4 Классы, объекты и методы

4.1. Введение

Эта глава начинается с объяснения концепции классов на реальном примере. Затем будут рассмотрены пять приложений, которые демонстрируют принципы создания и использования классов в приложениях. Первые четыре приложения открывают учебный проект по разработке класса для хранения информации об оценках. Последний пример познакомит читателя с типом `decimal` на примере класса, предназначенного для представления баланса банковского счета.

4.2. Классы, объекты, методы, свойства и переменные экземпляров

Начнем с простой аналогии, которая поможет вам понять суть классов. Допустим, вы ведете машину и хотите увеличить скорость, нажав педаль газа. Что должно произойти перед тем, как вы сможете это сделать? Для начала кто-то должен спроектировать вашу машину. Как правило, проектирование начинается с технических чертежей, в которых присутствует педаль газа. Педаль скрывает от водителя сложные механизмы, которые заставляют машину двигаться быстрее, — подобно тому, как педаль тормоза скрывает механизмы, замедляющие движение, а руль скрывает механизмы поворота. Все это позволяет легко управлять машиной человеку, не имеющему представления о работе двигателя, тормозов и поворотных механизмов.

К сожалению, вести технический чертеж невозможно. Прежде чем вы сможете сесть за руль, машину необходимо построить по техническому чертежу. В готовой машине будет присутствовать педаль, ускоряющая ее движение, но даже этого недостаточно — машина не будет ускоряться сама по себе, на педаль газа должен нажать водитель.

Методы

Пример с машиной поможет нам представить некоторые ключевые концепции программирования. Все операции в приложениях выполняются методами. По сути

метод описывает механизмы, которые непосредственно реализуют нужные операции. Метод скрывает от пользователя сложные низкоуровневые подробности — подобно тому, как педаль газа скрывает от водителя сложные механизмы, которые заставляют машину двигаться быстрее.

Классы

В языке C# все начинается с создания структурных элементов приложения, называемых *классами*. Классы (среди прочего) содержат методы подобно тому, как чертежи машины содержат (среди прочего) техническое описание педали газа. В класс включаются методы (один или несколько), предназначенные для выполнения операций класса. Например, класс, представляющий банковский счет, может содержать один метод для внесения денег на счет, другой метод для снятия денег со счета и третий метод для запроса текущего баланса.

Объекты

Класс — это всего лишь описание, и непосредственно использовать его невозможно (подобно тому, как невозможно вести технический чертеж машины). Прежде чем приложение сможет выполнять операции, описанные классом, необходимо сначала создать объект этого класса. Это одна из причин, по которым C# называется объектно-ориентированным языком программирования.

Вызовы методов

Когда вы управляете машиной, нажатие педали газа отправляет машине сообщение, приказывающее выполнить операцию — ускорить движение. Аналогичным образом отправляются сообщения объекту; каждое сообщение, называемое *вызовом метода*, приказывает вызванному методу объекта выполнить свою операцию.

Атрибуты

Кроме поддерживаемых операций, машина обладает многочисленными атрибутами: цвет, количество дверей, количество бензина в баке, текущая скорость и пробег. Эти атрибуты представляются в технических чертежах машины наряду с ее основными функциями. Во время ведения машины эти атрибуты всегда связываются с машиной. Каждая машина поддерживает собственный набор атрибутов. Например, каждая машина знает, сколько бензина находится в ее топливном баке, но ничего не знает об уровне бензина в других машинах. Аналогичным образом с каждым объектом связывается набор атрибутов, который сопровождает объект в процессе его использования приложением. Атрибуты определяются как часть класса объекта. Например, объект банковского счета может обладать атрибутом, представляющим текущий баланс. Каждый объект знает баланс того счета, который он представляет, но не балансы других счетов в банке. Атрибуты определяются в виде *переменных экземпляров* класса.

