2 ГЛАВА Классы и объекты

Хотите обмануть мага? Боже, какая детская непосредственность. Я же вижу вас насквозь.

Из к/ф «31 июня»

С классами мы уже сталкивались — собственно, ни одна наша программа не обошлась без класса. Об объектах речь тоже уже шла. Но назвать это знакомством с классами и объектами было бы с нашей стороны несколько самонадеянно. Здесь мы постараемся систематизировать и привести в толк наши познания в данной области. Мы это сделаем, поскольку альтернативы у нас нет — без классов и объектов о программировании в C# можно и не мечтать.

Описание класса

Это экспонаты. Отходы, так сказать, магического производства.

Из к/ф «Чародеи»

Мы уже примерно представляем, что такое класс. Во всяком случае, классы мы использовали каждый раз, когда писали программу. Вместе с тем каждый раз у нас был лишь один класс, причем довольно специфический — для этого класса мы не создавали объекты. Главное его предназначение

состояло в том, что в классе описывался метод `Main()`, во многих отношениях отождествляемый с программой. В этой главе мы перейдем на качественно новый уровень программирования — наши программы будут содержать несколько классов. Также мы узнаем, как на основе классов создаются объекты — ведь в конечном счете именно для создания объектов нужен класс.

ПРИМЕЧАНИЕ Это не всегда так. Есть классы, которые представляют интерес сами по себе, без всяких там объектов. Обычно это классы со статическими методами, которые играют роль библиотеки.

В этой главе мы расширим свои познания в области создания классов. Классы могут быть самыми разными, но нас пока интересуют наиболее простые варианты. В ближайшем будущем мы будем использовать следующий шаблон для создания классов:

```
class имя_класса{
   public тип_поля имя_поля;
   public тип_результата имя_метода(аргументы){
   // код метода
   }
}
```

ПРИМЕЧАНИЕ Члены класса (в данном случае поля и методы) могут быть *закрытыми* и *открытыми*. Закрытые члены класса — это члены, которые класс приберегает «для себя», то есть для использования исключительно в пределах класса. Открытые члены класса можно использовать не только внутри класса, но и вне его. Именно такие члены класса пока что нас и будут интересовать.

Блок с кодом класса начинается ключевым словом `class`, после которого указывается имя класса, а тело класса заключается в фигурные скобки. Собственно класс — то, что находится в фигурных скобках. А в скобках может находиться описание полей и методов (и некоторых членов класса). Поля описываются как обычные переменные: указывается тип переменной и ее имя. Описание метода выполняется по такому шаблону:

- идентификатор типа результата — ключевое слово, которое определяет тип значения, возвращаемого методом в качестве результата;
- имя метода;

- в круглых скобках указывается список аргументов. Аргументы перечисляются через запятую, для каждого аргумента указывается тип. Если аргументов нет, скобки все равно есть — пустые;
- программный код метода (тело метода) заключается в фигурные скобки.

И поля, и методы описываются с ключевым словом `public`, что означает их доступность за пределами класса.

В качестве иллюстрации рассмотрим программный код с описанием класса, представленный в листинге 2.1.

Листинг 2.1. Класс с полем и двумя методами

```
class MyClass{
// Поле класса:
   public string name;
// Метод класса для присваивания "имени":
   public void SetName(string arg){
      // Присваиваем значение полю name:
      name=arg;
      // Отображаем сообщение об изменении
      // значения поля name:
      Console.WriteLine("Присвоено значение полю name.");
   }
// Метод класса для отображения "имени":
   public void ShowName(){
      // Отображаем сообщение со значением
      // поля name:
      Console.WriteLine("Значение поля name: "+name);
   }
}
```

Наш класс называется `MyClass`. У класса одно поле и два метода. Поле называется `name`, и это поле текстовое — оно объявляется как переменная типа `string`. Что касается методов, то оба они не возвращают результат. Поэтому в качестве идентификатора типа результата использовано ключевое слово `void`.

У метода `SetName()` один текстовый аргумент (объявлен как `string arg`). В теле метода командой `name=arg` полю `name` присваивается значение, как у аргумента `arg`. Затем командой `Console.WriteLine("Присвоено значение полю name.")` в консоль выводится сообщение с информацией о том, что значение поля `name` изменено.

