Лекция 1. Введение

Введение в курс «Технологии проектирования программного обеспечения»

Организационная структура курса

Цели и задачи курса

- Изучить основные элементы базовой нотации языка UML (версия 1.5);
- Изучить методику построения визуальных моделей программных систем;
- Освоить методику проведения анализа программных систем (бизнес-анализ, анализ функциональных требований, анализ прецедентов, системный анализ);
- Ознакомится с особенностями проектирования программных систем.

Список рекомендуемой литературы

Парадигма объектно-ориентированного подхода

1. Г. Буч и д.р.«Объектно-ориентированный анализ и проектирование с примерами приложений», Москва, «Вильямс», 2008 г., 3-е издание.

Методика визуального моделирования

1. Дж. Рамбо, А. Якобсон, Г. Буч «UML Специальный справочник», Санкт-Петербург, «Питер», 2002 г.
2. Дж. Арлоу, А. Нейштадт, «UML-2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование». Санкт-Петербург, «Символ-Плюс», 2007 г., 2-е издание.
3. Дж. Рамбо, М. Блаха «UML-2 Объектно-ориентированное моделирование и разработка» Санкт-Петербург, «Питер», 2007 г., 2-е издание.

Модели жизненного цикла и процесса разработки

1. Основы программной инженерии (по SWEBOK, 2004 г.), перевод С. Орлик, 2004-2010 г.
2. Дж. Рамбо, А. Якобсон, Г. Буч «Унифицированный процесс разработки программного обеспечения», Санкт-Петербург, «Питер», 2002 г.
3. Ф. Кратчен в цикле статей Л.Новикова «Введение в Rational Unified Process»

CASE-инструментарий

1. У. Боггс, М. Боггс « UML, Rational Rose», Москва, «Лори», 2001 г.
2. С. Трофимов «Case-технологии. Практическая работа в Rational Rose», Москва, «Бином», 2001 г.

Основы анализа ПО

1. К. Вигерс «Разработка требований к программному обеспечению», Москва, «Русская Редакция», 2004 г.

Введение в проектирование

1. К. Ларман «Применение UML и шаблонов проектирования», Москва, «Вильямс», 2001 г.

Лекция 1. Введение

Глоссарий некоторых терминов

- ***Артефакт*** *(artifact)* – элемент информации, используемый или порождаемый в процессе разработки программного обеспечения.
- ***Архитектура*** *(architecture)* **–** логическая и физическая структура системы, сформированная всеми стратегическими и тактическими проектными решениями.
- ***Взаимодействие*** *(interaction)* – обмен сообщениями с целью достижения определенной цели.
- ***Классификатор*** (*classifier)* – дискретная концепция модели, обладающая индивидуальностью, состоянием и поведением, находящаяся в определенных отношениях с другими элементами модели.
- ***Модель*** *(model)* -- *абстракция* физической системы, рассматриваемая с определенной точки зрения, представленная на некотором языке или в графической форме.
- ***Метод*** *(method)* – это последовательный процесс создания моделей, которые описывают различные стороны разрабатываемой программной системы.
- ***Методология*** *(methodology)* – совокупность методов, применяемых в цикле разработки, объединенных общим философским подходом.
- ***Отношение*** *(relationship)* — семантическая связь между отдельными элементами модели
- ***Предметная область*** *(domain)* - часть реального мира, которая имеет существенное значение или непосредственное отношение к процессу функционирования программы.
- ***Роль*** *(role)* – имеющее имя специфическое поведение некоторой сущности, рассматриваемой в определенном контексте. Роль может быть статической или динамической..
- ***Семантика*** *(semantics)* – формальная спецификация значений и поведения чего-

Лекция 1. Тема 1

Сложность программных систем

Промышленным программным системам присуща *сложность* – один разработчик не в силах охватить все аспекты системы и ее сложность превышает возможности человеческого интеллекта. Сложные программные системы требуют *сопровождения*, *сохранения* и имеют тенденцию к *эволюции* в процессе их использования.

Сопровождение (maintenance) -- исправление выявленных ошибок; ***сохранение*** (preservation) – попытка продления функционирования устаревших и распадающихся частей ПО всеми возможными способами; ***эволюция*** (evolution) -- реакция на изменение технических требований.

Пять признаков сложной системы (из работ Саймона и Куртуа):

1. Сложные системы часто являются иерархическими и состоят из взаимозависимых подсистем, которые в свою очередь сами могут быть разделены на подсистемы и т.д. вплоть до самого низкого уровня.
2. Выбор, какие компоненты в данной системе считаются элементарными, относительно произволен и в большой степени оставляется на усмотрение исследователя
3. Внутрикомпонентная связь обычно сильнее, чем связь между компонентами.
4. Иерархические системы обычно состоят из немногих типов подсистем, по-разному скомбинированных и организованных.
5. Любая работающая сложная система является результатом развития работавшей более простой системы

Основные причины, приводящие к сложным программным системам

- сложность реальной предметной области (реального мира) из которой приходит заказ на разработку;
- трудность управления процессом разработки;
- необходимость обеспечить достаточную гибкость системы;
- неудовлетворительными способами описания поведения больших дискретных систем.

Лекция 1. Тема 2

Алгоритмическая и объектно-ориентированная декомпозиции

Проектирование сложных программных систем связано с необходимостью разделять их на все меньшие и меньшие подсистемы, каждую из которых можно совершенствовать независимо. Процесс разделения систем на более мелкие подсистемы называется *декомпозицией. Основными способами декомпозиции являются:*

Алгоритмическая декомпозиция *– это процесс разбиения системы по алгоритмам. Каждый модуль системы выполняет один из этапов (функций) общего процесса. Результат разбиения отражается в формуле:*

Программа = Данные + Алгоритмы

Объектно-ориентированная декомпозиция *– процесс разбиения системы на части, соответствующие классам и объектам предметной области.*
Объекты взаимодействуют друг с другом, обмениваясь сообщениями и выступая друг к другу в отношении «клиент/сервер». Сообщения, которые может принимать объект определены в его интерфейсе. В этом смысле посылка сообщения «объекту-серверу» эквивалентна вызову соответствующего метода объекта.

Объектно-ориентированная декомпозиция имеет несколько важных преимуществ перед алгоритмической: она уменьшает размер систем за счет повторного использования общих механизмов, ООС более гибки и проще эволюционируют, существенно снижается риск при создании сложной программной системы, т.к. она развивается из меньших систем, в которых мы уверены.

Объектно-ориентированный подход к разработке ПО

Объектно-ориентированный анализ и проектирование (ООАП, Object-Oriented Analysis/Design) - технология разработки программных систем, в основу которых положена объектно-ориентированная методология представления предметной области в виде объектов, являющихся экземплярами соответствующих классов.

Объектно-ориентированный анализ — это метод анализа, исследующий требования к системе с точки зрения классов и объектов, относящихся к словарю предметной области.

Объектно-ориентированное проектирование — это метод проектирования, сочетающий процесс объектно-ориентированной декомпозиции и систему обозначения для представления логической и физической, а также статической и динамической моделей проектируемой системы.

Объектно-ориентированное программирование — это метод программирования, основанный на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определенного класса, а классы являются членами определенной иерархии наследования.

Лекция 1. Тема 2

Принципы объектно-ориентированного подхода

Объектно-ориентированная технология основывается на **объектной модели** (object model), которая, согласно Г.Бучу, базируется на четырех главных принципах (главных в том смысле, что без любого из них модель не будет объектно-ориентированной): *абстрагирование, инкапсуляция, модульность, иерархичность, полиморфизм.*

Абстракция *(abstraction)* – выделяет **существенные характеристики** некоторого объекта, которые отличают от всех других объектов и четко определяют его концептуальные границы для наблюдателя.
Абстрагирование – процесс выявления абстракций.
Абстракция определяет границу представления соответствующего элемента *модели* и применяется для определения фундаментальных понятий ООП, таких как *класс* и *объект*.

Инкапсуляция *(encapsulation)* – процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение.
Инкапсуляция характеризует сокрытие отдельных деталей внутреннего устройства классов от внешних по отношению к нему объектов или пользователей.
Инкапсуляция определяет четкие границы между различными абстракциями.
Абстракция и инкапсуляция дополняют друг друга: абстрагирование направлено на наблюдаемое поведение объекта, а инкапсуляция занимается внутренним устройством.

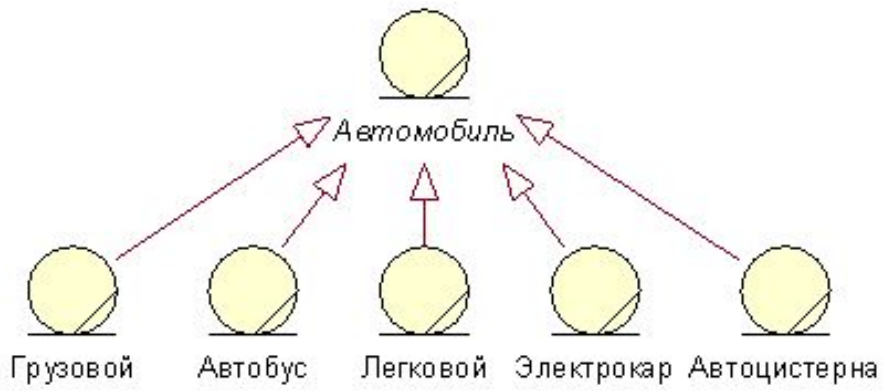
Модульность *(modularity)* – это свойство системы, которая была разложена на внутренние связанные, но слабо связанные между собой модули.

Лекция 1. Тема 2

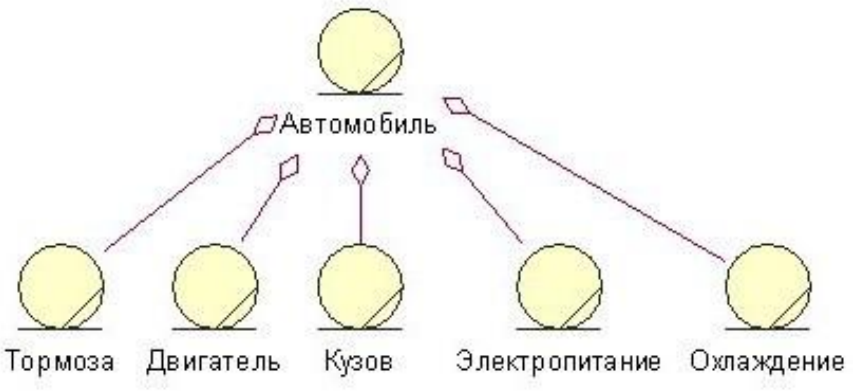
Принципы объектно-ориентированного подхода (продолжение)

Иерархия *(hierarchy)* – упорядочение абстракций, расположение их по уровням. (в другой литературе это понятие вводится как «наследование»)
Основными видами иерархических структур применительно к сложным системам являются

структура классов (иерархия «is-a» -- отношение «обобщение/специализация» или «наследование»)



структура объектов (иерархия «part of» -- часть от или отношение агрегации)



Полиморфизм *(polymorphism)* – наличие множества форм или реализаций конкретной функциональности
Под ***полиморфизмом*** (греч. Poly - много, morfos - форма) понимается свойство *объектов* принимать различные внешние формы в зависимости от обстоятельств.
С точки зрения ООА проявлением полиморфизма является то, что одну и ту же функциональность программной системы можно реализовать несколькими разными вариантами ее архитектуры.
Применительно к ООП *полиморфизм* означает, что действия, выполняемые одноименными методами, могут различаться в зависимости от того, к какому из *классов* относится тот или иной метод.

Лекция 1. Тема 2

Классы и объекты

Класс (class) представляет собой *абстракцию* совокупности реальных *объектов*, которые имеют общий набор свойств и обладают одинаковым поведением.

Объект (object) – это некая сущность реального мира или концептуальная сущность с четко определенными границами. Каждый объект в системе имеет три характеристики: **состояние**, **поведение** и **индивидуальность**. Объект в контексте ООП рассматривается как экземпляр соответствующего класса.
Объекты, которые не имеют идентичных свойств или не обладают одинаковым поведением, по определению, не могут быть отнесены к одному *классу*.
Состоянием (state) объекта называется одно из условий, в которых он может находиться. Состояние объекта характеризуется перечнем (обычно статическим) всех свойств данного объекта и текущими (динамическими) значениями каждого из этих свойств – атрибуты (attribute) объекта **Поведение** объекта (behavior) – определяет, как объект реагирует на запросы других объектов и что может делать сам объект.
Поведение объекта – это его наблюдаемая и проверяемая из вне деятельность. Поведение реализуется с помощью набора операций (operation) объекта.
Индивидуальность (identity) означает, что каждый объект уникален, даже если его состояние идентично состоянию другого объекта. Уникальность объекта реализуется в его имени: два разных объекта, принадлежащих одному классу не могут иметь одинаковое имя.

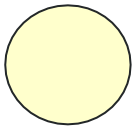
Качество классов

«*Хороший*» класс должен отражать только одну основную сущность. Названия классов выбирают в соответствии с понятиями предметной области. Имя класса должно быть существительным в единственном числе, наиболее точно характеризующем предмет.
«*Хороший*» класс должен содержать все необходимые атрибуты, характеризующие моделируемую сущность предметной области в контексте данного ПО и обладать необходимым и достаточным набором операций.

