Паттерны проектирования

ДЛЯ C# И ПЛАТФОРМЫ .NET CORE

Гаурав Арораа, Джеффри Чилберто

Packt

ПИТЕР®

Hands-On Design Patterns with C# and .NET Core

Write clean and maintainable code by using reusable solutions to common software design problems

Gaurav Aroraa
Jeffrey Chilberto

Packt

BIRMINGHAM - MUMBAI

Паттерны проектирования

ДЛЯ C# И ПЛАТФОРМЫ .NET CORE

Гаурав Арораа
Джеффри Чилберто

ПИТЕР®
Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону • Самара • Минск

2021

ББК 32.973.2-018-02
УДК 004.42
А84

Арораа Гаурав, Чилберто Джеффри

А84 Паттерны проектирования для C# и платформы .NET Core. — СПб.: Питер, 2021. — 352 с.: ил. — (Серия «Для профессионалов»).

ISBN 978-5-4461-1523-5

Паттерны проектирования — удобный прием программирования для решения рутинных задач разработки ПО. Грамотное использование паттернов позволяет добиться соответствия любым требованиям и снизить расходы. В этой книге описаны эффективные способы применения паттернов проектирования с учетом специфики языка C# и платформы .NET Core.

Кроме знакомых паттернов проектирования из книги «Банды четырех» вы изучите основы объектно-ориентированного программирования и принципов SOLID. Затем узнаете о функциональных, реактивных и конкурентных паттернах, с помощью которых будете работать с потоками и корутинами. Заключительная часть содержит паттерны для работы с микросервисными, бессерверными и облачно-ориентированными приложениями. Вы также узнаете, как сделать выбор архитектуры, например микросервисной или MVC.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018-02
УДК 004.42

ISBN 978-1789133646 англ.

ISBN 978-5-4461-1523-5

Краткое содержание

Оглавление

Часть III. Функциональное программирование, реактивное программирование и кодирование для облака

Приложения

Моей матери Лате Шримати Сантош и в память о моем отце, покойном Ш. Рамкришане, за их жертвенность и твердость духа. Моей младшей сестренке, также покойной крошке Канчан, за ее любовь и за то, что была моим маленьким талисманом.

Гаурав Аророаа

Моим родителям Френсис и Джойс которые неустанно поднимали на ноги своих детей с любовью, заботой и добротой. Моим братьям: Джеку — за мотивацию сохранять спокойствие в череде испытаний и Майку — за напоминание остановиться и насладиться жизнью.

Джеффри Чилберто

Предисловие

При разработке качественного ПО программисты стремятся избегать дублирования кода. Мы весьма часто применяем прием DRY — Don't Repeat Yourself («Не повторяйся») — даже не задумываясь! Разработчики обычно разделяют функциональность, создают многократно используемые методы и пишут вспомогательные классы. Паттерны проектирования создавались и обновлялись годами. Это полезные, общепринятые и пригодные к многократному применению решения повседневных проблем.

Гаурав и Джеффри собрали лучшие и наиболее широко используемые паттерны и применили их в мире открытого C# и .NET Core. Вы начнете с ООП, классов, объектов и будете продвигаться далее к наследованию, инкапсуляции и полиморфизму. Авторы описали такие подходы, как DRY, KISS и SOLID (вы очень скоро поймете, что все это значит!), и применили их к классическим паттернам, которые помогут вам разрабатывать чистое и надежное программное обеспечение.

Книга содержит примеры рабочего кода, которые покажут, как использовать получаемые знания в создании ПО на .NET Code и C# в наши дни. Вы освоите паттерны «Строитель», «Декоратор», «Фабрика», «Посетитель», «Стратегия» и многие другие.

Эти методики будут применены сначала к простой программе, а затем к веб-приложениям. Далее будут рассмотрены более сложные темы, включая конкурентность и параллелизм. Наконец, вы сможете воспользоваться паттернами на принципиально более высоком уровне, применяя решения, которые помогут вам переместить проекты в облако, где их можно будет масштабировать и легко обслуживать.

Надеюсь, что, как и я, вы по достоинству оцените эту книгу. Желаю получать удовольствие, работая с .NET Core, так же, как получали его мы, когда разрабатывали эту среду!

Скотт Хансельман,
руководитель партнерской программы
Microsoft .NET и Open Source Community

Эта книга — библия для каждого разработчика, желающего не только улучшить свои навыки программирования, но и, самое главное, научиться создавать надежные, масштабируемые и удобные в сопровождении решения. Здесь описана бóльшая часть практических рекомендаций, дополненных наглядными примерами.

Помимо паттернов проектирования, книга затрагивает архитектурные принципы и ключевые моменты работы в облаке, такие как безопасность и масштабирование.

Будучи архитектором решений, я ежедневно проектирую комплексные приложения, уделяя особое внимание разработке инфраструктур и инструментов, и все же поражаюсь качеству контента и охвату тем, представленных в этой книге. В ней содержится исчерпывающий список паттернов, рекомендуемых к ознакомлению всем, кто связан с разработкой и развертыванием ПО.

Такое полное и глубокое погружение в мир объектно-ориентированного программирования (ООП) и .NET Core полезно всем, кто заинтересован в разработке лучших приложений.

