Объектно-ориентированное проектирование ПС (часть 2)

Лекция 8

Шаблон Controller

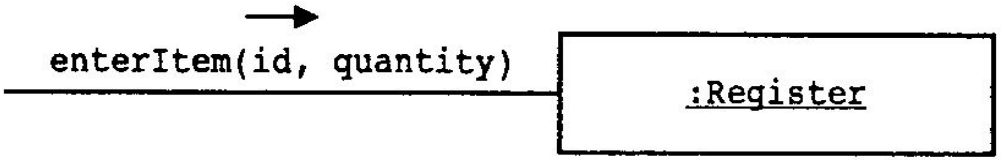
- ***Проблема*** Кто должен отвечать за обработку системных сообщений?
- ***Решение*** Обязанности по обработке системных сообщений назначаются классу, который:
  - представляет систему в целом;
  - представляет сценарий некоторого прецедента, в рамках которого обрабатываются системные сообщения

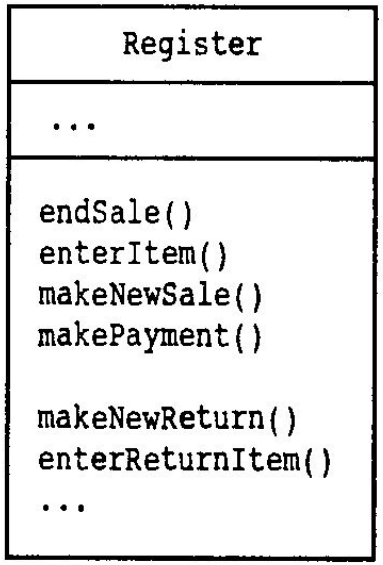
Контроллеры

- Классы, обязанности которых состоят в обработке системных сообщений называются *классами контроллера*
- Классы контроллера не относятся к интерфейсу пользователя
- В рассматриваемом примере возможны два варианта решения

1-Й ВАРИАНТ

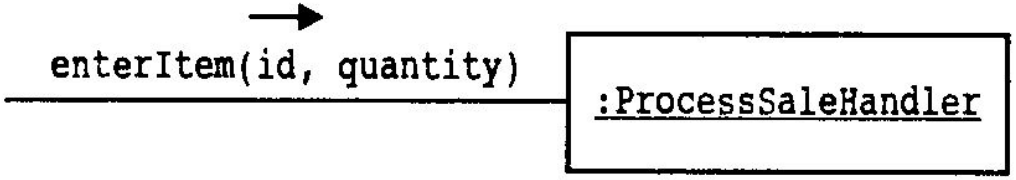
Все системные операции выполняются одним внешним контроллером

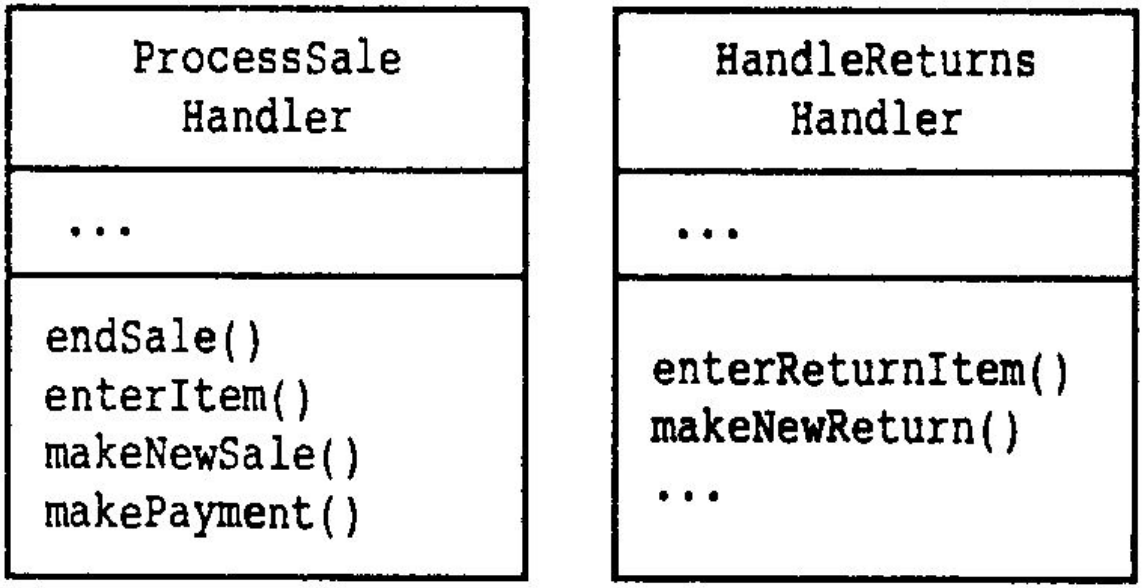




2-Й ВАРИАНТ

Системные операции распределены между несколькими контроллерами прецедента





Выбор варианта

- Выбор между вариантом использования внешнего контроллера (**facade controller**) и вариантом контроллеров прецедентов определяется, в основном требованиями соблюдения малой степени сцепления и высокой связности

Реализация прецедента показывает, как определенный прецедент представляется в рамках модели проектирования в терминах взаимодействующих объектов

РЕАЛИЗАЦИЯ ПРЕЦЕДЕНТА «ОФОРМЛЕНИЕ ПРОДАЖИ»

Диаграммы взаимодействия

- Реализация прецедентов представляется в виде диаграмм взаимодействия, начинающихся с соответствующего системного сообщения
- Диаграммы взаимодействия включают процессы передачи сообщений между программными объектами, участвующими в реализации прецедента

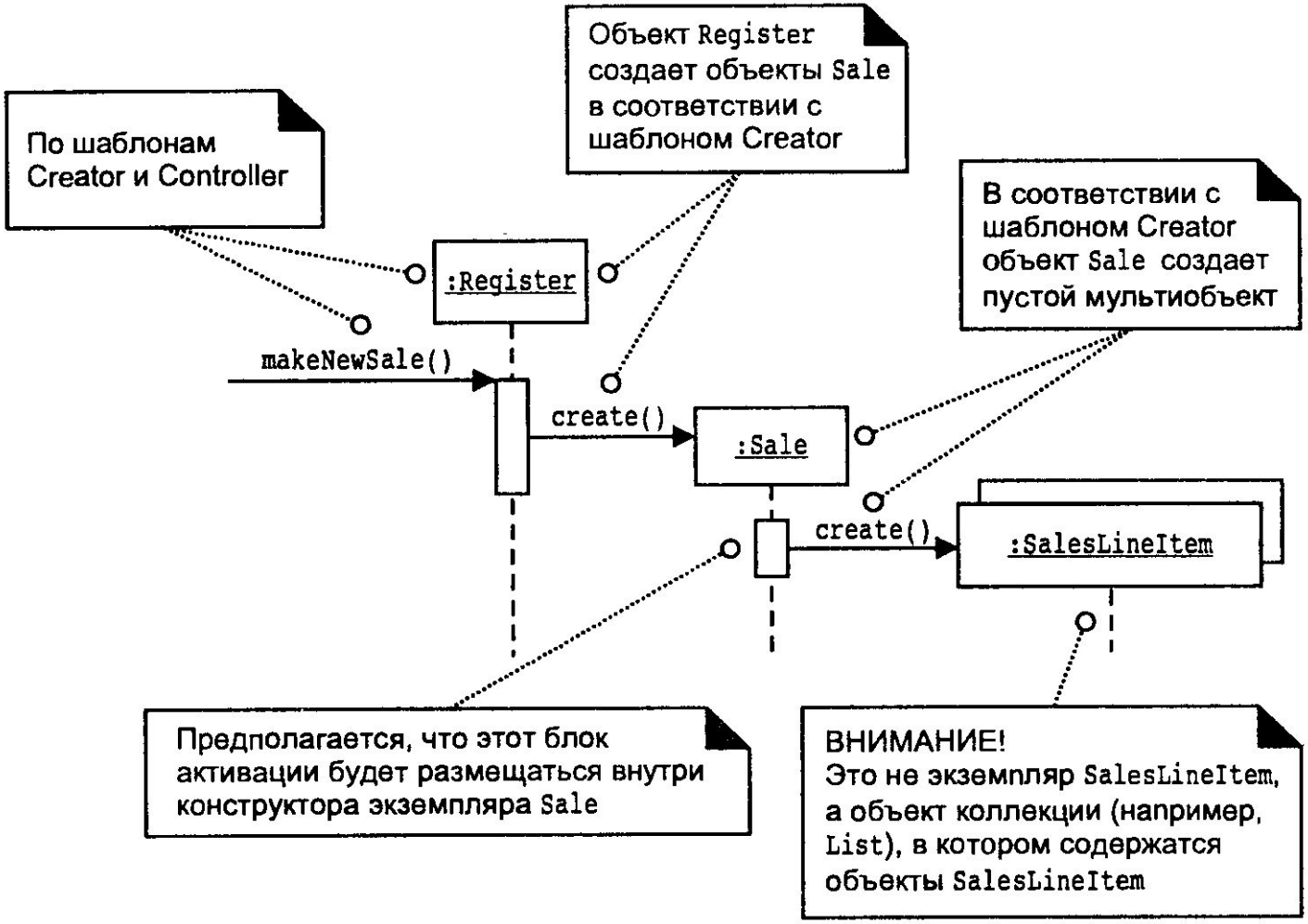
Прецедент «Оформление продажи»

- На текущей итерации рассмотрены следующие системные события и соответствующие операции:
  - Начать оформление покупки (makeNewSale)
  - Ввести данные товара (enterItem)
  - Завершить оформление покупки (endSale)
  - Оформить платеж (makePayment)
- Для наглядности каждая операция будет представлена отдельной диаграммой

Операция makeNewSale

- **Операция**
  - makeNewSale()
- **Ссылки**
  - Прецеденты: Оформление продажи
- **Предусловия**
  - Отсутствуют
- **Постусловия**
  - Создан экземпляр s класса Sale
  - Экземпляр s связан с объектом класса Registor
  - Установлены атрибуты экземпляра s

Диаграмма последовательности



Операция enterItem

- **Операция**
  - enterItem(iID: ItemID, quantity: integer)
- **Ссылки**
  - Прецеденты: Оформление продажи
- **Предусловия**
  - Инициирована продажа
- **Постусловия**
  - Создан экземпляр sli класса SalesLineItem
  - Экземпляр sli связан с объектом класса Sale
  - Атрибуту sli.quantity присвоено значение quantity
  - Экземпляр sli связан с классом ProductSpecification

