# Часть 1. Паттерны поведения

При разработке любого приложения, программист сталкивается с определенным набором типовых задач. Многие из них он решает практически не задумываясь и во многих случаях даже не замечает, что раз за разом использует известные паттерны проектирования.

Каждый раз, когда разработчик выделяет базовый класс или интерфейс для абстрагирования от некоторого процесса, он использует паттерн *Стратегия*. Сохранение данных или сериализация, шифрование или сортировка, валидация или чтение конфигурации, все это примеры стратегий. Стратегия позволяет отделить клиента от подробностей реализации некоторого процесса и позволяет заменять одну реализацию на другую во время исполнения.

Наследование позволяет расширять объектные системы, но оно же делает их излишне сложными. Паттерн *Шаблонный метод* позволяет формализовать отношения между базовым классом и его наследниками. Данный паттерн определяет базовый каркас алгоритма, давая возможность наследникам определить лишь недостающие куски головоломки. Шаблонный метод вместе с принципами проектирования по контракту позволяют сделать иерархии наследования более понятными и предсказуемыми с точки зрения разработчика.

Паттерн *Посредник* позволяет связать несколько автономных классов воедино для получения более высокоуровневого поведения. Вместо того, чтобы классы знали друг о друге, они будут знать лишь о посреднике, что уменьшит связанность, а также упростит их понимание и повторное использование.

*Итераторы* позволяют работать с различными коллекциями или последовательностями унифицированным образом. Большинство современных языков программирования, включая C#, поддерживают потребление итераторов на уровне языка с помощью цикла foreach, а также позволяют легко создавать итераторы с помощью блоков итераторов и ключевого слова yield return.

События в языке C# являются примером очередного паттерна проектирования - паттерна *Наблюдатель*. Наблюдатели уменьшают связанность между классом и его клиентами, поскольку позволяют уведомлять всех заинтересованных подписчиков не зная о их количестве и конкретных типах. Возросший интерес к функциональному программированию, привел к появлению реактивных расширений (reactive extension), которые являются смесью паттернов наблюдатель и итератор.

Объектно-ориентированное программирование славится своей расширяемостью путем порождения подклассов. Но легкость добавления подклассов идет рука об руку со сложностью добавления новых операций в базовый класс иерархии. Паттерн *Посетитель* решает эту проблему за счет двойной диспетчирезации, что позволяет легко добавлять новую операцию без изменения иерархии типов.