Государственный университет – Высшая школа экономики

Факультет Бизнес-Информатики

Кафедра Архитектуры программных систем

Объектно-ориентированный анализ и программирование на языке C#

Дополнение к 12-й лекции. Раздел 2. Введение в **UML**

Проф. Забудский Е.И.

Москва 2008

Лекция 12. Темы 5 и 6. Дополнение

2. Введение в Unified Modeling Language (UML).

Подробную информацию о **UML** можно получить на **Web**-странице некоммерческой организации **Unified Modeling Language (UML)** (**www.omg.org**) по адресу **www.omg.org/uml** .

Уважаемые студенты!

Основная цель, которую необходимо достигнуть в результате изучения дисциплины Объектно-ориентированный анализ и программирование – научиться разрабатывать компьютерные модели реальных и концептуальных систем соответствующих направлению Бизнес-информатика.

Необходимым условием усвоения дисциплины является ВАША **самостоятельная** работа

Советую Вам все материалы, подготовленные мной к лекциям и практическим занятиям, распечатать и прорабатывать их! Приведенные C#-программы реализовать в среде MS VS .NET 2005 и разобраться в них.

// **КОММЕНТАРИЙ**. На сайте http://www.firststeps.ru/ “Первые шаги” представлено много интересных обучающих материалов по различным интегрированным средам и языкам программирования, в том числе представлены **C#** и платформа **.NET** (step by step).

Содержание

Диаграммы UML

(см. примеры использования диаграмм и др. на **стр. 6** Web-page **[15] :**
6. Компьютерные модели реальных и концептуальных систем, разработанные в соответствии с парадигмой ООП **)**

1. Классификация диаграмм, принятые обозначения

UML 2 описывает **13** официальных типов диаграмм, перечисленных в табл. 1, классификация которых приведена на рис. 1. Диаграммы UML не являются центральной составляющей языка. Поэтому диаграммы определены не очень строго. Часто вполне допустимо присутствие элементов диаграммы одного типа в другой диаграмме. Стандарт UML указывает, что определенные элементы обычно рисуются в диаграммах соответствующего типа, но это не догма.

Таблица 1. Официальные типы диаграмм UML

##	Диаграмма	Цель	Глава книги [13]
1	**Деятельности**, с. 24	**Процедурное и параллельное поведение**	11
2	**Классов**, с. 14	**Классы, свойства и отношения**	3, 5
3	**Взаимодействия**, с. 22	**Взаимодействие между объектами; акцент на связях**	12
4	**Компонентов**, с. 25	**Структура и взаимосвязи между компонентами**	14
5	**Составных структур**	**Декомпозиция класса во время выполнения**	13
6	**Развертывания**, с. 27	**Развертывание артефактов в узлы**	8
7	**Обзора взаимодействий**	**Комбинация диаграммы последовательности** (с. 24) **и диаграммы деятельности** (с. 24)	16
8	**Объектов**, с. 16	**Вариант конфигурации экземпляров**	6
9	**Пакетов**, с. 29	**Иерархическая структура времени компиляции**	7
10	**Последовательности** [1], с. 20	**Взаимодействие между объектами; акцент на последовательности**	4 CRC
11	**Конечных автоматов**, с. 19	**Как события изменяют объект в течение его жизни**	10
12	**Временная**, с. 31	**Взаимодействие между объектами; акцент на синхронизации**	17
13	**Прецедентов**, с. 17	**Как пользователи взаимодействуют с системой**	9

[1] **используются также CRC-карточки (Class-Responsibility-Collaboration, класс-ответственность-взаимодействие, см. рис. 13)**

