Шаблон Controller (контроллер)

Проблема. *Кто должен отвечать за обработку входных системных событий?*

Системное событие (system event) – это событие высокого уровня, генерируемое внешним исполнителем (событие с внешним входом). Системные события связаны с системными операциями (system operation), т.е. операциями, выполняемыми системой в ответ на события.

Решение. *Делегирование обязанностей по обработке системных сообщений классу, удовлетворяющему одному из следующих условий.*

- *Класс представляет всю систему в целом, устройство или подсистему (внешний контроллер).*
- *Класс представляет сценарий некоторого прецедента, в рамках которого выполняется обработка всех системных событий, и обычно называется Handler, Coordinator или Session (контроллер прецедента или контроллер сеанса).*
- *Для всех системных событий в рамках одного сценария прецедента используется один и тот же класс-контроллер.*

Неформально, сеанс – это экземпляр взаимодействия с исполнителем. Сеансы могут иметь произвольную длину, но зачастую организованы в рамках прецедента (сеансы прецедента).

Заметим, что в этот перечень не включаются классы, реализующие окно, аплет, приложение, вид и документ. Такие классы не выполняют задачи, связанные с системными событиями. Они обычно получают сообщения и делегируют их контроллерам.

Контроллер (controller) – это объект, не относящийся к интерфейсу пользователя и отвечающий за обработку системных событий. Контроллер определяет методы для выполнения системных операций.

Какой класс должен выступать в роли контроллера для системных событий типа enterItem или endSale (Рис. 0.1)?

