Реализация вариантов использования на основе шаблонов GRASP

Рассмотрим вариант использования «Оформление продажи (Process Sale)» и следующие системные события: makeNewSale, enterItem, endSale, makePayment.

Для отражения процесса обработки каждого сообщения, соответствующего системному событию, требуется отдельная диаграмма кооперации (Рис. 0.1).

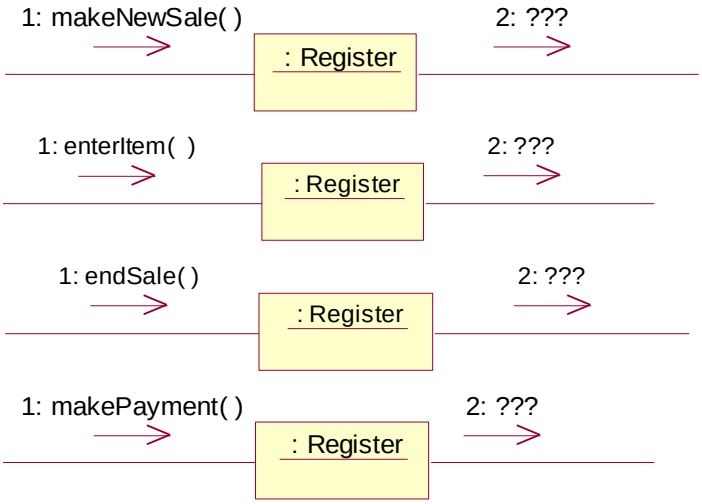

Рис. 0.1 Диаграммы кооперации и обработка сообщений о системных событиях

Если для этого используется диаграмма последовательностей, то все сообщения можно отобразить на одной и той же диаграмме (Рис. 0.2).

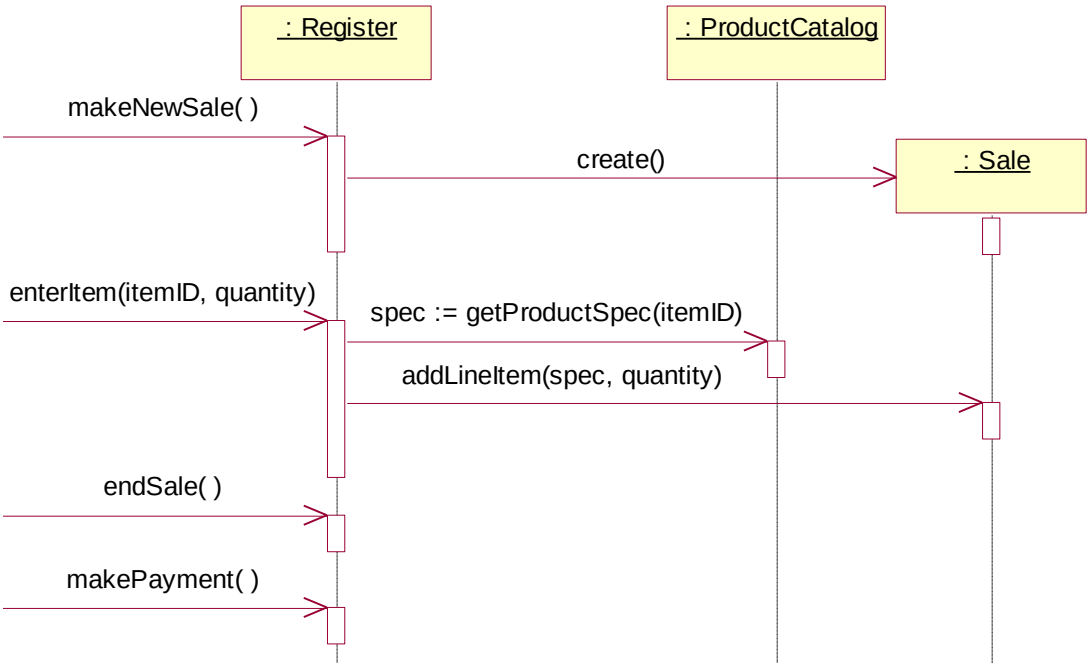

Рис. 0.2. Одна диаграмма последовательностей и обработка сообщений о системных событиях

Однако зачастую диаграмма последовательностей оказывается слишком сложной или большой. В этом случае, как и при использовании диаграмм взаимодействия, можно строить отдельную диаграмму последовательностей для каждого сообщения о системном событии.