Стефани Айскенс, Azure MVP

Об авторах

Гаурав Арораа получил степень магистра в области компьютерных наук, имеет сертификаты Microsoft MVP, Alibaba Cloud MVP, тренера/коуча по Scrum. Член *Компьютерного сообщества Индии (CSI)*, сотрудник IndiaMentor, сертифицированный специалист ITIL Foundation, APMG PRINCE-F и APMG PRINCE-P. Гаурав — разработчик открытого программного обеспечения, внесший вклад в развитие TechNet Wiki и основавший компанию Ovatic Systems Private Limited. За свою более чем 20-летнюю карьеру он обучил тысячи студентов. Вы можете написать Гауравy в Twitter по адресу @g_arora.

Спасибо моей жене Шуби Арораа и моему ангелу (дочери) Аачи Арораа, которые вытерпели мое отсутствие, пока я писал эту книгу. Спасибо всей команде Packt, особенно Чайтаньи, Акшите и Нее, координационная деятельность и коммуникабельность которых были так необходимы, а также Дениму Пинто, порекомендовавшему меня в качестве потенциального автора.

Джеффри Чилберто — консультант в сфере разработки программного обеспечения, специализирующийся на технологическом стеке Microsoft, включая Azure, BizTalk, ASP.NET, MVC, WCF и SQL Server, с опытом работы в различных сферах. Он участвовал в разработке банковских, телекоммуникационных и медицинских систем в Новой Зеландии, Австралии и США. Имеет степень бакалавра в области информационных и компьютерных наук, а также степень магистра по информационным технологиям и вычислительной технике.

Хотел бы поблагодарить мою семью за любовь, поддержку и вдохновение.

О научных редакторах

Сьеки Цааль — консультант по управлению, архитектор облачных систем Microsoft и Microsoft Azure MVP с более чем 15-летним стажем в области создания архитектур, разработки, консультирования и системного дизайна. Она работает в Capgemini — одной из крупнейших в мире консалтинговых компаний в сфере менеджмента и информационных технологий. Сьеки любит делиться знаниями и принимает очень активное участие в сообществе Microsoft как одна из основателей нидерландских пользовательских групп SP&C NL и MixUG. Кроме того, является членом правления Azure Thursdays. Сьеки часто выступает с лекциями и участвует в организации мероприятий. Она написала несколько книг, ведет блоги и принимает активное участие в сообществе Microsoft Tech Community (https://techcommunity.microsoft.com/).

У *Эфраима Кирякидиса* за плечами более 20 лет опыта в разработке ПО. Он получил диплом инженера в Университете имени Аристотеля в Салониках (Греция). Эфраим пользуется .NET с момента создания, с версии 1.0. В своей работе он в целом сфокусирован на технологиях Microsoft. В данный момент занимает пост старшего инженера в компании Siemens AG в Германии.

Введение

В этой книге на конкретных примерах описываются способы использования паттернов в современной разработке приложений. Количество паттернов, применяемых для поиска решений, огромно. Чаще всего разработчики пользуются ими, не понимая в полной мере, что и как эти паттерны делают. В издании рассматриваются паттерны от низкоуровневого кода до высокоуровневых концептуальных решений, которые работают в облачных системах.

Несмотря на то что большинство паттернов не требуют применения конкретного языка, для иллюстрации многих из них мы используем C# и .NET Core. Эти технологии были выбраны из-за их популярности и удобства архитектуры, на основе которой можно создавать различные решения — от простых консольных приложений до крупных корпоративных распределенных систем.

Описывая большое количество паттернов, эта книга знакомит со многими из них, позволяет глубже понять их на практике. Представленные паттерны были выбраны, чтобы проиллюстрировать определенные аспекты проектирования. Кроме того, добавлены ссылки на дополнительные материалы, которые помогут читателю подробнее изучить конкретный интересующий его паттерн.

И в простых сайтах, и в огромных корпоративных системах правильно подобранный паттерн может в корне изменить заведомо неверное решение, ведущее к краху проекта из-за его высокой стоимости и низкой производительности, и превратить это решение в выгодное, долгоживущее и успешное. Паттерны, описанные в книге, призваны преодолеть неизбежные проблемы, которые иначе не позволили бы вам оставаться конкурентоспособными. Они также позволят вашим приложениям достичь устойчивости и надежности, которыми должны обладать современные проекты.

Кому подойдет эта книга

Целевая аудитория — разработчики современных приложений. Поскольку книга содержит много кода, чтобы объяснить, как и где используются паттерны, подразумевается наличие у читателя опыта в разработке ПО. Не стоит рассматривать данную книгу как «программирование для чайников», скорее она отвечает на во-

прос «Как программировать лучше». Поэтому издание будет полезно начинающим и опытным программистам, архитекторам приложений и проектировщикам.

Структура издания

Глава 1 описывает объектно-ориентированное программирование и его применение в среде C#. Эта глава служит напоминанием о важных конструкциях и функциях ООП и C#, включающих наследование, инкапсуляцию и полиморфизм.