Лекция 1. Тема 2

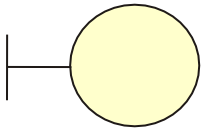
Моделирование классов в UML

Модели классов OOA Модель классов бизнес-анализа



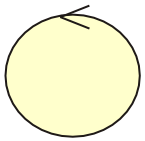
entity

entity (сущность) -- класс программной системы, отображение некой сущности предметной области. Основное назначение – хранение данных



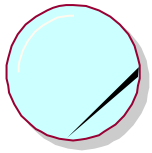
boundary

bounary (граничный класс) – класс программной системы, представляющий интерфейс между внешним пользователем (actor) и программной системой



control

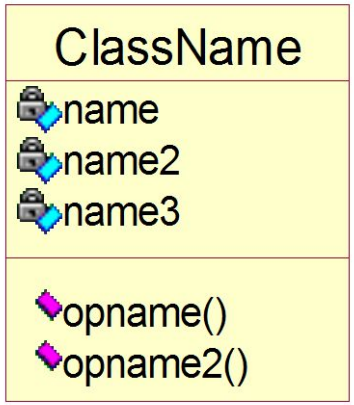
control (менеджер) – класс программной системы, отображающий элементы управления



business entity

business entity (бизнес-сущность)
– класс бизнес-анализа, служит для моделирования предметной области

Модель классов OOP



класс проектирования – класс программной системы, основной вид модели

Лекция 2. Тема 3

Назначение и основные элементы языка UML

UML (сокр. от англ. *Unified Modeling Language* — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения.

UML является языком широкого профиля, это открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой *UML моделью*.

UML был создан для определения, визуализации, проектирования и документирования в основном программных систем.

Использование UML не ограничивается моделированием программного обеспечения. Его также используют для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML позволяет разработчикам ПО достигнуть соглашения в графических обозначениях для представления общих понятий (таких как класс, компонент, обобщение (generalization), объединение (aggregation) и поведение) и больше сконцентрироваться на проектировании и архитектуре.

Лекция 2. Тема 3

Назначение и основные элементы языка UML

- UML позволяет фиксировать решения, принятые при создании различных программных систем.

- UML позволяет отображать и статическую структуру, и динамическое поведение системы.

- В UML имеются конструкции для распределения моделей по пакетам, с помощью которых разработчики могут разделять систему на составные части.

- В UML есть структуры, позволяющие отображать решения по реализации системы и организовывать выполняемые блоки программы в компоненты.

- UML не является языком программирования. Программный код можно получить на основе созданной модели, с другой стороны, на основе исходного кода можно восстановить UML-модели уже существующих программ.

- UML – не формальный язык для доказательства теорем. Он является языком моделирования общего назначения.

- UML – дискретный язык моделирования и не предназначен для разработки непрерывных систем, встречающихся в физике и математике.

Лекция 2. Тема 3

Назначение и основные элементы языка UML

История создания UML

В 1994 году Гради Буч и Джеймс Рамбо, работавшие в компании Rational Software, объединили свои усилия для создания нового языка объектно-ориентированного моделирования. За основу языка ими были взяты методы моделирования, разработанные Бучем (Booch) и Рамбо (Object Modeling Technique — OMT). OMT был ориентирован на анализ, а Booch — на проектирование программных систем. В октябре 1995 года была выпущена предварительная версия 0.8 унифицированного метода (англ. *Unified Method*). Осенью 1995 года к компании Rational присоединился Айвар Якобсон, автор метода Object-Oriented Software Engineering — OOSE. OOSE обеспечивал превосходные возможности для спецификации бизнес-процессов и анализа требований при помощи сценариев использования. OOSE был также интегрирован в унифицированный метод.

На этом этапе основная роль в организации процесса разработки UML перешла к консорциуму OMG (Object Management Group). Группа разработчиков в OMG, в которую также входили Буч, Рамбо и Якобсон, выпустила спецификации UML версий 0.9 и 0.91 в июне и октябре 1996 года.

На волне растущего интереса к UML к разработке новых версий языка в рамках консорциума UML Partners присоединились такие компании, как Digital Equipment Corporation, Hewlett-Packard, i-Logix, IntelliCorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle Corporation, Rational Software, Texas Instruments и Unisys. Результатом совместной работы стала спецификация UML 1.0, вышедшая в январе 1997 года. В ноябре того же года за ней последовала версия 1.1, содержавшая улучшения нотации, а также некоторые расширения семантики.

Последующие релизы UML включали версии 1.3, 1.4 и 1.5, опубликованные, соответственно в июне 1999, сентябре 2001 и марте 2003 года.

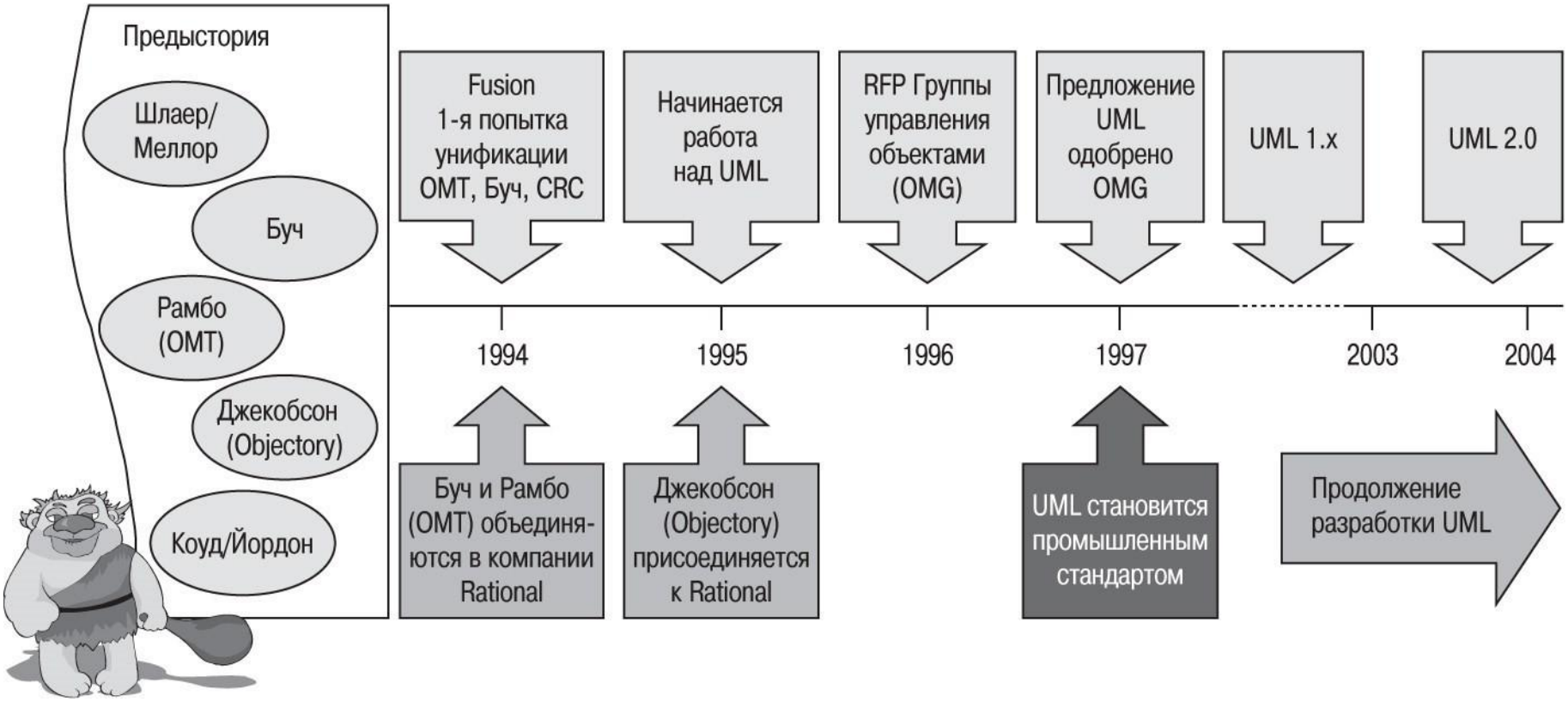
Формальная спецификация последней версии UML 2.0 опубликована в августе 2005 года. Семантика языка была значительно уточнена и расширена для поддержки методологии Model Driven Development — MDD (англ.).

UML 1.4.2 принят в качестве международного стандарта ISO/IEC 19501:2005.

Лекция 2. Тема 3

Назначение и основные элементы языка UML

Этапы создания UML

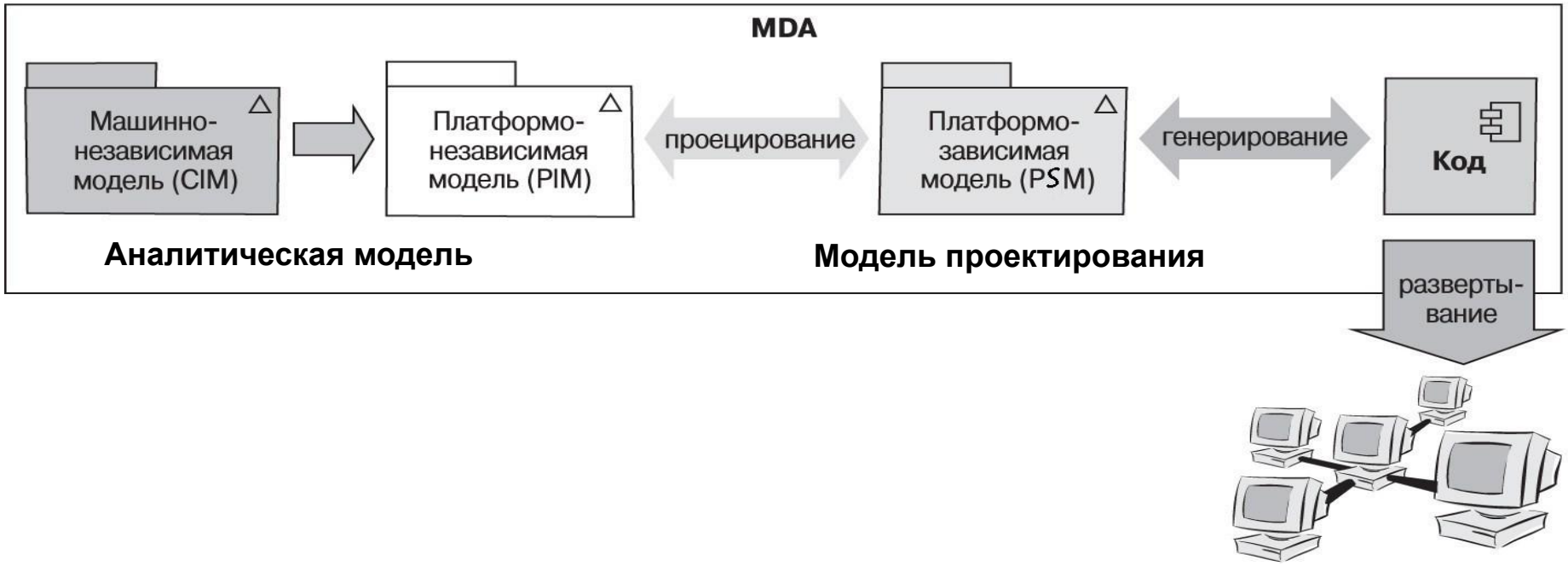


Лекция 2. Тема 3

Назначение и основные элементы языка UML

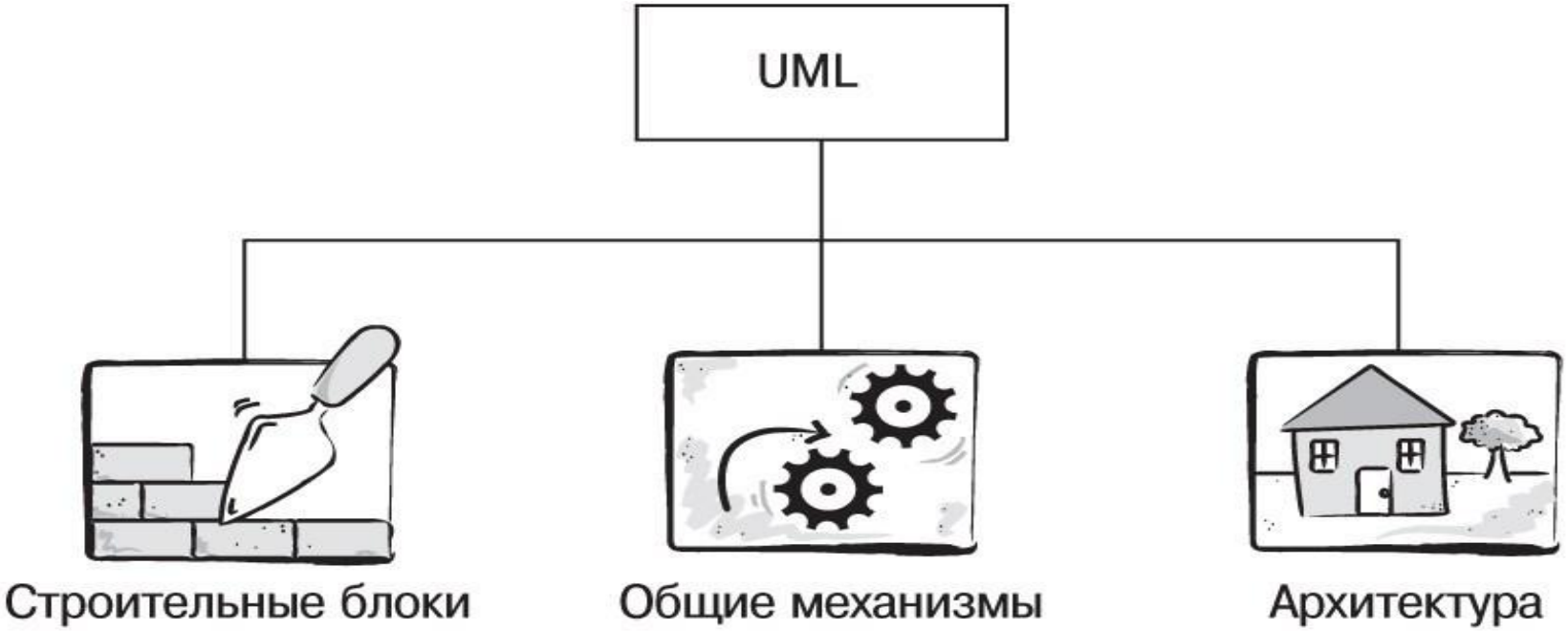
MDA (Model Driven Architecture) - будущее UML