Свойства, get- и set-методы

Следует учитывать, что атрибуты не всегда доступны напрямую. В конце концов, никакой банк не захочет пускать клиентов в хранилище, чтобы те могли пересчитать

деньги на своем счету. Вместо этого клиент обращается к кассиру или получает информацию на своей странице в Интернете. Этот принцип распространяется и на объекты: чтобы использовать переменные экземпляра, не обязательно иметь прямой доступ к ним — можно воспользоваться *свойствами* объекта. Свойства содержат get-методы для чтения значений переменных и set-методы для присваивания значений.

4.3. Объявление класса с методом и создание экземпляра класса

Начнем с примера, состоящего из классов `GradeBook` (ил. 4.1) и `GradeBookTest` (ил. 4.2). Класс `GradeBook` (объявленный в файле GradeBook.cs) выводит на экран приветствие, а класс `GradeBookTest` (объявленный в файле GradeBookTest.cs) используется для создания и выполнения операций с объектом класса `GradeBook`. Как принято, мы объявляем классы `GradeBook` и `GradeBookTest` в разных файлах, чтобы имя каждого файла соответствовало имени содержащегося в нем класса.

Выполните команду File ▸ New Project..., чтобы открыть диалоговое окно New Project, и выберите шаблон GradeBook Console Application. Переименуйте файл Program.cs в GradeBook.cs. Удалите весь код, сгенерированный IDE, и замените его кодом на ил. 4.1.

```
1  // Ил. 4.1: GradeBook.cs
2  // Объявление класса с одним методом
3  using System;
4
5  public class GradeBook
6  {
7     // Вывод приветствия для пользователя GradeBook
8     public void DisplayMessage()
9     {
10       Console.WriteLine( "Welcome to the Grade Book!" );
11    } // Конец метода DisplayMessage
12 } // Конец класса GradeBook
```

Ил. 4.1. Объявление класса с одним методом

Класс GradeBook

Объявление класса `GradeBook` (см. ил. 4.1) содержит метод `DisplayMessage` (строки 8–11) для вывода сообщения на экран. Сообщение выводится в строке 10. Вспомните, что класс ранее сравнивался с техническим чертежом — чтобы вывести сообщение, необходимо создать объект этого класса и вызвать его метод; это делается в листинге на ил. 4.2.

Объявление класса начинается в строке 5. Ключевое слово `public` является модификатором доступа. *Модификаторы доступа* определяют доступность свойств и методов объекта для других методов приложения. Пока мы будем просто объявлять все классы *открытыми*, то есть общедоступными. Каждое объявление класса

содержит ключевое слово `class`, за которым следует имя класса. Тело каждого класса заключается в пару фигурных скобок (`{` и `}`), как в строках 6 и 12 класса `GradeBook`.

Объявление метода DisplayMessage

В главе 3 каждый объявляемый класс содержит единственный метод с именем `Main`. Класс `GradeBook` также содержит один метод — `DisplayMessage` (строки 8–11). Напомним, что специальный метод `Main` всегда вызывается автоматически при запуске приложения. Большинство методов автоматически не вызывается. Как вы вскоре увидите, чтобы метод `DisplayMessage` выполнил свою операцию, его необходимо явно вызвать в программе.

Объявление метода начинается с ключевого слова `public`, которое указывает, что метод является открытым — то есть может вызываться за пределами тела объявления класса методами других классов. Ключевое слово `void` (тип возвращаемого значения) означает, что метод после завершения не возвращает никакой информации тому методу, из которого он был вызван. Если в методе указан тип возвращаемого значения, отличный от `void`, то после вызова метода и выполнения операции он возвращает результат заданного типа. Например, когда вы подходите к банкомату и запрашиваете баланс своего счета, банкомат возвращает числовое значение. Если метод `Square` возвращает квадрат своего аргумента, то в команде

```
int result = Square( 2 );
```

вызов `Square` должен вернуть значение 4, которое будет присвоено переменной `result`.

Мы уже использовали методы, возвращающие информацию, — например, в главе 3 метод `ReadLine` класса `Console` применялся для получения строки, введенной пользователем с клавиатуры. Метод `ReadLine` возвращает введенное значение для последующего использования в приложении.

