

Дисциплина

СПЕЦИФИКАЦИЯ, ПРОЕКТИРОВАНИЕ И АРХИТЕКТУРА ПРОГРАММНЫХ СИСТЕМ

Тема 7

ДИАГРАММЫ UML

Преподаватель

к.т.н. Романенко Сергей Александрович

Диаграмма вариантов использования (use case diagram)

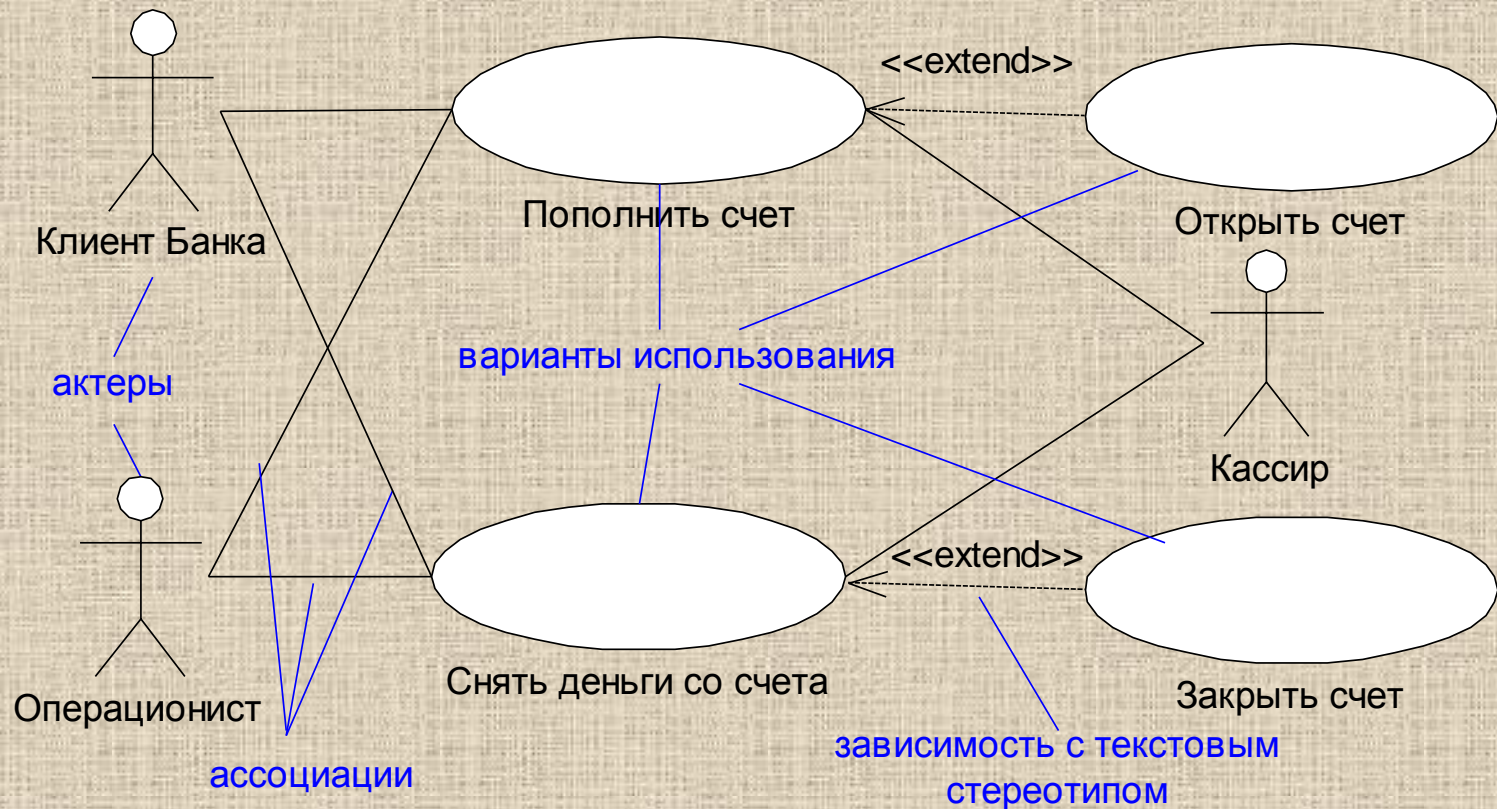
*Наиболее общее представление
функционального назначения системы*

Семантически вариант использования (use case) — это описание множества возможных последовательностей действий (событий), приводящих к значимому для действующего лица результату.

Каждая конкретная последовательность действий называется сценарием.

Диаграмма вариантов использования

Изображаются варианты использования проектируемой системы, заключенные в границу системы, и внешние актеры, а также определенные отношения между актерами и вариантами использования



Назначение диаграммы вариантов использования

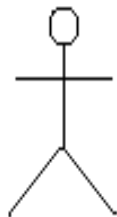
- ✓ Определить общие границы функциональности проектируемой системы в контексте моделируемой предметной области.
- ✓ Специфицировать требования к функциональному поведению проектируемой системы в форме вариантов использования.
- ✓ Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.
- ✓ Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями

Проектируемая система и ее окружение



Субъект (subject) – любой элемент модели, который обладает функциональным поведением

Основные обозначения на диаграмме вариантов использования



actor



use case



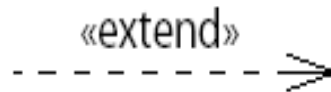
system boundary



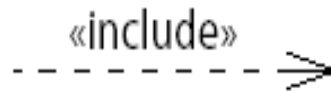
communication
association



generalization



extend



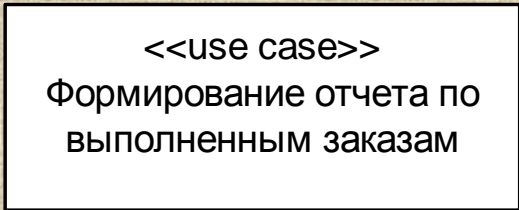
include

Вариант использования (use case)

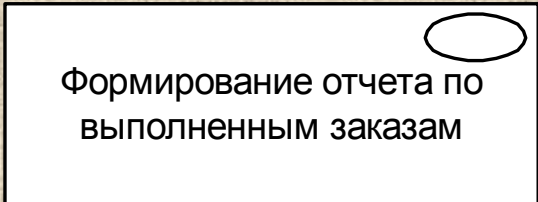
- Представляет собой общую спецификацию совокупности выполняемых системой действий с целью предоставления некоторого наблюдаемого результата, который имеет значение для одного или нескольких актеров
- Отвечает на вопрос «Что должна выполнять система?», не отвечая на вопрос «Как она должна выполнять это?»
- **Имя** – отглагольное существительное или глагол в неопределенной форме



Проверка состояния
текущего счета клиента



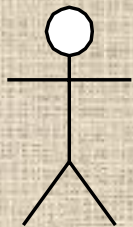
<<use case>>
Формирование отчета по
выполненным заказам



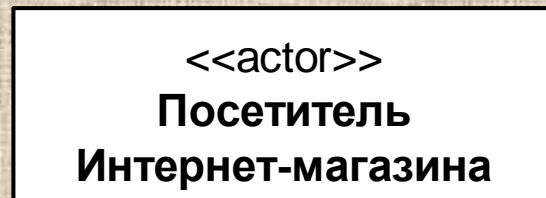
Формирование отчета по
выполненным заказам

Актер (actor)

- Любая внешняя по отношению к проектируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач
- *Примеры актеров:* кассир, клиент банка, банковский служащий, президент, продавец магазина, менеджер отдела продаж, пассажир авиарейса, водитель автомобиля, администратор гостиницы, сотовый телефон



Клиент банка

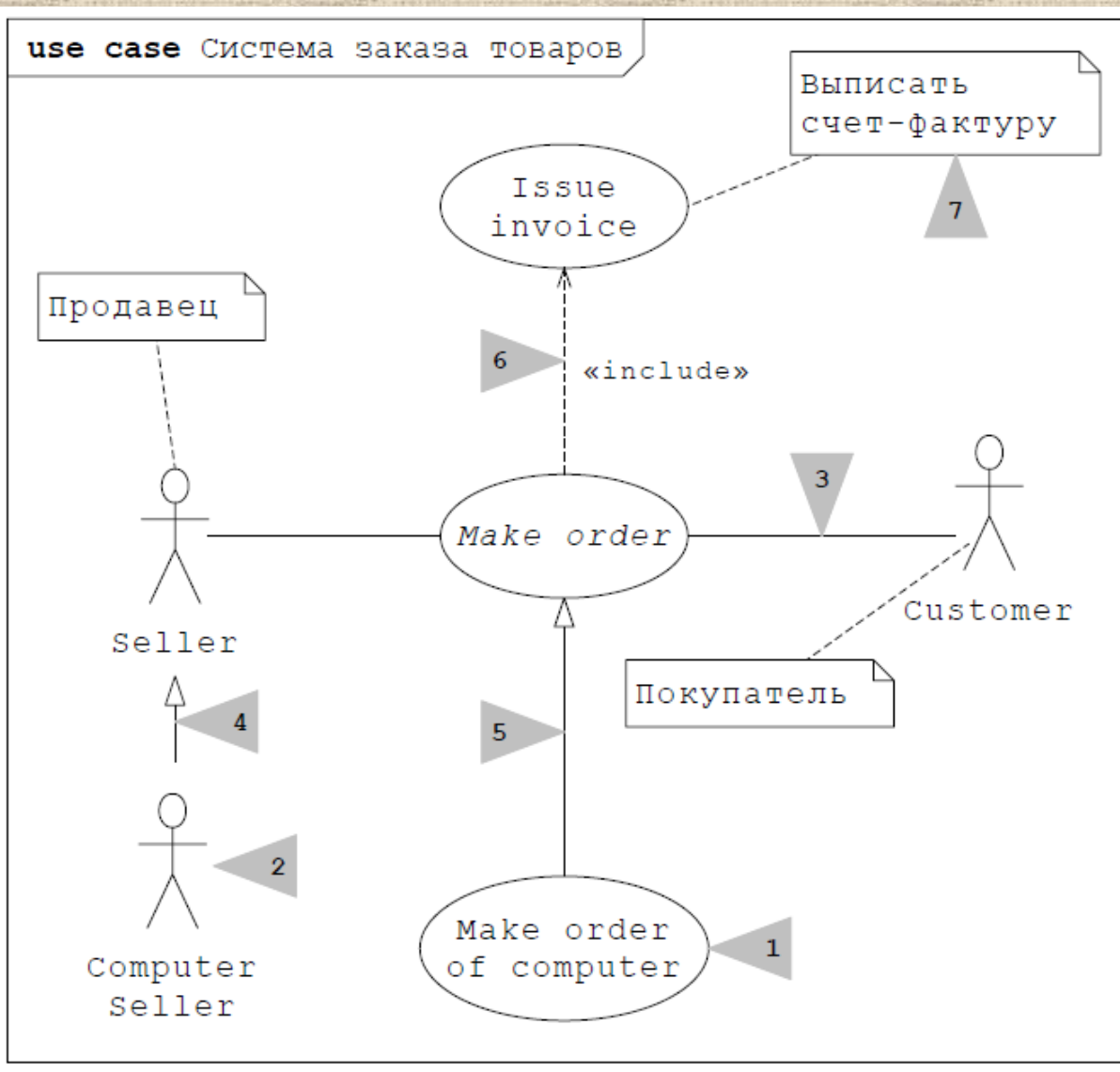


Удаленный
пользователь

Вопросы для идентификации актеров системы

- ✓ Какие организации или лица будут использовать систему
- ✓ Кто будет получать пользу от использования системы
- ✓ Кто будет использовать информацию от системы
- ✓ Будет ли использовать система внешние ресурсы
- ✓ Может ли один пользователь играть несколько ролей при взаимодействии с системой
- ✓ Могут ли различные пользователи играть одну роль при взаимодействии с системой
- ✓ Будет ли система взаимодействовать с законодательными, исполнительными, налоговыми или другими органами или автоматизированными системами

Отношения на диаграмме вариантов использования

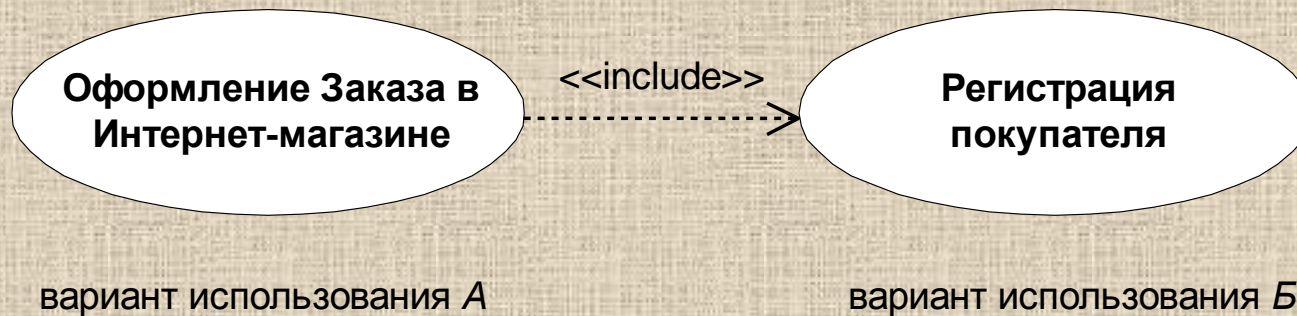


- ассоциация между действующим лицом и вариантом использования (3);
- обобщение между действующими лицами (4);
- обобщение между вариантами использования (5);
- зависимости между вариантами использования (6).

На диаграмме использования, как и на любой другой, могут присутствовать примечания (7).

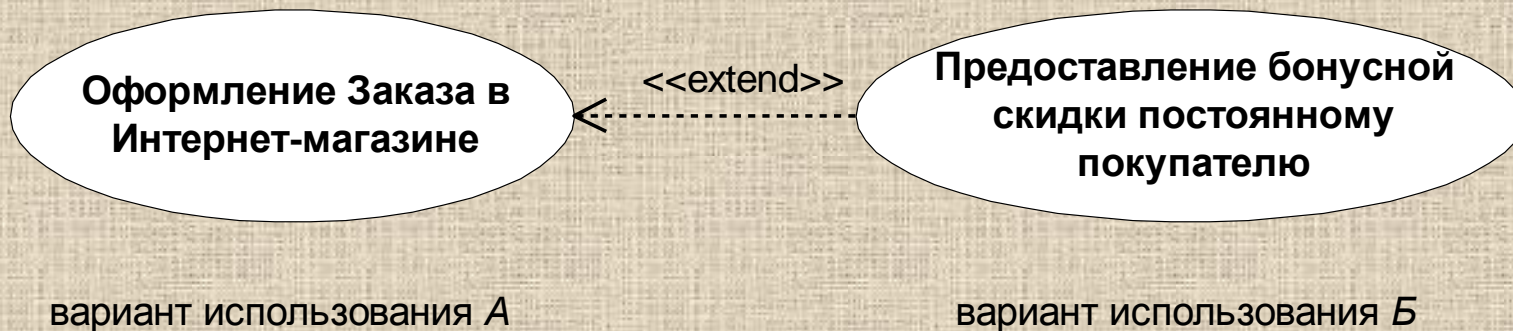
Отношение включения

- ✓ Отношение *зависимости* (*dependency*) определяется как форма взаимосвязи между двумя элементами модели, предназначенная для спецификации того обстоятельства, что изменение одного элемента модели приводит к изменению некоторого другого элемента
- ✓ Отношение *включения* (*include*) специфицирует тот факт, что некоторый вариант использования содержит поведение, определенное в другом варианте использования



Отношение расширения

- ✓ Отношение *расширения* (*extend*) определяет взаимосвязь одного варианта использования с некоторым другим вариантом использования, функциональность или поведение которого задействуется первым не всегда, а только при выполнении некоторых дополнительных условий.

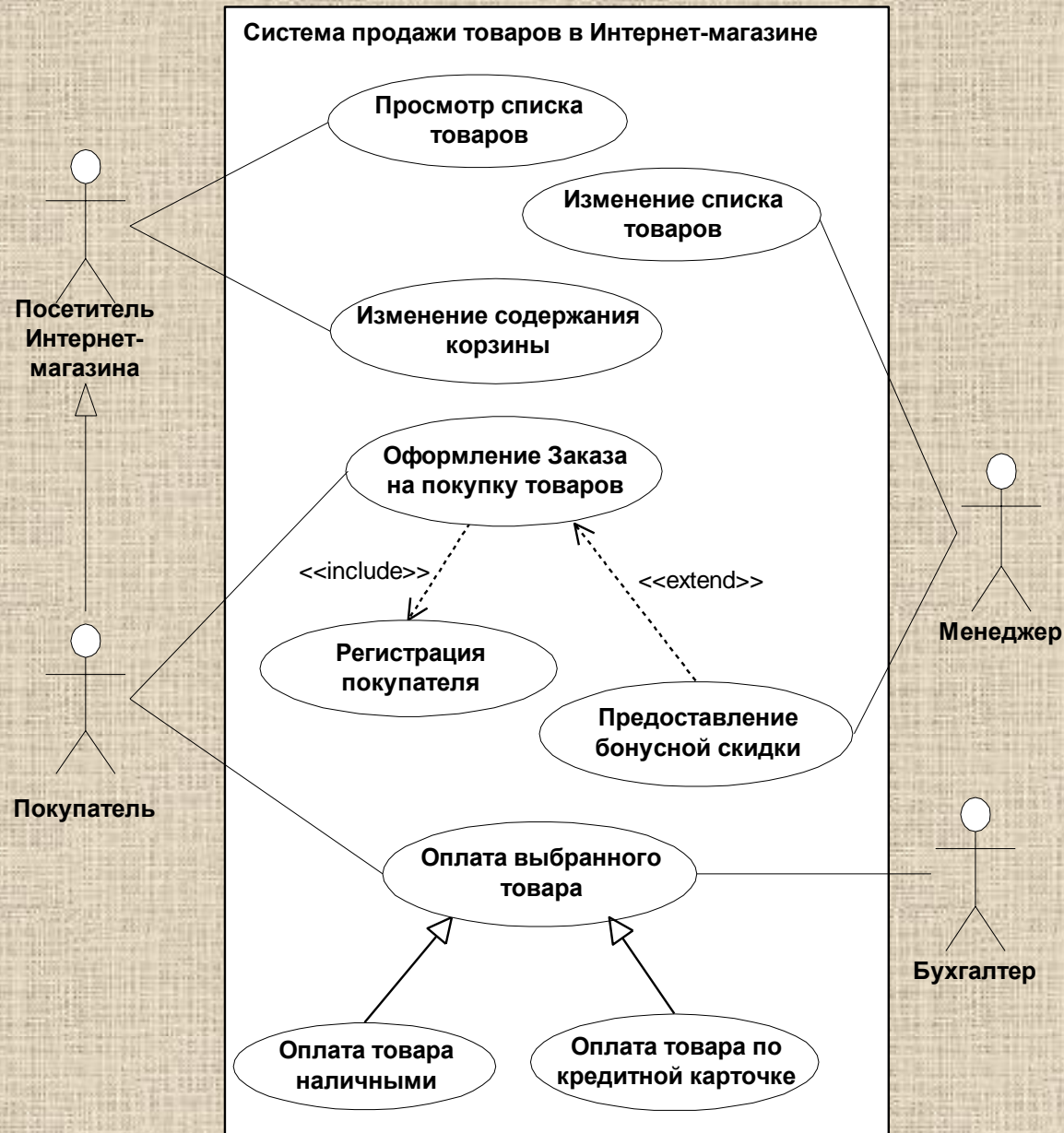


Отношение обобщения

- ✓ *Отношение обобщения (generalization relationship)* предназначено для спецификации того факта, что один элемент модели является специальным или частным случаем другого элемента модели



Пример диаграммы вариантов использования для системы продаж интернет-магазина



Формализация функциональных требований с применением диаграммы вариантов использования

- ✓ *Функциональные требования* определяют действия, которые должна быть способна выполнить система, без рассмотрения физических особенностей их реализации
- ✓ Тем самым функциональные требования определяют внешнее поведение системы
- ✓ Лучше всего они описываются в форме *модели вариантов использования*
- ✓ Каждому функциональному требованию в этом случае будет соответствовать отдельный *вариант использования*

Спецификация вариантов использования с помощью текстовых сценариев

Сценарий (scenario) – специально написанный текст, который описывает поведение моделируемой системы в форме последовательности выполняемых действий актеров и самой системы.

Актер	Цель № 1	Успех	Исключение № 1	Примечания
			Исключение № 2	
			Исключение № 3	
	Цель № 2	Успех	Исключение № 1	
			Исключение № 2	
			Исключение № 3	

Показатели качества для модели вариантов использования

- ✓ Все ли функциональные требования описываются вариантами использования?
- ✓ Не содержит ли модель вариантов использования ненужное поведение, которое отсутствует в требованиях?
- ✓ Действительно ли в модели необходимы все выявленные связи включения, расширения и обобщения?
- ✓ Правильно ли произведено деление модели на пакеты вариантов использования?
- ✓ Стала ли модель в результате деления на пакеты проще и удобнее для восприятия и сопровождения?
- ✓ Можно ли на основе модели вариантов использования составить четкое представление о функционировании системы в контексте ее пользователей?

Последовательность разработки вариантов использования

1. Определить главных (первичных) актеров и определить их цели по отношению к системе
2. Специфицировать все базовые (основные) варианты использования
3. Выделить цели базовых вариантов использования, интересы актеров в контексте этих вариантов использования, предусловия и постусловия вариантов использования
4. Написать успешный сценарий выполнения базовых вариантов использования
5. Определить исключения (неуспех) в сценариях вариантов использования и написать сценарии для всех исключений
6. Выделить варианты использования исключений и изобразить их со стереотипом «extend»
7. Выделить общие фрагменты функциональности вариантов использования и изобразить их отдельными ВИ со стереотипом «include»

Типичные ошибки при разработке диаграммы вариантов использования

- Превращение диаграммы вариантов использования в диаграмму деятельности за счет желания отразить все функциональные действия
- Инициатором действий является разрабатываемая система
- Спецификация атрибутов и операций классов до того, как сформулированы все варианты использования
- Задание слишком кратких имен вариантам использования
- Описание вариантов использования в терминологии, непонятной пользователям системы или заказчику
- Отсутствие описаний альтернативных последовательностей действий

Рекомендации по разработке диаграммы вариантов использования

- Рекомендуемое общее количество актеров в модели – не более 20, а вариантов использования – не более 50.
- Отдельный экземпляр варианта использования по своему содержанию является выполнением последовательности действий, которая инициализируется посредством передачи сообщения от экземпляра актера.
- Варианты использования могут быть специфицированы в виде текста, а в последующем – с помощью операций и методов вместе с атрибутами, в виде графа деятельности, посредством автомата или любого другого механизма описания поведения, включающего предусловия и постусловия.

Преимущества моделирования использования

- ✓ *Простые утверждения.*
- ✓ *Абстрагирование от реализации.*
- ✓ *Декларативное описание.*
- ✓ *Выявление границ.*

ДИАГРАММА КЛАССОВ

основная логическая модель

проектируемой системы

Описывает типы объектов системы и различного рода статические отношения, которые существуют между ними. На диаграммах классов отображаются также свойства классов, операции классов и ограничения, которые накладываются на связи между объектами

Моделирование структуры

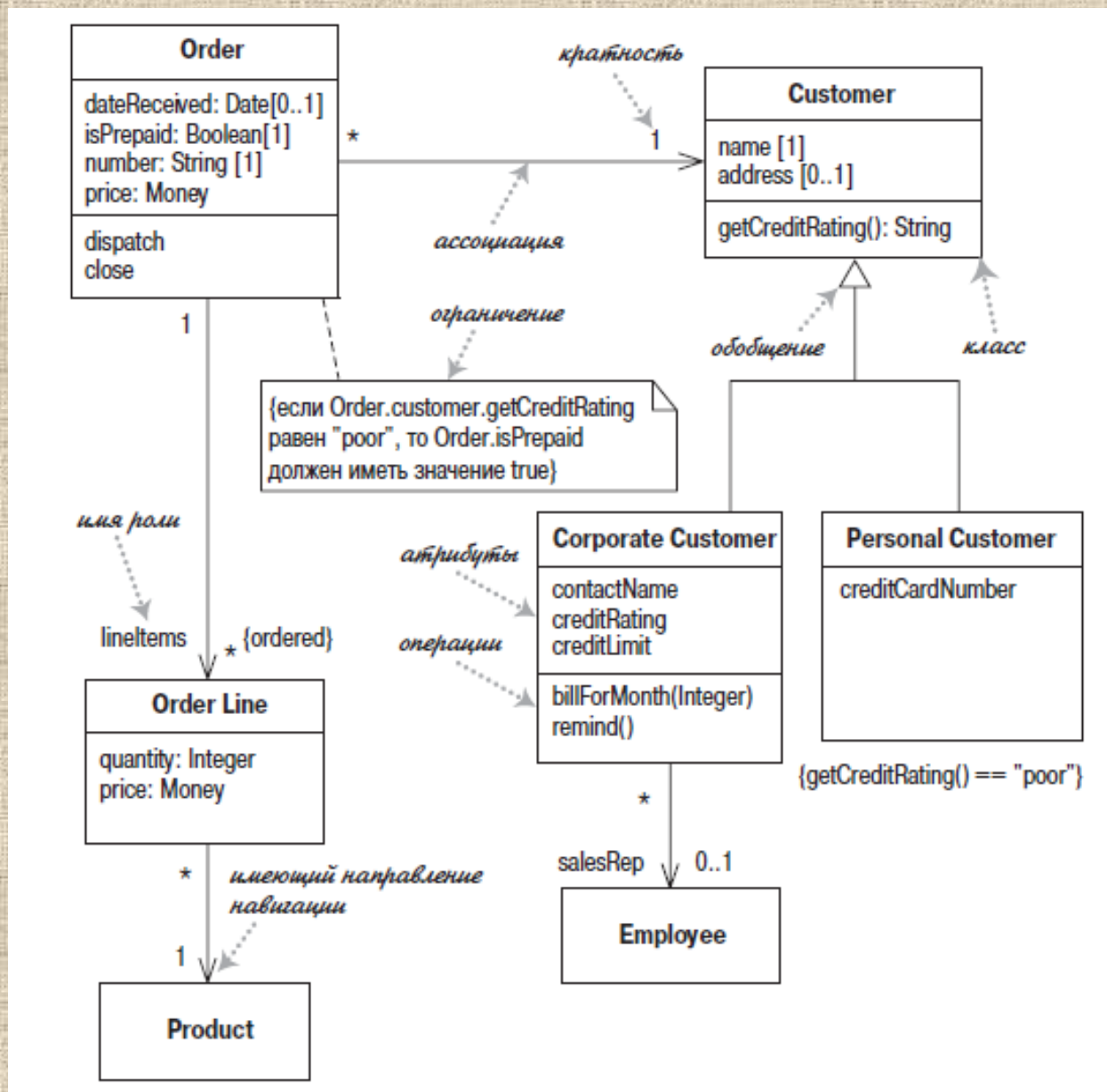
Структуру чего можно моделировать?

Важно! При моделировании структуры представляются такие аспекты архитектуры программы, которые не зависят от времени (являются статическими)

Виды структурного моделирования: пример классификации

1. Структура связей между объектами во время выполнения программы
2. Структура хранения данных
3. Структура программного кода
4. Структура компонентов в приложении
5. Структура сложных объектов, состоящих из взаимодействующих частей
6. Структура артефактов в проекте
7. Структура используемых вычислительных ресурсов

Пример простой диаграммы классов системы обработки заказов

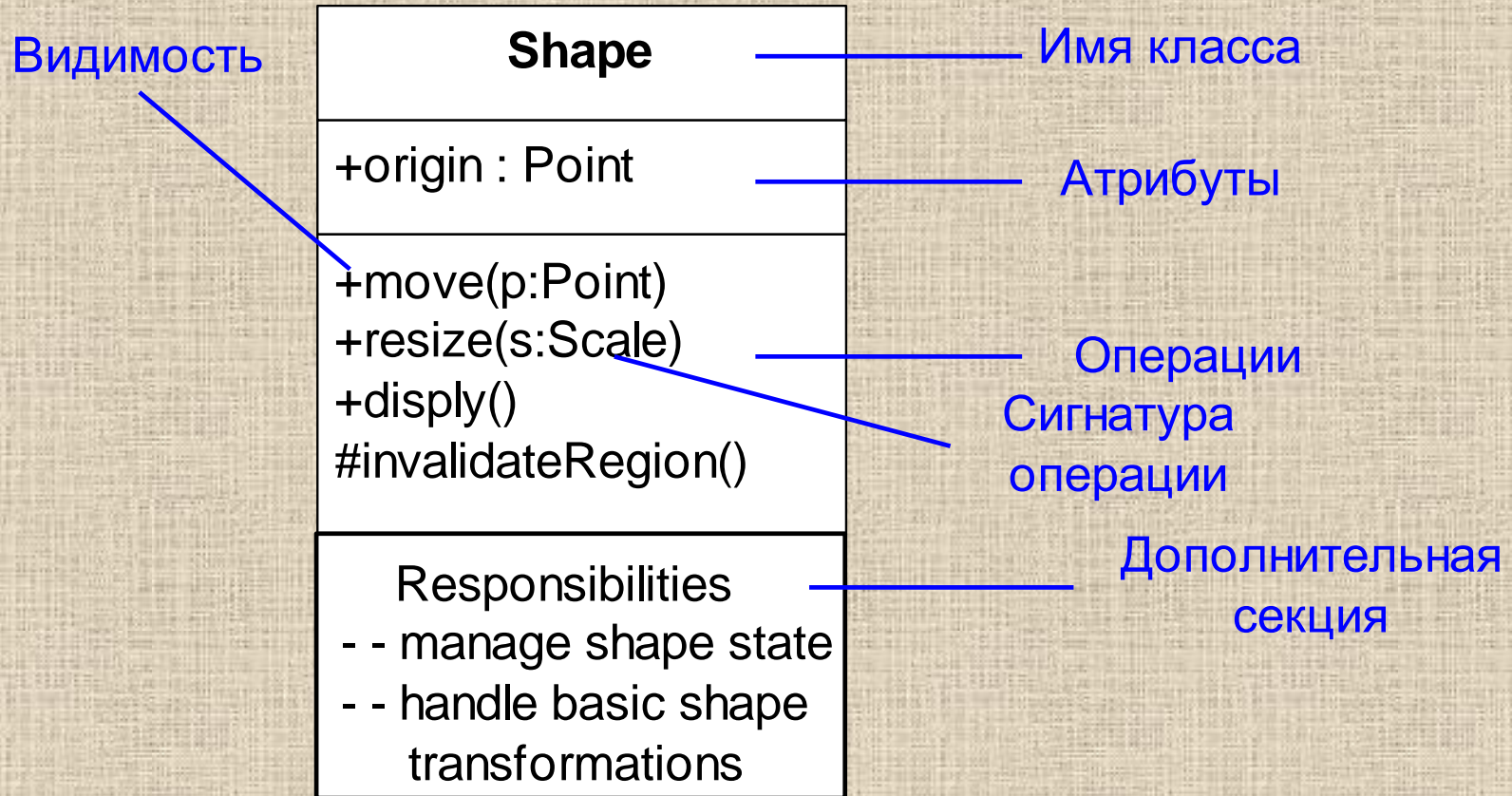


Примеры изображения классов



- ✓ **Класс (class)** – элемент модели, который описывает множество объектов, имеющих одинаковые спецификации характеристик, ограничений и семантики
- ✓ **Характеристика (feature)** – понятие, предназначенное для спецификации особенностей структуры и поведения экземпляров классификаторов
- ✓ **Структурная характеристика (свойство, атрибут)** является типизированной характеристикой классификатора, которая специфицирует структуру его экземпляров
- ✓ **Характеристика поведения (функции, методы)** является характеристикой классификатора, которая специфицирует некоторый аспект поведения его экземпляров

Элементы изображения класса



Правило записи имени класса

«стереотип» ИМЯ

- **Абстрактный (abstract)** - класс не имеет экземпляров или объектов, для обозначения его имени используется наклонный шрифт (*курсив*)
- **Конкретный класс (concrete)** – класс, на основе которого могут быть непосредственно созданы экземпляры или объекты
- **Квалифицированное имя (qualified name)** используется для того, чтобы явно указать, к какому пакету относится тот или иной класс. Для этого применяется специальный символ в качестве разделителя имени – двойное двоеточие “::”
- Имя класса без символа разделителя называется **простым именем** класса

Определенные в UML стереотипы классов

Стереотип

«actor»

«enumeration»

«exception»

«interface»

«signal»

«dataType»

«utility»

Описание

Действующее лицо

Перечислимый тип данных

Исключение

Все составляющие абстрактные

Экземплярами класса являются сигналы

Тип данных

Нет экземпляров, служба

Свойства (атрибуты) класса

Атрибут - это именованное место, в котором может храниться значение.

Атрибут служит для представления отдельной структурной характеристики или свойства, которое является общим для всех объектов данного класса.

видимость ИМЯ : тип = начальное_значение кратность {свойства}
--

Если имя атрибута подчеркнуто, то это означает, что областью действия данного атрибута является класс, а не экземпляр класса, как обычно.

Видимость атрибута класса

- + public (общедоступный). Общедоступный элемент является видимым всеми другими элементами приложения.
- private (закрытый). Закрытый элемент является видимым только внутри класса, который им владеет.
- # protected (защищенный). Защищенный элемент является видимым внутри класса и подклассам данного класса.
- ~ package (пакет). Элемент, помеченный как имеющий пакетную видимость, является видимым всеми элементами в ближайшем охватывающем пакете.

Важно! Если видимость не указана, то никакого значения видимости по умолчанию не подразумевается

видимость ИМЯ : тип = начальное_значение кратность {свойства}

Элементы записи атрибута

<имя> (name) представляет собой строку текста, которая используется в качестве идентификатора соответствующего атрибута и поэтому должна быть уникальной в пределах данного класса. Имя атрибута является единственным обязательным элементом в обозначении атрибута, должно начинаться со срочной (малой) буквы и, как правило, не должно содержать пробелов.

видимость **ИМЯ** : тип = начальное_значение кратность {свойства}

Элементы записи атрибута

<тип атрибута> (attribute type) есть имя классификатора (встроенного типа, класса, интерфейса и т.д. который является типом данного атрибута. Тип атрибута представляет собой имя некоторого типа данных, определенного в пакете. Типу атрибута должно предшествовать двоеточие.

<начальное_значение> (default) – некоторое выражение, которое служит для задания начального значения или значений данного атрибута в момент создания отдельного экземпляра соответствующего класса. Конкретное значение по умолчанию должно соответствовать типу данного атрибута. Если этот терм не указан, то значение атрибута на момент создания нового экземпляра класса не определено.

<кратность> (multiplicity) атрибута характеризует общее количество конкретных значений для атрибута, которые могут быть заданы для объектов данного класса.

видимость ИМЯ : тип = начальное_значение кратность {свойства}

Правила записи кратности

Кратность (multiplicity) множества — это множество чисел, которые задают все допустимые значения мощности для данного множества.

Нижняя граница .. ВЕРХНЯЯ ГРАНИЦА
--

Выражение кратности	Множество может иметь
0..*	Произвольное число элементов
*	Произвольное число элементов
1..*	Один или более элементов
0..1	Не более одного элемента
1..10	От одного до десяти элементов

Элементы записи атрибута

<свойства атрибута> (attribute modifier) представляет собой текстовое выражение, которое придает дополнительную семантику данному атрибуту. При этом набор возможных свойств атрибутов в языке UML 2.x фиксирован и может задаваться **ключевым словом** или **выражением-ограничением**.

Ключевые слова:

readOnly

union

subsets *имя_атрибута*

redefines *имя_атрибута*

ordered

unique

видимость ИМЯ : тип = начальное_значение кратность {свойства}

Примеры записи атрибутов

+ имяСотрудника : String {readOnly}

~ датаРождения : Data {readOnly}

/возрастСотрудника : Integer

+ номерТелефона : Integer [1..*] {unique}

– заработнаяПлата : Currency = 500.00

Операции класса

Операция (operation) класса служит для представления отдельной характеристики поведения, которая является общей для всех объектов данного класса.

Метод — это реализация операции класса

видимость ИМЯ (параметры) : тип {свойства}

Подчеркивание имени означает, что область действия операции — класс, а не объект

Список параметров операции класса

<список параметров> (parameter list) представляет собой перечень разделенных запятыми формальных параметров операции, каждый из которых имеет вид:

направление ПАРАМЕТР : тип = значение {свойства}
--

<направление> описывает семантическое назначение параметров, не конкретизируя конкретный механизм передачи

Ключевое слово Назначение параметра

in	Входной параметр — аргумент должен быть значением, которое используется в операции, но не изменяется
out	Выходной параметр — аргумент должен быть хранилищем, в которое операция помещает значение
inout	Входной и выходной параметр — аргумент должен быть хранилищем, содержащим значение. Операция использует переданное значение аргумента и помещает в хранилище результат
return	Значения этого параметра передаются в качестве возвращаемых значений вызывающему объекту после окончания выполнения операции

Свойства операций

redefines *<имя операции>* – данная операция переопределяет некоторую наследуемую операцию с именем *<имя операции>*

query – данная операция не изменяет состояния моделируемой системы и, соответственно, не имеет побочного эффекта

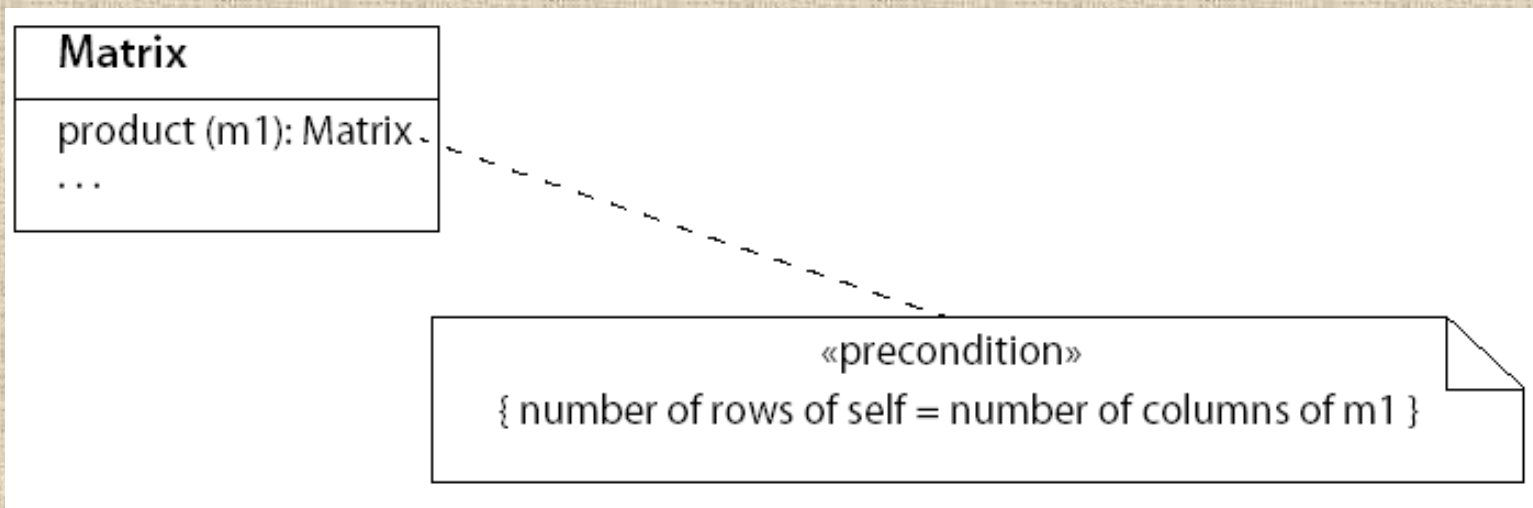
ordered – значения возвращаемого параметра являются упорядоченными
Предполагается, что кратность данного возвращаемого параметра должна быть больше 1

unique – значения возвращаемого параметра не могут повторяться.
Предполагается, что кратность данного возвращаемого параметра должна быть больше 1.

