ГЛАВА 5

ПАТТЕРНЫ ПОВЕДЕНИЯ

Паттерны поведения связаны с алгоритмами и распределением обязанностей между объектами. Речь в них идет не только о самих объектах и классах, но и о типичных схемах взаимодействия между ними. Паттерны поведения характеризуют сложный поток управления, который трудно проследить во время выполнения программы. Внимание акцентируется не на схеме управления как таковой, а на связях между объектами.

В паттернах поведения уровня класса для распределения поведения между разными классами используется наследование. В этой главе описано два таких паттерна. Из них более простым и широко распространенным является шаблонный метод (373), который представляет собой абстрактное определение алгоритма. Алгоритм здесь определяется пошагово. На каждом шаге вызывается либо примитивная, либо абстрактная операция. Алгоритм детализируется за счет подклассов, где определяются абстрактные операции. Другой паттерн поведения уровня класса — интерпретатор (287) — 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. Одноранговые объекты 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 (319). Посредник, находящийся между объектами-коллегами, обеспечивает косвенность ссылок, необходимую для разрыва лишних связей.

Паттерн цепочка обязанностей (263) 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 (339) определяет и поддерживает зависимости между объектами. Классический пример наблюдателя встречается в схеме «модель —представление — 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 (362) инкапсулирует алгоритм объекта, упрощая его спецификацию и замену. Паттерн команда (275) инкапсулирует запрос в виде объекта, который можно передавать как параметр, хранить в списке истории или использовать как-то иначе. Паттерн состояние (352) инкапсулирует состояние объекта таким образом, что при изменении состояния объект может изменять свое поведение. Паттерн посетитель (379) инкапсулирует поведение, которое в противном случае пришлось бы распределять между классами, а паттерн итератор (302) абстрагирует механизм доступа и обхода объектов из некоторого агрегата.

ПАТТЕРН CHAIN OF RESPONSIBILITY (ЦЕПОЧКА ОБЯЗАННОСТЕЙ)

Название и классификация паттерна

Цепочка обязанностей — паттерн поведения объектов.

Назначение

Позволяет избежать привязки отправителя запроса к его получателю, предоставляя возможность обработать запрос нескольким объектам. Связывает объекты-получатели в цепочку и передает запрос по этой цепочке, пока он не будет обработан.

Мотивация

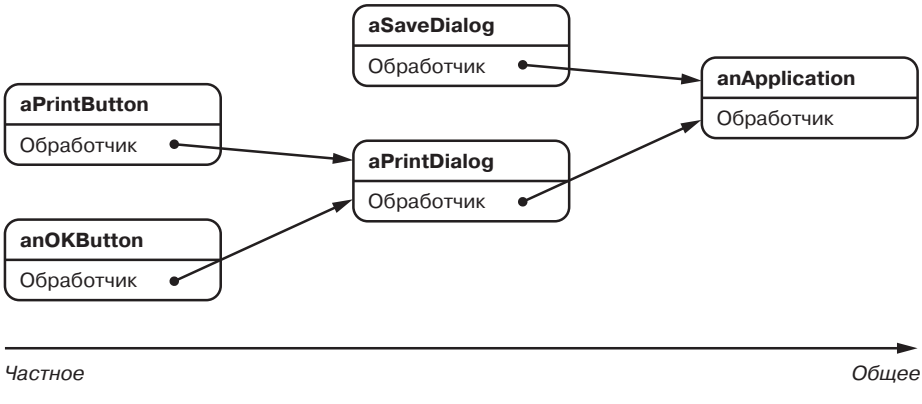
Рассмотрим контекстнозависимую оперативную справку в графическом интерфейсе: пользователь может получить дополнительную информацию по любой части интерфейса, просто щелкнув на ней мышью. Содержание

справки зависит от того, какая часть интерфейса была выбрана и в каком контексте. 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.

Следовательно, 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, а каким именно — зависит от контекста и имеющейся в наличии информации.

Проблема в том, что объект, *инициирующий* запрос (например, кнопка), не знает, какой объект в конечном итоге предоставит справку. Необходимо каким-то образом отделить кнопку — инициатор запроса от объектов, которые могут предоставить справочную информацию. Паттерн цепочка обязанностей показывает, как это может происходить.