MDA дает видение того, как разрабатывать программное обеспечение на основе моделей. Суть этого видения заключается в том, что модели управляют созданием исполняемой программной архитектуры. В настоящее время уже встречается подобный подход к разработке программного обеспечения, но MDA позволяет точно определить степень автоматизации данного процесса, чего до сих пор удавалось достичь довольно редко. В MDA создание программного обеспечения происходит в результате ряда трансформаций модели при поддержке инструмента моделирования MDA. Абстрактная машиннонезависимая модель (computer-independent model, CIM) используется как основа для платформонезависимой модели (platform-independent model, PIM). PIM трансформируется в платформозависимую модель (platform-specific model, PSM), которая преобразуется в код.

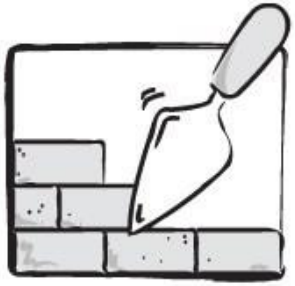


Назначение и основные элементы языка UML

Структура языка UML



- строительные блоки – основные элементы, отношения и диаграммы UML- модели;
- общие механизмы – общие UML-пути достижения определенных целей;
- архитектура – UML-представление архитектуры системы.



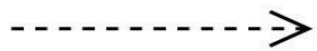
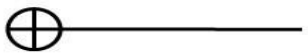
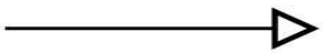
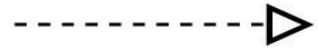
Строительные блоки

Все UML-сущности можно разделить на:

- **структурные сущности – существительные UML-модели, такие как класс, интерфейс, кооперация, прецедент, активный класс, компонент, узел;**
- **поведенческие сущности – глаголы UML-модели, такие как взаимодействия, деятельности, автоматы;**
- **группирующая сущность – пакет, используемый для группировки семантически связанных элементов модели в образующие единое целое модули;**
- **аннотационная сущность – примечание, которое может быть добавлено к модели для записи специальной информации, очень похожее на стикер.**

Отношения

Отношения позволяют показать взаимодействие в пределах модели двух или более сущностей.

Тип отношения	UML-синтаксис		Краткая семантика
	источник	цель	
Зависимость			Исходный элемент зависит от целевого элемента и изменение последнего может повлиять на первый.
Ассоциация			Описание набора связей между объектами.
Агрегация			Целевой элемент является частью исходного элемента.
Композиция			Строгая (более ограниченная) форма агрегирования.
Включение			Исходный элемент содержит целевой элемент.
Обобщение			Исходный элемент является специализацией более обобщенного целевого элемента и может замещать его.
Реализация			Исходный элемент гарантированно выполняет контракт, определенный целевым элементом.

Лекция 2. Тема 3

Назначение и основные элементы языка UML

Общая структура языка UML

Общая структура языка UML позволяет построить статическую и динамическую модели программной системы с точки зрения следующих четырех ее представлений:

Представление (view) – отражение модели поведения или структуры системы с определенной точки зрения

Лекция 2. Тема 3

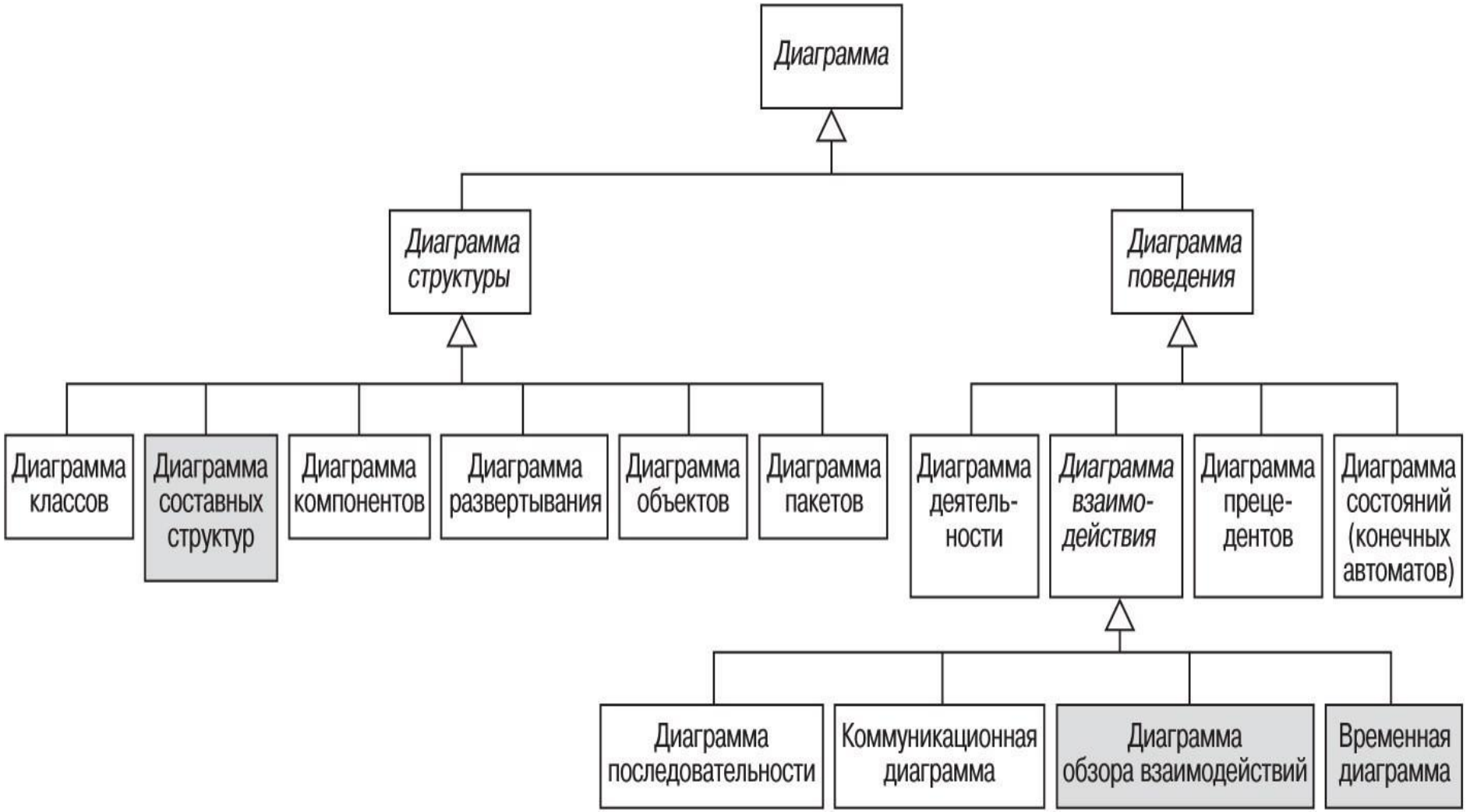
Назначение и основные элементы языка UML

Нотация языка версии 1.5 содержит 8 типов ***канонических диаграмм*** - основное средство разработки *моделей* на языке UML:

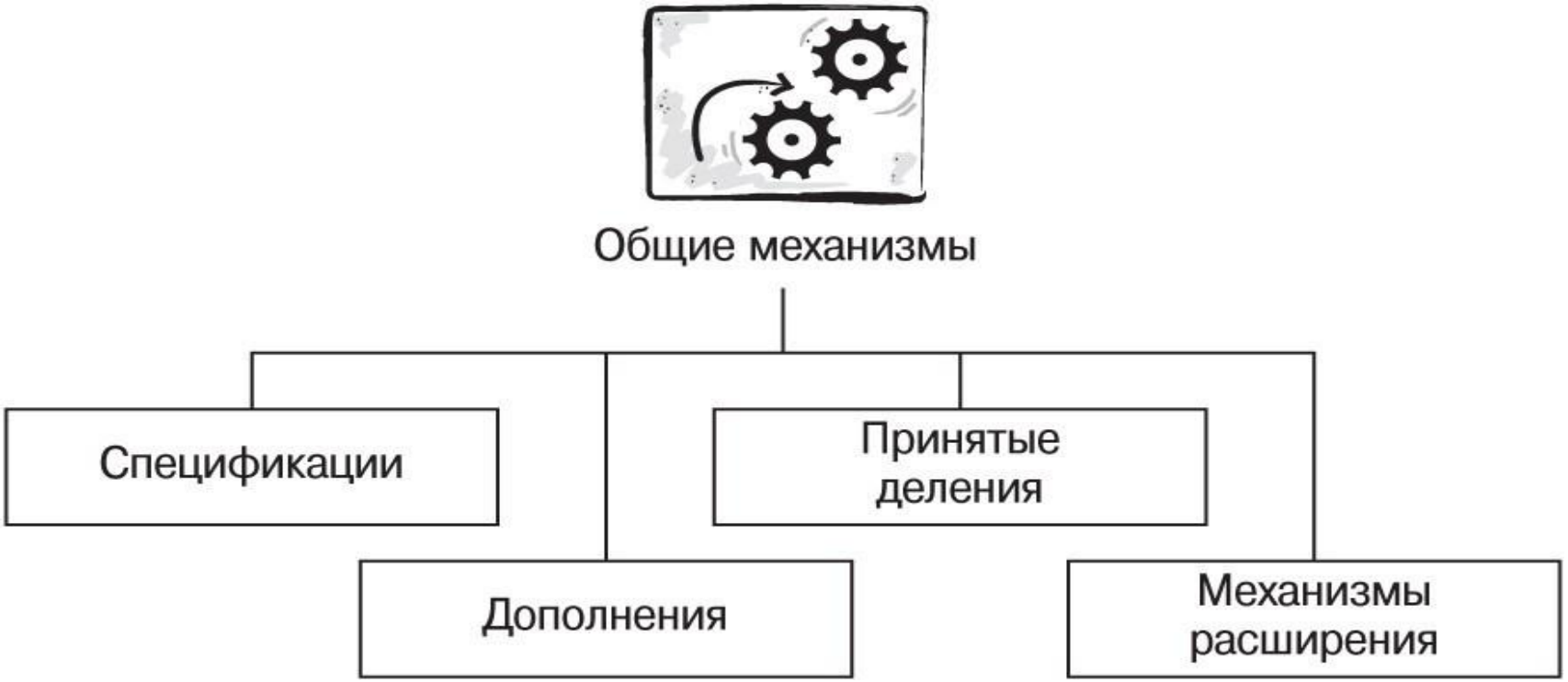
В рамках языка UML все представления о *модели* сложной системы фиксируются в виде специальных графических конструкций, получивших название *диаграмм, которые дополняются «механизмами расширения»*.

Диаграмма *(diagram*) — графическое представление совокупности элементов *модели* в форме связного графа, вершинам и ребрам (дугам) которого приписывается определенная семантика.

Лекция 2. Тема 3



Общие механизмы UML



Спецификации – это текстовые описания семантики элемента

Различные детали базовой модели могут быть раскрыты в виде *дополнений*, расширяющих это минимальное представление

Принятые деления описывают конкретные способы представления мира. В UML существует два принятых деления: классификатор/экземпляр и интерфейс/реализация.

Лекция 2. Тема 3

Назначение и основные элементы языка UML

Механизмы расширения

К механизмам расширения относятся:

- *«**стереотипы**» (stereotype)* – новый вид элемента модели, который обладает той же структурой, что и существующий элемент, но имеет дополнительные ограничения, другую интерпретацию и пиктограмму
- *«**именованные (помеченные) значения**» (tagged value*) – текстовая информация, прикрепляемая к любым типам элементов модели, представляющее пару «имя – значение». Имя в именованном значении называют тегом (tag).