Пример реализации вариантов использования

Реализацию варианта использования вполне можно проектировать непосредственно на основе текста из описания варианта использования. Кроме того, описание некоторых системных операций может быть более подробным или содержать некоторые специфические особенности. Ниже приведено описание варианта использования «Оформление продажи».

Табл. 0.1 Вариант использования ВИ1. Оформление продажи

Действия актанта	**Отклик системы**
1. Кассир открывает новую продажу	
2. Кассир вводит идентификатор товара	3. Система записывает наименование товара и выдаёт его описание, цену и общую стоимость. Цена вычисляется на основе набора правил
Кассир повторяет действия, описанные в пп. 2-3 для каждого наименования товара.	4. Система вычисляет общую стоимость покупки с налогом
5. Кассир вводит информацию об оплате	6. Система регистрирует продажу и отправляет информацию о ней внешней бухгалтерской системе и системе складского учёта. Система выдаёт товарный чек

Каждому отклику системы ставится в соответствие системная операция. Системные операции для варианта использования «Оформление продажи» приведены в Табл. 0.2.

Табл. 0.2 Описание системных операций для варианта использования «Оформление продажи»

Операция	**Предусловия**	**Постусловия**
makeNewSale()	Отсутствуют	Создан экземпляр s объекта Sale. Экземпляр Sale связан с объектом Register. Инициализированы атрибуты экземпляра s
enterItem (itemID, quantity)	Инициирована продажа	Создан экземпляр sli класса SalesLineItem Экземпляр sli связан с текущим экземпляром класса Sale Атрибуту sli.quantity присвоено значение quantity Экземпляр sli связан с классом ProductSpecification на основе соответствия идентификатора товара itemID
endSale()	Инициирована продажа	Атрибут Sale.isComplete принял значение true
makePayment()	Инициирована продажа	Создан экземпляр p объекта Payment Атрибут p.amountTendered принял значение amount Экземпляр p связан с текущим экземпляром класса Sale Текущий экземпляр sale связан с экземпляром класса store для его добавления в журнал регистрации продаж

Диаграмма взаимодействия должна обеспечивать выполнение постусловий описания системных операций. Проектное решение принимается на основе шаблонов GRASP.

makeNewSale

Выбор класса-контроллера

В данном случае, так как в системе существует лишь несколько системных операций, в качестве контроллера можно выбрать класс Register.

Создание нового экземпляра объекта Sale

Для создания программного объекта Sale воспользуемся шаблоном Creator, согласно которому обязанность по созданию новых экземпляров делегируется классу, содержащему, агрегирующему или записывающему информацию о создаваемых классах.

Проанализировав модель предметной области, выясняется, что запись информации о продажах может выполнять объект Register.

Кроме того, после создания объекта Sale необходимо создать пустую коллекцию (контейнер наподобие List в Java) для записи всех добавляемых впоследствии экземпляров SalesLineItem. Эта коллекция будет поддерживаться экземпляром Sale, который, согласно шаблону Creator, является наилучшим кандидатом для ее создания.

Таким образом, объект Register создает экземпляры Sale, а Sale, в свою очередь, создает пустой контейнер, представленный на диаграмме взаимодействия как сложный объект (Рис. 0.3).