У метода `ShowName()` аргументов нет. Единственной командой `Console.WriteLine("Значение поля name: "+name)` в теле метода отображается консольное сообщение с информацией о значении поля `name`.

ПРИМЕЧАНИЕ В методах почти массово используется обращение к полю name. Резонным образом возникает вопрос, о поле name какого объекта идет речь? Ведь у каждого объекта класса есть поле name. Поэтому сколько объектов класса, столько разных полей, и каждое называется name. Но проблемы здесь на самом деле нет — обращение выполняется к полю того объекта, из которого вызывается метод.

На этом описание класса заканчивается, и остается лишь проверить, какая от этого класса может быть польза. Нам предстоит несколько расширить программный код. Он будет содержать не только описание класса, но и инструкции по созданию объектов на основе этого класса.

Объектные переменные и создание объектов

> Очень убедительно. Мы подумаем, к кому это применить.
>
> *Из к/ф «31 июня»*

Нам предстоит еще одно усилие на пути изучения классов — мы будем создавать объекты. В C# процедура создания объекта (в широком смысле этого понятия) имеет некоторые особенности, если сравнивать, например, с созданием обычной переменной (не объекта). Условно процесс создания объекта можно разбить на два этапа:

- создание объектной переменной;
- создание объекта с присваиванием значения объектной переменной.

ВНИМАНИЕ Строго говоря, при создании объекта указывается не имя класса, а *конструктор* класса. Конструктор класса — это такой специальный метод. Одна из его особенностей состоит в том, что имя конструктора совпадает с именем метода. Даже если мы конструктор в классе не описывали, он все равно существует — это так называемый конструктор по умолчанию. У такого конструктора нет аргументов — отсюда и пустые круглые скобки после имени класса в инструкции создания объекта. Мы расставим все точки над *i* в вопросе создания объектов после того, как поближе познакомимся с конструкторами, интерфейсами и наследованием. Другими словами, вопрос этот не такой тривиальный, как может показаться на первый взгляд.

Объектная переменная создается абсолютно так же, как и «необъектная» переменная, с той лишь разницей, что в качестве идентификатора типа

указывается имя класса, для которого создается объектная переменная. Например, чтобы создать объектную переменную с именем `obj` для класса `MyClass`, можем воспользоваться командой `MyClass obj`. Однако объектная переменная — это еще не объект (хотя именно с помощью объектной переменной мы будем обращаться к объекту и выполнять с ним все основные операции). Для создания непосредственно объекта используем оператор `new`. Чтобы было понятно, какого класса объект создается, после оператора `new` указывается имя класса с пустыми круглыми скобками.

Например, для создания объекта класса `MyClass` можно воспользоваться инструкцией `new MyClass()`. Использование такой инструкции имеет два важных следствия:

- во-первых, создается объект класса `MyClass`;
- во-вторых, в качестве результата возвращается адрес этого объекта, или ссылка на объект.

Если есть результат, то его обычно куда-то записывают. Адреса объектов (ссылки на объект) записывают в объектные переменные (обычно класс объектной переменной должен совпадать с классом объекта). Собственно, объектная переменная и создается для того, чтобы в нее записать ссылку на объект. В этом смысле вполне логичными могли бы быть такие команды:

```
MyClass obj;
obj=new MyClass();
```

Благодаря этим двум инструкциям в наше распоряжение поступает объект класса `MyClass`, доступ к которому мы имеем через переменную `obj`. В дальнейшем, если это не будет приводить к недоразумениям, под объектом мы будем подразумевать как раз объектную переменную.

ПРИМЕЧАНИЕ Инструкции по созданию объектной переменной и объекта можно объединить в одну — совместить объявление объектной переменной и создание объекта. Так, альтернативой командам MyClass obj и obj=new MyClass() может быть одна-единственная команда MyClass obj=new MyClass().

Теперь мы практически готовы к тому, чтобы применить объекты на практике. Расправим наши крылья, воспользовавшись программным кодом, представленным в листинге 2.2.

