ООП в C#

(классы, объекты)

Андрей Голяков

Классы, объекты

- C# является полноценным объектно-ориентированным языком. Это значит, что программу на C# можно представить в виде взаимосвязанных взаимодействующих между собой объектов.
- Мы используем понятие класс, когда говорим о шаблоне объектов, с которыми работает программа.
- Например, класс “кредитная карта” описывает основные свойства и действия, которые можно производить с кредиткой.
- Объектом, или экземпляром класса, принято называть конкретный предмет, в нашем примере с классом "кредитная карта" объектами могут выступать конкретные кредитки, которыми мы пользуемся.

Классы, объекты

Поскольку мы говорим о классе как о модели предметов реального мира, следует сказать, что для разных областей будут важны разные характеристики этих объектов.

Пример класса "Кредитная карта":

- Имя держателя
- Номер
- Срок действия
- CVC-код
- Физический размер

Объектом же будет конкретная кредитка, имеющая конкретные значения указанных свойств:

- Имя держателя : Andrei Goliakov
- Номер : 2222 3333 4444 5555
- Срок действиям : 10/23
- CVC-код : 123
- Физический размер : 85.6 (мм) × 53.98 (мм)

Классы, объекты

Проект консольного приложения по умолчанию содержит класс Program, с которого и начинается выполнение программы (а именно с метода Main).

Этот метод данного класса является точкой входа в наше приложение (если не задано иное).

```
class Program
{
    static void Main()
    {
        ...
    }
}
```

Классы, объекты

Где определяется класс? Класс можно определять внутри пространства имен, вне пространства имен, внутри другого класса. Как правило, классы помещаются в отдельные файлы. Но в первом примере мы поместим новый класс в файле, где располагается класс Program. То есть файл Program.cs будет выглядеть следующим образом:

```
class Program
{
    static void Main()
    {
        ...
    }
}

class Person
{
    // our new empty class
}
```

Члены класса

Внутри класса определяются его члены:

- поля / fields: представляют собой данные, содержащиеся в классе, они служат для хранения информации об объекте,
- свойства / properties: представляют собой данные, совмещенные с реализацией доступа к этим данным, они также служат для хранения информации об объекте.
- методы / methods: функции, действия которые можно производить над или с помощью объекта.
- конструкторы / constructors: конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта.
- существуют также другие, более специфичные члены, такие как события, операторы и индексаторы; мы рассмотрим некоторые из них на следующих уроках.

Инкапсуляция в ООП

Инкапсуляция (encapsulation) - это механизм, который объединяет данные и код, манипулирующий этими данными, а также защищает и то и другое от внешнего вмешательства или неправильного использования с помощью регулирования доступа к внутренним членам.

Целью инкапсуляции является обеспечение согласованности внутреннего состояния объекта.

Например, представим класс описывающий кошелек, средства с которого можно снять в различной валюте:

```
класс Кошелек
    поле Доступные средства в рублях  : 6600
    поле Доступные средства в долларах : 100
```

Сейчас состояние объекта согласовано, так как количество денег в каждой из валют одинаково. Однако, в процессе работы с объектом такого класса его легко ввести в неконсистентное состояние, изменив доступную сумму только в одном месте, и не сделав этого во втором.

Инкапсуляция в ООП

Проблема такого класса в том, что он позволяет разработчику, использующему его, напрямую менять количество денег, доступных в каждой из валют.

```
класс Кошелек
    // поля лучше сделать закрытыми от разработчика
    закрытое снаружи поле Доступные средства в рублях   : 6600
    закрытое снаружи поле Доступные средства в долларах : 100

    // а вместо этого предоставить возможность менять количество
    // денег согласованно
    доступный метод Изменить количество средств в рублях(новая сумма)
        Доступные средства в рублях = новая сумма
        Доступные средства в долларах = новая сумма / 66

    доступный метод Изменить количество средств в долларах(новая сумма)
        Доступные средства в долларах = новая сумма
        Доступные средства в рублях = новая сумма * 66
```

Уровни доступа к членам класса

При объявлении членов класса разработчик также определяет и уровень доступа к каждому из них, определяющие возможность их использования из другого кода в вашей или в других сборках.