Именованные значения на *диаграммах* изображаются в форме строки текста, заключенного в фигурные скобки. При этом используется следующий формат записи: {тег = значение}. Теги встречаются в нотации языка UML, но их определение не является строгим, поэтому они могут быть указаны самим разработчиком.

- ***язык ограничений OCL*** *(Object Constraint Language)*, который является составной частью языка UML.

Ограничение *(constraint)* — некоторое логическое условие, ограничивающее семантику выбранного элемента модели.
Как правило, все *ограничения* специфицируются разработчиком, на *диаграммах* изображаются в форме строки текста, заключенного в фигурные скобки.

Лекция 3. Тема 3

Аналитическая модель

Что нужно знать и понимать, приступая к разработке любой программной системы?

Аналитическая модель

- как работает бизнес заказчика
- проблемы, которые нужно решить, и цели, которые планируется достигнуть с помощью программной системы
- пользователей и требования, которым должна удовлетворять программная система

Аналитическая модель

Чтобы понять как работает бизнес заказчика нужно:

- Определить понятия и сущности предметной области (*моделирование сущностей предметной области, их атрибутов и при необходимости их взаимосвязей*)
- Выявить заинтересованных лиц и определить бизнес-процессы (обязанности, потребности или сервисы), которые они используют (*моделирование действующих лиц и бизнес-прецедентов*)
- Описать бизнес-процессы (*моделирование бизнес- процессов*)

Аналитическая модель

Аналитическая модель – это точное, четкое представление задачи, позволяющее отвечать на вопросы и строить решения.

На этапе проектирования должна использоваться аналитическая модель, а не исходная формулировка задачи.

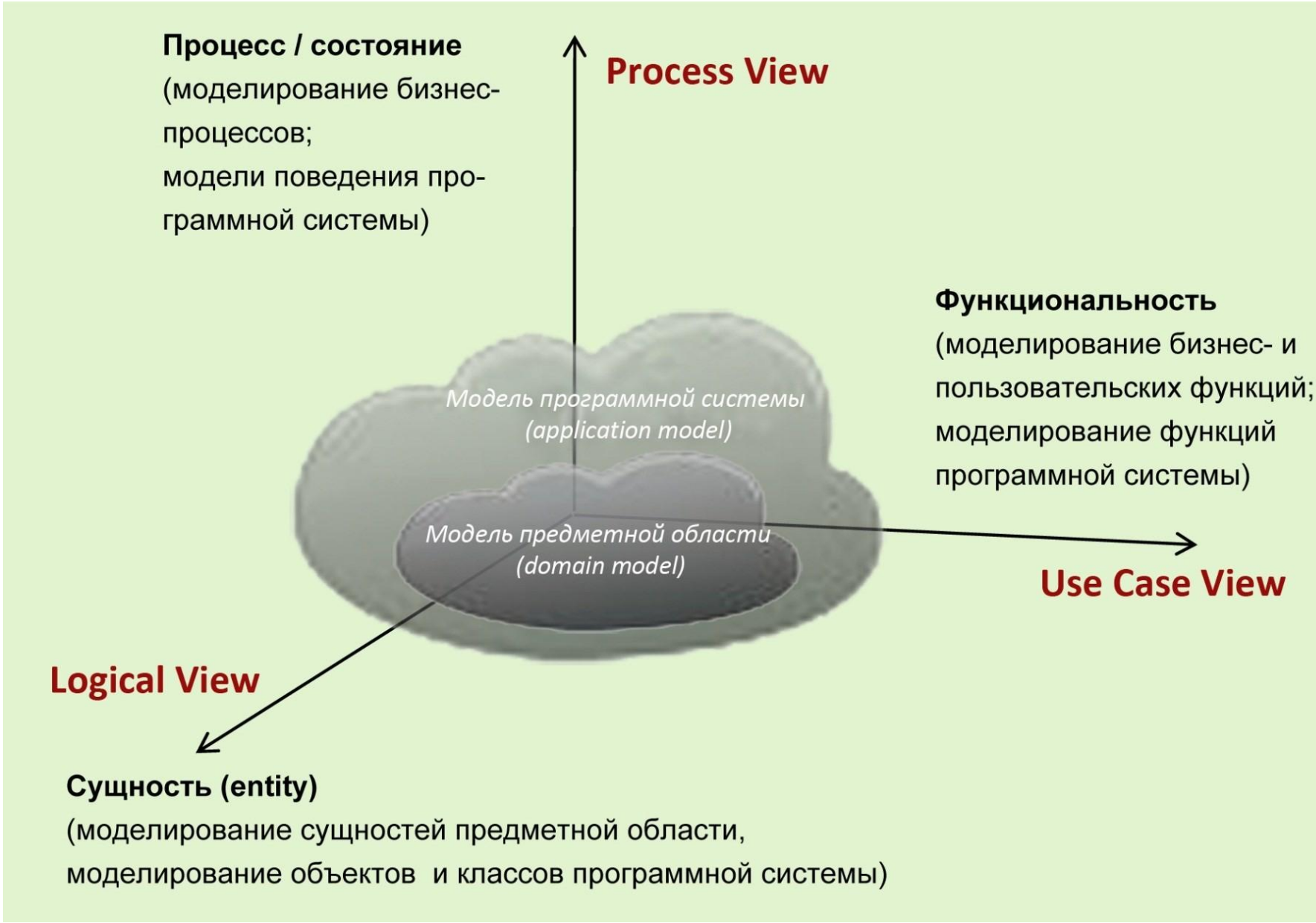
Аналитическая модель включает

- Модель предметной области (domain model);
- Модель программной системы (application model).

Представления аналитической модели

1. **Представление классов (Logical View)**. Моделируем: сущности предметной области (business entity), классы анализа (boundary, entity, controll);
2. **Представление процессов & состояний (Process View).** Моделируем: бизнес-процессы, последовательности действий в вариантах использования, алгоритмы операций;
3. **Представление прецедентов (Use Case View).** Моделируем:

Аналитическая модель



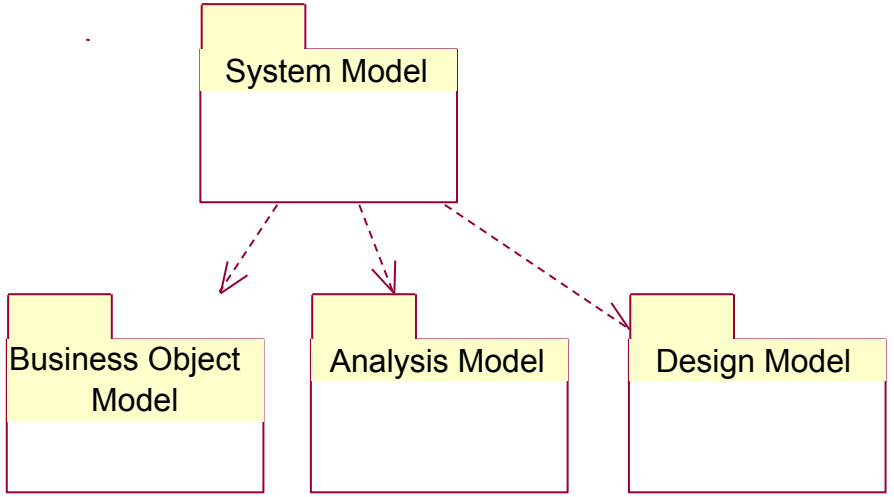
Лекция 3. Тема 4

Диаграммы UML

Организационная структура проекта (модели)

Диаграмма пакетов (package diagram)

Пакет (*package*) – базовый элемент модели, служащий для группировки элементов модели. В Package Diagram реализуется принцип модульности сложных программных систем.



В версии UML 1.5 Rational Rose 2003 диаграмма содержит два основных элемента: собственно пакет (Package) и единственную возможную связь между пакетами – зависимость (dependency), которая указывает на тот пакет, который вложен в данный.

Лекция 3. Тема 4

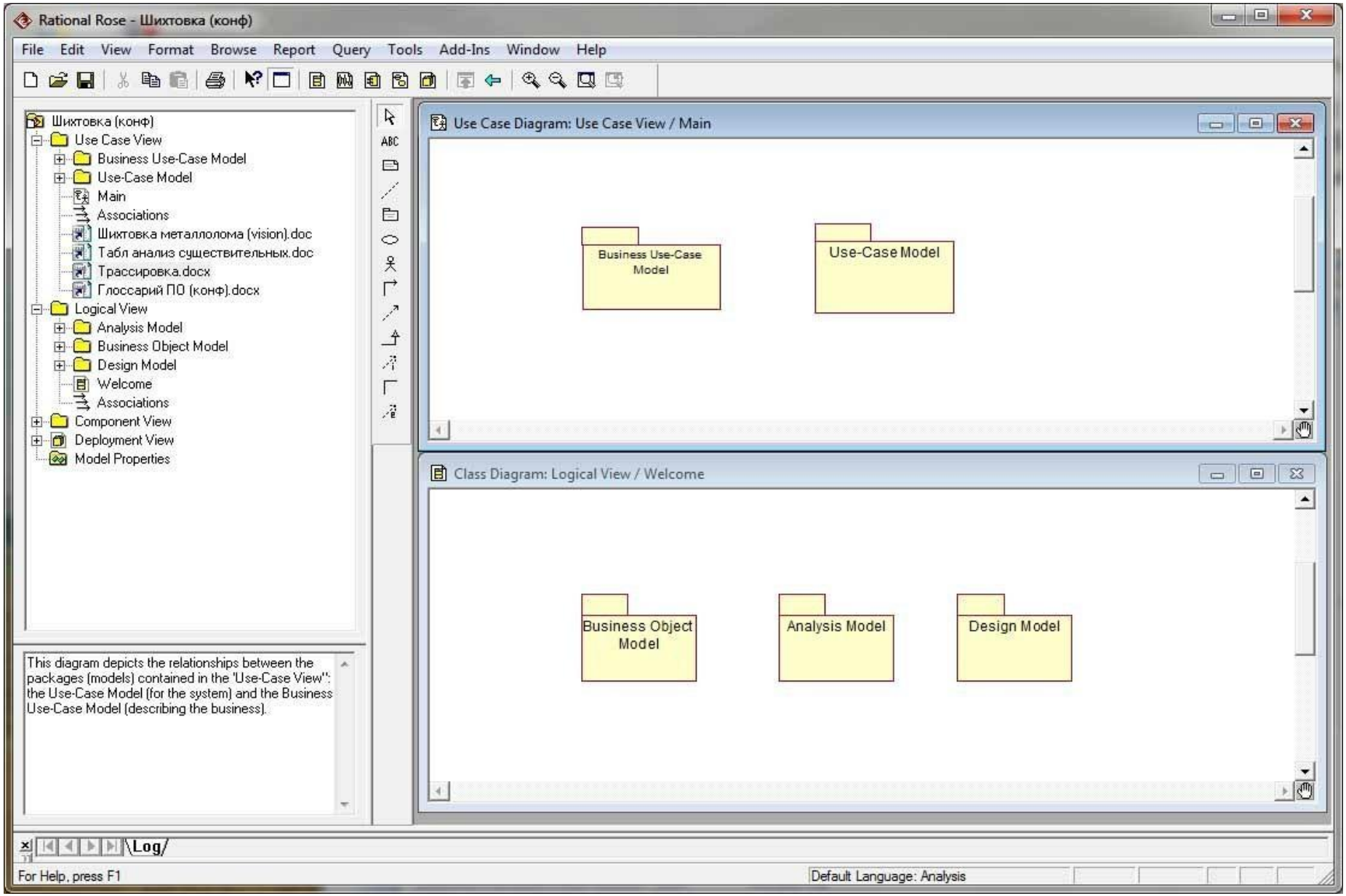
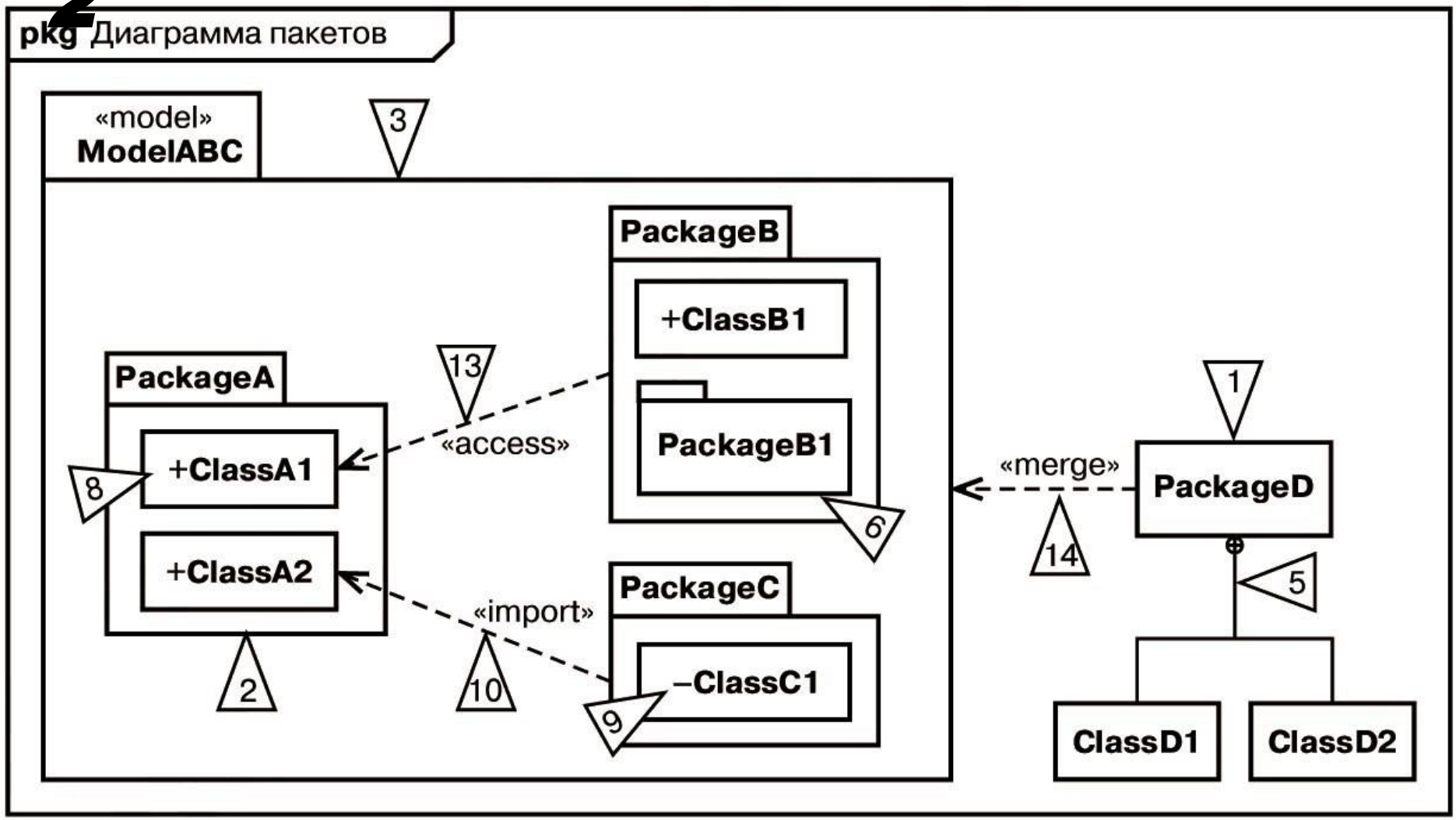