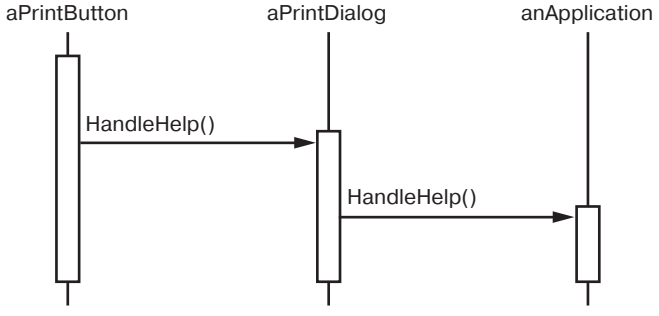
Идея паттерна заключается в том, чтобы разорвать связь между отправителями и получателями, дав возможность обработать запрос нескольким объектам. Запрос перемещается по цепочке объектов, пока не будет обработан одним из них.



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

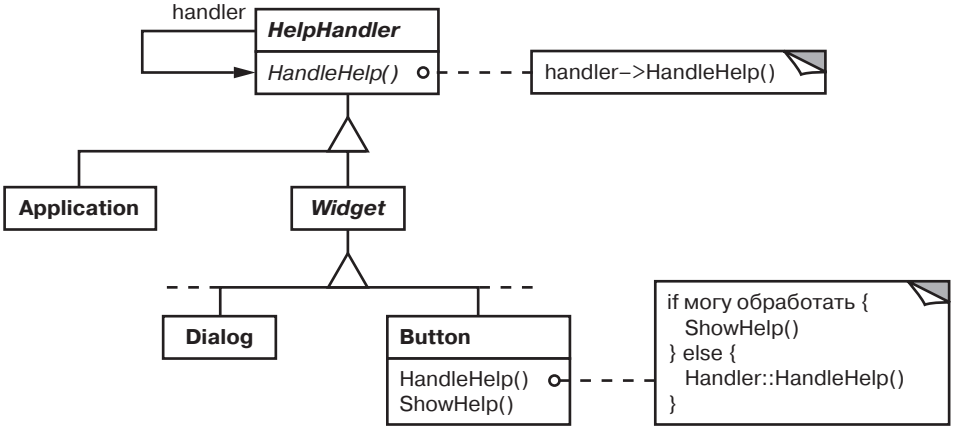
Допустим, пользователь запрашивает справку по кнопке Print (печать). Она находится в диалоговом окне `PrintDialog`, содержащем информацию об объ-

екте приложения, которому принадлежит (см. предыдущую диаграмму). На следующей диаграмме взаимодействий показано, как запрос на получение справки перемещается по цепочке.



В данном случае ни кнопка `aPrintButton`, ни окно `aPrintDialog` не обрабатывают запрос; он достигает объекта `anApplication`, который может его обработать или игнорировать. У клиента, инициировавшего запрос, нет прямой ссылки на объект, который его в конце концов выполнит.

Чтобы отправить запрос по цепочке и гарантировать анонимность получателя, все объекты в цепочке имеют единый интерфейс для обработки запросов и для доступа к своему *преемнику* (следующему объекту в цепочке). Например, в системе оперативной справки можно было бы определить класс `HelpHandler` (предок классов всех объектов-кандидатов или класс-примесь (mixin class)) с операцией `HandleHelp`. Тогда классы, которые хотят обрабатывать запросы, могут сделать `HelpHandler` своим родителем:



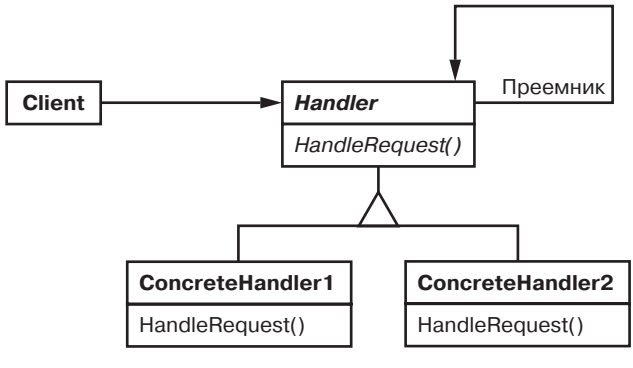
Для обработки запросов на получение справки классы `Button`, `Dialog` и `Application` пользуются операциями `HelpHandler`. По умолчанию операция `HandleHelp` просто перенаправляет запрос своему преемнику. В подклассах эта операция замещается, так что при благоприятных обстоятельствах может выдаваться справочная информация. В противном случае запрос отправляется дальше посредством реализации по умолчанию.

Применимость

Основные условия для применения паттерна цепочка обязанностей:

- запрос может быть обработан более чем одним объектом, причем настоящий обработчик заранее неизвестен и должен быть найден автоматически;
- запрос должен быть отправлен одному из нескольких объектов, без явного указания, какому именно;
- набор объектов, способных обработать запрос, должен задаваться динамически.

Структура



Типичная структура объектов выглядит примерно так: