Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Пермская государственная сельскохозяйственная академия
имени академика Д.Н. Прянишникова»

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

направление 230700 «Прикладная информатика»

ЛАБОРАТОРНОЕ ЗАНЯТИЕ № 8

Тема: **МОДЕЛЬ ПРОЕКТИРОВАНИЯ: ДИАГРАММЫ КЛАССОВ**

Учебные вопросы:

1. Модель проектирования: создание диаграммы классов.

Вопрос 1. Модель проектирования: создание диаграммы классов

Хотя в ходе лабораторных работ диаграммы классов создаются ***после*** диаграмм взаимодействия, на самом деле они зачастую разрабатываются параллельно. Имена многих классов, методов и типы отношений с помощью шаблонов распределения обязанностей можно определить уже на начальной стадии проектирования, до построения диаграмм взаимодействия. **Желательно выполнять проектирование следующим образом:** сначала стоит построить схематичные диаграммы взаимодействия, затем создать диаграммы классов, после чего детализировать диаграммы взаимодействия и т.д.

Диаграммы классов можно рассматривать как альтернативу **картам CRC**, обеспечивающую более удобное графическое представление информации при распределении обязанностей между классами.

Диаграмма классов, представленная на рис. 1.1, иллюстрирует фрагмент программного решения для классов **Register** и **Sale**.

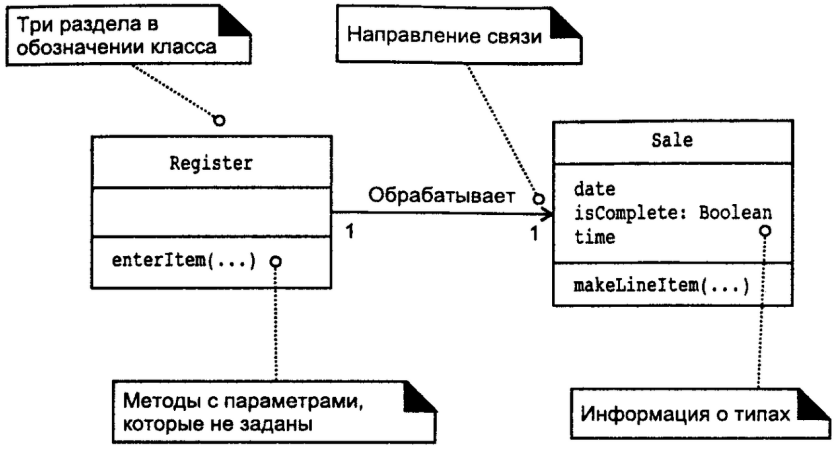

Рисунок 1.1 – Пример диаграммы программных классов

Диаграмма классов (design class diagram) иллюстрирует спецификации программных классов и интерфейсов (например, интерфейсов Java, C# и т.д.) в приложении. Обычно на такую диаграмму выносится следующая информация:

- Классы, ассоциации и атрибуты;
- Интерфейсы со своими операциями и константами;
- Методы;
- Информация о типах атрибутов;
- Способы навигации;
- Зависимости.

В отличие от диаграммы классов из модели предметной области, диаграммы классов проектирования отображают определения программных сущностей, а не понятия предметной области.

В RUP не определен отдельный артефакт под названием "диаграмма классов проектирования". Там определена лишь модель проектирования, которая может включать несколько типов диаграмм, в том числе диаграммы классов, взаимодействия и пакетов. Диаграммы классов из модели проектирования RUP иллюстрируют "классы проектирования" в терминах RUP. Поэтому зачастую диаграммы классов из модели проектирования называют просто диаграммами классов проектирования.

Классы из модели предметной области и модели проектирования

Напомним, что в модели предметной области объект Sale не представляет программную сущность, а определяет абстракцию понятия реального мира, состояние которой необходимо знать в системе. В отличие от модели предметной области, на диаграмме классов проектирования отображаются программные компоненты. Здесь элемент Sale представляет

собой программный класс (рис. 1.2).