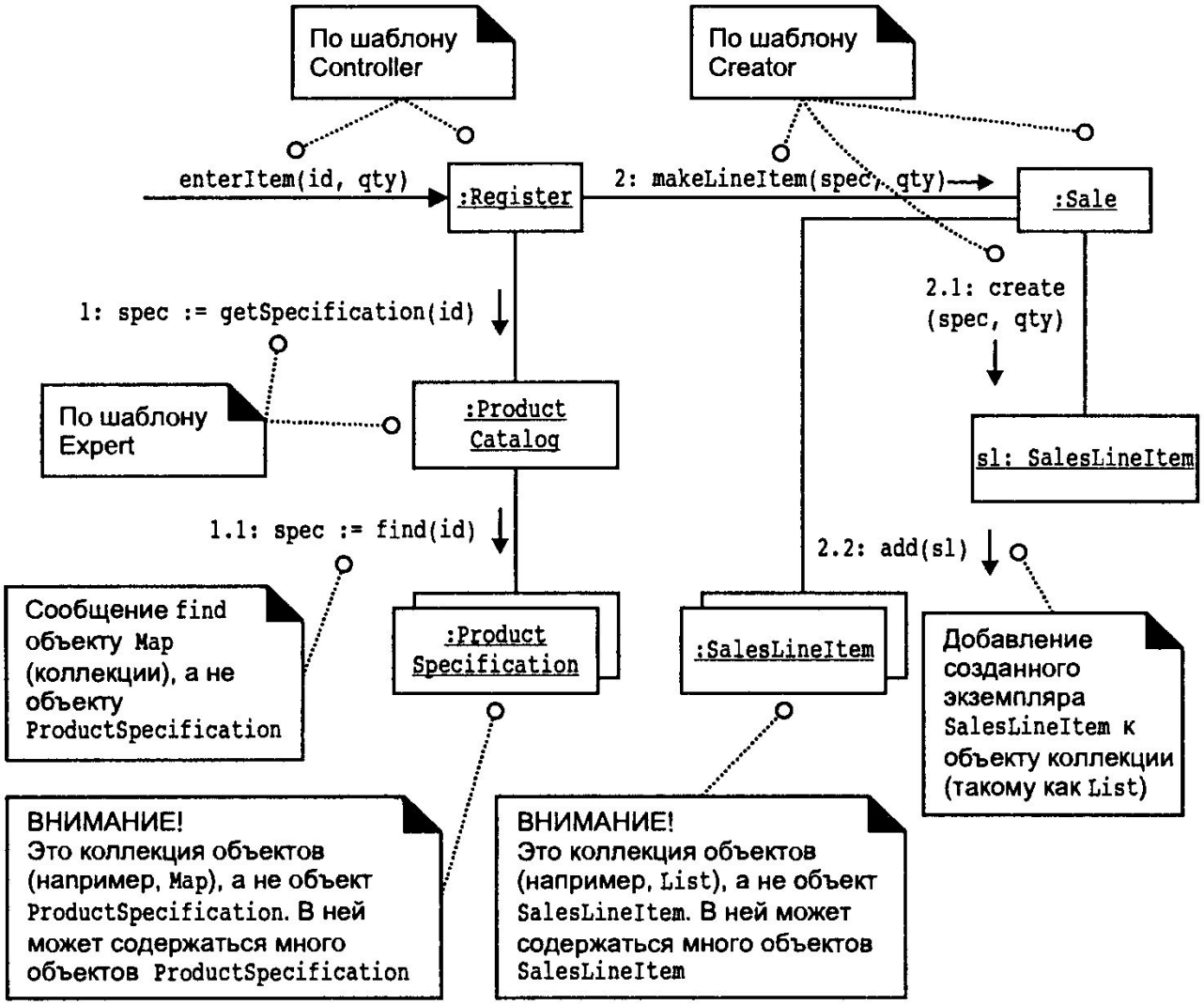
Обработка enterItem

- Ранее было установлено, что объект Sale должен создаваться объектом Registor
- Объект Sale создает объект SalesLineItem и добавляет его в свой контейнер
- Затем необходимо получить описание товара ProductSpecification для экземпляра SalesLineItem
- Кто должен поставлять эту информацию?

Класс ProductCatalog

- На основе анализа предметной области и в соответствии с шаблоном Expert эта обязанность должна быть назначена классу ProductCatalog
- Этот класс должен содержать коллекцию экземпляров класса ProductSpecification
- Запрос на получение описания товара должен отправлять объект Register

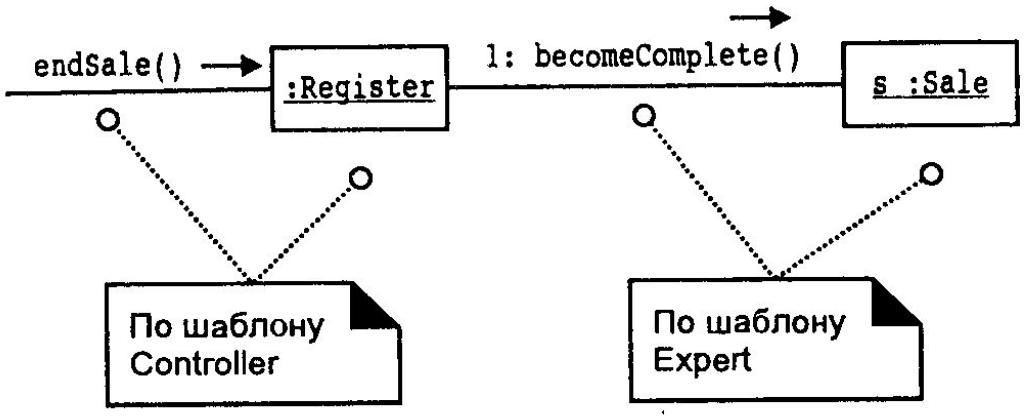
Диаграмма кооперации



Операция endSale

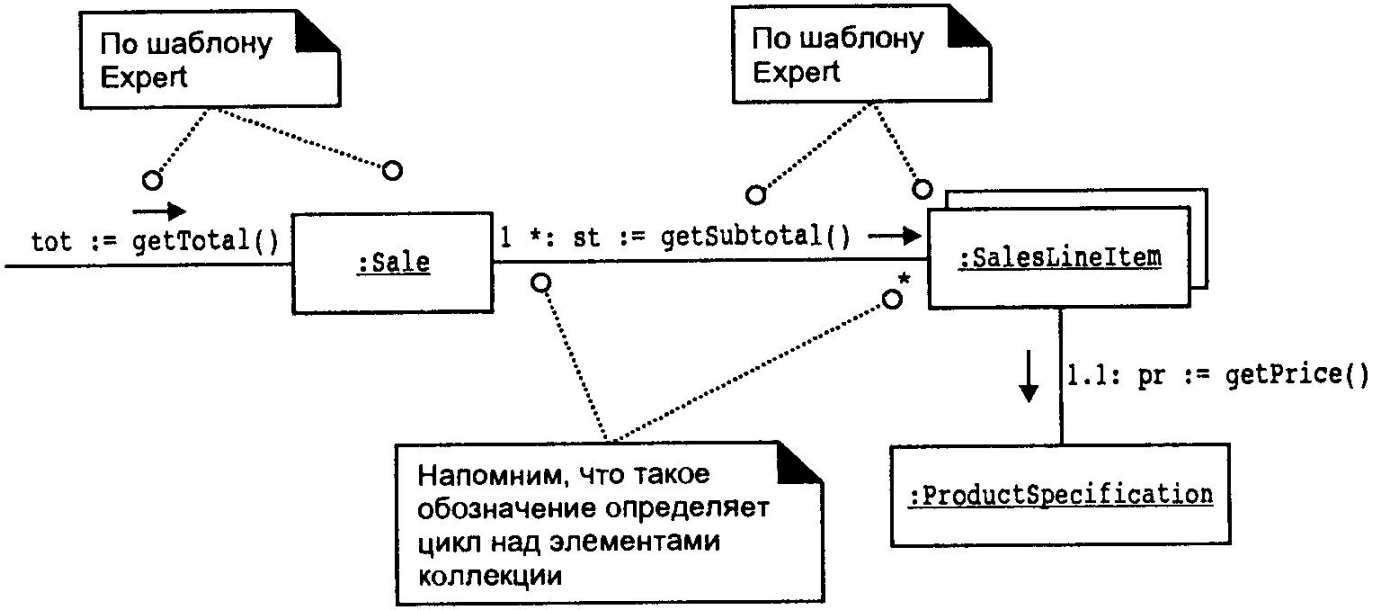
- **Операция**
  - endSale()
- **Ссылки**
  - Прецеденты: Оформление продажи
- **Предусловия**
  - Инициирована продажа
- **Постусловия**
  - Атрибуту isComplete экземпляра класса Sale присвоено значение true

Диаграмма кооперации



Вычисление общей стоимости

- В соответствии с описанием данного прецедента после завершения оформления продажи система должна вычислить общую стоимость покупки