ДИАГРАММА ПАКЕТОВ
(package diagram)

UML-2



Лекция 3. Тема 4

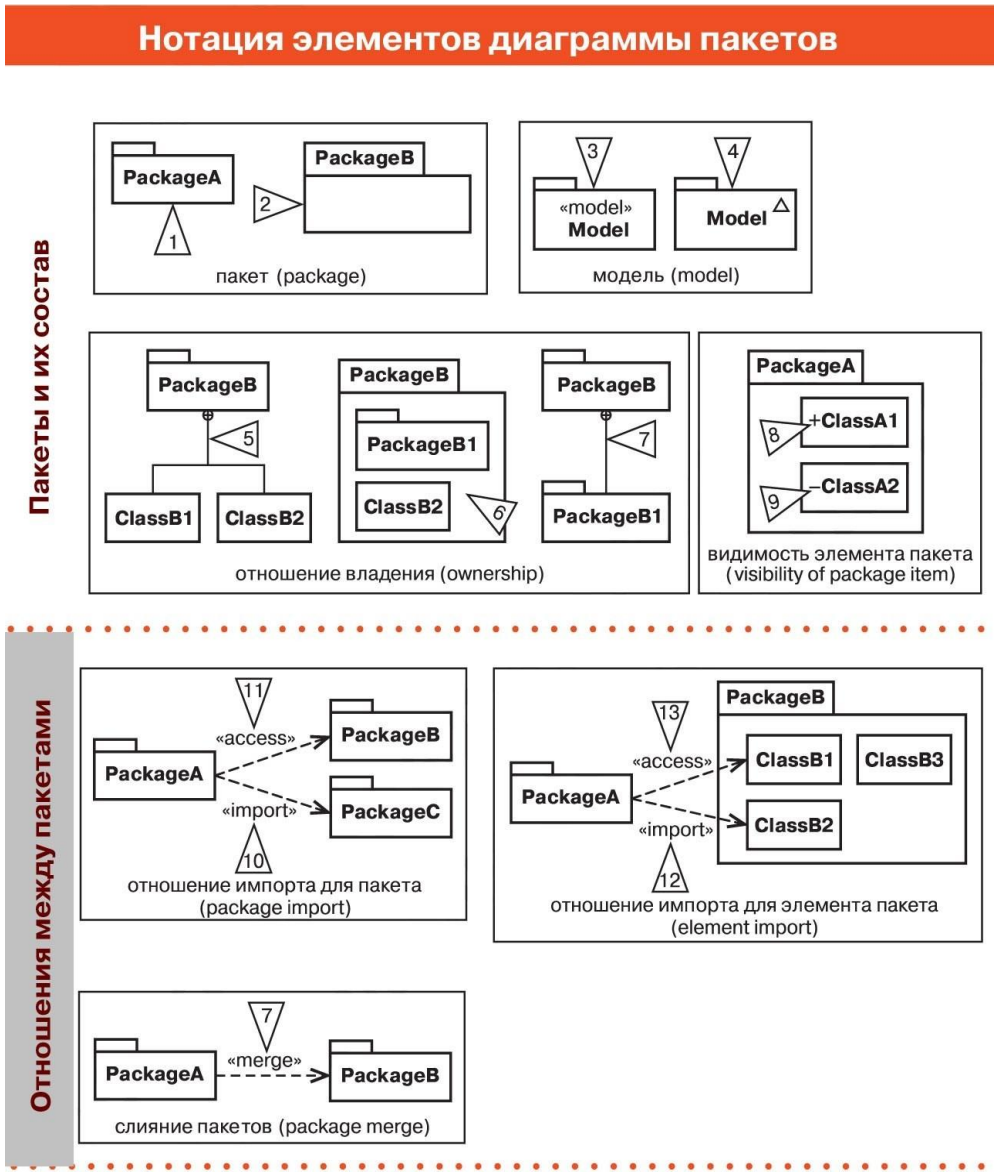
Основные сущности, используемые на диаграмме: **пакеты (1 и 2)**; их частные случаи, имеющие специальную нотацию **(3 и 4)**; пакеты, имеющие внутреннюю структуру, т.е. содержащие в себе другие пакеты и/или классы, которые показываются через **отношение владения (5, 6 и 7)**.

Для элементов пакета может быть указана **видимость: открытая (8)** или **закрытая (9)**.

Пакеты могут включать в себя элементы других пакетов, используя различные отношения между пакетами. Включение всех элементов одного пакета в другой возможно с помощью **отношения импорта**, которое существует в двух вариантах **(10 и 11)**.

Если требуется включить в пакет только некоторые элементы другого пакета, то варианты **отношения импорта (12 и 13)** можно применить только к ним.

Более сложное **отношение слияния (14)**



Лекция 3. Тема 5

Моделирование предметной области (domain model)

1. Моделирование предметной области является в RUP опциональной и выполняется для самих разработчиков, Заказчик является экспертом предметной области и не нуждается в ее моделировании.
2. Целью моделирования предметной области является выработка точной, четкой, доступной для понимания и, наконец, корректной модели реального мира.
3. В зависимости от конкретного проекта по усмотрению бизнес-аналитиков при анализе ПО могут создаваться следующие артефакты: глоссарий терминов предметной области, диаграмма бизнес-сущностей (классов ПО), диаграммы взаимодействия объектов ПО (моделирование связей), диаграммы бизнес-процессов или состояний, диаграммы функционеров (бизнес-актеров) и их функциональные обязанности, модели, описывающие состояния и условия перехода из одного в другое.

Лекция 3. Тема 5

Моделирование предметной области (domain model)

Диаграмма сущностей предметной области (business entity)

Цель: выявление и документирование бизнес-сущностей и их атрибутов.

Бизнес-сущность — некий объект ПО или концептуальное понятие ПО, характеризуемое набором данных (существенных признаков), прямо или косвенно связанное с проектируемой программной системой.

Бизнес-сущности – это кандидаты в классы и объекты ПС, отвечающие за ввод, изменение, удаление и хранение данных (атрибутов).

Методика выявления бизнес-сущностей:
чтение текста концепции и анализ имен существительных.

ПРИМЕР анализа текста:

Инструментом для этого служит ***КРАН***, который забирает ***МЕТАЛЛИЧЕСКИЕ КОМПОНЕНТЫ*** (МК) своим *магнитом* и потом опускает в дозировочный **КОНТЕЙНЕР**. В соответствии с ***ВЕСОМ НАБРАННЫХ МК*** подаются ***КОКС, ИЗВЕСТЬ, СПЕЦКОКС***. По заполнении контейнера его содержание со всеми занесенными в него металлическими компонентами направляется в **ЧАН** для плавления, в который добавляются ***ЛЕГИРУЮЩИЕ ДОБАВКИ***. В конце рабочего дня чан направляется в **ПЛАВИЛЬНУЮ ПЕЧЬ**. Так как в процессе плавления должны использоваться определенные соотношения веса различных МК, программа должна обеспечить требуемый состав **ПЛАВИЛЬНОЙ СМЕСИ**. Этот контроль и составление смеси компонентов для плавки называется ***ШИХТОВКОЙ***.

Программная система устанавливается на рабочем месте ***КРАНОВЩИКА*** и используется им для контроля над процессом шихтовки. Кроме того, она позволяет ***ТЕХНОЛОГУ*** задать определенную ***ПРОПОРЦИЮ МК*** в конечном ***СПЛАВЕ*** и осуществлять сохранение **информации** о составе и свойствах

базе данных

Моделирование бизнес-актеров и бизнес-функций

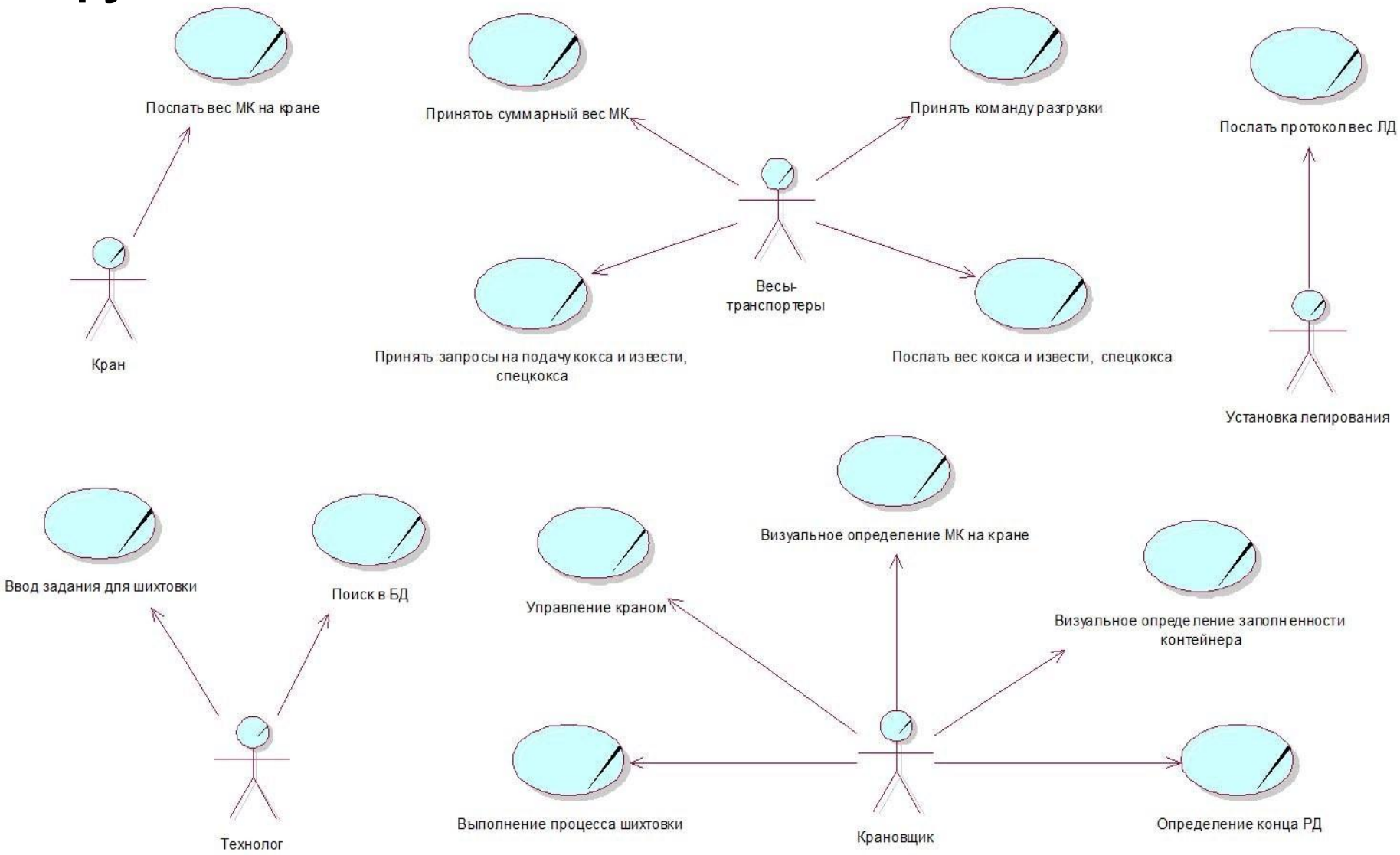
Модель бизнес-актеров и их функциональных обязанностей (бизнес-функций) строится в представлении Use Case View в папке Business Use Case Model (при вызове шаблона rational unified process) на диаграмме ВИ.