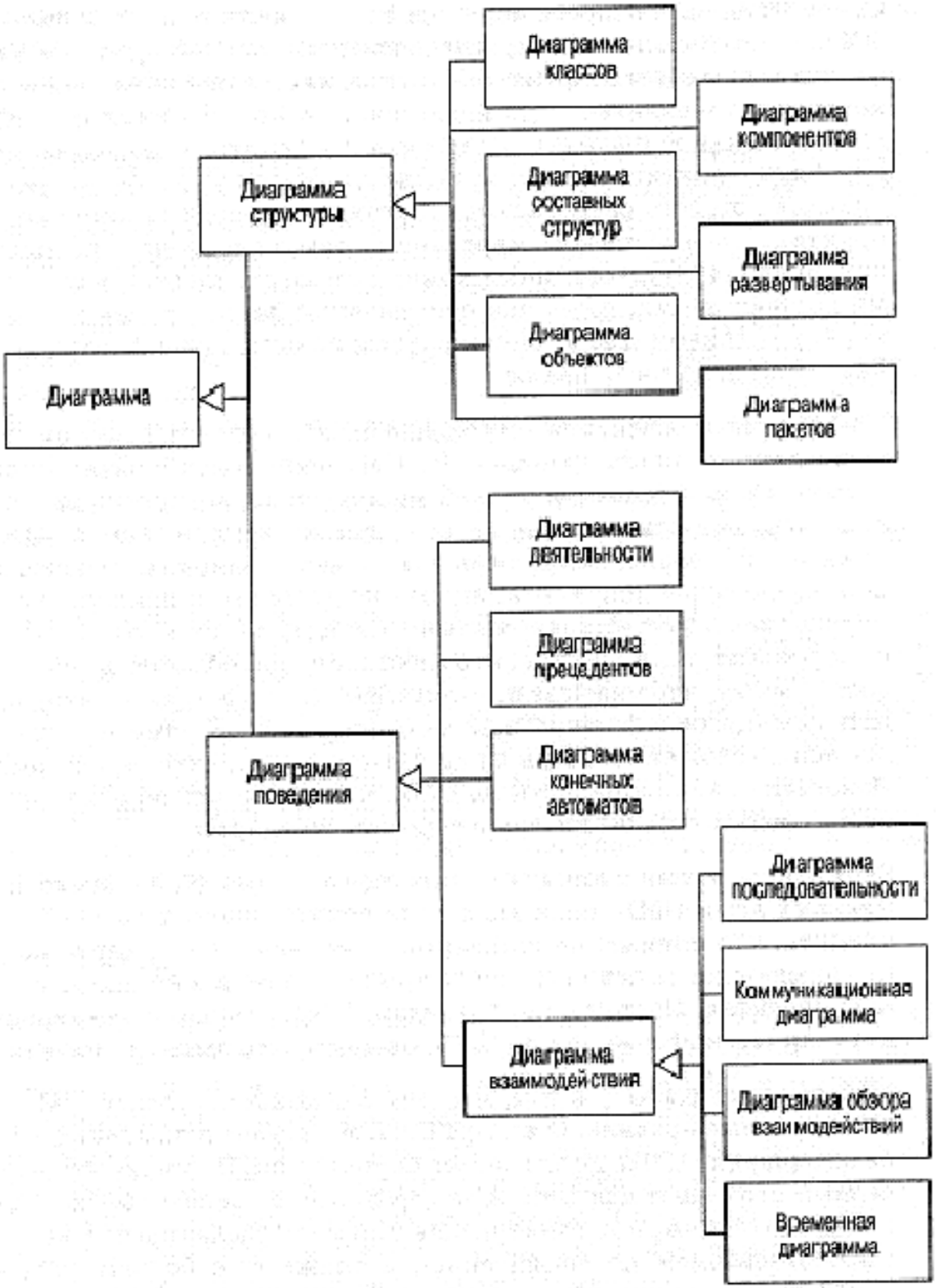

Рис. 1. Классификация **типов диаграмм UML**

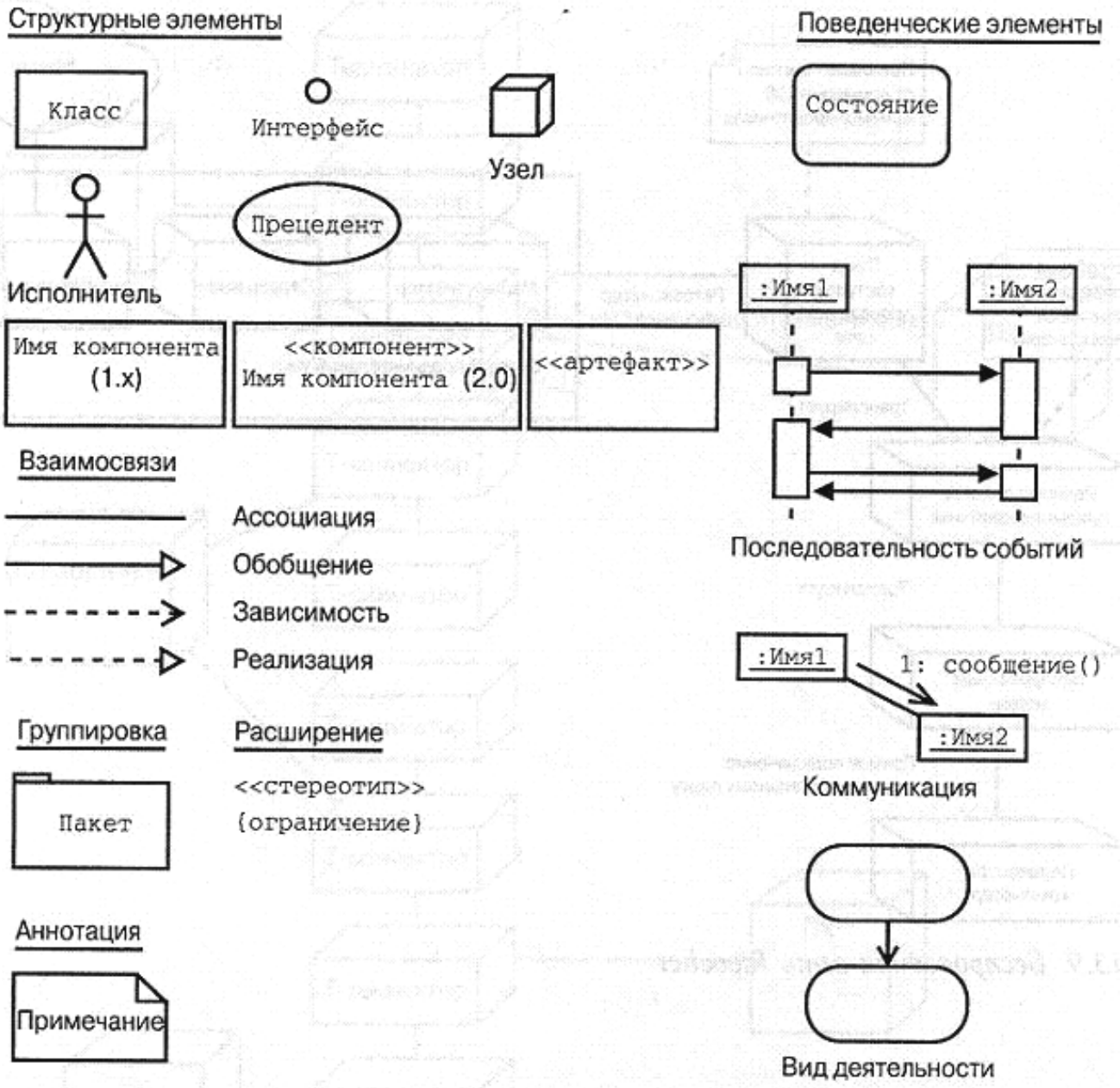

Рис. 2. Элементы **UML**

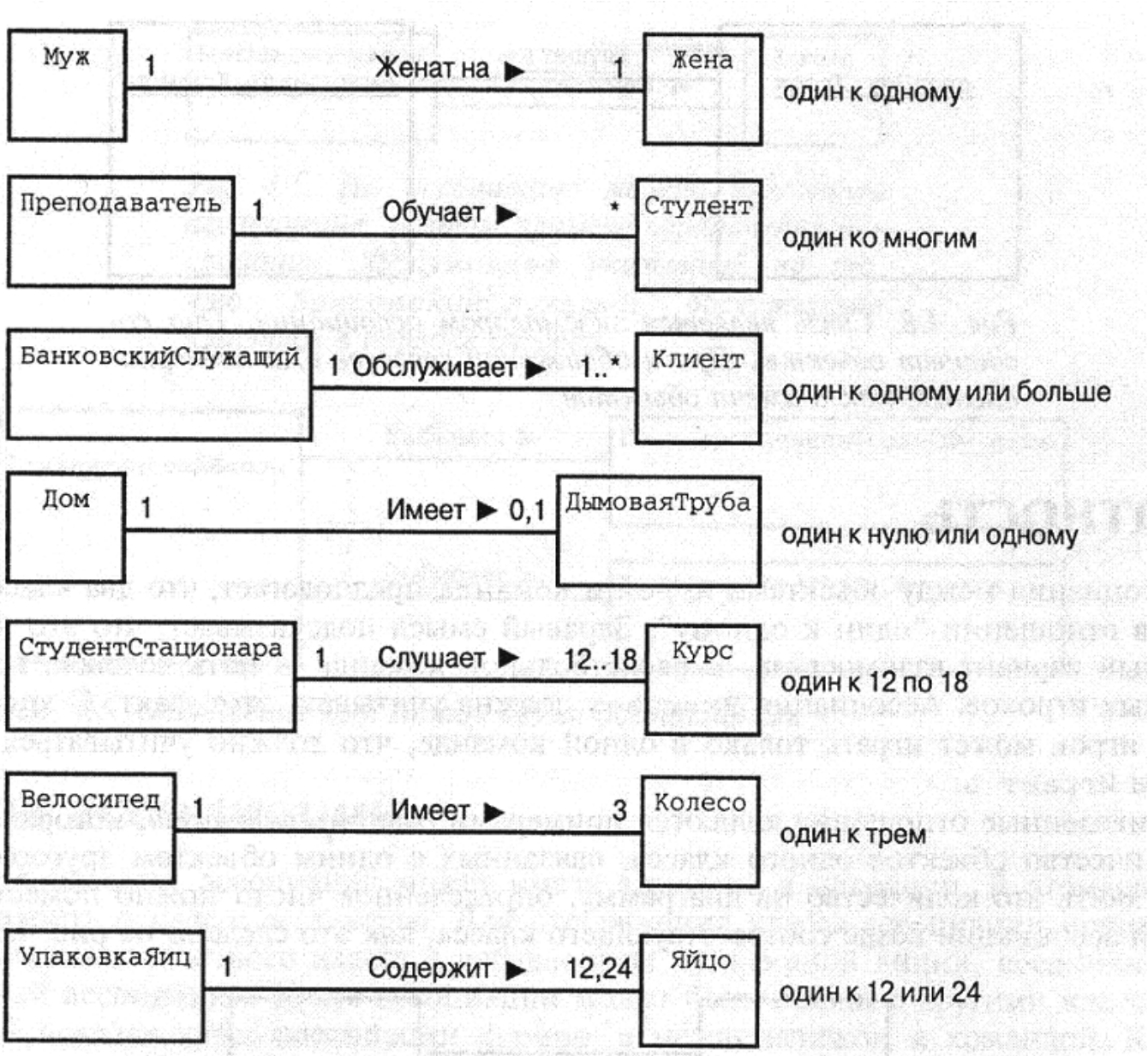

Рис. 3. Возможные значения **кратности** и их представления в **UML**

2. Изображение ассоциаций на диаграммах классов

/ см. также примеры ассоциации – Практ. зан. 3, с. 26…29. Лекции 12 и 13 - с. 23…32 /

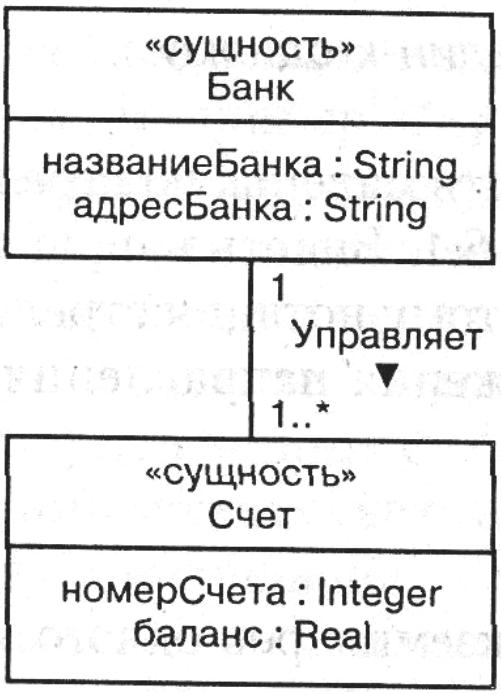

Рис. 4. Пример ассоциации **один-ко-многим**

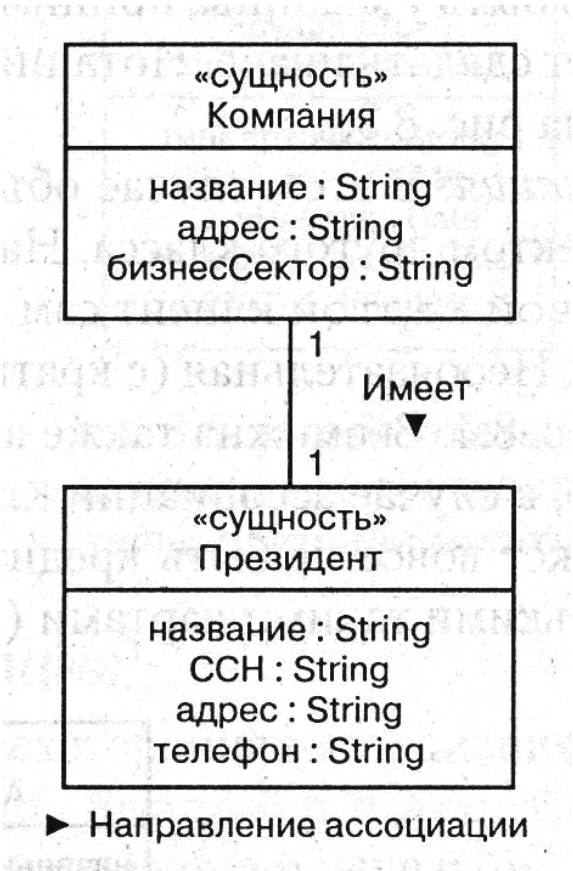

Рис. 5. Пример ассоциации **один-к-одному**

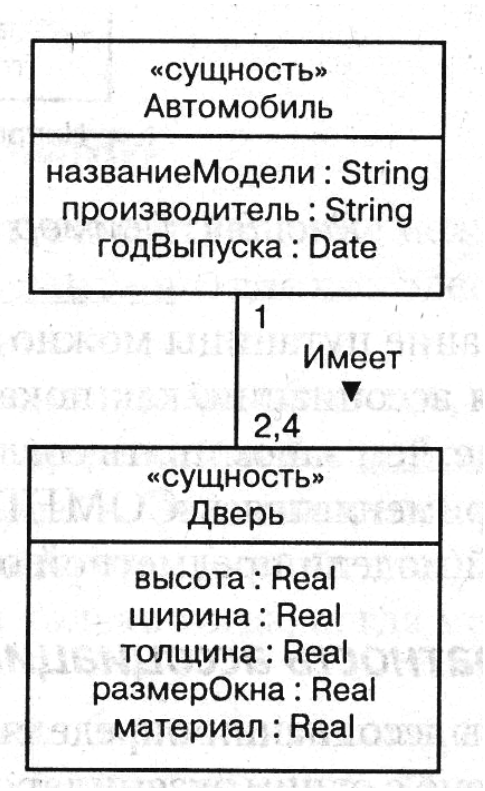

Рис. 6. Пример ассоциации **с заданной кратностью**

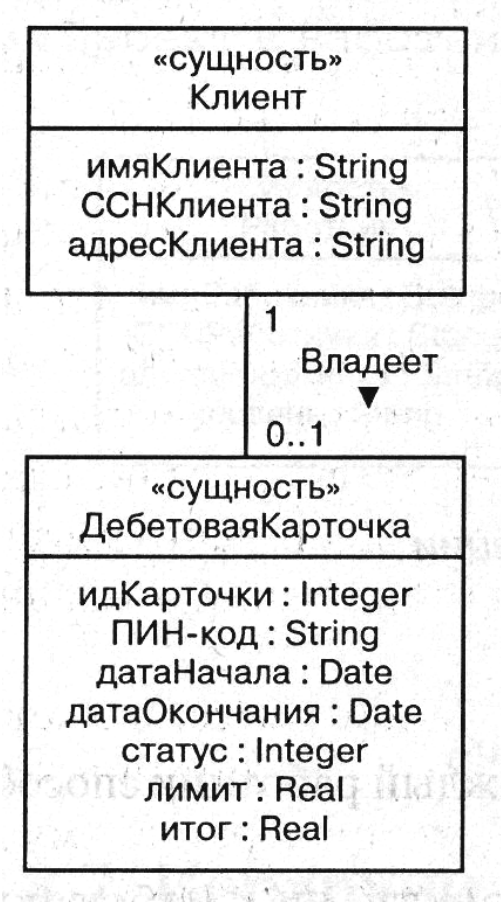

Рис. 7. Необязательная ассоциация (**с кратностью нуль** или **один**)

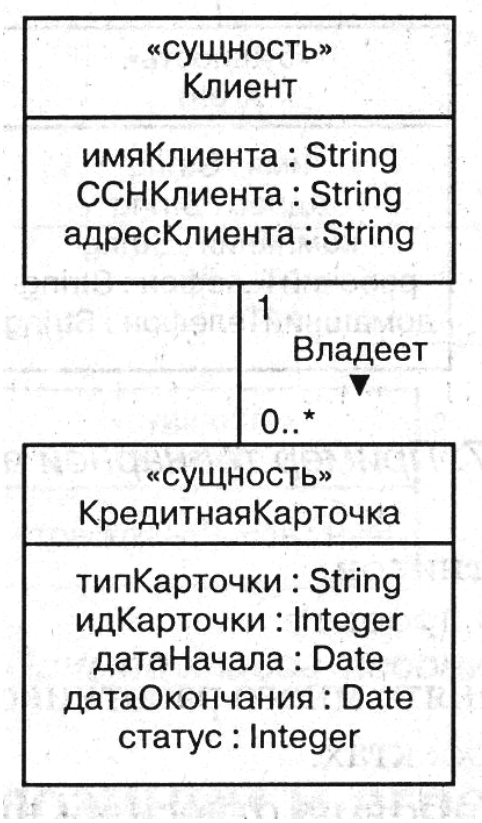

Рис. 8. Необязательная ассоциация (**с кратностью нуль, один** или **много**)

Рис. 9. Пример ассоциации **многие-ко-многим**

/**См. пример ассоциации** - Система Ипподромные Состязания (**Derby**) **с. 6** [WEB-page, p. 6]**/**