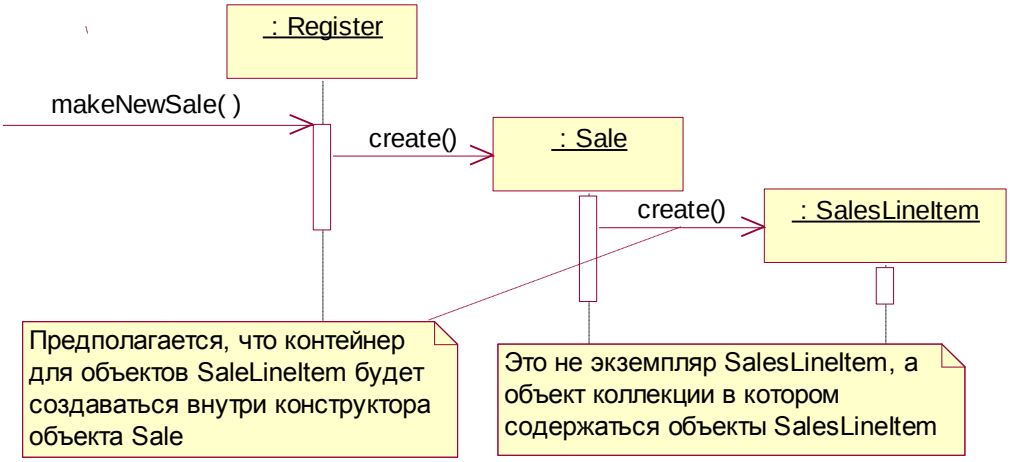

Рис. 0.3. Создание экземпляров Sale и сложного объекта

enterItem

Выбор класса-контроллера

Согласно шаблону Controller, в качестве контроллера будем использовать класс Register, как и для операции makeNewSale.

Создание экземпляров SalesLineItem

Анализируя модель предметной области, приходим к выводу, что объект Sale содержит объекты SalesLineItem, в связи с чем именно этому классу логично поручить создание экземпляров SalesLineItem.

В таком случае со временем объект Sale будет связан с новым экземпляром коллекции продаваемых товаров. Из постусловий следует, что при создании экземпляра SalesLineItem необходимо указывать количество единиц покупаемого товара, поэтому данное значение необходимо передавать из объекта Register объекту Sale, который, в свою очередь, передаст его в качестве параметра сообщения create.

Таким образом, согласно шаблону Creator, для создания объекта SalesLineItem объекту Sale передается сообщение makeLineItem. Объект Sale создает экземпляр SalesLineItem, а затем хранит этот новый экземпляр в своем постоянном контейнере.

Параметрами сообщения makeLineItem являются количество единиц товара quantity и спецификация товара ProductSpecification, соответствующая коду товара itemID.

Нахождение ProductSpecification

Экземпляр SalesLineItem необходимо связать со спецификацией ProductSpecification, соответствующей коду данного товара itemID. Это означает, что необходимо уметь по коду товара находить значение ProductSpecification.

Кто должен отвечать за нахождение значения ProductSpecification на основе кода товара itemID?

Согласно шаблону Expert, класс ProductCatalog является хорошим кандидатом для реализации обязанности поиска, поскольку обладает полной информацией обо **всех** объектах ProductSpecification.

Это можно реализовать, например, с помощью метода getSpecification.

Согласно приведенным выше рассуждениям, можно построить диаграмму взаимодействия, отражающую распределение обязанностей и способы взаимодействия между объектами (Рис. 0.4).

