ООП в C#

(наследование, полиморфизм, интерфейсы)

Вислобоков Денис

Наследование

Наследование является одним из фундаментальных атрибутов объектно-ориентированного программирования.

Оно позволяет определить дочерний класс, который использует (наследует), расширяет или изменяет возможности родительского класса.

Класс, члены которого наследуются, называется базовым классом. Класс, который наследует члены базового класса, называется производным (дочерним) классом.

C# и .NET поддерживают только одиночное наследование. Это означает, что каждый класс может наследовать члены только одного класса.

Наследование

```
class Person
{
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    public void Display()
    {
        Console.WriteLine(Name);
    }
}

class Employee : Person
{

}
```

```
static void Main(string[] args)
{
    Person p = new Person { Name = "Tom"};
    p.Display();
    p = new Employee { Name = "Sam" };
    p.Display();
    Console.Read();
}
```

Наследование

Пример простого наследования

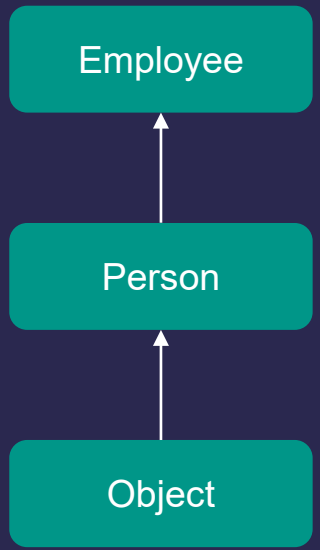
```
// Базовый класс
public   class Person
{
  public string Name { get; set; }
}

// Дочерний класс
public class Employee : Person {}

private static void Main()
{
  Person p1 = new Person
  {
    Name = "Andrei"
  };

  Employee e1 = new Employee
  {
    Name = "Sergei"
  };

  Console.WriteLine(p1.Name);
  Console.WriteLine(e1.Name);
}
```



Конструктор и ключевое слово base

Конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта .

```
class Person
{
    public string name;
    public int age;

    public Person() { name = "Неизвестно"; age = 18; }      // 1 конструктор

    public Person(string n) { name = n; age = 18; }         // 2 конструктор

    public Person(string n, int a) { name = n; age = a; }   // 3 конструктор

    public void GetInfo()
    {
        Console.WriteLine($"Имя: {name}  Возраст: {age}");
    }
}
```

Самостоятельная работа

Напишите собственный базовый класс BaseDocument, который бы описывал произвольный документ и имел бы:

- Свойства:
  - DocName типа string: наименование документа
  - DocNumber типа string: номер документа
  - IssueDate типа DateTimeOffset: дата выдачи
  - PropertiesString типа string: read-only свойство, формирующее строку для вывода на экран свойств этого класса
- Метод
  - WriteToConsole()

Затем напишите производный класс Passport, унаследованный от BaseDocument, который бы имел дополнительно свойства Country (string) для хранения страны и PersonName (string) для хранения имени владельца.

Напишите новую реализацию свойств PropertiesString и метод WriteToConsole() чтобы произвести сокрытие членов базового класса.

Создайте по одному экземпляру каждого класса в основном потоке программы, инициализируйте их свойства и выведите их на экран используя метод WriteToConsole() соответствующих классов.

Полиморфизм

Полиморфизм —слово греческого происхождения, означающее "многообразие форм" и имеющее несколько аспектов.

- Во время выполнения объекты производного класса могут обрабатываться как объекты базового класса в таких местах, как параметры метода и коллекции или массивы. Когда это происходит, объявленный тип объекта перестает соответствовать своему типу во время выполнения.
- Базовые классы могут определять и реализовывать виртуальные *методы*, а производные классы —переопределять их, т. е. предоставлять свое собственное определение и реализацию. Во время выполнения, когда клиент вызывает метод, CLR выполняет поиск типа объекта во время выполнения и вызывает перезапись виртуального метода. Таким образом, в исходном коде можно вызвать метод на базовом классе и привести версию производного класса метода, который необходимо выполнить.

Полиморфизм

Полиморфизм позволяет без лишнего вмешательства в код внести дополнительный функционал. Полиморфизм и наследование очень сильно связаны. Обычно они идут вместе друг с другом.

```
public class Animal {
     public void Say() {
         Console.WriteLine("Я зверь");
     }
}
```

```
public class Dog : Animal{
        public new void Say() {
            Console.WriteLine("Гав");
        }
    }

    public class Cat : Animal
    {
        public new void Say()
        {
            Console.WriteLine("Meow");
        }
    }
```

Интерфейсы

- В большинстве статей смысл интерфейса разъясняется как «договор» о том, что должен содержать класс, какие свойства и методы .
- К счастью возможности интерфейса намного интереснее. Интерфейс может задать общий признак для разнородных объектов, а это открывает огромные возможности по части гибкости кода.
- Интерфейс не может иметь полей и реализованных методов

```
public interface IAnimal
    {
        int Age; // Ошибка компиляции
        int GetAge(int value);
        int Size { get; set; }
    }
```

Интерфейсы

```
class First{}
class Second{}
```

```
class First{}:Animal
class Second{}:Animal
```

В чем разница?

Интерфейсы

```
var array = new IAnimal[2];
array[0] = new Cat();
array[1] = new Dog();
```

```
var summ = 0;
foreach (var animal in array) {
                summ += animal.Size;
}
```

Домашнее задание

Написать интерфейс одной записи будильника IReminderItem (как будильник в телефоне), который будет иметь

- Свойства:
  - AlarmDate типа DateTimeOffset (дата/время будильника)
  - AlarmMessage типа string (сообщение, соответствующее будильнику)
  - TimeToAlarm типа TimeSpan (время до срабатывания будильника), должно быть read-only, рассчитываться как текущее время минус AlarmDate
  - IsOutdated типа bool (просрочено ли событие), должно быть read-only, рассчитываться как
    - true, если TimeToAlarm больше либо равно 0
    - false, если TimeToAlarm меньше 0
- Методы:
  - Конструктор, который будет инициализировать значения AlarmDate и AlarmMessage.
  - WriteProperties(), который будет выводить на экран все свойства экземпляра класса в формате “Имя поля : значение”.

В основном потоке программы создать два экземпляра класса ReminderItem, который наследуется от IReminderItem и вывести их параметры на экран.

Спасибо за внимание.