<ограничение> – выражение, которое специфицирует некоторое ограничение, применяемое к данной операции

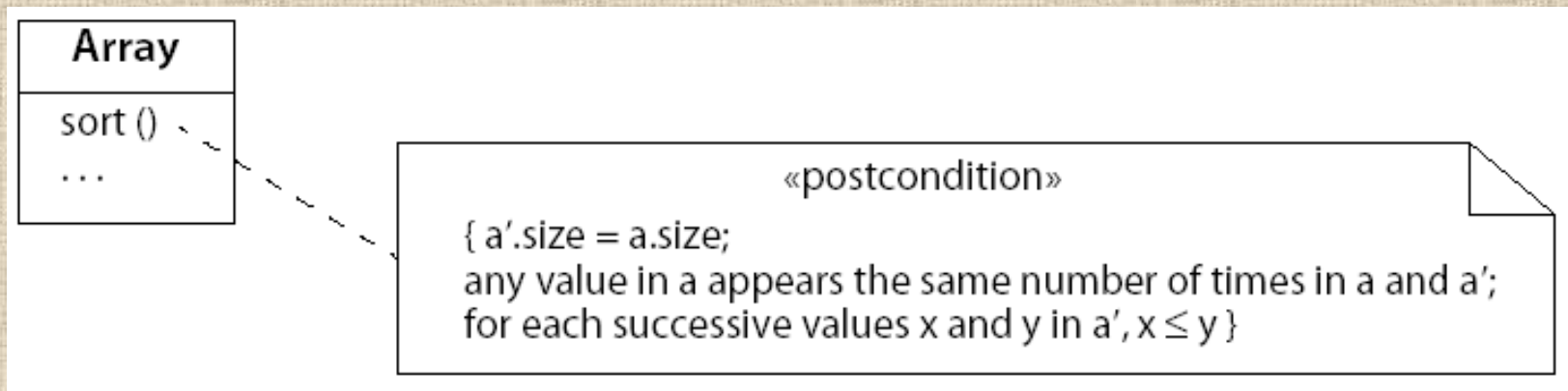
Предусловие операции

**определяет условие, которое должно быть истинным,
когда эта операция вызывается**



Постусловие операции

**определяет условие, которое должно быть истинным,
когда вызов операции успешно завершился, в
предположении, что все предусловия были
удовлетворены**



Примеры записи операций

+добавить(**in** номерТелефона : Integer [*] {unique})

–изменить(**in** заработнаяПлата : Currency)

+создать() : Boolean

toString(**return** : String)

toString() : String

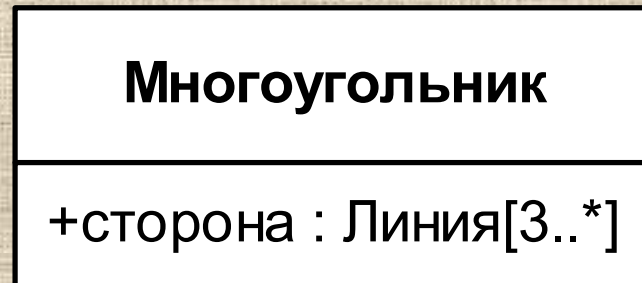
Отношения на диаграмме классов

<i>Relationship</i>	<i>Function</i>	<i>Notation</i>
association	Представление произвольного отношения между экземплярами классов	—
generalization	Отношение типа "Общее-Частное", обладающая свойством наследования свойств	—▷
aggregation	Отношение типа "Часть-Целое"	—◇
composition	Более сильная форма отношения типа "Часть-Целое"	—◆
realization	Отношение между спецификацией и ее выполнением	- - - ▷
dependency	Направленное отношение между двумя элементами модели с открытой семантикой	- - - →

Ассоциация

- ✓ **Ассоциация (association)** – произвольное отношение или взаимосвязь между классами
- ✓ **Имя конца ассоциации** специфицирует **роль (role)**, которую играет класс, расположенный на соответствующем конце рассматриваемой ассоциации
- ✓ **Видимость конца ассоциации** специфицирует возможность доступа к соответствующему концу ассоциации с других ее концов
- ✓ **Кратность конца ассоциации** специфицирует возможное количество экземпляров соответствующего класса, которое может соотноситься с одним экземпляром класса на другом конце этой ассоциации
- ✓ **Символ наличия навигации (navigable)** изображается с помощью простой стрелки в форме буквы «V» на конце ассоциации
- ✓ **Символ отсутствия навигации (non navigable)** изображается с помощью буквы «X» на линии у конца ассоциации

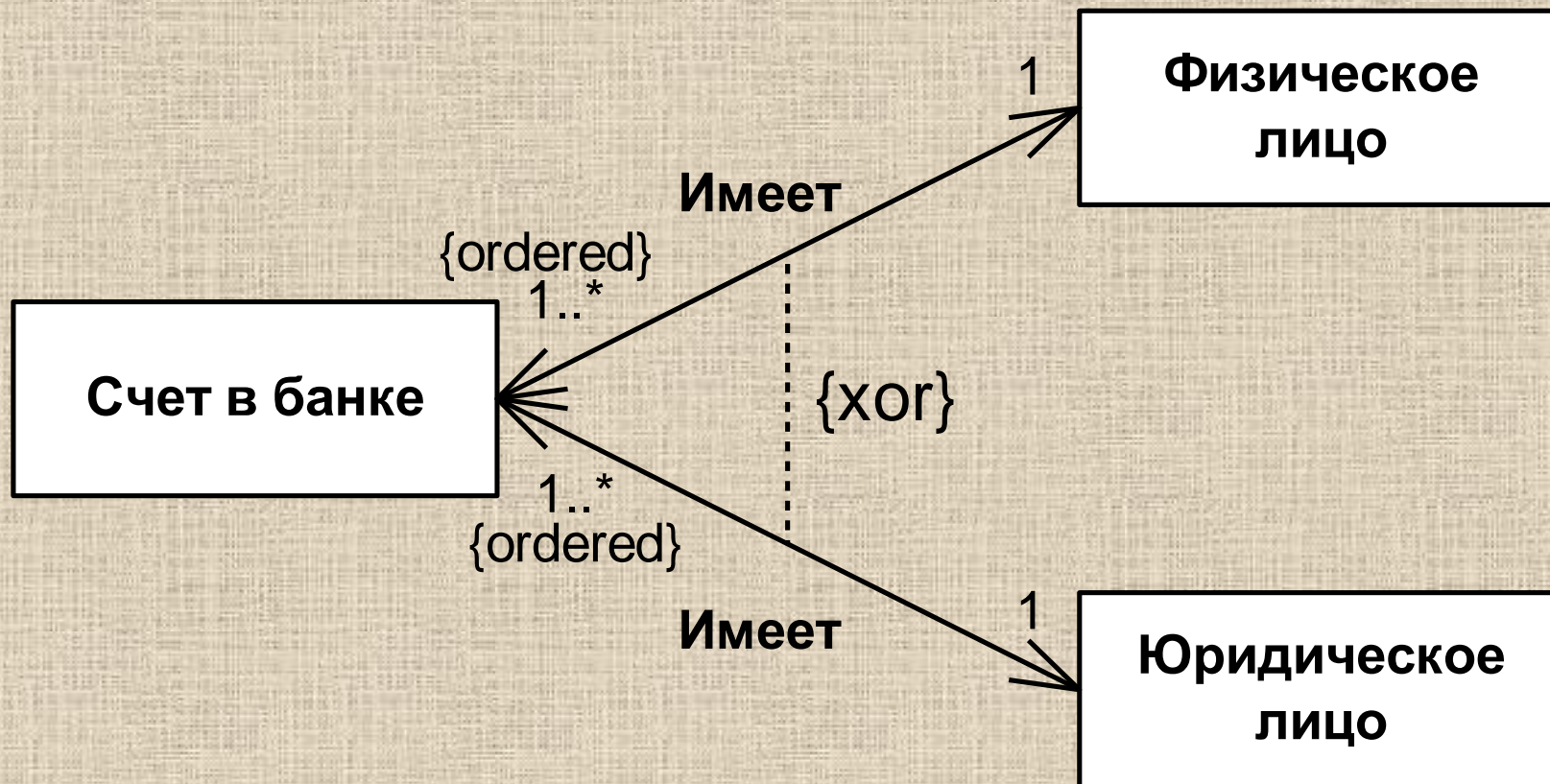
Ассоциация с навигацией и эквивалент класс с атрибутом



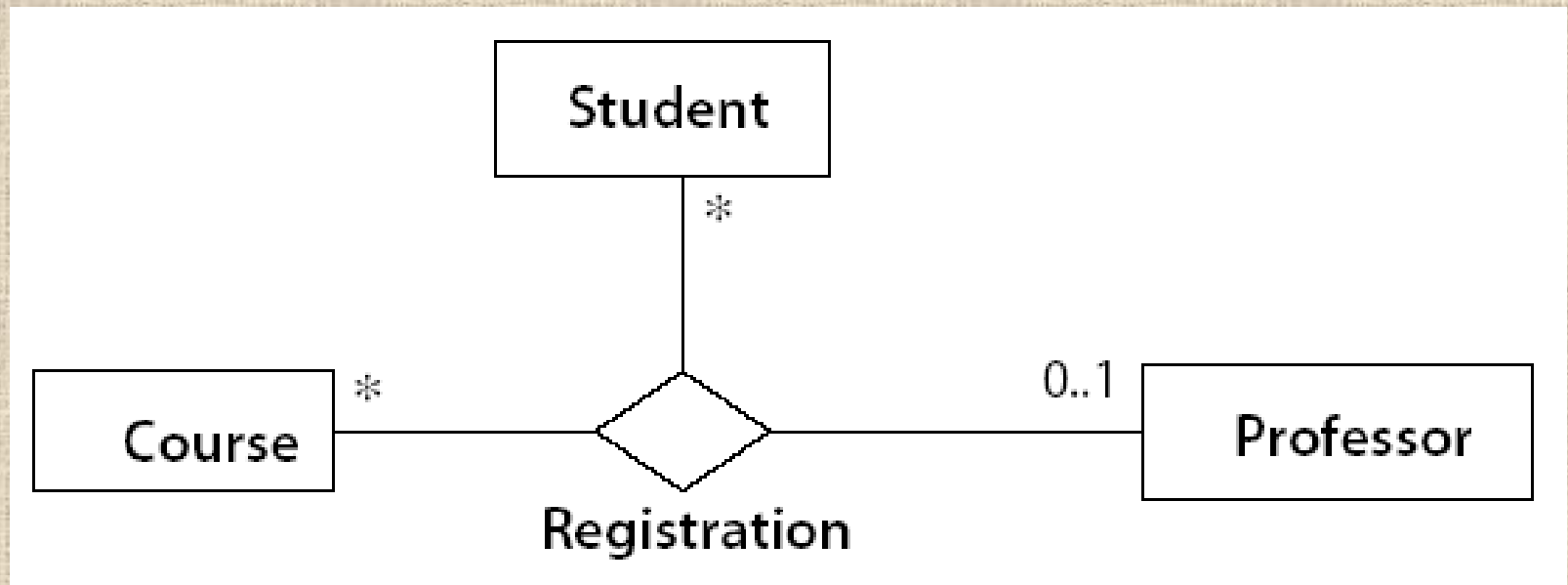
Варианты отображения навигации и кратности у концов ассоциации



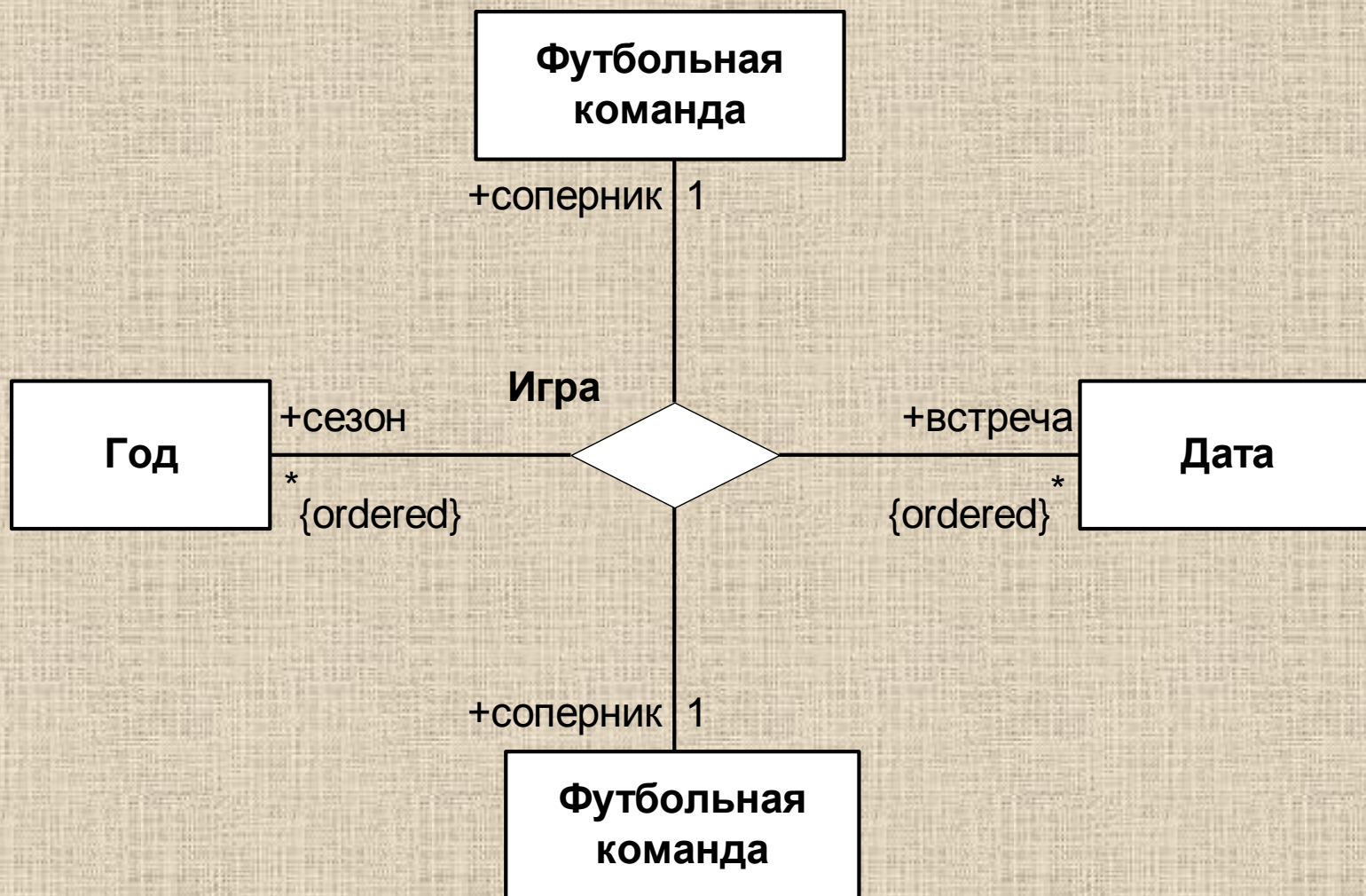
Исключающая ассоциация между тремя классами



Пример тернарной ассоциации

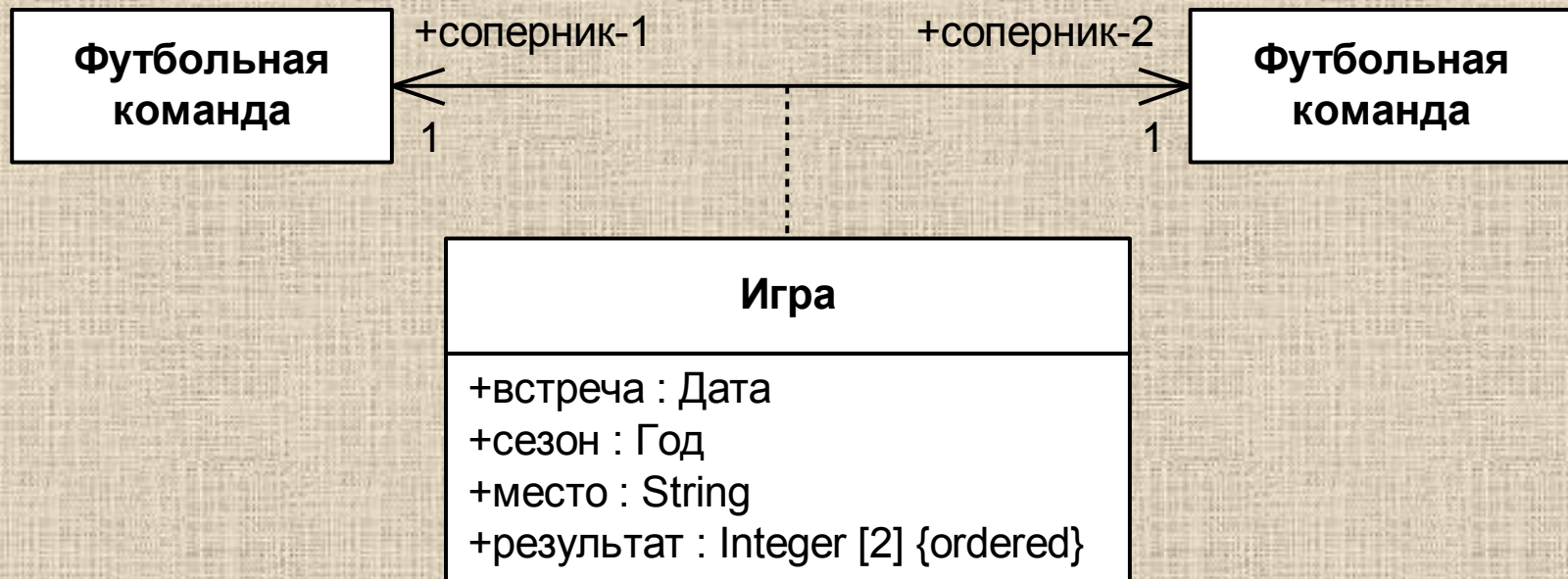


Пример четырехарной ассоциации



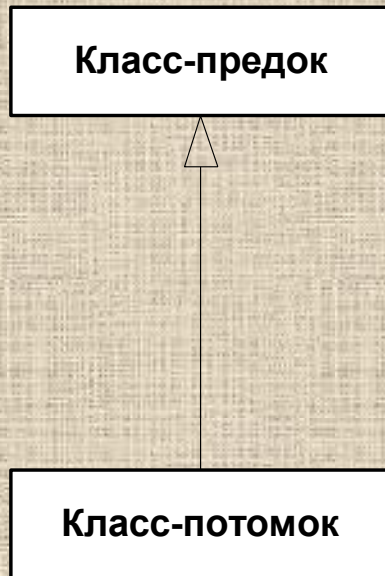
Ассоциация класс

элемент модели, который имеет свойства как ассоциации, так и класса, и предназначенный для спецификации дополнительных свойств ассоциации в форме атрибутов и, возможно, операций класса.

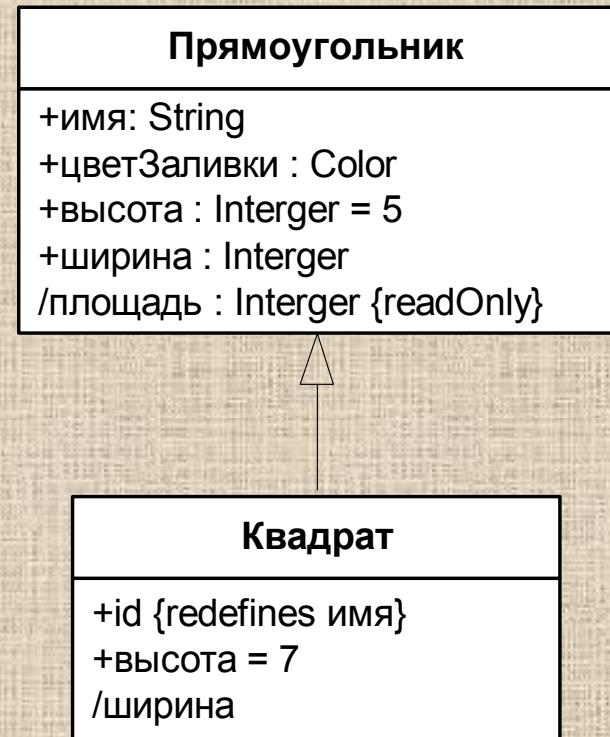


Обобщение

отношение между более общим классом (родителем или предком) и более специальным классом (дочерним или потомком)



(a)

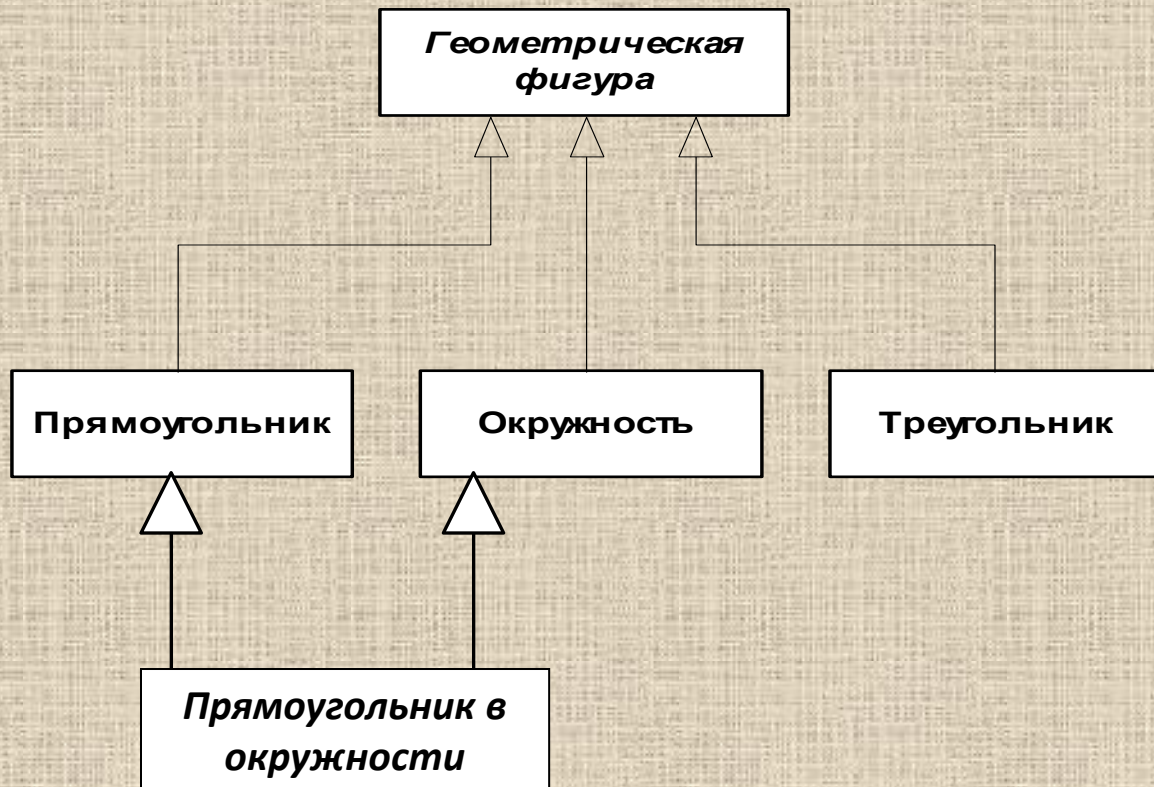


(б)

Примеры отношения обобщения



Множественное наследование – в языке UML разрешено



Агрегация

**направленное отношение между двумя классами,
предназначенное для представления ситуации, когда один из
классов представляет собой некоторую сущность, которая
включает в себя в качестве составных частей другие сущности**



Пример отношения агрегации

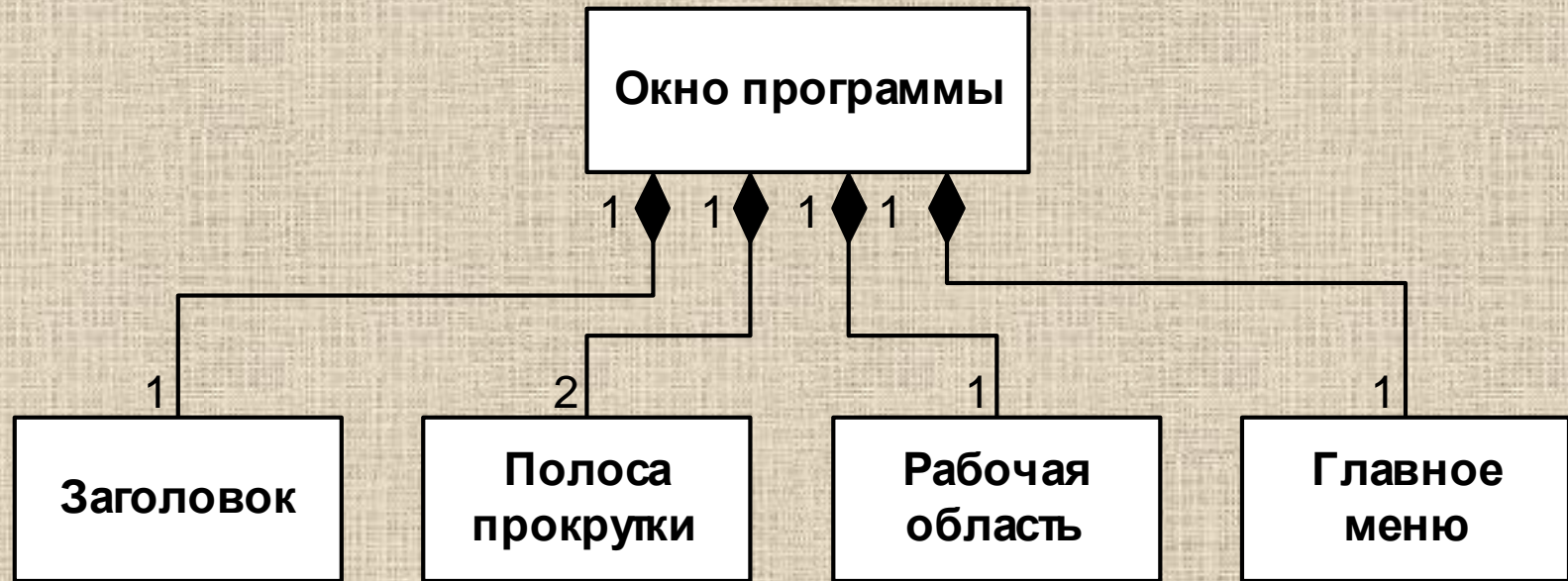


Композиция

композиционная агрегация предназначена для спецификации более сильной формы отношения "часть-целое", при которой с уничтожением объекта класса-контейнера уничтожаются и все объекты, являющиеся его составными частями.

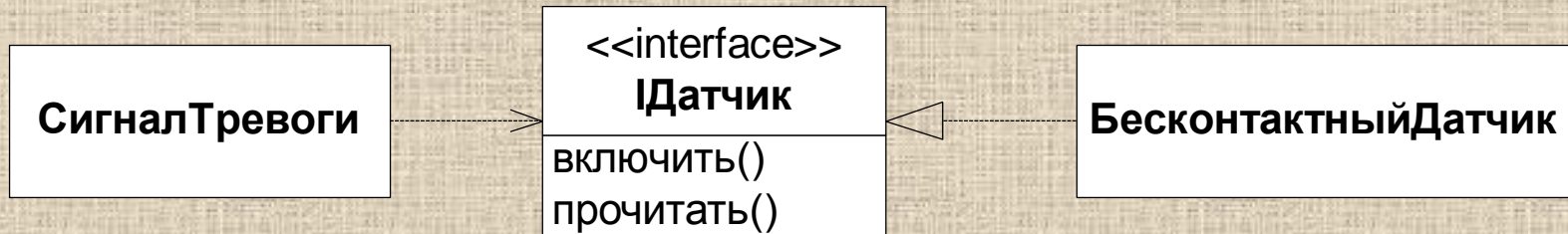
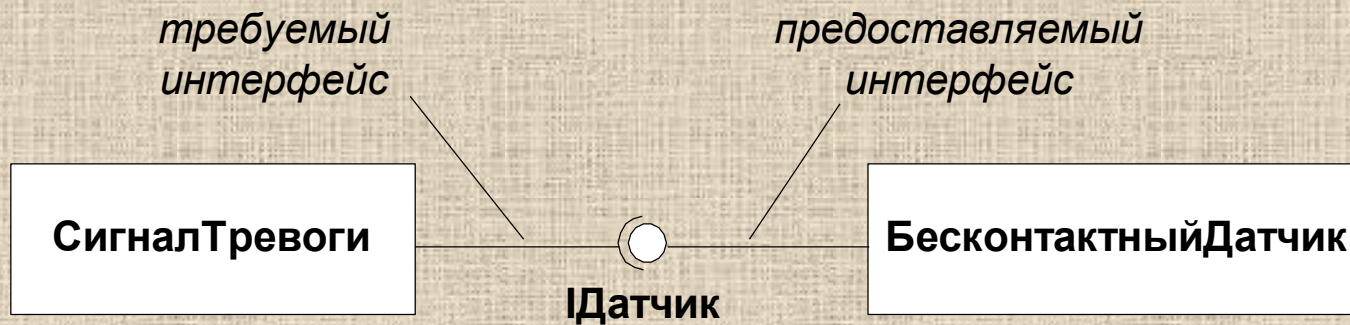


Пример отношения композиции



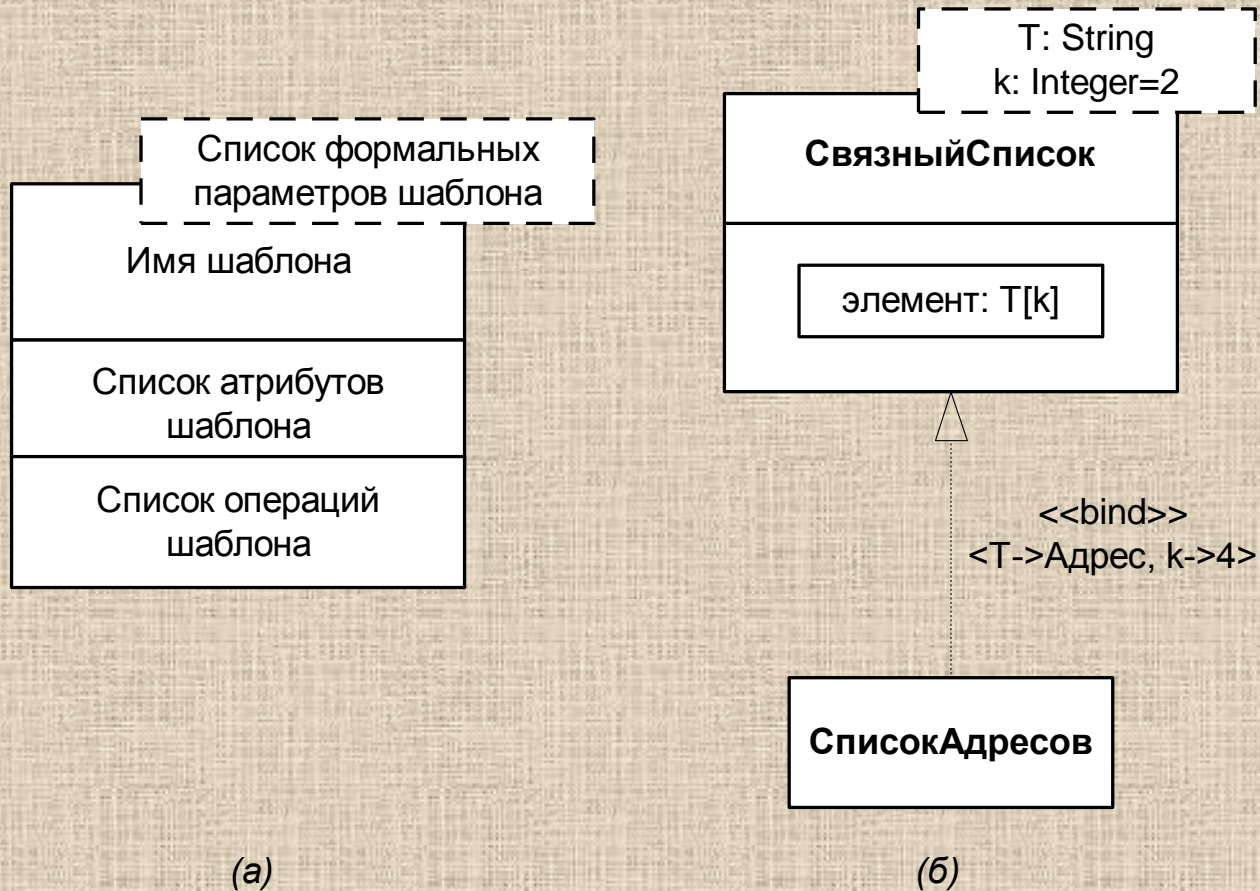
Интерфейс

вид класса, который представляет собой объявление множества общедоступных характеристик и обязанностей



Шаблон

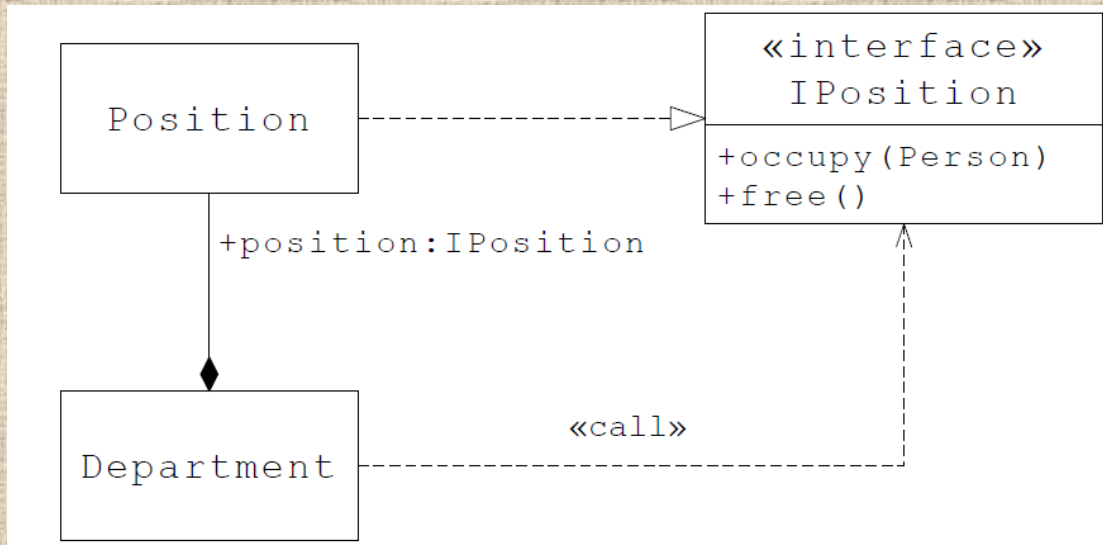
**класс, который в своем описании имеет
несколько формальных параметров**



Зависимость

Ключевое слово	Значение
«call» (вызывать)	Источник вызывает операцию в цели
«create» (создавать)	Источник создает экземпляр цели
«derive» (производить)	Источник представляет собой производное цели
«instantiate» (создать экземпляр)	Источник является экземпляром цели. (Обратите внимание, что если источник является классом, то сам класс является экземпляром класса класс; то есть целевой класс – это метакласс)
«permit» (разрешать)	Цель разрешает источнику доступ к ее закрытой функциональности
«realize» (реализовать)	Источник является реализацией спецификации или интерфейса, определенного целью
«refine» (уточнить)	Уточнение означает отношение между различными семантическими уровнями; например, источник может быть классом разработки, а цель – соответствующим классом анализа
«substitute» (заменить)	Источник может быть заменен целью
«trace» (проследить)	Используется, чтобы отследить такие моменты, как требования к классам или как изменения одной ссылки модели влияют на все остальное
«use» (использовать)	Для реализации источника требуется цель

Пример отображения зависимости



**Фрагмент
диаграммы классов
информационной
системы отдела
кадров**

Допустим, что класс **Department** для реализации операций связанных с движением кадров, использует операции класса **Position**, позволяющие занимать и освобождать должность — другие операции класса **Position** классу **Department** не нужны. Для этого можно определить соответствующий интерфейс **IPosition** и связать его отношениями с данными классами.

Типичные примеры применения диаграммы классов

для моделирования словаря системы: Моделирование словаря системы предполагает принятие решения о том, какие абстракции являются частью системы, а какие - нет. С помощью диаграмм классов вы можете определить эти абстракции и их обязанности.

для моделирования простых коопераций: Кооперация - это сообщество классов, интерфейсов и других элементов, работающих совместно для обеспечения некоторого кооперативного поведения, более значимого, чем сумма составляющих его элементов. Например, моделируя семантику транзакций в распределенной системе, вы не сможете понять происходящие процессы, глядя на один-единственный класс, поскольку соответствующая семантика обеспечивается несколькими совместно работающими классами. С помощью диаграмм классов удастся визуализировать и специфицировать эти классы и отношения между ними.

для моделирования логической схемы базы данных: Логическую схему можно представлять себе как чертеж концептуального проекта базы данных. Во многих сферах деятельности требуется хранить устойчивую (persistent) информацию в реляционной или объектно-ориентированной базе данных. Моделировать схемы также можно с помощью диаграмм классов.

Всегда ли надо создавать новые классы для новых задач?



Как использовать ранее разработанные классы?

- 1. Отношение обобщения (наследование)**
- 2. Агрегация (композиция)**

1. Использование ранее принятых решений. Зачем начинать все "с нуля", повторяя уже однажды сделанные действия?
2. Делаем решение мобильным и расширяемым. Используя уже существующие классы и создавая на их основе новые, мы можем развивать решение, добавляя лишь необходимые нам в данный момент детали - атрибуты и операции.
3. Существующие классы, как правило, хорошо отлажены и показали себя в работе. Разработчику не надо тратить время на кодирование, отладку, тестирование и т. д., - мы работаем с хорошо отлаженным и проверенным временем кодом, который зарекомендовал себя в других проектах и в котором уже выявлено и исправлено большинство ошибок.

Прямое и обратное проектирование диаграммы классов

Прямым проектированием (Forward engineering) называется процесс преобразования модели в код путем отображения на некоторый язык реализации. Процесс прямого проектирования приводит к потере информации, поскольку написанные на языке UML модели семантически богаче любого из существующих объектно-ориентированных языков.

Обратным проектированием (Reverse engineering) называется процесс преобразования в модель кода, записанного на каком-либо языке программирования. В результате этого процесса вы получаете огромный объем информации, часть которой находится на более низком уровне детализации, чем необходимо для построения полезных моделей. В то же время обратное проектирование никогда не бывает полным.

Хорошо структурированная диаграмма классов

Создавая диаграмму классов на языке UML, помните, что это всего лишь графическое изображение статического вида системы с точки зрения проектирования. Взятая в отрыве от остальных, ни одна диаграмма классов не может и не должна охватывать этот вид целиком. Диаграммы классов описывают его исчерпывающе, но каждая в отдельности - лишь один из его аспектов.

1. Заостряет внимание только на одном аспекте статического вида системы с точки зрения проектирования.
2. Содержит лишь элементы, существенные для понимания данного аспекта.
3. Показывает детали, соответствующие требуемому уровню абстракции, опуская те, без которых можно обойтись;
4. Не настолько лаконична, чтобы ввести читателя в заблуждение относительно важных аспектов семантики.

Полезные правила разработки диаграмм классов

1. Дайте диаграмме имя, связанное с ее назначением.
2. Располагайте элементы так, чтобы свести к минимуму число пересекающиеся линий.
3. Пространственно организуйте элементы так, чтобы семантически близкие сущности располагались рядом.
4. Чтобы привлечь внимание к важным особенностям диаграммы, используйте примечания и цвет.
5. Старайтесь не показывать слишком много разных видов отношений; как правило, в каждой диаграмме классов должны доминировать отношения какого-либо одного вида.