Имя метода

За типом возвращаемого значения указывается имя метода `DisplayMessage` (строка 8). Как правило, именами методов являются глаголы или глагольные конструкции, а именами классов — существительные. По общепринятой схеме имена методов начинаются с прописной буквы, а все последующие слова в имени также начинаются с прописных букв. Круглые скобки после имени метода указывают, что определяется именно метод, а не что-то другое. Пустая пара круглых скобок (строка 8) означает, что метод не требует дополнительной информации для выполнения своей операции. Строка 8 обычно называется *заголовком метода*. Тело каждого метода заключается в фигурные скобки, как показано в строках 9 и 11.

Тело метода

Тело метода содержит команды, выполняющие операцию метода. В нашем случае метод содержит всего одну команду (строка 10), которая выводит в консольном окне приветствие `"Welcome to the Grade Book!"` и символ новой строки. После выполнения команды операция метода считается выполненной.

Использование класса GradeBook

Следующий шаг — использование класса `GradeBook` в приложении. Как вы узнали из главы 3, выполнение каждого приложения начинается с метода `Main`. Класс `GradeBook` не может быть основным классом приложения, потому что в нем нет метода с именем `Main`. В главе 3 такой проблемы не было, потому что каждый объявляемый класс содержал метод `Main`. Следовательно, нам придется либо объявить отдельный класс, содержащий метод `Main`, либо добавить метод `Main` в класс `GradeBook`.

Чтобы подготовить вас к более масштабным приложениям, которые встретятся вам в этой книге и в реальной работе, для тестирования классов этой главы будет использоваться отдельный класс (`GradeBookTest` в данном случае), содержащий метод `Main`.

Добавление класса в проект Visual C#

Для каждого примера этой главы мы добавим в консольное приложение новый класс. Щелкните правой кнопкой мыши на имени проекта в окне Solution Explorer и выберите в контекстном меню команду Add ▸ New Item.... В открывшемся диалоговом окне Add New Item выберите вариант Code File, введите имя нового файла (GradeBookTest.cs) и щелкните на кнопке Add. В проект добавляется новый пустой файл. Включите в него код с ил. 4.2.

```
 1 // ил. 4.2: GradeBookTest.cs
 2 // Создание объекта GradeBook и вызов его метода DisplayMessage.
 3 public class GradeBookTest
 4 {
 5    // Метод Main начинает выполнение программы
 6    public static void Main( string[] args )
 7    {
 8       // Создание объекта GradeBook и присваивание его myGradeBook
 9       GradeBook myGradeBook = new GradeBook();
10
11      // Вызов метода DisplayMessage объекта myGradeBook
12      myGradeBook.DisplayMessage();
13   } // Конец Main
14 } // Конец класса GradeBookTest
```

```
Welcome to the Grade Book!
```

Ил. 4.2. Создание объекта GradeBook и вызов его метода DisplayMessage

Класс GradeBookTest

Объявление класса `GradeBookTest` (см. ил. 4.2) содержит метод `Main`, управляющий работой приложения. Любой класс, содержащий метод `Main` (строка 6), может использоваться для выполнения приложения. Объявление класса начинается в строке 3 и заканчивается в строке 14. Класс содержит только метод `Main`; это типично для многих классов, которые просто начинают выполнение приложения.

Метод Main

В строках 6–13 объявляется метод `Main`. Чтобы метод `Main` мог начать выполнение приложения, в объявлении должно присутствовать ключевое слово `static` (строка 6),

которое указывает, что метод `Main` является статическим. Статические методы отличаются тем, что они могут вызываться без предварительного создания объекта класса (в нашем случае `GradeBookTest`), в котором объявлен метод. Статические методы более подробно рассматриваются в главе 7.

