Абстрактные классы и члены классов

Кроме обычных классов в C# есть **абстрактные классы**. Абстрактный класс похож на обычный класс. Он также может иметь переменные, методы, конструкторы, свойства. Единственное, что при определении абстрактных классов используется ключевое слово **abstract**:

```
abstract class Human
{
    public int Length { get; set; }
    public double Weight { get; set; }
}
```

Но главное отличие состоит в том, что мы **не можем** использовать конструктор абстрактного класса для создания его объекта. Например, следующим образом:

```
Human h = new Human();
```

Зачем нужны абстрактные классы? Допустим, в нашей программе для банковского сектора мы можем определить две основных сущности: клиента банка и сотрудника банка. Каждая из этих сущностей будет отличаться, например, для сотрудника надо определить его должность, а для клиента - сумму на счете. Соответственно клиент и сотрудник будут составлять отдельные классы Client и Employee. В то же время обе этих сущности могут иметь что-то общее, например, имя и фамилию, какую-то другую общую функциональность. И эту общую функциональность лучше вынести в какой-то отдельный класс, например, Person, который описывает человека. То есть классы Employee (сотрудник) и Client (клиент банка) будут производными от класса Person. И так как все объекты в нашей системе будут представлять либо сотрудника банка, либо клиента, то напрямую мы от класса Person создавать объекты не будем. Поэтому имеет смысл сделать его абстрактным:

```
abstract class Person
{
    public string Name { get; set; }

    public Person(string name)
    {
        Name = name;
    }

    public void Display()
    {
        Console.WriteLine(Name);
    }
}
```

Абстрактные классы и члены классов

```
class Client : Person
{
    public int Sum { get; set; }    // сумма на счету

    public Client(string name, int sum)
        : base(name)
    {
        Sum = sum;
    }
}

class Employee : Person
{
    public string Position { get; set; } // должность

    public Employee(string name, string position)
        : base(name)
    {
        Position = position;
    }
}
```

Затем мы сможем использовать эти классы:

```
Client client = new Client("Tom", 500);
Employee employee = new Employee("Bob", "Apple");
client.Display();
employee.Display();
```

Или даже так:

```
Person client = new Client("Tom", 500);
Person employee = new Employee("Bob", "Операционист");
```

Но мы НЕ можем создать объект **Person**, используя конструктор класса **Person**:

```
Person person = new Person("Bill");
```

Однако несмотря на то, что напрямую мы не можем вызвать конструктор класса Person для создания объекта, тем не менее конструктор в абстрактных классах то же может играть важную роль, в частности, инициализировать некоторые общие для производных классов переменные и свойства, как в случае со свойством Name. И хотя в примере выше конструктор класса Person не вызывается, тем не менее производные классы Client и Employee могут обращаться к нему.

Абстрактные члены классов

Кроме обычных свойств и методов абстрактный класс может иметь абстрактные члены классов, которые определяются с помощью ключевого слова **abstract** и не имеют никакого функционала. В частности, абстрактными могут быть:

- Методы
- Свойства
- Индексаторы
- События

Абстрактные члены классов не должны иметь модификатор private. При этом производный класс обязан переопределить и реализовать все абстрактные методы и свойства, которые имеются в базовом абстрактном классе. При переопределении в производном классе такой метод или свойство также объявляются с модификатором **override** (как и при обычном переопределении виртуальных методов и свойств). Также следует учесть, что если класс имеет хотя бы один абстрактный метод (или абстрактные свойство, индексатор, событие), то этот класс должен быть определен как **абстрактный**.

Абстрактные члены также, как и виртуальные, являются частью полиморфного интерфейса. Но если в случае с виртуальными методами мы говорим, что класс-наследник наследует реализацию, то в случае с абстрактными методами наследуется интерфейс, представленный этими абстрактными методами.

Абстрактные классы и члены классов

Абстрактные методы

Например, сделаем в примере выше метод Display абстрактным:

```
abstract class Person
{
    public string Name { get; set; }

    public Person(string name)
    {
        Name = name;
    }

    public abstract void Display();
}

class Client : Person
{
    public int Sum { get; set; }    // сумма на счету

    public Client(string name, int sum)
: base(name)
    {
        Sum = sum;
    }
    public override void Display()
    {
        Console.WriteLine($"{Name} имеет счет на сумму {Sum}");
    }
}
```

Абстрактные классы и члены классов

```
class Employee : Person
{
    public string Position { get; set; } // должность

    public Employee(string name, string position)
: base(name)
    {
        Position = position;
    }

    public override void Display()
    {
        Console.WriteLine($"{Position} {Name}");
    }
}
```

Абстрактные классы и члены классов

Абстрактные свойства

Следует отметить использование абстрактных свойств. Их определение похоже на определение автосвойств. Например:

```
abstract class Person
{
    public abstract string Name { get; set; }
}

class Client : Person
{
    private string name;

    public override string Name
    {
        get { return "Mr/Ms. " + name; }
        set { name = value; }
    }
}

class Employee : Person
{
    public override string Name { get; set; }
}
```

В классе Person определено абстрактное свойство Name. Оно похоже на автосвойство, но это не автосвойство. Так как данное свойство не должно иметь реализацию, то оно имеет только пустые блоки get и set. В производных классах мы можем переопределить это свойство, сделав его полноценным свойством (как в классе Client), либо же сделав его автоматическим (как в классе Employee).

Абстрактные классы и члены классов

Отказ от реализации абстрактных членов

Производный класс обязан реализовать все абстрактные члены базового класса. Однако мы можем отказаться от реализации, но в этом случае производный класс также должен быть определен как абстрактный:

```
abstract class Person
{
    public abstract string Name { get; set; }
}

abstract class Manager : Person
{
}
```

Абстрактные классы и члены классов

Пример абстрактного класса

Хрестоматийным примером является система геометрических фигур. В реальности не существует геометрической фигуры как таковой. Есть круг, прямоугольник, квадрат, но просто фигуры нет. Однако же и круг, и прямоугольник имеют что-то общее и являются фигурами:

```
// абстрактный класс фигуры
abstract class Figure
{
    // абстрактный метод для получения периметра
    public abstract float Perimeter();
    // абстрактный метод для получения площади
    public abstract float Area();
}
// производный класс прямоугольника
class Rectangle : Figure
{
    public float Width { get; set; }
    public float Height { get; set; }

    public Rectangle(float width, float height)
    {
        this.Width = width;
        this.Height = height;
    }
    // переопределение получения периметра
    public override float Perimeter()
    {
        return Width * 2 + Height * 2;
    }
    // переопрелеление получения площади
    public override float Area()
    {
        return Width * Height;
    }
}
```