ДИАГРАММА ОБЪЕКТОВ

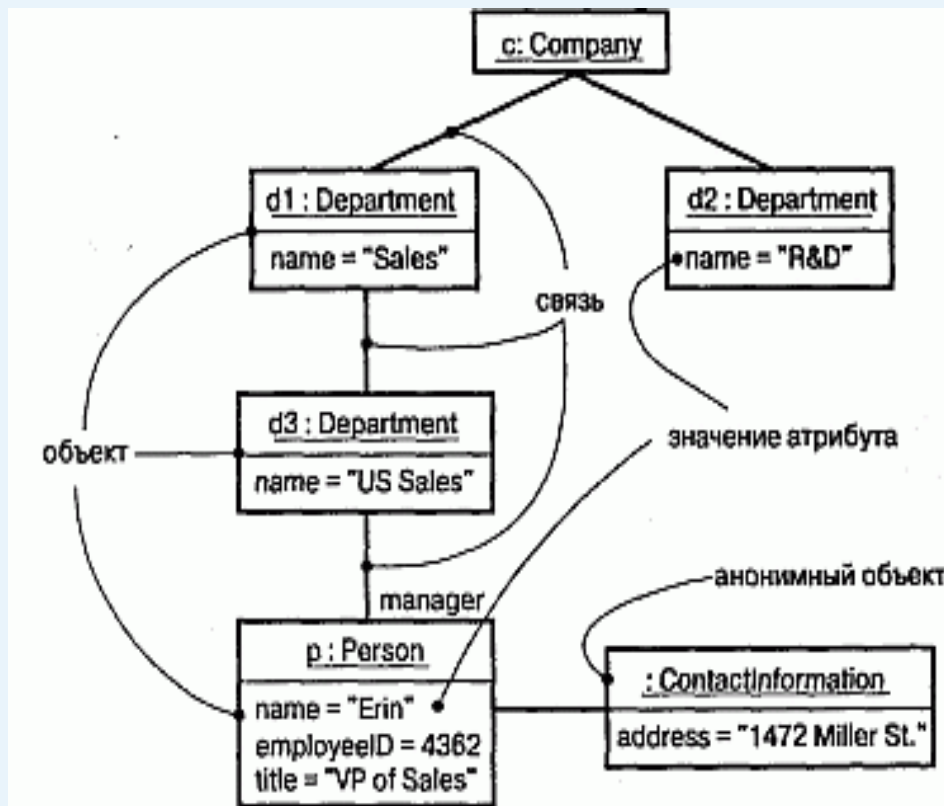
является экземпляром диаграммы классов

Диаграммы объектов позволяют моделировать экземпляры сущностей, которые содержатся в диаграммах классов. На диаграмме объектов показано множество объектов и отношений между ними в некоторый момент времени.



«Снимок» программы в фиксированный момент времени

Диаграммой объектов (Object diagram) называется диаграмма, на которой показаны объекты и их отношения в некоторый момент времени. Графически диаграмму объектов представляют в виде графа, состоящего из вершин (объектов) и ребер (отношения).



Объект (object) является отдельным экземпляром класса, который создается на этапе реализации модели или выполнения программы.

Примеры графического изображения объектов

- o: C — для объекта специфицировано собственное имя объекта и имя класса.
- o — для объекта специфицировано только собственное имя объекта.
- : C — для объекта специфицировано только имя класса (анонимный объект).

квадрат : Прямоугольник

квадрат

: Прямоугольник



Примеры графического изображения объектов

Point
x: Real y: Real
rotate (angle: Real) scale (factor: Real)

descriptor (a class)

<u>p1: Point</u>
x = 3.14 y = 2.718

instances (two objects)

Point object named p1

The instances may show values.

<u>:Point</u>
x = 1 y = 1.414

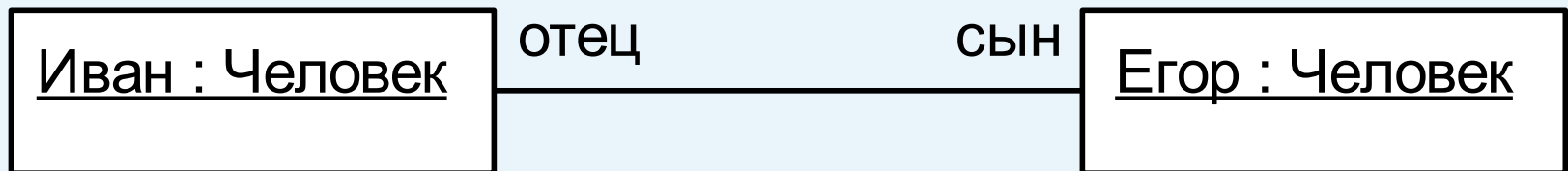
anonymous Point object

No need to show fixed parts such as operations shared by all instances.



Связь (link)

является экземпляром произвольной ассоциации,
которая обеспечивает канал для направленной
передачи сообщений между объектами



Слот (slot)

предназначен для представления того, что сущность (объект класса), имеет конкретное значение или значения для некоторой своей структурной характеристики

<u>мойАдрес: Адрес</u>
имяУлицы: String="Садовая" номерДома : Integer = 25 номерКвартиры : Integer = 15

<u>: Квадрат</u>
вершина = (1, 10) стороны = (25, 10) цветГраницы = черный цветЗаливки = белый

отсутствие слота для некоторой характеристики в спецификации экземпляра не означает, что представляемая сущность не имеет этой характеристики, но означает только лишь то, что эта характеристика не представляет интереса в модели

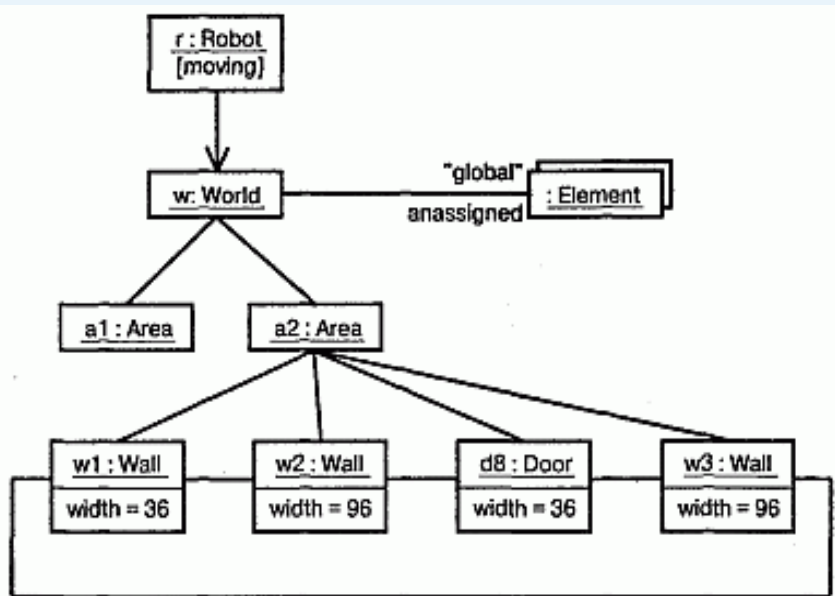


Моделирование объектной структуры

1. Идентифицируйте механизм, который собираетесь моделировать. Механизм представляет собой некоторую функцию или поведение части моделируемой системы, являющееся результатом взаимодействия сообщества классов, интерфейсов и других сущностей.
2. Для каждого обнаруженного механизма идентифицируйте классы, интерфейсы и другие элементы, участвующие в кооперации, а также отношения между ними.
3. Рассмотрите один из сценариев использования работы механизма. Заморозьте этот сценарий в некоторый момент времени и изобразите все объекты, участвующие в механизме.
4. Покажите состояние и значения атрибутов каждого такого объекта, если это необходимо для понимания сценария.
5. Покажите также связи между этими объектами, которые представляют экземпляры существующих ассоциаций.



Пример диаграммы объектов



**часть механизма робота,
предназначенного для расчета
модели мира, в котором тот
перемещается**

Как видно из диаграммы, робот распознал, что замкнутое помещение, в котором он находится, имеет с трех сторон стены, а с четвертой - дверь

Один из объектов соответствует самому роботу (r, экземпляр класса Robot); в настоящий момент он находится в состоянии moving (движется). Этот объект связан с экземпляром w класса World (Мир), являющегося абстракцией модели мира робота.

Объект w связан с мультиобъектом, который состоит из экземпляров класса Element, описывающего сущности, опознанные роботом, но еще не включенные в его модель мира. Эти элементы помечены как части глобального состояния робота.

В текущий момент времени экземпляр w связан с двумя экземплярами класса Area. У одного из них (a2) показаны его собственные связи с тремя объектами класса Wall (Стена) и одним - класса Door (Дверь). Указана ширина каждой из трех стен и обозначено, что каждая связана с соседними.

Обратное проектирование диаграммы объектов

1. Выберите, что именно вы хотите реконструировать. Обычно базовой точкой является какая-либо операция или экземпляр конкретного класса.
2. С помощью инструментальных средств или просто пройдясь по сценарию, зафиксируйте выполнение системы в некоторый момент времени.
3. Идентифицируйте множество интересующих вас объектов, сотрудничающих в данном контексте, и изобразите их на диаграмме объектов.
4. Если это необходимо для понимания семантики, покажите состояния объектов.
5. Чтобы обеспечить понимание семантики, идентифицируйте связи, существующие между объектами.
6. Если диаграмма оказалась слишком сложной, упростите ее, убрав объекты, не существенные для прояснения данного сценария. Если диаграмма слишком проста, включите в нее окружение некоторых представляющих интерес объектов и подробнее покажите состояние каждого объекта.



Характеристики хорошо структурированной диаграммы объектов

- ✓ акцентирует внимание на одном аспекте статического вида системы, с точки зрения проектирования или процессов;
- ✓ представляет лишь один из кадров динамического сценария, показанного на диаграмме взаимодействия;
- ✓ содержит только существенные для понимания данного аспекта элементы;
- ✓ уровень ее детализации соответствует уровню абстракции системы (показывайте только те значения атрибутов и дополнения, которые существенны для понимания);
- ✓ не настолько лаконична, чтобы ввести читателя в заблуждение относительно важной семантики



Правила построения диаграмм объектов

- давайте ей имя, соответствующее назначению;
- располагайте элементы так, чтобы число пересечений было минимальным;
- располагайте элементы так, чтобы семантически близкие сущности оказывались рядом;
- используйте примечания и цвет для привлечения внимания к важным особенностям диаграммы;
- включайте в описания каждого объекта значения, состояния и роли, если это необходимо для понимания ваших намерений



МОДЕЛИРОВАНИЕ ПОВЕДЕНИЯ

Модель вариантов использования - «ЧТО делает система?»»

Модели классов и объектов – «ИЗ ЧЕГО состоит система?»»

Модель поведения – это АЛГОРИТМ РАБОТЫ системы.

Средства моделирования поведения в UML:

1. Описание поведения с явным выделением состояний, задается **диаграммами автомата (диаграммы состояний)**.
2. Описание поведения с явным выделением потоков данных и управления, задается **диаграммами деятельности**.
3. Описание поведения как упорядоченной последовательности сообщений задается **диаграммами взаимодействия в четырех формах**;
4. Описание параллельного поведения, задается **специальными средствами на каждой из диаграмм** описывающих поведение.



ДИАГРАММА СОСТОЯНИЙ (ДИАГРАММА КОНЕЧНОГО АВТОМАТА)

назначение - описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла

Диаграмма состояний является графом специального вида, который представляет некоторый автомат.

Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние.



Пример диаграммы состояний

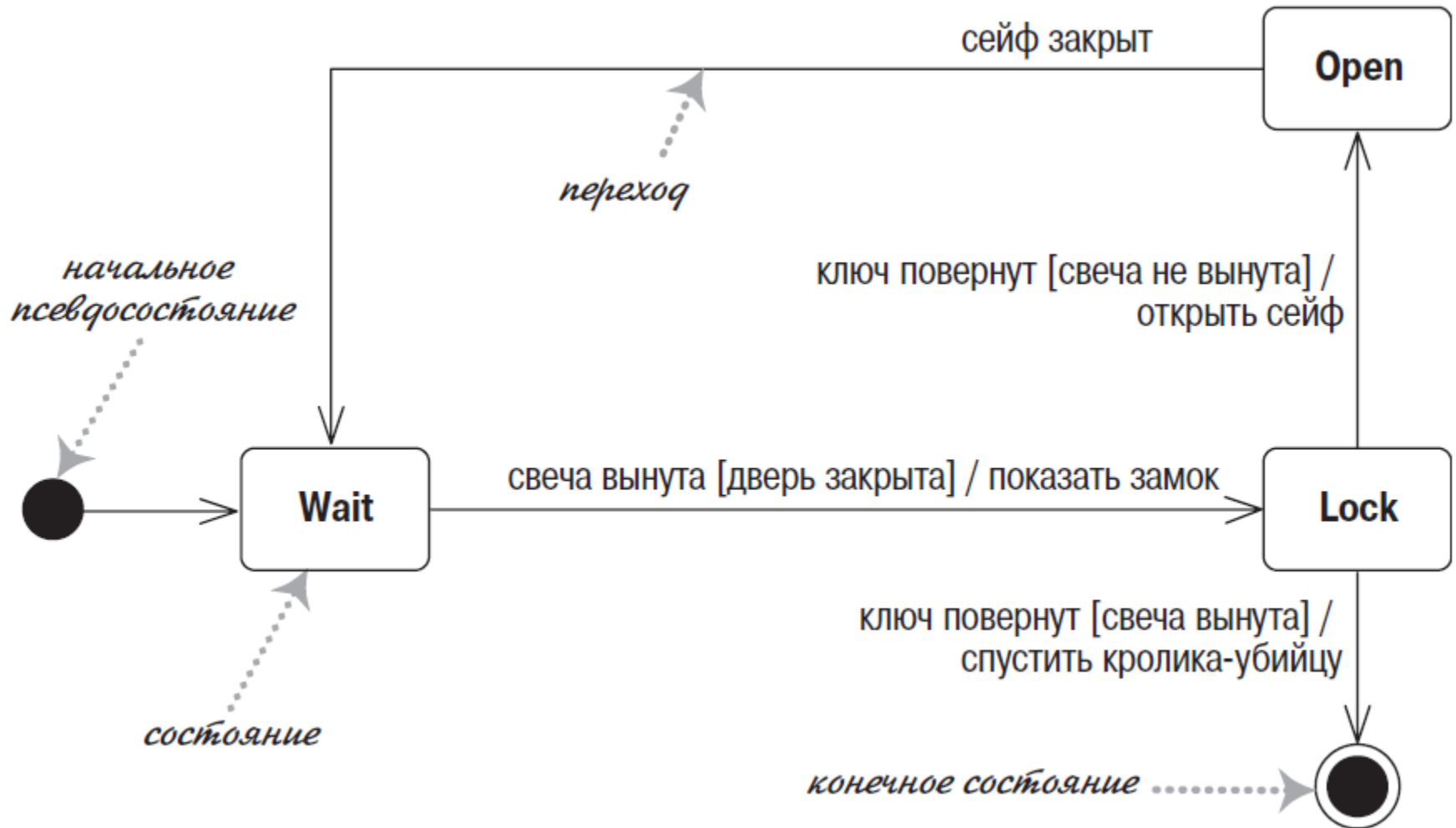


Диаграмма состояний: основные понятия

Автомат (State machine) описывает поведение в терминах последовательности состояний, через которые проходит объект в течение своего жизненного цикла, отвечая на различные события, а также его реакции на эти события.

Состояние (State) - это ситуация в жизни объекта, на протяжении которой он удовлетворяет некоторому условию (инварианту), выполняет определенную деятельность или ожидает какого-то события.

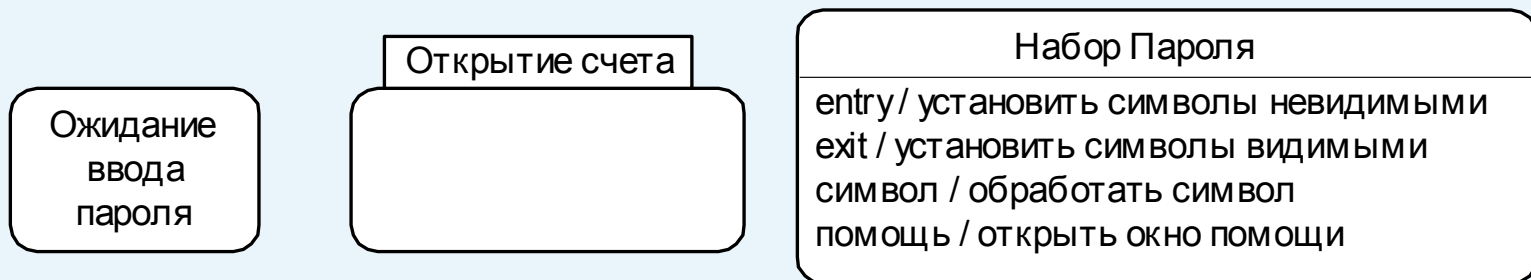
Событие (Event) - это спецификация существенного факта, имеющего место в пространстве и во времени, инициирующего переход из одного состояния в другое.

Переход (Transition) - это отношение между двумя состояниями, показывающее, что объект, находящийся в первом состоянии, должен выполнить определенные действия и перейти во второе состояние, как только произойдет указанное событие и будут удовлетворены определенные условия.



Простое состояние

- Состояние, которое не имеет внутренних подсостояний
- На одной диаграмме нежелательно показывать одно и то же именованное состояние дважды, поскольку это может привести к недоразумению



Структура простого состояния

- **имя** (name);
- **действие при входе** (entry action);
- **действие при выходе** (exit action);
- описание множества **внутренних переходов** (internal transitions) и соответствующих **действий**;
- **внутренняя деятельность** (do activity);
- множество **отложенных событий** (defer events).



Простое состояние с внутренними действиями

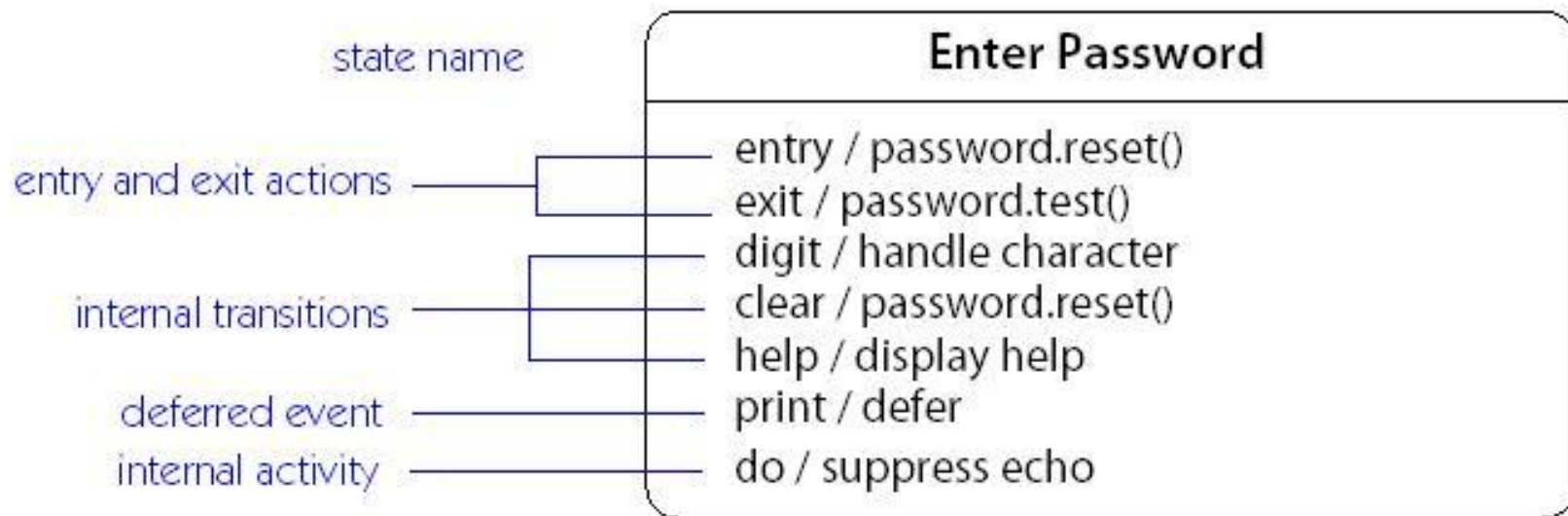
entry / action-expression

do / activity-expression

exit / action-expression

event-name / defer

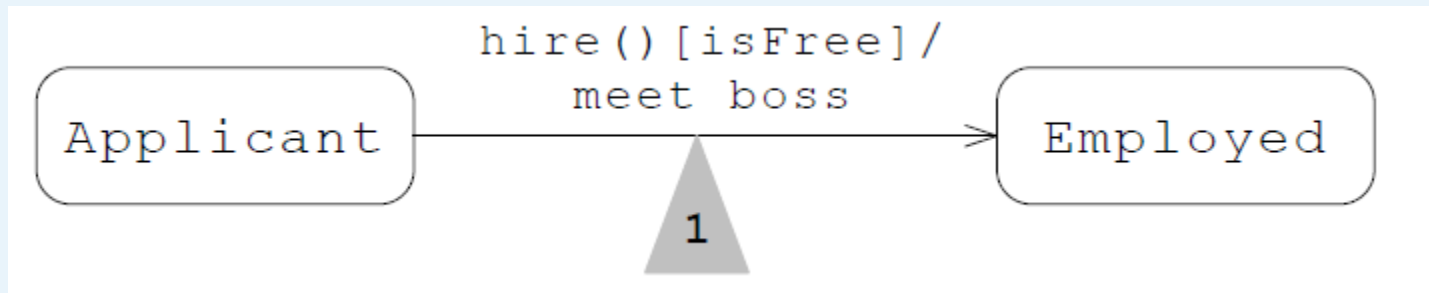
include machine-name(argument_{list},)opt



Простой переход

- Означает перемещение из одного состояния в другое
- Каждый переход имеет метку, состоящую из следующих частей:

Событие [Сторожевое условие] / Действие

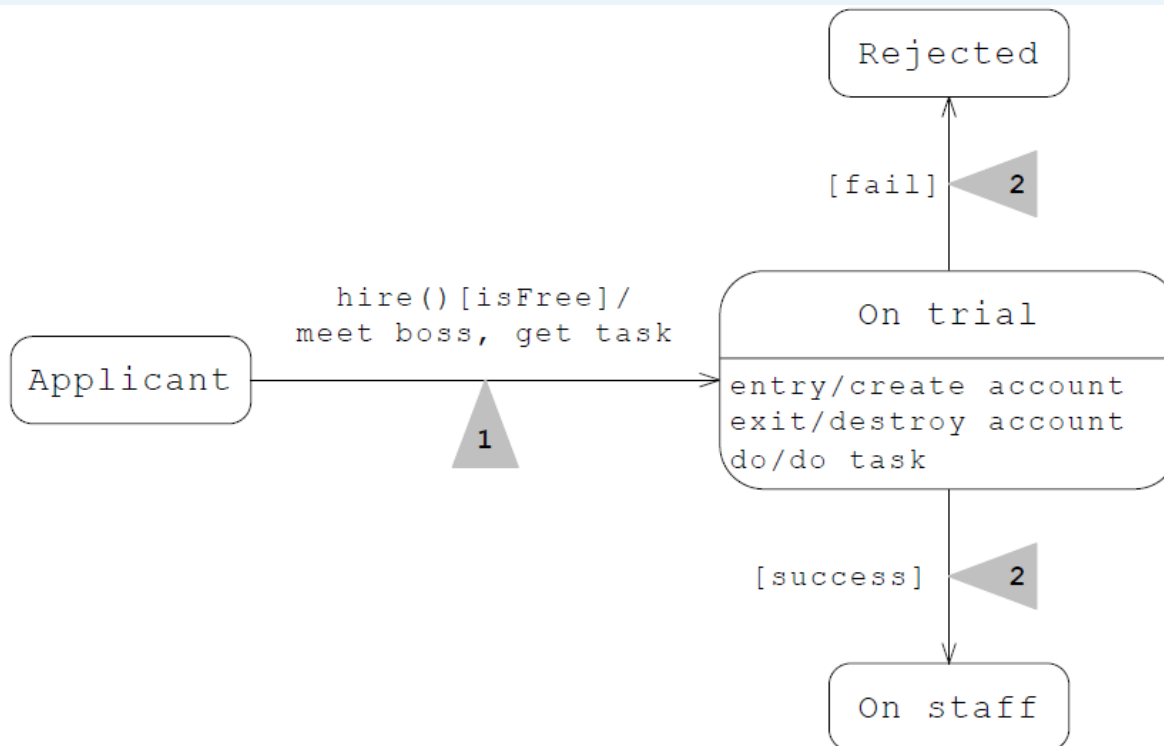


Событие перехода

Событие перехода (trigger event) - входной символ (стимул), который вкупе с текущим состоянием автомата определяет следующее состояние.

UML допускает переходы без событий — переходы по завершении.

Переход по завершении (completion transition) — это переход, который происходит по окончании внутренней деятельности.



1 – простой переход

2 – переход по завершении

Сторожевое условие перехода

Сторожевое условие (guard) — это логическое выражение, которое должно оказаться истинным для того, чтобы возбужденный переход сработал.

Значение сторожевого условия вычислить заранее, на этапе моделирования, невозможно. Сторожевое условие должно проверяться динамически, во время выполнения.

Если сторожевое условие ложно, то переход не срабатывает и событие теряется. Даже если впоследствии сторожевое условие станет истинным, переход сможет сработать, только если повторно возникнет событие перехода.



Действие при переходе

Действие (action) — это непрерываемое извне атомарное вычисление, чье время выполнения пренебрежимо мало.

Действие является атомарным и непрерываемым.

При выполнении действия на переходе или в состоянии не могут происходить события, прерывающие выполнение действия. Точнее говоря, событие может произойти, но система обязана задержать его обработку до окончания выполнения действия.

Действие является безальтернативным и завершаемым.

Раз начавшись, действие выполняется до конца. Оно не может "раздумать" выполняться или выполняться неопределенно долго.



Сегментированные переходы

Сегменты перехода (transition segment) — это части, на которые может быть разбита линия перехода.

Разбивающими элементами являются следующие фигуры:

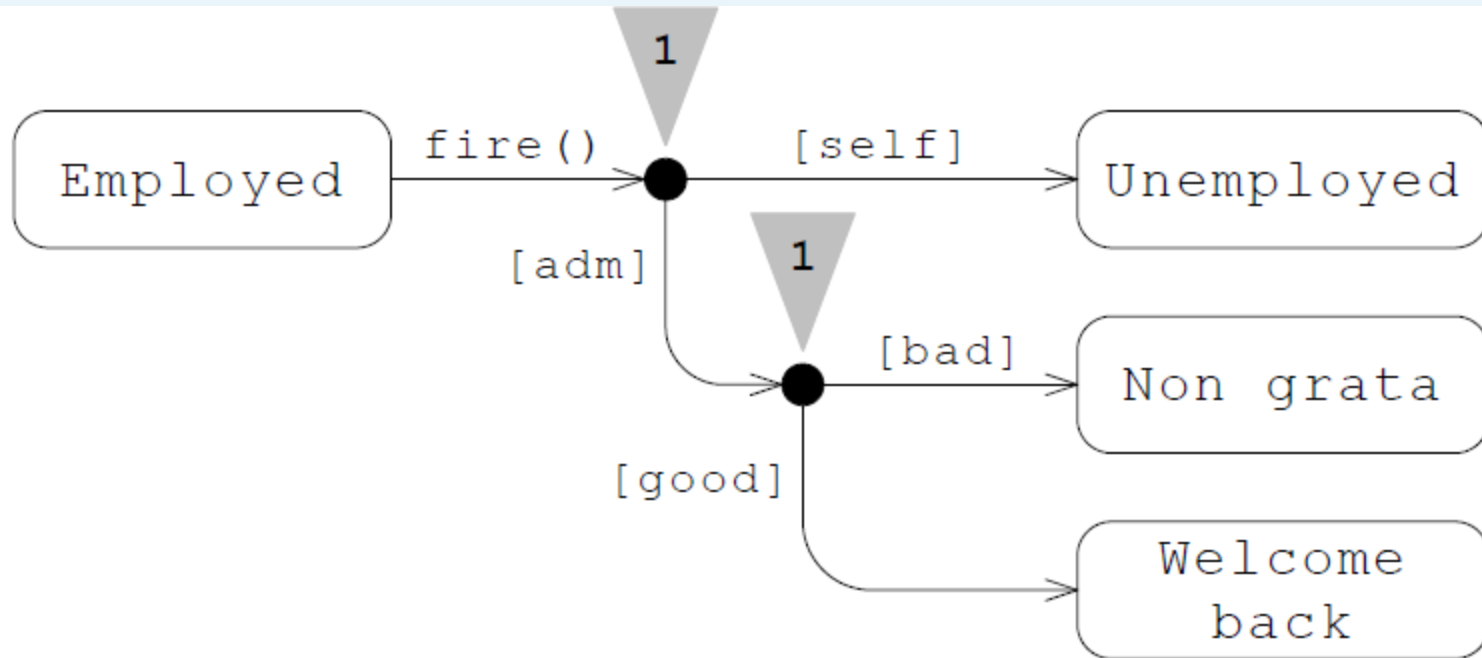
- **переходное состояние** (junction state)
(изображается в виде небольшого кружка);

- **состояние выбора** (choice)
(изображается в виде ромба);

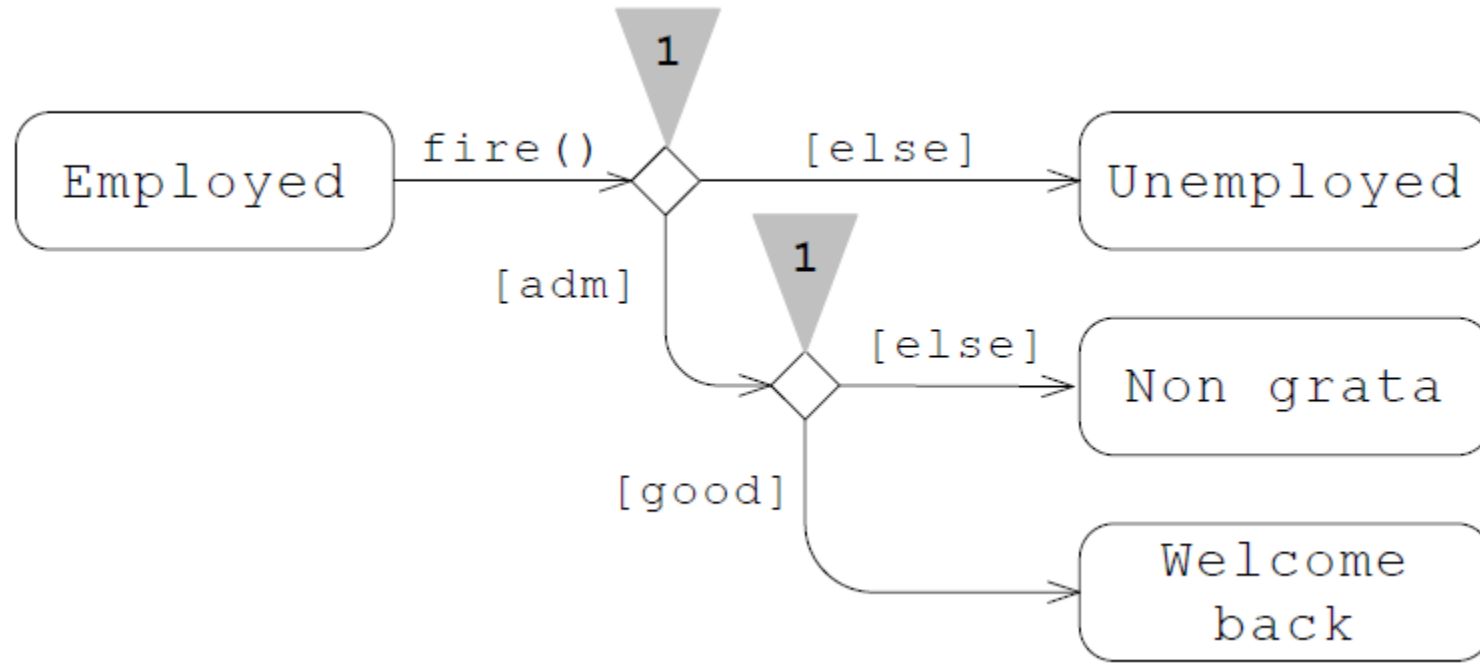
Несколько переходов, исходящих из данного состояния и имеющих общее событие перехода, можно объединить в дерево сегментированных переходов. Сегмент перехода, начинающийся в исходном состоянии, называется **корневым**, сегменты, заканчивающиеся в целевых состояниях, называются **листовыми**. Событие перехода может быть указано только для корневого сегмента, действия на переходе могут быть указаны только для листовых сегментов, а сторожевые условия могут быть указаны для любых сегментов.



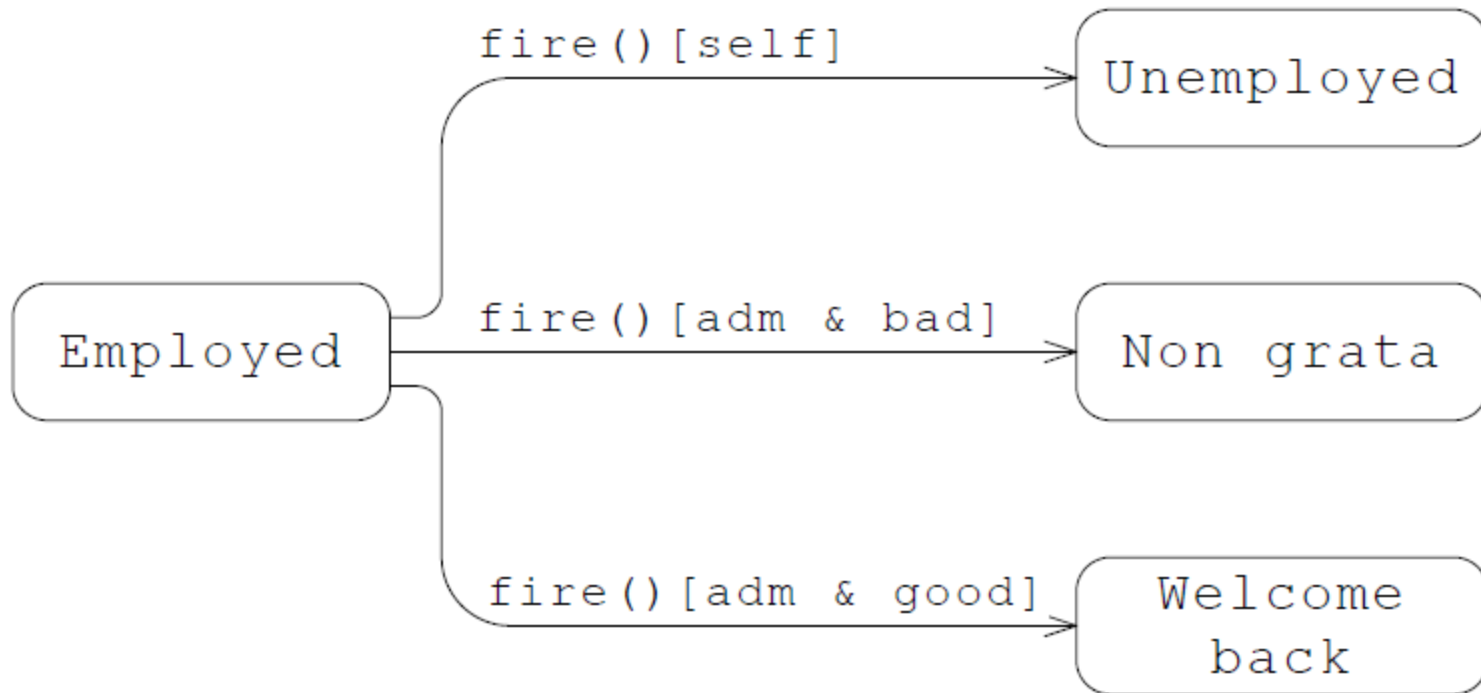
Пример сегментированного перехода с переходными состояниями



Пример сегментированного перехода с состояниями выбора



Та же семантика без сегментированных переходов



Составное состояние

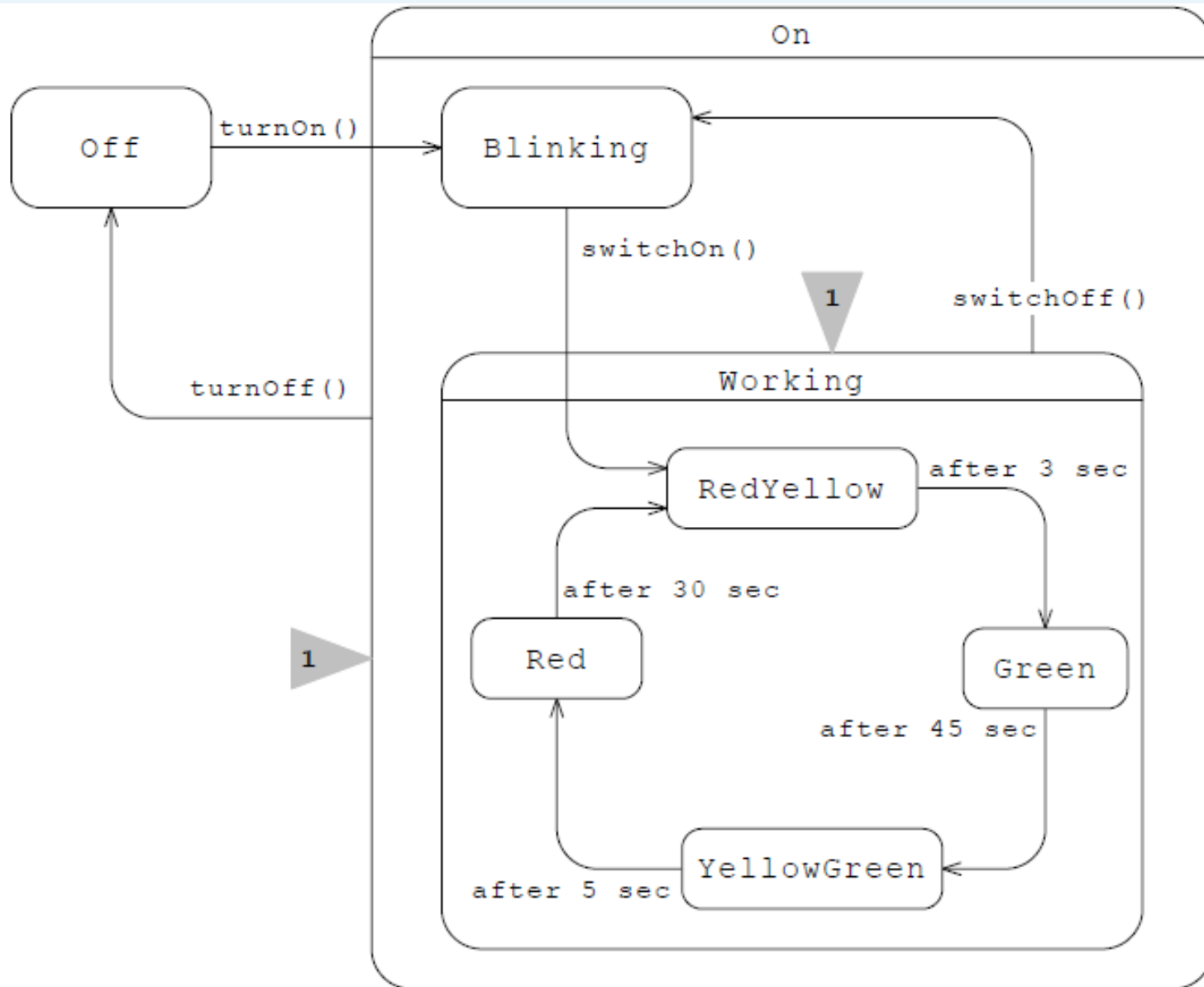
Составное состояние — это состояние, в которое вложена машина состояний.

Если вложена только одна машина, то состояние называется **последовательным**.

