Практическая работа №1

1 Теоретические сведения

1.1 Классы и объекты

Класс – основной тип данных языка C#. Класс представляет собой конструкцию, которая объединяет поля, свойства и методы. Класс является определением для создания объектов или экземпляров класса.

Синтаксис объявления:

```
[modifier] class ClassName {
  // class body
}
```

В начале идет модификатор доступа, а после ключевого слова class, имя класса, в фигурных скобках находиться тело класса, которое может содержать поля, свойства и методы – члены класса.

Простой пример объявления класса для хранения данных о пользователе:

```
public class User
{
    public string UserName;
    public byte UserAge;
}
```

Для создания экземпляра класса используется оператор new:

```
User u = new User();
```

Класс может содержать в себе следующие члены:

- Конструкторы;
- Константы;

- Поля;
- Методы;
- Свойства;
- Операторы;
- Вложенные типы данных;
- Деструкторы.

Доступ к членам экземпляра класса осуществляется через оператор ”.”, например u.UserAge = 21;

Модификатор доступа – определяет откуда можно обращаться к классу или его членам.

В языке C# доступны следующие уровни доступа:

- public – максимально доступный уровень, не налагает никаких ограничений;
- protected – доступ разрешен из текущего класса или его наследников;
- internal – доступ ограничен текущей сборкой(в пределах программы, библиотеки);
- protected internal – комбинация из предыдущих двух модификаторов;
- private – доступ разрешен только в текущем классе;
- private protected – доступ разрешен в текущем классе и классах наследниках расположенных в той же сборке.

Если классу или члену не задан модификатор доступа, то устанавливается модификатор по умолчанию internal для класса и private для членов.

Конструктор – специальный метод который вызывается при создании нового экземпляра класса, он выделяет память необходимую для хранения объекта, и как правило выполняет инициализацию полей и свойств. Имя конструктора должно быть идентично имени класса. Если в классе не задан конструктор, то компилятор генерирует конструктор по умолчанию без параметров.

Пример конструктора класса:

```
public class TrackPoint
{
    //
    public float X;
```

```
    public float Y;

    //
    public TrackPoint(float x, float y)
    {
        //
        X = x;
        Y = y;
    }
}
```

Сейчас при создании объекта класса TrackPoint необходимо передавать в конструктор аргументы:

```
var tp = new TrackPoint(2f, 3f);
```

Класс может содержать сколько угодно конструкторов, которые создаются по аналогии с перегрузкой методов:

```
public class RGBColor
{
    //
    public int Red;
    public int Green;
    public int Blue;

    //
    public RGBColor()
    {
    }

    //
    public RGBColor(int r, int g = 0, int b = 0)
    {
        Red = r;
        Green = g;
        Blue = b;
    }
}
```

Создание экземпляров класса:

```
var c1 = RGBColor();
var c2 = RGBColor(10, 20);
```

Начиная с 7.0 версии языка C# конструктор с одним выражением можно записать в сокращенной форме:

```
public class Dog
{
    public string Name;
    public uint Weight;

    //
    public Dog(string n) => Name = n; //
    public Dog(string n, uint w) => SetParameters(n, w); //                    (                    )

    private void SetParameters(string n, uint w)
    {
        Name = n;
        Weight = w;
    }
}
```

this – указывает на текущий экземпляр класса. Обычно используется для разделения параметров конструктора от полей с такими же названиями:

```
public class Worker
{
    private string workerName;

    public Worker(string workerName)
    {
        //
        this.workerName = workerName;
    }
}
```

Поля класса – это переменные которые объявлены внутри класса. Не рекомендуется использовать публичные поля, доступ к ним должен осуществляться посредством свойств и методов, а инициализация обычно делается из конструктора.

Используйте поля с модификаторами доступа private или protected, использование public полей является плохой практикой.

Константы – это идентификаторы, значение которых задается во время компиляции программы, и не может быть изменено в процессе выполнения приложения.

```
internal class CircleFigure
{
    //
    internal const double Pi = 3.1415;
    //
    private double r;

    internal CircleFigure(double radius)
    {
        r = radius;
    }

    //
    internal double GetRadius() => r;

    //
    internal double CalculateArea() => Pi * r * r;
}
```

2 Практические задания

2.1 Задание 1

Реализуйте класс Point объявляющий точку. Любая точка должна обладать координатами x, y. Класс должен иметь конструктор инициализирующий поля объекта, а также конструктор без аргументов инициализирующий поля объекта нулями. А также реализовывать методы взаимодействия с другими точками и числами:

- void AddNumber(int n) добавление к координатам точки числа n
- void MinusNumber(int n) вычитание числа n из координат точки
- void AddPoint(Point p) сложение координат точки у которой вызывается метод с координатами точки p
- void MinusPoint(Point p) вычитание координат точки у которой вызывается метод

Для решения задания создайте файл Point с классом Point и опишите реализацию класса. Задание считается пройденным, если после запуска команды dotnet test все тесты успешно пройдены.

3 Развертывание и использование проекта

Проект выполняется в Visual Studio Code с плагинами C# и Dotnet Essentials.

Если вы не знакомы с системой контроля версий, то можете скачать проект в виде zip архива.

После скачивания проекта необходимо установить зависимости проекта dotnet restore в папке проекта.

Для успешной сдачи необходимо, что бы написанный вами код проходил автоматические тесты, которые можно запустить командой dotnet test в каталоге Test.