Глава 2 каталогизирует и представляет различные паттерны, используемые в современной разработке программного обеспечения. Эта глава исследует ряд паттернов и их каталогов, таких как SOLID, паттерны «Банды четырех» и паттерны корпоративной интеграции. Вдобавок в ней обсуждаются жизненный цикл и другие методики разработки ПО.

Глава 3 погрузит в паттерны проектирования, используемые для создания приложений на C#. На примере приложения-образца будут показаны разработка через тестирование, концепция продукта с минимальным функционалом и некоторые паттерны «Банды четырех».

Глава 4 продолжит знакомить с паттернами, используемыми в приложениях на C#. Будут введены такие понятия, как внедрение зависимости и инверсия управления, а также продолжится изучение паттернов проектирования, включая «Одиночку» и «Фабрику».

Глава 5 основана на главах 3 и 4 и описывает паттерны в .NET Core. Некоторые паттерны, включая «Внедрение зависимости» и «Фабрику», будут также пересмотрены с учетом среды .NET Core.

Глава 6 продолжает знакомить с .NET Core, описывая функции, поддерживаемые в разработке веб-приложений, в процессе создания тестового приложения. Эта глава содержит руководство по созданию базового веб-приложения, представляет важные характеристики, которыми оно должно обладать, и рассказывает, как создавать веб-страницы с поддержкой CRUD.

Глава 7 тоже посвящена разработке веб-приложений с помощью .NET Core и представляет различные архитектурные паттерны, а также варианты решений вопроса безопасности. Кроме того, описывает аутентификацию и авторизацию, а также модульные тесты с использованием Moq — фреймворка для создания имитаций реальных объектов.

Глава 8 углубляется в разработку веб-приложений и описывает понятие «конкурентность» при разработке приложений на C# и .NET Core. Будет рассмотрен паттерн async/await, а также многопоточность и конкурентность, которым посвящен

отдельный раздел. Кроме того, речь пойдет о Parallel LINQ с отложенным запуском и приоритетом потоков.

Глава 9 описывает функциональное программирование в .NET Core. Сюда входят функции языка C#, поддерживающие функциональное программирование, и их применение в тестовом приложении вместе с паттерном «Стратегия».

Глава 10 продолжает тему разработки веб-приложений .NET Core с помощью паттернов реактивного программирования и методик, используемых для создания адаптивных и масштабируемых сайтов. Глава описывает принципы реактивного программирования, включая паттерны `Reactive` и `IObservable`. Кроме того, рассматривает различные фреймворки, включая популярный проект .NET Rx Extensions, а также демонстрирует паттерн *«Модель — представление — модель представления»* (Model — View — View Model, MVVM).

Глава 11 освещает паттерны, используемые в проектировании баз данных, а также описывает сами БД. Будет показан практический пример применения паттерна CQRS, в том числе с помощью проектирования базы данных в виде журнала учета.

Глава 12 представляет разработку приложений в контексте облачных решений, включая пять ключевых аспектов: масштабирование, доступность, безопасность, проектирование приложений и DevOps. Будут описаны важнейшие паттерны, используемые в облачно-ориентированных решениях, включая различные типы масштабирования, и представлены паттерны, применяемые для событийно-ориентированных архитектур, интегрированной безопасности, кэша и телеметрии.

Приложение А содержит обсуждение дополнительных паттернов и практические рекомендации. Сюда включены разделы о моделировании примеров использования, рекомендованных методик и дополнительных паттернов, таких как пространственная архитектура и контейнерные приложения.

Приложение Б дает ответы на вопросы, размещенные в конце каждой главы.

Как получить максимум от книги

Предполагается, что вы уже немного знакомы с принципами ООП и языком C#. В книге описываются продвинутые темы, однако материал нельзя воспринимать как пошаговую инструкцию к применению. Цель издания — улучшить навыки разработчиков и проектировщиков с помощью широкого спектра паттернов, методик и принципов. По аналогии с ящиком для инструментов книга предлагает современным разработчикам эти самые инструменты для перехода от низкоуровневого проектирования к созданию более высокоуровневых архитектур, а также важные паттерны и принципы, широко используемые в настоящий момент.

Вдобавок эта книга обращает внимание на следующие моменты, призванные дополнить знания читателя:

- знание принципов SOLID, практические рекомендации с примерами кода на C#7.x и NET Core 2.2;
- углубленное понимание классических паттернов проектирования (от «Банды четырех»);
- принципы функционального программирования и рабочие примеры на языке C#;
- реальные примеры архитектурных паттернов (MVC, MVVM);
- понимание облачной ориентированности, микросервисов и пр.

Загрузка файлов с примерами

Вы можете скачать примеры из этой книги на GitHub по ссылке github.com/PacktPublishing/Hands-On-Design-Patterns-with-C-and-.NET-Core. В случае обновления кода примеры также будут обновлены в репозитории.

Код в действии

Перейдите по ссылке bit.ly/2KuuNgQ, чтобы просмотреть видео с примерами кода.

Полноцветные изображения

Мы также предоставляем PDF-файл с цветными изображениями схем и рисунков, используемыми в оригинале данной книги: static.packt-cdn.com/downloads/9781789133646_ColorImages.pdf.

Условные обозначения

В книге используется ряд условных обозначений.