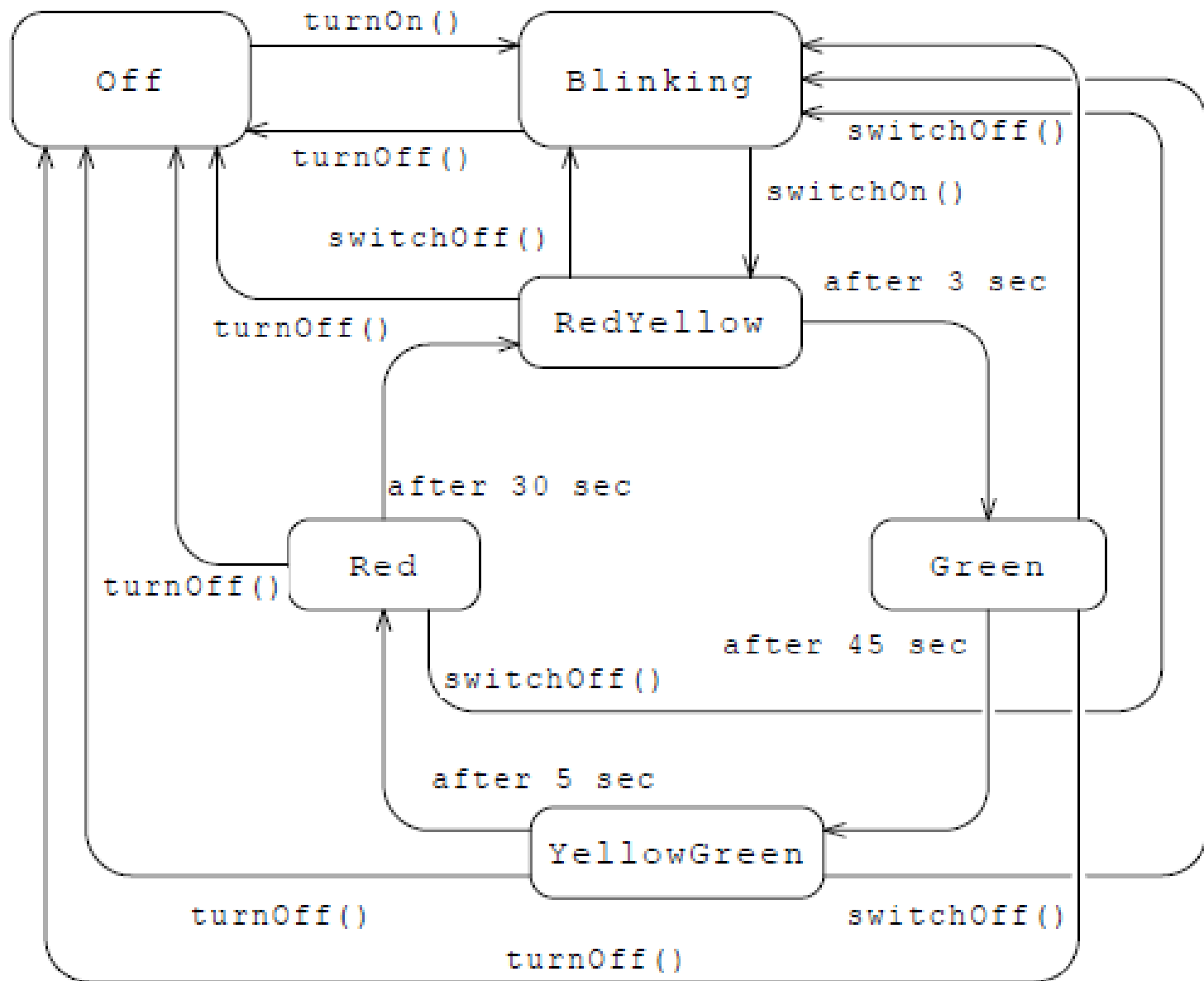
Если вложено несколько машин состояний — **параллельным**.
В UML 2 параллельные состояния переименованы в **ортогональные**.



Пример использования составного состояния: светофор



Модель светофора без составных состояний



Специальные состояния

специальное состояние состоянием не является, в том смысле, что автомат не может в нем "пребывать" и оно не может быть текущим активным состоянием

Начальное состояние (initial state) — это специальное состояние, соответствующее ситуации, когда машина состояний еще не работает. Изображается в виде закрашенного кружка. Не имеет таких составляющих, как действия на входе, выходе и внутренняя активность, но оно обязано иметь исходящий переход. Не может иметь события перехода, но может иметь сторожевое условие. Начальное состояние может иметь действие на переходе — это действие выполняется до начала работы машины состояний.

Заключительное состояние (final state) — это специальное состояние, соответствующее ситуации, когда машина состояний уже не работает. Изображается в виде закрашенного кружка, который обведен дополнительной окружностью. Не имеет таких составляющих, как действия на входе, выходе и внутренняя активность, но имеет входящий переход, ведущий из того состояния, которое является последним состоянием в данном сеансе работы конечного автомата.



Специальные состояния (продолжение)

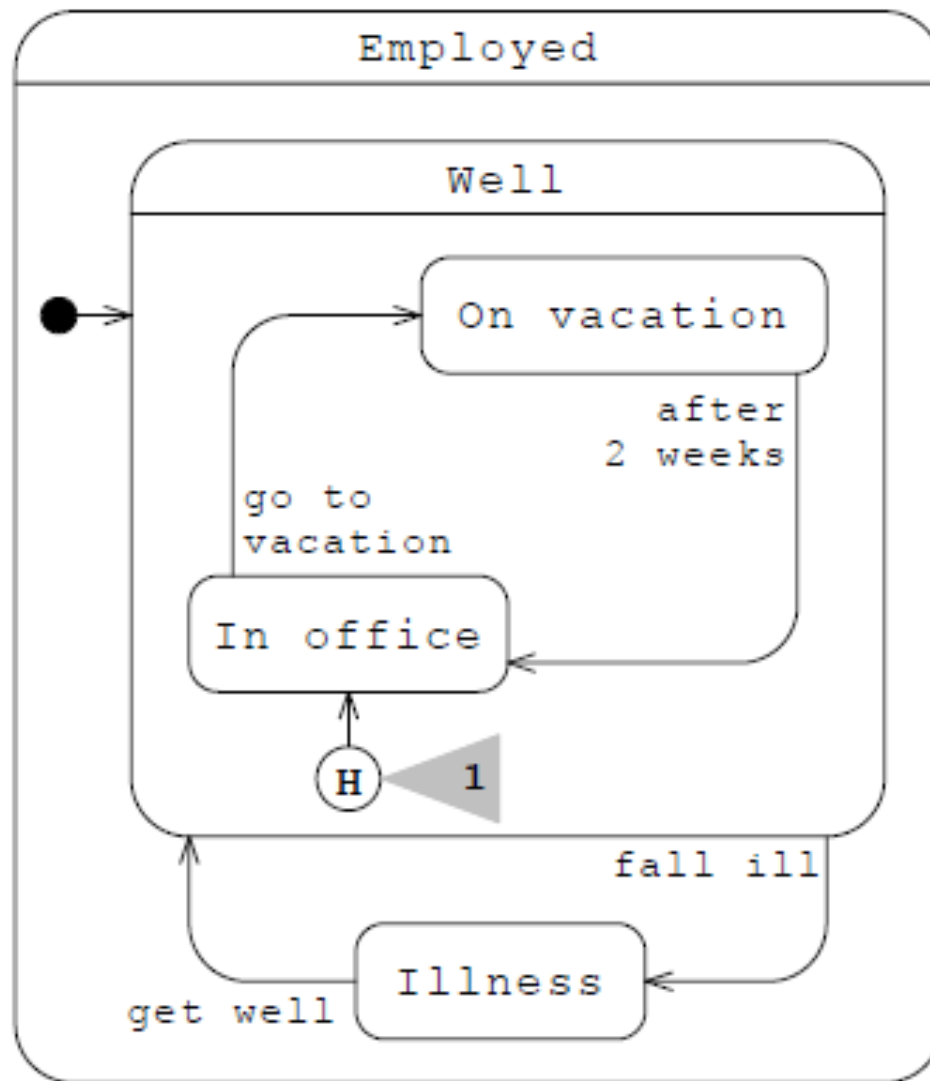
Историческое состояние может использоваться во вложенной машине состояний внутри составного состояния.

При первом запуске машины состояний историческое состояние означает в точности тоже, что и начальное: оно указывает на состояние, в котором находится машина в начале работы. Если в данной машине состояний используется историческое состояние, то при выходе из объемлющего составного состояния запоминается то состояние, в котором находилась вложенная машина перед выходом. При повторном входе в данное составное состояние в качестве текущего состояния восстанавливается то запомненное состояние, в котором машина находилась при выходе.

Историческое состояние заставляет автомат помнить, в каком состоянии его прервали в прошлый раз и "продолжать начатое".



Пример отображения специальных состояния



События перехода

Типы событий перехода:

- событие вызова,
- событие сигнала,
- событие таймера,
- событие изменения.

Событие вызова (call event) — это событие, возникающее при вызове метода класса.

Событие сигнала (signal event)— это событие, возникающее при посылке сигнала.

Событие таймера (time event)— это событие, которое возникает, когда истек заданный интервал времени с момента попадания автомата в данное состояние.

Событие изменения (change event)— это событие, которое возникает, когда некоторое логическое условие становится истинным, будучи до этого ложным.



Событие вызова: пример

Person

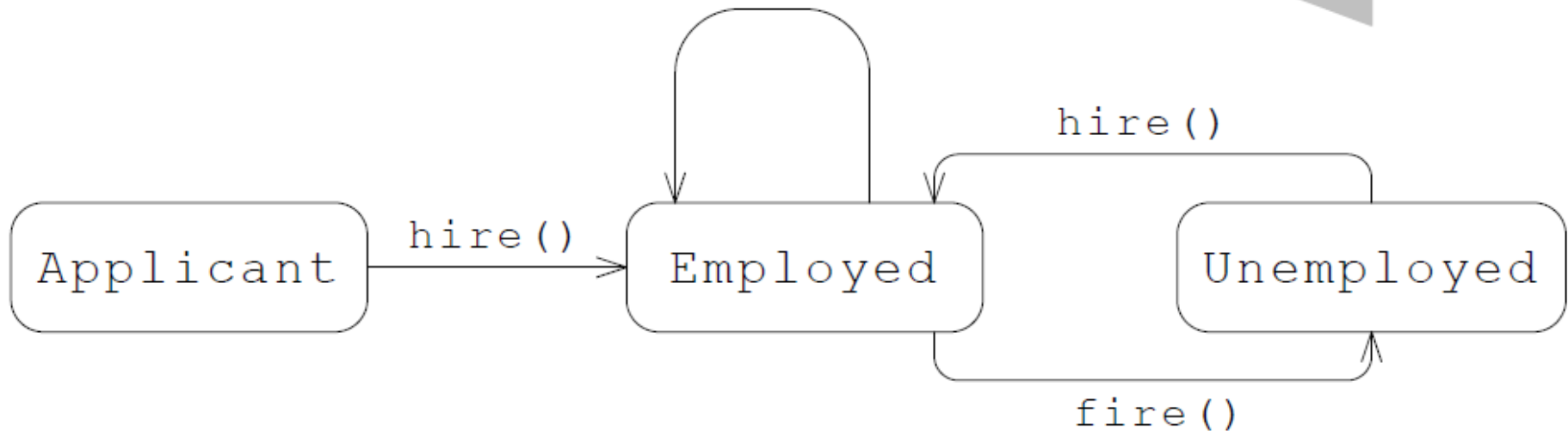
```
+assign(newPos:Position):Boolean  
+hasPosition():Boolean
```

Position

```
+occupy(person:Person):Boolean  
+isFree():Boolean
```

```
assign(newPos)[newPos.isFree()]/  
newPos.occupy(self),return(true)
```

1



Событие сигнал

Синтаксически сигнал — это экземпляр класса со стереотипом «signal».

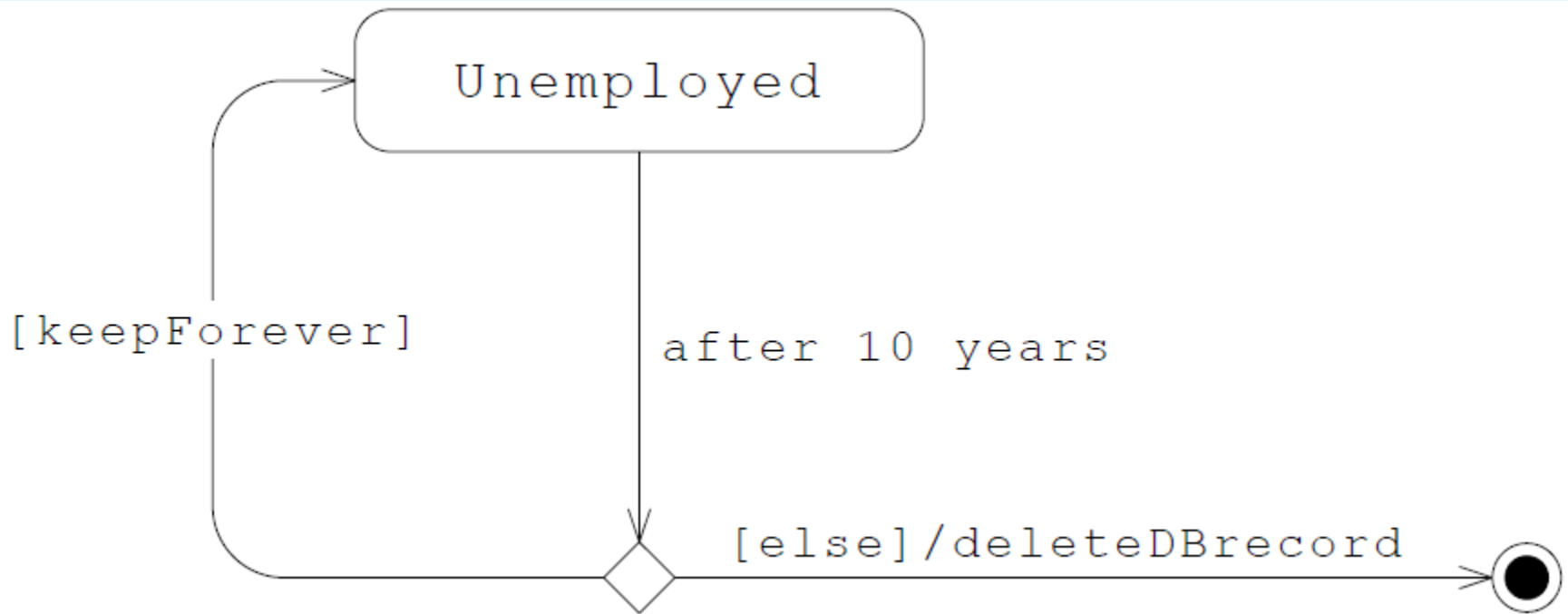
Семантически сигнал — это именованный объект, который создается другим объектом (отправителем) и обрабатывается третьим объектом (получателем).

Сигнал может иметь атрибуты (параметры). Сигнал может иметь операции. Одна из них считается определенной по умолчанию. Она имеет стандартное имя **send()** и параметр, являющийся множеством объектов, которым отправляется сигнал. Это операция — конструктор, который создает экземпляр классификатора, то есть сигнал.



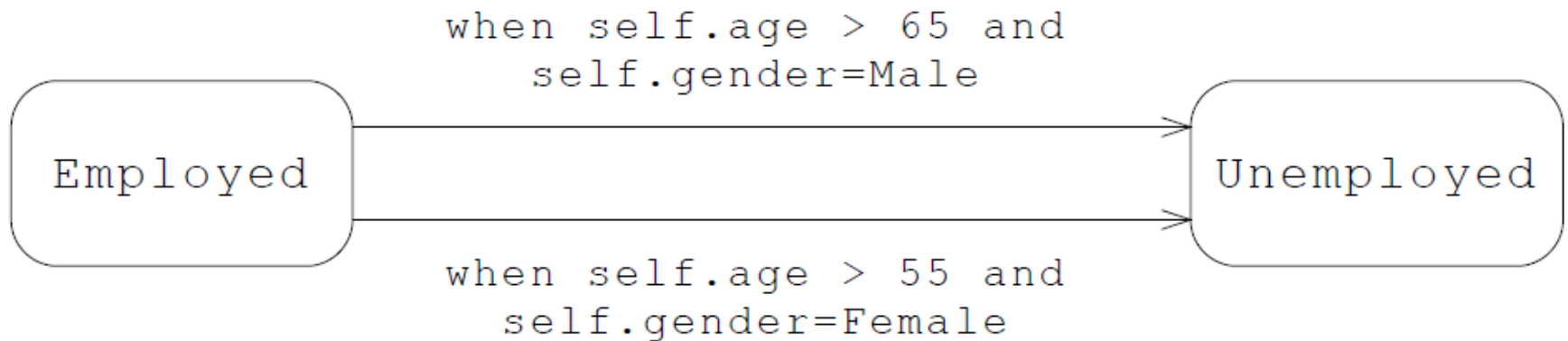
Событие таймера (ключевое слово `after`)

Информационная система должна хранить информацию не только о работающих, но и об уволенных сотрудниках. По прошествии 10 лет после увольнения, если эта информация не нужна более, она уничтожается.



Событие изменение

Событие изменения (change event)— это событие, которое возникает, когда некоторое логическое условие становится истинным, будучи до этого ложным.
Синтаксически событие изменения записывается с помощью ключевого слова **when**, за которым указывается логическое выражение (условие).



*По достижении определенного возраста
(55 лет для женщин и 60 лет для мужчин)
сотрудник увольняется на пенсию.*



Когда применяются диаграммы состояний

- Диаграммы состояний хороши для описания поведения одного объекта в нескольких прецедентах.
- Не старайтесь нарисовать их для каждого класса системы. Такой подход часто применяется в целях формально строгой полноты, но почти всегда это напрасная трата сил. Применяйте диаграммы состояний только для тех классов, которые проявляют интересное поведение, когда построение диаграммы состояний помогает понять, как все происходит.



ДИАГРАММА ДЕЯТЕЛЬНОСТИ

это технология, позволяющая описывать логику процедур и алгоритмов

- ✓ **акцентирует внимание на последовательности выполнения определенных действий или элементарных операций, которые в совокупности приводят к получению желаемого результата**
- ✓ **графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами – переходы от одного состояния действия к другому**
- ✓ **поддерживает моделирование параллельных процессов (важное отличие от привычных блок-схем)**



Пример диаграммы деятельности

Диаграмма деятельности позволяет выбирать порядок действий. Диаграмма только устанавливает правила обязательной последовательности действий, которым необходимо следовать. Это важно для моделирования бизнес-процессов, поскольку эти процессы часто выполняются параллельно.

Такие диаграммы также полезны при разработке параллельных алгоритмов, в которых независимые потоки могут выполнять работу параллельно.

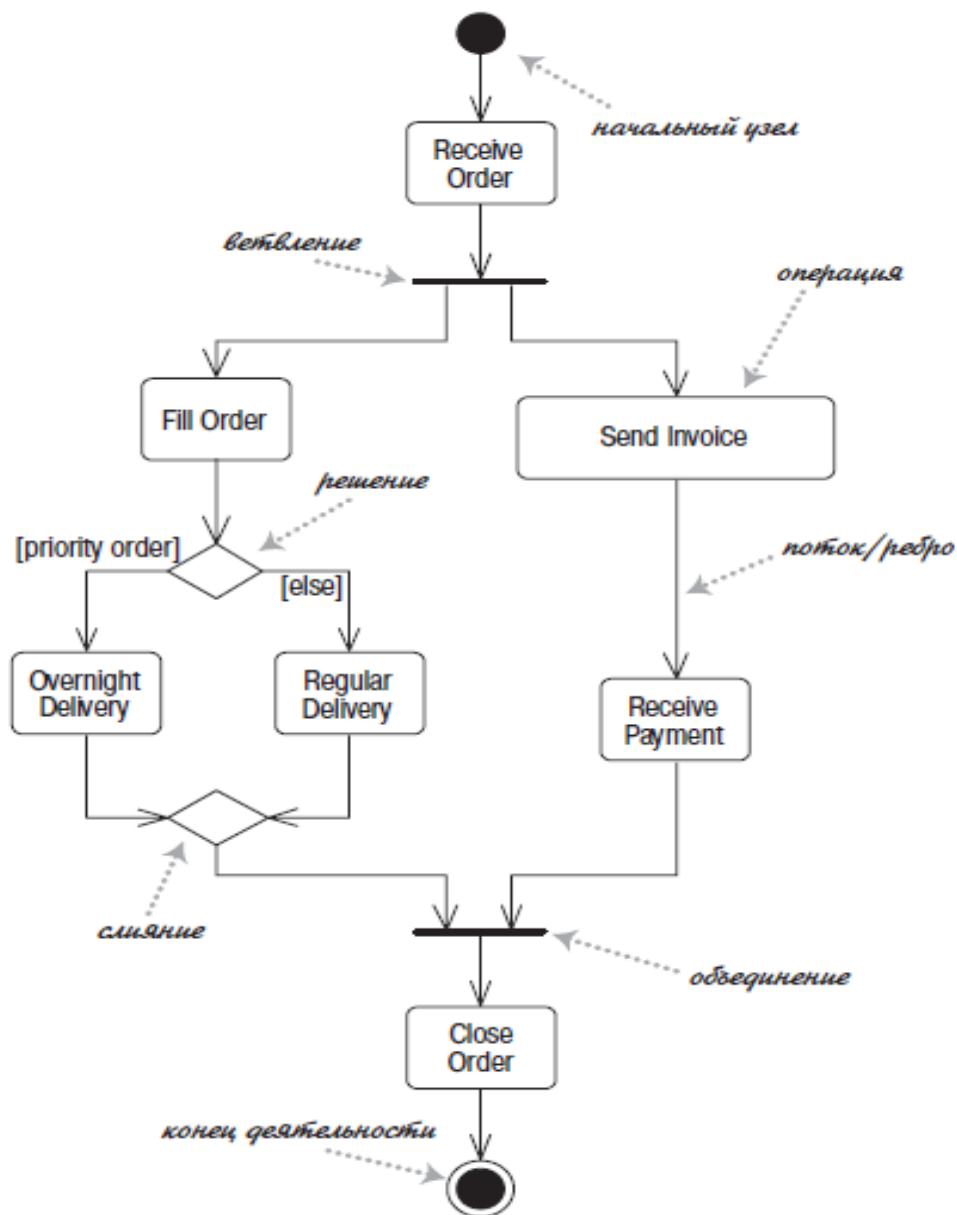


Диаграмма деятельности: основные понятия

Деятельность (*activity*) является спецификацией параметризованного поведения в форме координируемой последовательности действий.

Действие (*action*) представляет собой элементарную единицу спецификации поведения, которая не может быть далее декомпозирована в форме деятельности. Действие является непрерываемым извне, безусловным и завершаемым.

Переход (*transition*) представляет переход из одного состояния действия в следующее.



Граф деятельности

Семантически граф деятельности (activity graph) — это множество сущностей, которыми являются действия или деятельности, и отношения между этими сущностями, которые задают порядок их выполнения.

Синтаксически граф деятельности — это нагруженный ориентированный граф, в котором используются узлы четырех типов: **узлы действий**, **узлы деятельности**, **узлы управления** и **узлы объектов**, а дуги являются потоками управления или потоками данных.

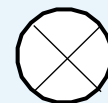


Узлы графа деятельности



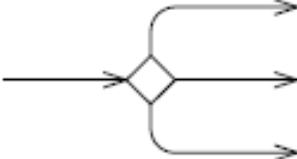
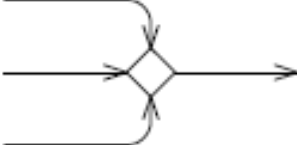
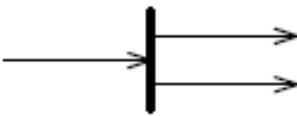

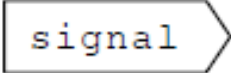
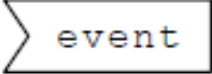
Имя
действия

Имя
деятельности
⌚

Имя
объекта



Концепция узлов управления в UML 1.x

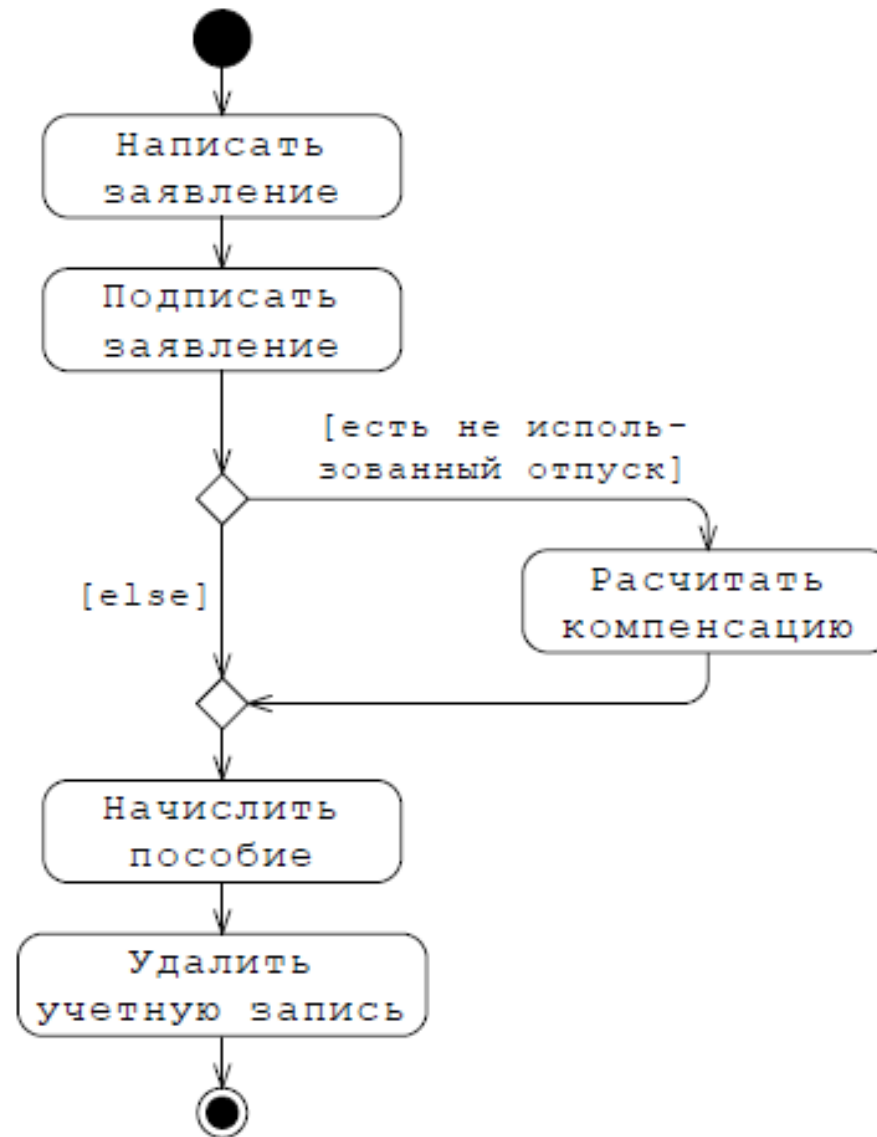
Название	Изображение	Что обозначает
Начальное состояние (initial node)		Начало деятельности
Заключительное состояние (final node)		Завершение деятельности
Разветвление управления (decision node)		Начало альтернативных ветвей деятельности
Объединение управления (merge node)		Конец альтернативных ветвей деятельности
Развилка управления (fork node)		Начало параллельных ветвей деятельности
Слияние управления (join node)		Конец параллельных ветвей деятельности
Посылка сигнала (send)		Действие посылки сигнала
Прием сигнала (accept)		Ожидание события прихода сигнала

В UML 1 описана семантика обычных блок-схем, в которых нет никаких событий, а есть последовательная передача управления следующей деятельности по завершении предыдущей деятельности.

Такие переходы называются **переходами по завершении**, или **не-триггерными**, поскольку управление по завершении работы в исходном состоянии немедленно передается дальше.



Пример диаграммы деятельности в uml 1



Операционная семантика диаграммы деятельности в UML 2.0

Определена не через машины состояний, а через сети Петри

Маркеры — это абстрактные конструкции, которые вводятся для удобства описания динамического процесса выполнения статически определенного графа деятельности. Это элемент модели, предназначенный для представления некоторого объекта, данных или управления и существующий на диаграмме деятельности в отдельном узле.

Маркер управления – маркер, не содержащий никакой дополнительной информации.

Маркер данных – маркер, содержащий ссылку на объект или структуру данных.



Семантика деятельности в UML 2.0

- Каждый маркер отличается от любого другого, даже если он содержит то же значение, что и другой
- Любой узел деятельности может начать свое выполнение, только если удовлетворены специфицированные условия для его входных маркеров, причем эти условия зависят от вида узла
- Когда узел начинает свое выполнение, маркеры принимаются из некоторых или всех его входных дуг, а специальный маркер размещается в этом узле
- Когда узел завершает выполнение, специальный маркер удаляется из этого узла, а другие маркеры предлагаются в некоторых или всех его выходных дугах
- Выполнение действия становится возможным, когда удовлетворены предварительные условия для его потоков управления и объектов



Семантика деятельности в UML 2.0

- Выполнение действия поглощает входные маркеры управления и маркеры объектов и удаляет их из источников дуг управления и из аргументов
- Если на одной дуге являются доступными несколько маркеров управления, то они все поглощаются
- Действие продолжает выполнение до тех пор, пока оно не будет завершено
- После завершения действия оно предлагает маркеры объектов во все его результаты, а маркеры управления во все выходящие из него дуги управления, и на этом формально оно заканчивается

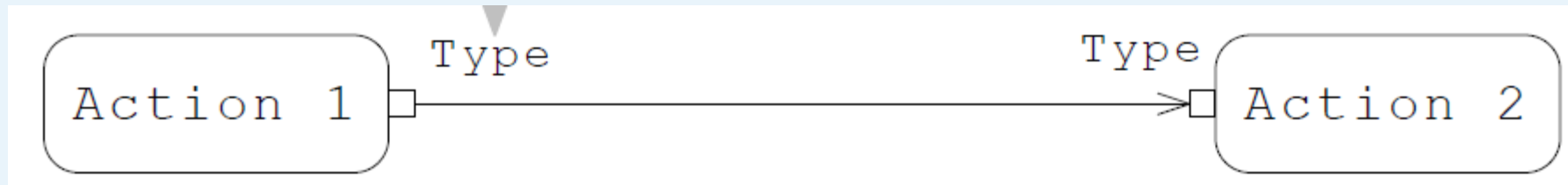


Нотация контактов

Контакт (pin) — это указание аргумента или результата действия.
Контакт может иметь имя и тип.

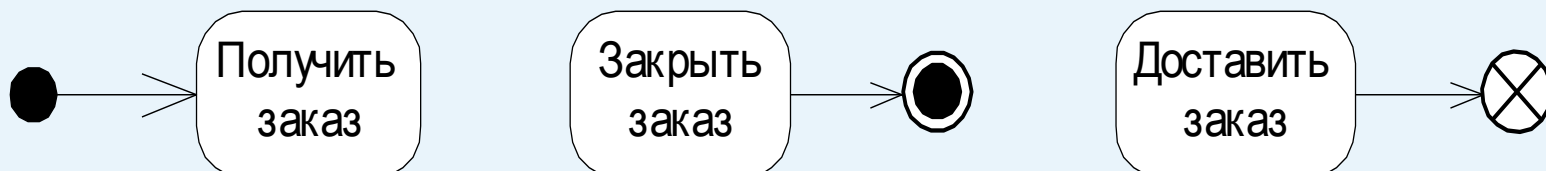
Через контакты «путешествуют» маркеры данных по графу деятельности в процессе выполнения. Через входные контакты действие принимает свои аргументы, а через выходные контакты выдает свои результаты.

Графически контакты изображаются в виде маленьких квадратиков, прикрепленных к сторонам фигуры, изображающей действие.



Узлы управления

- ✓ **Начальный узел (initial node)** является узлом управления, в котором начинается поток при вызове деятельности
- ✓ Узел **финала деятельности (activity final node)** является узлом управления, который прекращает или останавливает все потоки в деятельности
- ✓ **Узел финала потока (flow final node)** является финальным узлом, который завершает отдельный поток управления или поток объектов, не завершая содержащей его деятельности

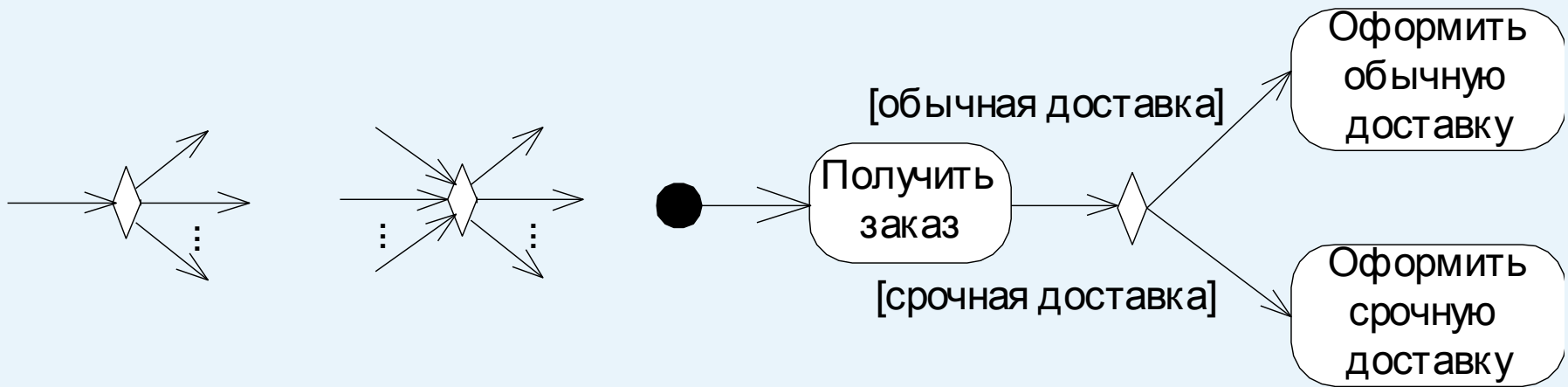


Узел решения

- ✓ Является узлом управления, который выбирает между выходящими потоками
- ✓ Если для узла решения при оценивании оказываются справедливыми более одного сторожевого условия, то семантика такого поведения в языке UML 2.x не определена, поскольку среди выходящих дуг возникает состязание за прием маркера
- ✓ Чтобы гарантировать выполнение только одного сторожевого условия, иногда удобно использовать процедуру проверки до первого истинного условия

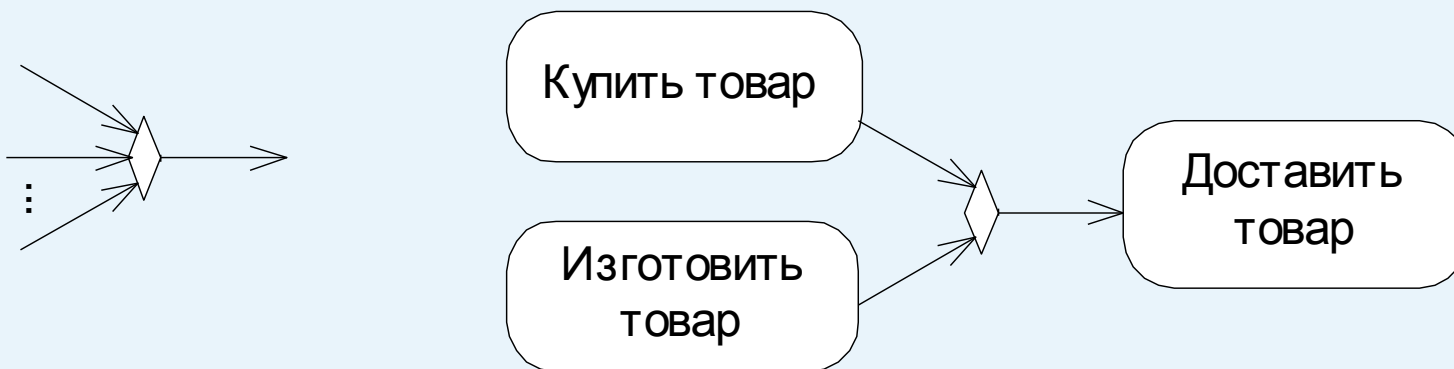


Варианты изображения узла решения

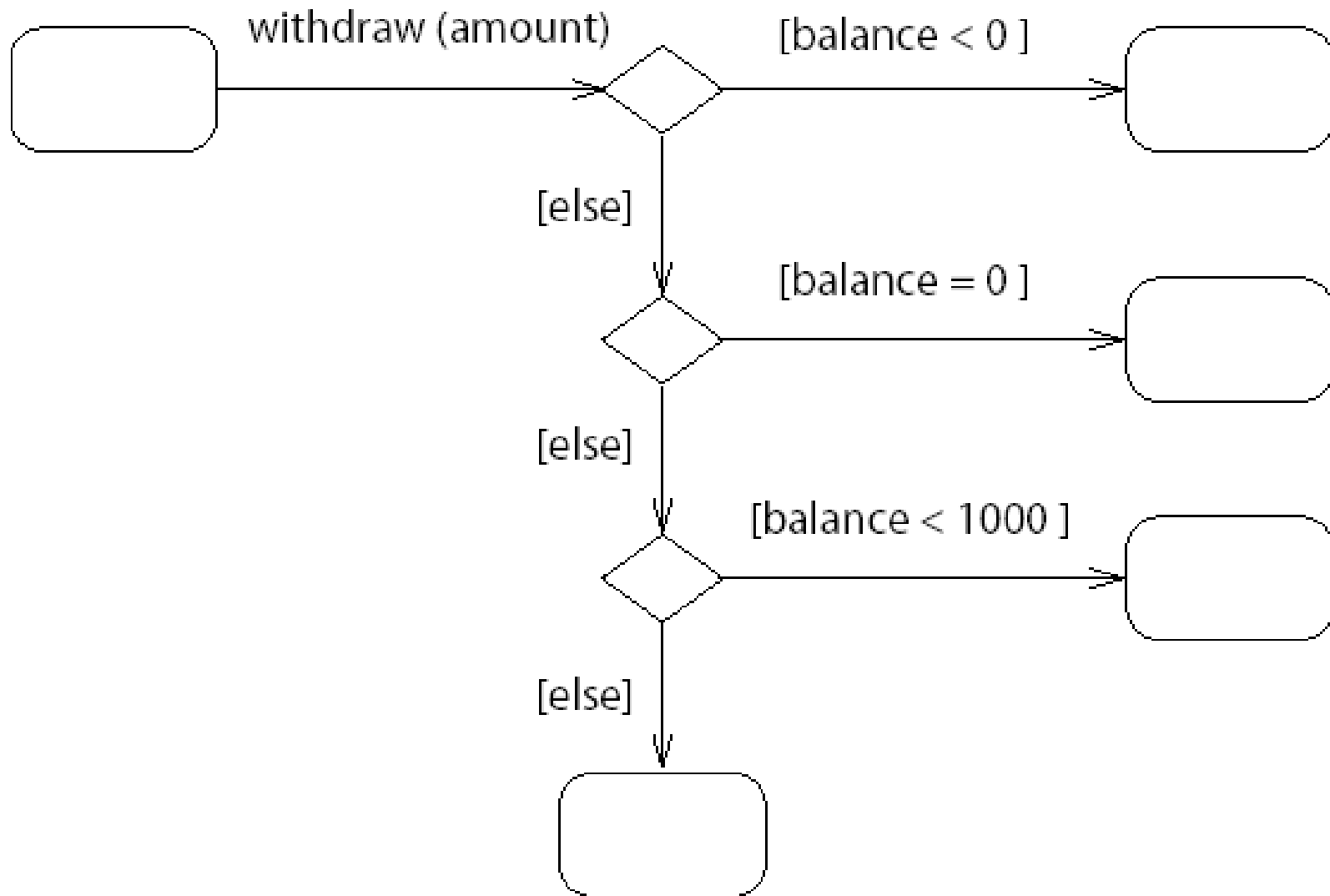


Узел слияния

- ✓ является узлом управления, который соединяет вместе несколько альтернативных потоков

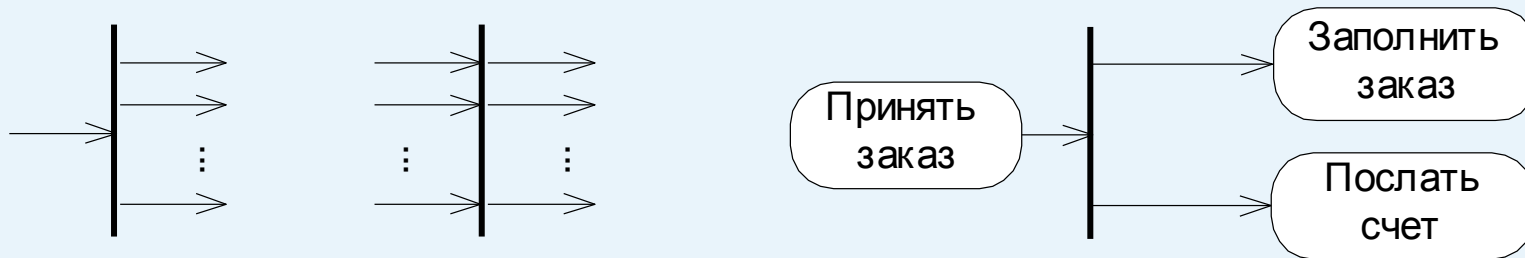


Пример последовательного ветвления



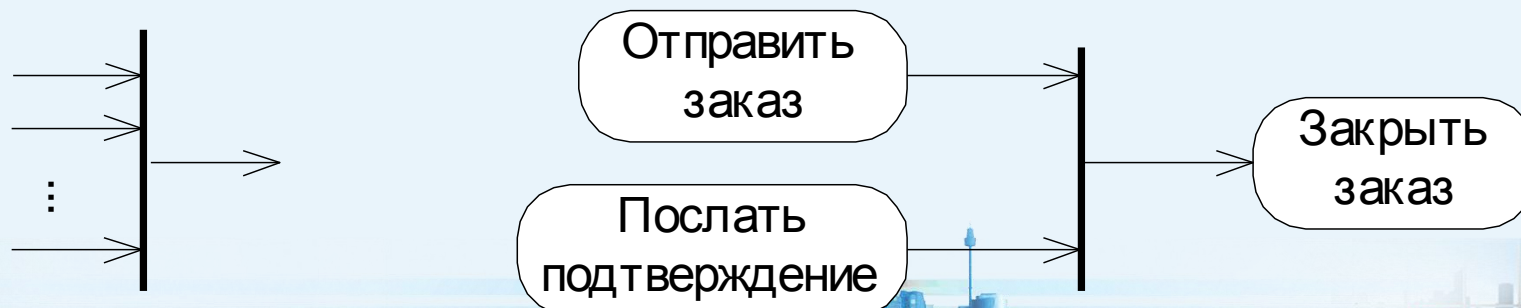
Узел разделения

- ✓ Является узлом управления, который расщепляет поток на несколько параллельных потоков
- ✓ Дуги, выходящие из узла разделения, дополнительно могут иметь сторожевые условия, при невыполнении которых могут возникать паузы с передачей маркеров по этим дугам