3. Иерархии классов

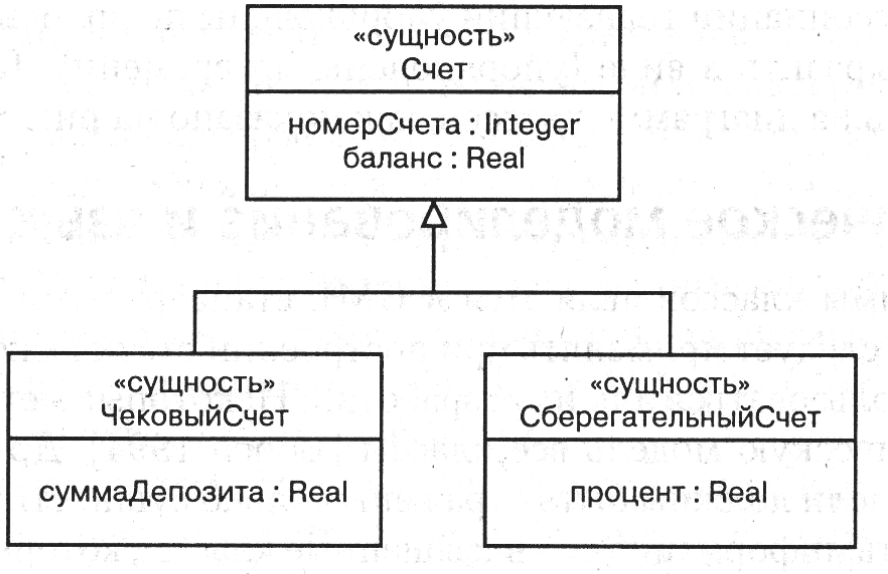

Рис. 10. Пример иерархии **обобщения/специализации**

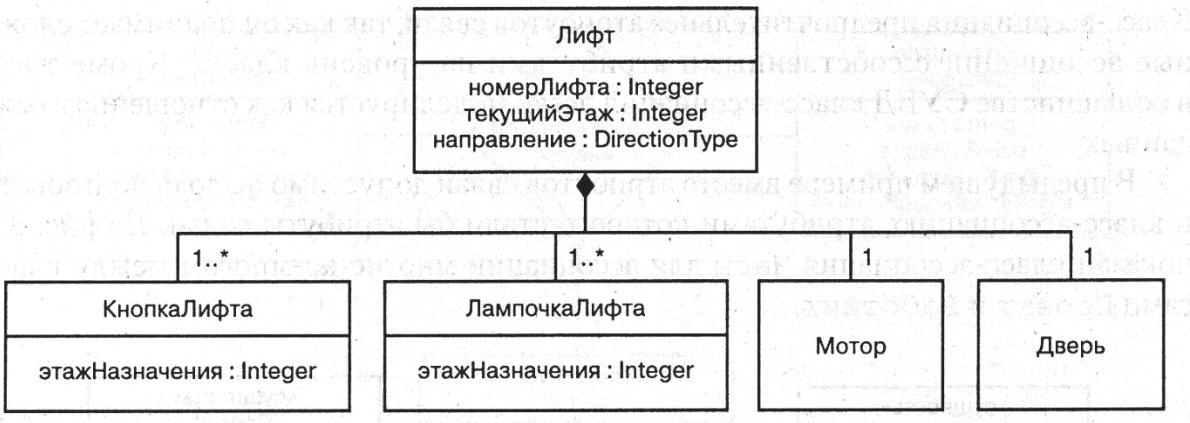

Рис. 11. Пример иерархии **композиция**

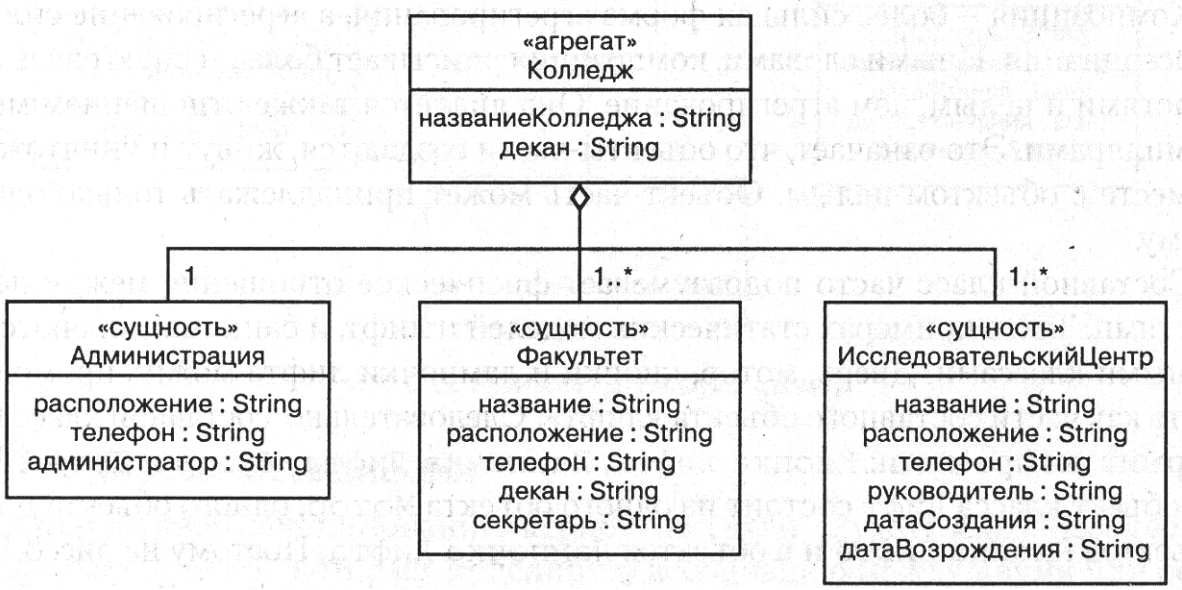

Рис. 12. Пример иерархии **агрегация**

/ см. иерархию агрегации, композиции и ассоциации – Практ. зан. 3, с. 26…29. Лекции 12 и 13 - с. 23…32 /

4. CRC-карточки

Важным моментом в **CRC**-методике является определение ответственностей. Ответственность (**responsibility**) - это краткое описание того, что объект должен делать: операция, которую выполняет объект, некоторый объем знаний, который объект поддерживает, или какие-либо важные решения, принимаемые объектом. Идея состоит в том, чтобы вы могли взять любой класс и сформулировать его разумно ограниченные обязанности. Такой образ действия поможет вам яснее представить себе архитектуру классов.

Вторая буква «**C**» (в **CRC**) означает взаимодействие (**collaboration**): другие классы, с которыми должен работать рассматриваемый класс. Это дает вам некоторое представление о связях между классами, но все еще на высоком уровне.

Одно из главных преимуществ CRC-карточек состоит в том, что они способствуют живому обсуждению проектов в среде разработчиков. Если в процессе работы над шаблоном поведения вы хотите посмотреть, как классы его реализуют, то вычерчивание диаграмм взаимодействия, описанных в разделах **9, 10, 11**, может занять слишком много времени. Обычно требуется просмотреть варианты, а рисование и стирание вариантов на диаграммах может быть очень утомительным. С помощью **CRC**-карточек разработчики моделируют взаимодействие, поднимая карточки и передавая их по кругу. Это позволяет быстро просчитать варианты.

В процессе работы вы создаете представление об ответственностях и записываете их на карточки. Размышления об ответственностях важны, поскольку рассеивают представление о классах как о бессловесных хранителях данных и помогают членам команды лучше понять поведение каждого класса на высшем уровне. Ответственность может соответствовать операции (методу), атрибуту или, что более точно, неограниченной группе атрибутов и операций.

Рис. 13. Пример **CRC**-карточки

/см. использование карточек CRC - Компьютерная Игра (**Blackjack**) с. 14…16 [WEB-page, p. 6]/

(см. также в Лекции 14 с. 7…13).

5. Диаграмма классов

Диаграмма классов 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. В **UML** термин функциональность (**feature**) применяется в качестве основного термина, описывающего **и свойства, и операции класса.**

На рис. 14 изображена типичная модель класса, понятная каждому, кто имел дело с обработкой заказов клиентов. Прямоугольники на диаграмме представляют классы и разделены на три части: **имя класса** (жирный шрифт), его **атрибуты** и его **операции (методы)**. На рис. 14 также показаны два вида связей между классами: **ассоциации** и **обобщения** (см. также рис. 10…12).

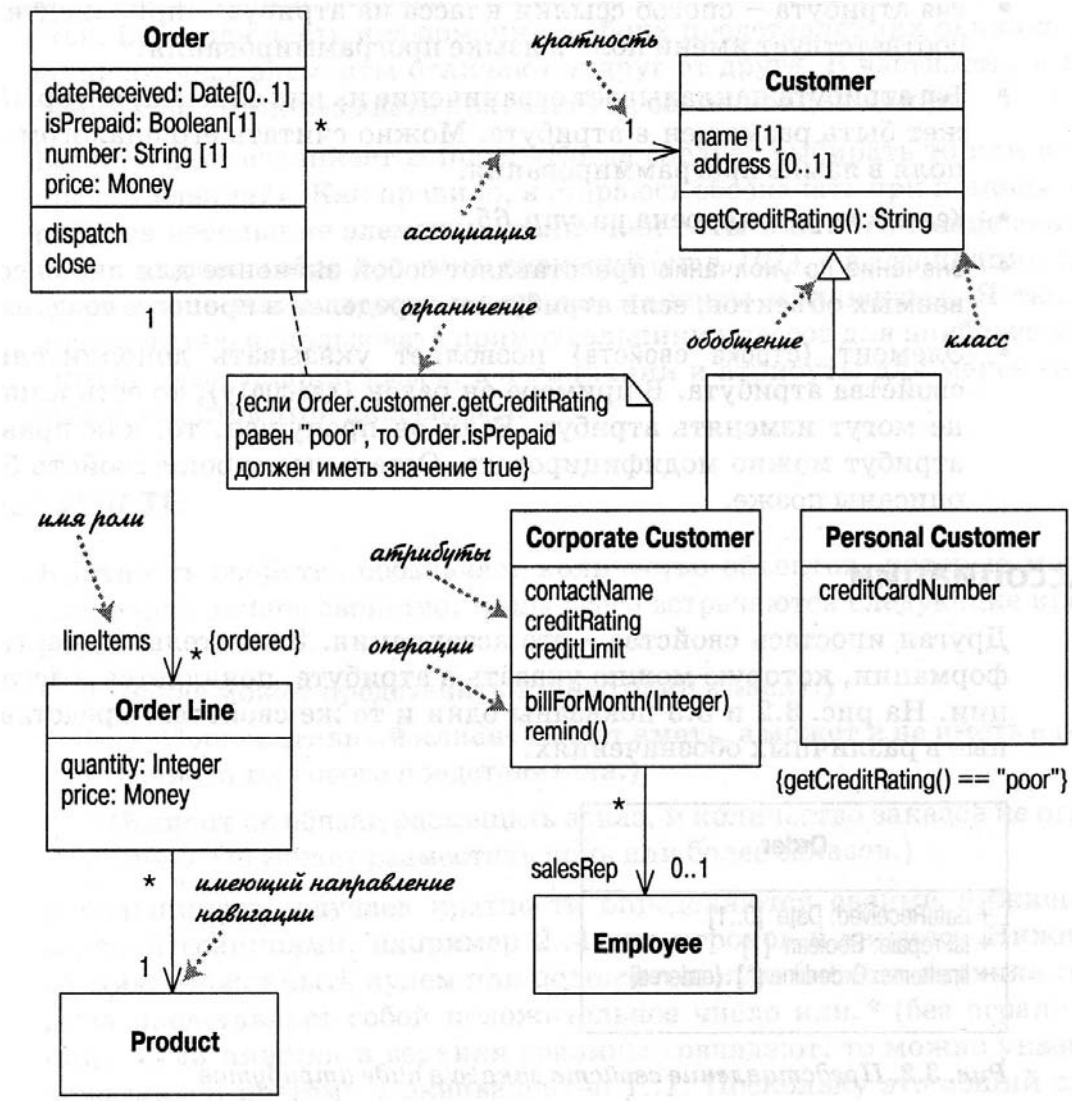

Рис. 14. Простая диаграмма классов

/ **см. также диаграммы классов:** - Компьютерная Игра (**Blackjack**) с. 16

- Система Ипподромные Состязания (**Derby**) **с. 6**

- Система Домовладелец (**LandLord**)) с. 17 [WEB-page, p. 6]/

5.1. Кратность

Кратность обозначает количество объектов, которые могут заполнять данное свойство. Чаще всего встречаются следующие кратности (**см. рис. 3**):

- **1** (Заказ может представить только один клиент.)
- **0..1** (Корпоративный клиент может иметь, а может и не иметь единственного торгового представителя.)
- * (Клиент не обязан размещать заказ, и количество заказов не ограничено. Он может разместить ноль или более заказов.)

В большинстве случаев кратности определяются своими нижней и верхней границами, например **2..4** для игроков в карты. Нижняя граница может быть **нулем** или **положительным числом**, верхняя граница представляет собой **положительное число** или * (без ограничений). Если нижняя и верхняя границы совпадают, то можно указать одно число; поэтому **1 эквивалентно 1..1**. Поскольку это общий случай, * является сокращением **0..***.

При рассмотрении атрибутов могут встретиться термины, имеющие отношение к кратности.

- **Optional** (необязательный) предполагает нулевую нижнюю границу.
- **Mandatory** (обязательный) подразумевает, что нижняя граница равна или больше **1**.
- **Single-valued** (однозначный) - для такого атрибута верхняя граница равна **1.**
- **Multivalued** (многозначный) имеет в виду, что верхняя граница больше 1; обычно *.

5.2. Зависимость

Считается, что между двумя элементами существует зависимость (**dependency**), если изменения в определении одного элемента (**сервера**) могут вызвать изменения в другом элементе (**клиенте**). В случае классов зависимости появляются по разным причинам (**NB**):

1. **один** класс посылает сообщение **другому** классу;
2. один класс владеет другим классом **как частью своих данных**;
3. один класс использует другой класс **в качестве параметра** операции (метода) **/ см., например, Лекция 4 Листинг 6, строка 64 /.**

Если класс изменяет свой интерфейс, то сообщения, посылаемые этому классу, могут стать недействительными (**NB**).

По мере роста систем необходимо все более и более беспокоиться об управлении зависимостями. Если зависимости выходят из-под контроля, то каждое изменение в системе оказывает действие, нарастающее волнообразно по мере увеличения количества изменений. Чем больше волна, тем труднее что-нибудь изменить.

UML позволяет изобразить зависимости между элементами всех типов. Зависимости можно использовать всякий раз, когда надо показать, как изменения в одном элементе могут повлиять на другие элементы.

На рис. 15 показаны зависимости, которые можно обнаружить в многоуровневом приложении. Класс **Benefits Window** (**Окно льгот**) - это пользовательский интерфейс, или класс представления, зависящий от класса **Employee** (**Сотрудник**). Класс **Employee** - это объект предметной области, который представляет основное поведение системы, в данном случае бизнес-правила. Это означает, что если класс **Employee** изменяет свой интерфейс, то, возможно, и класс **Benefits Window** также должен измениться.

/ **см. также иллюстрацию зависимости между классами** - Лекции 12 и 13 - **с. 22, 23**/

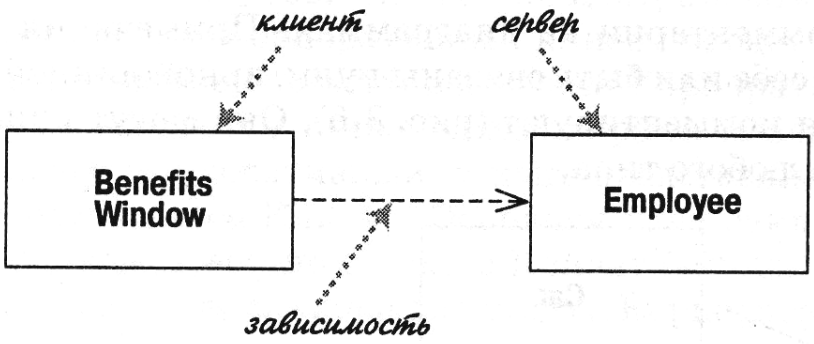

Рис. 15. Пример **зависимостей**

Здесь важно то, что зависимость имеет только одно направление и идет от класса представления к классу предметной области. Таким образом, есть возможность свободно изменять класс **Benefits Window**, не оказывая влияния на объект **Employee** или другие объекты предметной области. Строгое разделение логики представления и логики предметной области, когда представление зависит от предметной области, но не наоборот, - это ценное правило, которому необходимо следовать (см. **Лекция 15, раздел 2.** ОО подход к созданию **UI; реализация шаблона M-V-C**).