Моноширинным шрифтом обозначены код в тексте, имена таблиц баз данных и т. п. Например: «В данном классе есть три метода: `CounterA()`, `CounterB()` и `CounterC()`, представляющих отдельную стойку для получения билетов».

Блок кода (листинг) оформляется так:

```
3-counters are serving...
Next person from row
Person A is collecting ticket from Counter A
Person B is collecting ticket from Counter B
Person C is collecting ticket from Counter C
```

Когда мы хотим обратить ваше внимание на конкретную часть листинга, то необходимые и ключевые строки выделяем **полужирным моноширинным** шрифтом:

```
public bool UpdateQuantity(string name, int quantity)
{
    lock (_lock)
    {
        _books[name].Quantity += quantity;
    }

    return true;
}
```

Любой ввод и вывод в оболочке командной строки выглядит следующим образом:

```
dotnet new sln
```

Курсивным шрифтом обозначены новые термины и важные слова.

URL, команды меню, элементы управления и заголовки окон выделяются шрифтом без засечек, например: «Команда Create New Product (Создать новый продукт) позволит добавить новый продукт, а Edit (Редактировать) — изменять и обновлять уже имеющиеся продукты».

Важные замечания и предупреждения обозначены так.

Советы и подсказки обозначены таким образом.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Часть I

Основы паттернов проектирования в C# и .NET Core

В этой части книги вы по-новому взглянете на паттерны проектирования. Мы изучим такие темы, как ООП, паттерны, методики и принципы SOLID. К концу части вы сможете самостоятельно создавать собственные паттерны.

Часть I состоит из таких глав:

- глава 1 «Обзор ООП в .NET Core и C#»;
- глава 2 «Современные паттерны и принципы проектирования ПО».

1 Обзор ООП в .NET Core и C#

Самые популярные за последние более чем 20 лет языки программирования были основаны на принципах *объектно-ориентированного программирования (ООП)*. Популярность языков с поддержкой ООП объясняется возможностью абстрагировать сложную логику программы в структуру, называемую объектом, которую намного проще объяснить и понять, а также легче использовать повторно. По сути, ООП есть подход к проектированию программного обеспечения. Точнее, это паттерн разработки ПО, который задействует концепцию объектов, включающих в себя данные и функциональность. По мере развития индустрии программного обеспечения в ООП появились паттерны для часто встречающихся задач, поскольку эффективно решались одни и те же вопросы, но в разных контекстах и отраслях. По мере продвижения ПО от мейнфреймов к клиентам и серверам, а затем к облакам появлялись новые паттерны, помогающие уменьшить стоимость разработки и повысить надежность программного обеспечения. В этой книге мы исследуем паттерны проектирования, начиная с их появления в ООП и заканчивая созданием паттернов проектирования облачного ПО.

ООП основано на концепции объекта. Объект обычно содержит данные, известные как свойства или поля, а также код или поведение, известное как методы.

Паттерны проектирования — решения распространенных проблем, с которыми программисты сталкиваются в процессе создания ПО. Эти решения построены на опыте того, что работает, а что — нет, и опробованы и протестированы многими разработчиками в различных ситуациях. Преимущество использования паттернов, основанных на предыдущем опыте, заключается в том, что они избавляют от необходимости повторять одни и те же действия снова и снова. Вдобавок применение паттернов дает ощущение уверенности в том, что при решении проблемы не появятся новые ошибки или затруднения.

Эта глава описывает ООП и его применение в C#. Обратите внимание: данная информация — просто краткое введение в тему, она не является полноценным руководством по ООП или языку C#. Напротив, глава рассказывает об ООП и C# достаточно подробно для того, чтобы познакомить вас с паттернами проектирования, которые будут обсуждаться в других главах.

В этой главе будут рассмотрены следующие темы:

- дискуссия об ООП, о работе классов и объектов;
- наследование;
- инкапсуляция;
- полиморфизм.

Технические требования

В этой главе содержатся примеры кода, объясняющие принципы качественной разработки. Код упрощен и предназначен лишь для демонстрации. В большинстве примеров используется консольное приложение на основе .NET Core, написанное на C#.

Чтобы запустить и выполнить код, вам потребуется следующее:

- Visual Studio 2019 (вы также можете запустить приложение, используя Visual Studio 2017 версии 3 и новее);
- .NET Core;
- SQL Server (в этой главе используется Express Edition).

Установка Visual Studio

Чтобы запустить примеры кода, вам необходимо установить Visual Studio. Для этого выполните следующие шаги.

1. Скачайте Visual Studio по ссылке docs.microsoft.com/ru-ru/visualstudio/install/install-visual-studio.
2. Следуйте инструкциям по установке. Доступны различные версии Visual Studio; в этой главе мы используем Visual Studio для Windows.

Вы также можете применить другую интегрированную среду разработки на ваше усмотрение.

Установка .NET Core

Если вы еще не установили .NET Core, то следуйте инструкции ниже.

1. Скачайте .NET Core по ссылке dotnet.microsoft.com/download.
2. Далее следуйте указаниям по установке соответствующей библиотеки: dotnet.microsoft.com/download/dotnet-core/2.2.