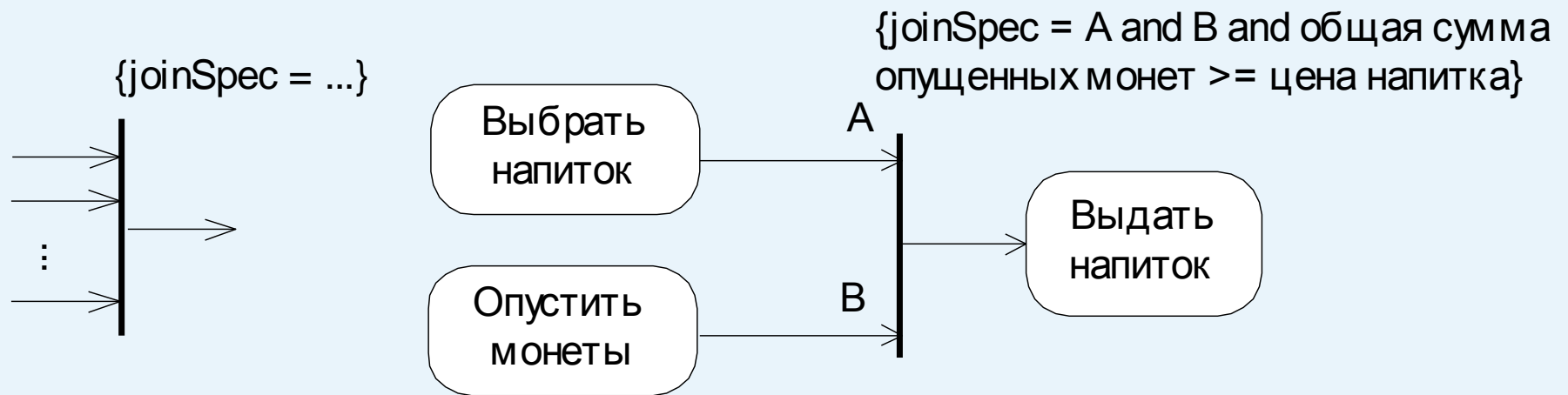


Узел соединения

- ✓ Является узлом управления, который синхронизирует несколько потоков
- ✓ Узлы соединения могут иметь дополнительную логическую спецификацию условий, при выполнении которых они должны генерировать маркер на выходе
- ✓ Если для узла соединения существуют маркеры во всех его входящих дугах, то выходящей дуге предлагаются маркеры согласно следующим правилам:
 - Если все маркеры, предлагаемые на входящих дугах, являются маркерами управления, то выходящей дуге предлагается один маркер управления
 - Если часть маркеров, предлагаемых на входящих дугах, являются маркерами управления, а другие являются маркерами данных, то выходящей дуге предлагаются только маркеры данных
 - Они предлагаются выходящей дуге в том же порядке, в каком предлагаются на входе этого узла соединения



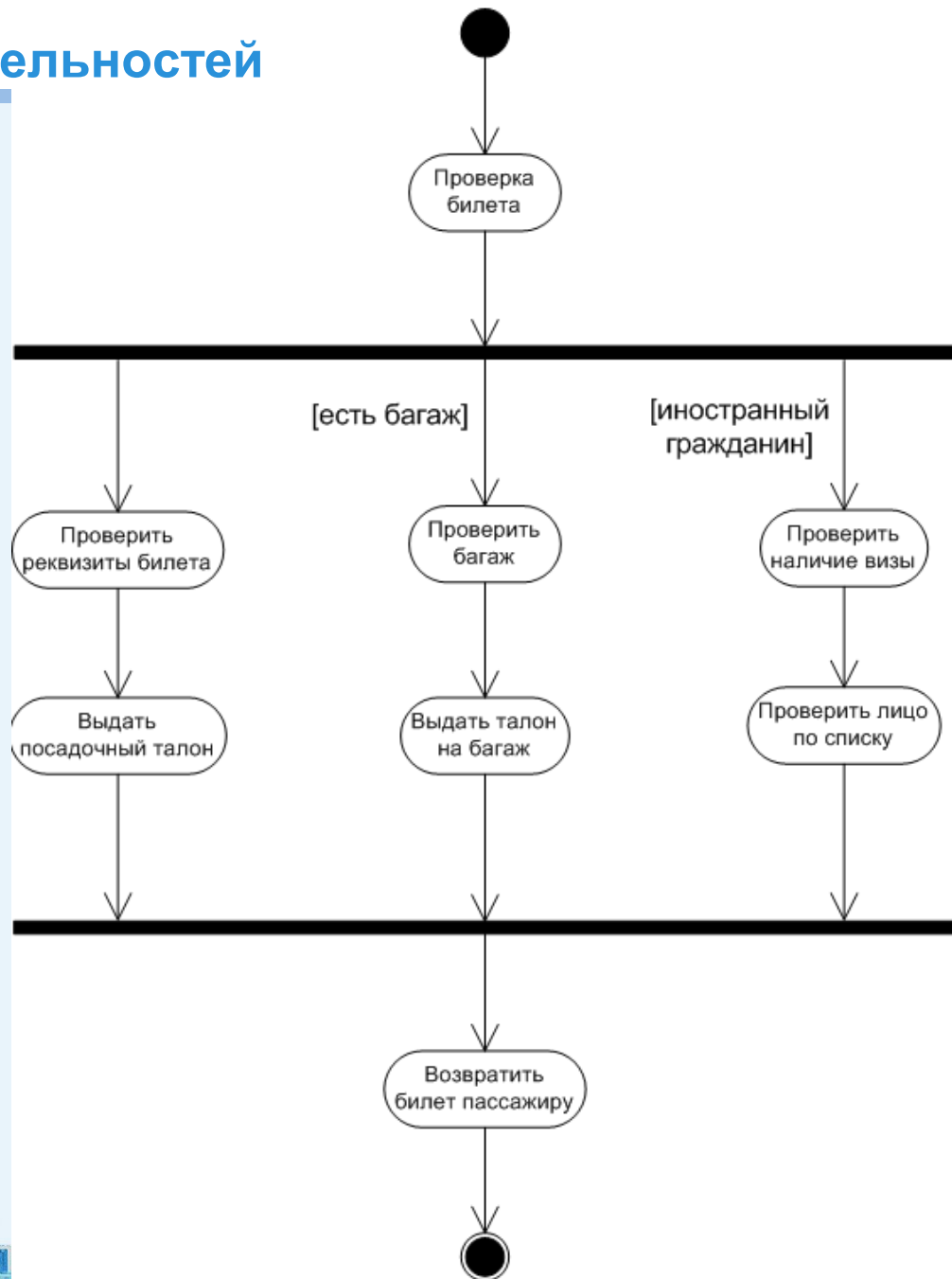
Пример изображения узла соединения с дополнительной спецификацией



Пример

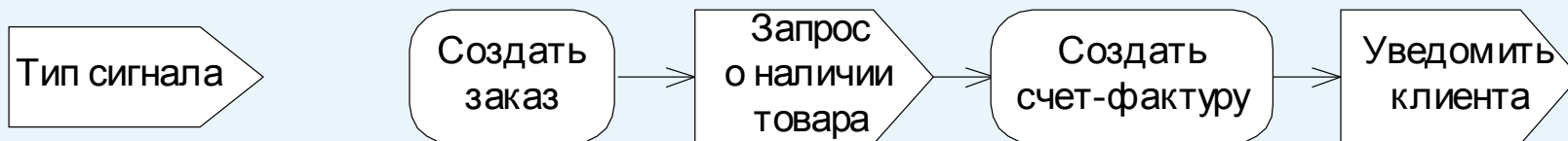
условно-параллельных деятельности

Дуги, выходящие из узла разделения, дополнительно могут иметь сторожевые условия, при невыполнении которых могут возникать паузы с передачей управления по этим дугам

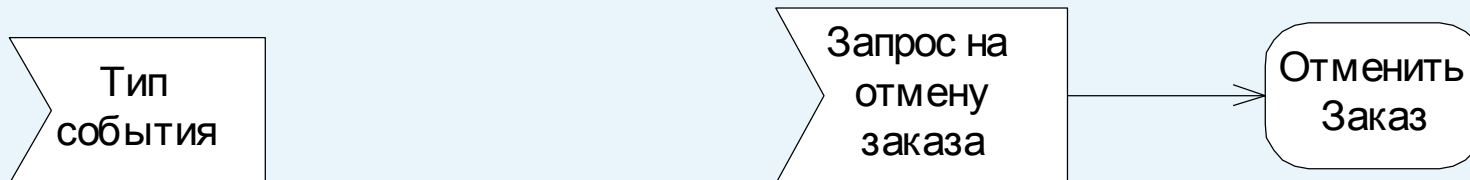


Специальные действия

- ✓ **Действие передачи сигнала (*send signal action*)** является действием, которое на основе своих входов создает экземпляр сигнала и передает его объекту цели



- ✓ **Действие приема события (*accept event action*)** является действием, которое ожидает наступление некоторого события

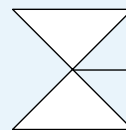


Ожидание временного события

- ✓ Специальный случай действия приема события
- ✓ Если наступившее событие является временным событием, то объект должен зафиксировать значение момента времени, когда наступило соответствующее событие



Наступил
конец месяца



Подготовить
отчет о
продажах



Узел объекта

- Узел объекта для маркеров объектов, находящихся в специальном состоянии, дополнительно содержит спецификацию этого состояния, которая записывается в прямых скобках ниже имени типа
- Узел объекта для маркеров, содержащих множества объектов различных типов, содержит имена всех этих объектов
- Узлы объектов с сигналом в качестве типа изображаются с помощью специального символа, внутри которого записывается имя типа сигнала

Имя объекта
[состояние]

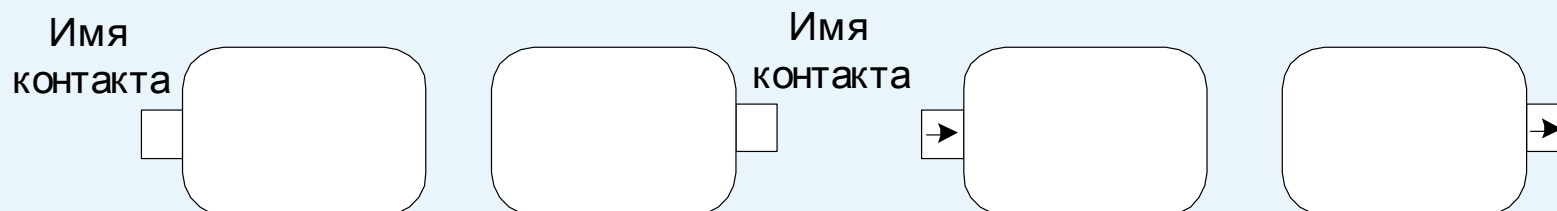
Множество
имен

Имя
сигнала

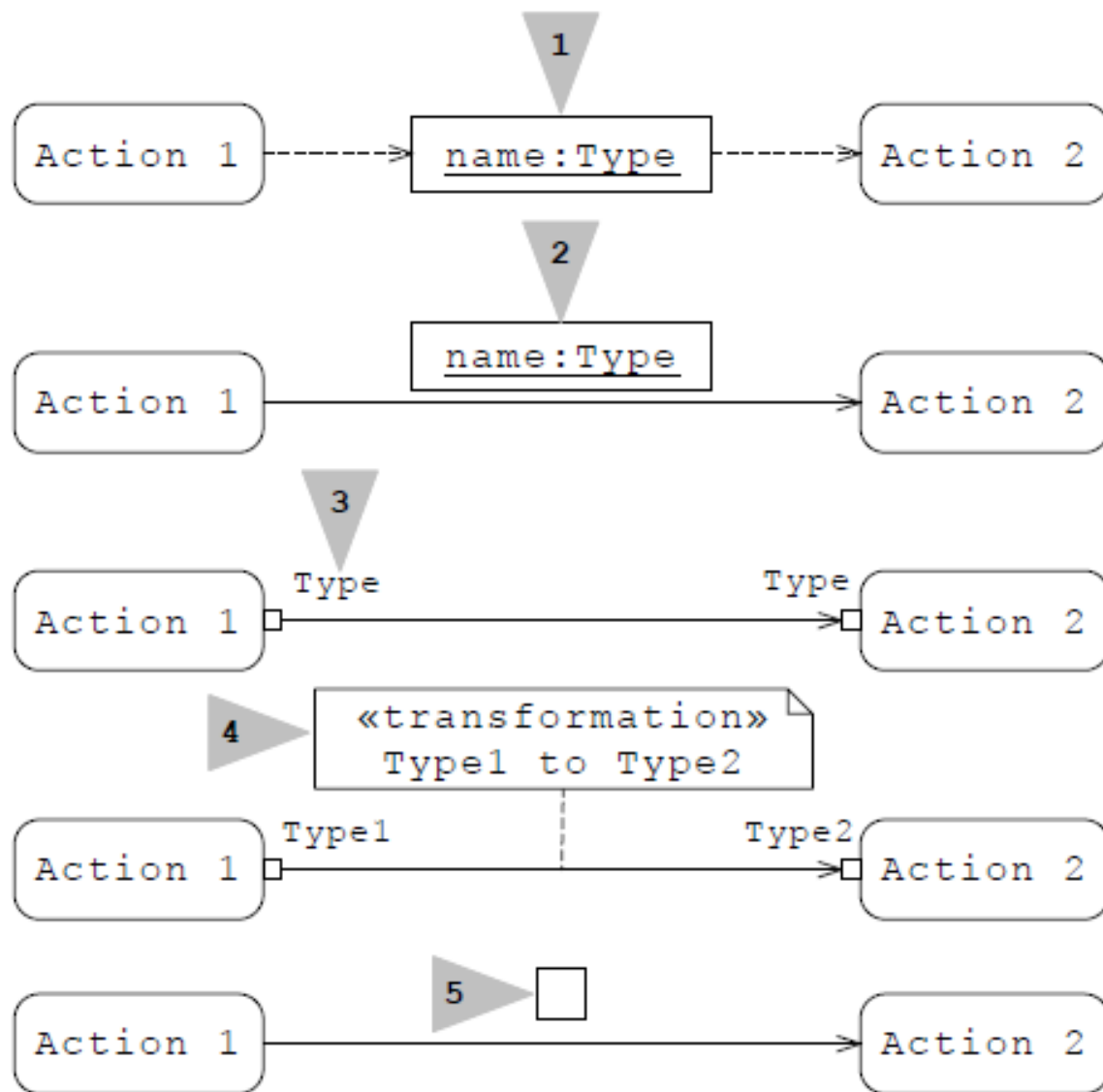


Входные и выходные контакты действий

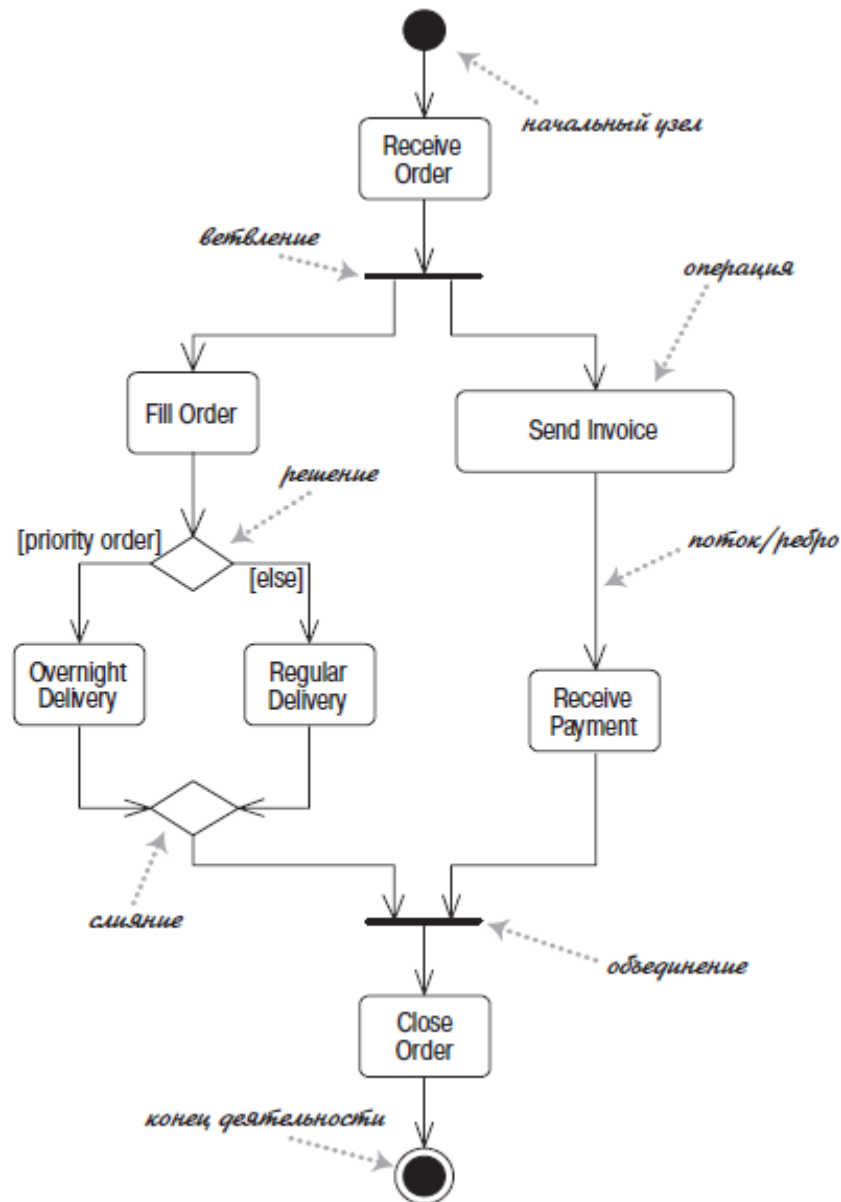
- **Входной контакт (*input pin*)** является узлом объекта, который принимает значения от других действий в форме потока объектов
- **Выходной контакт (*output pin*)** является узлом объекта, который поставляет значения другим действиям в форме потока объектов.



Варианты отображения передачи объектов и потоков данных

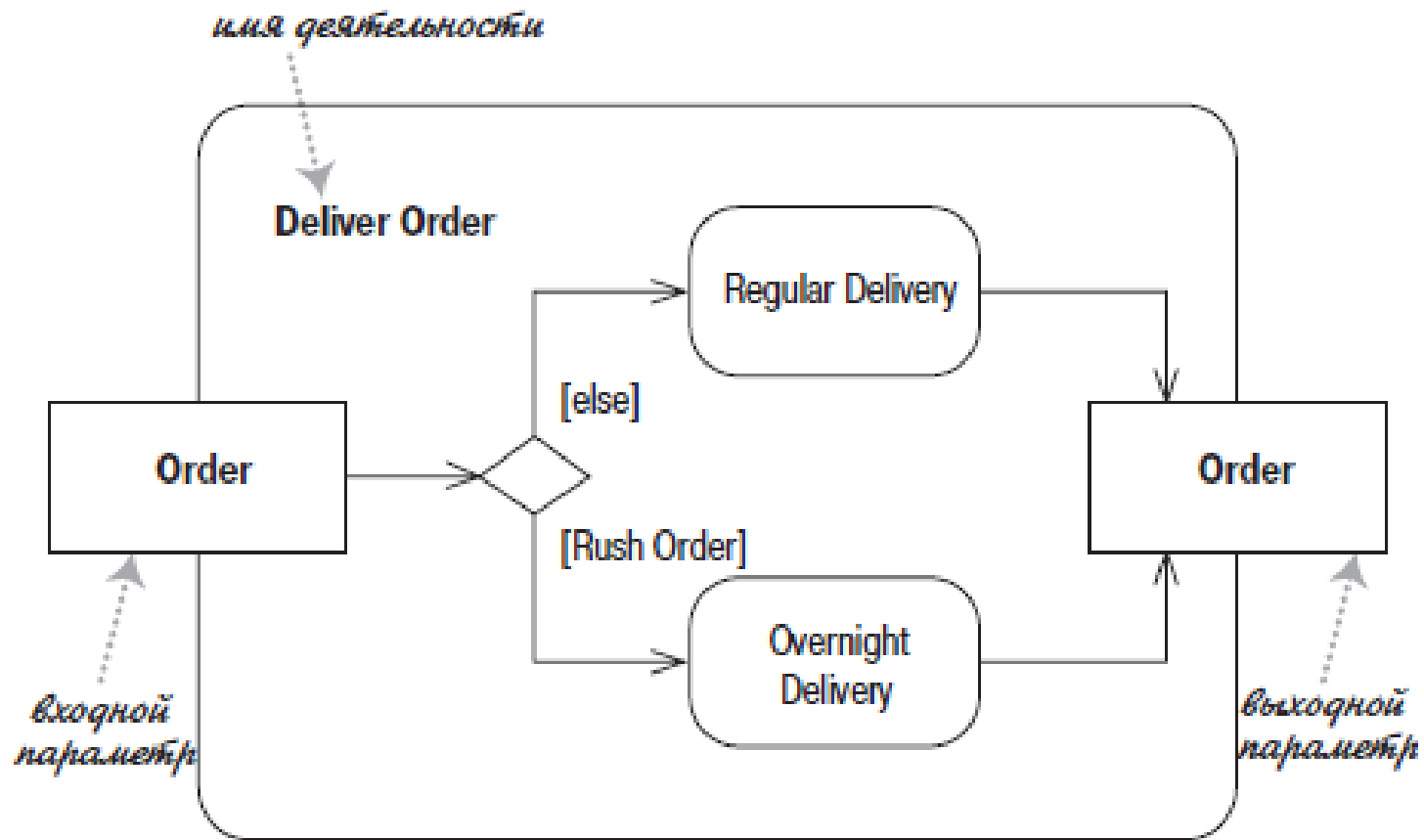


Декомпозиция деятельности



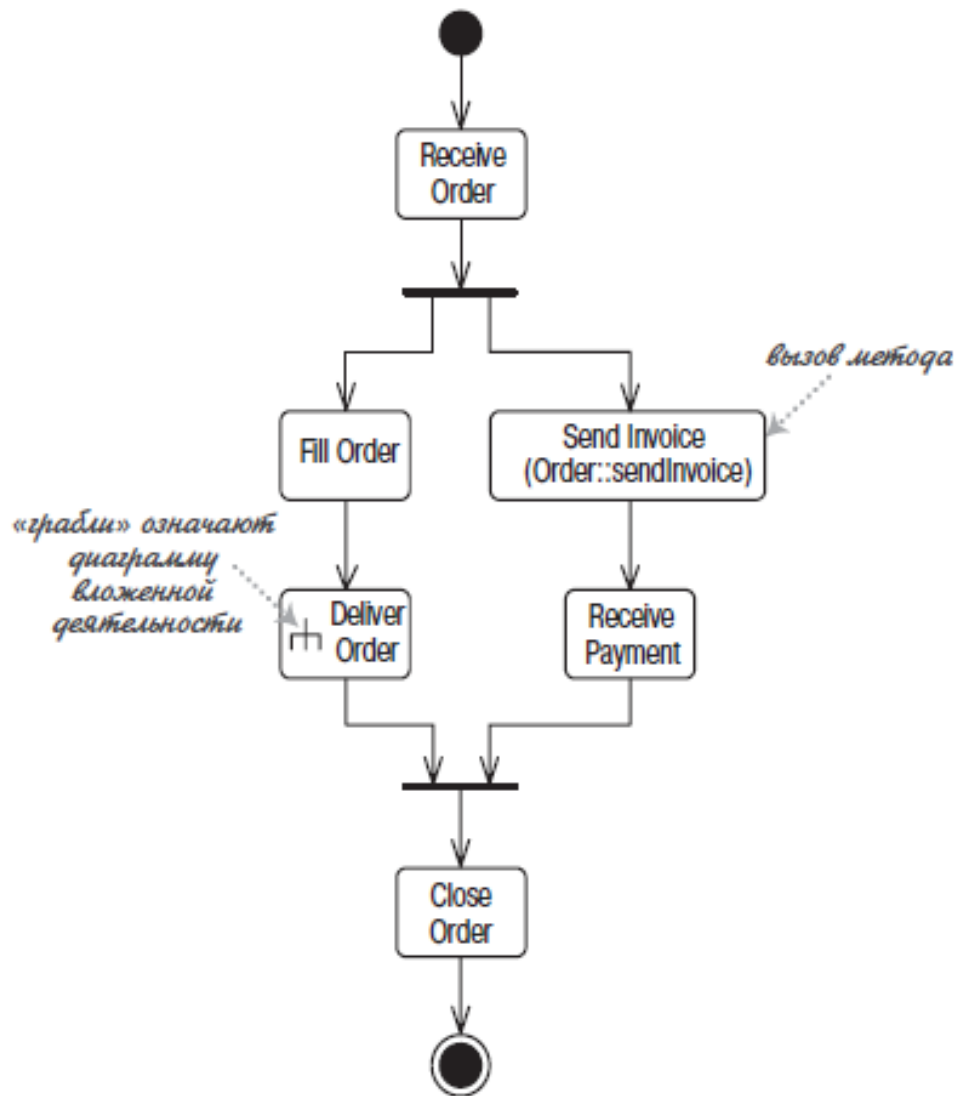
Простая диаграмма
деятельности

Декомпозиция деятельности



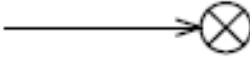
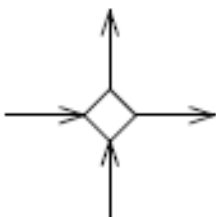


Дополнительная диаграмма деятельности

Декомпозиция деятельности



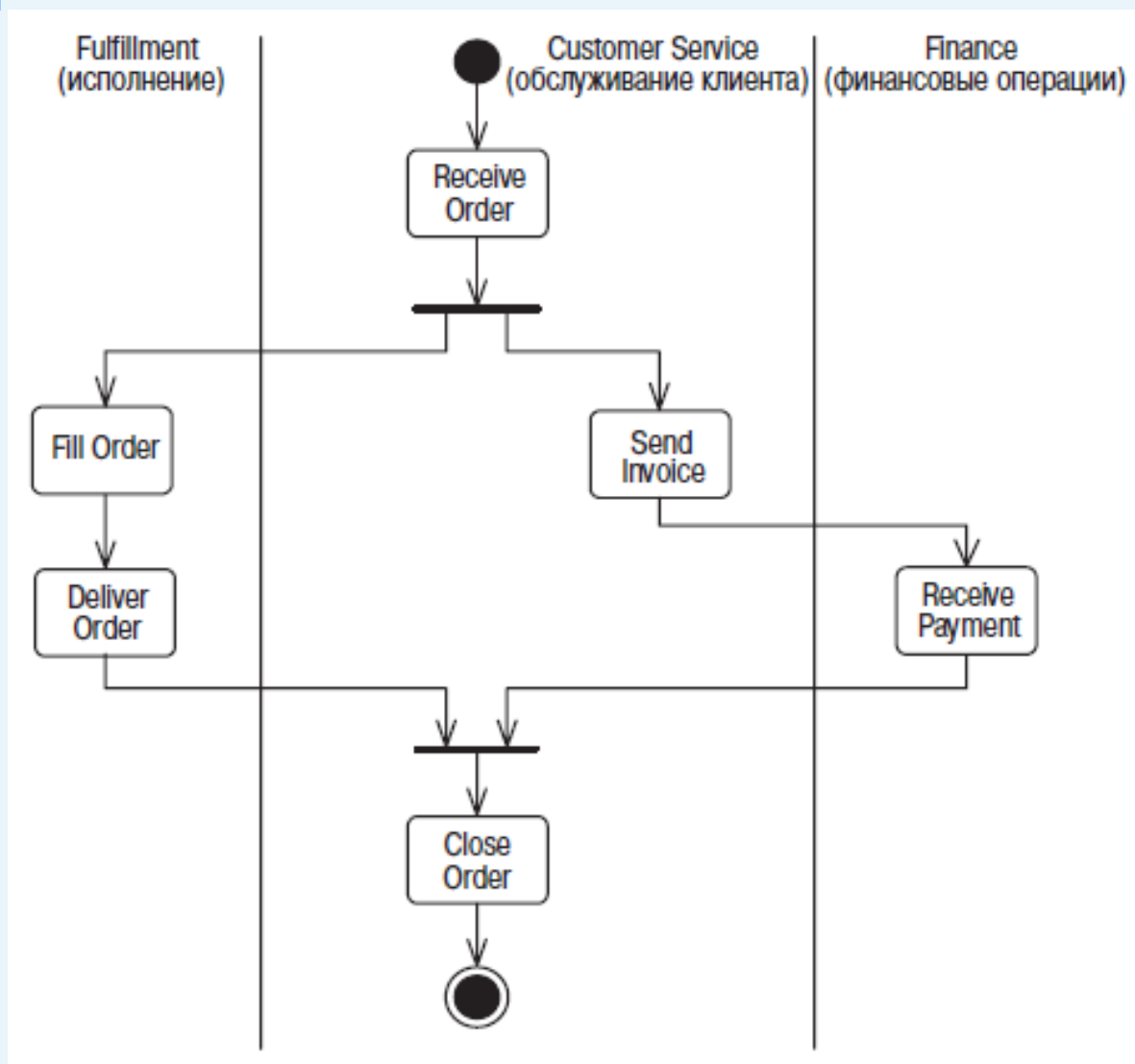
Простая диаграмма деятельности, модифицированная для вызова вложенной диаграммы деятельности

Дополнительные узлы управления в UML 2

Название	Изображение	Что обозначает
Заключительное состояние потока (final flow node)		Завершение одного потока в деятельности. Если в деятельности есть другие параллельные потоки, они продолжают.
Комбинированное соединение/разветвление управления		Последовательность из узла соединения и узла разветвления.
Комбинированное слияние/развилка управления		Последовательность из узла слияния и узла развилки.
Прием сигнала от таймера		Узел, являющийся источником маркера управления по истечении заданного интервала времени



Разделы



Можно разбить диаграмму деятельности на разделы (partitions), чтобы показать, кто что делает, то есть какие операции выполняет тот или иной класс или подразделение предприятия.

В UML 2 сетка может быть двумерной.

Сигналы

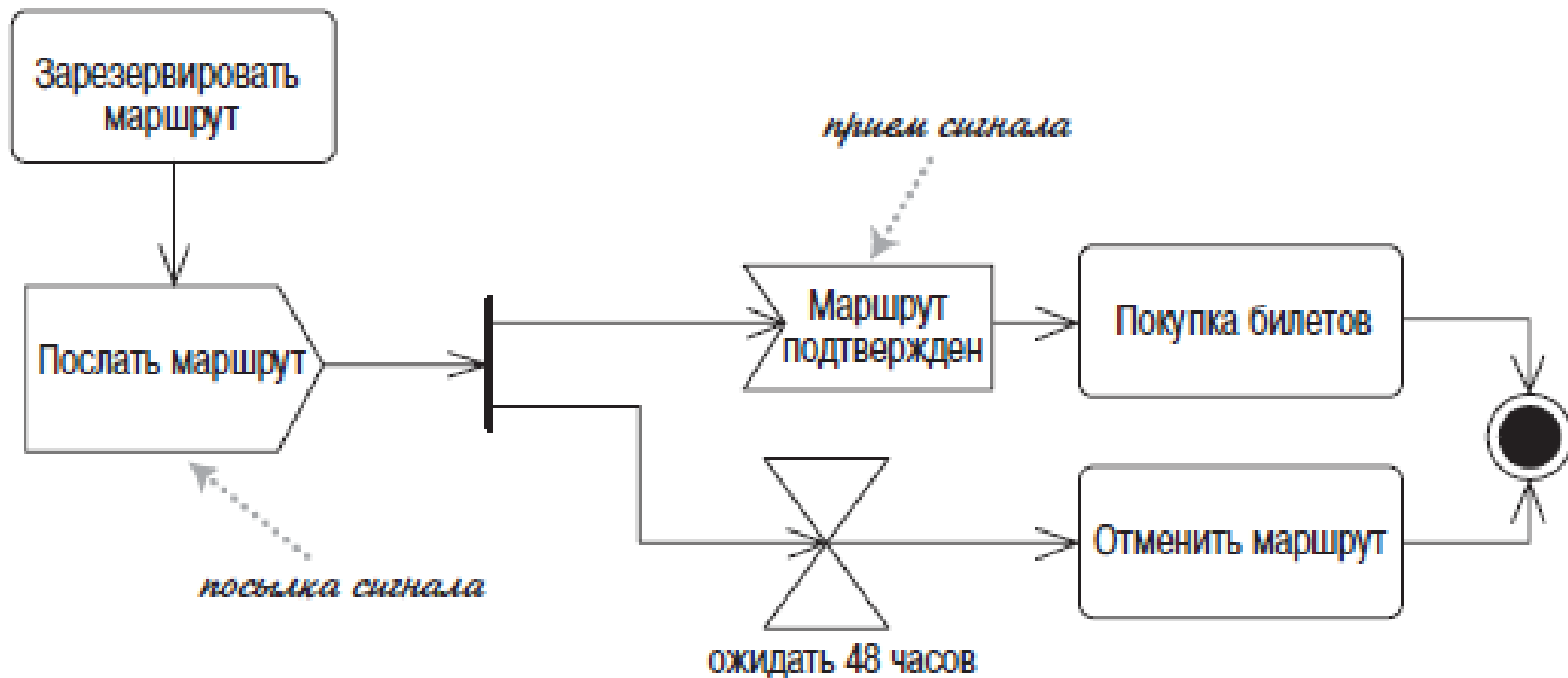
Временной сигнал (time signal) приходит по прошествии времени. Такие сигналы могут означать конец месяца в отчетном периоде или приходиться каждую секунду в контроллере реального времени.

Диаграммы деятельности имеют четко определенную стартовую точку, соответствующую вызову программы или процедуры. Кроме того, операции могут отвечать на сигналы.



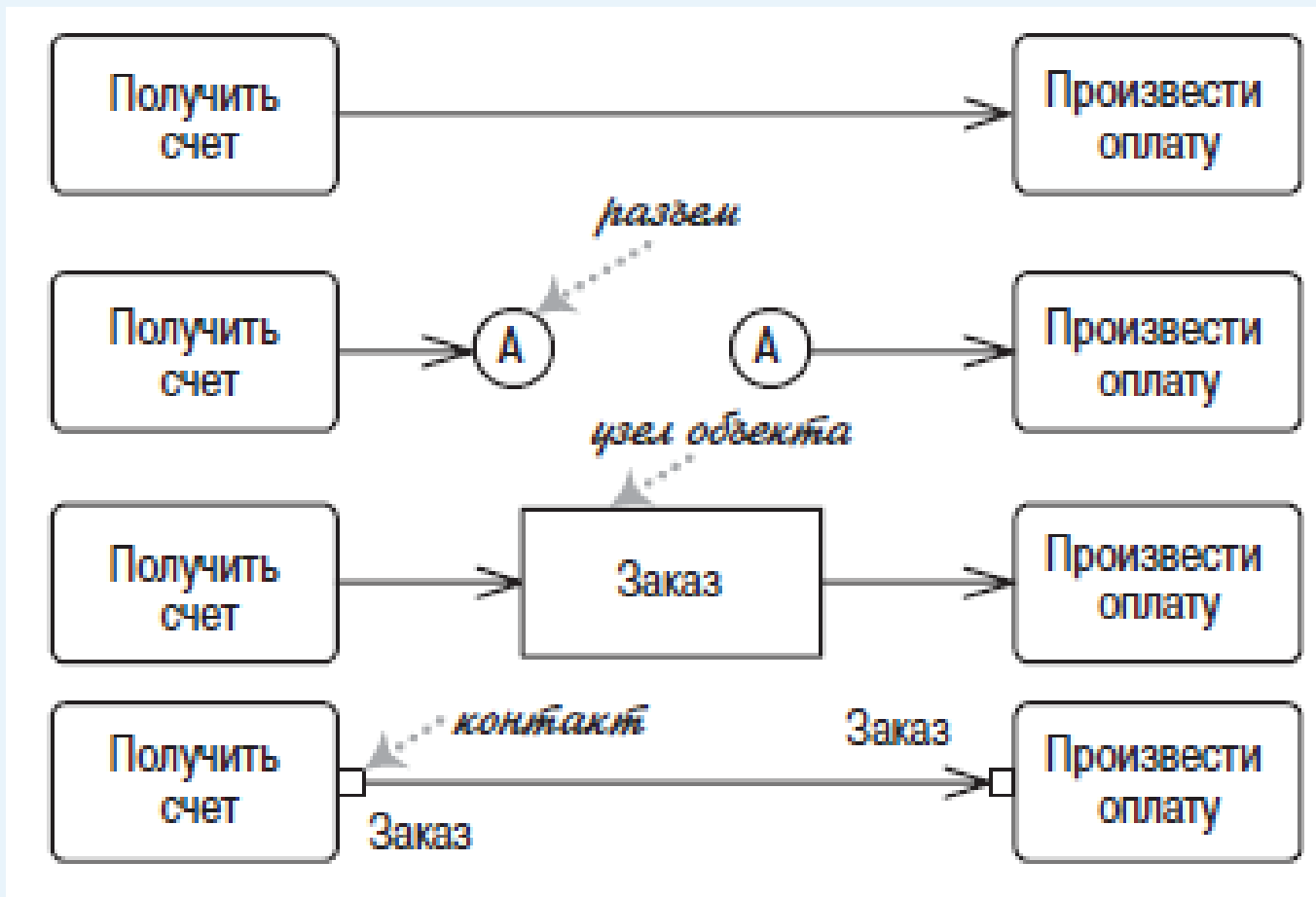
Пример отправки сигнала

Деятельности могут как принимать сигналы, так и посылать их. Это полезно, когда мы посылаем сообщение, а затем должны ожидать ответа, перед тем, как продолжить.

