Глава 5. Паттерны поведения

Паттерны поведения связаны с алгоритмами и распределением обязанностей между объектами. Руечь в них идет не только о самих объектах и классах, но и о типичных способах взаимодействия. Паттерны поведения характеризуют сложный поток управления, который трудно проследить во время выполнения программы. Внимание акцентировано не на потоке управления как таковом, а на связях между объектами.

В паттернах поведения уровня класса используется наследование – чтобы распределить поведение между разными классами. В этой главе описано два таких паттерна. Из них более простым и широко распространенным является **шаблонный метод**, который представляет собой абстрактное определение алгоритма. Алгоритм здесь определяется пошагово. На каждом шаге вызывается либо примитивная, либо абстрактная операция. Алгоритм «обрастает мясом» за счет подклассов, где определены абстрактные операции. Другой паттерн поведения уровня класса – **интерпретатор**, который представляет грамматику языка в виде иерархии классов и реализует интерпретатор как последовательность операций над экземплярами этих классов.

В паттернах поведения уровня объектов используется не наследование, а композиция. Некоторые из них описывают, как с помощью кооперации множество равноправных объектов справляется с задачей, которая ни одному из них не под силу. Важно здесь то, как объекты получают информацию о существовании друг друга. Объекты-коллеги могут хранить ссылки друг на друга, но это увеличит степень связанности системы. При максимальной степени связанности каждому объекту пришлось бы иметь информацию обо всех остальных. Эту проблему решает паттерн **посредник**. Посредник, находящийся между объектами-коллегами, обеспечивает косвенность ссылок, необходимую для разрывания лишних связей.

Паттерн **цепочка обязанностей** позволяет и дальше уменьшать степень связанности. Он дает возможность посылать запросы объекту не напрямую, а по цепочке «объектов-кандидатов». Запрос может выполнить любой «кандидат», если это допустимо в текущем состоянии выполнения программы. Число кандидатов заранее не определено, а подбирать участников можно во время выполнения.

Паттерн **наблюдатель** определяет и отвечает за зависимости между объектами. Классический пример **наблюдателя** встречается в схеме **модель/вид/контроллер** языка Smalltalk, где все виды модели уведомляются о любых изменениях ее состояния.

Прочие паттерны поведения связаны с инкапсуляцией поведения в объекте и делегированием ему запросов. Паттерн **стратегия** инкапсулирует алгоритм объекта,

упрощая его спецификацию и замену. Паттерн команда инкапсулирует запрос в виде объекта, который можно передавать как параметр, хранить в списке истории или использовать как-то иначе. Паттерн состояние инкапсулирует состояние объекта таким образом, что при изменении состояния объект может изменять поведение. Паттерн посетитель инкапсулирует поведение, которое в противном случае пришлось бы распределять между классами, а паттерн итератор абстрагирует способ доступа и обхода объектов из некоторого агрегата.

Паттерн Chain of Responsibility

Название и классификация паттерна

Цепочка обязанностей – паттерн поведения объектов.

Назначение

Позволяет избежать привязки отправителя запроса к его получателю, давая шанс обработать запрос нескольким объектам. Связывает объекты-получатели в цепочку и передает запрос вдоль этой цепочки, пока его не обработают.

Мотивация

Рассмотрим контекстно-зависимую оперативную справку в графическом интерфейсе пользователя, который может получить дополнительную информацию по любой части интерфейса, просто щелкнув на ней мышью. Содержание справки зависит от того, какая часть интерфейса и в каком контексте выбрана. Например, справка по кнопке в диалоговом окне может отличаться от справки по аналогичной кнопке в главном окне приложения. Если для некоторой части интерфейса справки нет, то система должна показать информацию о ближайшем контексте, в котором она находится, например о диалоговом окне в целом.

Поэтому естественно было бы организовать справочную информацию от более конкретных разделов к более общим. Кроме того, ясно, что запрос на получение справки обрабатывается одним из нескольких объектов пользовательского интерфейса, каким именно – зависит от контекста и имеющейся в наличии информации.

Проблема в том, что объект, *инициирующий* запрос (например, кнопка), не располагает информацией о том, какой объект в конечном итоге предоставит справку. Нам необходим какой-то способ отделить кнопку-инициатор запроса от объектов, владеющих справочной информацией. Как этого добиться, показывает паттерн цепочка обязанностей.

Идея заключается в том, чтобы разорвать связь между отправителями и получателями, дав возможность обработать запрос нескольким объектам. Запрос перемещается по цепочке объектов, пока один из них не обработает его.

Первый объект в цепочке получает запрос и либо обрабатывает его сам, либо направляет следующему кандидату в цепочке, который ведет себя точно так же. У объекта, отправившего запрос, отсутствует информация об обработчике. Мы говорим, что у запроса есть *анонимный получатель* (implicit receiver).