Полная версия исходного кода доступна на GitHub. Код, представленный в данной главе, может быть неполным, поэтому мы рекомендуем вам скачать код для запуска примеров (github.com/PacktPublishing/Hands-On-Design-Patterns-with-C-and-.NET-Core/tree/master/Chapter1).

Используемые в книге модели

Как практическое руководство, книга содержит много примеров кода на C# наряду с различными схемами и рисунками, призванными объяснить те или иные концепции. Это не книга по *унифицированному языку моделирования* (Unified Modeling Language, UML), однако тем, кто знаком с UML, многие схемы будут известны. В этом подразделе приводятся примеры схем классов, используемых в издании.

При обозначении класса указываются его поля и методы, разделенные пунктирными линиями. В случаях, важных для обсуждения, типы доступа будут обозначены так: `-` для приватных (закрытых), `+` для публичных (открытых), `#` для защищенных и `~` для внутренних. На рис. 1.1 это показано на примере класса `Car` с приватной переменной `_name` и публичным методом `GetName()`.

Когда описываются отношения между объектами, ассоциация показана сплошной линией, агрегация — в виде ромба, а композиция — в виде закрашенного ромба. При необходимости рядом с классом указывается количество его экземпляров. На рис. 1.2 показано, что у класса `Car` есть один водитель — `Owner` и до трех пассажиров — `Passengers`; кроме того, он имеет четыре колеса — `Wheels`.

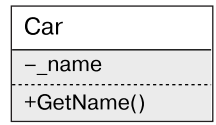


Рис. 1.1

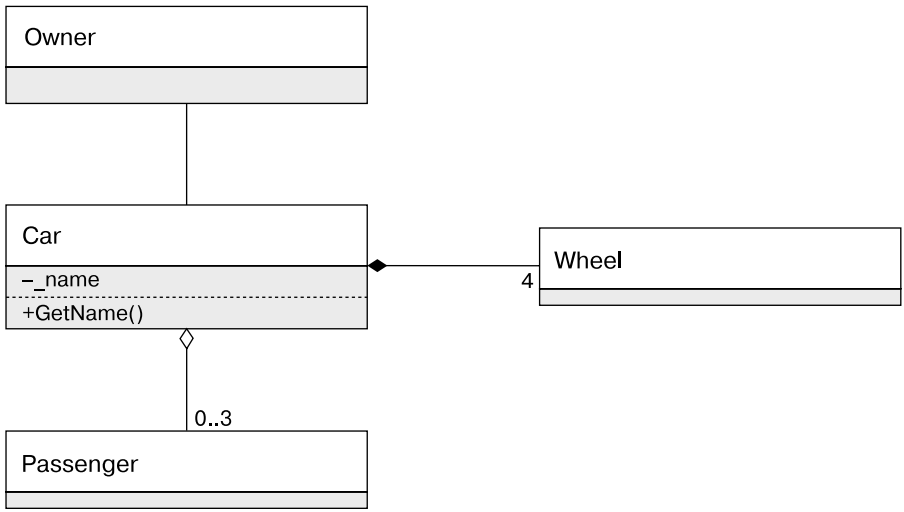


Рис. 1.2

Наследование обозначено с помощью пустого треугольника и сплошной линии, которые ведут к базовому классу. На рис. 1.3 отображены отношения между базовым классом `Account` и расширяющими его классами `SavingsAccount` и `CheckingAccount`.

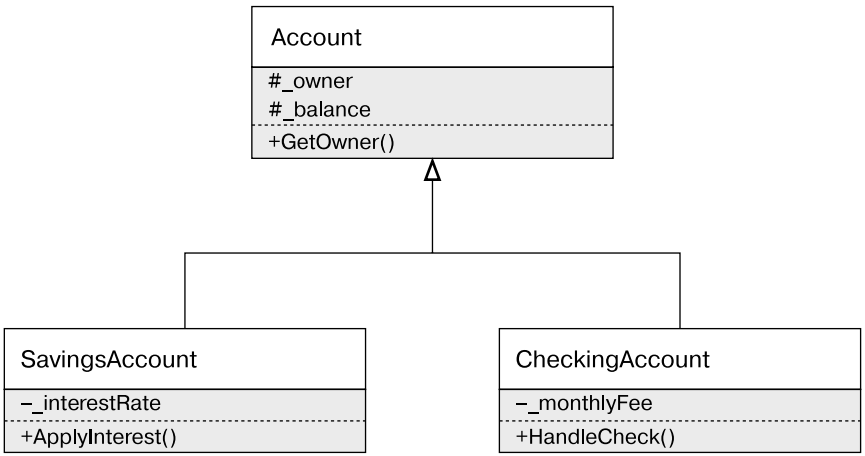


Рис. 1.3

Интерфейсы обозначаются похожим образом, но с использованием пунктирной линии и дополнительной метки `<<interface>>` (рис. 1.4).