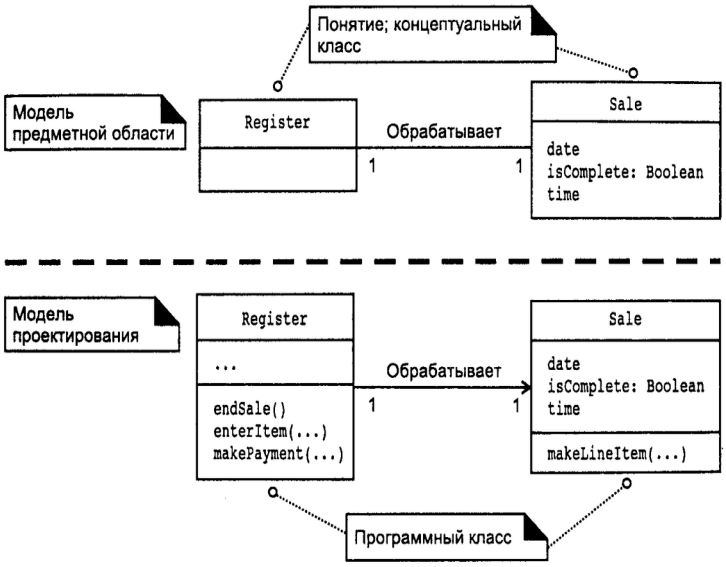

Рисунок 1.2 – Классы модели предметной области и модели проектирования

Создание диаграммы классов для POS-системы ТТ

Алгоритм построения диаграмм классов

1. Идентификация классов, которые должны участвовать в программном решении. Эту задачу можно решить, внимательно изучив все диаграммы взаимодействия и выбрав упомянутые на них классы.

Для POS-приложения к числу таких классов относятся следующие:

Register **Sale**

ProductCatalog **Product Specification**

Store **SalesLineItem**

Payment

2. Нанесение этих классов на диаграмму и добавление атрибутов, определенных с использованием модели предметной области (рис. 1.3).

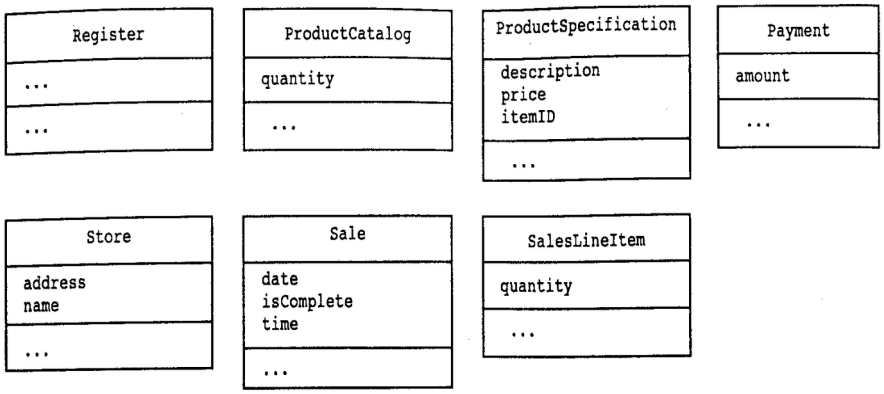

Рисунок 1.3 – Программные классы приложения

Заметим, что многие понятия модели предметной области, такие как **Cashier**, на диаграмме отсутствуют. Дело в том, что на данной итерации нет необходимости разрабатывать их программное представление. Однако на последующих итерациях, при появлении новых требований и разработке новых прецедентов, они, возможно, будут внесены в диаграмму. *Например*, при реализации требований к безопасности и процессу регистрации желательно ввести программный класс **Cashier**.

Добавление имен методов

Методы каждого класса можно определить путем анализа диаграмм взаимодействия. Например, если сообщение **makeLineItem** передается экземпляру класса Sale, то в классе Sale

должен быть определен метод **makeLineItem** (рис. 1.4).

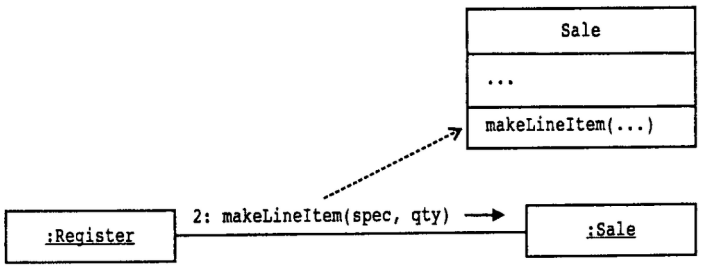

Рисунок 1.4 – Имена методов из диаграмм взаимодействия

Таким образом, имена всех сообщений, передаваемых классу **X**, отображенные на всех диаграммах взаимодействия, определяют большую часть методов этого класса.