5.3. Статические (static) и динамические классы

С атрибутами и операциями (методами) связано понятие статических и динамических классов. Для динамического класса (**instance scope**) каждый экземпляр имеет собственное значение атрибутов и операций (методов). Все экземпляры статического класса (**classilier scope**) имеют общее значение для каждого атрибута или операции. В последнем случае имена атрибутов и операций подчеркиваются. Статический класс используется в том случае, когда заданная группа экземпляров должна совместно использовать значения **закрытого** атрибута. Динамические классы встречаются гораздо чаще.

6. Диаграммы объектов

Диаграмма классов содержит общую **декларативную** информацию о свойствах класса и его атрибутах, а также о связи данного класса с другими. **Диаграмма объектов** содержит более конкретную информацию об экземплярах класса и их связях с другими экземплярами во времени.

В качестве примера рассмотрим фрагмент шахматной партии, показанный на рис. 16

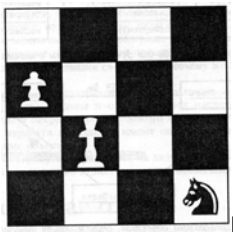
мм

Рис. 16. Фрагмент шахматной партии

Диаграмма классов помогает понять общие правила игры в шахматы (в частности, если в ней описываются операции **ходКонем(), ходФерзем()** и **ходПешкой(),** она не проясняет конкретную позицию, показанную на рис. 16. В этом поможет диаграмма объектов. На рис. 17 показана модель позиции, изображенной на рис. 16, в которой все связи между объектами имеют имена.

Рис. 17. **Диаграмма объектов**, отражающая фрагмент шахматной партии, показанной на рис. 16

Диаграмма классов содержит **общую** информацию о классах.

Для изображения **конкретных** экземпляров классов и их взаимосвязей используются **диаграммы объектов.**

7. Диаграммы прецедентов

Моделирование системы с точки зрения пользователя — это задача прецедентов. Рассмотрим прецеденты и их функции. Прецеденты визуализируются с помощью диаграмм.

Прецедент — это конструкция, помогающая аналитику определить способ использования системы. Набор прецедентов описывает систему в терминах действий, выполняемых пользователем.

Прецедент — это **набор сценариев использования системы**. Каждый сценарий описывает последовательность действий. Каждая последовательность действий инициируется пользователем, другой системой, аппаратным средством или в какой-либо момент времени. **Сущности, инициирующие сценарии, называются исполнителями (actor)**. Результат этих действий должен быть полезен исполнителю.

7.1. Зачем нужны прецеденты

Прецедент заставляет потенциальных пользователей думать о системе со своих позиций. Пользователям не всегда легко выразить свои требования к системе. Поскольку обычно процесс разработки системы представлял собой бессистемную процедуру с кратким анализом, то пользователей несколько удивлял интерес к их мнению.

Основная идея заключается в том, чтобы привлечь пользователей к разработке системы на ранних стадиях ОО **анализа** (**см. в** Лекции 12 и 13 - **с. 33… 51**) и ОО **проектирования (см.** Лекцию 14). Это повышает вероятность создания полезной системы, позволяет сконцентрироваться на мнении людей, а не на компьютерных понятиях, с которыми не могут работать реальные пользователи.

Прецедент — это конструкция, позволяющая описать систему с точки зрения потенциальных пользователей. Прецедент представляет собой набор сценариев, инициируемых **исполнителями** (людьми, аппаратными средствами, другими системами или интервалами времени). Результат прецедента должен быть полезен исполнителю, инициировавшему этот прецедент, либо какому-то другому исполнителю.

Прецеденты можно использовать повторно. Один из способов предполагает включение шагов одного прецедента в последовательность действий другого. Другой путь сводится к созданию нового прецедента путем добавления нескольких шагов к существующему — расширению прецедента.

Опрос пользователей — лучший способ определения прецедентов. При этом важно выявить

предусловия **(см. далее разд. 7.3)** для инициализации прецедента и **постусловия** **(см. разд. 7.2)**, реализуемые в результате выполнения прецедента.

Опрос пользователей осуществляется после составления перечня классов-кандидатов. Это обеспечивает базовую терминологию для общения с пользователями. Желательно опрашивать группы пользователей. Задачей этого этапа является формирование списка возможных прецедентов и всех возможных исполнителей.

Описание прецедентов — это мощный инструмент, позволяющий аналитику понять принципы работы системы и сформулировать требования к системе с точки зрения ее будущих пользователей.

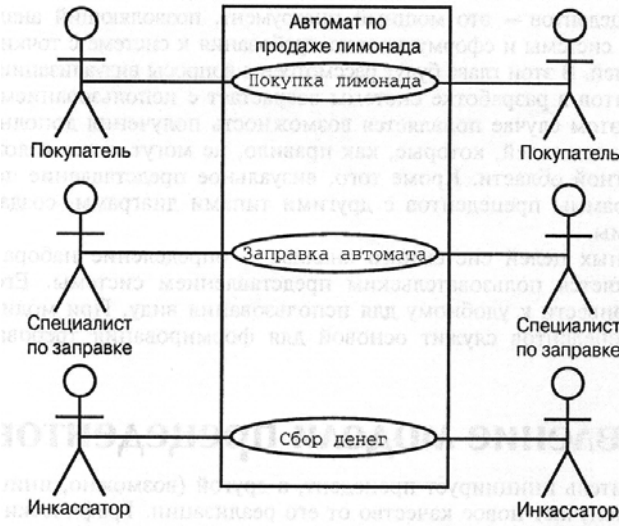

Рис. 18. **Диаграмма прецедентов** для автомата по продаже лимонада

Роль прецедентов в разработке системы возрастает с использованием **UML** для их визуализации. В этом случае появляется возможность получения дополнительной информации от пользователей, которые, как правило, не могут четко изложить все свои знания о предметной области. Кроме того, визуальное представление позволяет комбинировать диаграммы прецедентов с другими типами диаграмм, создаваемыми при разработке системы.

Одна из главных целей системного анализа — определение набора прецедентов. Такой набор является **пользовательским** представлением системы. Его необходимо упорядочить и привести к удобному для использования виду. При модификации системы каталог прецедентов служит основой для формирования требований к новой версии системы.

Прецедент — мощное средство для формирования функциональных требований к системе. Диаграммы прецедентов — еще более мощный инструмент**:**

- они визуализируют прецеденты,
- служат основой для взаимодействия между аналитиками и пользователями,
- а также между аналитиками и клиентами.

На диаграмме **прецедент** обозначается **эллипсом**. **Исполнители** обозначаются упрощенной **фигуркой**. Линия ассоциации соединяет исполнителя с прецедентом. Обычно **прецеденты располагаются внутри прямоугольника**, отображающего границы системы.

Включение представляется линией **зависимости** со стереотипом <<включает». Два других типа взаимосвязи прецедентов — это **обобщение**, при котором один прецедент наследует значение и поведение другого, и **группирование**, позволяющее систематизировать набор прецедентов. Обозначение обобщения на диаграмме совпадает с обозначением наследования для классов. Группирование изображают с помощью пакетов.

Диаграммы прецедентов активно используются в процессе анализа. Сначала на основе общения с заказчиком строятся диаграммы классов, обеспечивающие основу для интервьюирования пользователей. В результате предлагается высокоуровневая диаграмма прецедентов, которая отражает функциональные требования к системе. Для создания моделей прецедентов нужно детально изучить каждый высокоуровневый прецедент. На основе полученных моделей прецедентов выполняется проектирование и разработка системы.

ОО подход и **прецеденты** —два краеугольных камня **UML** (**см. в** Лекции 12 и 13 - **с. 33… 51**).

7.2. Постусловие - это высказывание относительно того, как будет выглядеть окружающий мир **после** выполнения операции (метода). Например, если мы определяем для числа операцию «извлечь квадратный корень», постусловие может принимать форму **input = result * result**, где **result** является выходом, а **input** - исходное значение числа. Постусловие - это хороший способ выразить, что должно быть сделано, не говоря при этом, как это сделать. Другими словами, постусловия позволяют отделить **интерфейс** метода от его **реализации**.

7.3. Предусловие - это высказывание относительно того, как должен выглядеть окружающий мир **до** выполнения операции (метода). Для операции «извлечь квадратный корень» можно определить предусловие **input >= 0**. Такое предусловие утверждает, что применение операции «извлечь квадратный корень» для отрицательного числа является ошибочным и последствия такого применения не определены.

На первый взгляд эта идея кажется неудачной, поскольку нам придется выполнить некоторые дополнительные проверки, чтобы убедиться в корректности выполнения операции «извлечь квадратный корень». При этом возникает важный вопрос: на кого ляжет ответственность за выполнение этой проверки.

Предусловие явным образом устанавливает, что за подобную проверку отвечает вызывающий объект. Без такого явного указания обязанностей мы можем получить либо недостаточный уровень проверки (когда каждая из сторон предполагает, что ответственность несет другая сторона), либо чрезмерную проверку (когда она будет выполняться обеими сторонами). Излишняя проверка тоже плоха, поскольку это влечет за собой дублирование кода проверки, что, в свою очередь, может существенно увеличить сложность программы.

/**см. иллюстрацию прецедентов:** - Компьютерная Игра (**Blackjack**) **с. 7…11;**
- Система Домовладелец (**LandLord**) **с. 6…13** [WEB-page, p. 6]/

8. Диаграмма состояний (конечных автоматов)

Диаграмма состояний UML представляет:

- состояния объекта,
- переходы между ними,
- а также показывает начальное и конечное состояние объекта.

Диаграмму состояний иногда называют **автоматом состояний**, поскольку она отражает все эти изменения.

Диаграмма состояний по своей сути значительно отличается **от диаграммы классов, диаграммы объектов** или **диаграммы прецедентов**, и это отличие очень важно. Эти диаграммы моделируют поведение всей **системы** или группы **классов**, объектов или прецедентов. Диаграмма состояний показывает состояния одного объекта.

8.1 Основные обозначения

На рис. 19 состояния показаны в виде прямоугольников **со скругленными углами**, а **переходы** между состояниями — **сплошными линиями со стрелкой**. Она указывает на состояние, в на-

правлении которого происходит переход. Закрашенный круг символизирует начальную точку, а "глазок" — конечную.

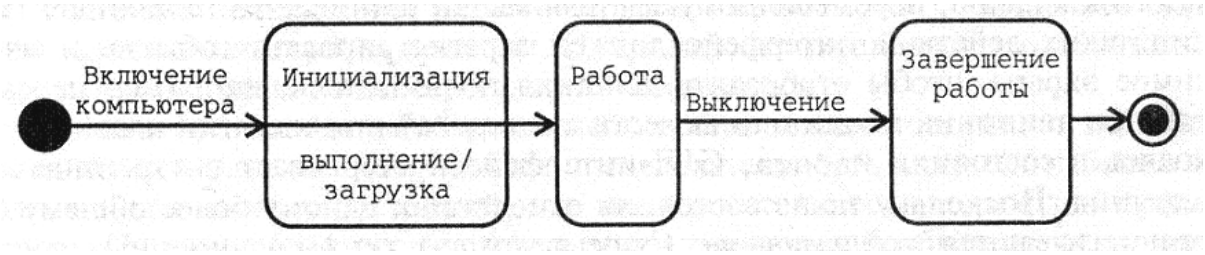

Рис. 19. **Диаграмма состояний GUI** с указанием переключающих событий, действия и безусловных переходов

8.2. Зачем нужны диаграммы состояний

На диаграмме состояний отображаются все переходы между состояниями одного объекта системы.

Диаграммы состояния используются **для исследования поведения** объектов в системе. **Диаграмма классов и диаграмма объектов** отображают только **статическое** состояние системы. На них представлены иерархии и ассоциации, и из них можно узнать о возможном перечне действий системы, но ничего нельзя узнать о деталях **динамического поведения**.

Диаграммы состояний дают полную информацию о желаемом поведении. Ясное представление о поведении объекта повышает вероятность того, что группа разработчиков создаст систему, удовлетворяющую выдвинутым требованиям.

8.3. С чего начинать построение диаграмм состояний?