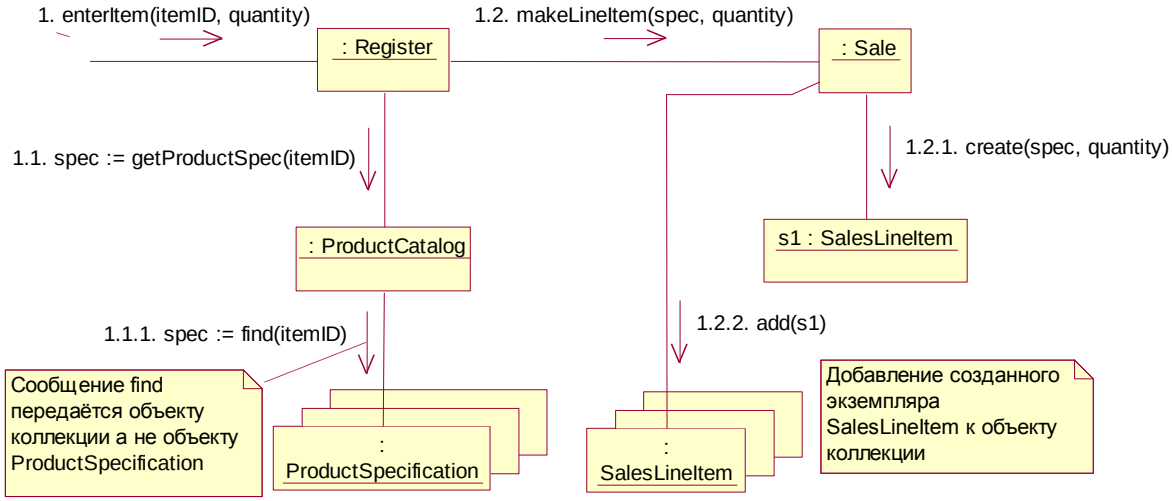

Рис. 0.4. Диаграмма взаимодействия для системной операции enterItem

Независимое от языка

Сообщения сложным объектам

Отправка сообщения сложному объекту в UML интерпретируется как передача сообщения объекту-контейнеру, а не каждому элементу набора объектов. Это особенно очевидно для таких операций, как find и add.

endSale

Выбор класса-контроллера

В соответствии с шаблоном Controller, в качестве контроллера будем использовать класс Register, как и для системной операции enterItem.

Установка значения атрибута Sale.isComplete

Какой класс должен отвечать за установку атрибута isComplete объекта Sale равным значению true?

Согласно шаблону Expert, эту обязанность следует возложить на класс Sale, поскольку он владеет атрибутом isComplete. Тогда класс Register будет отправлять объекту Sale сообщение becomeComplete, требующее установки этого атрибута в значение true.

Sale.getTotal

Не каждая диаграмма взаимодействия начинается с сообщения о системном событии. Она может начинаться с любого сообщения, для которого необходимо показать виды взаимодействия между объектами.

Диаграмма взаимодействия представлена на Рис. 0.5. Сначала объекту Sale передается сообщение getTotal. Затем объект Sale отправляет сообщение getSubtotal каждому связанному с ним экземпляру SalesLineItem. В свою очередь, каждый экземпляр SalesLineItem отправляет сообщение getPrice связанному с ним объекту ProductSpecification.

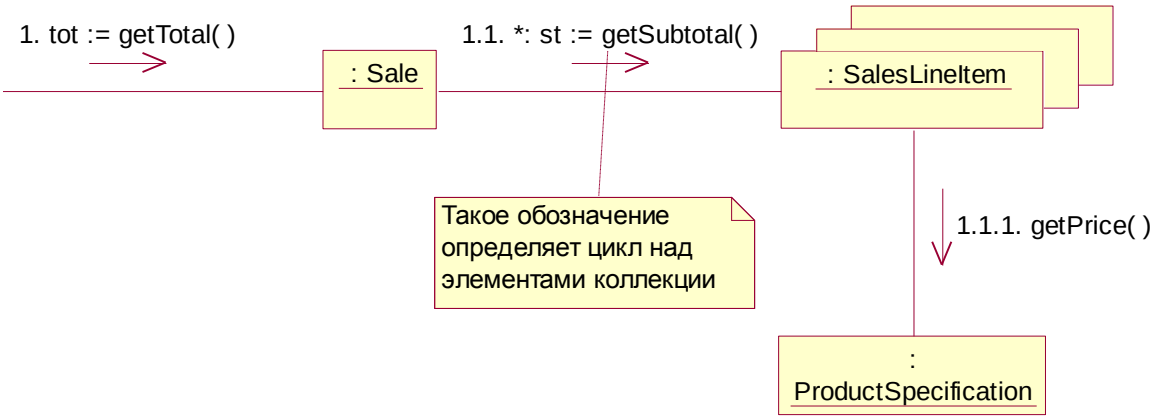

Рис. 0.5 Диаграмма взаимодействия для метода Sale.getTotal

Поскольку арифметические действия обычно не изображаются с помощью сообщений, их можно проиллюстрировать путем добавления на диаграмму дополнительной информации в виде ограничений или описаний алгоритма.