Изучив все диаграммы взаимодействия для POS-приложения, получим имена методов, представленные на рис. 1.5.

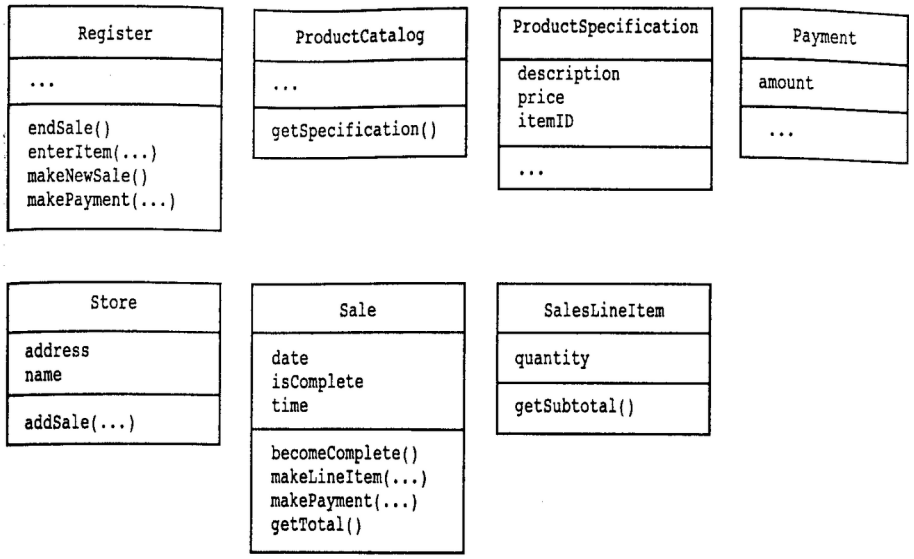

Рисунок 1.5 – Методы приложения

Выбор имен методов

При выборе имен методов необходимо руководствоваться следующими соображениями:

1. Интерпретировать сообщение create ( ).
2. Описывать методы доступа.
3. Интерпретировать сообщения сложным объектам.
4. Использовать синтаксис языка программирования.

Имена методов: *create*

Сообщение create на языке UML представляет собой абстрактную форму инициализации или инстанцирования. При переходе к реализации системы на объектно-ориентированном языке программирования процесс передачи сообщения необходимо выразить средствами языка программирования с использованием его идиом для инстанцирования и инициализации. В языке C++, Java или Smalltalk не существует реального метода create. Например, на языке C++ память выделяется с помощью оператора new, после которого следует имя конструктора.

Поскольку инициализация – это очень типичный вид деятельности, который по-разному выражается в различных языках программирования, методы создания объектов и конструкторы на диаграммах классов зачастую не отображаются.

Имена методов: методы доступа

Методы доступа (accessing methods) – это методы получения или установки значений атрибутов. В некоторых языках программирования для каждого атрибута принято создавать свои методы получения и установки значений, а сами атрибуты объявлять в закрытой области доступа (для обеспечения инкапсуляции). Эти методы обычно не отображаются на диаграмме классов, чтобы не загромождать ее лишней информацией, поскольку при наличии N атрибутов у класса появляется 2N стандартных метода. Например, метод getPrice (или price) класса Prod-

uctSpecification не показан на диаграмме классов, поскольку это простой метод доступа.

Имена методов: сложные объекты

Сообщения сложному объекту интерпретируются как сообщения контейнеру или объекту-коллекции. *Например*, следующее сообщение find сложному объекту можно интерпретировать как сообщение объекту-контейнеру, такому как Map в Java, map в C++ или Dictionary в Smalltalk (рис. 1.6).