Создание объекта GradeBook

В нашем приложении требуется вызвать метод `DisplayMessage` класса `GradeBook` для вывода приветствия в консольном окне. В общем случае вы не сможете вызвать метод, принадлежащий другому классу, до создания объекта этого класса (строка 9). Мы начинаем с объявления переменной `myGradeBook`. Переменная объявляется с типом `GradeBook` — класс, объявленный в листинге на ил. 4.1. Каждый новый класс, созданный вами, становится новым типом в C#, который может использоваться для объявления переменных и создания объектов. Новые классы становятся доступными для всех классов того же проекта.

Переменная `myGradeBook` (строка 9) инициализируется результатом выражения создания объекта `new GradeBook()`. Оператор `new` создает новый объект класса, указанного справа от ключевого слова (то есть `GradeBook`). Круглые скобки справа от `GradeBook` обязательны. Как будет показано в разделе 4.10, круглые скобки в сочетании с именем класса представляют вызов конструктора — метода, используемого только в момент создания объекта для инициализации его данных. Далее вы увидите, что в круглые скобки можно заключить данные, определяющие исходные значения данных объекта. Пока мы оставим круглые скобки пустыми.

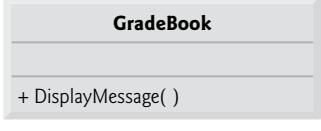
Вызов метода DisplayMessage объекта GradeBook

Теперь переменная `myGradeBook` может использоваться для вызова метода `DisplayMessage`. В строке 12 вызывается метод `DisplayMessage` (строки 8–11 на ил. 4.1): вызов состоит из имени переменной `myGradeBook`, оператора «точка» (`.`), имени метода `DisplayMessage` и пустой пары круглых скобок. Этот вызов метода отличается от вызовов методов из главы 3, выводивших информацию в консольном окне, — тогда во всех случаях передавались аргументы с выводимыми данными. В начале строки 12 (ил. 4.2) префикс «`myGradeBook.`» указывает, что метод `Main` должен использовать объект `GradeBook`, созданный в строке 9. Пустые круглые скобки в строке 8 на ил. 4.1 означают, что методу `DisplayMessage` для выполнения его операции не требуется никакой дополнительной информации. По этой причине при вызове метода (строка 12 на ил. 4.2) за именем метода вставляется пустая пара круглых скобок, которая означает, что методу `DisplayMessage` не передаются аргументы. Когда метод `DisplayMessage` завершит свою операцию, метод `Main` продолжает выполняться в строке 13. Здесь метод `Main` заканчивается, а выполнение приложения завершается.

Диаграмма классов UML для класса GradeBook

На ил. 4.3 представлена диаграмма классов UML для класса `GradeBook` на ил. 4.1. Графический язык UML используется программистами для стандартизированного представления объектно-ориентированных систем. В UML каждый класс на

диаграмме классов представляется прямоугольником, состоящим из трех отделений. Верхнее отделение содержит имя класса, выровненное по центру и выводимое жирным шрифтом. Среднее отделение содержит атрибуты класса, соответствующие переменным экземпляров и свойствам C#. На ил. 4.3 среднее отделение пусто, потому что версия класса `GradeBook` на ил. 4.1 не имеет атрибутов. В нижнем отделении перечисляются операции класса, соответствующие методам в C#. В UML за именем операции ставится пара круглых скобок. Класс `GradeBook` содержит всего один метод `DisplayMessage`, поэтому в нижнем отделении ил. 4.3 указана одна операция с этим именем. Методу `DisplayMessage` для выполнения его операции не нужна никакая дополнительная информация, поэтому за именем `DisplayMessage` на диаграмме классов следуют пустые круглые скобки (как и в объявлении метода в строке 8 на ил. 4.1). Знак «плюс» (`+`) перед именем операции сообщает, что `DisplayMessage` является открытой операцией в UML (то есть открытым методом в C#). Диаграммы классов UML часто используются для представления краткой информации об атрибутах и операциях классов.