Существуют следующие модификаторы доступа:

- * public - Доступ к типу или члену возможен из любого другого кода в той же сборке или другой сборке, ссылающейся на него.
- * private - Доступ к типу или члену возможен только из кода в том же классе (по умолчанию).
- protected - Доступ к типу или члену возможен только из кода в том же классе либо в классе, производном от этого класса.
- internal - Доступ к типу или члену возможен из любого кода в той же сборке, но не из другой сборки.
- protected internal - Доступ к типу или члену возможен из любого кода в той сборке, где он был объявлен, или из производного класса в другой сборке.
- private protected - Доступ к типу или члену возможен только из его объявляющей сборки из кода в том же классе либо в типе, производном от этого класса.

Пример класса Person

```
class Person
{
	int _age;			// implicitly private field: bad practice! always use "private"!
					// underscore "_" prefix is used for member variables

	public int Age			// public property
	{
		get			// getter of the property
		{
			return _age;
		}
		set			// setter of the property
		{
			if (value > 0 && value < 140) // logic of set
			{
				_age = value;
			}
		}
	}

	public string Name { get; set; } // another one public property with no additional logic
}
```

Самостоятельная работа

Объявить класс домашнего питомца с именем Pet.

Определить в классе следующие закрытые поля:

- _birthPlace: место рождения животного (страна, город)

Следующие общедоступные поля:

- Kind: вид животного (один из определённых вариантов Mouse, Cat, Dog)
- Name: кличка питомца

Следующие общедоступные свойства:

- Sex: пол животного (одна латинская буква: M - мужской, F - женский)
- Age: возраст животного (в годах)

Пример решения

```
public class Pet
{
    public enum AnimalKind { Mouse, Cat, Dog }

    private string _birthPlace;

    public AnimalKind Kind;
    public string Name;

    public char Sex { get; set; }
    public byte Age { get; set; }
}
```

Пример создания экземпляра класса

Класс становится обычным для C# типом данных, и мы просто создаем переменную этого класса с помощью ключевого слова new. После этого с помощью точки можно получить доступ к доступным полям и свойствам созданного объекта:

```
using System;
class Program
{
      static void Main()
      {
            Person p1 = new Person();

            // p1._height - inaccessible as it is private
            // p1._age - inaccessible as it is private
            p1.Name = "Andrei";
            p1.Age = 36;

            Console.WriteLine($"Name: {p1.Name}, Age: {p1.Age}.");
            // Name: Andrei, Age: 36.
      }
}
... // assuming that class defined below in the code
```

Пример создания экземпляра класса

Можно сразу инициализировать значения доступных снаружи членов (public) при создании экземпляра класса. Для этого вместо круглых скобок пишут фигурные, а внутри перечисляют разделенные запятыми пары Поле = Значение. Вот пример аналогичного по смыслу кода:

```
using System;
class Program
{
      static void Main()
      {
            Person p1 = new Person
            {
                  Name = "Andrei",
                  Age = 36
            };

            Console.WriteLine($"Name: {p1.Name}, Age: {p1.Age}.");
            // Name: Andrei, Age: 36.
      }
}
... // assuming that class defined below in the code
```

Самостоятельная работа

В основном потоке программы:

Создать экземпляр класса Pet с именем pet1 и заполнить его поля.

Вывести на экран строку в формате:

$"{pet1.Name} is a {pet1.Kind} ({pet1.Sex}) of {pet1.Age} years old."

Ниже создать еще один экземпляр класса Pet с именем pet2, задав ему значения полей при инициализации используя фигурные скобки.

Вывести на экран такую же строку, только теперь уже для pet2:

$"{pet2.Name} is a {pet2.Kind} ({pet2.Sex}) of {pet2.Age} years old."

Пример решения

```
static void Main(string[] args)
{
     Pet pet1 = new Pet();
     pet1.Kind = Pet.AnimalKind.Cat;
     pet1.Name = "Tom";
     pet1.Sex = 'M';
     pet1.Age = 8;

     Console.WriteLine(
          $"{pet1.Name} is a {pet1.Kind} ({pet1.Sex}) of {pet1.Age} years old.");

     Pet pet2 = new Pet
     {
          Kind = Pet.AnimalKind.Mouse,
          Name = "Minnie",
          Sex = 'F',
          Age = 1
     };

     Console.WriteLine(
          $"{pet2.Name} is a {pet2.Kind} ({pet2.Sex}) of {pet2.Age} years old.");
}
```

Инкапсуляция, read-only свойство #1

Чтобы унифицировать формат вывода данных о нашей персоне, воспользуемся свойствами, доступными только для чтения – read-only property – у таких свойств есть описан только метод get (а метод set – отсутствует).