Цель моделирования: изучение действующих лиц ПО и их функциональных обязанностей для понимания бизнес-процессов, выявления пользователей системы, уточнения высокоуровневых бизнес-целей.

Business Actor — это штатная единица (действующее лицо, функционер) в предметной области, который взаимодействует с ПС, либо не взаимодействует, но является участником бизнес-процесса. Чаще всего это штатный работник предприятия или должность (генеральный директор, главный инженер, экономист, зам. директора по производству и т.д.), но может быть и обособленной штатной системой, исполняющей определенные функциональные обязанности (например: кран, весы- транспортеры и т.д.).

Business Use Case — это функциональные (должностные) обязанности, возложенные на данного функционера. Фиксируются только те из

ПРИМЕР диаграммы бизнес-актеров и бизнес-функций :



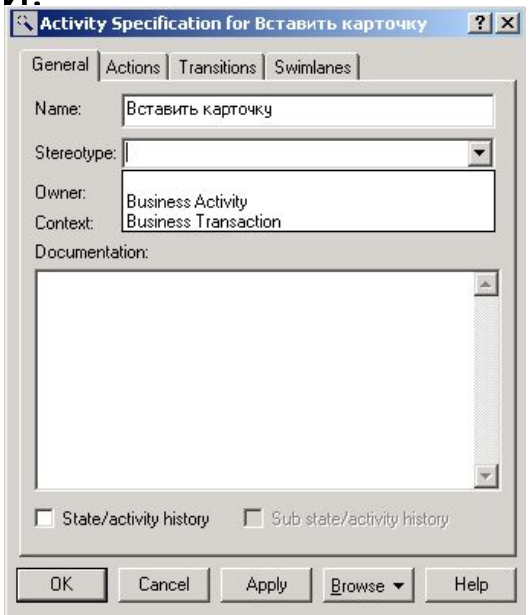
Лекция 4. Тема 6

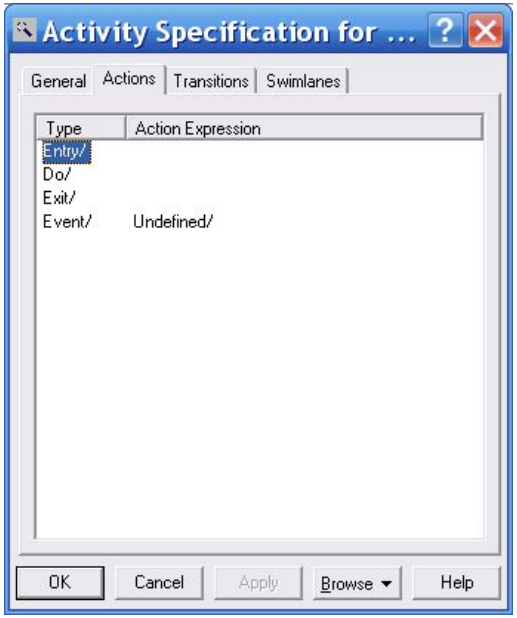
Моделирование бизнес-процессов

Диаграммы деятельности (activity diagram)

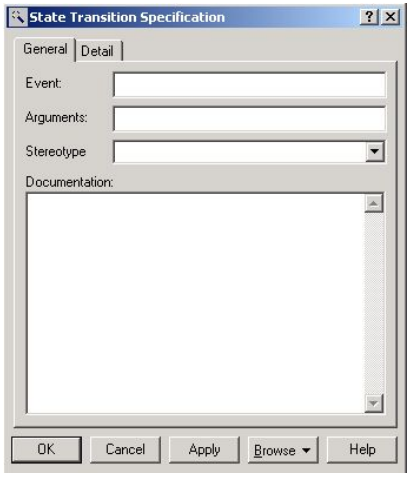
Activity – действие, деятельность, **состояние деятельности** (*activity, action state*). **State Transition** – переход от одной деятельности к другой, предполагает, что входная деятельность уже завершилась.
Обладает спецификациями, в которые входят: **Event** (событие, это то, что вызывает переход), **Arguments** (аргументы), **Stereotype** (стереотип) и **Documentation**.
Состояние деятельности не может иметь внутренних переходов, поскольку оно является элементарным и не может быть прервана никакими внешними событиями.

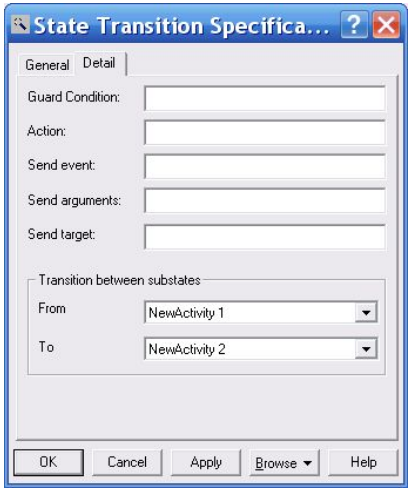
Action – отдельное действие, совершаемое в рамках данной activity.
Entry – действие, совершаемое на входе в activity;
Do – действие, совершаемое в процессе activity;
Exit – действие, совершаемое



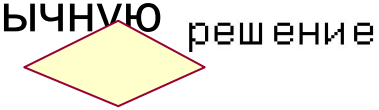


Лекция 4. Тема 6





Спецификация State Transition.
На вкладке Detail вводятся дополнительные **Guard Condition** ограждающие условия (определяют, когда переход может быть выполнен, а когда нет), действия и посылаемые события.
Для отображения события на диаграмме можно использовать как имя операции, так и обычную фразу.

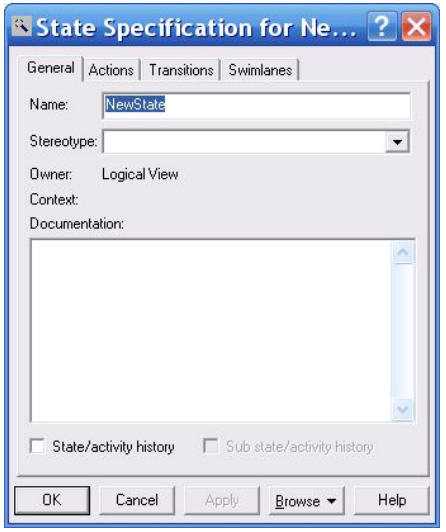


Decision (решение) показывает зависимость дальнейшей работы от внешних условий или решений.

Если из *состояния действия* выходит единственный переход, то его можно никак не помечать. Если же таких переходов несколько, то при моделировании последовательной деятельности запускается только один из них. В этом случае для каждого из таких переходов должно быть явно записано собственное граничное условие в прямых скобках. При этом для всех выходящих из некоторого состояния деятельности переходов должно выполняться требование истинности только одного из них. Подобный случай встречается тогда, когда последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения промежуточного результата.

Лекция 4. Тема 6

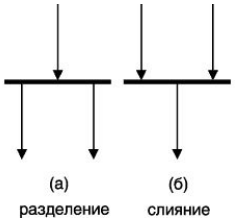
Инструмент **State** – состояние предназначен для обозначения ситуации ожидания, например, в течение жизни объекта, когда объект ожидает некоторое событие или находится в некотором состоянии. Спецификация State аналогична спецификации Activity.



Инструмент **Start State** – начальное состояние, в котором система находится сразу после включения, либо объект сразу после своего создания, либо начальные условия use case. Начальное состояние обязательно. На диаграмме может быть только одно начальное состояние.

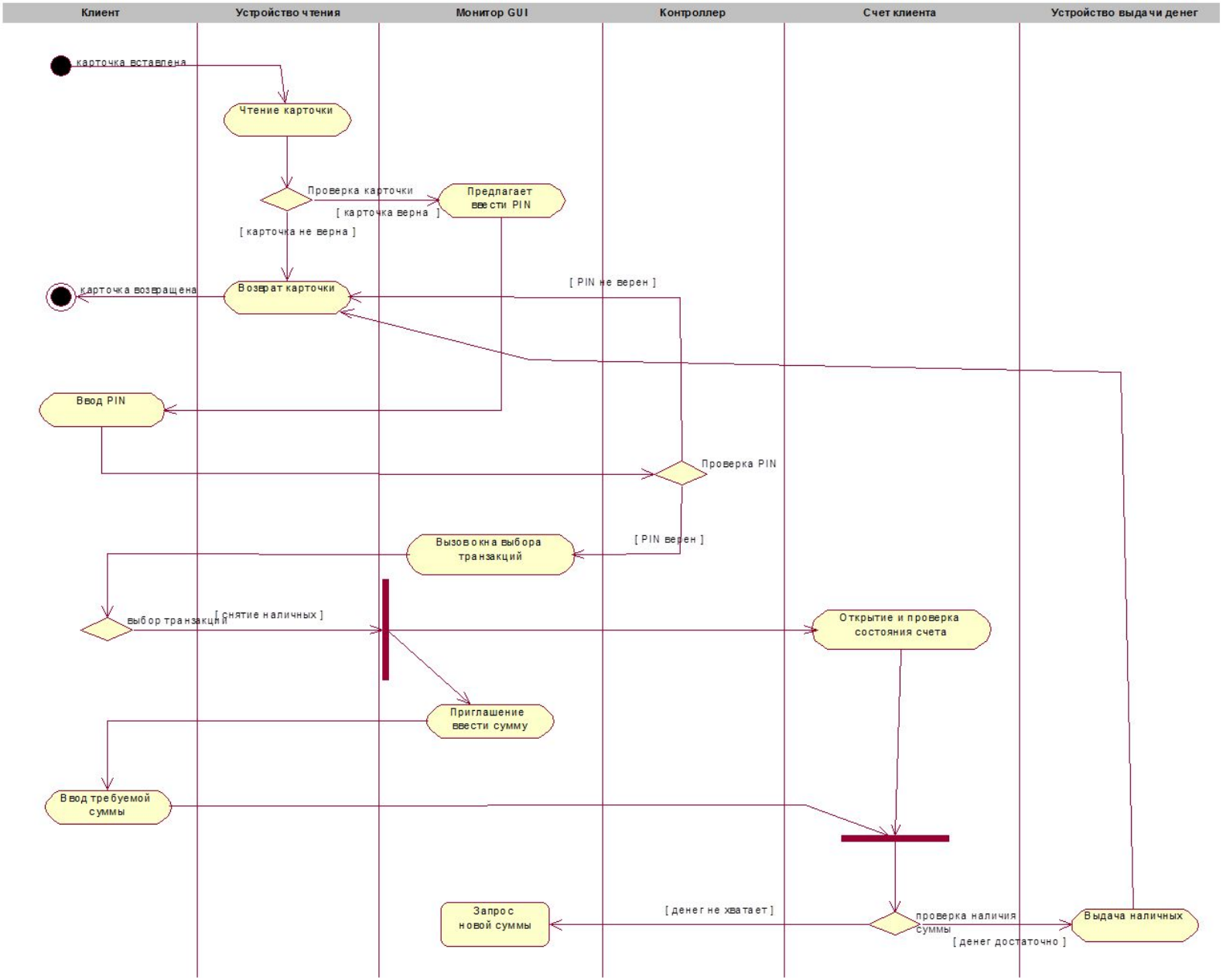
Инструмент **Stop State** – конечным состоянием называется то в котором система находится непосредственно перед выключением, либо объект перед уничтожением, либо use case перед завершением. Конечное состояние не является обязательным, их может быть сколько угодно.

Инструмент **Synchronization** – синхронизация позволяет определить независимо выполняемые переходы. При этом переходы могут разделяться на несколько выполняемых независимо (разделение) или, наоборот, несколько входящих переходов будут сливаться в один исходящий.