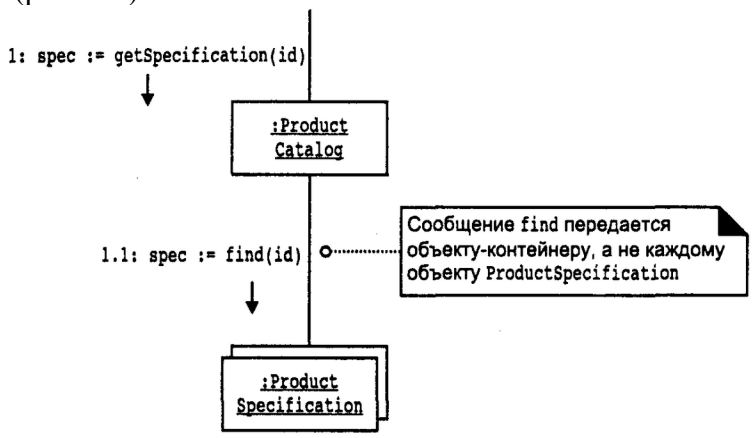

Рисунок 1.6 – Сообщение сложному объекту

Таким образом, метод find не является частью класса ProductSpecification, а относится к интерфейсу сложного объекта. Поэтому некорректно добавлять метод find в спецификацию класса ProductSpecification.

Классы или интерфейсы-контейнеры (такие как java.util.Map) – это элементы предопределенной библиотеки, которые не отображаются на диаграмме классов, поскольку они лишь загромождают диаграмму и не несут новой информации.

Имена методов: синтаксис с учетом языка

В некоторых языках, таких как Smalltalk, синтаксическая форма описания метода отличается от базового формата UML вида ***имяМетода (списокПараметров).*** На диаграммах классов рекомендуется использовать базовый формат языка UML, даже если планируется реализовывать систему на языке с другим синтаксисом. Переход к синтаксису конкретного языка целесообразно выполнять на этапе генерации кода, а не в процессе создания диаграмм классов. Однако UML допускает применение различных синтаксисов для спецификации методов.

Добавление дополнительной информации о типах

На диаграмме классов можно отображать информацию о типах атрибутов, параметрах методов и возвращаемых значениях. Принимая решение о том, следует ли отображать эту информацию, необходимо учитывать следующие соображения.

Диаграмма классов создается для ее пользователей:

- **Если генерация кода будет выполняться автоматически с использованием CASE-средств, то на диаграмме классов необходимо отобразить полную и исчерпывающую информацию.**
- **Если диаграмма классов создается для программистов, то избыточные детали могут загромождать диаграмму.**

Например, всегда ли необходимо отображать на диаграмме все параметры; информацию об их типах? Это зависит от того, насколько информация очевидна для пользователей диаграммы.

На диаграмме классов, представленной на рис. 1.7, показана дополнительная информация о типах параметров.