Ил. 4.3. Диаграмма классов UML показывает, что класс GradeBook содержит открытую операцию DisplayMessage

4.4. Объявление метода с параметром

В нашей аналогии с машинами из раздела 4.2 упоминалось, что нажатие педали газа отправляет машине сообщение о выполнении операции — повышении скорости. Но насколько должна повыситься скорость? Чем дольше вы удерживаете педаль, тем больше прирост скорости. Таким образом, сообщение должно включать как выполняемую операцию, так и дополнительную информацию, используемую для выполнения этой операции. Дополнительная информация называется *параметром* — значение параметра помогает машине определить величину ускорения. Аналогичным образом метод может потребовать передачи одного или нескольких параметров, представляющих дополнительную информацию для выполнения его операции. При вызове метода для каждого из параметров передаются значения, называемые *аргументами*. Например, методу `Console.WriteLine` должен передаваться аргумент с данными, выводимыми в консольном окне. Аналогичным образом для внесения средств на банковский счет метод `Deposit` определяет параметр, представляющий вносимую сумму. При вызове метода `Deposit` параметру метода присваивается значение переданного аргумента. Метод зачисляет эту сумму на счет, увеличивая баланс счета.

В нашем следующем примере объявляется класс `GradeBook` (ил. 4.4) с методом `DisplayMessage`, который выводит в приветствии название учебного курса (пример

выполнения приведен на ил. 4.5). Название курса передается в параметре нового метода `DisplayMessage`.

```
// Ил. 4.4: GradeBook.cs
// Объявление класса с методом, получающим параметр.
using System;

public class GradeBook
{
   // Вывод приветствия для пользователя GradeBook
   public void DisplayMessage( string courseName )
   {
     Console.WriteLine( "Welcome to the grade book for\n{0}!",
        courseName );
   } // Конец метода DisplayMessage
} // Конец класса GradeBook
```

Ил. 4.4. Объявление класса с методом, получающим параметр

```
// Ил. 4.5: GradeBookTest.cs
// Создание объекта GradeBook и передача строки
// его методу DisplayMessage.
using System;

public class GradeBookTest
{
   // Метод Main начинает выполнение программы
   public static void Main( string[] args )
   {
      // Создание объекта GradeBook и присваивание его myGradeBook
      GradeBook myGradeBook = new GradeBook();

      // Запрос названия учебного курса
      Console.WriteLine( "Please enter the course name:" );
      string nameOfCourse = Console.ReadLine(); // Чтение строки текста
      Console.WriteLine(); // Вывод пустой строки

      // Вызов метода DisplayMessage объекта myGradeBook
      // и передача nameOfCourse в аргументе
      myGradeBook.DisplayMessage( nameOfCourse );
   } // Конец Main
} // Конец класса GradeBookTest
```

```
Please enter the course name:
CS101 Introduction to C# Programming

Welcome to the grade book for
CS101 Introduction to C# Programming!
```

Ил. 4.5. Создание объекта GradeBook и вызов его метода DisplayMessage с передачей строки

Прежде чем браться за рассмотрение новых возможностей класса `GradeBook`, давайте посмотрим, как этот класс используется из метода `Main` класса `GradeBookTest` (см. ил. 4.5). Строка 12 создает объект класса `GradeBook` и присваивает его переменной

myGradeBook. Строка 15 предлагает пользователю ввести название учебного курса. Строка 16 читает название, введенное пользователем, и присваивает его переменной nameOfCourse; для ввода данных используется метод ReadLine класса Console. Пользователь вводит название курса и нажимает клавишу Enter, чтобы передать название приложению. Клавиша Enter вставляет в конец введенной последовательности символ новой строки. Метод ReadLine читает введенные символы до символа новой строки и возвращает строку с предшествующими символами (не включая символ новой строки, который отбрасывается).

В строке 21 вызывается метод DisplayMessage объекта myGradeBook. Имя nameOfCourse в круглых скобках — аргумент, передаваемый методу DisplayMessage для выполнения его операции. Значение переменной nameOfCourse в Main становится значением метода параметра courseName метода DisplayMessage (строка 8 на ил. 4.4). При выполнении приложения метод DisplayMessage включает в приветствие введенное название курса (см. ил. 4.5).