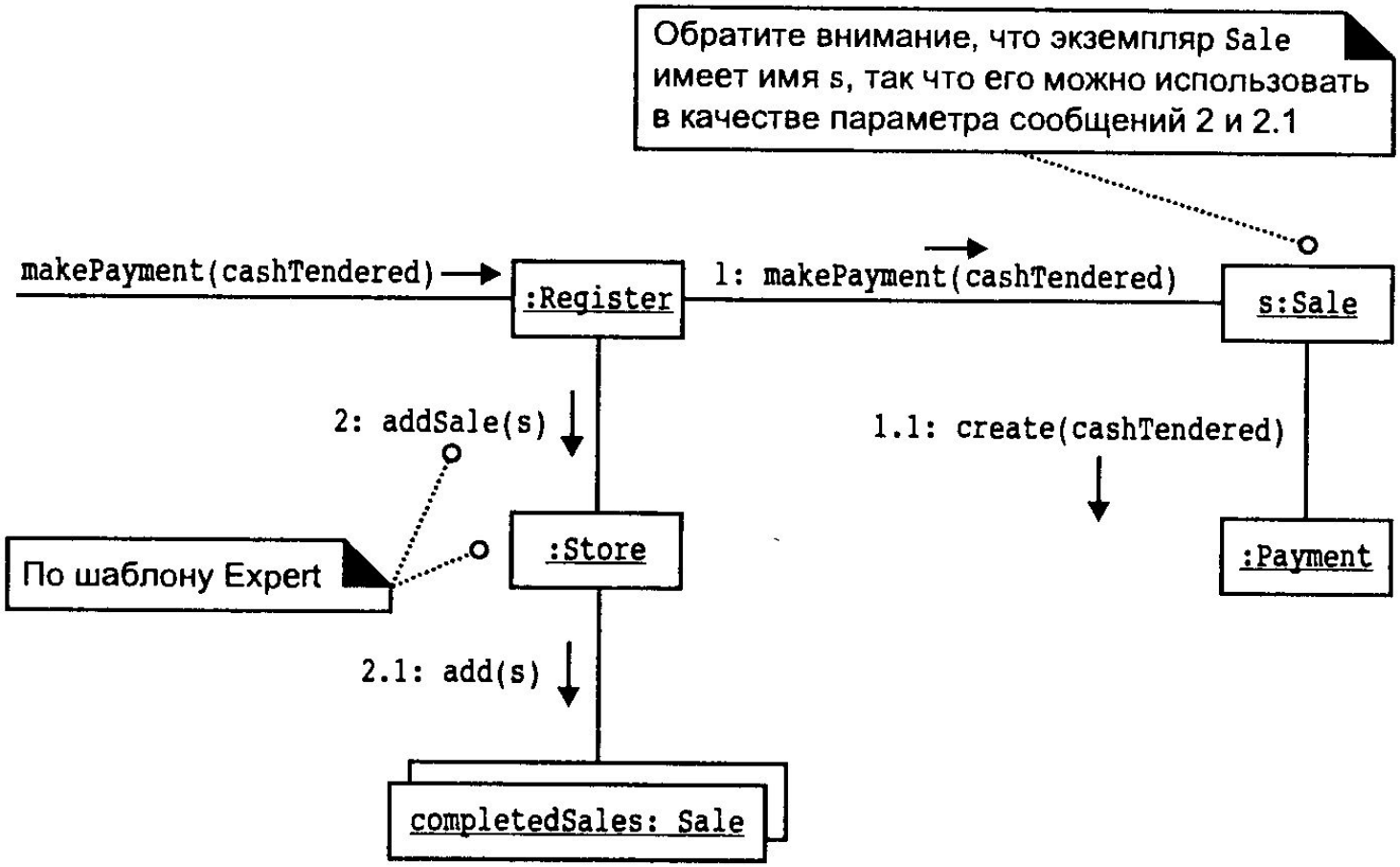
Операция makePayment

- **Операция** makePayment(amount: Money)
- **Ссылки** Прецеденты: Оформление продажи
- **Предусловия** Инициирована продажа
- **Постусловия**
  - Создан экземпляр p класса Payment
  - Экземпляр p связан с объектом класса Sale
  - Атрибуту amountTendered присвоено значение amount
  - Экземпляр Sale связан с экземпляром класса Store для его добавления в журнал продаж

Создание Payment

- В соответствии с шаблоном Creator и с учетом требований слабого сцепления и сильной связности обязанность создавать экземпляры класса Payment назначена объекту класса Sale, активно использующему информацию экземпляра Payment

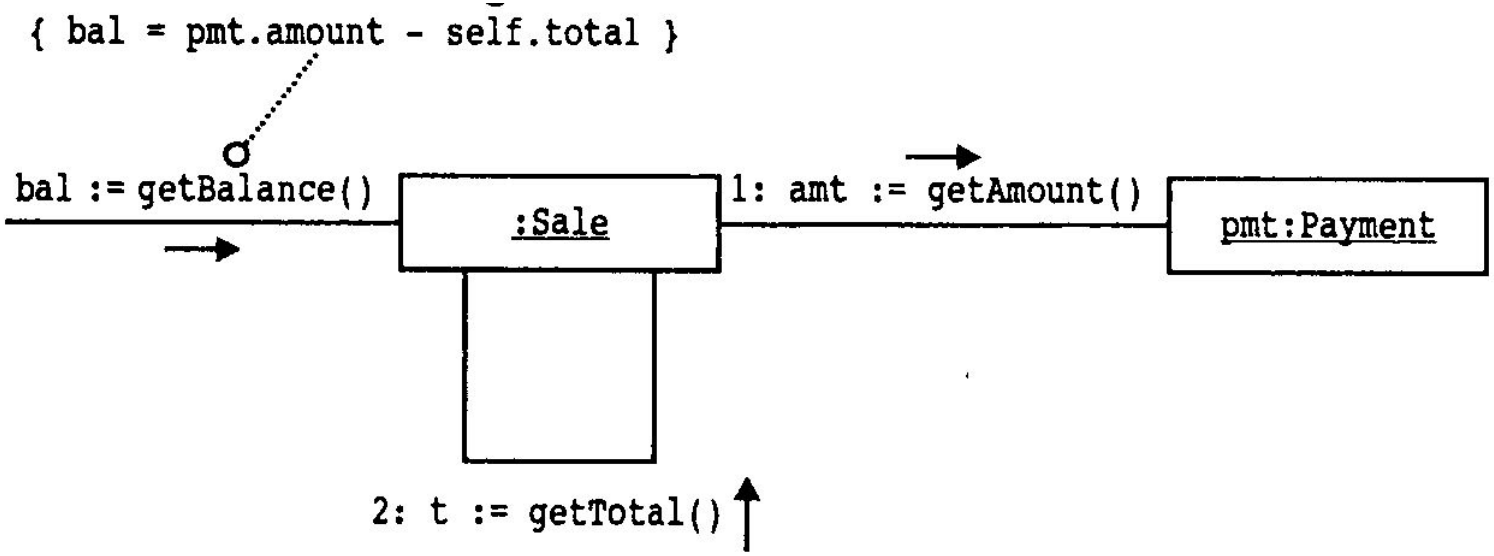
Регистрация покупки



Вычисление сдачи

- В роли частичных экспертов могут выступать объекты классов:
  - Sale (информация о полной стоимости)
  - Payment (информация о внесенной покупателем сумме)
- Экземпляр класса Sale является создателем объекта класса Payment и может запросить у него сумму платежа

Вычисление сдачи



Практически все системы включают прецедент «Запуск системы» и системную операцию, связанную с запуском приложения

РЕАЛИЗАЦИЯ ПРЕЦЕДЕНТА «ЗАПУСК СИСТЕМЫ»

Запуск приложения

- Представляется системной операцией StartUp, которая является абстракцией реального процесса загрузки и инициализации приложения
- Диаграмма взаимодействия для операции StartUp строится в последнюю очередь, когда уже известна информация об основных системных операциях

Запуск приложения

- Должен быть выбран исходный объект предметной области; соответствующий программный объект создается первым
- Этот объект наделяется обязанностью создания других объектов, наличие которых необходимо для начала работы приложения, а также их инициализации

Выбор исходного объекта

- В качестве исходного выбирается объект, наиболее приближенный к корню иерархии объектов
- В нашем случае это может быть Register либо Store
- Исходя из требования высокой степени связности, выберем объект Store

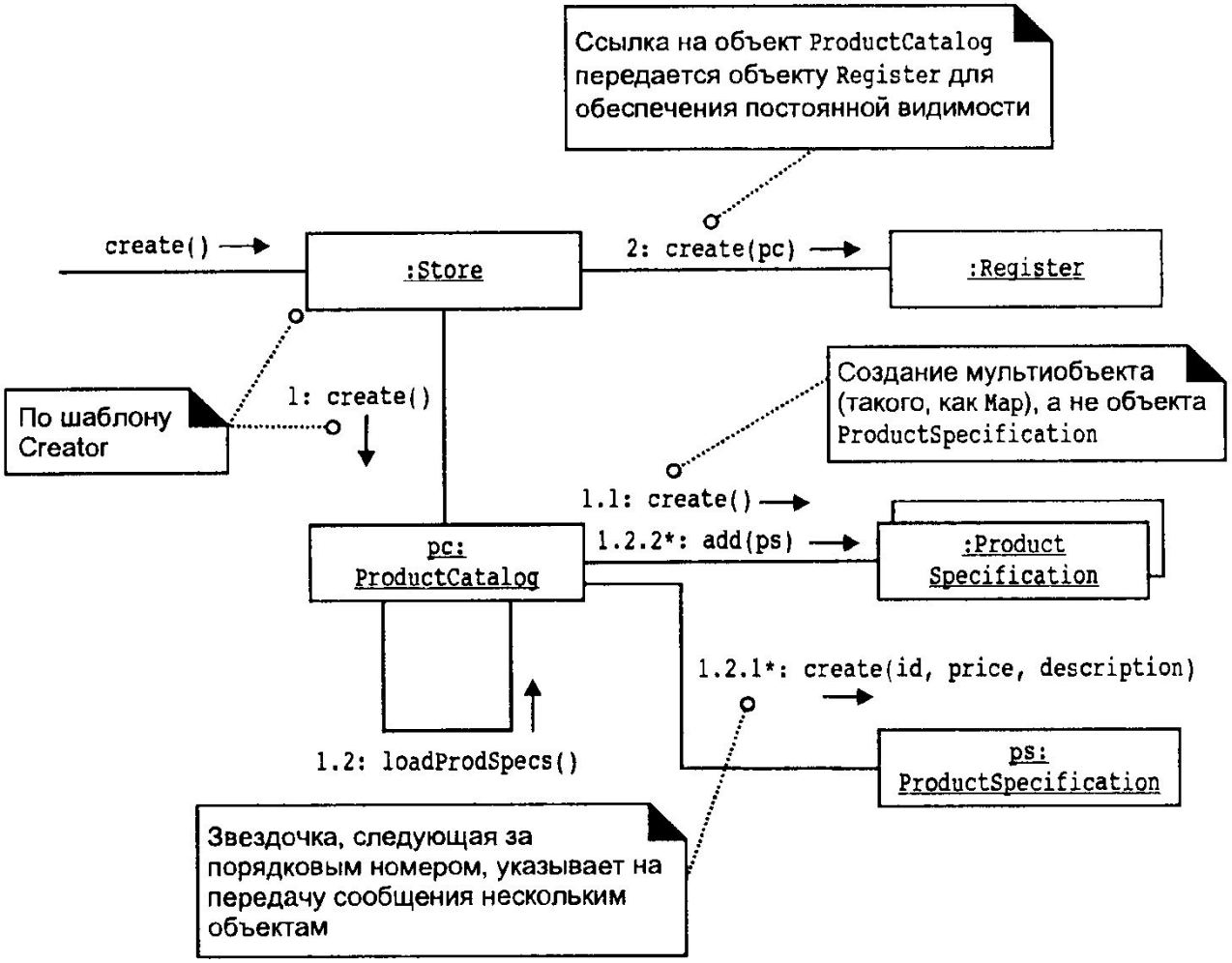
Операция StartUp

- **Операция** StartUp()
- **Ссылки** Прецеденты: Запуск системы
- **Предусловия** Отсутствуют
- **Постусловия**
  - Создан экземпляр класса Store
  - Создан экземпляр pc класса ProductCatalog
  - Экземпляр pc связан с экземпляром класса Store
  - Создан экземпляр ps класса ProductSpecification
  - Экземпляр ps связан с объектом класса Product Catalog