Какой класс должен передавать сообщение getTotal объекту Sale? Скорее всего, это должен быть объект уровня представления, например объект JFrame Java.

makePayment

Системная операция makePayment выполняется при вводе кассиром внесенной за покупку суммы. Проектное решение должно обеспечить выполнение постусловий системной операции makePayment.

Создание экземпляра Payment

При распределении обязанностей, связанных с созданием новых экземпляров, необходимо применять шаблон Creator.

Какой класс записывает, агрегирует, наиболее активно использует или содержит объекты Payment? Одним из кандидатов на выполнение этой обязанности может быть класс Register, поскольку он по логике записывает сведения о платежах. При таком проектном решении сокращается разрыв между терминологией предметной области и именами программных классов. Еще одним кандидатом может выступать класс Sale, поскольку он наиболее активно использует информацию объекта Payment.

Таким образом, имеем двух кандидатов.

- Register
- Sale

Это приводит к необходимости применения главной идеи проектирования.

При наличии двух альтернативных вариантов проектирования их следует рассматривать с точки зрения связывания и зацепления, а также, по возможности, будущих изменений. Выбор целесообразно остановить на

варианте с хорошими показателями в области связывания и зацепления, обладающем высокой устойчивостью к возможным изменениям в будущем

Рассмотрим каждый из этих вариантов "с точки зрения" шаблонов High Cohesion и Low Coupling. Если экземпляр объекта Payment создается классом Sale, то работа (обязанности) класса Register облегчается. Кроме того, классу Register не нужно знать о существовании экземпляра Payment, поскольку он может записывать его опосредованно через объект Sale. Это обеспечит низкий уровень связывания объекта Register. Полученная диаграмма представлена на Рис. 0.6.