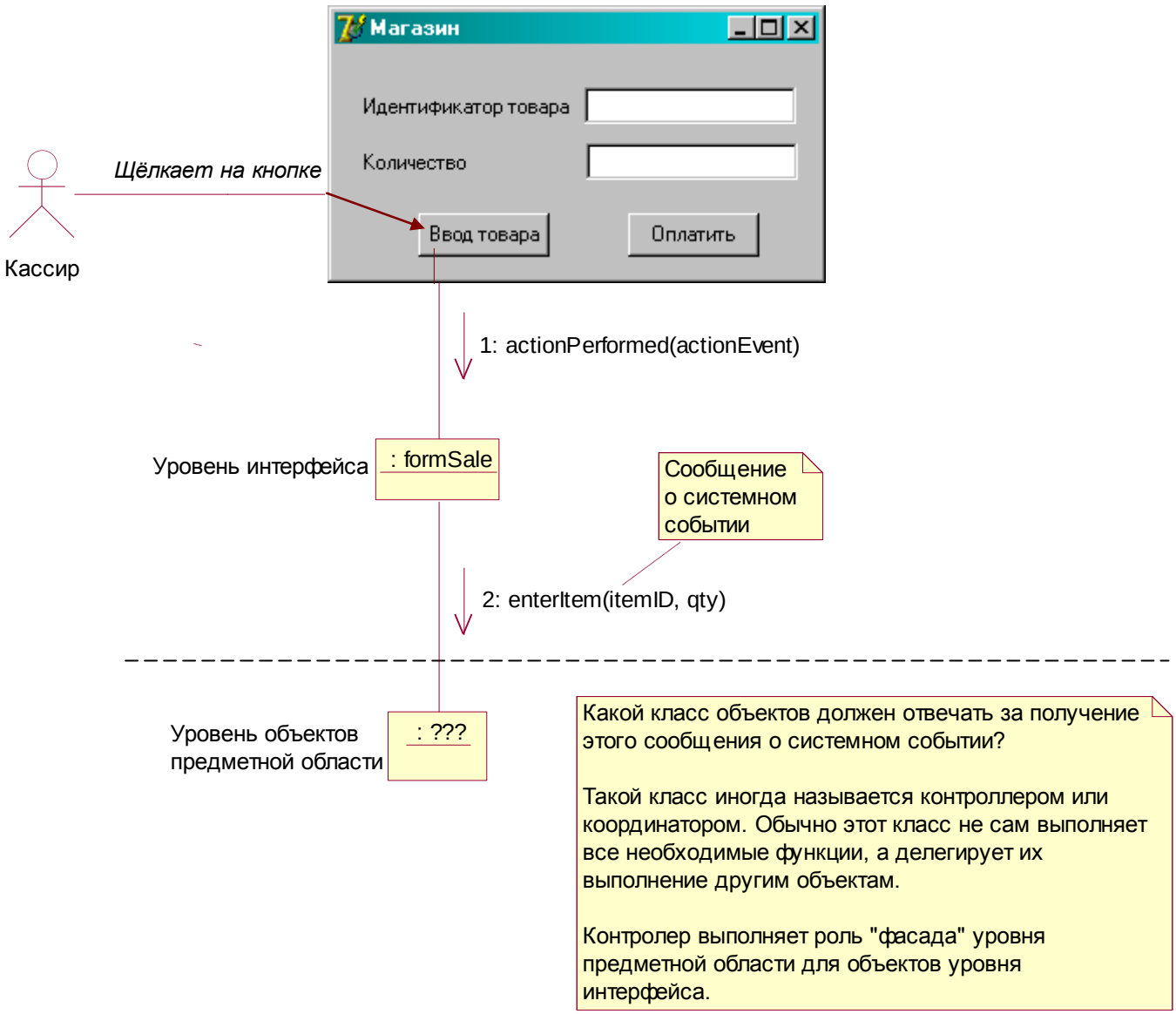


Рис. 0.1. Кто является контроллером для события enterItem?

Согласно шаблону Controller, возможны следующие варианты.

Класс, представляющий всю систему в целом, устройство или подсистему.

Класс, представляющий получателя или искусственный обработчик всех системных событий некоторого сценария прецедента.

В терминах диаграммы взаимодействий должен использоваться один из вариантов, представленных на Рис. 0.2.

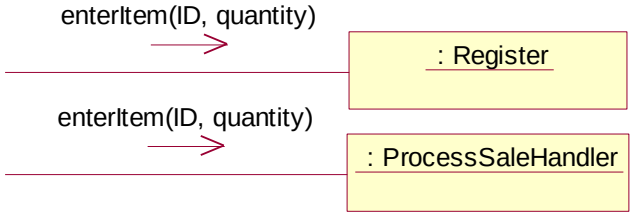


Рис. 0.2 Варианты контроллеров

Выбор наиболее подходящего контроллера определяется зацеплением и связыванием.

Системные операции, идентифицированные в процессе анализа поведения системы, на этапе проектирования присваиваются одному или нескольким классам контроллеров, например Register (Рис. 0.3).