АРХИТЕКТУРНОЕ РЕШЕНИЕ 4.1

Обычно объекты создаются оператором new. Исключение составляют строковые литералы, заключенные в кавычки, — например, "hello". Строковые литералы неявно создаются C# тогда, когда они впервые встречаются в коде.

Подробнее об аргументах и параметрах

При объявлении метода необходимо указать в его объявлении, нужны ли методу дополнительные данные для выполнения его операции. Дополнительная информация включается в список параметров в круглых скобках за именем метода. Список может содержать любое количество параметров (или не содержать ни одного). Каждый параметр объявляется как переменная, то есть с указанием типа и идентификатора. В нашем случае тип string и идентификатор courseName сообщают, что методу DisplayMessage для выполнения его операции требуется строковое значение. В момент вызова значение аргумента, указанное при вызове, присваивается соответствующему параметру (courseName в данном случае) в заголовке метода. Затем тело метода использует параметр courseName для обращения к значению. В строках 10–11 на ил. 4.4 значение параметра courseName выводится с использованием форматного элемента {0} в первом аргументе WriteLine. Имя переменной-параметра (см. ил. 4.4, строка 8) может совпадать с именем переменной-аргумента или отличаться от него (см. ил. 4.5, строка 21).

Метод может объявить несколько параметров; в этом случае каждое объявление параметра отделяется от следующего запятой. Количество аргументов при вызове метода должно соответствовать количеству обязательных параметров в списке параметров из объявления вызываемого метода. Кроме того, типы аргументов в вызове метода должны быть совместимы с типами соответствующих параметров в объявлении метода. (Как вы узнаете в следующих главах, тип аргумента и тип соответствующего параметра не обязаны точно совпадать.) В нашем примере вызов метода передает один аргумент типа string (nameOfCourse объявляется с типом

string в строке 16 на ил. 4.5); в объявлении метода указан один параметр типа string (строка 8 на ил. 4.4). Таким образом, тип аргумента при вызове метода в данном случае точно совпадает с типом параметра в заголовке метода.

ТИПИЧНАЯ ОШИБКА 4.1

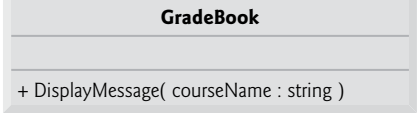
Если количество аргументов при вызове метода не соответствует количеству обязательных параметров в объявлении метода, происходит ошибка компиляции.

ТИПИЧНАЯ ОШИБКА 4.2

Если типы аргументов при вызове метода несовместимы с типами соответствующих параметров в объявлении метода, происходит ошибка компиляции.

Обновленная диаграмма классов UML для класса GradeBook

Диаграмма классов UML на ил. 4.6 моделирует класс GradeBook на ил. 4.4. Как и в листинге на ил. 4.4, этот класс GradeBook содержит открытую операцию DisplayMessage, однако данная версия DisplayMessage имеет параметр. В UML моделирование параметров несколько отличается от C#: в круглых скобках за именем операции сначала указывается имя параметра, за ним следует двоеточие и тип параметра. В UML поддерживаются некоторые типы данных, сходные с типами C#. Например, типы UML String и Integer соответствуют типам C# string и int. К сожалению, не все типы C# имеют аналоги в UML. По этой причине, а также для предотвращения путаницы между типами UML и типами C# мы используем на диаграммах классов UML только типы C#. Метод DisplayMessage класса Gradebook (см. ил. 4.4) имеет строковый параметр с именем courseName, поэтому на ил. 4.6 в круглых скобках после имени DisplayMessage указывается параметр courseName : string.