Листинг 2.2. Название листинга

```
using System;
class MyClass{
// Поле класса:
   public string name;
```

продолжение ⇩

Листинг 2.2 (*продолжение*)

```
// Метод класса для присваивания "имени":
   public void SetName(string arg){
      // Присваиваем значение полю name:
      name=arg;
      // Отображаем сообщение об изменении
      // значения поля name:
      Console.WriteLine("Присвоено значение полю name.");
   }
// Метод класса для отображения "имени":
   public void ShowName(){
      // Отображаем сообщение со значением поля name:
      Console.WriteLine("Значение поля name: "+name);
   }
}
// Класс с методом Main():
class ObjDemo{
   // Главный метод программы:
   public static void Main(){
        // Объектная переменная класса MyClass:
        MyClass cat;
        // Создание объекта класса MyClass:
        cat=new MyClass();
        // Создание объекта и переменной класса MyClass:
        MyClass dog=new MyClass();
        // Полю name объекта cat присваивается значение:
        cat.name="Мурчик";
        // Полю name объекта dog присваивается значение:
        dog.SetName("Шарик");
        // Отображается значение поля name объекта cat:
        cat.ShowName();
        // Отображается значение поля name объекта dog:
        dog.ShowName();
        Console.ReadLine();
   }
}
```

ВНИМАНИЕ Метод Main() мы описали с атрибутом public. Здесь мы последовали общей рекомендации: описывать главный метод программы как открытый. Вместе с тем и без этого атрибута программа будет работать.

Это полный программный код, в котором, помимо уже знакомого нам класса `MyClass`, есть еще один класс, `ObjDemo`, в котором описан метод

Main(). В этом методе, в свою очередь, создаются и используются объекты класса MyClass. Поскольку программный код класса MyClass мы уже анализировали, остановим свое внимание на программном коде метода Main(). Вкратце сюжет пьесы такой. Создается два объекта cat и dog класса MyClass. Полям name этих объектов присваиваются значения, после чего значения этих полей выводятся в консоль. Объект cat создается в два этапа. Сначала командой MyClass cat объявляется объектная переменная класса MyClass. Непосредственно создание объекта класса MyClass и присваивание ссылки на этот объект переменной cat выполняется командой cat=new MyClass(). Создание второго объекта выполняется с помощью команды MyClass dog=new MyClass(). Здесь и объектная переменная создается, и объект, и ссылка на объект присваивается объектной переменной.

На следующем витке эволюции полям новоиспеченных объектов присваиваются значения. Для объекта cat мы используем простую прямую команду cat.name="Мурчик". Здесь мы встречаемся с примером точечного синтаксиса. Это классика жанра — для ссылки на поле name объекта cat мы указываем имя объекта и, через точку, имя поля. Присваиваемое полю значение указано справа от оператора присваивания. По-другому мы поступаем с объектом dog. Для этого командой dog.SetName("Шарик") из объекта dog вызывается метод SetName() с текстовым аргументом, который присваивается полю name этого объекта. Здесь мы также имеем дело с точечным синтаксисом.

ПРИМЕЧАНИЕ Обратите внимание на то, что в соответствии с кодом метода SetName() в консоль выводится сообщение о присвоении значения полю name.

Кульминацией программы являются команды cat.ShowName() и dog.ShowName(), которыми в консоль выводятся сообщения о значении полей name соответствующих объектов.

ВНИМАНИЕ Напоминаем, что команда Console.ReadLine() нужна исключительно для того, чтобы окно консоли не закрылось сразу по выполнении предыдущих инструкций.

Результат выполнения программы представлен на рис. 2.1, где показано консольное окно с сообщениями, которые увидит пользователь.

Это общая схема создания и использования объектов и объектных переменных. Теперь мы будем постепенно усовершенствовать методы работы с объектами.

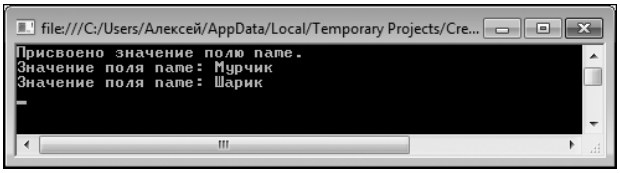

Рис. 2.1. Результат выполнения программы, в которой использованы объекты

Перегрузка методов

Нет, такой хоккей нам не нужен!

Н. Озеров

Перегрузка методов — весьма полезный и перспективный механизм, который позволяет создавать очень гибкие и эффективные методы. В общих чертах суть перегрузки методов состоит в том, что в классе можно создавать (описывать) несколько вариантов одного и того же метода. «Несколько вариантов» в данном случае означает, что все эти методы имеют одинаковые названия, но при этом различаются количеством и (или) типом аргументов.