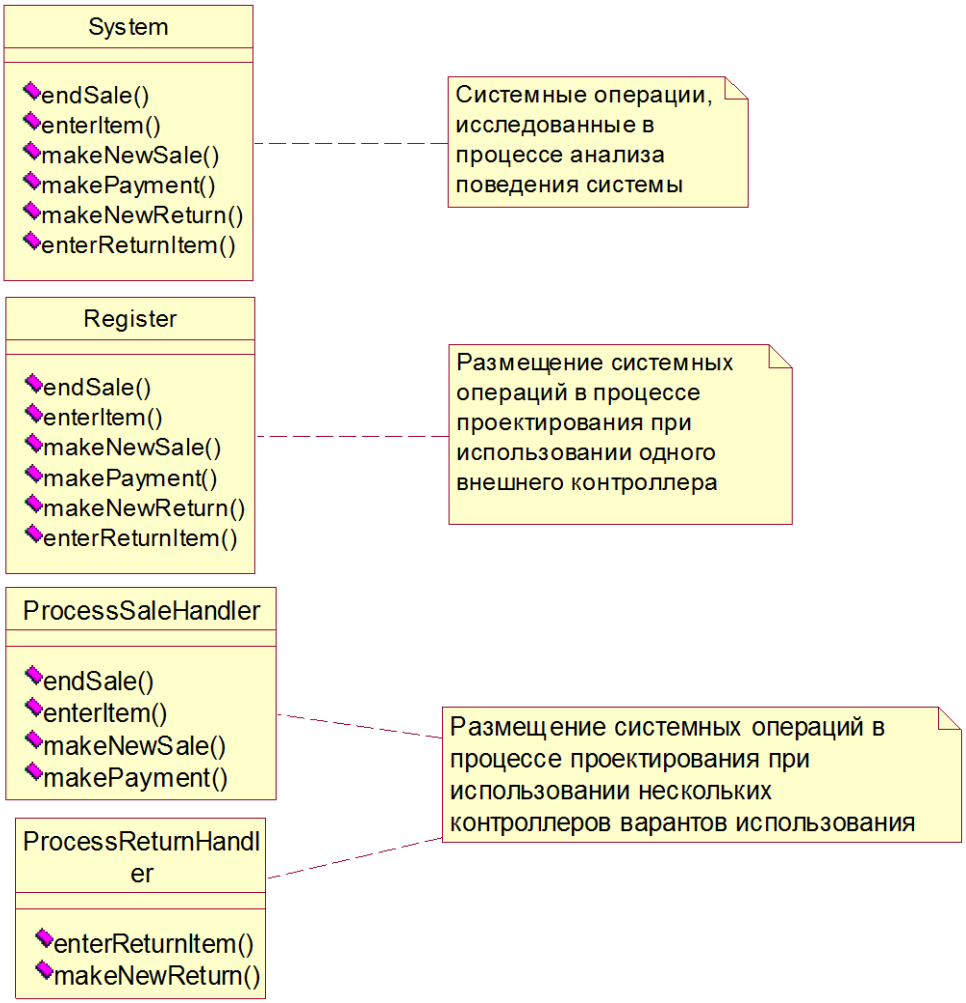


Рис. 0.3 Распределение системных операций

Большинство систем получает внешние события. Обычно они связаны с графическим интерфейсом пользователя. Кроме того, системе могу передаваться внешние сообщения, например, при обработке телекоммуникационных сигналов или сигналов от датчиков в системах управления.

Во всех случаях при использовании объектно-ориентированного подхода для обработки внешних событий применяются контроллеры. Шаблон Controller обеспечивает наиболее типичные проектные решения для этого случая. Как видно из Рис. 0.1, контроллер – это своеобразный вид интерфейса между уровнями предметной области и графического представления.

Типичной ошибкой при создании контроллеров является возложение на них слишком большого числа обязанностей. Обычно, контроллер должен лишь делегировать функции другим объектам и координировать их деятельность, а не выполнять эти действия самостоятельно.

Контроллеры делятся на два типа:

1. *Внешний контроллер (facade controller). Представляет всю "систему", устройство или подсистему.* Основная идея сводится к выбору некоторого класса, имя которого охватывает все слои приложения. *Этот класс обеспечивает главную точку вызова всех служб из интерфейса пользователя и обращения к остальным слоям*. Этот класс может

представлять физический объект, телекоммуникационный переключатель, всю программную часть системы, или любые другие понятия, выбранные разработчиком для представления системы в целом. Внешние контроллеры удобно использовать в том случае, когда существует лишь несколько системных событий или системные сообщения невозможно перенаправить другим контроллерам, наподобие системы обработки сообщений.

2. *Контроллер прецедента (use case controller). Для каждого прецедента должен существовать отдельный контроллер. Контроллеры прецедентов следует использовать в том случае, когда применение внешнего контроллера приводит к слабой степени зацепления или высокой степени связывания*. Контроллер прецедента вводится в том случае, если существующий контроллер слишком "раздувается" при возложении на него дополнительных обязанностей. Контроллеры прецедентов следует применять при наличии большого числа системных событий, распределенных между различными процессами

Преимущества

- *Улучшение условий для повторного использования компонентов*. Применение этого шаблона обеспечивает обработку процессов предметной области на уровне реализации объектов, а не на уровне интерфейса. Обязанности контроллера могут быть технически реализованы в объектах интерфейса, однако в этом случае программный код и логические решения, относящиеся к процессам предметной области, будут жестко связаны с элементами интерфейса, например с окнами. При этом снижается эффективность повторного использования компонентов в других приложениях, поскольку процессы предметной области ограничены рамками интерфейса (например, связаны с оконным объектом), что может оказаться неприемлемым в других приложениях. Делегирование выполнения системных операций специальному контроллеру облегчает повторное использование логики обработки подобных процессов в последующих приложениях.
- *Контроль состояния прецедента*. Иногда необходимо удостовериться, что системные операции выполняются в некоторой определенной последовательности. Например, необходимо гарантировать, чтобы операция makePayment выполнялась только после операции endSale, для чего необходимо накапливать информацию о последовательности событий. Для этой цели удобно использовать контроллер, особенно контроллер прецедента.

Полиморфизм. Чистая синтетика. Перенаправление

Шаблон Polymorphism (полиморфизм)

***Проблема**. Как обрабатывать альтернативные варианты поведения на основе типа? Как создавать подключаемые программные компоненты?*

***Решение.** Если поведение объектов одного типа (класса) может изменяться, обязанности распределяются для различных вариантов поведения с использованием полиморфных операций для этого класса.*

Пример. Система должна поддерживать работу различных внешних систем вычисления налоговых платежей. Каждая система вычисления налоговых платежей имеет свой интерфейс и обладает собственным поведением.

Какие объекты должны отвечать за обработку различных внешних интерфейсов систем вычисления налоговых платежей?