Операция StartUp

- **Постусловия (продолжение)**
  - Создан экземпляр класса Register
  - Экземпляр класса Store связан с экземпляром класса Register
  - Экземпляр pc связан с экземпляром класса Register

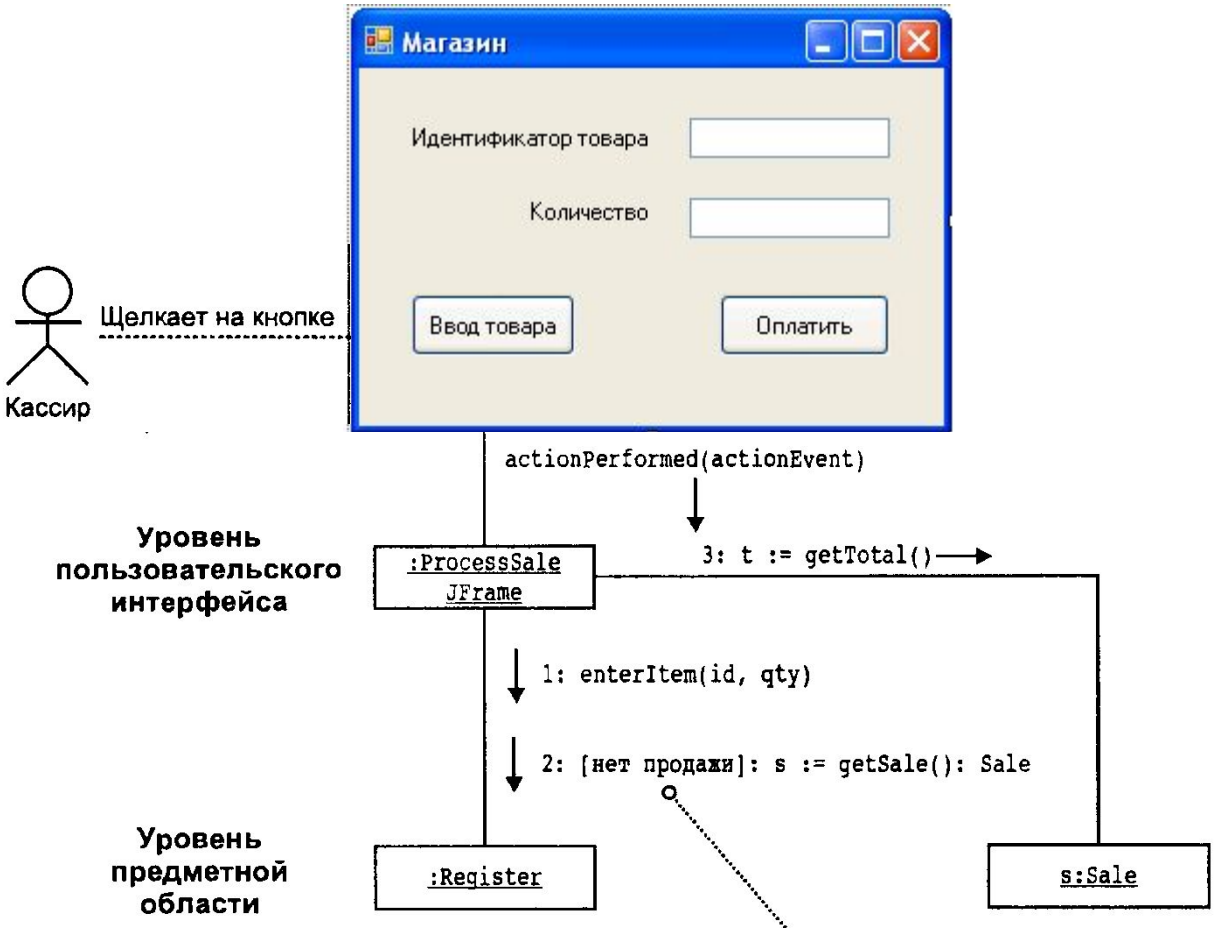
Диаграмма кооперации



Подключение уровня интерфейса

- До сих пор проектирование велось на уровне объектов предметной области
- Для подключения уровня пользовательского интерфейса потребуем, чтобы инициализирующая программа создавала и объекты уровня пользовательского интерфейса и исходный объект уровня предметной области с их связыванием

Связь уровней приложения



СОЗДАНИЕ ДИАГРАММЫ КЛАССОВ

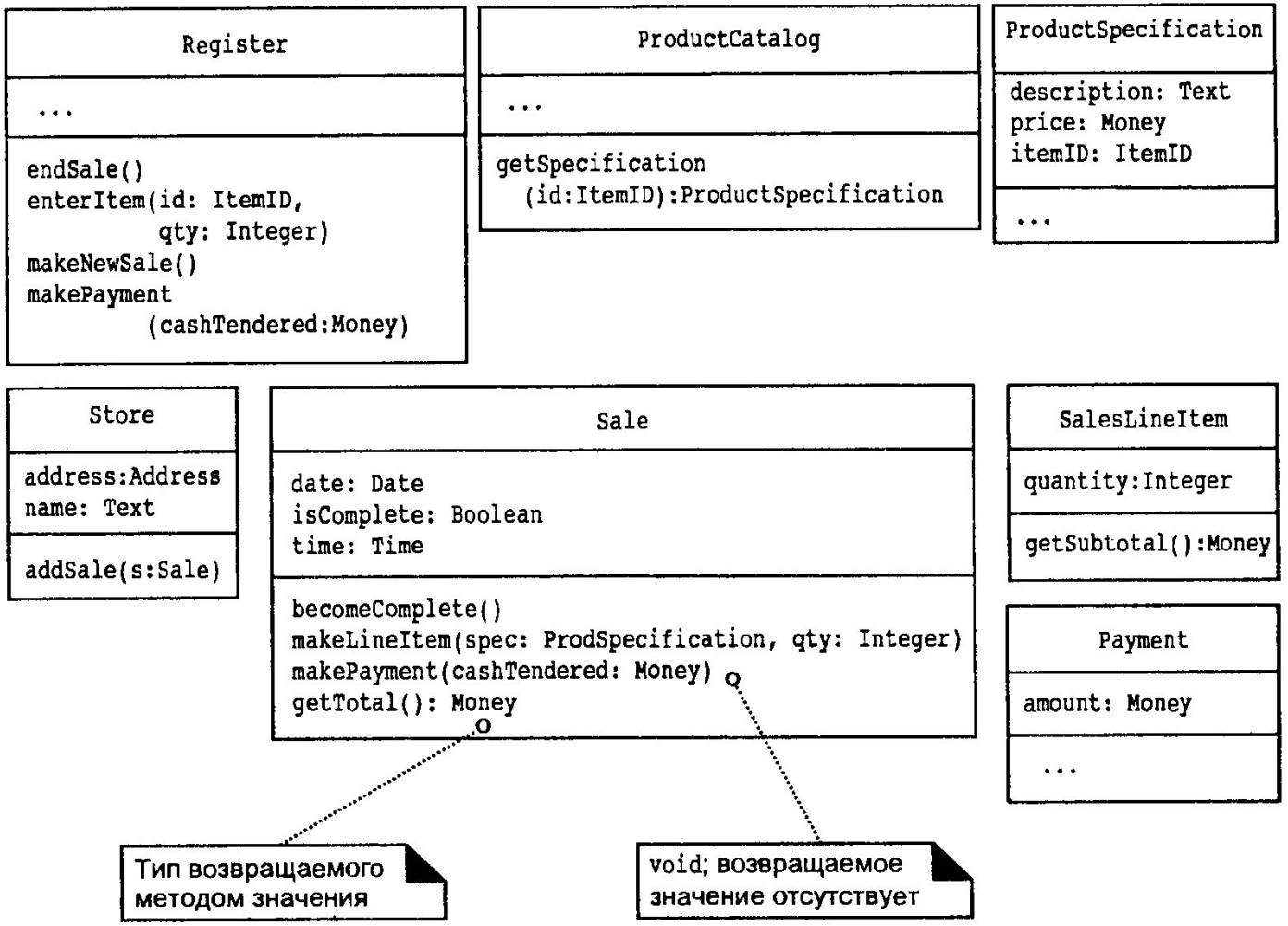
Идентификация классов

- На основе анализа диаграмм взаимодействия можно выделить следующие классы:
  - Register
  - Sale
  - ProductCatalog
  - ProductSpecification
  - SalesLineItem
  - Store
  - Payment

Определение методов классов

- Сообщения, передаваемые классу, определяют большую часть его методов
- Иногда на диаграмме классов можно размещать дополнительную информацию о типах передаваемых методами параметров и возвращаемых результатов