Добавим в наш класс read-only свойство PropertiesString типа string:

```
class Person
{
      ...

      // read-only properties have get method only!
      public string PropertiesString
      {
            get
            {
                  return $"Name: {Name}, Age: {Age}.";
            }
      }
}
```

Инкапсуляция, read-only свойство #2

:

```
using System;
class Program
{
      static void Main()
      {
            Person p1 = new Person();
            p1.Name = "Andrei";
            p1.Age = 36;

            // Now we shouldn't think about the format of output for parameters in the main code
            // We can just call PropertiesString where all the logic is encapsulated
            Console.WriteLine(p1.PropertiesString);

            // The line below will not be compiled as the property is read-only!
            p1.PropertiesString = "Try to override read-only property"; // compile error!
      }
}
```

Самостоятельная работа

Модифицируйте класс домашнего питомца с именем Pet таким образом, чтобы у него появилось read-only свойство Description, формирующее строку вывода информации о питомце на экран.

Модифицируйте соответственно основную программу, чтобы она использовала свойство Description для вывода информации о питомце.

Пример решения

```
public class Pet
{
     ...
     public string Description
     {
          get { return $"{Name} is a {Kind} ({Sex}) of {Age} years old."; }
     }
}

...

static void Main()
{
     ...
     Console.WriteLine(pet1.Description);

     ...
     Console.WriteLine(pet1.Description);
}
```

Самостоятельная работа

Добавьте логику проверки значения при установке пола (свойство Sex): задавать можно заглавные или строчные буквы M или F.

Обеспечьте вывод значения пола в верхнем регистре.

Пример решения (изменения класса Pet)

```
private char _sex;

public char Sex
{
     get
     {
          return _sex;
     }
     set
     {
          if (value == 'f' || value == 'F' || value == 'm' || value == 'M')
          {
               _sex = char.ToUpper(value);
          }
          else
          {
               throw new Exception("Invalid value");
          }
     }
}
```

Самостоятельная работа

В классе Pet добавьте метод SetBirthPlace со строковым параметром для установки места рождения питомца.

В свойство Description класса Pet добавьте упоминание места рождения.

В основном потоке программы задать место рождения для каждого из двух питомцев.

Пример решения (изменения класса Pet)

```
public void SetBirthPlace(string birthPlace)
{
    _birthPlace = birthPlace;
}

public string Description
{
    get
    {
        return $"{Name} is a {Kind} ({Sex}) of {Age} years old" +
            $" (birth place: {_birthPlace}.";
    }
}
```

Пример решения (метод Main)

```
Pet pet1 = new Pet();
pet1.Kind = Pet.AnimalKind.Cat;
pet1.Name = "Tom";
pet1.Sex = 'M';
pet1.Age = 8;
pet1.SetBirthPlace("Moscow");

Console.WriteLine(pet1.Description);


Pet pet2 = new Pet
{
     Kind = Pet.AnimalKind.Mouse,
     Name = "Minnie",
     Sex = 'F',
     Age = 1
};
pet2.SetBirthPlace("St.Petersburg");

Console.WriteLine(pet2.Description);
```

Домашнее задание

Написать консольное приложение, запрашивающее имя и возраст для трех человек. Затем программа должна вывести на экран информацию о людях и их возрастах через 4 года в следующем формате (схожая задача задавалась на третьем уроке):

```
Name: <name of the person # 1>, age in 4 years: <age of the person #1 in 4 years>
```

- Программа не должна закрываться пока не нажата любая клавиша.
- Необходимо выполнить задание с использованием массива объектов нового класса и циклом for!
- В классе должны быть определены следующие свойства:
  - Имя (обычное свойство),
  - Возраст (обычное свойство),
  - Возраст через четыре года (read-only свойство),
  - Итоговая строка вывода информации (read-only свойство).

- Не забывать после создания массива инициализировать каждый объект внутри перед началом использования с помощью ключевого слова **new**

Домашнее задание (пример работы)

Пример работы программы (при вводе данных без ошибок):

```
> Enter name 0: Andrei /это ввод пользователя/
> Enter age 0: 36 /это ввод пользователя/
> Enter name 1: Vasili /это ввод пользователя/
> Enter age 1: 32 /это ввод пользователя/
> Enter name 2: Marina /это ввод пользователя/
> Enter age 2: 18
> Name: Andrei, age in 4 years: 40.
> Name: Vasili, age in 4 years: 36.
> Name: Marina, age in 4 years: 22.
> Press any key to continue..
```

Спасибо за внимание.