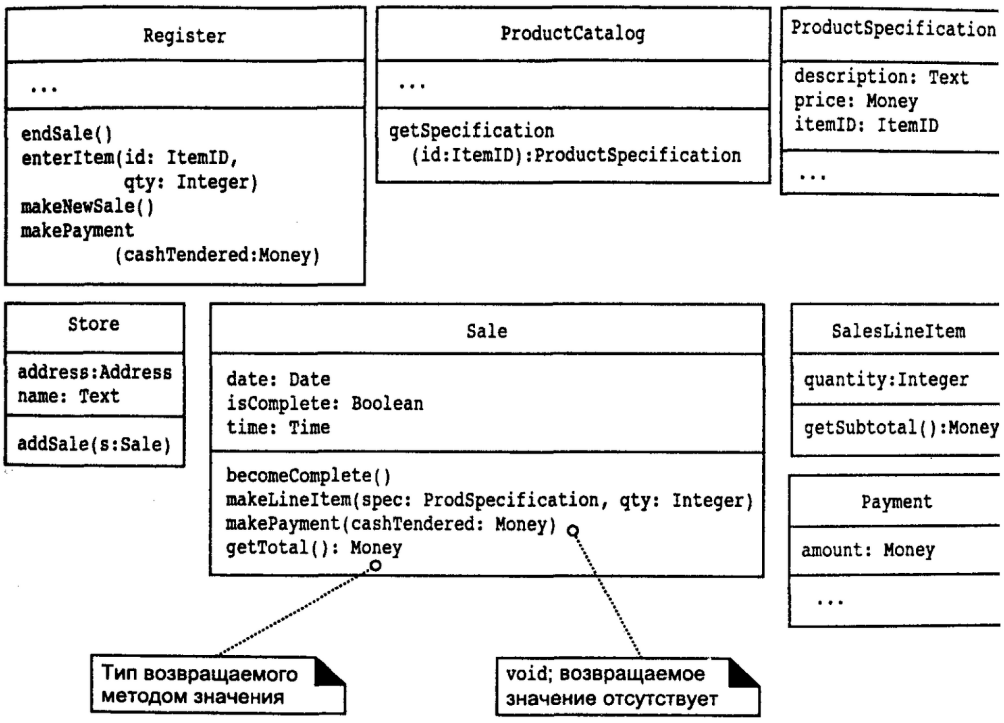

Рисунок 1.7 – Добавление информации о типах

Добавление ассоциаций и информации о навигации

Каждый конец линии ассоциации называется ***ролью*** (role). На диаграмме классов роль может отмечаться стрелкой (стрелкой навигации), указываю Щей направление связи. ***Информация о навигации*** (navigability) – это свойство роли, указывающее возможное направление передачи информации от объекта-источника к целевому классу. Информация о навигации связана с видимостью объектов, обычно – с видимостью, обеспечиваемой посредство» атрибутов (рис. 1.8).

Линия ассоциации со стрелкой навигации обычно интерпретируется как видимость целевого класса для класса-источника, обеспечиваемая с помощью атрибутов. В процессе реализации на объектно-ориентированном языке программирования она обычно выражается в том, что один из атрибутов класса-источника является ссылкой на экземпляр целевого класса. В частности, один из атрибутов класса Register должен являться ссылкой на экземпляр класса Sale.

Большинство (если не все) ассоциаций на диаграммах классов должно быть снабжено необходимыми стрелками навигации.

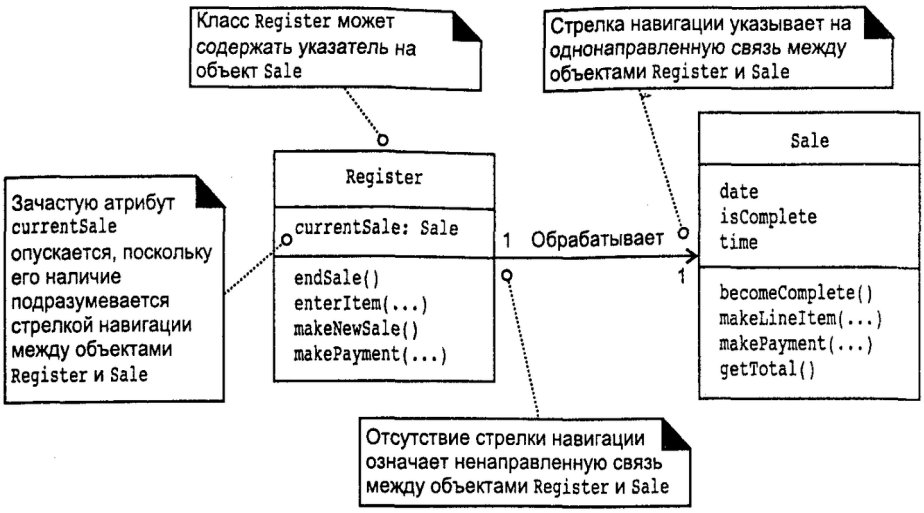

Рисунок 1.8 – Информация о навигации или видимости, обеспечиваемой посредством атрибутов

На диаграммах классов ассоциации выбираются по жесткому программно-

ориентированному критерию: отображаются те ассоциации, которые необходимы для удовлетворения требований видимости и памяти, вытекающих из диаграмм взаимодействий. Такой подход отличается от подхода к отображению ассоциаций в модели предметной области, где ассоциации отображают зависимости объектов предметной области. Здесь снова проявляются различия между моделями проектирования и предметной области: первая описывает программные компоненты, а вторая – результаты анализа предметной области.