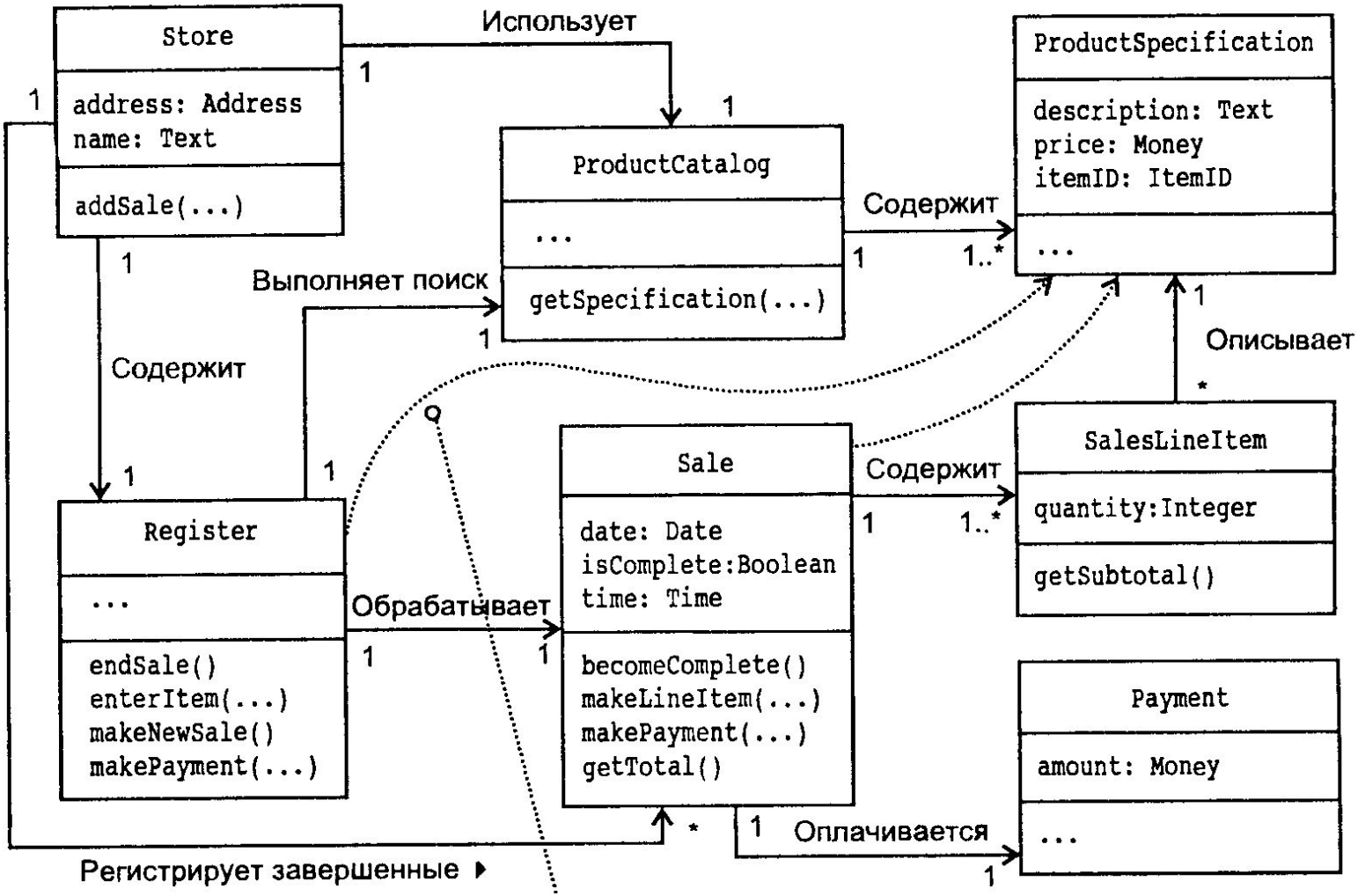
Методы классов



Ассоциации и навигация

- Линии связи и навигации устанавливаются на основе анализа диаграмм взаимодействия
- Такие ассоциации интерпретируются как видимость целевого класса для класса-источника, обеспечиваемая с помощью атрибутов (атрибут класса-источника является ссылкой на экземпляр целевого класса)

Ассоциации между классами



Отношения зависимости

- Означает наличие у одного из классов информации о другом классе
- Изображается пунктирной линией
- Объект класса Register получает информацию об объекте класса ProductSpecification в виде возвращаемого значения метода getSpecification, а объект класса Sale – через параметр метода makeLineItem

РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ

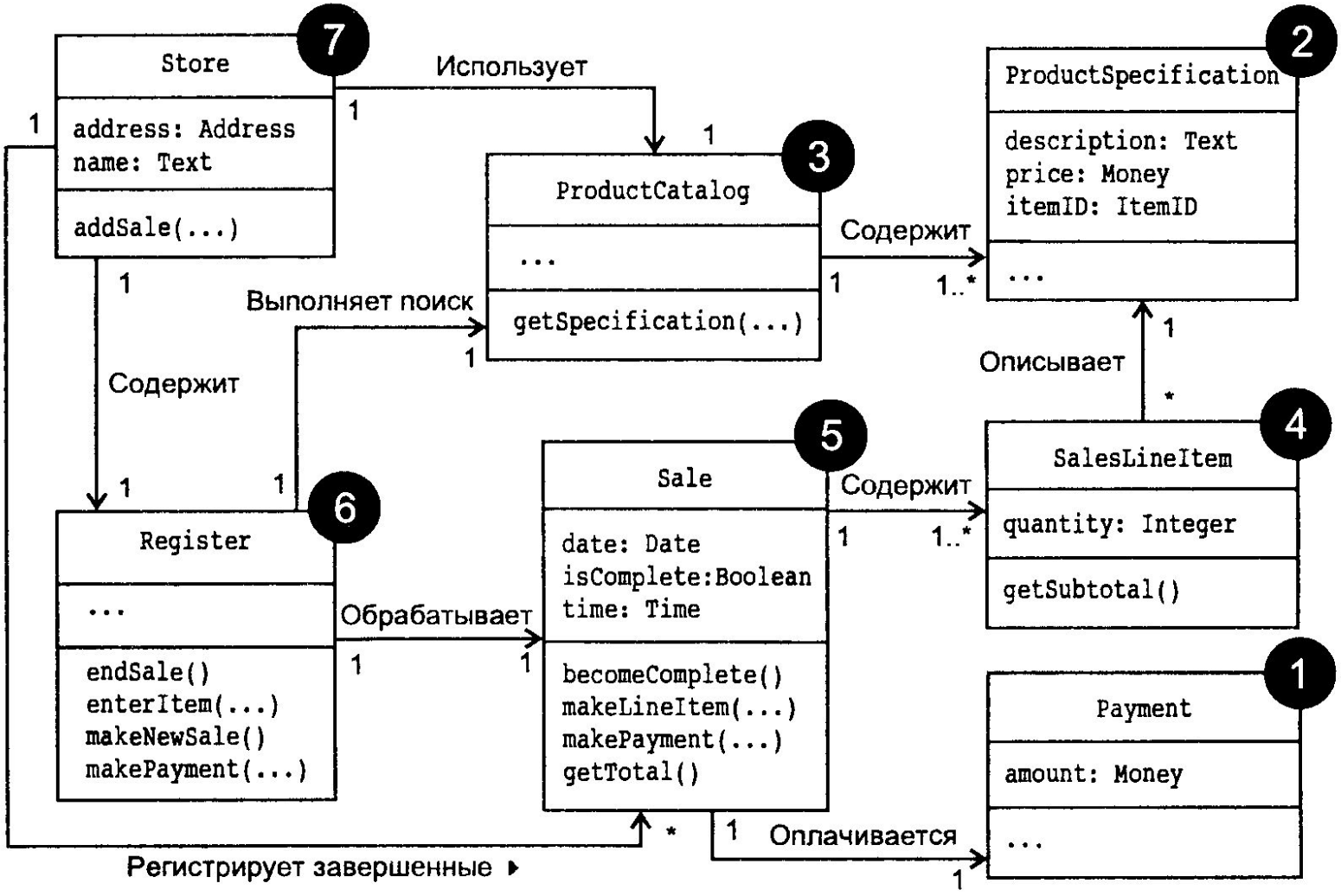
Программный код

- На заключительном этапе итерации проектное решение отображается в программный код
- При этом производится описание классов с добавлением атрибутов-ссылок, а также записывается реализация методов на основе соответствующих диаграмм взаимодействия

Порядок реализации классов

- Классы нужно реализовывать и тестировать в рамках модулей, начиная с минимально сцепленных
- В нашем проекте это классы Payment и ProductSpecification
- Затем реализуются классы со все большей степенью сцепления
- Примеры реализации

Вариант последовательности реализации



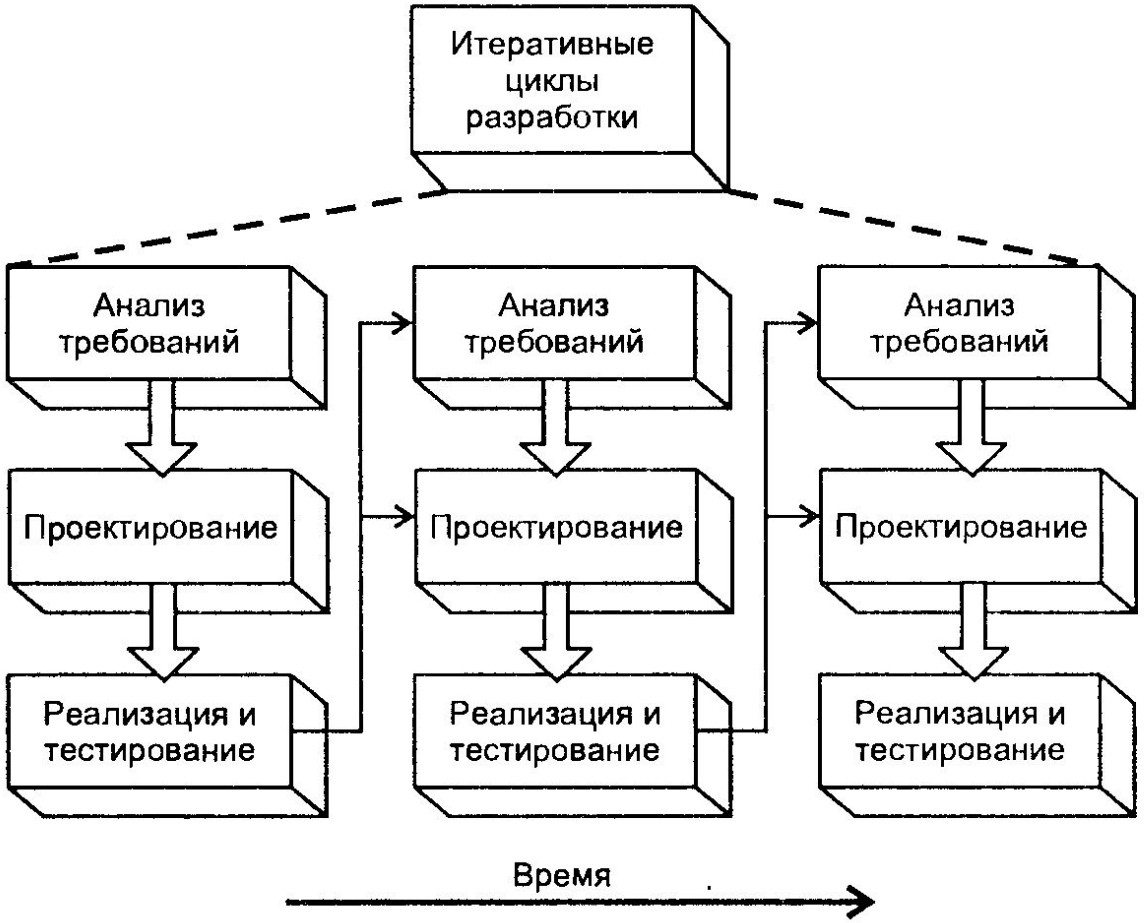
Тестирование модулей

- Осуществляется с использованием утилит модульного тестирования (NUnit, JUnit и т.д.)
- Широко используемой является практика *программирования на основе тестирования*, когда тесты пишутся *до* написания кода
- Тем самым обеспечивается тотальный и неформальный характер тестирования