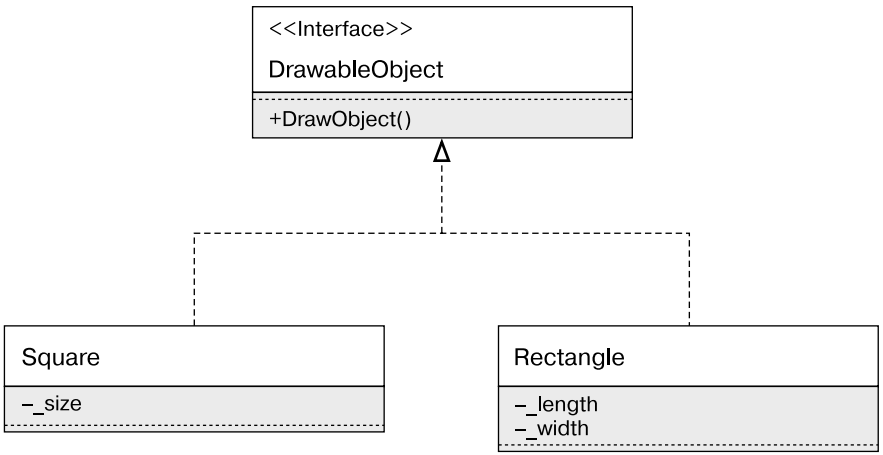


Рис. 1.4

В этом подразделе мы рассматриваем модели, используемые в книге. Такой стиль/подход выбран в расчете на то, что он знаком большинству читателей.

Определение ООП. Как работают классы и объекты

ООП — подход к разработке ПО, использующий объекты, определяемые в виде классов. Эти определения содержат поля, иногда называемые атрибутами, для сохранения данных, и методы для поддержки функциональности. Первый ООП-язык был симуляцией реальных систем. Он известен как Simula (en.wikipedia.org/wiki/Simula) и разработан в Норвежском вычислительном центре в 1960-е годы. Первый полноценный ООП-язык появился на свет в 1970-е и назывался Smalltalk (https://ru.wikipedia.org/wiki/Smalltalk). Он был разработан для устройства Dynabook (history-computer.com/ModernComputer/Personal/Dynabook.html), персонального компьютера, созданного Алланом Кеем. Smalltalk в какой-то степени стал прародителем некоторых других известных ООП-языков, таких как Java, C++, Python и C#.

ООП базируется на объектах, содержащих данные. Эта парадигма программирования позволяет разработчикам организовывать код в абстрактную или логическую структуру, называемую объектом. Он может содержать как данные, так и поведение.

ООП позволяет:

- *применить модуляризацию* — приложение разбивается на несколько разных модулей;
- *повторно использовать ПО* — приложение собирается из уже имеющихся или новых модулей. В следующих разделах мы подробно обсудим концепции ООП.

Определение ООП

Ранее подходы к программированию имели определенные ограничения и часто были сложными в сопровождении. ООП предлагает новую парадигму в разработке приложений, которая более выгодна по сравнению с другими подходами. Концепция организации кода в объекты проста для понимания и дает большое преимущество при освоении нового паттерна. Объяснить концепцию можно на примерах из реального мира. Сложные системы могут быть описаны с помощью более мелких блоков (то есть *объектов*). Это позволяет разработчикам анализировать отдельные компоненты и понимать их роль в самом решении.

Памятуя об этом, определим программу следующим образом: «*Программа — это список инструкций, указывающих компилятору языка, что ему делать*».

Как видите, объект предоставляет способ организации списка инструкций логическим образом. Возвращаясь к примеру с домом, инструкции архитектора помогают нам построить дом, но сами они домом не являются. Инструкции — абстрактное представление дома. Определение класса — нечто подобное, так как описывает

характеристики объекта. Объект создается на основе определения класса, это часто называется *инстанцированием* (то есть *созданием экземпляра класса*).

Чтобы лучше понять ООП, следует отметить два других важных подхода к программированию.

- *Структурное программирование*. Это понятие было введено Эдсгером Дейкстрой в 1966 году. Структурное программирование — парадигма программирования, которая решает проблему управления 1000 строк кода и разделяет их на меньшие фрагменты. Их обычно называют *подпрограммами*, *блочными структурами*, циклами `for`, `while` и пр. К языкам программирования, использующим структурные методики, относятся ALGOL, Pascal, PL/I и др.
- *Процедурное программирование*. Эта парадигма появилась из структурного программирования и базируется на том, как выполняется вызов (или *процедурный вызов*). К языкам, использующим процедурные методы программирования, относятся COBOL, Pascal и C. Последний пример подобного языка — Go, выпущенный в 2009 году.

Процедурные вызовы

Процедурный вызов происходит, когда выполняется набор операторов, то есть процедура.

Главная проблема указанных подходов в том, что чем больше становятся программы, тем сложнее их обслуживать. Программы с более сложными и громоздкими кодовыми базами затрудняют применение этих двух парадигм; как следствие, усложняются анализ и сопровождение кода. Чтобы можно было решить эти проблемы, ООП предоставляет определенные механизмы, такие как:

- наследование;
- инкапсуляция;
- полиморфизм.

В следующих разделах мы подробно обсудим эти понятия.

Наследование, инкапсуляцию и полиморфизм иногда называют тремя столпами ООП.

Прежде чем начнем, обсудим некоторые структуры, присутствующие в ООП.

Класс

Класс — это групповое или шаблонное определение методов и переменных, которые описывают объект. Другими словами, класс — это образец, содержащий определение переменных и методов, общих для всех экземпляров класса. Экземпляр класса называется объектом.