Согласно шаблону Polymorphism, необходимо распределить обязанности по адаптации к различным типам налоговых систем. Для этого можно использовать полиморфную операцию getTaxes (Рис. 0.1).

Рис. 0.1. Использование полиморфизма при адаптации к различным внешним системам вычисления налоговых платежей

Объекты, обеспечивающие адаптацию, – это не внешние калькуляторы, а локальные программные объекты, представляющие внешние системы вычисления налоговых платежей. При отправке сообщения локальному объекту выполняется обращение к внешней системе с использованием ее собственного программного интерфейса.

Каждому методу getTaxes в качестве параметра передается объект Sale, чтобы система вычисления налоговых платежей могла проанализировать продажу. Реализации каждого такого метода отличаются. Например, объект TaxMasterAdapter может адаптировать запросы к системе Tax-Master и т.п.

Полиморфизм – это основной принцип проектирования поведения системы в рамках обработки аналогичных ситуаций. Если обязанности в системе распределены на основе шаблона Polymorphism, то такую систему можно легко модифицировать, добавляя в нее новые вариации. Например, добавив класс для адаптации новой системы вычисления

налоговых платежей со своим полиморфным методом getTaxes, разработчик никак не повлияет на дееспособность уже существующей части системы.

Преимущества

- *С помощью этого шаблона впоследствии можно легко расширять систему, добавляя в нее новые вариации.*
- *Новые реализации можно вводить без модификации клиентской части приложения.*

Шаблон Pure Fabrication (чистая синтетика)

***Проблема**. Какой класс должен обеспечить реализацию шаблонов High Cohesion и Low Coupling или других принципов проектирования, если на базе имеющиеся классов невозможно обеспечить подходящего решения?*

***Решение.** Присвоить группу обязанностей с высокой степенью зацепления искусственному классу, не представляющему конкретного понятия из предметной области, т.е. синтезировать искусственную сущность для поддержки высокого зацепления, слабого связывания и повторного использования.*

Такой класс является продуктом нашего воображения и представляет собой синтетику (fabrication). В идеале, присвоенные этому классу обязанности поддерживают высокую степень зацепления и низкое связывание, так что структура этого синтетического класса является очень прозрачной или чистой (pure). Отсюда и название: Pure Fabrication ("чистая синтетика").

Объектно-ориентированные системы отличаются тем, что программные классы реализуют понятия предметной области, как, например, классы Sale и Customer. Однако существует множество ситуаций, когда распределение обязанностей только между такими классами приводит к проблемам с зацеплением и связыванием, т.е. с невозможностью повторного использования кода.

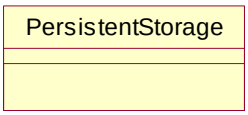
***Пример**. Предположим, необходимо сохранять экземпляры класса Sale в реляционной базе данных. Согласно шаблону Information Expert, эту обязанность можно присвоить самому классу Sale. Однако следует принять во внимание следующие моменты.*

- *Данная задача требует достаточно большого числа специализированных операций, связанных с взаимодействием с базой данных и никак не связанных с понятием самой продажи. Поэтому класс Sale получает низкую степень зацепления.*
- *Класс Sale должен быть связан с интерфейсом реляционной базы данных, поэтому возрастает степень связывания.*
- *Хранение объектов в реляционной базе данных – это достаточно общая задача, решение которой необходимо для многих классов. Возлагая эту обязанность на класс Sale, разработчик не обеспечивает возможности повторного использования этих операций и вынужден дублировать их в других классах.*

Поэтому, несмотря на то, что по логике вещей класс Sale является хорошим кандидатом для выполнения обязанности самосохранения в базе данных, согласно шаблону Information Expert, такое распределение обязанностей приводит к низкой степени зацепления, сильному связыванию и невозможности повторного использования кода. Это именно та безвыходная ситуация, в которой необходимо "синтезировать" нечто новое.

Естественным решением данной проблемы является создание нового класса, ответственного за сохранение объектов некоторого вида на

постоянном носителе, например в реляционной базе данных. Его можно назвать PersistentStorage (Постоянное хранилище). Этот класс является продуктом нашего воображения, что полностью соответствует шаблону Pure Fabrication.



Преимущества

- *При использовании шаблона Pure Fabrication реализуется шаблон High Cohesion*, поскольку обязанности передаются отдельному классу, сконцентрированному на решении специфического набора взаимосвязанных задач.
- *Повышается потенциал повторного использования*, поскольку чисто синтетические классы можно применять в других приложениях.