Синхронизация артефактов

- На этапе реализации программного кода проясняются многие детали, не учтенные на стадиях анализа и проектирования
- Поэтому после завершения кодирования необходимо произвести синхронизацию основных артефактов – концептуальной модели, проектного решения и программного кода

Переход к новой итерации



ВТОРАЯ ИТЕРАЦИЯ

Задачи

- Реализация поддержки внешних служб (начисление налоговых платежей)
- Сложные правила вычисления стоимости
- Подключаемые бизнес-правила

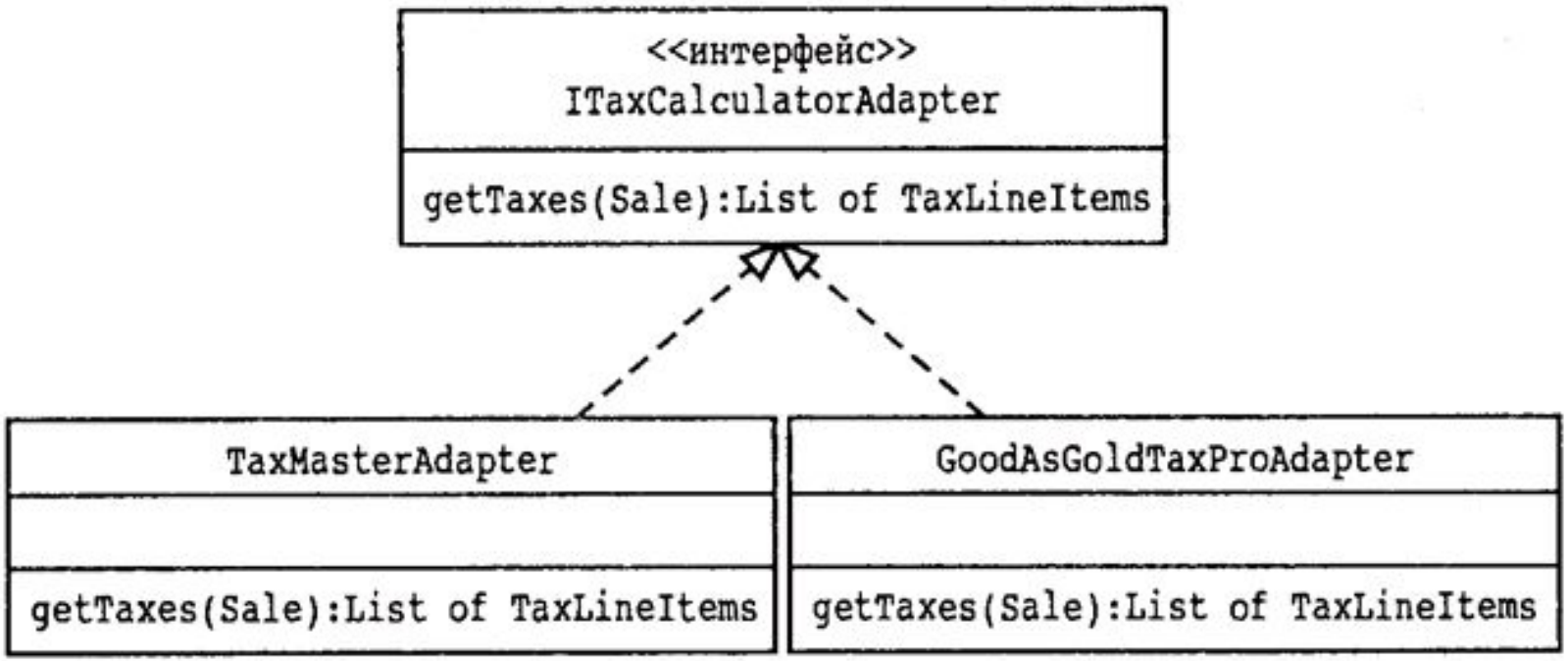
Задача1: Поддержка внешних служб

- Каждая из систем начисления налоговых платежей имеет собственный интерфейс и обладает собственным поведением
- Необходимо обеспечить возможность подключения разрабатываемой системы к любой из систем начисления налоговых платежей

Шаблон Adapter

- ***Проблема*** Как обеспечить взаимодействие с различными внешними системами?
- ***Решение*** Преобразовать интерфейс внешней системы к другому виду с помощью промежуточного объекта-адаптера

Пример


<<интерфейс>>
ITaxCalculatorAdapter
getTaxes(Sale):List of TaxLineItems
TaxMasterAdapter
getTaxes(Sale):List of TaxLineItems
GoodAsGoldTaxProAdapter
getTaxes(Sale):List of TaxLineItems

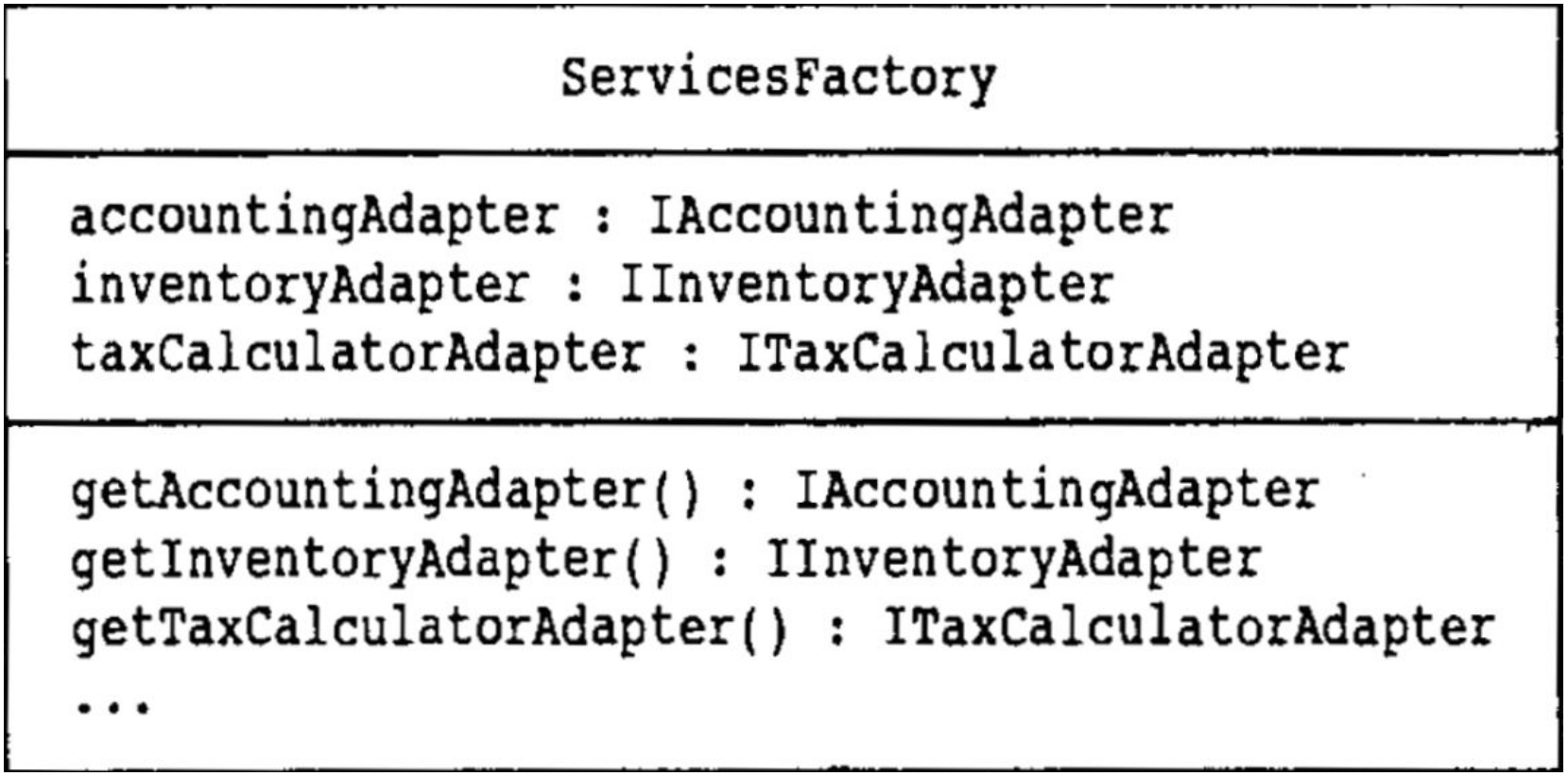
Шаблон Factory

- ***Проблема*** Какой класс должен отвечать за создание объектов-адаптеров? Как определять тип создаваемого адаптера?
- ***Решение*** Создать искусственный (не представляющий понятия предметной области) объект и назначить ему группу обязанностей по созданию других объектов

Обоснование решения

- Действительно, если бы обязанности по созданию новых объектов были поручены одному из объектов уровня предметной области (например, Register), то это нарушило бы принцип разделения обязанностей
- Кроме того, это уменьшило бы связность соответствующего объекта уровня предметной области

Объект-фабрика



Объект-фабрика

- Методы класса-фабрики возвращают результат с типом интерфейса, а не класса
- Это позволяет объекту-фабрике создавать любую реализацию интерфейса
- Информацию о типе создаваемого адаптера объект-фабрика должен получать из внешнего источника