ПРИМЕЧАНИЕ Процедура перегрузки методов есть не только в C#, но и в C++ и Java. Во всех случаях общий подход универсален — у перегружаемых методов одинаковые названия, но при этом разные варианты методов должны быть различимы. В принципе идентификацию того или иного варианта метода (поскольку у всех у них одинаковые названия) можно выполнять на основе списка аргументов и (или) типа результата. В C# такая идентификация выполняется только на основе списка аргументов метода. У разных версий перегружаемого метода должно быть разное количество аргументов или аргументы должны быть разного типа. Обычно правильная фраза звучит так: «при перегрузке метода неизменно название, но разная *сигнатура*». Под сигнатурой в C# подразумевают имя метода и список его аргументов. Обратите внимание: тип результата в понятие «сигнатура» не входит!

Реализуется перегрузка метода достаточно просто. Каждый вариант метода описывается как отдельный метод. Важно только помнить, что разные варианты должны быть различимы на уровне аргументов. Ведь количество и тип переданных методу аргументов являются индикаторами того, какой вариант метода необходимо вызывать в том или ином случае. Проиллюстрируем это на конкретном примере. Обратимся к программному коду, представленному в листинге 2.3.

Листинг 2.3. Перегрузка методов

```
using System;
class Person{
   // Закрытое числовое поле:
   int age;
   // Закрытое текстовое поле:
   string name;
   // Открытый метод для отображения полей:
   public void show(){
      Console.WriteLine("Имя: "+name);
      Console.WriteLine("Возраст: "+age);
   }
   // Открытый перегруженный метод для
   // присваивания значения полям.
   // Версия перегруженного метода
   // с двумя аргументами:
   public void set(int n,string arg){
      age=n;
      name=arg;
   }
   // Версия метода без аргументов:
   public void set(){
      age=0;
      name="Нет имени";
   }
   // Версия метода с одним числовым аргументом:
   public void set(int n){
      // Вызывается версия метода с двумя аргументами:
      set(n,"Нет имени");
   }
   // Версия метода с одним текстовым аргументом:
   public void set(string arg){
      // Вызывается версия метода
      // с двумя аргументами:
      set(0,arg);
   }
}
class PersonDemo{
   // Главный метод программы:
   public static void Main(){
      // Создание объекта fellow класса Person:
      Person fellow=new Person();
      // Вызов версии метода set() с одним
      // числовым аргументом:
      fellow.set(100);
```

продолжение ⇨

Листинг 2.3 (*продолжение*)

```
        // Отображение результата:
        fellow.show();
        // Вызов версии метода set() с одним
        // текстовым аргументом:
        fellow.set("Колобок");
        // Отображение результата:
        fellow.show();
        // Вызов версии метода set() с двумя аргументами:
        fellow.set(10,"Буратино");
        // Отображение результата:
        fellow.show();
        // Вызов версии метода set() без аргументов:
        fellow.set();
        // Отображение результата:
        fellow.show();
        Console.ReadLine();
    }
}
```

Перегруженный метод находим в классе `Person`. У класса два поля (целочисленное `age` и текстовое `string`) и два метода (`show()` и `set()`) — правда, один из этих методов (метод `set()`) перегружается. Для этого метода описано четыре различных версии: с двумя аргументами, без аргументов, с одним текстовым аргументом и одним целочисленным аргументом.

ПРИМЕЧАНИЕ Поля у класса тоже не очень простые. Они описаны без ключевого слова public. Такие поля являются закрытыми и недоступны вне класса. Поэтому в программном коде класса эти поля можно использовать, а вот обратиться напрямую к полям вне класса не получится. Например, в главном методе программы создается объект fellow класса Person. И хотя у объекта fellow есть поля name и age, использовать инструкцию вида fellow.name или fellow.age не получится.

С методом `show()` все просто — он нужен для отображения значений полей `name` и `age` объекта, из которого вызывается метод. Нас интересует метод `set()`. С помощью метода задаются значения полей `name` и `age`. Мы перегружаем метод для того, чтобы можно было по-разному задавать значения полей объекта. Естественным представляется вариант, когда мы указываем в качестве аргументов метода `set()` значения, которые присваиваются полям объекта. В этом случае первый, числовой, аргумент определяет значение поля `age`, а второй, текстовый, аргумент задает значение поля `name`.