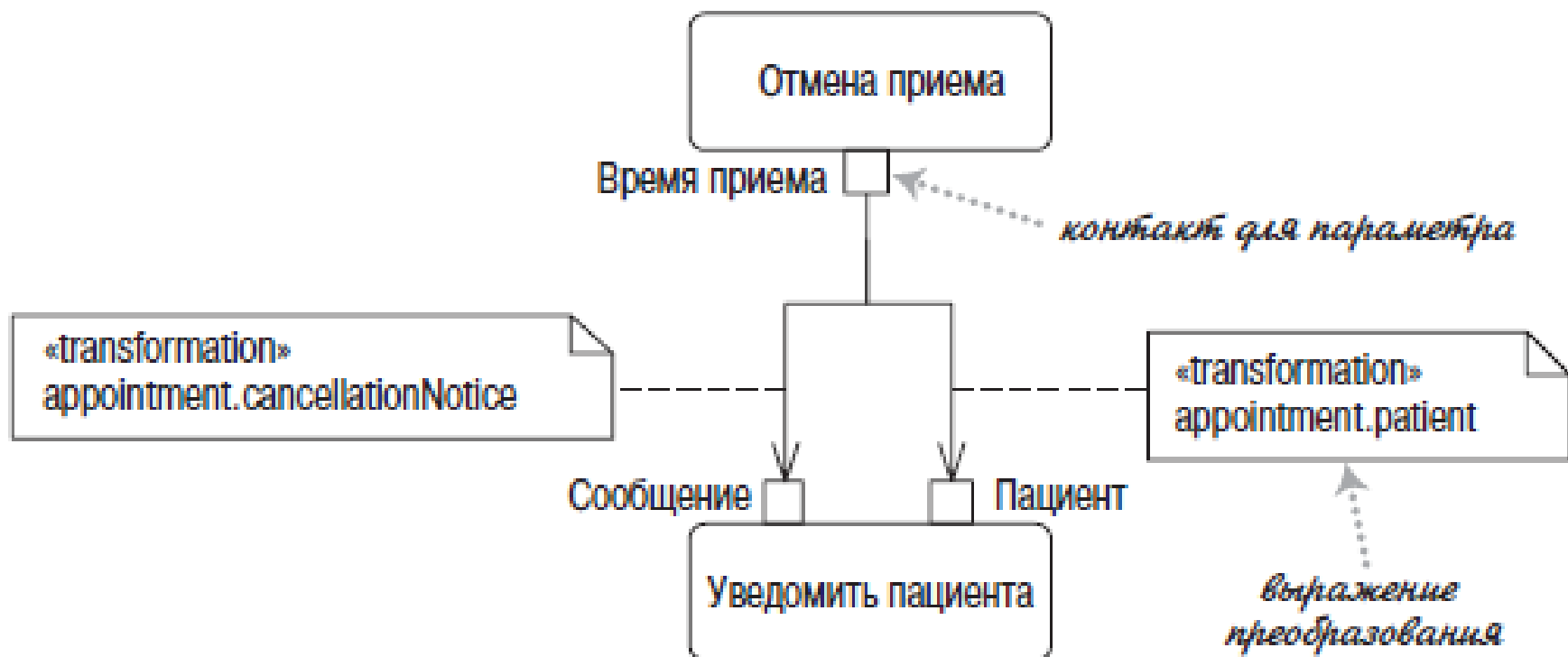


На диаграмме в гонке участвует два потока: первый, достигший финального состояния, выигрывает и прерывает выполнение другого потока. Хотя блоки приема сигналов только ожидают внешнего события, мы можем также показать входящий в него поток. Это означает, что мы не начинаем прослушивание до тех пор, пока поток не инициирует прием.

Потоки и ребра в UML 2



Контакты и преобразования в UML 2



Когда применяют диаграммы деятельности?

Самое большое достоинство диаграмм деятельности заключается в том, что они поддерживают и стимулируют применение параллельных процессов. Именно благодаря этому они представляют собой мощное средство моделирования потоков работ.



ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ

это способ описания поведения системы «на примерах»

Диаграмма последовательности служит для представления взаимодействия элементов модели в форме последовательности сообщений и соответствующих событий на линиях жизни объектов



Пример диаграммы последовательности

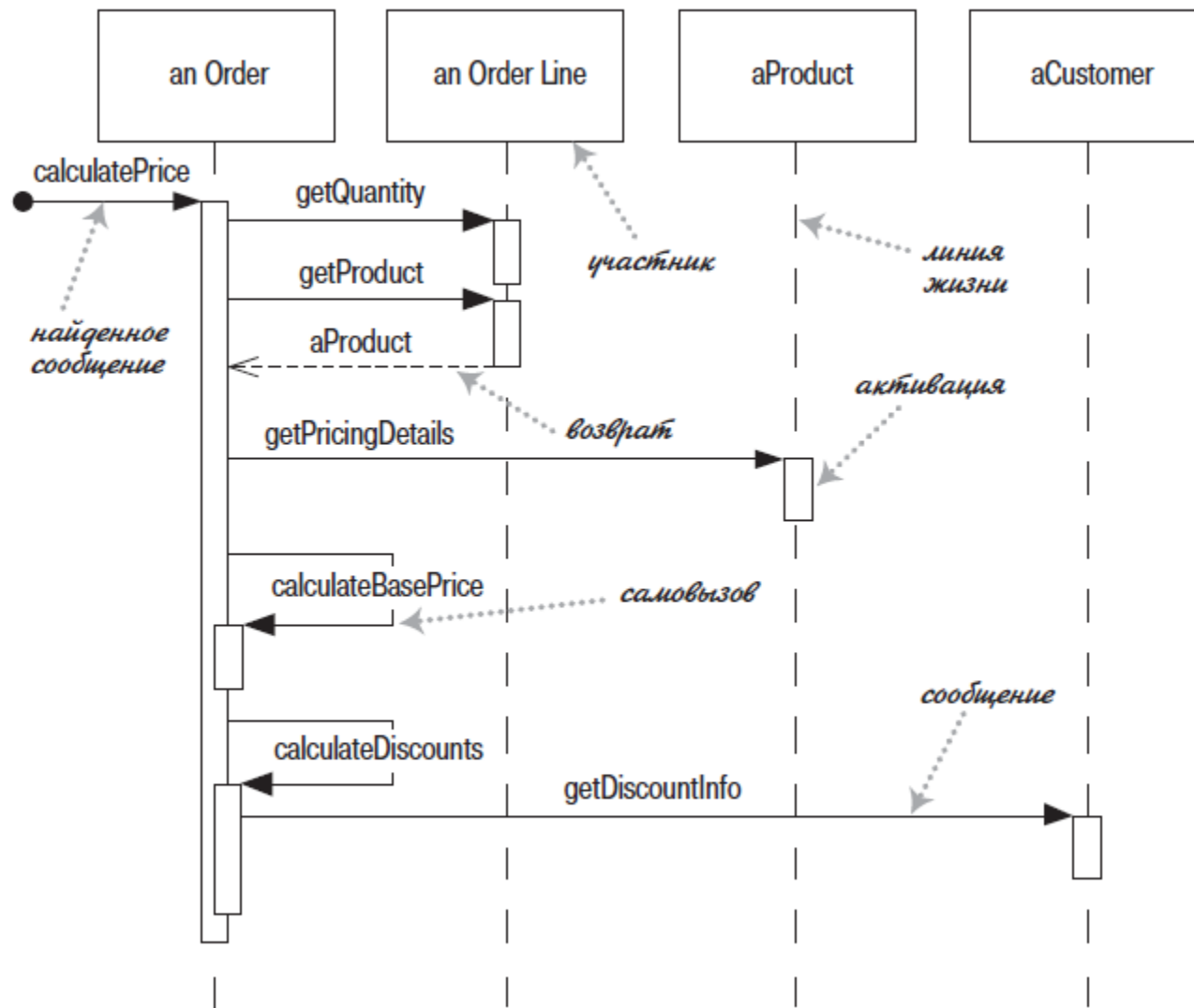


Диаграмма последовательности: основные понятия

Взаимодействие (interaction) — называется поведение, выражающееся в обмене сообщениями между множеством объектов в некотором контексте, в результате чего достигается определенная цель.

Сообщение (Message) - это спецификация обмена данными между объектами, при котором передается некая информация в расчете на то, что в ответ последует определенное действие.

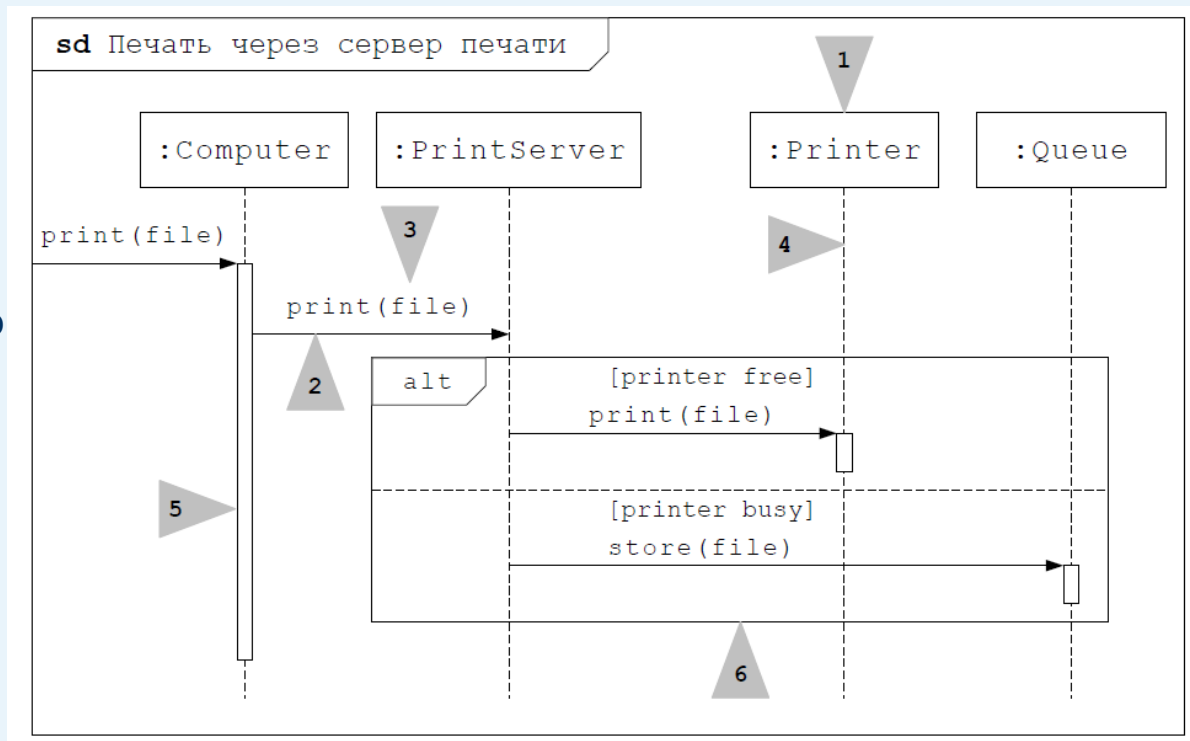
Масштаб для оси времени на диаграмме последовательности не указывается, поскольку эта диаграмма предназначена для моделирования только лишь временного порядка следования сообщений типа "раньше-позже"

Контекст . Контекстом взаимодействия объектов может быть система, подсистема, операция класса.



Диаграмма последовательности: графическая нотация

На диаграмме последовательности применяют один основной тип сущностей — экземпляры классов, компонентов, действующих лиц (1) и один тип отношений — связи (2), по которым происходит обмен сообщениями (3). Предусмотрено несколько способов посылки сообщений, которые в графической нотации различаются видом стрелки, соответствующей отношению.



Объекты — прямоугольник с именем экземпляра классификатора.

Пунктирная линия, выходящая из него, называется линией жизни (lifeline) (4).

Фигуры в виде узких полосок - это графический комментарий, показывающий отрезки времени, в течении которых объект владеет потоком управления (execution occurrence) (5) или другими словами имеет место активация (activation) объекта.

Вложенные фреймы (combined fragment) (6) позволяют на диаграмме последовательности отражать и алгоритмические аспекты протокола взаимодействия.

Элементы графической нотации

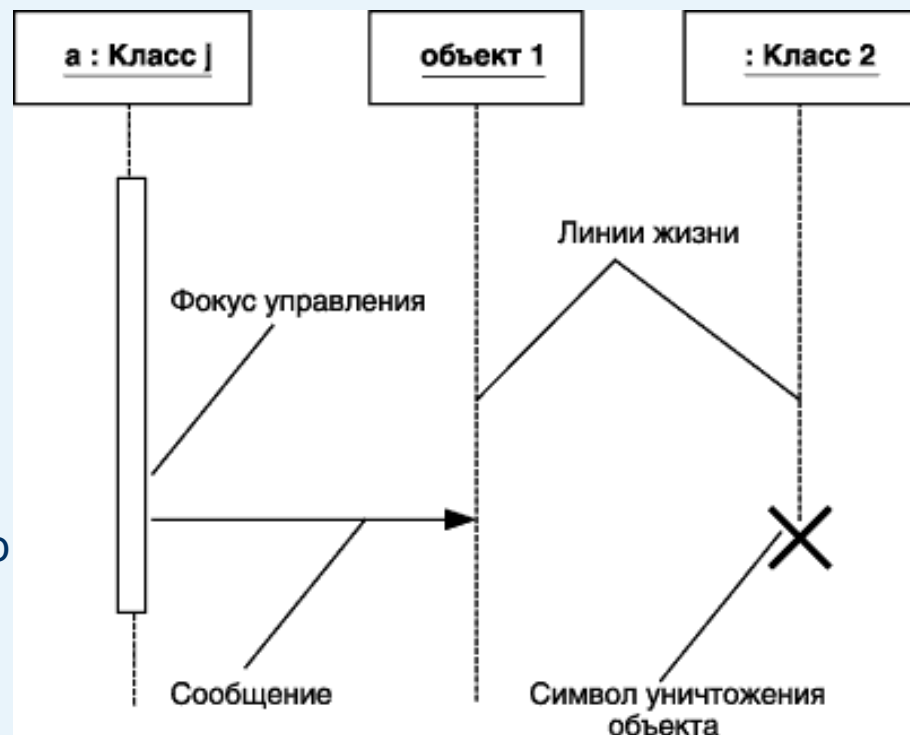
- ✓ Прямоугольник с непрерывными сторонами, который называется фреймом (frame) диаграммы
- ✓ В верхнем левом углу прямоугольника фрейма изображается небольшой пятиугольник, в который помещается ключевое слово `sd`, за которым следует имя взаимодействия и его параметр
- ✓ Порядок наступления событий вдоль линий жизни имеет значение для обозначения последовательности, в которой эти наступления события происходят
- ✓ Однако, абсолютные расстояния между наступлениями событий на линиях жизни не имеют семантики
- ✓ Другими словами, время на диаграмме последовательности имеет шкалу порядка, а не шкалу отношений



Линия жизни на диаграмме последовательности

представляет одного индивидуального участника взаимодействия или отдельную взаимодействующую сущность

Каждый объект графически изображается в форме прямоугольника и располагается в верхней части своей линии жизни. Внутри прямоугольника записываются собственное имя объекта со строчной буквы и имя класса, разделенные двоеточием. При этом вся запись подчеркивается, что является признаком объекта, который представляет собой экземпляр класса. Если на диаграмме последовательности отсутствует собственное имя объекта, то при этом должно быть указано имя класса. Такой объект считается **анонимным**. Может отсутствовать и имя класса, но при этом должно быть указано собственное имя объекта. Такой объект считается **сиротой**.



Фокус управления

В процессе функционирования объектно-ориентированных систем одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия или в состоянии пассивного ожидания сообщений от других объектов. Чтобы явно выделить подобную активность объектов, в языке UML применяется специальное понятие, получившее название **фокуса управления (focus of control)**.

Фокус управления изображается в форме вытянутого узкого прямоугольника, верхняя сторона которого обозначает начало получения фокуса управления объекта (начало активности), а ее нижняя сторона – окончание фокуса управления (окончание активности). Этот прямоугольник располагается ниже обозначения соответствующего объекта и может заменять его линию жизни, если на всем ее протяжении он является активным.



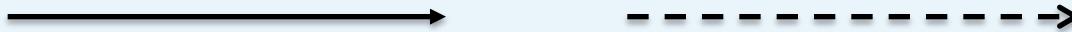
Сообщение

элемент модели, предназначенный для представления отдельной коммуникации между линиями жизни некоторого взаимодействия

Сообщение (message) представляет собой законченный фрагмент информации, который отправляется одним объектом другому. При этом прием сообщения инициирует выполнение определенных действий, направленных на решение отдельной задачи тем объектом, которому это сообщение отправлено.

Таким образом, сообщения не только передают некоторую информацию, но и требуют или предполагают от принимающего объекта выполнения ожидаемых действий.

Сообщения могут инициировать выполнение операций объектом соответствующего класса, а параметры этих операций передаются вместе с сообщением. На диаграмме последовательности все сообщения упорядочены по времени своего возникновения в моделируемой системе. В таком контексте каждое сообщение имеет направление от объекта, который инициирует и отправляет сообщение, к объекту, который его получает



Стереотипы сообщений

Действие, являющееся результатом получения сообщения, - это исполняемое предложение, которое образует абстракцию вычислительной процедуры. Действие может привести к изменению состояния.

call (вызвать) - вызывает операцию, применяемую к объекту. Объект может послать сообщение самому себе, что приведет к локальному вызову операции.

return (возвратить) - возвращает значение вызывающему объекту.

send (послать) - посылает объекту сигнал.

create (создать) - создает новый объект.

destroy (уничтожить) - удаляет объект. Объект может уничтожить самого себя.



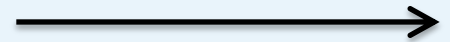
Сорт сообщений

synchCall – *синхронное* сообщение, которое соответствует синхронному вызову операции

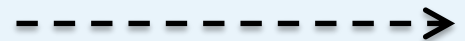
Синхронные сообщения обычно представляют вызовы методов и изображаются сплошной линией с закрашенной стрелкой



asynchCall – *асинхронное* сообщение, которое соответствует асинхронному вызову операции, изображается сплошной линией с открытой стрелкой в форме буквы "V".



ответное (reply) от вызова метода, изображается пунктирной линией с открытой стрелкой в форме буквы "V"



создания объекта (object creation) также изображается пунктирной линией с открытой стрелкой в форме буквы "V"



Вид сообщения

complete – **полное сообщение**, для которого существует источник и получатель, изображается рассмотренным ранее образом в зависимости от сорта сообщения.

unknown – **неизвестное сообщение**, для которого отсутствуют событие передачи и событие приема.

Эти сообщения не должны представляться на диаграмме последовательности.

lost – **потерянное сообщение**, для которого существует источник и отсутствует приемник, изображается в форме небольшого черного круга на конце стрелки сообщения. Оно интерпретируется как сообщение, которое никогда не достигнет своего места назначения

found – **найденное сообщение**, для которого существует приемник и отсутствует источник, изображается в форме небольшого черного круга на начальном конце сообщения. Оно интерпретируется как сообщение, инициатор которого находится за пределами области описания



Пример диаграммы последовательности

Сценарий

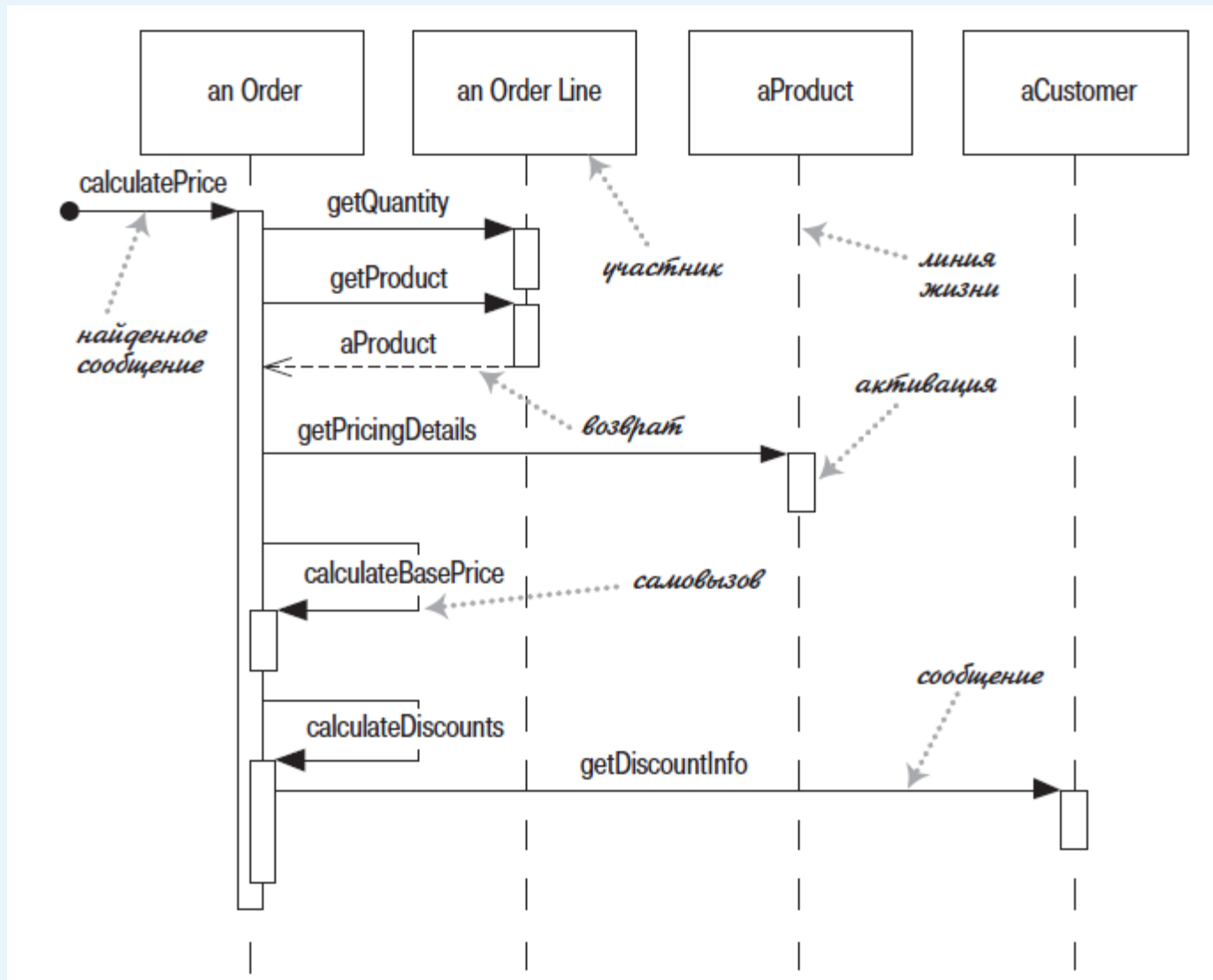
Предположим, что у нас есть заказ, и мы собираемся вызвать операцию для определения его стоимости.

При этом объекту заказа (**Order**) необходимо просмотреть все позиции заказа (**Line Items**) и **определить их** цены, основанные на правилах построения цены продукции в строке заказа (**Order Line**).

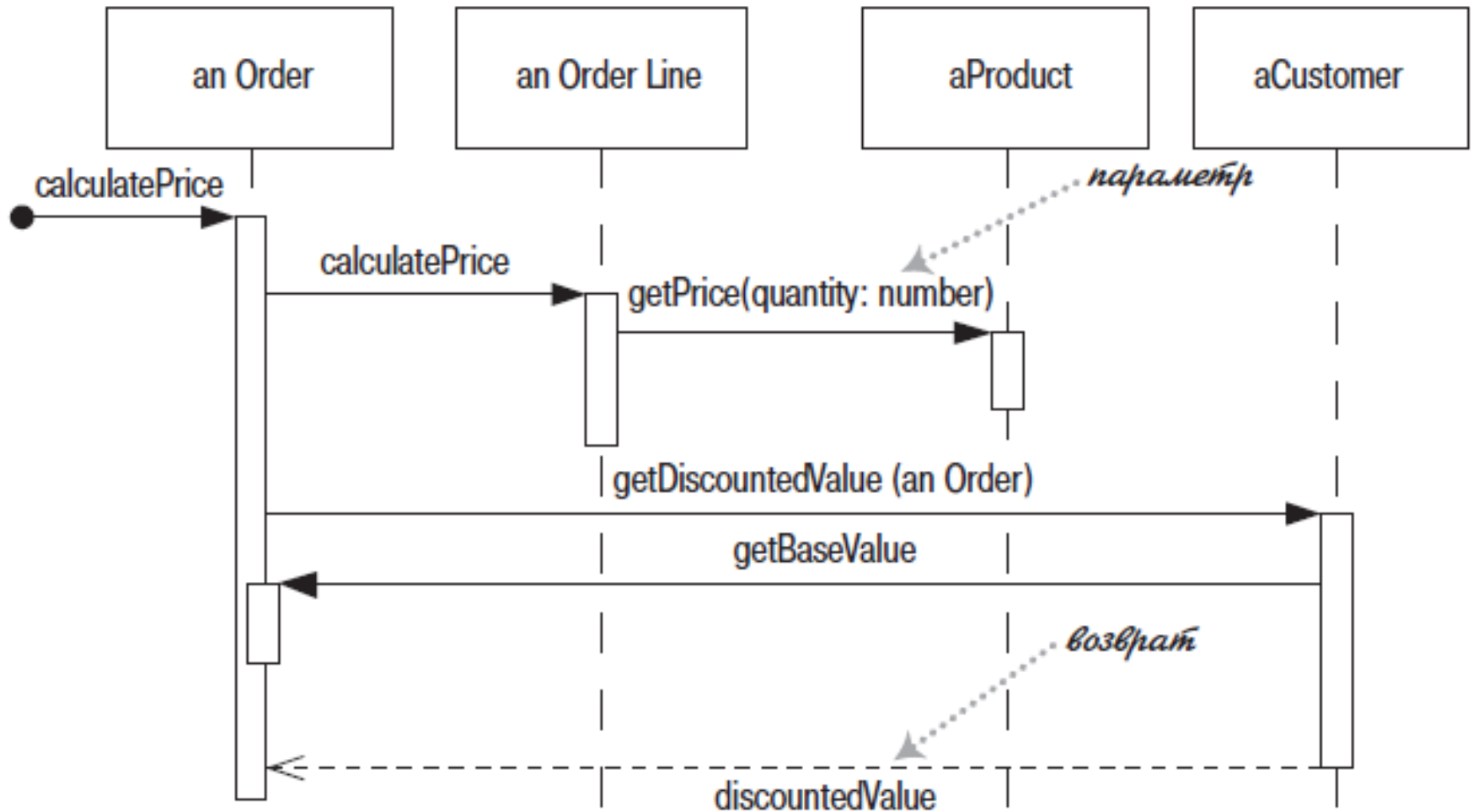
Проделав это для всех позиций заказа, объект заказа должен вычислить общую скидку.



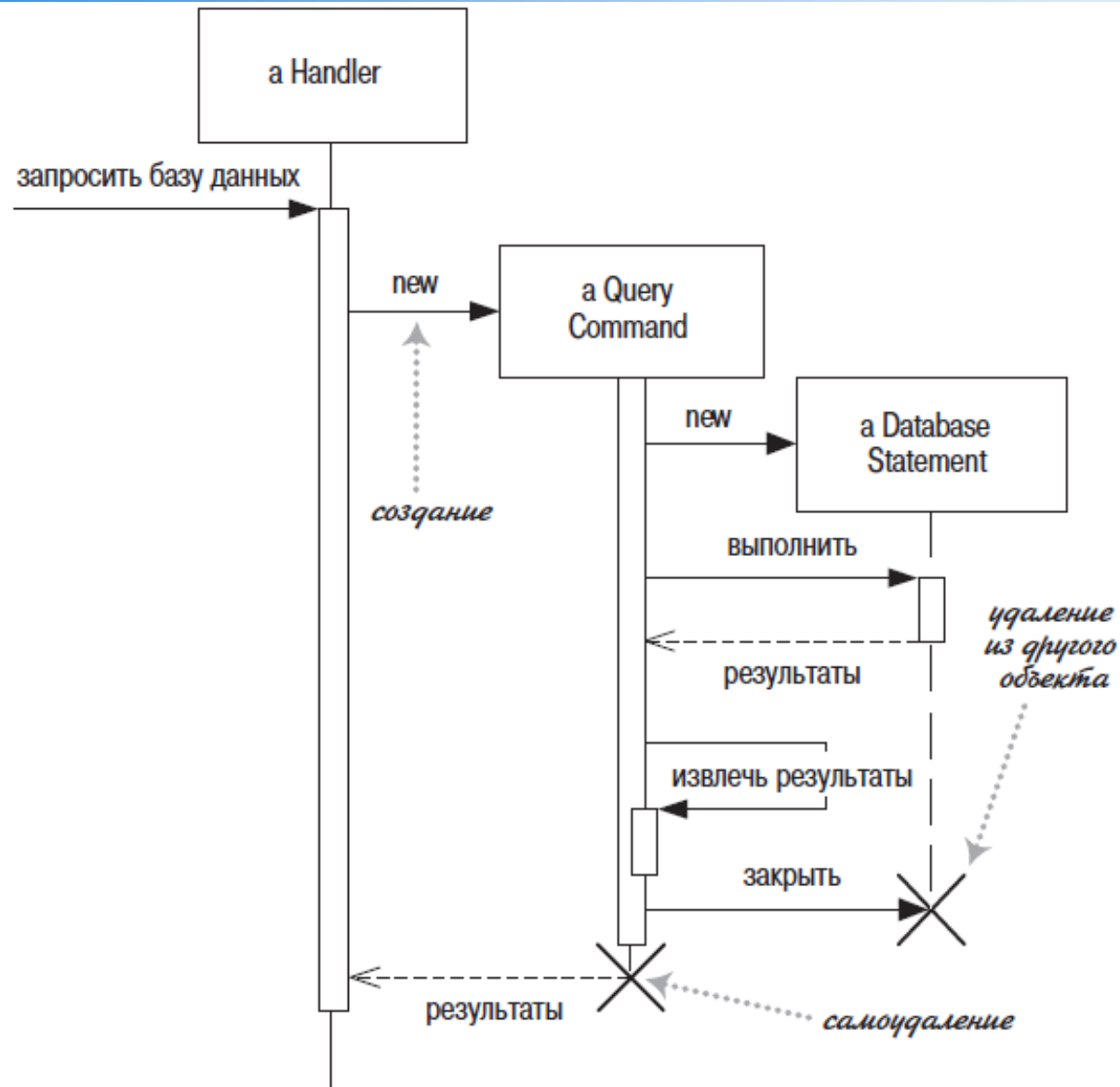
Реализация централизованного управления



Реализация распределенного управления



Создание и удаление участников взаимодействия



Циклы, условия и тому подобное

Проблема: как изображать на диаграмме последовательности циклы и ветвления?

Важно! Диаграмма последовательности не предназначена для этого.

Диаграммы последовательности применяются для визуализации процесса взаимодействия объектов, а не как средство моделирования алгоритма управления.

Фреймы взаимодействий, комбинированные фрагменты – средство разметки диаграммы последовательности.

Операнд взаимодействия (interaction operand) – отдельный фрагмент взаимодействия, предназначенный для использования в качестве внутренней части фрейма взаимодействия.

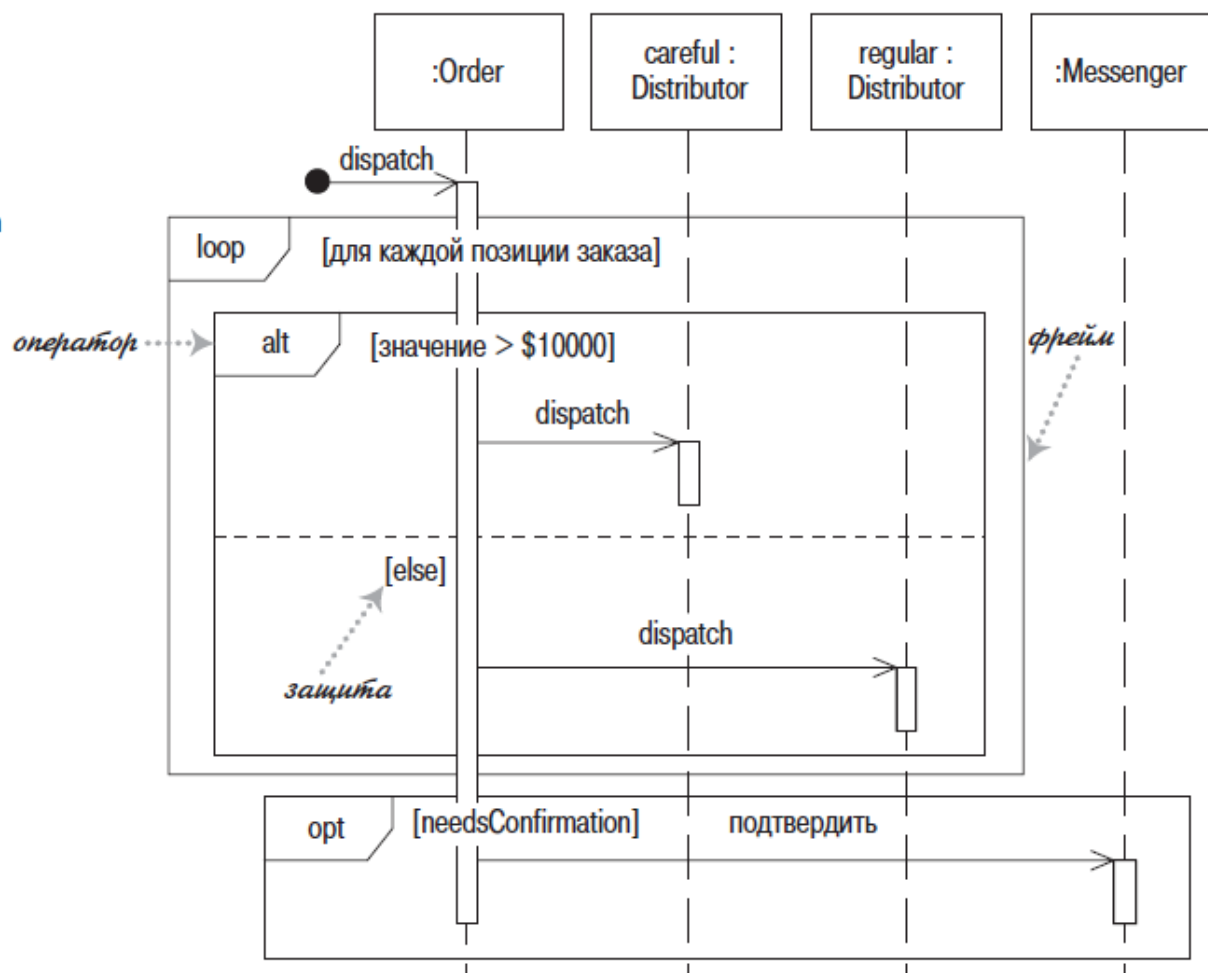
Ограничение взаимодействия (interaction constraint) представляет собой логическое выражение, которое выступает в роли сторожевого условия некоторого операнда во фрейме взаимодействия.



Фреймы взаимодействия: пример графического отображения

```
foreach (lineitem)
  if (product.value > $10K)
    careful.dispatch
  else
    regular.dispatch
  end if
end for
if (needsConfirmation) messenger.confirm
end procedure
```

В основном фреймы состоят из некоторой области диаграммы последовательности, разделенной на несколько фрагментов. Каждый фрейм имеет оператор, а каждый фрагмент может иметь защиту



Операторы фреймов

Оператор	Значение
alt	несколько альтернативных фреймов (alternative); выполняется только тот фрейм, условие которого истинно
break	специфицирует фрейм, который представляет некоторый сценарий завершения
critical	критическая область (critical region); фрагмент может иметь только один поток, выполняющийся за один прием
consider	специфицирует фрейм <i>Рассмотрение</i> (consider), в котором изображены только те типы сообщений, какие должны рассматриваться в этом фрагменте
ignore	специфицирует фрейм <i>Игнорирование</i> (ignore), в котором имеются некоторые типы сообщений, не изображенные на данной диаграмме
loop	Цикл (loop); фрагмент может выполняться несколько раз, а защита обозначает тело итерации
neg	Отрицательный (negative) фрагмент; обозначает неверное взаимодействие
opt	Необязательный (optional) фрагмент; выполняется, только если условие истинно. Эквивалентно alt с одной веткой
par	Параллельный (parallel); все фрагменты выполняются параллельно
ref	Ссылка (reference); ссылается на взаимодействие, определенное на другой диаграмме.
strict	Специфицирует фрейм <i>Строгое следование</i> (strict sequencing), который состоит из нескольких операндов и представляет строгий порядок следования поведений отдельных операндов
sd	Диаграмма последовательности (sequence diagram); используется для очерчивания всей диаграммы последовательности, если это необходимо

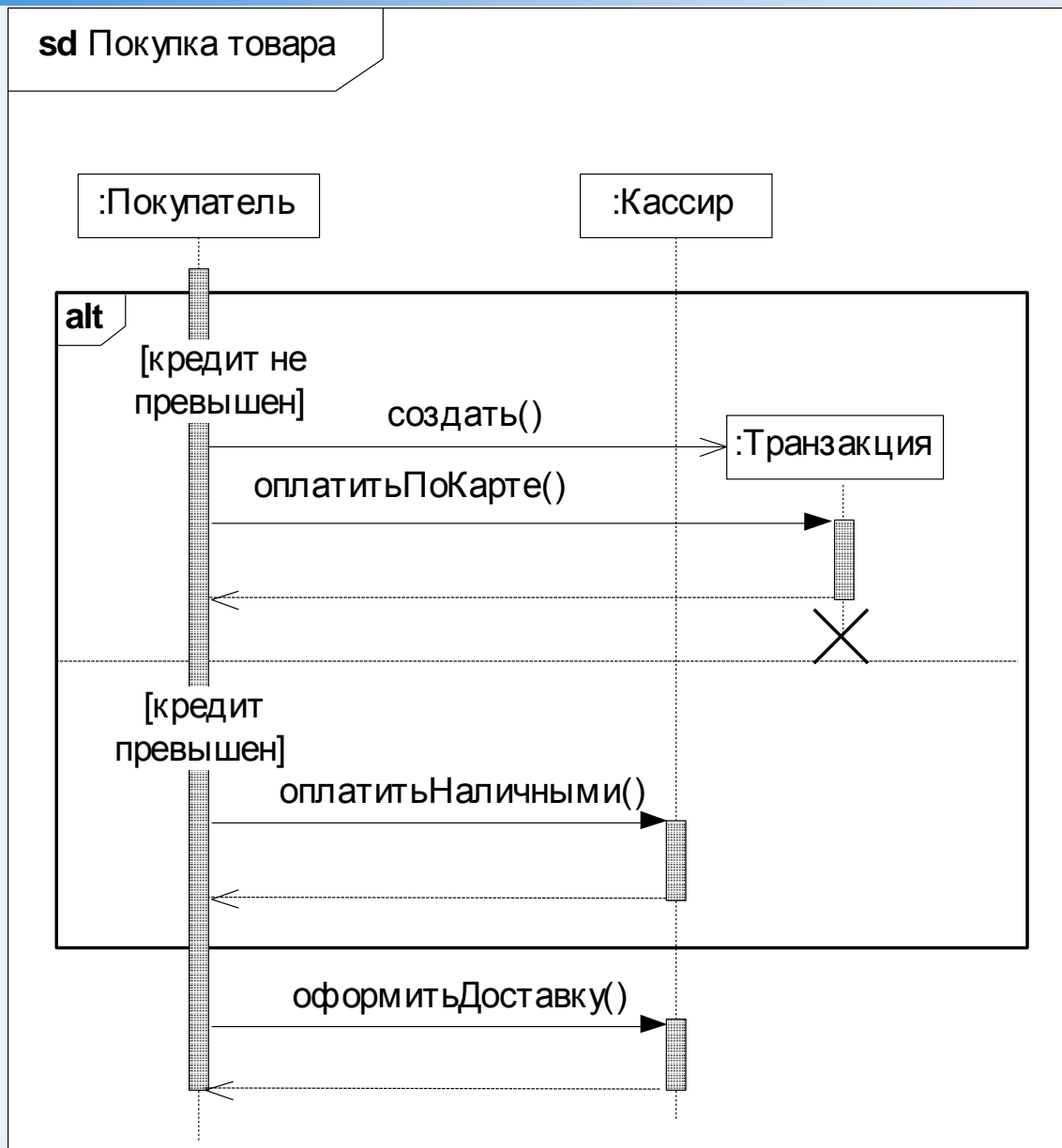


Альтернатива (alt)

- ✓ Несколько альтернативных фрагментов (alternative); выполняется только тот фрагмент, условие которого истинно
- ✓ Выбор может быть сделан не более одного из операндов
- ✓ Выбранный операнд должен иметь явное или неявное выражение сторожевого условия, которое в этой точке взаимодействия должно принимать значение «истина»
- ✓ Если операнд не имеет никакого сторожевого условия, то неявно предполагается, что сторожевое условие имеет значение «истина»
- ✓ Операнд, помеченный сторожевым условием [else], обозначает отрицание дизъюнкции всех других сторожевых условий этого фрейма взаимодействия



Пример фрейма взаимодействия «Альтернатива» (alt)



Завершение (break)

- ✓ Завершение; специфицирует фрейм, который представляет некоторый сценарий завершения
- ✓ Этот сценарий выполняется вместо оставшейся части фрагмента взаимодействия, который содержит этот соответствующий операнд.
- ✓ Обычно оператор Завершение содержит некоторое сторожевое условие
- ✓ Если это сторожевое условие принимает значение "истина", то выполняется комбинированный фрагмент Завершение, а оставшаяся часть фрагмента взаимодействия, содержащего этот операнд, игнорируется
- ✓ Если сторожевое условие операнда Завершение принимает значение "ложь", то операнд Завершение игнорируется, и выполняется оставшаяся часть фрагмента взаимодействия, содержащего этот операнд