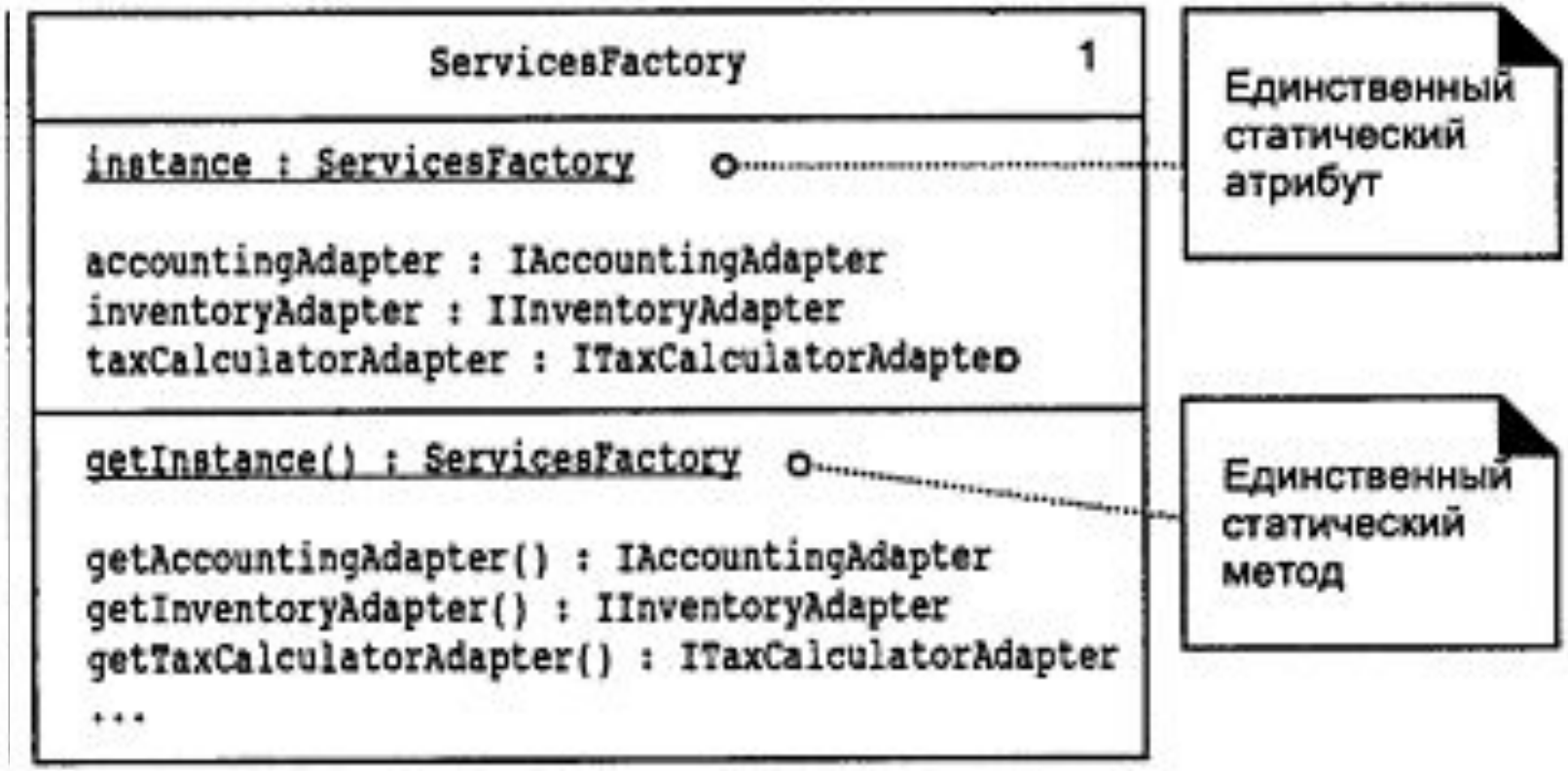
Шаблон Singleton

- Использование объекта-фабрики порождает новую проблему проектирования – кто должен создавать саму фабрику и как получить к ней доступ?
- При этом очевидно, что в рамках приложения нужен всего лишь один экземпляр фабрики, доступ к которому возможен для любых объектов

Шаблон Singleton

- ***Проблема*** Как обеспечить взаимодействие с объектом-фабрикой различных объектов системы, используя при этом единственную точку доступа?
- ***Решение*** Определить статический метод класса, возвращающий объект-фабрику

Шаблон Singleton



Реализация метода getInstance()

```
// статический метод
public static synchronized ServicesFactory getInstance( )
{
 if (instance == null)
        instance = new ServicesFactory( );
 return instance;
}
```

Пример обращения к объекту-фабрике

```
public class Register
{
    public void initialize ( )
    {…выполняем необходимые действия
     // обращаемся к объекту-фабрике через вызов
     // статического метода getInstance ( )
       ServicesFactory. getInstance ( ). getAccountAdapter();
     …выполняем необходимые действия
    }
     // другие методы
}
```

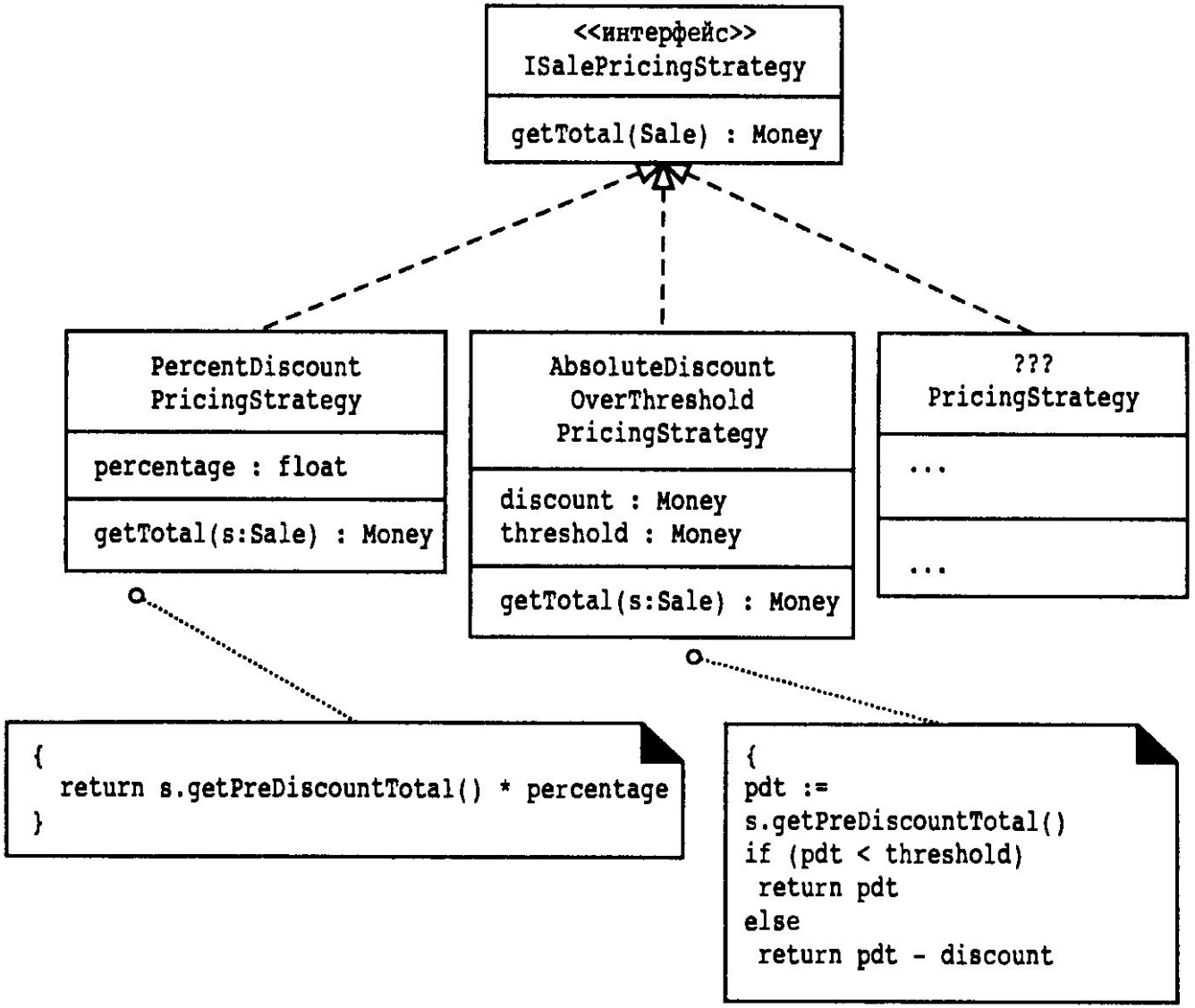
Задача2: Сложные правила вычисления стоимости

- Проблема заключается в обеспечении возможности вычисления стоимости покупки с учетом различного рода скидок (сезонных, постоянным клиентам и т.д.)
- Политика скидок может изменяться с течением времени, причем достаточно часто

Шаблон Strategy

- ***Проблема*** Как спроектировать надежные, но изменяемые алгоритмы (стратегии)? Каким способом вносить изменения?
- ***Решение*** Определить для каждого алгоритма отдельный класс со стандартным интерфейсом