Требуемые свойства видимости и ассоциации между классами определяются на основе диаграмм взаимодействия. Вот типичные ситуации, требующие определения ассоциаций с указанием направления связи от объекта A к объекту B:

- Объект A отправляет сообщение объекту B
- Объект A создает экземпляр объекта B
- Объект A должен поддерживать связь с объектом B

Например, из диаграммы взаимодействия, представленной на рис. 1.9, видно, что объект Store должен быть связан с создаваемыми им экземплярами объектов Register и ProductCatalog, причем эта связь должна быть направлена от объекта Store. Целесообразно также установить связь объекта ProductCatalog с создаваемой им коллекцией объектов ProductSpecification. На самом деле практически всегда направление связи соответствует направлению от объекта-создателя к создаваемым им объектам. На диаграмме классов такие связи представляются в виде ассоциаций.

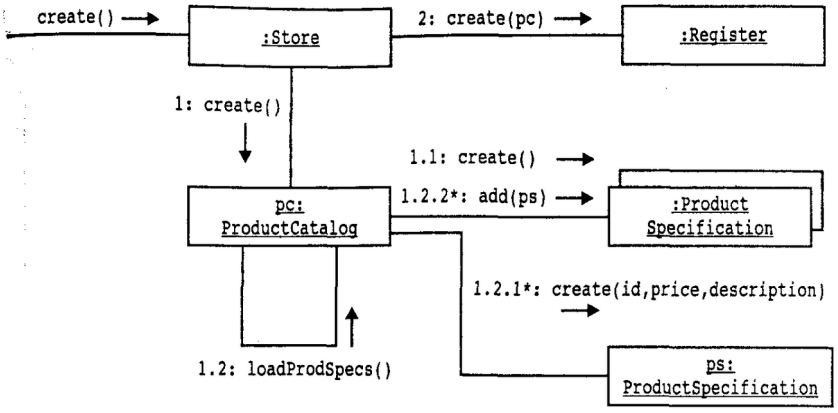

Рисунок 1.9 – Направление связи определяется из диаграмм взаимодействия

Пользуясь приведенными выше критериями для определения ассоциаций и направления связей на основе анализа всех диаграмм взаимодействия, для POS-приложения TT можно построить диаграмму классов, представленную на рис. 1.10. (Для большей ясности дополнительная информация о типах на этой диаграмме не отображается.)

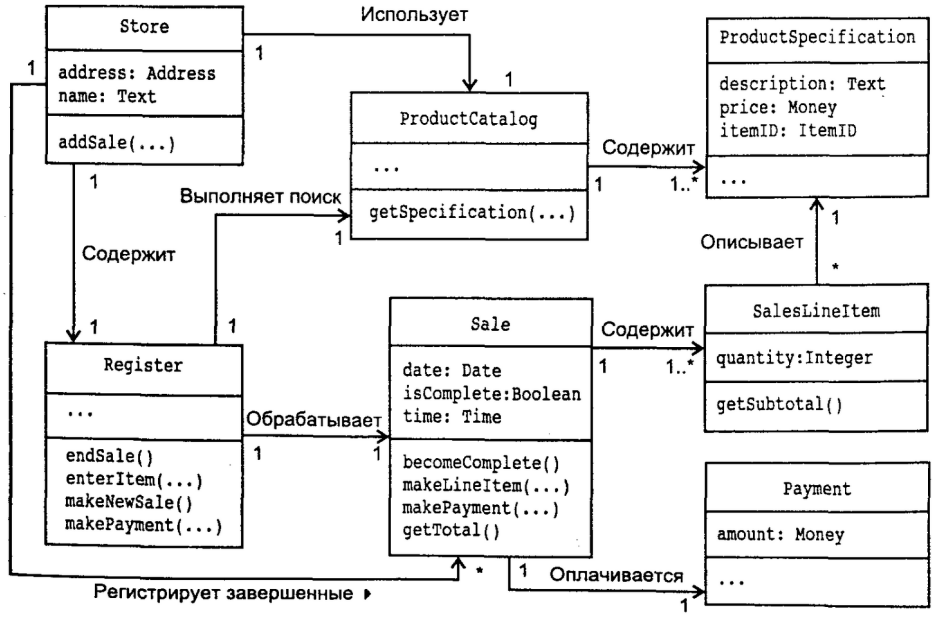

Рисунок 1.10 – Ассоциации с указанием направления связи

Заметим, что эти ассоциации не совпадают с набором ассоциаций, сгенерированным для диаграммы классов из модели предметной области. Например, в модели предметной области между классами Register и **ProductCatalog** отсутствует ассоциация «Находит в», поскольку на этапе создания модели предметной области эта взаимосвязь не казалась важной. Однако в процессе создания диаграмм взаимодействия было решено, что программный объект Register должен быть связан с программным объектом **ProductCatalog** с целью нахождения спецификации **ProductSpecification**.