Посмотрим на следующий пример кода:

```
public class PetAnimal
{
    private readonly string PetName;
    private readonly PetColor PetColor;

    public PetAnimal(string petName, PetColor petColor)
    {
        PetName = petName;
        PetColor = petColor;
    }

    public string MyPet() => $"My pet is {PetName}
      and its color is {PetColor}.";
}
```

В коде мы имеем класс `PetAnimal`, содержащий два приватных поля — `PetName` и `PetColor`, а также метод `MyPet()`.

Объект

В реальном мире объектам присущи две характеристики: состояние и поведение. Эти характеристики — не что иное, как состояние объекта. Возьмем для примера любое животное: у собаки и у кошки есть кличка. Так, собаку одного из авторов зовут Эйс, а кошку — Клементина. Им также присуще разное поведение: к примеру, собаки лают, а кошки мяукают.

В подразделе «Определение ООП» мы обсудили, что ООП — это модель программирования, призванная объединить состояние или структуру (данные) и поведение (методы), чтобы предоставить программный функционал. В предыдущем примере различные состояния животных являются данными, а их поведение — это метод.

Объект хранит информацию (обычно данные) в атрибутах и задает поведение через методы.

В терминах ООП-языка, такого как C#, объект — это экземпляр класса. В нашем предыдущем примере реальный объект `Dog` мог бы быть объектом класса `PetAnimal`.

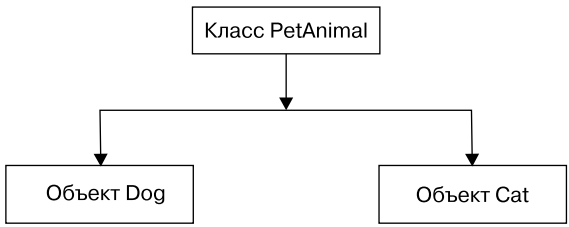
Объекты могут быть либо конкретными (то есть реальными, например собака или кошка, или любой тип файлов, скажем физический или электронный файл), либо же концептуальными, такими как схемы баз данных или образцы кода.

В следующем фрагменте кода показан объект, объединяющий в себе данные и метод, и пример его использования:

```
namespace OOPExample
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("OOP example");
            PetAnimal dog = new PetAnimal("Ace", PetColor.Black);
            Console.WriteLine(dog.MyPet());
            Console.ReadLine();
            PetAnimal cat = new PetAnimal("Clementine", PetColor.Brown);
            Console.WriteLine(cat.MyPet());
            Console.ReadLine();
        }
    }
}
```

В этом фрагменте кода мы создали два объекта: `dog` и `cat`. Они являются двумя разными экземплярами класса `PetAnimal`. Вы могли заметить, что поля или описания, содержащие данные о животных, получают свои значения из метода-конструктора. Это специальный метод, используемый для создания экземпляра класса.

Визуализируем данный пример — взгляните на рис. 1.5.



Наглядная иллюстрация объекта класса

Рис. 1.5

Это наглядное представление нашего предыдущего примера кода, в котором мы создали два разных объекта, `Dog` и `Cat`, класса `PetAnimal`. Схема интуитивно понятна: объект класса `Dog`, равно как и `Cat`, — экземпляр класса `PetAnimal`.

Ассоциации

Ассоциации объекта — важная особенность ООП. Подобно тому как в реальном мире объекты связаны друг с другом отношениями, в ООП ассоциации позволяют определять отношения между объектами типа *has-a* («имеет»). Например, велосипедист *имеет* велосипед, а кот *имеет* нос.

Существует несколько типов отношений *has-a*.

- *Ассоциация* — используется для описания отношений между объектами без описания собственности, как, например, отношения между машиной и человеком (в этом случае отношения могут быть представлены вождением). Человек может водить несколько машин, и одним автомобилем может управлять несколько человек.
- *Агрегация* — специальная форма ассоциации. Объекты также имеют жизненный цикл, но в данном случае предусматривается владение. Это значит, что дочерний объект не может принадлежать другому родительскому объекту. Агрегация — однонаправленный тип отношений, в которых объекты существуют независимо друг от друга. Например, отношения ребенка и родителей — это агрегация, так как каждый ребенок имеет родителей, однако не у всяких родителей есть ребенок.
- *Композиция* — определяет отношение удаления. Она описывает отношения между двумя объектами, где один (дочерний) зависит от другого (родительского). Если родитель удален, то все его дочерние объекты удаляются автоматически. Рассмотрим это на примере дома и комнаты. В одном доме может быть множество комнат, но одна комната не может быть одновременно во множестве домов. Если мы разрушим дом, то комната тоже будет уничтожена.

Посмотрим, как эти принципы применяются в C#, немного расширив предыдущий пример с животными за счет добавления класса `PetOwner`. Он может быть ассоциирован с одним или несколькими экземплярами `PetAnimal`. Поскольку у данного класса может не быть владельца, такое отношение является агрегацией. `PetAnimal` относится к `PetColor`, и в этой системе `PetColor` существует, только если относится к `PetAnimal`, делая ассоциацию композицией.

На рис. 1.6 показана как агрегация, так и композиция.