Шаблон Strategy



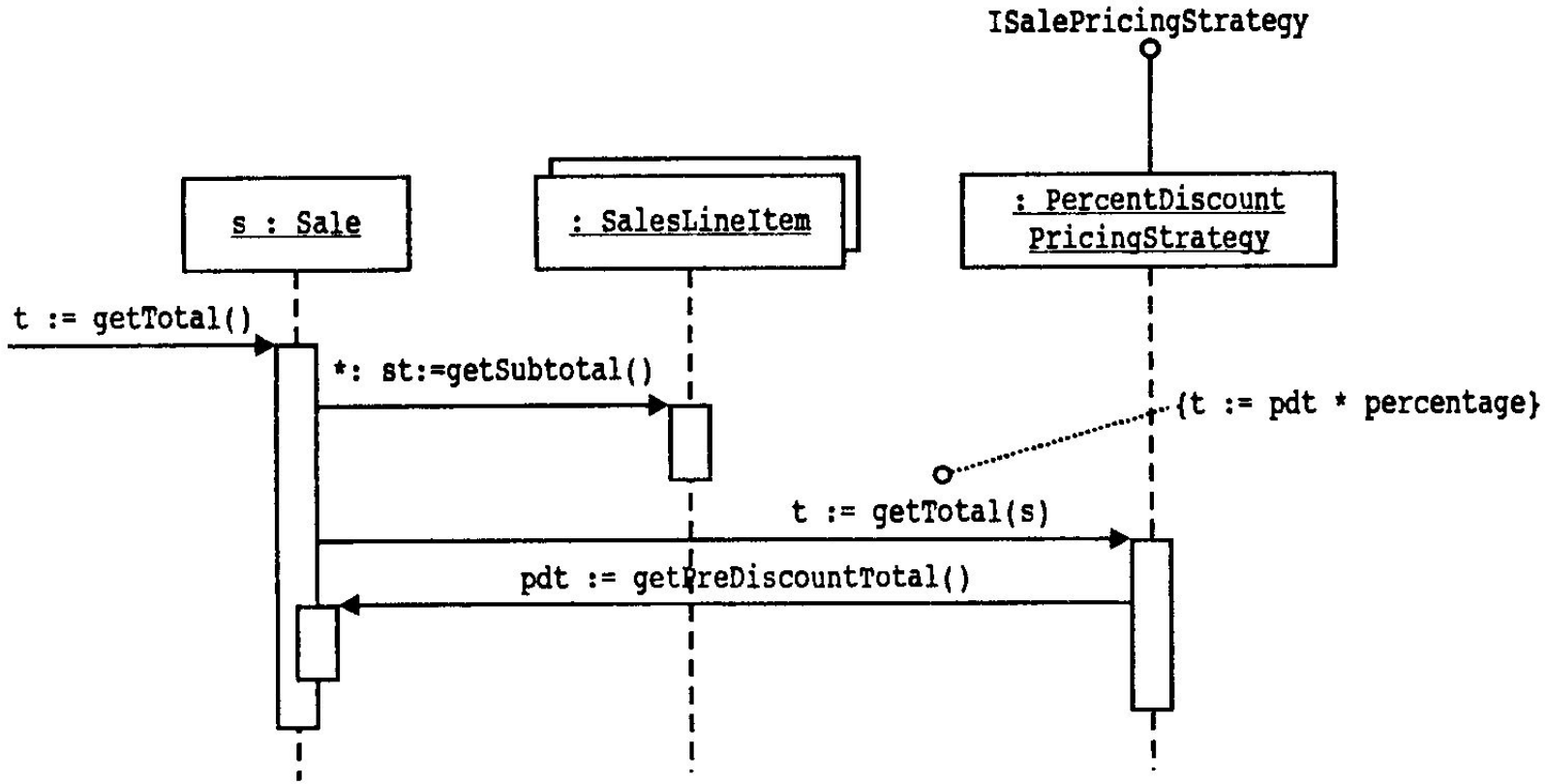
Терминология

- Объекты классов, реализующих различные стратегии называются *объектами стратегий*
- Объект, к которому применяется варьируемый алгоритм называется *контекстным объектом*
- В нашем примере контекстным объектом является объект класса Sale

Взаимодействие объекта-стратегии с объектом Sale

- При отправке объекту класса Sale сообщения getTotal он делегирует часть своих задач объекту стратегии
- При этом контекстный объект передает объекту стратегии ссылку на самого себя (this) для обеспечения параметрической видимости

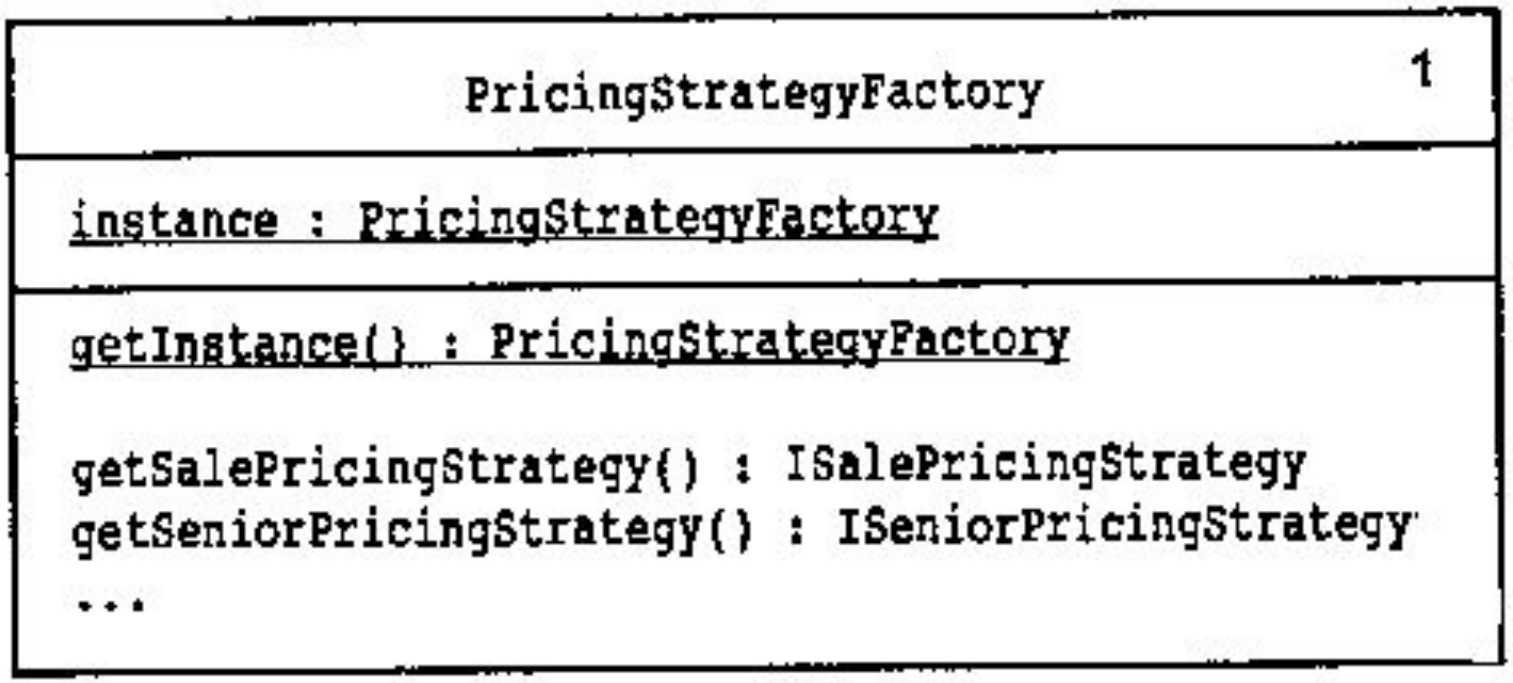
Взаимодействие объекта-стратегии с объектом Sale



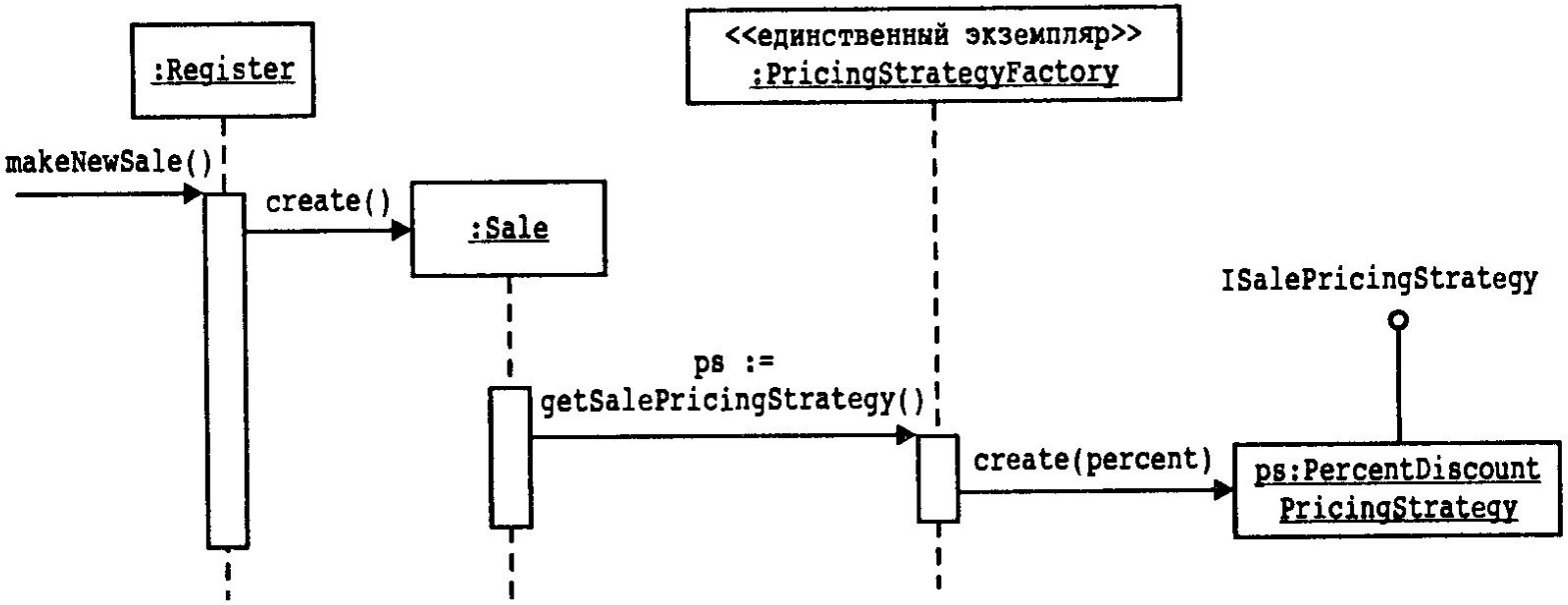
Создание объектов стратегий

- Так же, как и в случае адаптеров, создание объектов стратегий следует осуществлять на основе шаблона Factory
- Единственный объект-фабрика должен получать и выполнять запрос на создание объекта стратегии определенного типа

Фабрика стратегий



Создание объекта стратегии



Почему две фабрики

- «Каждому виду объектов своя фабрика» – такой подход позволяет понизить степень связанности в системе
- Объект-фабрика ServicesFactory имеет дело только с объектами-адаптерами, а объект-фабрика PricingStrategyFactory – с объектами-стратегиями

Задача 3:Подключение бизнес-правил

- Требуется подключить правила, отменяющие некоторые действия, например:
  - при наличии подарочного сертификата может быть приобретен только один товар – все операции **enterItem**, кроме первой должны быть отменены;
  - при проведении благотворительной акции могут приобретаться товары стоимостью не выше некоторой пороговой

Генератор правил

- Реализацию тех или иных бизнес-правил целесообразно выделить в особую подсистему – «генератор правил»
- Правила и их реализация подвержены частым изменениям, поэтому связывание с этой подсистемой должно быть минимальным

Шаблон Facade

- ***Проблема*** Как обеспечить унифицированный интерфейс с набором разрозненных интерфейсов и реализаций, подверженных частым изменениям
- ***Решение*** Определить одну точку взаимодействия с подсистемой – фасадный объект и возложить на него обязанность по взаимодействию с компонентами подсистемы

Шаблон Facade

- В данном случае можно определить подсистему «генератор правил», которую можно реализовать либо на основе шаблона Strategy, либо с помощью интерпретатора правил, считывающих и интерпретирующих набор правил if – then
- Фасадный объект для такой подсистемы можно назвать RuleEngineFacade

Пример обращения к объекту фасада

```
public class Sale
{   public void makeLineItem (  )
    {
     SalesLineItem sli = new SalesLineItem (spec, quant);
     // обращение к фасадному объекту
     if (RuleEngineFacade.getInstance( ).isInvalid(sli, this))
            return;
     lineItems.add(sli);
    }
    // другие методы
}
```

Конец лекции