Добавление зависимостей

В языке UML существует обозначение для *отношения зависимости* (dependency relationship), указывающего, что один элемент (любого типа, включая классы, прецеденты и т.д.) знает о другом элементе. Такое отношение отображается пунктирной линией со стрелкой.

На диаграмме классов отношение зависимости отображает видимость между классами, отличную от обеспечиваемой посредством атрибутов, т.е. глобальную, локальную видимость или видимость, обеспечиваемую с помощью параметров. Видимость, обеспечиваемая посредством атрибутов, отображается сплошной линией ассоциации со стрелкой, которая указывает направление связи. *Например*, программный объект **Register** получает возвращаемый объект типа **ProductSpecification** из сообщения, отправляемого им объекту **ProductCatalog**.

Таким образом, для объекта **Register** обеспечивается кратковременная локальная видимость объекта **ProductSpecification**. Объект **Sale** получает **ProductSpecification** в качестве параметра метода **makeLineItem** (видимость, обеспечиваемая посредством параметров).

Такие способы обеспечения видимости отображаются пунктирными линиями со стрелками, определяющими отношение зависимости (рис. 1.11). Линии зависимости не обязательно должны быть изогнутыми; просто так было удобно графически отобразить их в данном примере.

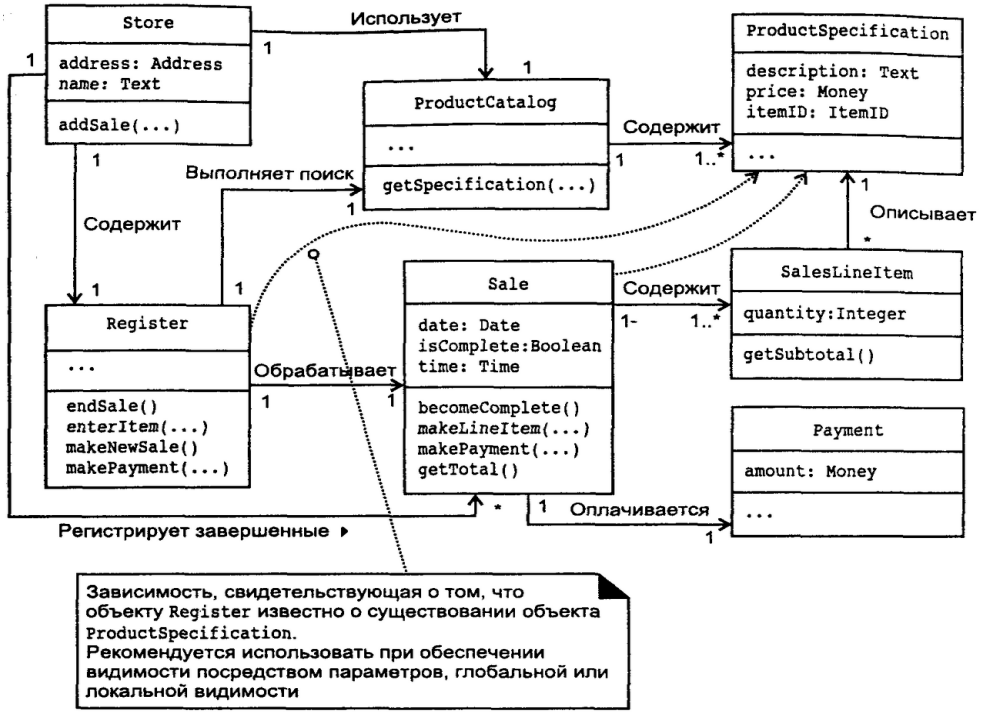

Рисунок 1.11 – Отношения зависимости, задающие видимость, отличную от видимости, обеспечиваемой посредством атрибутов

Задание на самостоятельную работу (для выбранной темы индивидуального проекта):

1. Построить диаграмму (программных) классов для модели проектирования (для основного успешного сценария прецедента).