Процесс создания диаграмм состояний напоминает создание диаграммы классов или прецедентов. В диаграмме классов **сначала перечисляются все классы**, **а затем определяются ассоциации**. В диаграмме состояний **сначала отображаются все состояния объекта**, **а затем строятся переходы.**

/см. диаграммы состояний: - Система Высотные Лифты Здания (**Elevator**) **с. 9, 10**;
Система Ипподромные Состязания (**Derby**) **с. 6…8** [WEB-page, p. 6]**/**

9. Диаграммы последовательностей

Диаграммы **состояний** отражают изменение состояний одного объекта.

Однако **UML** позволяет показать взаимодействие объектов друг с другом, причем включается важное измерение: **время**. Основная идея сводится к тому, что взаимодействие объектов происходит в заданной последовательности, и для выполнения этой последовательности от начала до конца требуется время.

9.1. Что такое диаграмма последовательностей?

Диаграмма последовательностей состоит из:

- обычных объектов, представленных в виде прямоугольников (с подчеркнутыми именами),
- сообщений, изображенных сплошными линиями со стрелками,
- а также вертикальной оси времени, определяющей последовательность событий.

9.2. Автомат по продаже лимонада

На рис. 18 представлена диаграмма прецедентов для модели автомата по продаже лимонада. **Прецедент — это набор сценариев**.

Диаграмма последовательностей позволяет моделировать сценарии прецедента. Построим модель сценариев прецедента Покупка лимонада.

Целесообразно начать с построения диаграммы классов, моделирующей элементы автомата для продажи лимонада. Предположим, что в реализации этого сценария участвуют три объекта: **1)** лицевая панель, **2)** реестр для сбора монет, **3)** а также отсек для хранения лимонада и его передачи на лицевую панель.

Лицевая панель выполняет следующие функции:

- принимает монеты и заказ;
- отображает приглашения;
- получает сдачу от реестра и возвращает ее покупателю;
- возвращает монеты;
- получает лимонад из отсека, где он хранится, и передает его покупателю.

Реестр выполняет следующие действия:

- получает заказ покупателя (деньги и сорт лимонада) от лицевой панели;
- считает деньги;
- находит сдачу.

Отсек для хранения лимонада выполняет операции:

- проверяет наличие выбранного сорта лимонада;
- выдает стакан напитка.

Рассмотрим автомат для продажи лимонада как агрегат, содержащий эти три компонента. На рис. 20 показана соответствующая диаграмма классов.

Промоделируем основной **успешный** сценарий прецедента Покупка лимонада: покупатель бросает в автомат нужную сумму и выбирает марку лимонада. Поскольку речь идет об основном успешном сценарии, в автомате имеется по крайней мере один стакан лимонада нужного сорта, который и предлагается покупателю.

Последовательность действий по реализации сценария будет следующей.

1. Покупатель помещает монету в щель на лицевой панели автомата и выбирает сорт лимонада.
2. Монета попадает в реестр, который ее обрабатывает.
3. Для рассматриваемого основного сценария считаем, что нужный сорт лимона да имеется и реестр дает команду отсеку доставить лимонад к лицевой панели автомата.

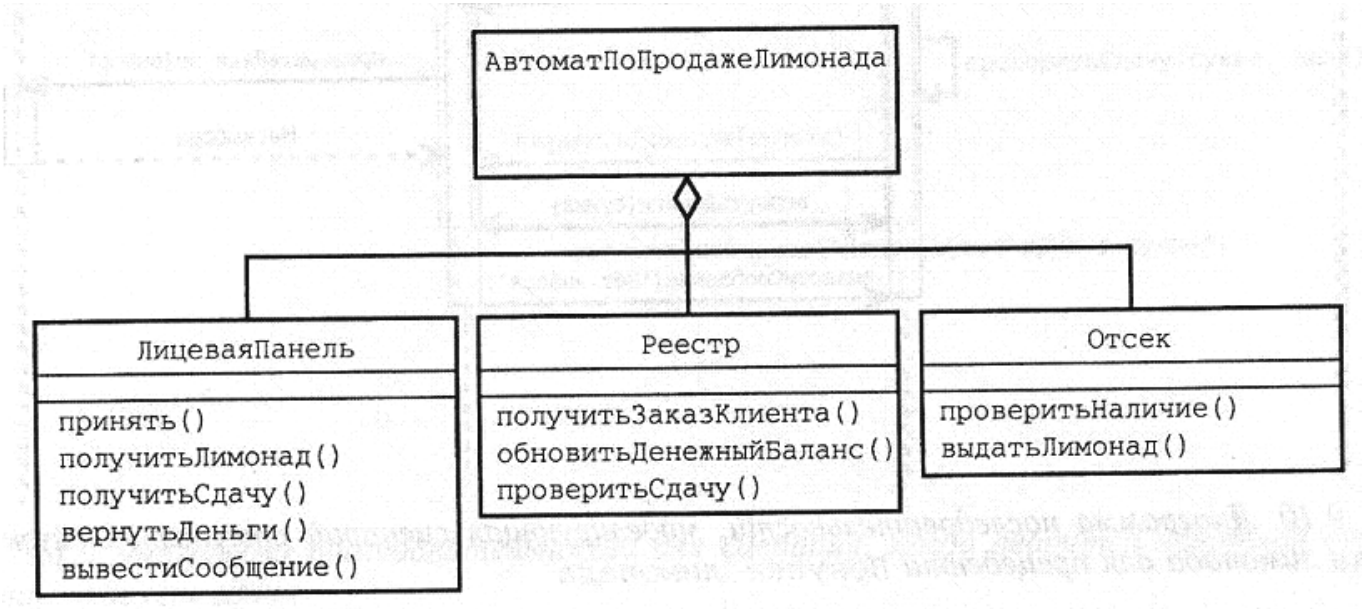

Рис. 20. **Диаграмма классов** автомата по продаже лимонада

На рис. 21 показана диаграмма последовательностей, моделирующая эти действия.

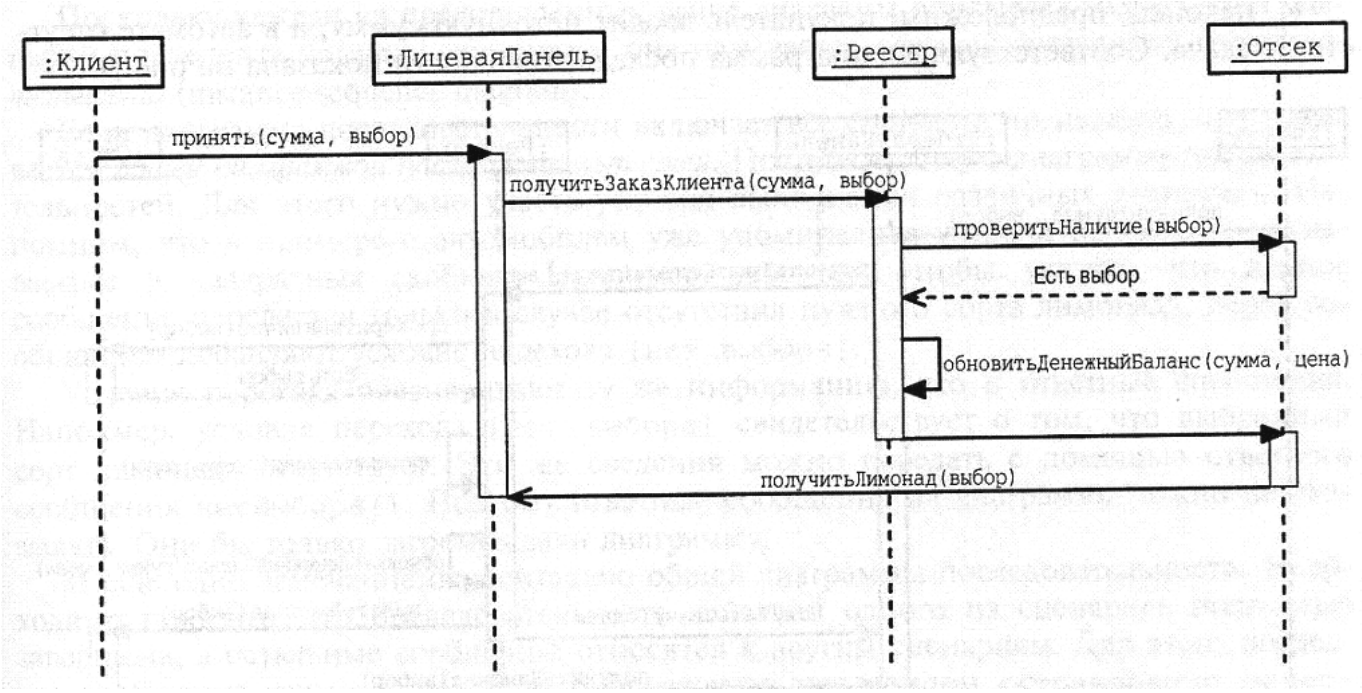

Рис. 21. Диаграмма последовательности,
моделирующая основной успешный сценарий прецедента **Покупка лимонада**

Это лишь один из сценариев прецедента. В другом сценарии выбранного сорта лимонада может и не оказаться.

Диаграмма последовательностей UML добавляет измерение времени ко взаимодействию объектов. Объекты располагаются в верхней части диаграммы, а время изменяется сверху вниз. От каждого отображенного на диаграмме объекта опускается **линия его жизни (пунктир)**.

Передаваемые объектами сообщения изображаются в виде линий, соединяющих одну линию жизни с другой, со стрелками на конце. Расположение сообщений вдоль вертикального направления соответствует времени их передачи. Переданные ранее сообщения находятся ближе к верхней части диаграммы Чем позже передается сообщение, тем ближе к нижней части диаграммы последовательностей оно располагается. Узкий прямоугольник на линии жизни объекта представляет **точку активации** — выполнение одной из операций объекта. На диаграмме последовательностей можно указать состояния объекта, разместив их вдоль линии жизни.

Диаграмма прецедентов представляет либо **один** сценарий прецедента, либо объединяет **все** сценарии. Каждому прецеденту соответствует **диаграмма последовательностей**. Один сценарий отображается на *частной* диаграмме, а несколько сценариев — на *общей*. На общей диаграмме последовательностей нередко отображают оператор "если". Условия для операторов "если" заключаются в квадратные скобки.

Если последовательность операций включает создание объекта, он представляется обычным образом. Его позиция вдоль вертикального направления должна соответствовать времени его создания.

/См. диаграммы последовательностей - Система Домовладелец (**LandLord**) **с. 17…19** [WEB-page, p. 6]/

10. Диаграммы коммуникации (взаимодействия)

Подобно диаграммам последовательностей, диаграммы коммуникации **отражают взаимодействие объектов**. На них **изображаются объекты вместе с сообщениями**, которыми они обмениваются.

Полезно работать с обоими видами диаграмм. Диаграмма последовательностей **акцентирует внимание на временном упорядочении** взаимодействий. Диаграмма коммуникации ориентирована **на состояние и общую организацию** взаимодействующих объектов.

Существует еще один способ понять различие между ними:

- диаграмма последовательностей упорядочена в соответствии **со временем**,
- диаграмма коммуникации — в соответствии с пространственным расположением объектов.

Оба вида диаграмм отражают взаимодействие объектов, поэтому **их относят к диаграммам взаимодействия.**

10.1 Что представляет собой диаграмма коммуникации?

В диаграмме объектов (**см. разд. 6**) описываются объекты и их взаимосвязь друг с другом. Диаграмма коммуникации является расширением понятия диаграммы объектов. В дополнение к связям между объектами в диаграмму коммуникации включают сообщения, которые объекты передают друг другу.

Чтобы понять взаимосвязь диаграмм коммуникации с объектными диаграммами, можно воспользоваться следующей аналогией. Представьте себе отличие моментального снимка от фрагмента видеосъемки. Объектная диаграмма — это моментальный снимок. На нем показаны экземпляры класса и их **взаимосвязь в конкретный момент времени**. Диаграмма коммуникации — это видеофильм. Она отражает **взаимодействие объектов во времени**.

Чтобы изобразить обращение к объекту, параллельно линии, которая соединяет объекты, рисуют стрелку. Направление стрелки указывает на объект, к которому обращаются. **Метка возле стрелки служит для описания этого сообщения**. В сообщении, как правило, дается команда объекту-получателю выполнить одну из операций. Стрелки при этом выполняют ту же функцию, что и на диаграммах последовательностей.

Любую диаграмму последовательностей (**см. разд. 9**) можно преобразовать в диаграмму коммуникации, и **наоборот**. Следовательно, последовательную информацию можно описать и в диаграмме коммуникации. Для этого к метке сообщения нужно добавить номер, соответствующий номеру этого сообщения в последовательности сообщений. Номер от самого сообщения отделяется двоеточием.

10.2. Автомат для продажи лимонада

Вернемся к примеру автомата для продажи лимонада (**рис. 18, 20 и 21**) и рассмотрим диаграмму коммуникации, соответствующую диаграмме последовательности, представленной на **рис. 21**.

Диаграмма коммуникации для основного успешного сценария прецедента Покупка лимонада показана на **рис. 22**.