Пример фрейма Завершение

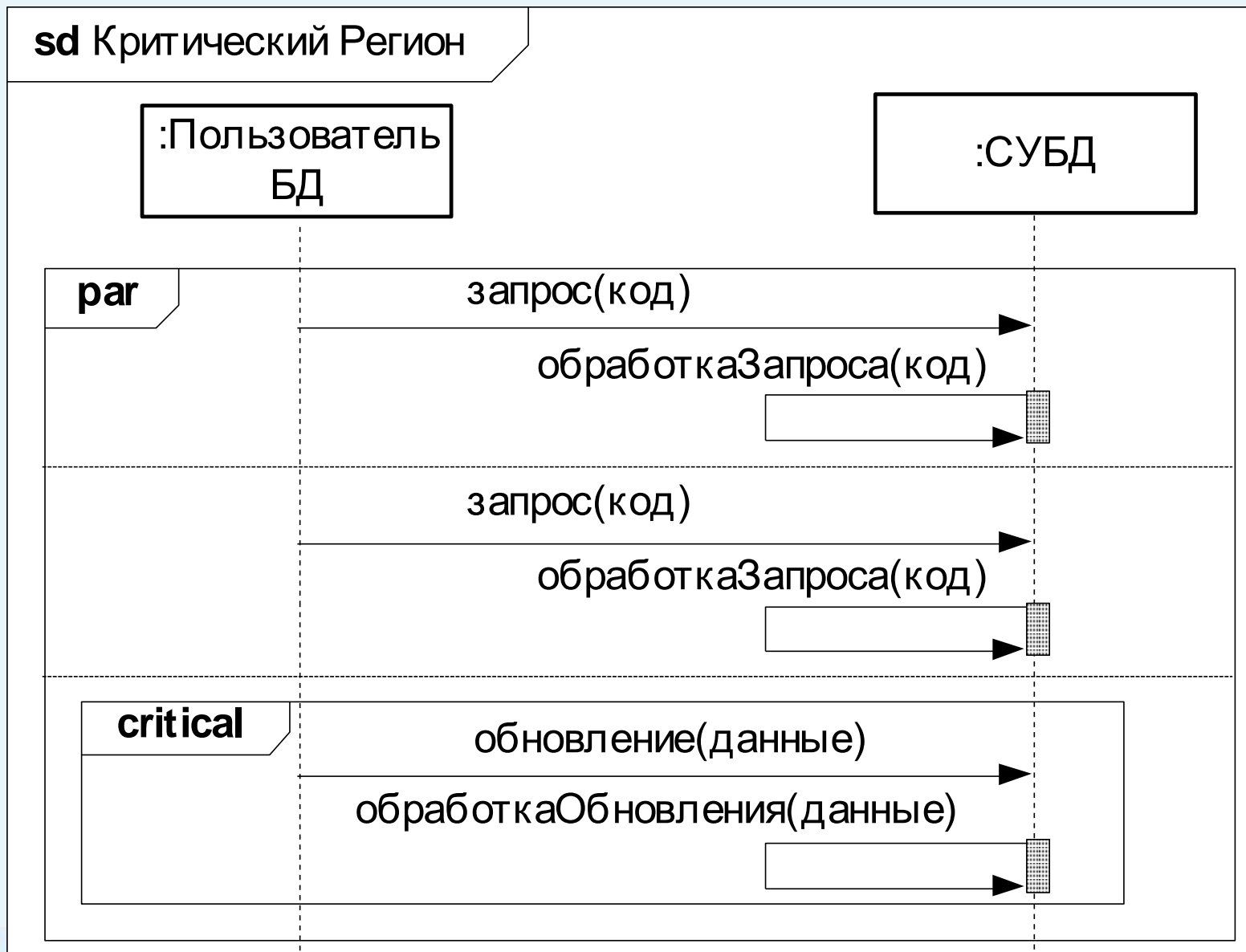


Критический регион (critical)

- ✓ Критическая область (critical region); фрагмент может иметь только один поток, выполняющийся за один прием
- ✓ Критический регион рассматривается как неделимый при определении множества возможных траекторий диаграммы или региона, который его содержит
- ✓ Множество траекторий критического региона не может прерываться другими событиями, происходящими вне этого региона
- ✓ На практике Критический регион используется, как правило, совместно с оператором параллельности



Пример фрейма Критический региона (critical)



Рассмотрение (consider)

- ✓ Оператор взаимодействия **consider** специфицирует фрейм *Рассмотрение* (consider), в котором изображены только те типы сообщений, какие должны рассматриваться в этом фрагменте
- ✓ Это эквивалентно определению того, что при рассмотрении данного фрагмента игнорируются любые другие сообщения, которые не изображены в этом фрейме.
- ✓ Для фрагмента Рассмотрение используется нотация фрейма с оператором, в качестве которого используется ключевое слово **consider**

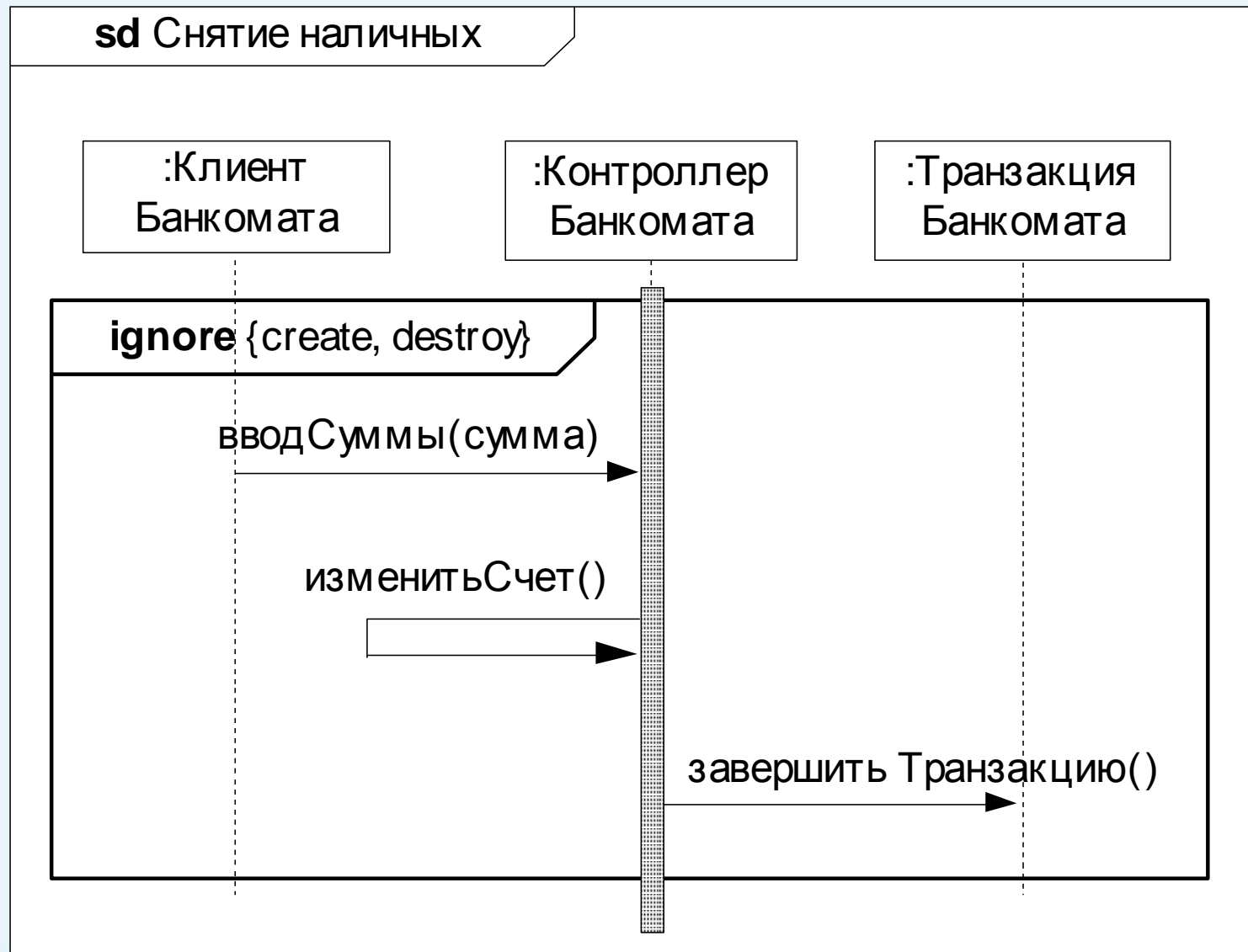


Игнорирование (ignore)

- ✓ Оператор взаимодействия **ignore** специфицирует фрейм *Игнорирование* (ignore), в котором имеются некоторые типы сообщений, не изображенные на данной диаграмме
- ✓ Эти типы сообщений рассматриваются как несущественные и могут появляться в траекториях при выполнении соответствующего фрагмента.
- ✓ Для фрагмента Игнорирование используется нотация фрейма с оператором, в качестве которого используется ключевое слово **ignore**
- ✓ В фигурных скобках перечисляются игнорируемые взаимодействия



Пример фрейма Игнорирование (ignore)



Цикл (loop)

- ✓ Оператор взаимодействия **loop** специфицирует фрейм *Викл* (loop), который представляет собой циклическое повторение некоторой последовательности сообщений.
- ✓ Операнд цикла повторяется несколько раз
- ✓ Дополнительное сторожевое условие может включать нижний и верхний пределы числа повторений цикла, а также некоторое логическое выражение.
- ✓ $\langle \text{minint} \rangle$ неотрицательное натуральное число, которое обозначает минимальное количество итераций цикла;
- ✓ $\langle \text{maxint} \rangle$ натуральное число, которое обозначает максимальное количество итераций цикла.
- ✓ Значение $\langle \text{maxint} \rangle$ должно быть больше или равно $\langle \text{minint} \rangle$ | '*'. Здесь символ '*' означает бесконечность
- ✓ После того, как минимальное число повторений будет выполнено, проверяется логическое выражение сторожевого условия
- ✓ Если это логическое выражение принимает значение "ложь", то выполнение цикла на этом заканчивается
- ✓ Если же логическое выражение принимает значение "истина", а количество выполненных итераций не превышает значения $\langle \text{maxint} \rangle$, то происходит еще одно выполнение цикла
- ✓ После этого снова проверяется логическое выражение сторожевого условия, аналогично процедуре выполнения минимального числа повторений



Пример фрейма Цикл (Loop)

sd Снятие наличных

:Клиент
Банкомата

:Клавиатура
Банкомата

:Контроллер
Банкомата

loop (1, 3)

[result==false]

вводПИНкода()

result=проверитьКод()



Отрицание (neg)

- ✓ Оператор взаимодействия **neg** специфицирует фрейм *Отрицание* (negative), представляющий траектории, которые определяются как недействительные или недопустимые
- ✓ Множество траекторий, которые определяют фрейм с оператором взаимодействия **neg**, равно множеству траекторий, заданных его единственным операндом
- ✓ При этом в это множество входят только недействительные или запрещенные траектории
- ✓ Все фреймы, кроме Отрицания, рассматриваются в положительном смысле, т.е. они описывают траектории, которые являются допустимыми и возможными.



Пример фрейма Отрицание (neg)



Необязательный (opt)

- ✓ Оператор взаимодействия **opt** специфицирует фрейм *Необязательный* (option), который представляет выбор поведения, когда или выполняется единственный операнд, или вовсе ничего не выполняется
- ✓ Оператор выбора семантически эквивалентен альтернативному фрейму, в котором имеется один операнд с непустым содержанием, а второй операнд отсутствует.
- ✓ Необязательный фрейм состоит из одного операнда со сторожевым условием
- ✓ Операнд выполняется, если выполнено сторожевое условие. В противном случае операнд не выполняется.



Параллельный (par)

- ✓ Оператор взаимодействия **par** специфицирует фрейм *Параллельный* (parallel), который представляет некоторое параллельное выполнение взаимодействий своих операндов
- ✓ Наступление событий у различных операндов могут чередоваться во времени произвольным образом
- ✓ Внутри каждого операнда соблюдается порядок следования сообщений
- ✓ Каждый операнд изображается в отдельном регионе, который отделяется от других регионов пунктирной линией



Точное следование (strict)

- ✓ Оператор взаимодействия **strict** специфицирует фрейм *Строгое следование* (strict sequencing), который состоит из нескольких операндов и представляет строгий порядок следования поведений отдельных операндов
- ✓ Данный оператор указывает, что операнды верхнего уровня фрейма выполняются в строго определенном порядке (сверху вниз) и не перекрываются.
- ✓ Строгий порядок следования означает, что вертикальная координата вложенных фреймов имеет значение на всем протяжении границ фрейма, а не только для одной линии жизни
- ✓ Вертикальная позиция спецификации наступления события задается вертикальной позицией соответствующей точки
- ✓ Вертикальная позиция других фреймов взаимодействия задается самой верхней вертикальной позицией соответствующих фреймов.

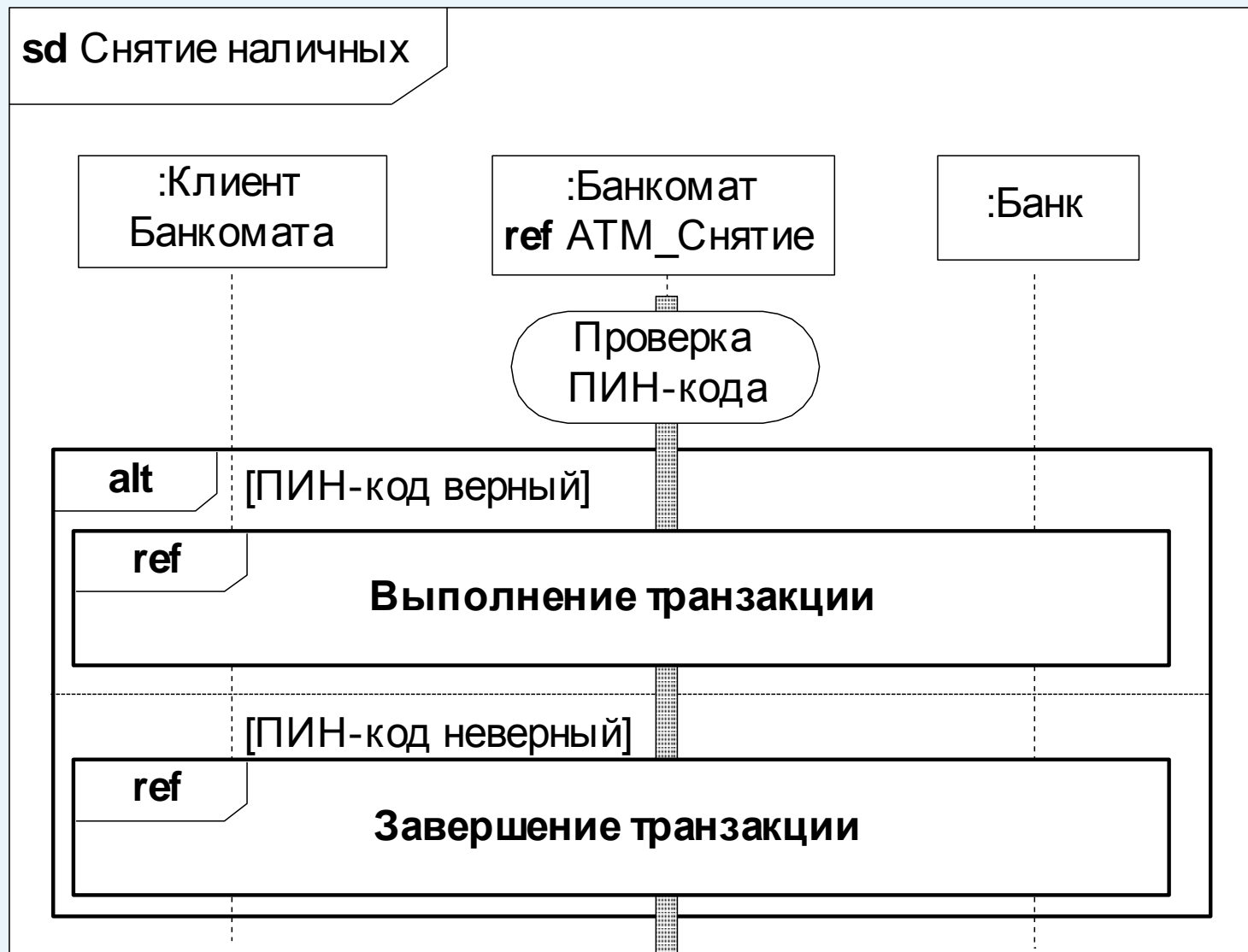


Инвариант состояния

- Является некоторым ограничением времени выполнения, которое должно быть выполнено для отдельных участников взаимодействия
- Инвариант состояния изображается в форме символа состояния на линии жизни соответствующего участника взаимодействия
- Символ состояния представляет эквивалент ограничения, которое проверяет состояние объекта, представленного данной линией жизни
- Это может быть внутреннее состояние поведения объекта соответствующего класса или некоторое внешнее состояние, основанное на представлении “черный ящик” для данной линии жизни.



Пример представления инварианта состояния в форме символа состояния

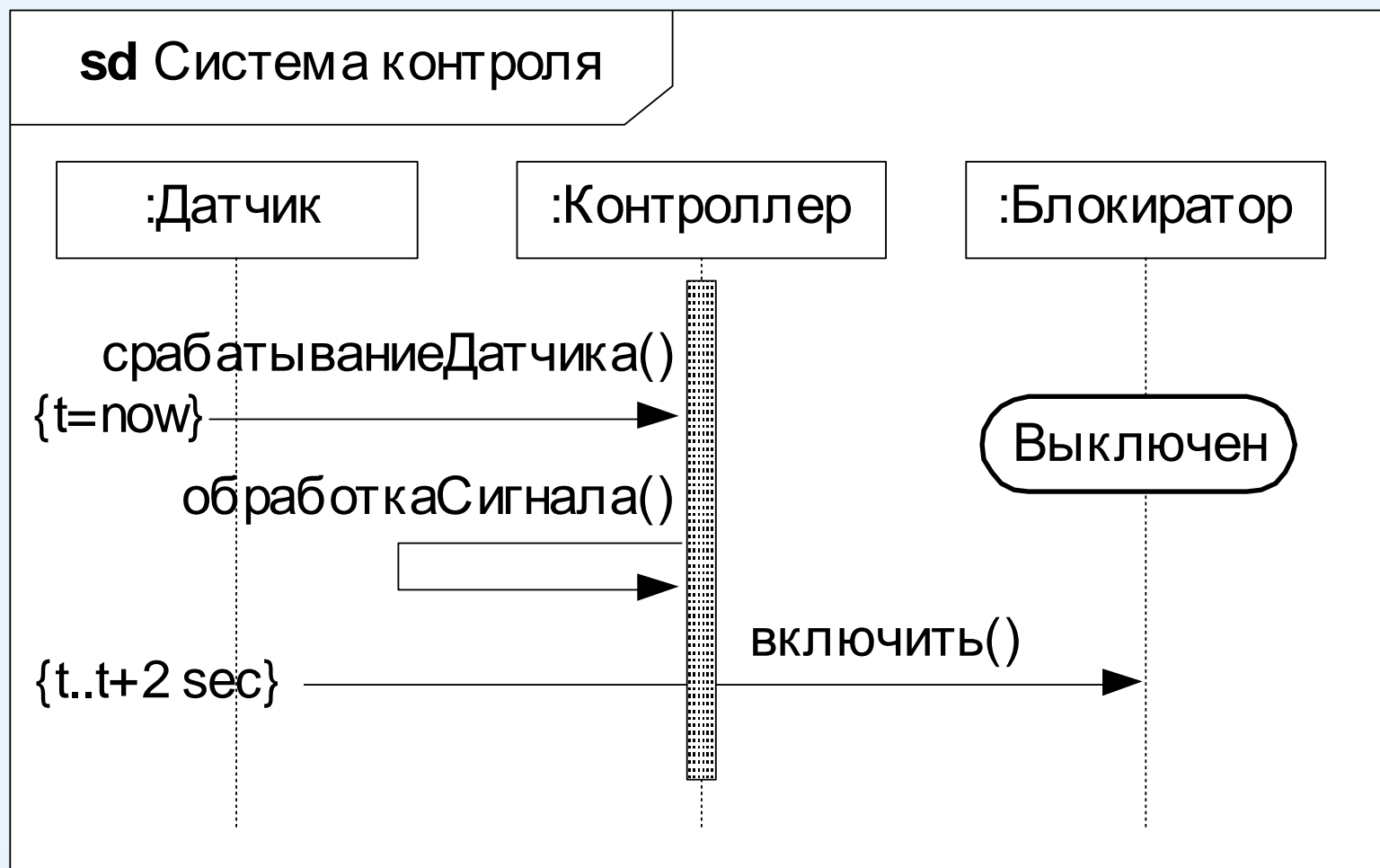


Пример представления инварианта состояния в форме ограничения



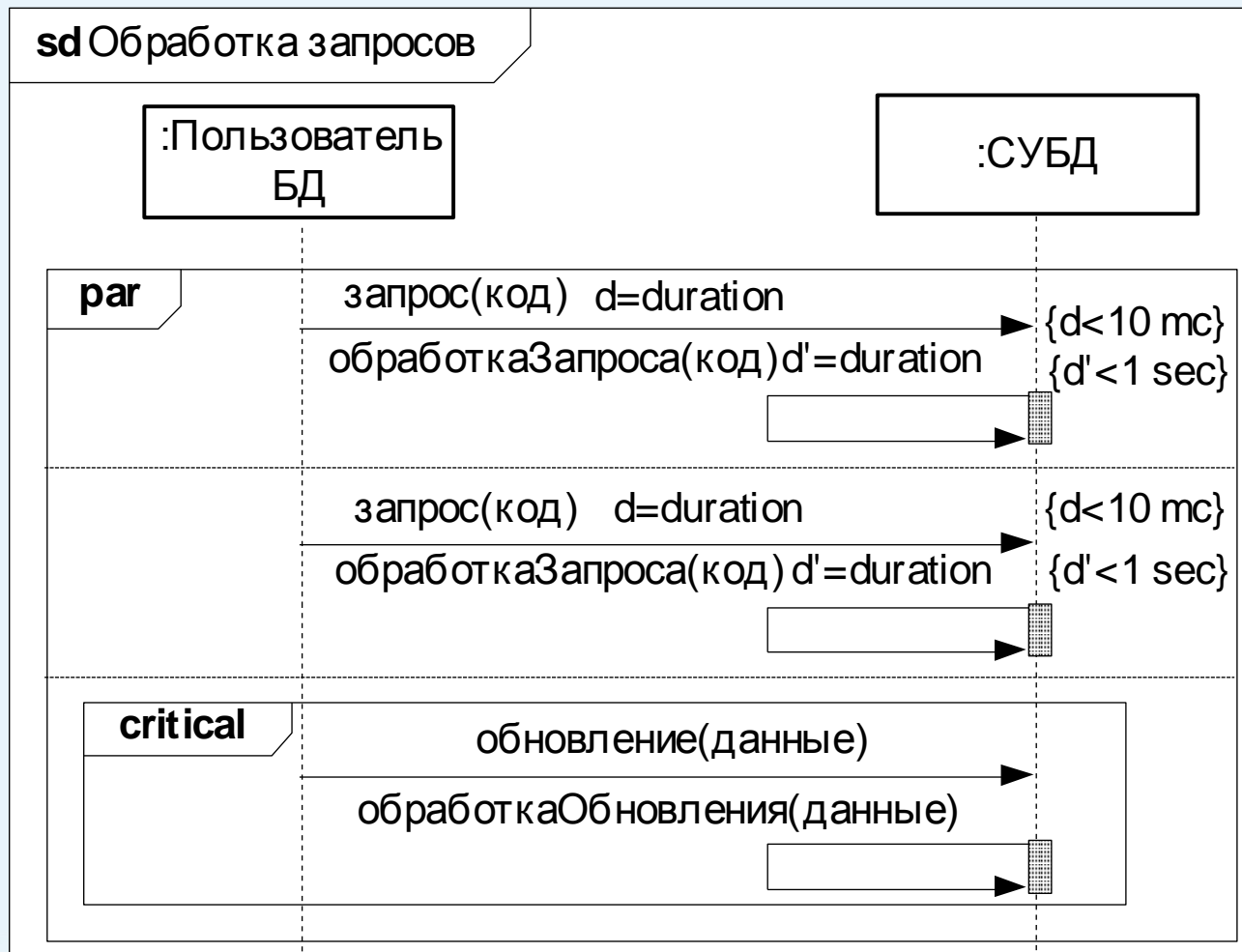
Временное ограничение

представляет собой специальное ограничение, записанное в форме временного интервала.



Ограничение продолжительности

определяет ограничение, которое ссылается на некоторый интервал продолжительности



Рекомендации по построению диаграмм последовательности

- ✓ Построение диаграммы последовательности целесообразно начинать с выделения из всей совокупности тех и только тех классов, объекты которых участвуют в моделируемом взаимодействии. После этого все объекты наносятся на диаграмму с соблюдением некоторого порядка инициализации сообщений (крайний слева – объект, иницирующий взаимодействие). Здесь необходимо установить, какие объекты будут существовать постоянно, а какие временно – только на период выполнения ими требуемых действий.
- ✓ При спецификации сообщений следует учитывать те роли, которые играют сообщения в системе. При необходимости уточнения этих ролей надо использовать их разновидности и стереотипы. Для уничтожения объектов, которые создаются на время выполнения своих действий, нужно предусмотреть явное сообщение.
- ✓ Ветвления процесса взаимодействия можно изобразить на одной диаграмме с использованием соответствующих графических примитивов. Однако каждый альтернативный поток управления может существенно затруднить понимание построенной модели. Поэтому общим правилом является визуализация каждого потока управления на отдельной диаграмме последовательности. В этой ситуации такие отдельные диаграммы должны рассматриваться совместно как одна модель взаимодействия.
- ✓ Далее, при необходимости, вводятся временные ограничения на выполнение отдельных действий в системе. Для простых асинхронных сообщений временные ограничения могут отсутствовать. Необходимость синхронизировать сложные потоки управления, как правило, требует введение в модель таких ограничений.



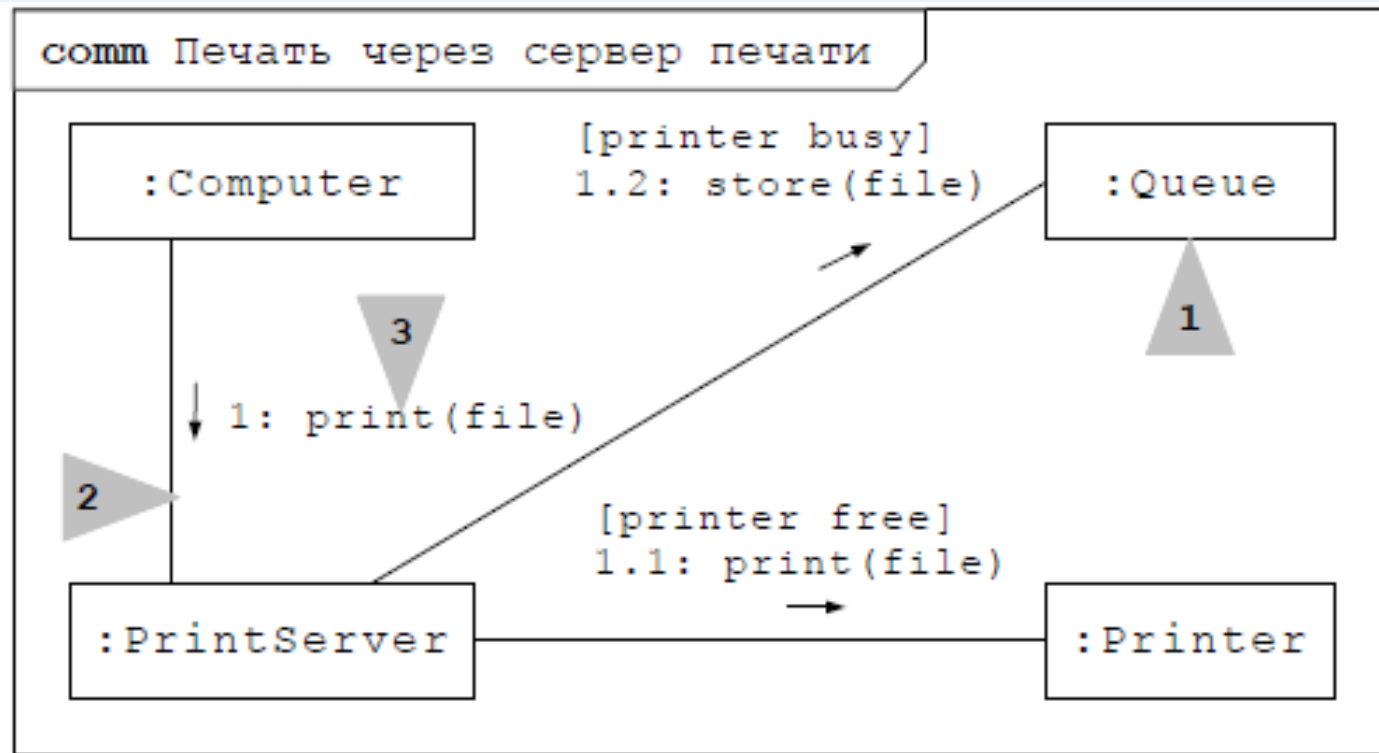
ДИАГРАММА КОММУНИКАЦИИ

предназначена для представления взаимодействия в контексте внутренней архитектуры системы и передаваемых сообщений; семантически эквивалентна диаграмме последовательности

- ✓ Диаграмма коммуникации имеет вид графа, вершинами которого являются роли взаимодействия, изображенные в виде прямоугольников
- ✓ Эти вершины соответствуют линиям жизни и изображаются в своем структурном контексте
- ✓ Ребрами графа являются связи, по которым проходят маршруты коммуникации
- ✓ Линии жизни могут обмениваться сообщениями, которые изображаются в виде небольших стрелок с некоторым именем, расположенных возле линий связей



Основные элементы диаграммы коммуникаций



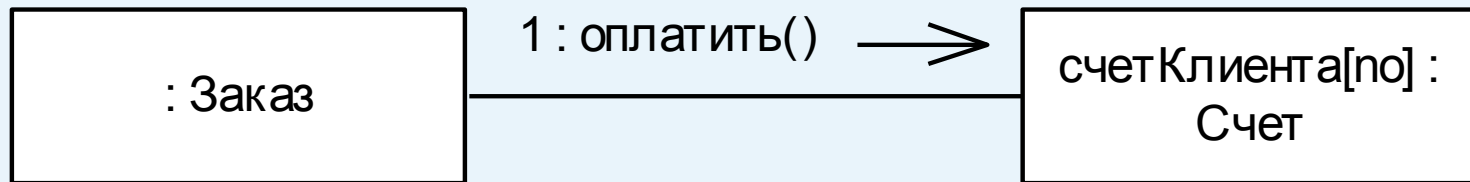
Изображаются экземпляры взаимодействующих классов (1) и один тип отношений — связи (2). Акцент делается не на времени, а на структуре связей между конкретными экземплярами. Взаимное положение элементов на диаграмме коммуникаций не имеет значения — важны только связи, вдоль которых передаются сообщения (3)

Связь (link) и сообщение (message)

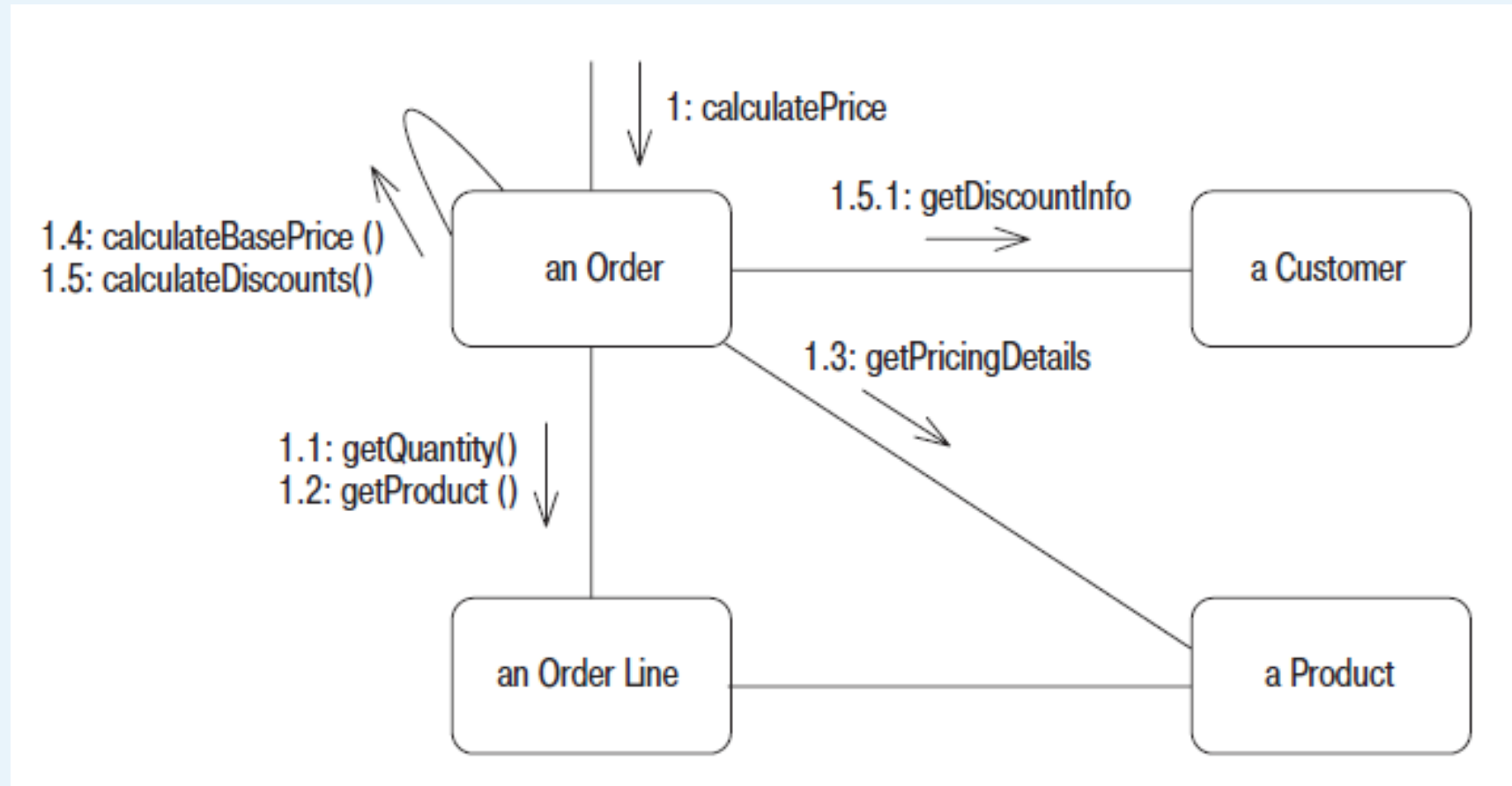
является экземпляром произвольной ассоциации, которая обеспечивает канал для направленной передачи сообщений между линиями жизни

Сообщение изображается в форме символа стрелки рядом с линией связи, которое передается в указанном стрелкой направлении по данной связи.

Рядом со стрелкой указывается идентификатор сообщения, записанный в специальном формате.



Пример диаграммы коммуникаций



Вложенная нумерация позволяет указать в явном виде вызов одного метода из другого (см. метод `getDiscountInfo`)



Примеры формата записи сообщений

2:display (x, y)

Simple message

1.3.1:p:= find(specs)

Nested call with return value

[x < 0] 4:invert (x, color)

Conditional message

3.1*:update ()

Iteration

A3,B4/ C2:copy(a,b)

Synchronization with other threads

Кроме чисел в сообщениях можно увидеть и буквы. Эти символы обозначают различные потоки управления. Так, A3 и B4 могут быть различными потоками.



Процессы и нити

Активный объект - это объект, который владеет процессом или нитью и может инициировать управляющее воздействие.

Активный класс - это класс, экземплярами которого являются активные объекты. Графически изображается в виде прямоугольника с жирными границами.

Процесс (*Process*) - это ресурсоемкий поток управления, который может выполняться параллельно с другими процессами.

Нить (*Thread*) - это облегченный поток управления, который может выполняться параллельно с другими нитями в рамках одного процесса.

Процессы и нити изображаются в виде стереотипных активных классов.



Отличия процессов от нитей

Процесс известен самой операционной системе и выполняется в независимом адресном пространстве. Все процессы, исполняемые на некотором узле, пользуются равными правами и претендуют на общие ресурсы, которыми располагает узел. Процессы никогда не бывают вложенными. Если узел оборудован несколькими процессорами, то на нем достижим истинный параллелизм. Если же есть только один процессор, то возможна лишь иллюзия параллелизма, обеспечиваемая операционной системой.

Нить называют облегченным потоком управления, поскольку она требует меньше системных ресурсов. Нить может быть известна операционной системе, но чаще всего бывает скрыта от нее внутри некоего процесса, в адресном пространстве которого она и исполняется. Все нити, существующие в контексте некоторого процесса, имеют одинаковые права и претендуют на ресурсы, доступные внутри процесса. Нити также не бывают вложенными. В общем случае существует лишь иллюзия параллельного выполнения нитей, поскольку не нить, а процесс является объектом планирования операционной системы.



Синхронизация

Проблема. Когда поток управления проходит через некоторую операцию, то эта операция является точкой выполнения. Если операция определена в некотором классе, то можно сказать, что точкой выполнения является конкретный экземпляр этого класса. В одной операции (и, стало быть, в одном объекте) могут одновременно находиться несколько потоков управления, а бывает и так, что разные потоки находятся в разных операциях, но все же в одном объекте.

Проблема возникает тогда, когда в одном объекте находится сразу несколько потоков управления.

Это классическая проблема взаимного исключения. Ошибки при обработке этой ситуации могут стать причиной различных видов конкуренции между потоками (Race conditions) и их взаимной интерференции.

Решение: трактовка объекта как критической области. Суть заключается в присоединении к операциям, определенным в классе, некоторых синхронизирующих свойств.



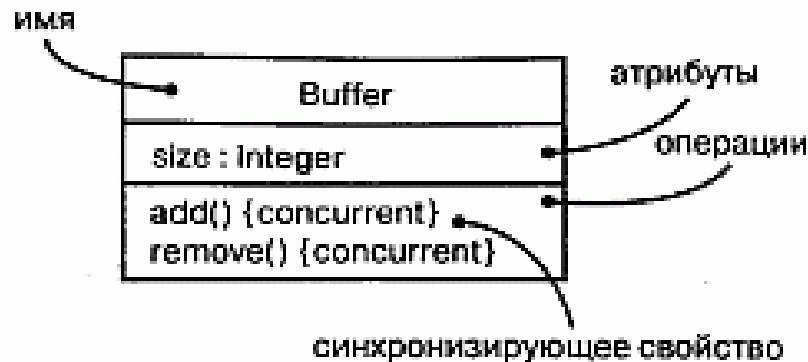
Разновидности синхронизации в UML

sequential (последовательная) - вызывающие стороны должны координировать свои действия еще до входа в вызываемый объект, так что в любой момент времени внутри объекта находится ровно один поток управления. При наличии нескольких потоков управления не могут гарантироваться семантика и целостность объекта.