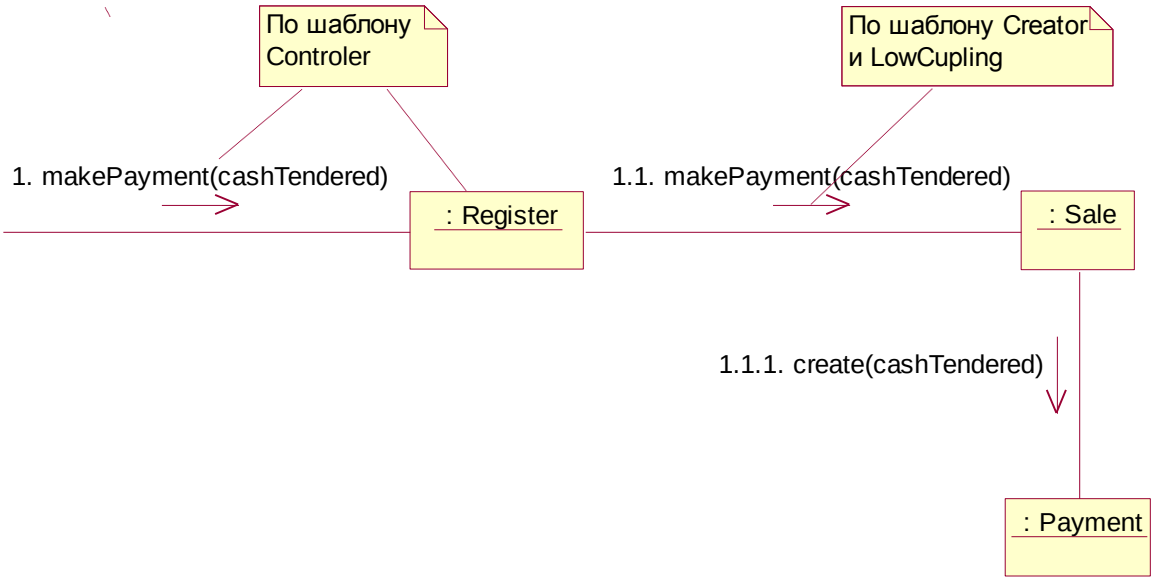

Рис. 0.6. Диаграмма взаимодействия для операции Register.makePayment

Регистрация покупки

Какой из классов обладает всей информацией о продажах и может выполнять регистрацию?

С точки зрения сокращения разрыва между терминологией предметной области и именами программных классов лучше всего, чтобы всей необходимой информацией обладал объект Store (Магазин), поскольку он связан с финансовой деятельностью предприятия (Рис. 0.7).

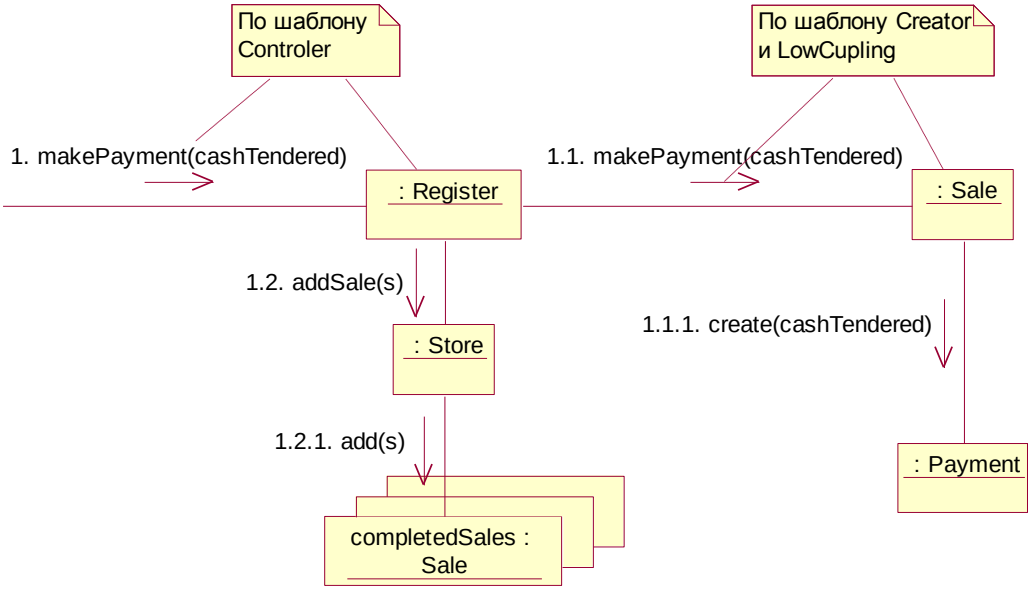

Рис. 0.7. Регистрация совершенной покупки

Вычисление причитающейся сдачи

Теперь необходимо реализовать процесс вычисления причитающейся сдачи, сумма которой должна отображаться на экране и печататься на чеке.

Как обычно, если задача не сводится к выбору контроллера или к созданию новых объектов (а в данном случае это именно так), при распределении обязанностей необходимо воспользоваться шаблоном Information Expert и ответить на следующий вопрос.

Какой из классов отвечает за вычисление суммы сдачи (баланса торговой операции)?

Для вычисления баланса необходимо знать общую стоимость покупки и внесенную покупателем сумму. Следовательно, при решении этой проблемы в качестве частичных экспертов должны выступать классы Sale и Payment.

Если основная обязанность по вычислению баланса возлагается на класс Payment, то для него следует обеспечить видимость класса Sale, с помощью которого необходимо получить общую стоимость покупки. Такой подход повышает уровень связывания классов в проекте и не соответствует основному принципу шаблона Low Coupling.