Данная модель основана на UML и может быть вам незнакома, поэтому выделим в ней ключевые моменты. Класс отображен как блок, состоящий из имени, атрибутов и методов, разделенных линиями. Пока проигнорируем символы перед именем метода, например `+` и `–`; мы рассмотрим эти модификаторы доступа чуть позже, при обсуждении инкапсуляции. Ассоциации обозначены с помощью линий, соединяющих классы. В случае композиции используется сплошной ромб, направленный к родителю; пустой ромб служит для обозначения агрегации. Обратите внимание: представленная схема позволяет показать количество возможных потомков. На ней класс `PetOwner` может иметь 0 или более классов `PetAnimal` (знак * говорит о том, что количество ассоциаций не ограничено).

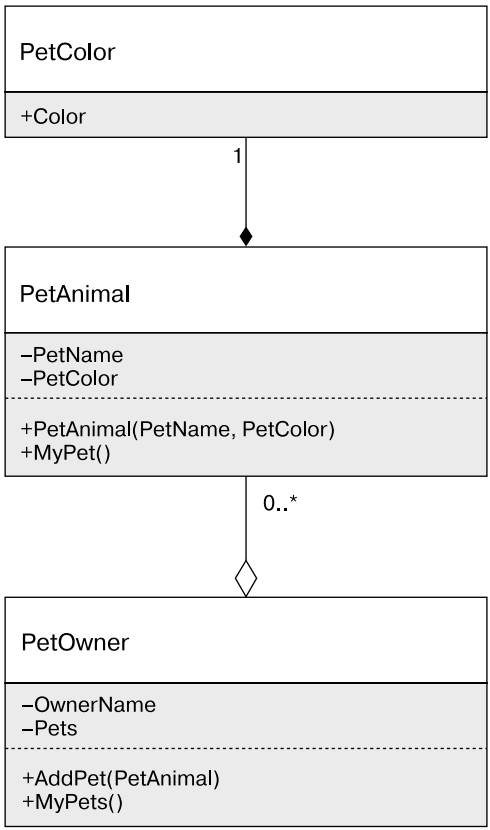


Рис. 1.6

UML

UML — язык моделирования, служащий для разработки программного обеспечения. Он был создан более 20 лет назад и поддерживается Группой управления объектами (Object Management Group, OMG). Пройдя по ссылке www.uml.org, можно узнать подробности.

Интерфейс

Интерфейс в C# определяет содержимое объекта или его контракт: в частности, методы, свойства, события или индексы объекта. Однако интерфейсы не предоставляют реализацию. В отличие от базового класса, предоставляющего и контракт, и реализацию, у них не может быть атрибутов. Класс, который реализует интерфейс, должен реализовывать все, что определено в интерфейсе.

Абстрактный класс

Абстрактный класс — это гибрид интерфейса и базового класса, предоставляющий и реализацию, и атрибуты, а также методы, которые должны быть определены в дочерних классах.

Сигнатура

Термин, которым тоже можно описать контракт объекта.

Наследование

Наследование — один из самых важных принципов ООП. Наследование между классами позволяет определить *тип* отношения: например, машина — *тип* транспорта. Важность принципа в том, что он позволяет объектам одного типа иметь схожие функции и признаки. Допустим, у нас есть система управления книжным интернет-магазином. Мы можем иметь один класс для хранения информации о бумажной копии книги, другой — для хранения данных о цифровой копии. Схожие признаки этих классов, такие как название, издатель и автор, могут храниться в каком-то третьем классе. Тогда классы бумажного и цифрового изданий должны наследоваться от него.

Для описания классов в наследовании есть разные термины: дочерние, или производные, классы наследуются от другого класса, в то время как классы, от которых наследуют, называются базовыми, или родительскими.

Далее мы подробно обсудим наследование.

Типы наследования. Наследование помогает определить дочерний класс. Этот класс наследует поведение родительского, или базового.

Наследование в C# обозначается двоеточием (:).

Посмотрим на различные типы наследования.

- *Одиночное наследование* — часто встречающийся тип наследования, описывает один класс, унаследованный от другого.

  Вернемся к упомянутому ранее классу `PetAnimal` и используем наследование для определения классов `Dog` и `Cat`. С его помощью мы можем определить некоторые атрибуты, общие для обоих классов. Так, кличка животного и цвет могут быть общими, поэтому располагаются в базовом классе. Специфические свойства кота или собаки, напротив, должны быть определены в отдельных классах: например, звуки, которые издают собаки и кошки. На рис. 1.7 показан класс `PetAnimal` с двумя дочерними классами.

Гаурав Аророаа, Джеффри Чилберто

Паттерны проектирования для C# и платформы .NET Core

Перевел с английского *С. Черников*

Заведующая редакцией	*Ю. Сергиенко*
Руководитель проекта	*С. Давид*
Ведущий редактор	*Н. Гринчик*
Научный редактор	*М. Сагалович*
Литературный редактор	*Н. Хлебина*
Художественный редактор	*В. Мостипан*
Корректоры	*Е. Павлович, Е. Рафалюк-Бузовская*
Верстка	*Г. Блинов*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 01.2021. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —
Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 15.01.21. Формат 70×100/16. Бумага офсетная. Усл. п. л. 28,380. Тираж 500. Заказ 0000.