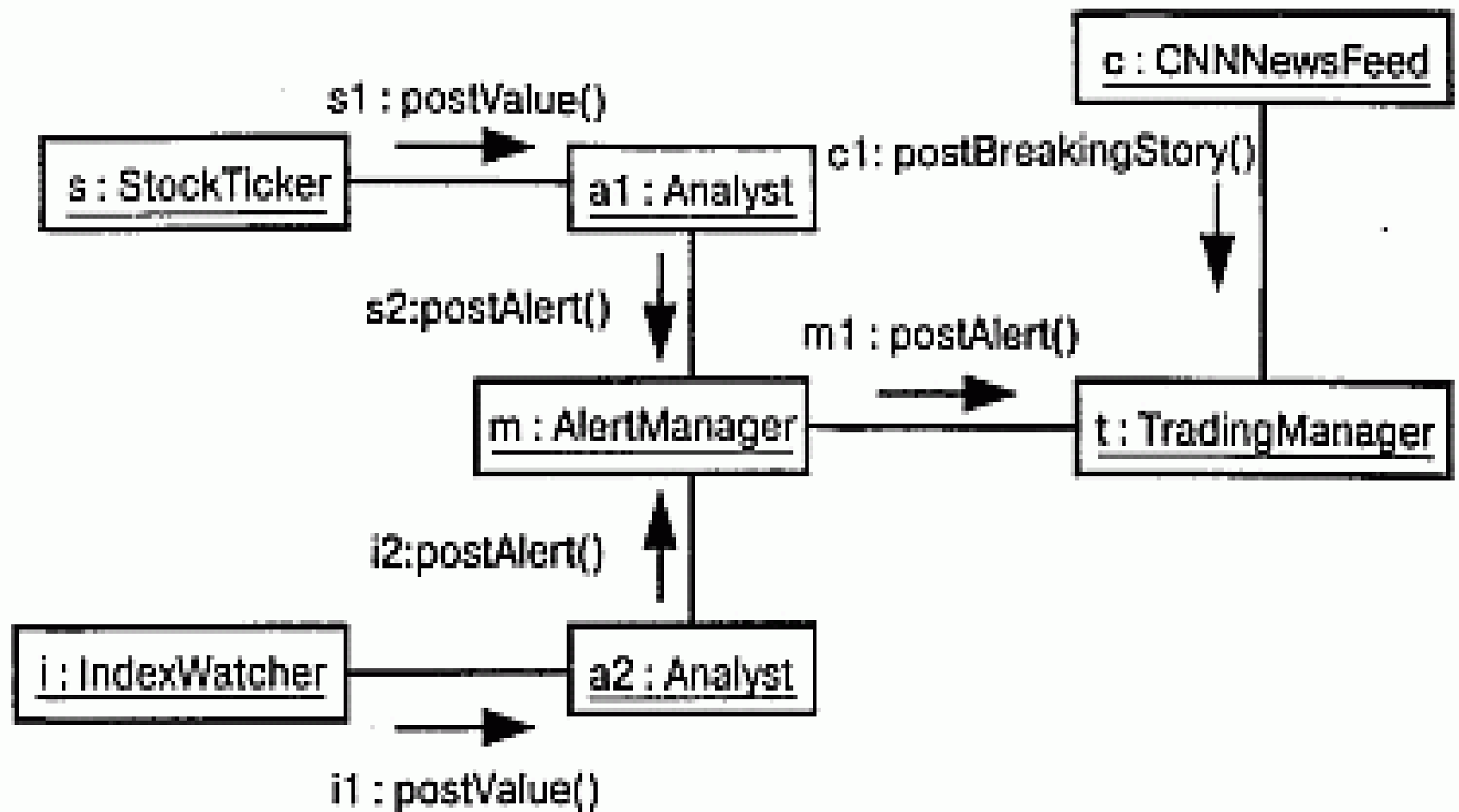
guarded (охраняемая) - семантика и целостность объекта гарантируются при наличии нескольких потоков управления путем упорядочения вызовов всех охраняемых операций объекта. По существу, в каждый момент времени может выполняться ровно одна операция над объектом, что сводит такой подход к последовательному.

concurrent (параллельная) - семантика и целостность объекта при наличии нескольких потоков управления гарантируются тем, что операция рассматривается как атомарная.

В любом языке, поддерживающем параллельность, все три подхода можно реализовать с помощью семафоров (Semaphores)



Пример моделирования нескольких потоков управления



ДИАГРАММЫ РЕАЛИЗАЦИИ

с точки зрения реализации, проектируемая система состоит из компонентов (представленных на **диаграммах компонентов**), распределенных по вычислительным узлам (представленным на **диаграммах размещения**)



ВАЖНО!

В UML 2 по сравнению с UML 1 произошло значительное изменение, а именно, понятие "компонент" было разделено на две составляющие: логическую и физическую.

Логическая составляющая, продолжая носить имя **компонент** (*component*), является элементом **логической модели системы**.

Физическая составляющая, называемая **артефактом** (*artifact*), олицетворяет **физический элемент** проектируемой системы, размещающийся **на вычислительном узле** (*node*).



Диаграммы реализации позволяют моделировать:

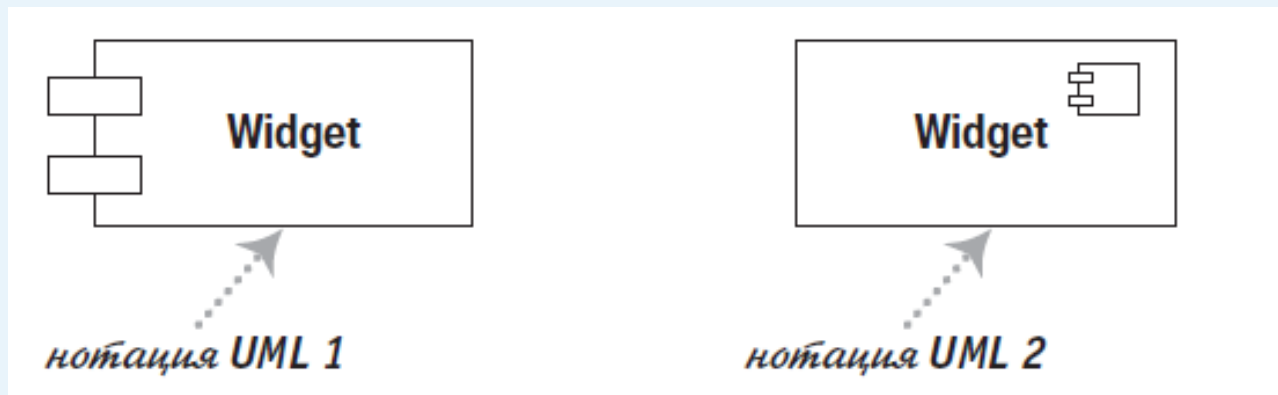
- **структуру логических элементов (компонентов);**
- **отображения логических элементов (компонентов) на физические элементы (артефакты);**
- **структуру используемых ресурсов (узлов) с распределенными по ним физическими элементами (артефактами)**



Диаграмма компонентов

Компонент (component)— это модульный фрагмент логического представления системы, взаимодействие с которым описывается набором обеспеченных и требуемых интерфейсов.

Компонент UML является частью модели, и описывает логическую сущность, которая существует **только во время проектирования** (design time), хотя в дальнейшем ее можно связать с физической реализацией (артефактом) времени исполнения (run time).

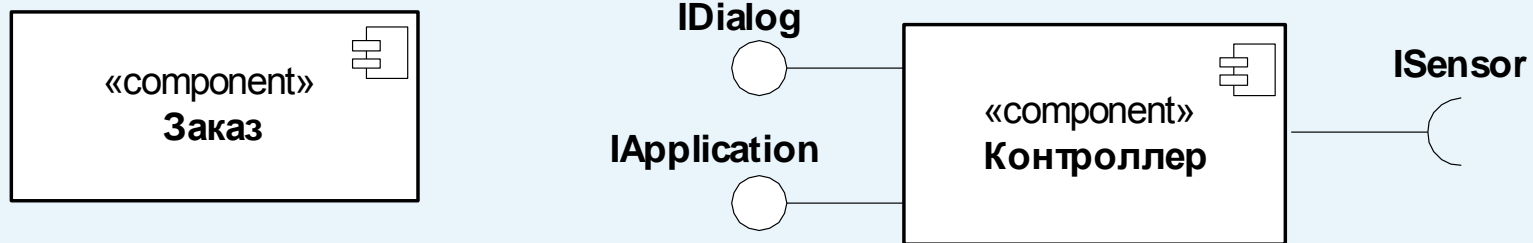


Примеры стереотипов компонентов

Стереотип	Описание
«buildComponent»	компонент, служащий для разработки приложения
«entity»	постоянно хранимый информационный компонент, представляющий некоторое понятие предметной области
«service»	функциональный компонент без состояния, возвращающий запрашиваемые значения без побочных эффектов
«subsystem»	единица иерархической декомпозиции большой системы



Примеры изображения компонентов



Артефакты

Артефакт — это любой созданный искусственно элемент программной системы

К артефактам могут относиться исполняемые файлы, исходные тексты, веб-страницы, справочные файлы, сопроводительные документы, файлы с данными, модели и многое другое, являющееся физическим элементом информации.

Другими словами, артефактами являются те информационные элементы, которые тем или иным способом используются при работе программной системы и входят в ее состав.

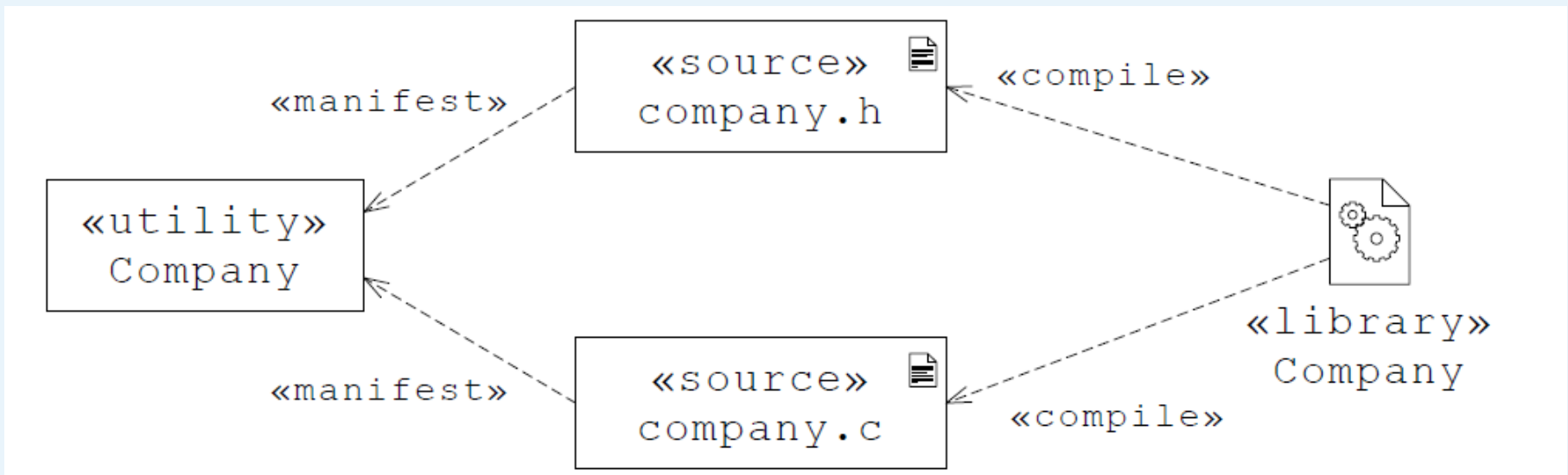


Примеры стереотипов артефактов

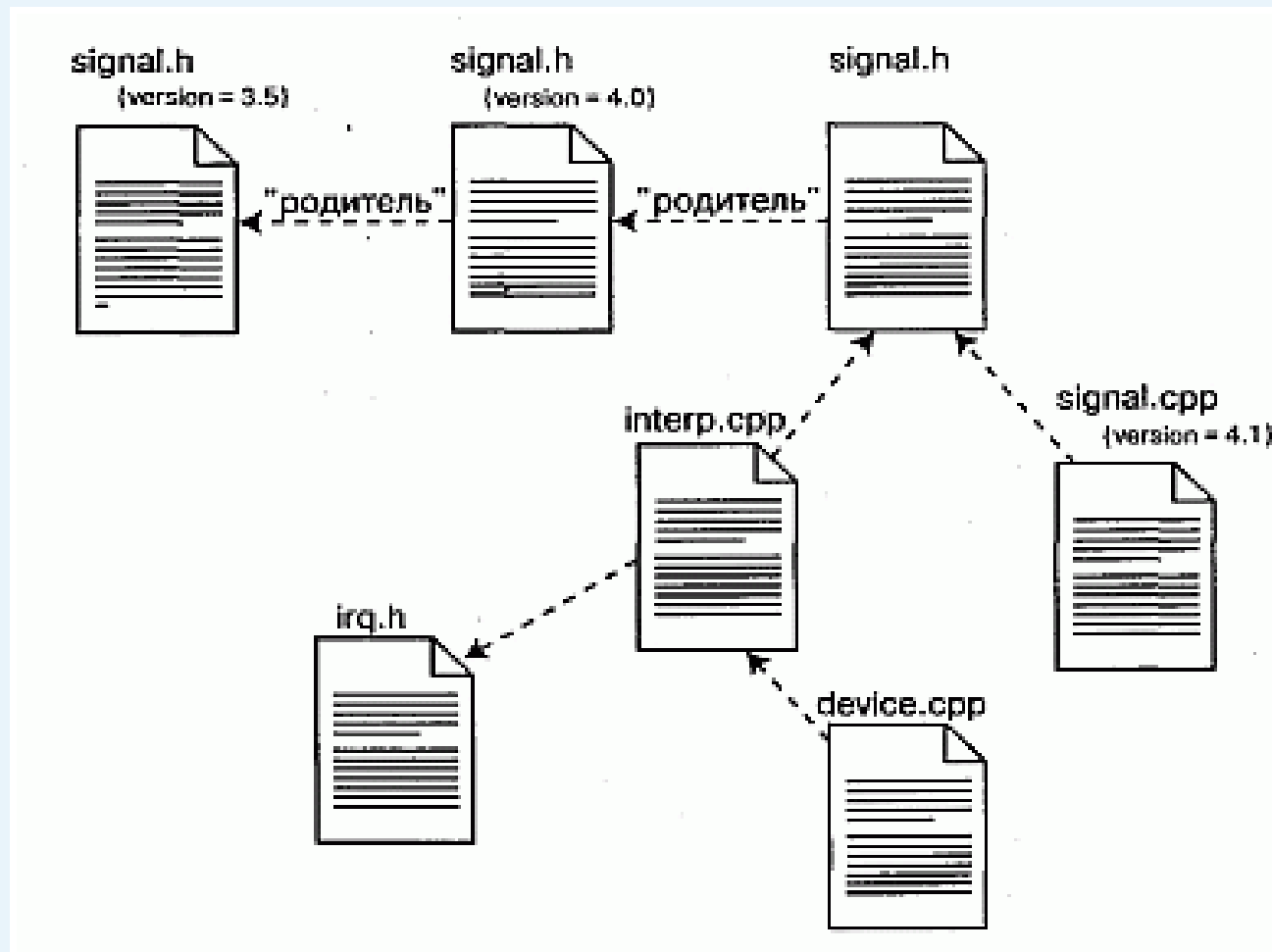
Стереотип	Описание
«file»	Файл любого типа, хранимый в файловой системе
«document»	артефакт, представляющий файл (документ), который не является ни файлом исходных текстов, ни исполняемым файлом
«executable»	выполнимая программа любого вида. Подразумевается по умолчанию, если никакой стереотип не указан
«library»	статическая или динамическая библиотека
«script»	файл, содержащий текст, допускающий интерпретацию соответствующими программным средствами
«source»	файл с исходным кодом программы

Манифестация

Манифестация — это отношение зависимости со стереотипом **«manifest»**, связывающее элемент модели (например, класс или компонент) и его физическую реализацию в виде артефакта.



Пример: моделирование исходного кода



Пример: исполняемая версия

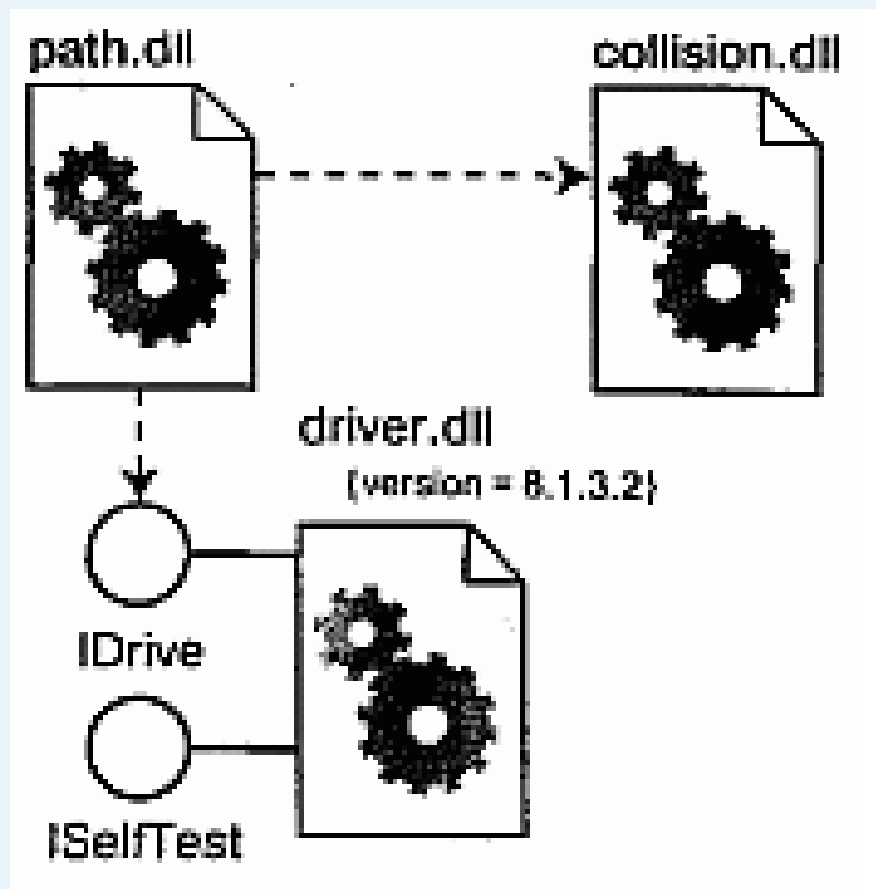


Диаграмма развертывания

- Отражает, на какой платформе и на каких вычислительных средствах реализована программная система.
- Применяется для представления общей конфигурации и топологии распределенной программной системы и содержит распределение компонентов по отдельным узлам системы.
- Показывает наличие физических соединений – маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.



Правила построения диаграммы развертывания

- ✓ Показывать элементы и компоненты программы, существующие только на этапе ее исполнения (runtime).
- ✓ Представлять только компоненты-экземпляры программы, являющиеся исполнимыми файлами или динамическими библиотеками.
- ✓ Компоненты, которые не используются на этапе исполнения, на диаграмме развертывания не показывать.
- ✓ Содержит графические изображения процессоров, устройств, процессов и связей между ними.
- ✓ Является единой для системы в целом
- ✓ Как правило, разработка диаграммы развертывания является последним этапом спецификации модели



Основные цели построения диаграммы развертывания

- Определить распределение компонентов системы по ее физическим узлам.
- Показать физические связи между всеми узлами реализации системы на этапе ее исполнения.
- Выявить узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности.



Диаграмма развертывания

Узел (node) — это физический вычислительный ресурс, участвующий в работе системы.

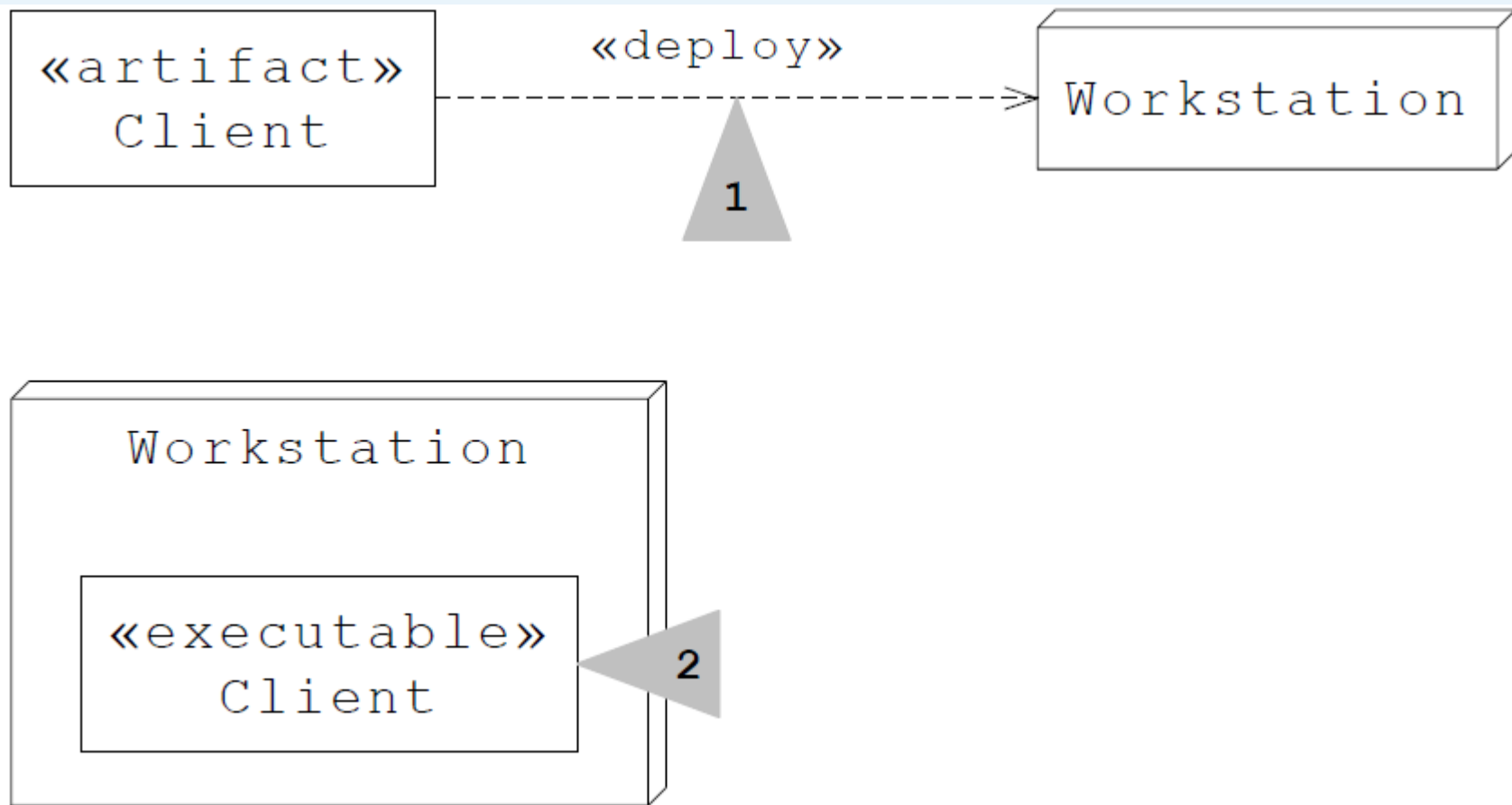
В UML 2 предусмотрено два стереотипа для узлов «executionEnvironment» и «device».

Узел со стереотипом «**executionEnvironment**» позволяет моделировать аппаратно-программную платформу, на которой происходит выполнение приложения.

Узел со стереотипом «**device**» также моделирует аппаратно-программную платформу, но допускает возможность вложение одного узла в другой.



Элементы диаграммы развертывания

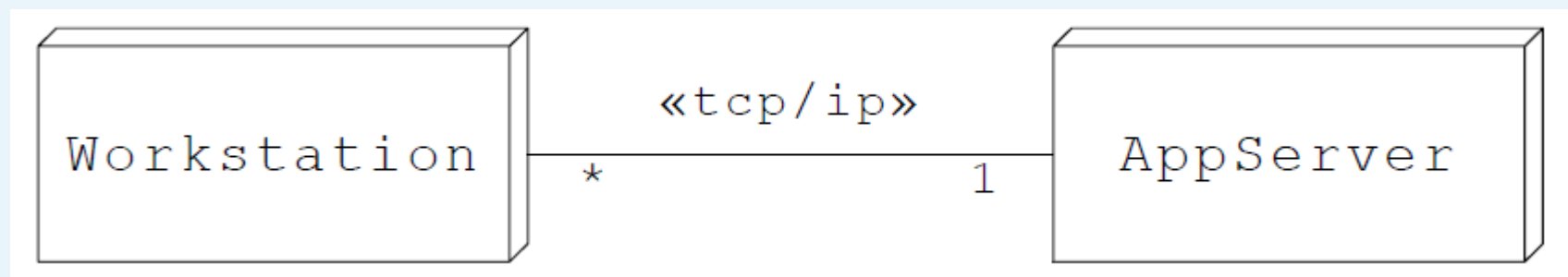


Артефакты системы во время ее работы размещаются на узлах, что графически выражается либо их перечислением внутри узла, либо отношением зависимости со стереотипом «deploy» между артефактом и узлом, либо изображением артефакта внутри изображения узла.

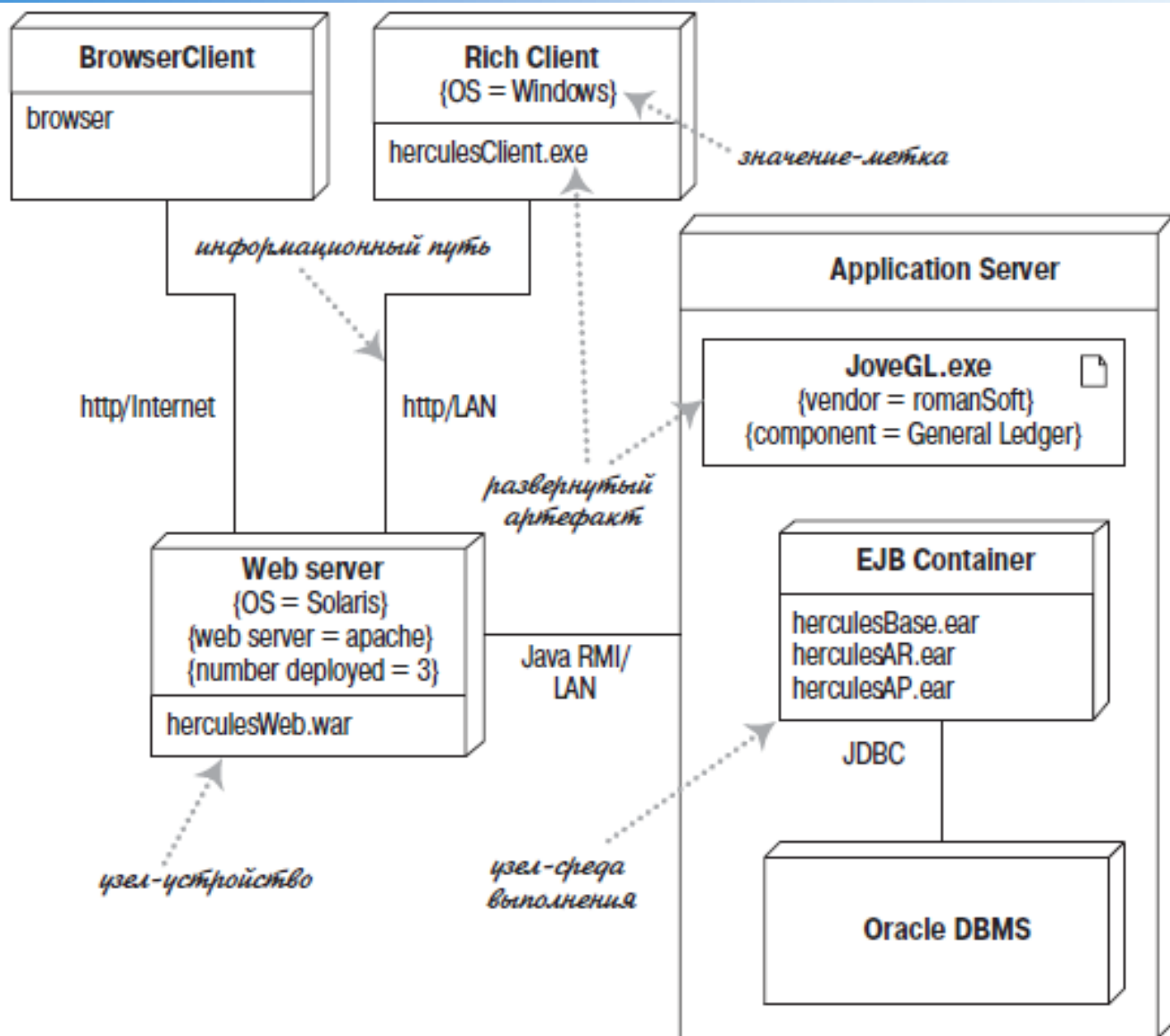


Отношение ассоциации между узлами на диаграмме развертывания

Отношение ассоциации между узлами означает возможность обмена сообщениями. Применительно к вычислительным сетям, состоящим из узлов, ассоциация означает наличие канала связи. Если нужно указать дополнительную информацию о свойствах канала, то это можно сделать, используя общие механизмы: стереотипы



Пример диаграммы развертывания



Пример диаграммы развертывания

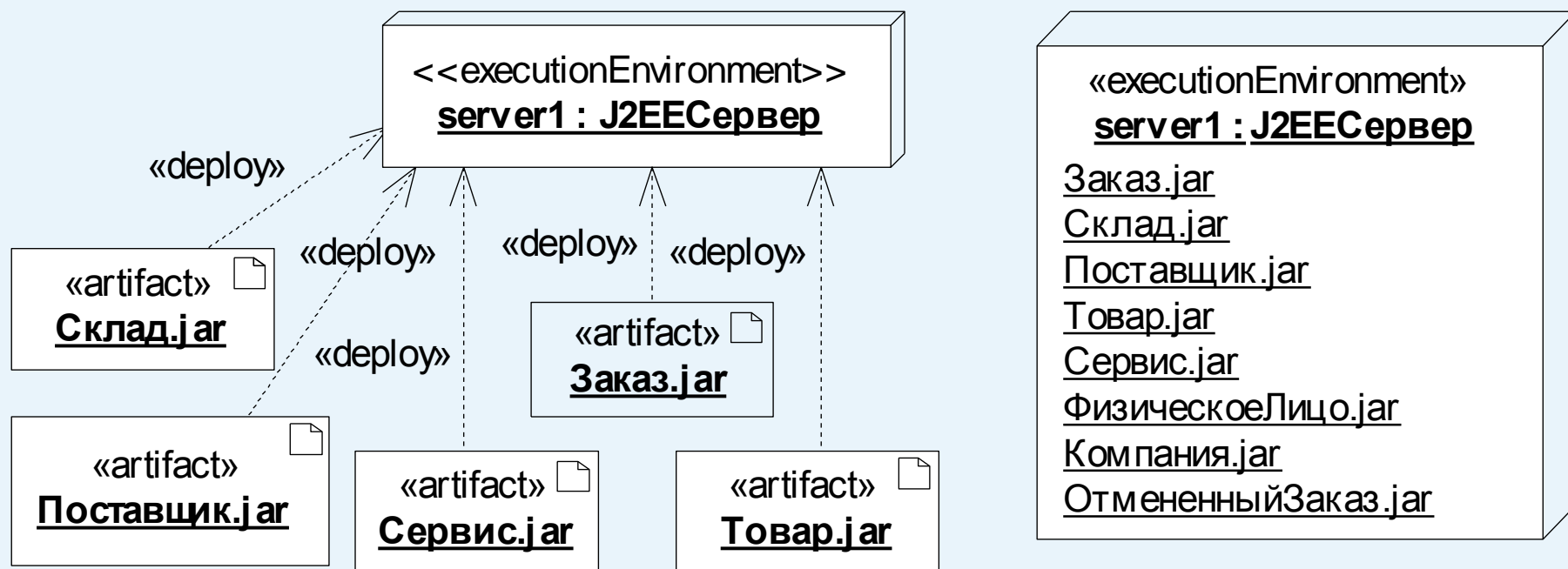


ДИАГРАММА ПАКЕТОВ

***предназначена для представления
размещения элементов модели в пакетах и
спецификации зависимостей между пакетами
и их элементами***

***единственное средство, позволяющее
управлять сложностью самой модели***

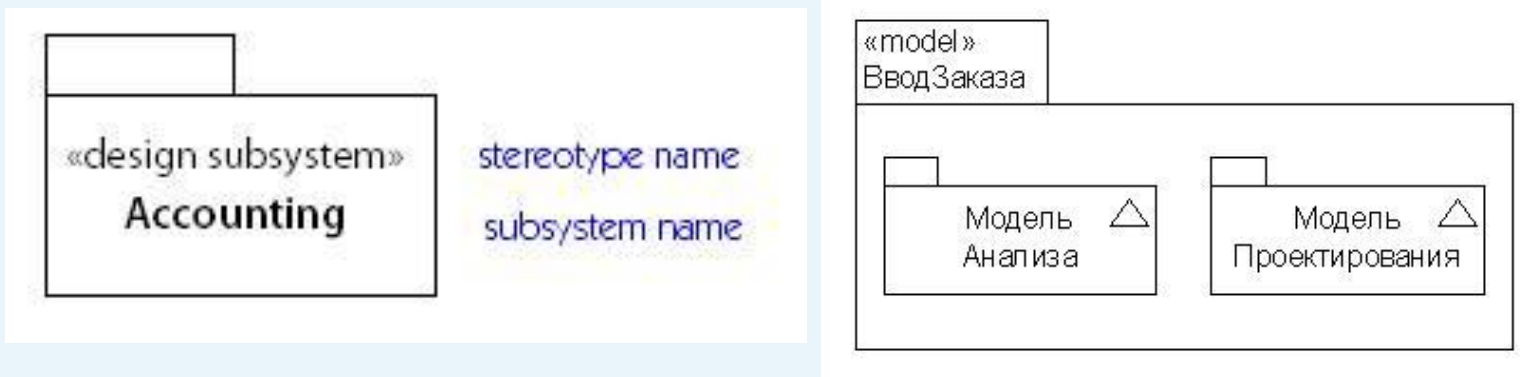


Диаграмма пакетов: основные понятия

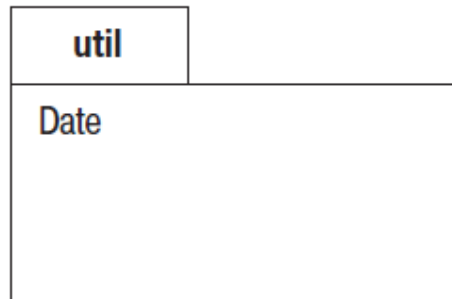
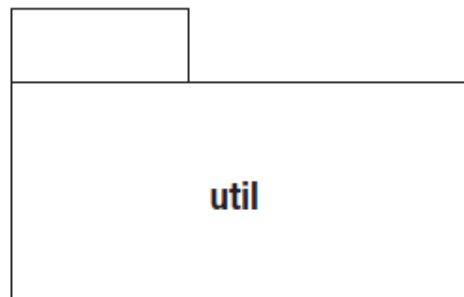
Пакет (package) – элемент модели, используемый для группировки других элементов модели.

Используется для представления моделей подсистем в виде пакетов.

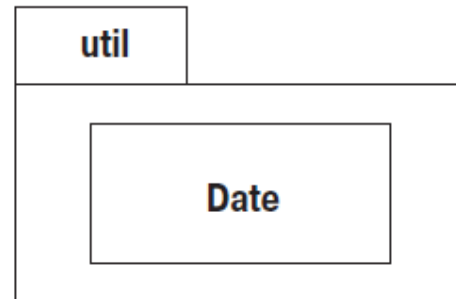
В терминах программирования пакеты в UML соответствуют таким группирующим конструкциям, как пакеты в Java и пространства имен в C++ и .NET.



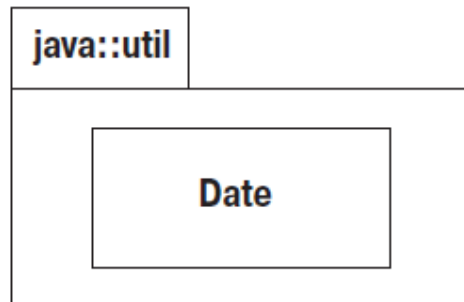
Способы изображения пакетов на диаграммах



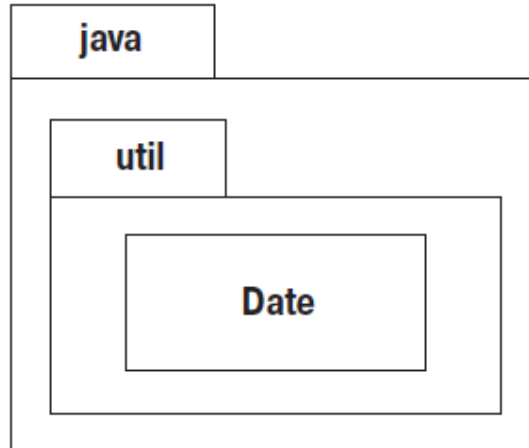
Содержимое, перечисленное
в прямоугольнике



Содержимое в виде диаграммы
в прямоугольнике



Полностью определенное
имя пакета



Вложенные пакеты



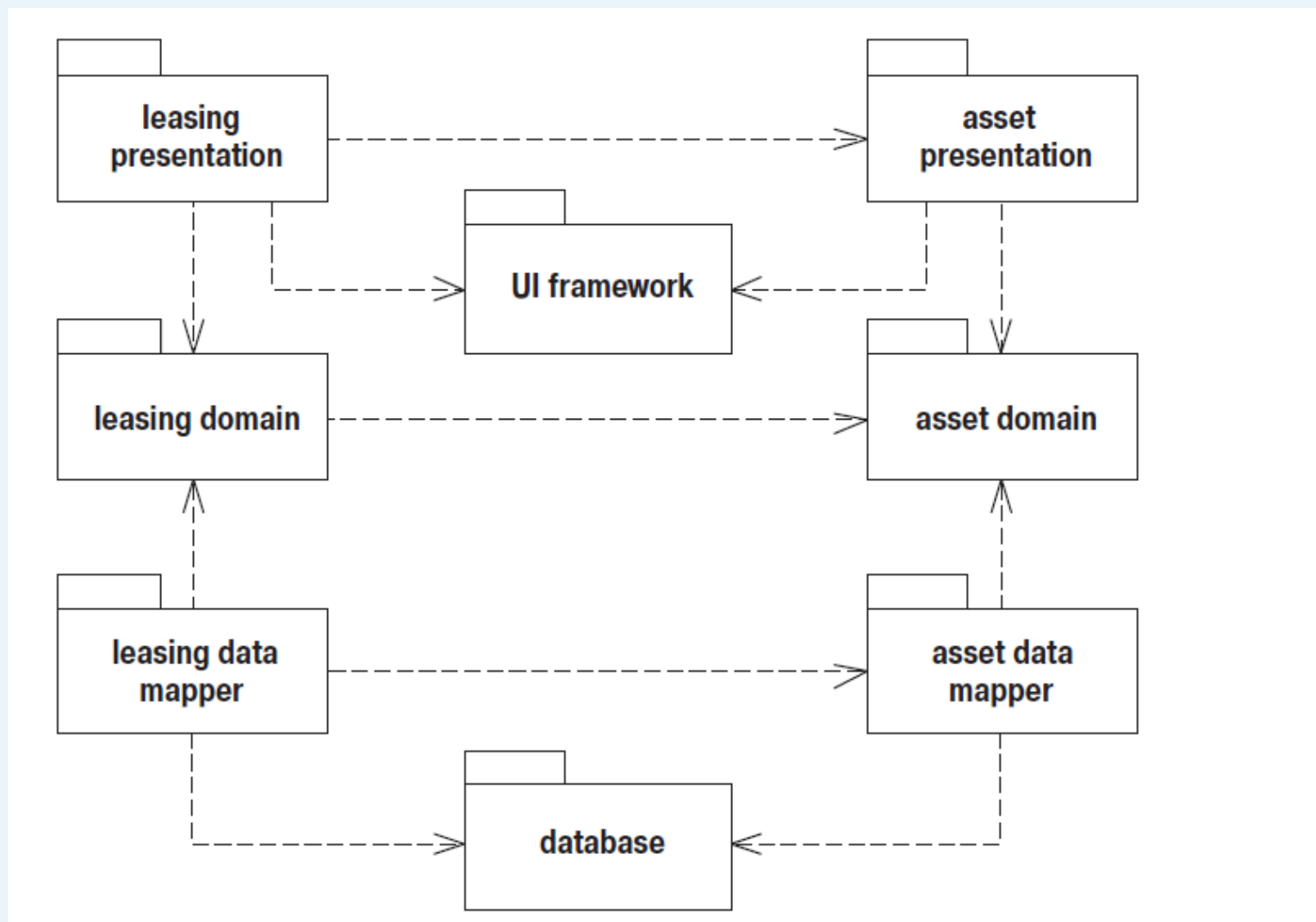
Полностью определенное
имя класса

Отношение зависимости между пакетами

Модель трехуровневой архитектуры доступа к удаленной базе данных



Пример диаграммы пакетов для промышленного приложения



Импорт пакетов

- Направленное отношение между пакетами, при котором члены одного пакета могут быть добавлены в пространство имен другого пакета
- Импорт пакета позволяет ссылаться только на общедоступные члены импортируемого пакета в другом пространстве имен, используя при этом неквалифицированные имена
- Концептуально импорт пакета эквивалентен импорту элемента для каждого индивидуального элемента импортируемого пространства имен, за исключением случая, когда уже существует отдельно определенный импорт элемента



Примеры импорта и доступа пакетов