Здесь приводится еще один пример вложенного сообщения. Ответное сообщение ЕстьВыбор является вложенным для операции проверить (Выбор). Его номер 3.1.

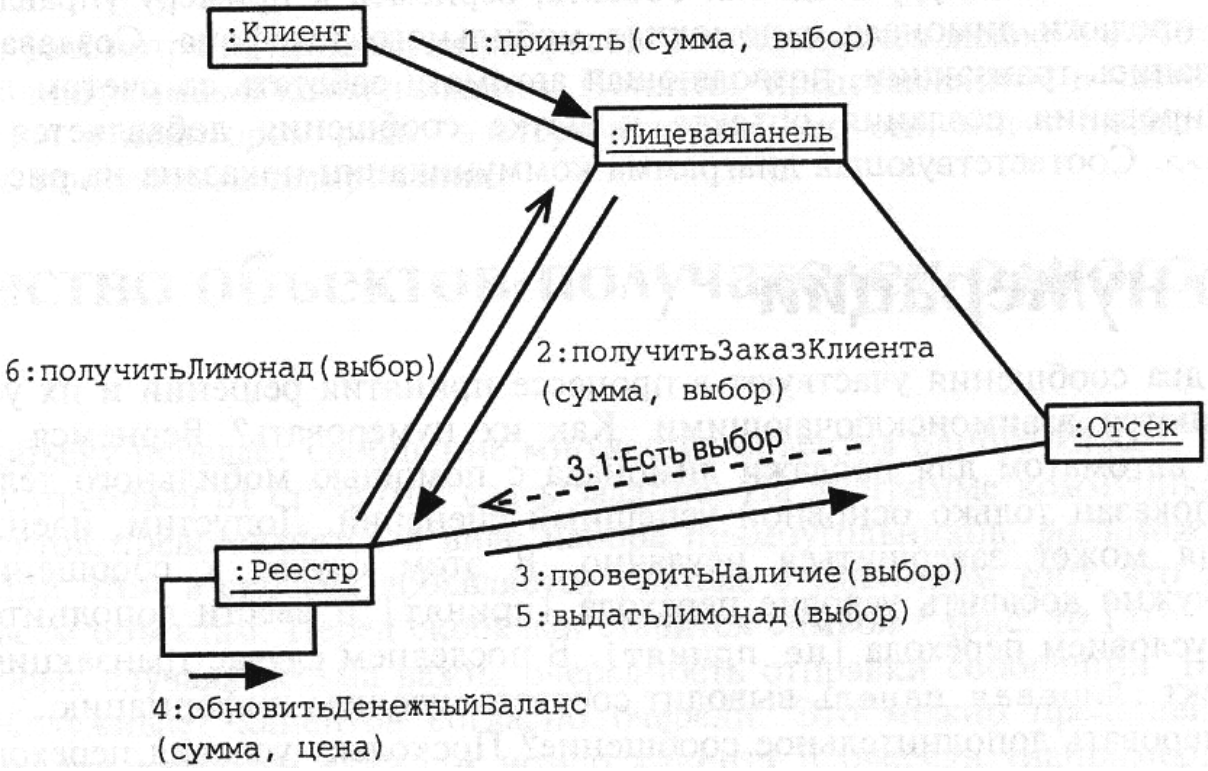

Рис 22. Диаграмма коммуникации
для основного успешного сценария прецедента Покупка лимонада (**см. рис. 18, 20 и 21**)

Диаграмма коммуникации — еще один способ представить информацию, которая ранее была отображена на диаграмме последовательности. Эти два типа диаграмм семантически эквивалентны, но при этом при разработке модели системы желательно использовать обе диаграммы. Диаграмма последовательностей упорядочена в соответствии со временем, диаграмма коммуникации — в соответствии со взаимосвязью объектов.

На диаграмме коммуникации ассоциации между объектами изображаются в виде сообщений, передаваемых объектами друг другу. **Сообщение** представляют в виде **стрелки** возле соединительной линии, а **имя сообщения** указывается **рядом со стрелкой**. Порядковый номер указывает место данного сообщения в общей последовательности.

Условия отображают в квадратных скобках. Чтобы описать цикл, надо поместить перед левой скобкой маленькую звездочку.

Некоторые сообщения являются дочерними по отношению к другим. В системе нумерации сообщений они описываются точно так же, как заголовки и подзаголовки в технических руководствах — с использованием десятичной точки для обозначения уровня вложенности.

Диаграмма коммуникации позволяет моделировать множество объектов-получателей одного класса независимо от того, получают они сообщения по очереди или одновременно. Можно также изображать активные объекты, управляющие потоком сообщений, а также синхронизированные сообщения.

11. Диаграммы видов деятельности

Диаграммы видов деятельности описывают стадии какой-либо операции (метода) или процесса.

Диаграмма видов деятельности UML очень похожа на старые блок-схемы. В ней точками принятия решений и переходов описывается последовательность шагов (названных с достаточной точностью **видами деятельности**). Такая схема достаточно удобна для отображения бизнес-процессов или операций (методов). Поэтому диаграммы видов деятельности являются неотъемлемой частью системного анализа.

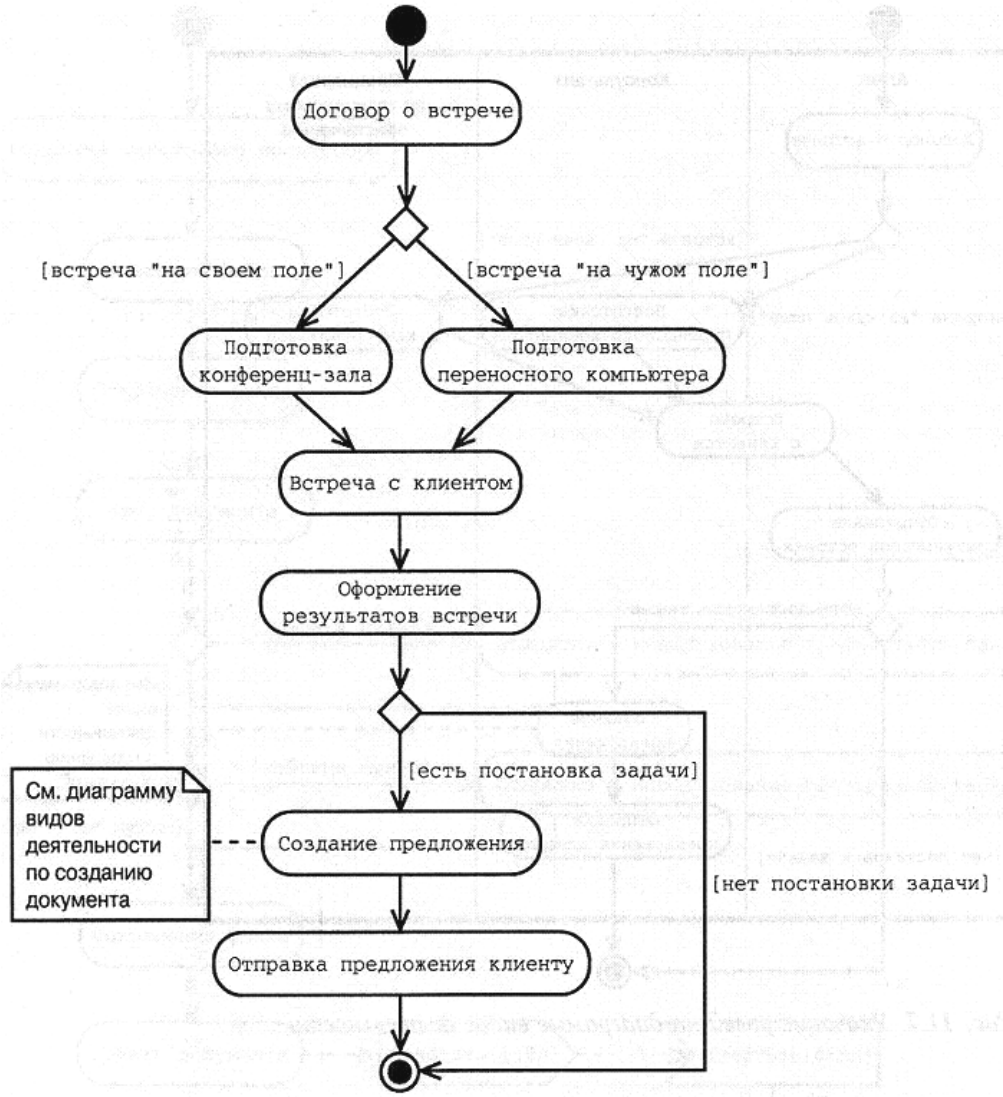

Рис. 23. **Диаграмма видов деятельности** для бизнес-процесса встречи с новым клиентом

11.1. Что такое диаграмма видов деятельности?

Первое и самое главное заключается в том, что с помощью диаграммы видов деятельности реально описывается все, что происходит во время какой-либо операции (метода) или процесса.

Каждый вид деятельности изображается прямоугольником со скругленными углами — более узким и овальным, чем символ состояния в "Диаграммах состояний" (**см. разд. 9, рис. 19**). После завершения одного вида деятельности переход к следующему происходит автоматически. Переход от одного вида деятельности к другому изображается стрелкой. Как и на диаграмме состояний, на диаграмме видов деятельности имеется начальная точка, изображенная в виде **закрашенного кружка**, и конечная точка в виде "**глазка**".

Различают также **обзорную диаграмму взаимодействия,** которая обеспечивает общий каркас **диаграммы видов деятельности**, элементами которой являются **диаграммы взаимодействия (последовательностей, см. рис. 21)**.

/**см. диаграмму действий** - Система Домовладелец (**LandLord**) **с. 13, 14** [WEB-page, p. 6]/

12. Диаграммы компонентов

Рассматриваются диаграммы UML, с помощью которых описываются **программные** компоненты.

12.1. Что такое компонент?

Программный компонент является модулем или частью системы. Поскольку он содержит реали-

зацию одного или нескольких классов, то находится в компьютере, а не в воображении аналитиков. Компонент обеспечивает интерфейс с другими компонентами.

В **UML 1.x** компонентами считались таблицы, файлы данных, исполняемые файлы, документы и динамически подключаемые библиотеки (**dll**). На самом деле для уточнения этих понятий при моделировании эти типы элементов назывались **компонентами развертывания, рабочими и исполняемыми** компонентами. В **UML 2.0** все эти понятия называют общим именем — артефакт, под которым подразумевают **фрагмент информации**, **используемый или генерируемый системой.**

В отличие от артефакта, компонент определяет **функциональность** системы. Он представляет собой программную реализацию одного или нескольких классов. Следовательно, артефакт (**исполняемый**) — это реализация компонента.

Модели компонентов и их взаимосвязей строят в следующих целях.

1. Клиенты смогут увидеть структуру законченной системы.
2. Разработчики смогут представить себе структуру будущей работы.
3. Редакторы, ответственные за предоставление документации и файлов справки, смогут лучше понять суть разработки.
4. Компоненты можно будет использовать многократно.

Одним из важнейших аспектов компонентов является потенциальная возможность их **многократного использования**.

12.2. Компоненты и интерфейсы

Работая с компонентами, приходится иметь дело с их интерфейсами. Объект должен "показать лицо" окружению, чтобы другие объекты (в том числе и люди), могли давать объекту указания о выполнении операций. Этим "лицом" и является объектный **интерфейс**.

12.3. Несколько слов об интерфейсах

Интерфейс класса — это набор операций (public-методов), который определяет поведение класса. **Интерфейс класса** — это набор операций, которые класс предоставляет другим классам.

Связь между классом и его интерфейсом называется **реализацией**.

Интерфейс, используемый классом, такой же, как и интерфейс его программной реализации (**компонента**). Это означает, что способы описания интерфейсов класса и компонента одинаковы

Операции компонента выполняются только через его интерфейс. Как и в случае класса, связь между компонентом и его интерфейсом называется **реализацией**.

Интерфейс компонента может быть **открытым**, и операции этого интерфейса могут использоваться другими компонентами. Другими словами, компонент может получать доступ к услугам другого компонента. Говорят, что компонент, обеспечивающий доступ, предоставляет **экспортируемый интерфейс**. Для компонента, который пользуется таким доступом, этот интерфейс называется **импортируемым**.

12.4. Что такое диаграмма компонентов?

Диаграмма компонентов содержит:

- компоненты,
- интерфейсы
- и их взаимосвязи.

В ней также могут участвовать и другие типы обозначений.

Диаграмма компонентов — это **модульное представление компьютерной системы**. Артефакт — это **фрагмент информации**, который используется или создается в программной системе. Компоненты определяют функциональность программной системы.

В **UML 1.x** изображением компонента служил прямоугольник с двумя маленькими прямоугольниками на его левой стороне.

В **UML 2.0** компонент обозначается прямоугольником с ключевым словом <<**компонент**>> (**рис. 24**). Для соблюдения преемственности обозначений в **UML 2.0** рекомендуется в верхнем правом углу нового обозначения добавлять пиктограмму компонента из **UML 1.x**.

