параметрами. Продумать, какие свойства должны быть доступны только для просмотра, а какие – и для изменения значения. Переопределить стандартный метод ToString() для перевода информации о книге в строковый формат. Определить методы для сравнения двух книг по указанному свойству
11	Создать Класс Room, в котором описать поля ширина, высота, длина комнаты и количества окон (ширина, высота). Описать метод который находит площадь комнаты (вместе с окнами и дверями).Описать метод поиска площади стен комнаты за вычетом оконных и дверных проемов. Написать метод, который рассчитывает сколько обойных трубок нужно потратить на переклейку данной комнаты (размер трубки 0,53м на 10 м) сколько это будет стоить, цену трубки пользователь вводит сам с клавиатуры.
12	Создать класс Employe, в котором описать поля фамилия, имя, год рождения, должность, оклад, год поступления на работу. Описать метод увиличения/уменьшение оклада работника, метод определения возраста работкника, срока работы в данной компании, определение количества дней до юбилея, если больше чем один год то вывести количество лет и дней, при этом учитывать что есть года високосные, а есть невисокосные
13	Создать класс Book, в котором описать поля название книги, количество страниц, цена книги. Описать метод увеличения цены

	книги в 2 раза, если в названии книги есть слово «программирование». Написать метод который высчитывает стоимость страницы книги. Если год издания превышает 5 лет, то стоимость книги уменьшается на 5 процентов, и каждые пять лет продолжает уменьшаться, максимальное уменьшение составляет 50 процентов
14	Создать класс ComplexNumber с полями содержащими действительную и мнительную часть числа. Описать метод поиска модуля комплексного числа, описать метод поиска комплексного числа обратное заданному, написать методы увеличения/уменьшения действительной и мнимой части на заданное число
15	Написать класс Roll (правильной дроби), содержащий поле числителя и знаменателя. Выразить значение дроби в процентах. Написать метод, вычисления суммы цифр знаменателя. Написать метод определения является ли числитель простым или составным, проверить на сокращение дроби , если дробь возможно сократить, то сократить.
16	Написать класс Vector с полями x1, y1, x2, y2. Написать метод определяющий длину вектора, метод определения равен ли угол наклона вектора 45 градусов, метод вывода координаты точки середины вектора.
17	Написать класс Box, с полями ширина, высота, и глубина. Написать метод поиска площади, объема. Написать метод нахождения самой длинной диагонали. Написать метод который определяет форму коробки (куб, полукуб, параллелепипед)
18	Класс Product с полями наименование, цена в рублях, бренд, год изготовления товара. Написать метод перевода цены в доллары и евро по текущему курсу. Метод, определяющий возраст товара, метод увеличения цены на 20% если в наименовании товара присутствует стово «TV»
19	Класс Rectangle с полями координаты левого верхнего угла и нижнего правого. Написать метод который рисует заданный прямоугольник звездочками на консоли, метод определяющий является ли это прямоугольник квадратом.