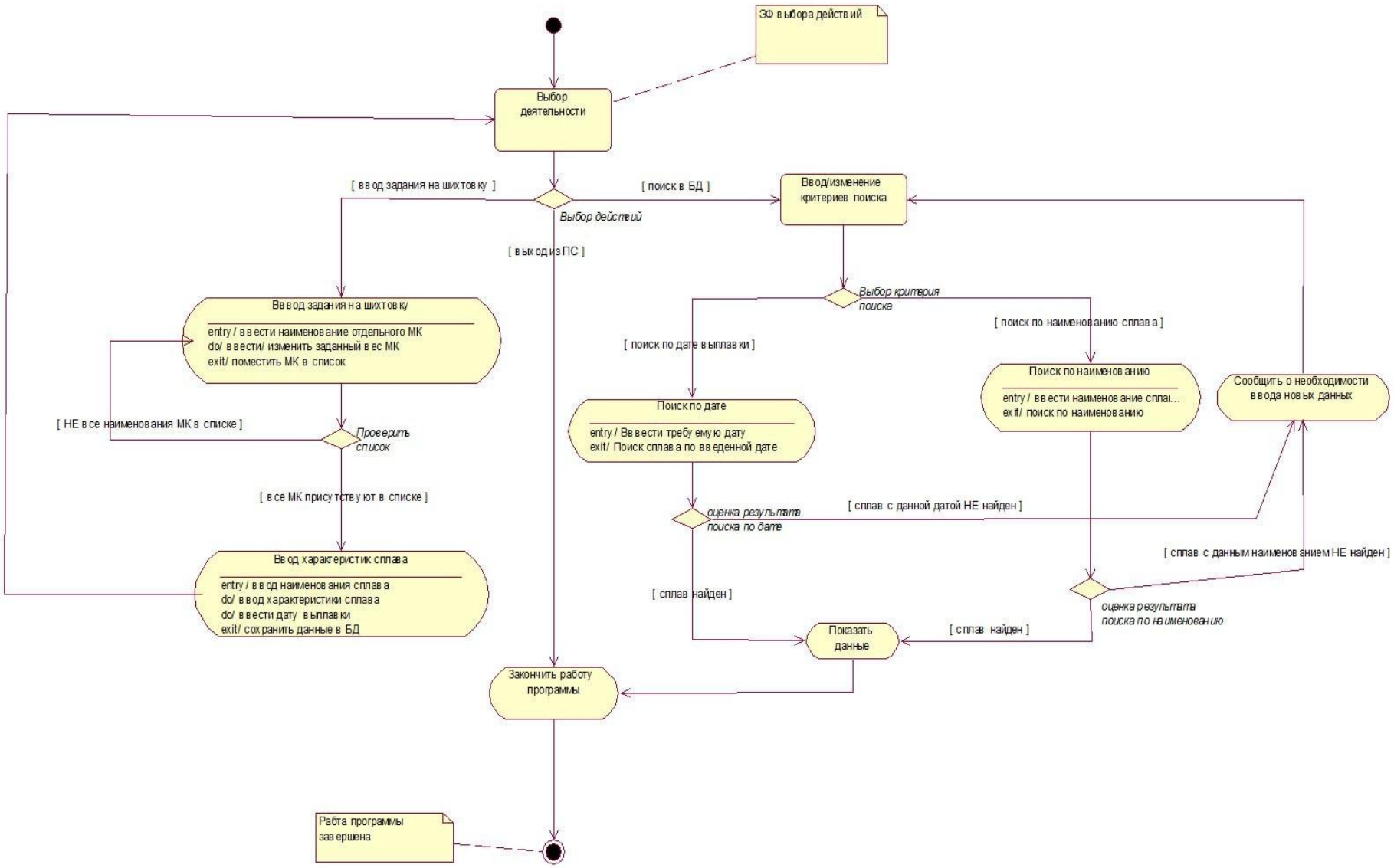


Инструмент **Swimlanes** (плавательные дорожки) -- позволяет моделировать последовательность действий различных объектов и связи между ними. При помощи этого элемента можно моделировать бизнес-процессы организации, отражая на диаграмме различные подразделения и объекты, играющие важные роли в модели бизнеса. Позволяет показать, кто выполняет те или иные роли в процессе.

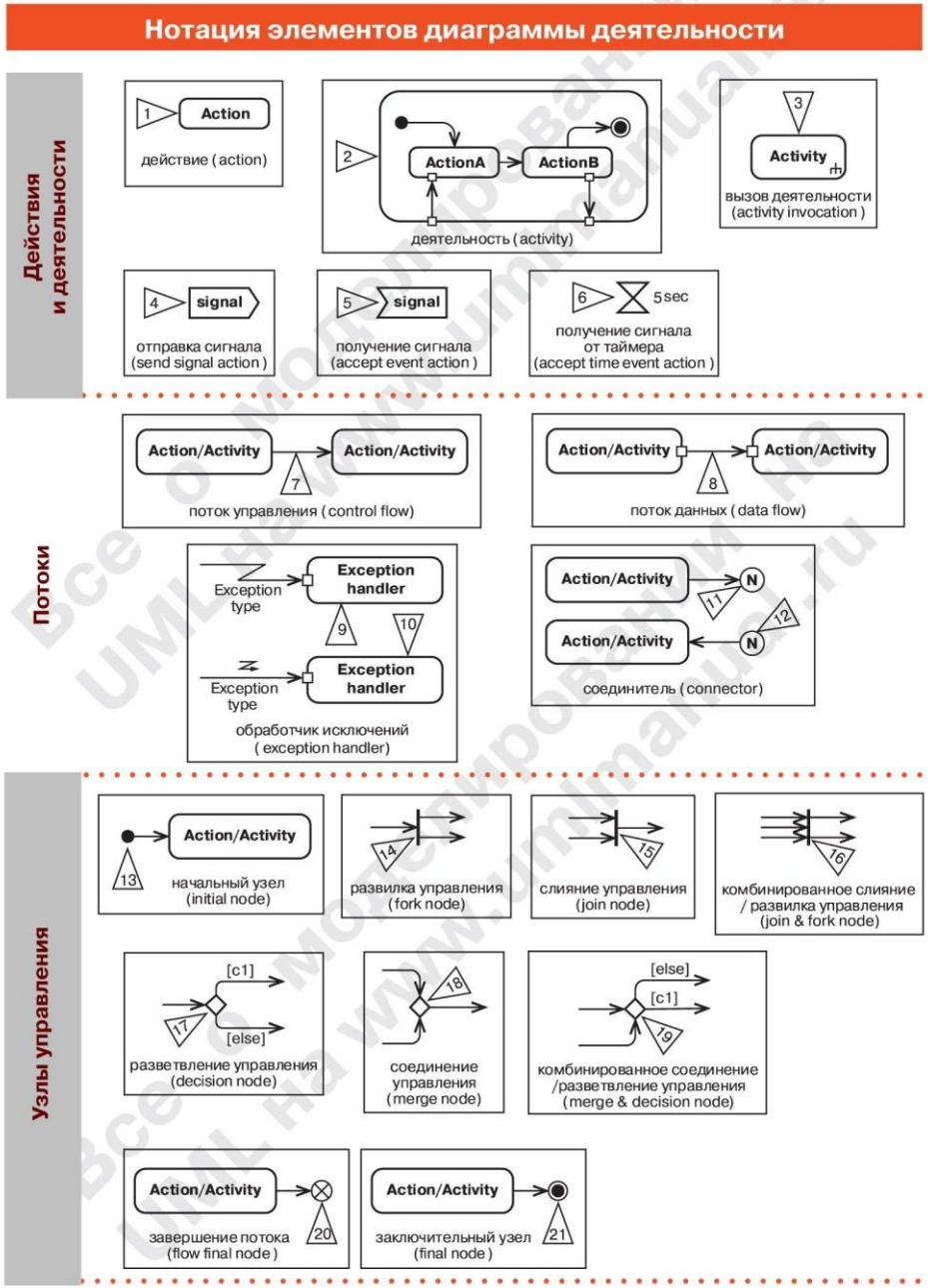
Лекция 4. Тема 6. Примеры activity diagram

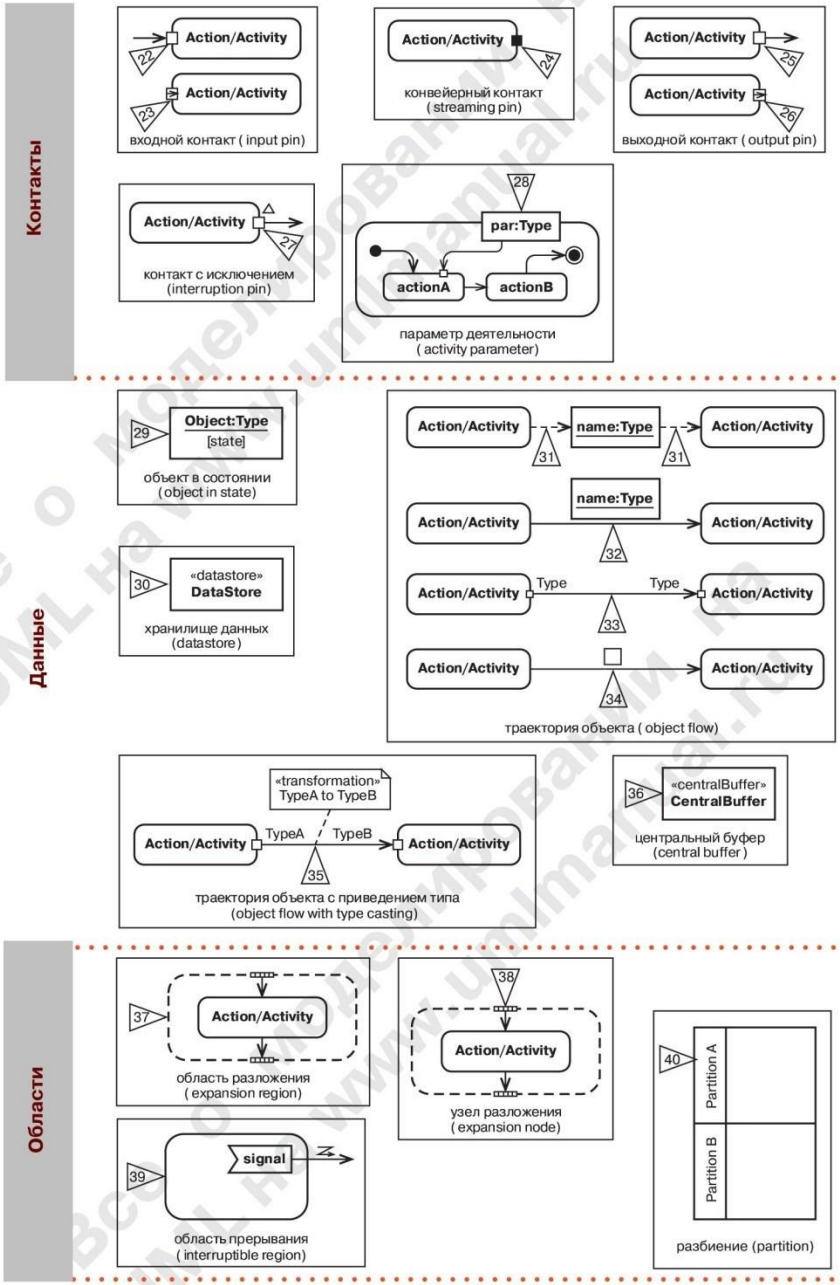


Лекция 4. Тема 6. Примеры activity diagram



Лекция 4. Тема 6. Нотация UML-2 activity diagram



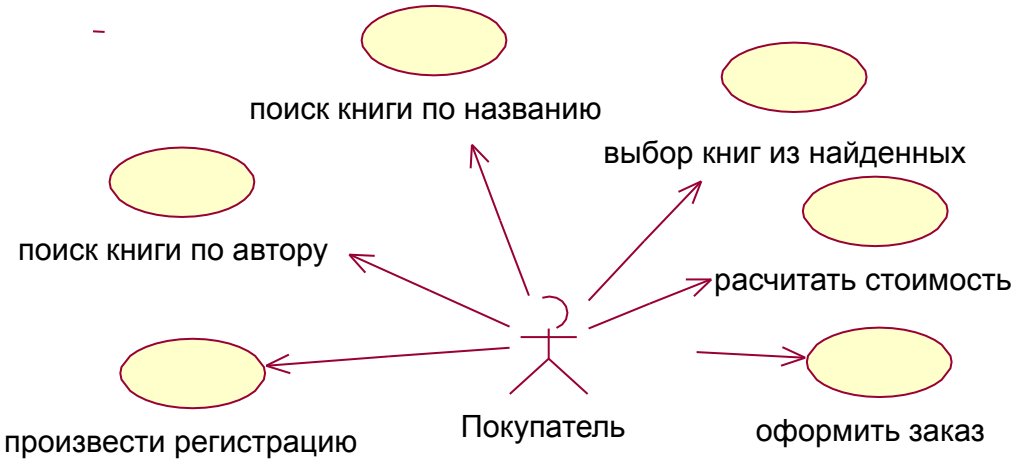


Лекция 4. Тема 4

Диаграммы UML

Диаграмма вариантов использования (use case diagram)

Это диаграмма на которой представлены отношения между действующими лицами и вариантами использования.



Вариант использования (прецедент, Use Case) — внешняя спецификация последовательности действий, которые **система** может выполнять в процессе взаимодействия с **действующими лицами** (актер, актант, actor) для получения определенного значимого для них результата.
Вариант использования служит для описания сервисов, которые система предоставляет *актеру*, другими словами каждый *вариант использования* определяет набор действий, совершаемый системой при диалоге с *актером*.

Действующее лицо (актант, актер, actor) – абстрактное ролевое описание внешнего пользователя (или нескольких пользователей), находящегося вне системы и взаимодействующего с ней.
Актант может быть трех типов: быть одной из ролей конкретного физического лица, быть одной из ролей внешней удаленной системы (подсистемы) и исполнять роль временного таймера (момент времени, временной интервал).