Обозначение артефакта — это прямоугольник с ключевым словом <<**артефакт**>>. В верхнем правом углу можно использовать символ комментария.

Интерфейс можно представить двумя способами. В одном случае — это прямоугольник, содержащий информацию об интерфейсе. Он соединяется с компонентом **пунктирной** линией со стрелкой в виде **полого треугольника**. В другом случае — **это маленький кружок**, соединенный с компонентом **сплошной линией**. В **UML 2.0** для изображения интерфейса, экспортируемого одним компонентом и импортируемого другим, применяется **кружочек с гнездом**. При этом кружочек обозначает экспортируемый интерфейс, а гнездо — импортируемый.

На рис. 24 приведена диаграмма **компонентов (UML 2.0)** на которой изображена **реализация** и **зависимость** — связь между компонентом и интерфейсом, с помощью которого осуществляется доступ к другому компоненту. **Зависимость** обозначается **пунктирной линией со стрелкой**. Нижняя диаграмма на рис. 24 соответствует сокращенным обозначениям.

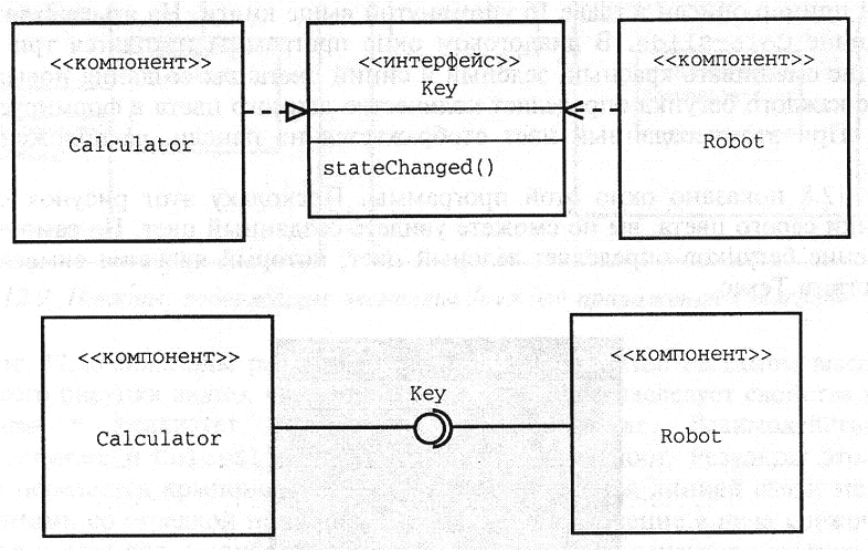

Рис. 24. Два способа изображения **реализации** и **зависимости** на одной диаграмме

13. Диаграммы развертывания

На диаграммах развертывания представляются аппаратные средства, т.е. реальное оборудование. Аппаратные средства очень важны в многокомпонентных системах. В современном компьютерном мире такие системы являются распределенными, предоставляют большие возможности и могут использоваться на множестве различных платформ.

13.1. Что такое диаграмма развертывания?

Диаграммы развертывания отражают:

- способ воплощения артефактов (**артефакт**, - это **фрагмент информации**, **используемый или генерируемый системой**) в физической системе
- и способ соединения аппаратных средств между собой.

Главным аппаратным элементом является **узел** — общее название для любого вычислительного ресурса.

В **UML 1.x** зачастую выделяли два типа узлов: **процессоры** (узлы, выполняющие команды компонента) и **устройства** (периферийные аппаратные средства, которые не выполняют команды компонентов, а осуществляют интерфейс с внешним миром). Хотя такая классификация в UML 1.x не была формализована, она широко использовалась.

В **UML 2.0 устройство** формально определяется как узел, выполняющий артефакты

(под **артефактом** понимается исполняемый код) .

В **UML 2.0** узел изображается в виде куба (как и в **UML 1.x**), с которым связано определенное имя и необязательное ключевое слово <<**устройство**>>.

Линия, соединяющая два куба, представляет **соединение двух узлов**. Соединение — это не обязательно кусок провода или кабель. Как известно, существуют также и беспроводные соединения, например, на основе инфракрасного излучения и спутниковой связи.

Со смещением акцентов в сторону артефактов в **UML 2.0** появилось множество связанных с ними понятий. Одно из них — **спецификация развертывания** — артефакт, обеспечивающий параметры для другого артефакта. Хорошим примером является команда инициализации, используемая для модемных соединений. Это строка символов, в которой задаются значения характеристик модема.

13.2. Применение диаграмм развертывания

В современных многопроцессорных системах могут содержаться **узлы**, расположенные далеко друг от друга. Для более полного представления этой картины строятся диаграмм развертывания для **сетей**.

13.3. Домашняя система (сеть)

В модели домашней системы символом **узла** обозначены дополнительные **периферийные элементы**..

Способ использования узла в данном контексте в **UML 2.0** называется **ненормативным**. Строго говоря, в **UML 2.0 узел** представляет собой **элемент аппаратных средств**, который может выполнять вычисления. Поскольку система содержит периферийные устройства, их логично включить в модель. Чтобы отличить периферийные устройства от основных, соответствующие ненормативные узлы можно пометить ключевым словом <<**периферия**». Однако это ключевое слово не входит в спецификацию **UML**. Имя такого ненормативного узла является достаточно информативным, и применение дополнительных ключевых слов становится ненужным.

Диаграмма развертывания изображена на рис. 25. На ней показано широкополосное соединение с провайдером **Internet** и подключение к Internet самого провайдера. **Облако**, представляющее Internet, и "**молния**", символизирующая беспроводное соединение, не являются стандартными обозначениями UML, но их использование на диаграмме весьма полезно для повышения ясности модели.

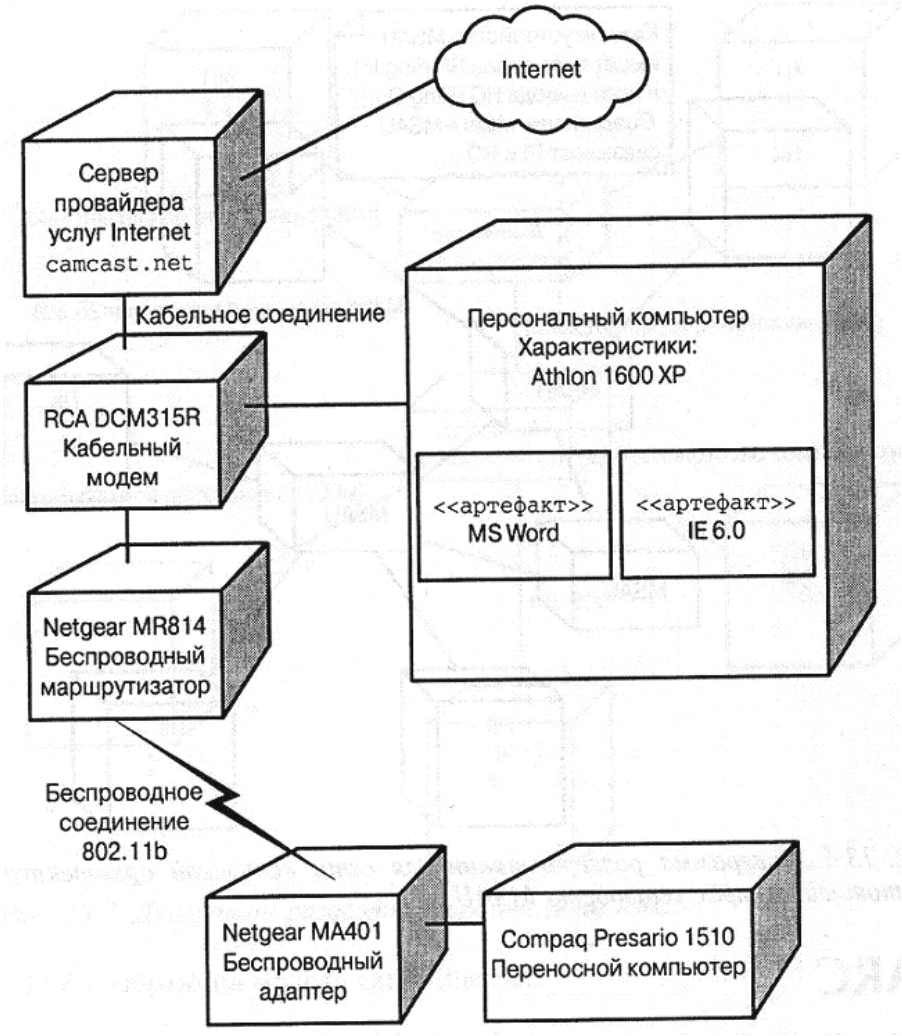

Рис. 25. Диаграмма развертывания для домашней системы

Диаграмма развертывания **UML** описывает физическую систему в готовом виде. Система состоит из **узлов**, каждый из которых изображается в виде **куба**. **Линия** между двумя кубами символизирует соединение двух узлов. На каждом из узлов можно отобразить соответствующие **артефакты**.

Диаграммы развертывания очень удобны для моделирования сетей. Например, кольцо с маркерным доступом, сеть **ARC**, сеть **Ethernet** и беспроводная сеть **Ricochet**.

14. Диаграммы пакетов

Диаграммы **пакетов** способствуют более глубокому пониманию других диаграмм. Диаграммы пакетов получили настоящий статус лишь в **UML 2.0**.

14.1. Для чего нужны пакеты

Как следует из названия, пакет предназначен для группирования элементов диаграмм (например, классов /см. разд. 5/ или прецедентов /см. разд. 7/). Для помещения элементов в пакет их нужно разместить внутри изображения в виде папки. Если пакету присвоено имя, значит, это же имя связано и с группой элементов. В терминах языка **UML** пакет предоставляет для **содержащихся** в нем элементов **пространство имен**.

Для ссылки на элемент пакета используется следующая форма записи**:**

ИмяПакета: :ЭлементПакета (например, **Инструмент: :Молоток**).

Такая система именования определяет **полностью определенные имена** элементов.

14.2. Отношения между пакетами

Пакеты могут связываться друг с другом одним из трех следующих способов. Один пакет может **обобщать** другой пакет; может **зависеть** от другого пакета или **уточнять** его.

Обобщение и **зависимость** в контексте других элементов **UML** уже рассматривались выше. **Уточнение** имеет отношение к уровню детализации.

14.3. Объединение пакетов

Пакет может объединяться с другим пакетом. Отношение **объединения** является разновидностью отношения **зависимости** между пакетом, который **объединяет** *(***источник***)*, и пакетом, который **объединяется** *(***целевой***)*. Результатом объединения является преобразование пакета-источника.

Рассмотрим пример. Предположим, имеются пакеты **Компьютеры** и **Телефоны**, в которых содержатся какие-то классы. Третий пакет **Компьютерная телефония** объединяется с каждым из двух первых пакетов. Эти пакеты с их содержимым показаны на рис. 26. Обратите внимание на то, что пакет **Компьютерная телефония** пуст.