Ил. 4.6. Диаграмма классов UML показывает, что класс GradeBook содержит открытую операцию DisplayMessage с параметром courseName типа string

Директивы using

Обратите внимание на директиву using на ил. 4.5 (строка 4). Она сообщает компилятору, что приложение использует классы из пространства имен System — такие, как класс Console. Почему директива using необходима для использования класса Console, но не для класса GradeBook? Классы, откомпилированные в одном проекте (такие, как GradeBook и GradeBookTest), связаны особыми отношениями. По умолчанию считается, что такие классы принадлежат одному пространству имен. Директива using не обязательна, когда класс использует другой класс из того же пространства имен, — например, когда класс GradeBookTest использует класс

GradeBook. Для простоты в наших примерах из этой книги пространства имен не объявляются. Любые классы, не включенные в конкретное пространство имен, неявно помещаются в так называемое *глобальное пространство имен*.

Вообще говоря, директива using в строке 4 не обязательна, если мы всегда будем ссылаться на класс Console в виде System.Console, то есть с указанием пространства имен и имени класса. Такие имена классов называются *полными* (fully qualified). Например, строка 15 может быть записана в виде

```
System.Console.WriteLine( "Please enter the course name:" );
```

Большинство программистов C# считают, что полные имена неудобны, и предпочитают использовать директивы using.

4.5. Переменные экземпляров и свойства

В главе 3 все переменные приложения объявлялись в методе Main. Переменные, объявленные в теле метода, называются *локальными переменными* и могут использоваться только в этом методе. После завершения метода значения его локальных переменных теряются. Вспомните, о чем говорилось в разделе 4.2: объект обладает атрибутами, которые сопровождают его при использовании в приложении. Такие атрибуты существуют до вызова метода объекта и продолжают существовать после того, как метод будет выполнен.

В объявлениях классов атрибуты представляются переменными. Такие переменные называются *полями* и объявляются в объявлении класса, но за пределами объявлений методов этого класса. Когда каждый объект класса поддерживает собственную копию атрибута, поле, представляющее атрибут, также называется *переменной экземпляра* — каждый объект (экземпляр) класса имеет собственную копию переменной. В главе 10 будет рассмотрена другая разновидность полей: статические переменные, совместно используемые всеми объектами одного класса.

Класс обычно содержит одно или несколько свойств для работы с атрибутами, принадлежащими конкретному объекту класса. В примере, приведенном в этом разделе, представлен класс GradeBook с переменной экземпляра courseName, представляющей название учебного курса конкретного объекта GradeBook, а также свойством CourseName для работы с courseName.

Класс GradeBook с переменной экземпляра и свойством

В следующем приложении (ил. 4.7–4.8) класс GradeBook (см. ил. 4.7) хранит название курса в переменной экземпляра, чтобы его можно было использовать в любой момент во время выполнения приложения. Класс также содержит один метод DisplayMessage (строки 24–30) и одно свойство CourseName (строки 11–21). Вспомните, о чем говорилось в главе 2: свойства предназначены для манипуляций с атрибутами объекта. Например, в этой главе мы использовали свойство Text элемента управления Label для определения текста надписи. На этот раз свойство используется в коде, а не в окне

свойств IDE. Для этого мы сначала объявляем свойство членом класса `GradeBook`. Как вы вскоре увидите, свойство `CourseName` класса `GradeBook` может использоваться для хранения названия курса в `GradeBook` (в переменной экземпляра `courseName`) или чтения названия курса `GradeBook` (из переменной экземпляра `courseName`).

Метод `DisplayMessage`, вызываемый без параметров, выводит приветствие с названием курса. Однако в этой реализации метод читает название курса из переменной экземпляра `courseName` с использованием свойства `CourseName`.

```
 1  // Ил. 4.7: GradeBook.cs
 2  // Класс GradeBook с закрытой переменной экземпляра courseName
 3  // и открытым свойством для чтения и записи его значения.
 4  using System;
 5
 6  public class GradeBook
 7  {
 8     private string courseName; // Название курса GradeBook
 9
10     // Свойство для чтения и записи названия курса
11     public string CourseName
12     {
13        get
14        {
15           return courseName;
16        } // Конец get
17        set
18        {
19           courseName = value;
20        }
21     } // Конец свойства CourseName
22
23     // Вывод приветствия для пользователя GradeBook
24     public void DisplayMessage()
25     {
26        // Использование свойства CourseName для чтения
27        // названия курса из объекта GradeBook
28        Console.WriteLine( "Welcome to the grade book for\n{0}!",
29           CourseName ); // Вывод свойства CourseName
30     } // Конец метода DisplayMessage
31 } // Конец класса GradeBook
```