Если же основная обязанность по вычислению баланса возлагается на класс Sale, то для него следует обеспечить видимость класса Payment, от которого необходимо получить внесенную покупателем сумму. Поскольку класс Sale является создателем экземпляров Payment, такая видимость уже обеспечена, и данный подход не повышает уровень связывания классов в проекте, что соответствует основному принципу шаблона Low Coupling и, следовательно, является более предпочтительным.

Таким образом, диаграмма взаимодействия приобретает следующий вид (Рис. 0.8).

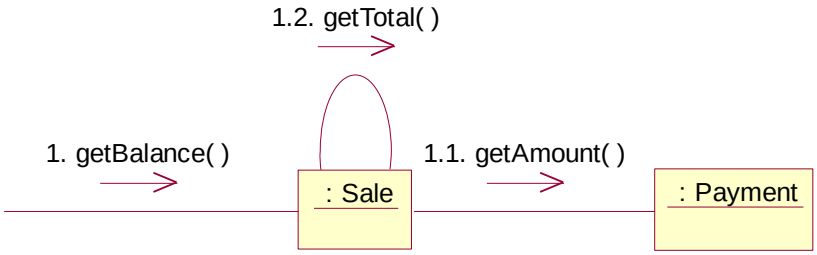

Рис. 0.8. Диаграмма взаимодействия для операции Sale.getBalance

Создание диаграммы взаимодействия для системной операции startup

Практически все системы включают некоторую исходную системную операцию, связанную с запуском приложения.

Хотя системная операция startup выполняется одной из первых, ее диаграмму взаимодействия разрабатывают после рассмотрения всех остальных системных операций. При таком подходе гарантируется, что к моменту разработки системной операции startup известна вся существенная информация, связанная с инициализацией системы и необходимая для поддержки диаграмм взаимодействия последующих системных операций.

Как запускаются приложения

Операция startup – это абстрактное представление этапа инициализации при запуске приложения. Чтобы разобраться, как разрабатывать диаграмму взаимодействия для такой операции, необходимо понять основные принципы запуска приложения. Способ запуска приложения и инициализации данных зависит от операционной системы и выбранного языка программирования.

В любом случае, *как правило, разрабатывается исходный объект предметной области (initial domain object), который создается первым среди других объектов предметной области*.

После создания исходного объекта предметной области ему в обязанность вменяется создание дочерних объектов предметной области. Например, если в качестве исходного объекта предметной области выбран объект Store, то он может отвечать за создание объекта Register.

Способ создания исходного специального объекта зависит от выбранной объектной технологии. Например, в приложении Java для его создания можно использовать метод main.

```
public class Main {
  public static void main (String [] args) {
    // Store - исходный объект предметной области.
    //Он отвечает за создание
    //других объектов предметной области.
    Store store = new Store();
    Register register = store.getRegister();
   ProcessSaleJFrame frame = new ProcessSaleJFrame(register);
  }
}
```

Интерпретация системной операции startup

Из приведенных рассуждений следует, что системная операция startup – это зависимая от языка программирования абстракция. На этапе проектирования определяется, где создается исходный объект и нужно ли ему передавать управление процессом. Обычно при наличии графического интерфейса пользователя исходный объект предметной области не получает управления процессом. Ему передается управление лишь при отсутствии интерфейса пользователя.

Диаграмма взаимодействия для операции startup представляет события, происходящие после создания исходного объекта предметной области, а также (не обязательно) события, происходящие при передаче управления этому объекту. Она не включает никаких предварительных или последующих видов деятельности, связанных с объектами уровня графического интерфейса пользователя, если таковой существует.

Итак, операцию startup можно интерпретировать следующим образом.