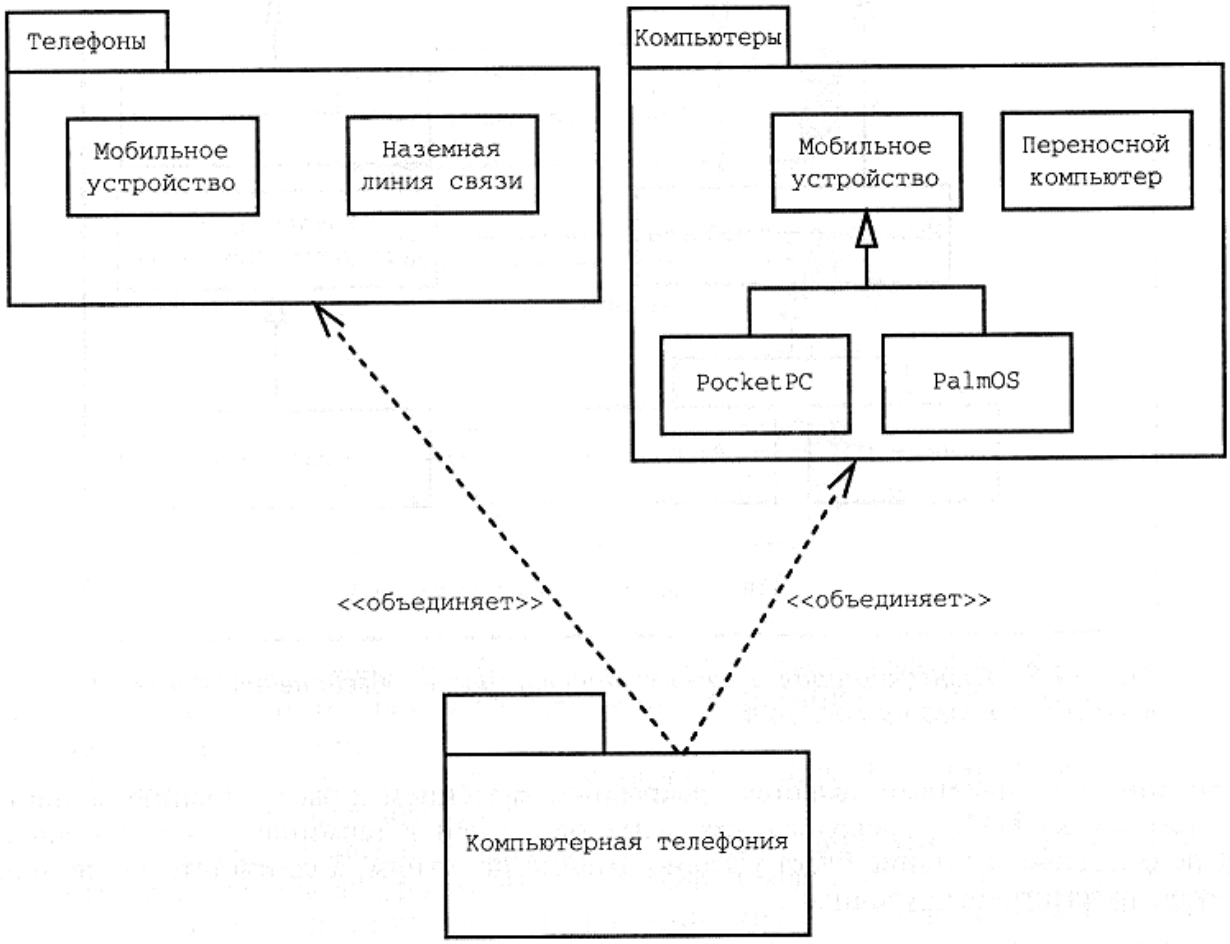

Рис. 26. Моделирование объединения пакета с двумя другими пакетами

Объединение приводит к трансформации **пакета Компьютерная телефония**, как показано на рис. 27. Импортируются все классы из двух пакетов-источников. В соответствии с системой именования отношения наследования для классов **Переносной компьютер** и **Наземная линия связи** показаны вместе с именами исходных целевых пакетов.

Отношение **наследования** для класса **Мобильное устройство** иллюстрирует важный момент объединения. Если выполняется объединение пакетов и в результирующем пакете содержатся классы с теми же именами, то класс из целевого пакета будет содержать атрибуты и операции всех одноименных классов из пакетов-источников. В рассматриваемом примере класс **Мобиль-**

ное устройство из пакета **Компьютерная телефония** унаследует соответствующие члены от классов каждого пакета-источника. Фактически класс **Компьютерная телефония: :Мобильное устройство** будет представлять собой некий "интеллектуальный" мобильный телефон с компьютерными возможностями. В свою очередь, существование классов-наследников **PocketPC** и **PalmOS** говорит о том, что "интеллектуальный" телефон может работать под управления разных операционных систем (в примере — двух).

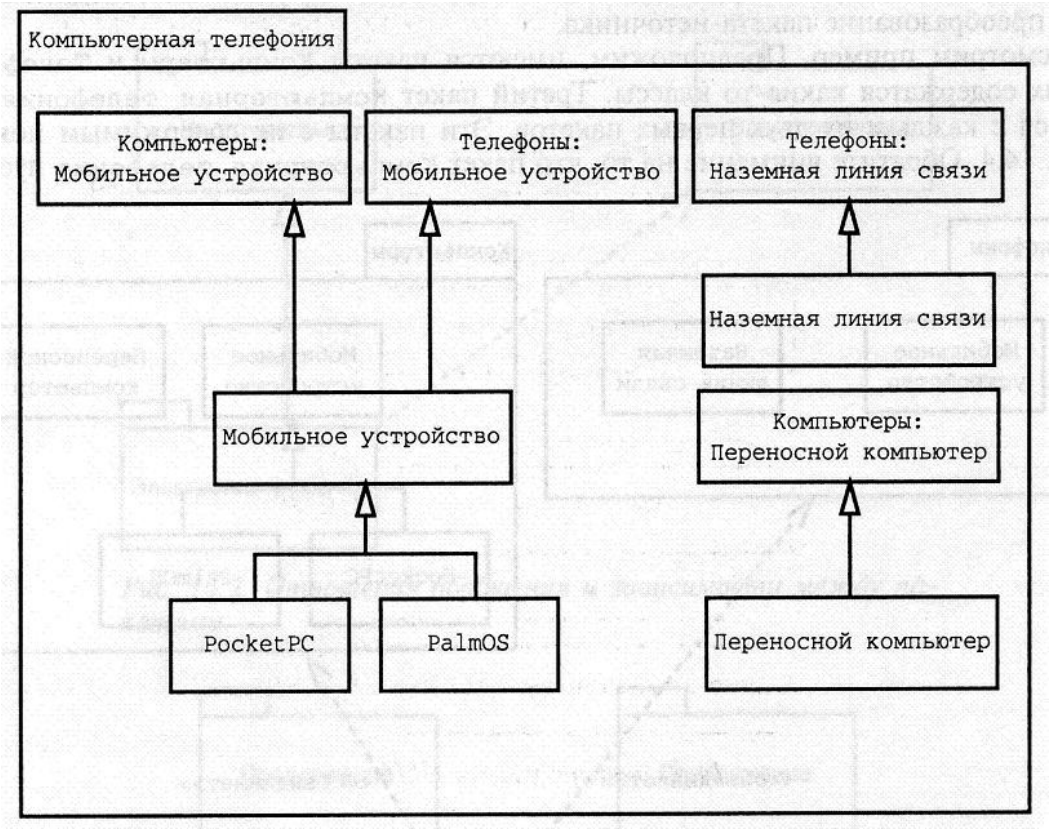

Рис. 27. Трансформация целевого пакета после объединения пакетов, представленного на рис. 26

15. Временная диаграмма

Если внимательно посмотреть на диаграмму последовательностей, показанную на рис. 21, то можно заметить, что длительность пребывания в каждом из состояний нигде явно не указана. Эту проблему позволяет решить временная диаграмма (**timing diagram**), которая появилась в стандарте **UML 2.0**. Пример такой диаграммы показан на рис. 28.

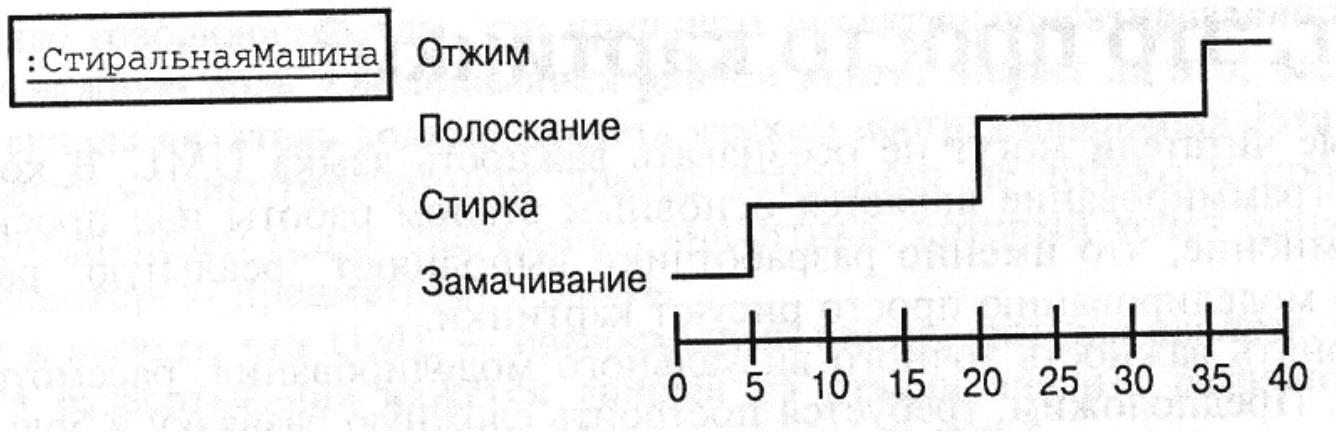

Рис. 28. Временная диаграмма UML

16. Зачем так много различных диаграмм?

Диаграммы **UML** позволяют описать систему с разных точек зрения. Необязательно использовать все диаграммы в каждой модели **UML**. Большинство моделей содержит набор приведенных выше диаграмм.

Зачем нужны различные представления системы? Обычно системой занимается несколько специалистов, которых интересуют различные ее аспекты. Рассмотрим пример со стиральной машиной. Если вы разрабатываете двигатель стиральной машины, у вас одно представление о системе. А если пишете инструкцию по эксплуатации, то, соответственно, другое. Если вы проектируете общий вид машины, ваше представление о системе будет кардинально отличаться от того, которое вы бы имели, желая просто постирать белье.

Точное проектирование системы требует наличия всех возможных представлений, а диаграммы UML позволяют объединить отдельные представления. Основная цель состоит в удовлетворении требований каждого заинтересованного лица.

Может, это просто картинки?

Некоторые студенты «не любят» блок-схемы и не осознают важность языка **UML**. В конце концов, разве не программирование является основным этапом работы над проектом? Зачастую бытует мнение, что именно разработчики-программисты выполняют "реальную" работу, а специалисты по моделированию просто рисуют картинки.

Чтобы понять важность точного визуального моделирования, рассмотрим следующий пример. Предположим, требуется построить сложную развязку в большом городе, включающую сеть мостов и подземных туннелей.

При небрежном моделировании чертежи могут содержать неточную информацию. Допустим, на одном чертеже автор не учел строение, расположенное на месте предполагаемой развязки. На другом чертеже создатели на несколько метров ошиблись при отображении плана подземных туннелей. Такая ошибка может выявиться только в процессе строительства, когда рабочие не смогут правильно соединить два туннеля. Устранение подобных "оплошностей" может обойтись в очень крупную сумму и привести к срыву сроков выполнения проекта. **Звучит убедительно?**

Системы разрабатываются людьми. Поэтому использование простой системы обозначений в процессе разработки поможет избежать многих ошибок.

Система обозначений UML стала стандартом в области разработки систем. Это результат работы Гради Буча (Grady Booch), Джеймса Румбаха (James Rumbaugh) и Айвара Якобсона (Ivar Jacobson). UML содержит набор разных диаграмм и поэтому поддерживает стандарт, позволяющий системному аналитику строить комплексный пакет документации, которым могут пользоваться клиенты, программисты и другие специалисты, участвующие в разработке. Все эти диаграммы нужны, так как каждая из них адресована отдельному заинтересованному лицу системы.

Модель UML описывает, что предположительно будет делать система (**это поведение**). Но ничего не говорится о том, как система будет это делать (**это реализация**).

Литература

№ пп	Заглавие	Автор	Год	Полочный индекс (ВШЭ)	Кол-во экз.
1	UML и Rational Rose	/ Боггс, У. [1]	2001	**004 Б733**	6
2	Применение UML и шаблонов проектирования	/ Ларман, К. [1]	2001	**004 Л253**	3
3	Базы данных и UML	/ Мюллер, Р. Дж. [1]	2002	**004 М982**	4
4	Визуальное моделирование с помощью Rational Rose 2002 и UML	/ Кватрани, Т. [1]	2003	**004 К324**	11
5	Анализ требований и проектирование систем	/ Мацяшек, Л. А. [1]	2002	**004 М369**	3
6	An introduction to object-oriented analysis	/ Brown, D. W. [1]	2002	**004 B89**	1
7	UML and the unified process **[полный текст]**	/ Favre, Liliana [1]	2003	**Доступно в БД ebrary с компьютеров ГУ-ВШЭ**	0
8	UML for real **[полный текст]**	/ A [4640]	2003	**Доступно в БД ebrary с компьютеров ГУ-ВШЭ**	0
9	Enterprise Java with UML **[полный текст]**	/ Arrington, C. T [1]	2001	**Доступно в БД ebrary с компьютеров ГУ-ВШЭ**	0
10	UML. Проектирование систем реального времени, параллельных и распределенных приложений	/ Гома, Х. [1]	2002	**004 Г64**	30
11	Освой самостоятельно UML за 24 часа	/ Шмуллер, Дж. [1]	2005	**004 Ш752**	10
12	UML **Подробная информация о книге**	/ Буч, Г. [2]	2006	**004 Б947**	20
13	UML. Основы	/ Фаулер, М. [3]	2006	**004 Ф282**	12
14	UML	/ Леоненков, А. В. [1]	2006	**004 Л473**	2

15. Забудский Е.И. Учебно-методический комплекс дисциплины «Объектно-ориентированный анализ и программирование». М.: Кафедра Архитектуры программных систем ГУ-ВШЭ, 2005... 2008.

Internet-ресурс – http://new.hse.ru/C7/C17/zabudskiy-e-i/default.aspx .

Подробную информацию о **UML** можно получить на **Web**-странице некоммерческой организации **Unified Modeling Language (UML)** (**www.omg.org**) по адресу **www.omg.org/uml** .