Ил. 4.7. Класс GradeBook содержит закрытую переменную courseName и открытое свойство для чтения и записи ее значения

Преподаватель обычно ведет несколько учебных курсов с разными названиями. В строке 8 объявляется переменная `courseName` с типом `string`. Строка 8 объявляет переменную экземпляра, потому что переменная объявляется в теле класса (строки 7–31), но вне тел метода класса (строки 24–30) и свойства (строки 11–21). Каждый экземпляр (то есть объект) класса `GradeBook` содержит одну копию каждой переменной экземпляра. Если создать два объекта `GradeBook`, то каждый объект будет содержать свою копию `courseName`. Все методы и свойства класса `GradeBook` могут напрямую обращаться к переменной экземпляра `courseName`, но для работы с переменными экземпляров лучше использовать свойства (как это делается в строке 29

метода `DisplayMessage`). Причины, на которых базируются такие рекомендации, вскоре станут понятны.

Модификаторы доступа public и private

Большинство объявлений переменных экземпляров начинается с ключевого слова `private` (как в строке 8). Ключевое слово `private`, как и `public`, является модификатором доступа. Переменные, свойства и методы, объявленные с модификатором доступа `private`, доступны только для членов класса, в котором они объявлены (например, для свойств и методов). Таким образом, переменная `courseName` может использоваться только в свойстве `CourseName` и методе `DisplayMessage` класса `GradeBook`.

АРХИТЕКТУРНОЕ РЕШЕНИЕ 4.2

Снабжайте каждое объявление поля и метода модификатором доступа. Как правило, переменные экземпляров должны объявляться закрытыми, а методы и свойства — открытыми. Если модификатор доступа перед членом класса отсутствует, по умолчанию подразумевается закрытый уровень доступа. Как вы вскоре увидите, некоторые методы можно объявлять закрытыми, если они будут вызываться только из других методов этого класса.

АРХИТЕКТУРНОЕ РЕШЕНИЕ 4.3

Объявляя переменные экземпляров класса закрытыми, а методы и свойства класса — открытыми, вы упрощаете отладку, поскольку проблемы с обработкой данных локализуются в методах и свойствах класса (закрытые переменные экземпляров доступны только для этих методов и свойств).

Объявление переменных экземпляров с модификатором доступа `private` называется сокрытием информации (или *инкапсуляцией*). Когда приложение создает объект класса `GradeBook`, переменная `courseName` инкапсулируется (скрывается) в объекте, а обращаться к ней могут только члены класса объекта.

Чтение и запись закрытых переменных экземпляров

Как разрешить программе работать с закрытыми переменными экземпляров класса, но проследить за тем, чтобы они имели корректные значения? Нужно предоставить в распоряжение программиста контролируемые механизмы чтения и записи значения переменной. И хотя вы можете определить методы вида `GetCourseName` и `SetCourseName`, свойства C# предоставляют более элегантное решение. Давайте посмотрим, как объявлять и использовать свойства.

Класс GradeBook со свойством

Объявление свойства `CourseName` класса `GradeBook` расположено в строках 11–21 на ил. 4.7. Свойство начинается в строке 11 с модификатора доступа (в данном случае `public`), за которым следует тип, представляемый свойством (`string`) и имя свойства (`CourseName`). Имена свойств формируются по тем же правилам, что и имена методов и классов.

Свойства содержат методы доступа, которые реализуют чтение и запись данных. Объявление свойства может содержать get-метод доступа и/или set-метод.