1. *На одной диаграмме взаимодействия отображается передача сообщения create ( ) для создания исходного объекта предметной области.*
2. *(Дополнительно.) Если исходному объекту передается управление процессом, то на второй диаграмме взаимодействия отображается передача сообщения run ( ) (или эквивалентного ему сообщения) исходному объекту.*

Операция startup в нашем примере

Системная операция startup выполняется при включении менеджером питания POS-системы и загрузке программной части системы. Предположим, исходному объекту предметной области управление не передается. Оно передается на уровень графического интерфейса пользователя (например, JFrame Java) после создания исходного объекта предметной области. Тогда диаграмма взаимодействия для системной операции startup должна содержать лишь передачу сообщения create () для создания исходного объекта.

Выбор исходного объекта предметной области

Какой класс следует выбрать в качестве исходного объекта предметной области?

В качестве исходного объекта предметной области выбирается класс, максимально приближенный к корню иерархии агрегации объектов предметной области. В качестве такого объекта можно выбрать внешний контроллер, такой как Register, или любой другой объект, содержащий все или большую часть других объектов предметной области, например Store.

Выбор между этими объектами осуществляется на основе шаблонов High Cohesion и Low Coupling. В рассматриваемом приложении выберем в качестве исходного объекта класс Store.

Проектное решение: store.create ()

Из рассмотренных выше диаграмм взаимодействия следует необходимость инициализации для следующих объектов.

- *Необходимо создать экземпляры объектов Store, Register, ProductCatalog и ProductSpecification.*
- *Объект ProductCatalog необходимо связать с объектом ProductSpecification.*
- *Объект Store необходимо связать с объектом ProductCatalog.*
- *Объект Store необходимо связать с объектом Register.*
- *Объект Register необходимо связать с объектом ProductCatalog.*

На Рис. 0.9 показано соответствующее проектное решение.

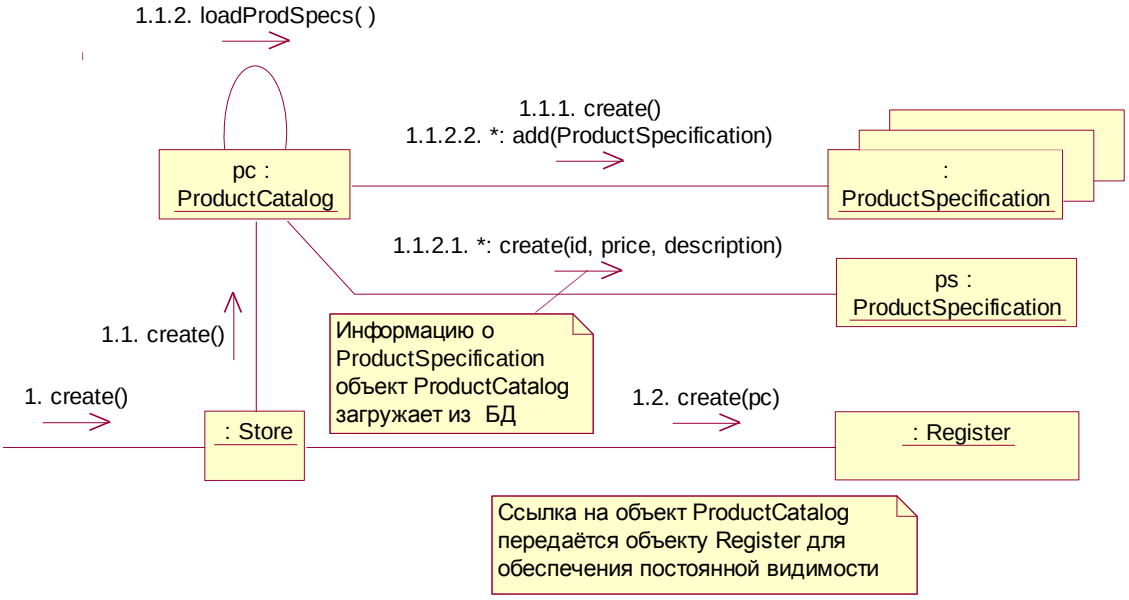

Рис. 0.9. Создание исходного объекта предметной области и последующих объектов