Лекция 4. Тема 4

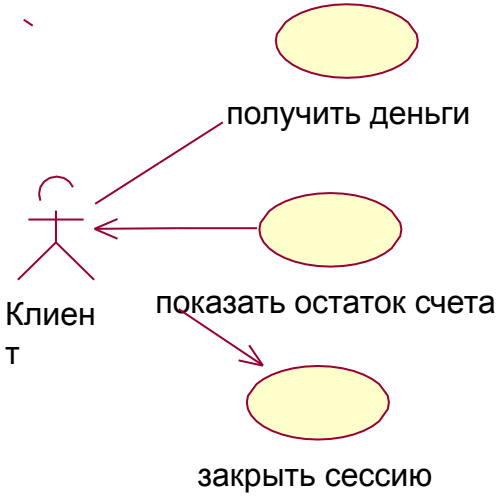
Диаграмма вариантов использования (use case diagram)

Диаграмма вариантов использования иллюстрирует функциональные требования к системе, она показывает, **что** и **для кого** должна делать система.

Цели при создании диаграмм вариантов использования

- определение общих границ моделируемой предметной области;
- документирование общих требований к функциональному поведению системы;
- определение круга пользователей системы и их связей с системой;
- подготовка исходной документации для взаимодействия разработчиков системы с ее заказчиками.

Отношения на диаграммах вариантов использования



Ассоциация *(association relationship)* – единственное возможное отношение между актером и прецедентом. Каждая ассоциация подразумевает наличие взаимодействия и соответственно канала связи и интерфейса (граничного объекта, *boundary*) между актером и программной системой.

Ассоциация бывает двунаправленной (сообщение посылается от актера к системе и от системы к актеру), а также однонаправленной (изображается линией со стрелкой). В случае, если стрелка направлена в сторону варианта использования, то это означает, что актер инициирует исполнение данного прецедента. Если стрелка направлена к актеру, то это показывает, что он получает от системы справочную информацию.

Ассоциация может иметь некоторые дополнительные обозначения, например, имя и кратность

Лекция 4. Тема 4

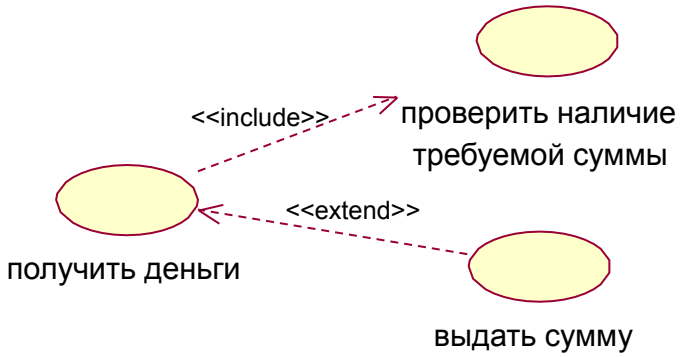
Диаграммы UML

Отношения на диаграммах вариантов использования (продолжение)

Один *актер, используя несколько ассоциаций,* может взаимодействовать с несколькими *вариантами использования*. В этом случае этот *актер* обращается к нескольким сервисам данной системы. В свою очередь один *вариант использования* может взаимодействовать с несколькими *актерами*, предоставляя для всех них свой сервис.

Отношения между вариантами использования

Варианты использования, определенные в рамках одной моделируемой системы, могут также взаимодействовать между собой. Однако, характер этого взаимодействия отличается от взаимодействия с актерами и наличия граничного объекта при этом не подразумевается.



Включение *(include relationship)* **--** каждый экземпляр первого *варианта использования* всегда включает в себя функциональное поведение или выполнение второго *варианта использования*. В этом смысле поведение второго *варианта использования* является частью поведения первого *варианта использования*. Графически данное отношение обозначается пунктирной линией со стрелкой, направленной от базового *варианта использования* к включаемому *варианту использования*, которая помечается стереотипом <<include>>.

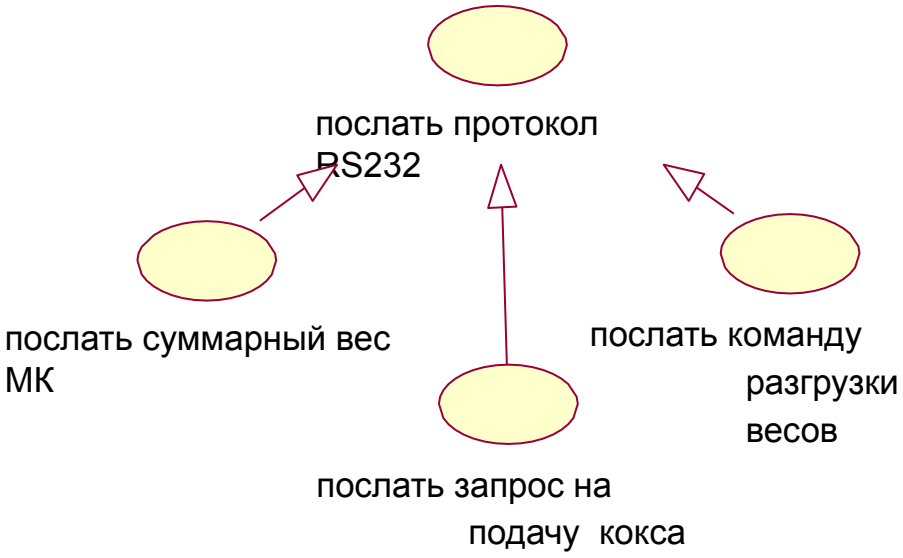
Расширение *(extend relationship)* -- определяет взаимосвязь базового *варианта использования* с другим *вариантом использования*, функциональное поведение которого задействуется базовым не всегда, а только при выполнении дополнительных условий. Графически обозначается в форме пунктирной линии со стрелкой, направленной от *расширяющего прецедента* к базовому *варианту использования* и помеченой стереотипом <<extend>>

Лекция 4. Тема 4

Диаграммы UML

Отношения на диаграммах вариантов использования (продолжение)

Семантика отношения ***расширения*** *(extend relationship)* определяется следующим образом. Если базовый *вариант использования* выполняет некоторую последовательность действий, которая определяет его поведение, и при этом имеется точка *расширения* на экземпляр другого *варианта использования*, которая является первой из всех точек *расширения* у базового *варианта*, то проверяется логическое условие данного отношения. Если это условие выполняется, исходная последовательность действий расширяется посредством *включения* действий другого *варианта использования*. Следует заметить, что условие отношения *расширения* проверяется лишь один раз — при первой ссылке на точку *расширения*, и если оно выполняется, то все расширяющие *варианты использования* вставляются в базовый *вариант*.



Обобщение *(generalization relationship)* – аналогично наследованию и применяется в том случае, когда необходимо отметить, что дочерние *варианты использования* кроме присущего им специфического поведения обладают всеми особенностями поведения родительских *вариантов использования.* Стрелка отношения обобщения указывает на родительский вариант использования

Лекция 4. Тема 4

Диаграммы UML

Стереотипы для моделирования бизнес-актеров и бизнес-функций

Моделированием функционеров предметной области (бизнес-актеров), их функциональных обязанностей, а также бизнес-процессов выделяется в отдельный раздел аналитики – бизнес-анализ. В представлении use-case view в соответствии с USDP создается папка business use-case, а для моделирования предметной области используются следующие стереотипы:

Бизнес-актер *(business actor)* – индивидуум (штатная должность), группа, организация или система, которые взаимодействуют с моделируемой бизнес-системой, но не являются ее частью. Примерами *бизнес-актеров* являются клиенты, покупатели, поставщики, партнеры. Общее свойство *бизнес-актеров* состоит в том, что они являются инициаторами или клиентами бизнес-процессов моделируемой системы.

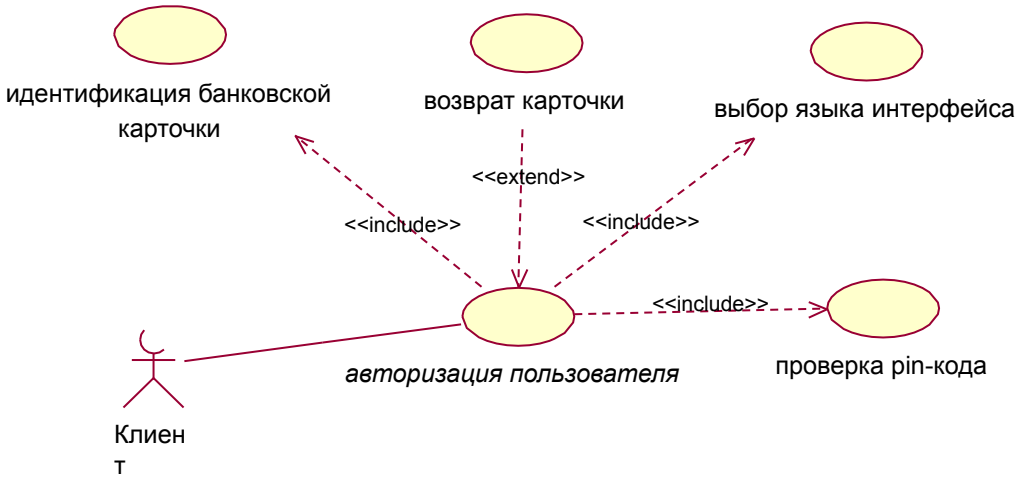
Сотрудник *(business worker)* – индивидуум (штатная должность), группа, организация или система, которые действуют внутри моделируемой бизнес-системы, взаимодействуют с другими *сотрудниками* и являются участниками бизнес-процесса моделируемой системы. Примерами *сотрудников* являются менеджеры, администраторы, кассиры, инженеры. Общее свойство *сотрудников* заключается в том то, что они являются субъектами и входят в состав моделируемой системы.

Бизнес-вариант использования *(business use case)* – определяет последовательность действий моделируемой системы, направленную на выполнение отдельного бизнес-процесса.

Лекция 4. Тема 4

Диаграммы UML

Абстрактные варианты использования



Если собственных действий и функциональности, то он называется абстрактным и его название пишется курсивом.

Лекция 5. Тема 5

Использование вариантов использования для формализации функциональных требований

Требования и их классификация

Требование *(requirement)* -- желательное свойство, характеристика или условие, которым должна удовлетворять система в процессе своей эксплуатации.

Применительно к программным системам предложена следующая *классификация требований*, которая получила название модели ***FURPS+***, что соответствует первым буквам соответствующих категорий требований на английском языке:

- функциональные требования *(Functionality)*
- требования удобства использования *(Usability)*
- требования надежности (Reliability)
- требования производительности *(Performance)*
- требования возможности сопровождения *(Supportability)*

символом "+" обозначены дополнительные условия, к которым относятся:

- проектные ограничения
- требования управления системой
- требования к графическому интерфейсу пользователя
- физические требования
- юридические требования

Большинство перечисленных требований относятся к категории *нефункциональных* или *атрибутов системы*. Однако, функциональные требования, которые специфицируют основное назначение системы, являются важнейшими, они должны быть определены и систематизированы в логически связанные группы.

Если X действительно является функцией системы, то имеет смысл следующее предложение: система должна выполнять «X»

Лекция 5. Тема 5

Использование вариантов использования для формализации функциональных требований

Категории функций

- **Очевидные** – их выполнение очевидно для пользователя. В основном, это функции бизнес-логики работы системы. Они являются ответом на главный вопрос анализа: «**Что должна делать система?**».
- **Скрытые** – должны выполняться незаметно для пользователя. Это касается многих базовых технических функций, таких как сохранение информации на постоянном носителе, передача протоколов. Скрытые функции зачастую необоснованно упускаются в процессе определения требований к системе.
- **Дополнительные** – необязательные функции, добавление которых не приведет к существенному удорожанию проекта и не повлияет на выполнение остальных функций.

Анализ функциональных требований. Трассировочная матрица

Большинство пользователей (заказчиков) не в силах сформулировать конкретные и четкие функциональные требования к системе, а ограничивается лишь их концептуальным описанием в форме **пожеланий** *(vision)*.

Каждое пожелание необходимо зафиксировать, уточнить, конкретизировать его смысл и поставить ему в соответствие одну или несколько функций системы.

Некоторые пожелания могут быть абстрактными, т.е. им не будет соответствовать ни одна функция системы. Абстрактные пожелания фиксируются, но из дальнейшего рассмотрения исключаются.

Пожелания и соответствующие им функции системы сохраняются в трассировочной матрице (таблице) *(Traceability Matrix).*

Лекция 5. Тема 5

Использование вариантов использования для формализации функциональных требований

Анализ функциональных требований. Трассировочная матрица (продолжение)

Наличие скрытых функций системы о которых Заказчик может и не иметь представления усложняет процесс анализа функциональных требований и может привести к существенному увеличению объема проекта и сроков его реализации, чем это предполагалось на этапе заключения договора.

Поэтому подход к их выявлению должен быть итерационным. Сначала фиксируются все очевидные функции системы и те скрытые функции, которые можно сразу выявить из логики работы системы. Остальные скрытые функции выявляются на этапах построения диаграмм вариантов использования *(use case diagram)* и, особенно, при описании сценариев и также вносятся в таблицу трассировки.

На основе уточненной трассировочной матрицы создается список функциональных требований (functional requirements, functional specification), который должен быть согласован с Заказчиком.

Лекция 5. Тема 5

Использование вариантов использования для формализации функциональных требований

Трассировочная матрица

Вариант использования (Use Case), другое название прецедент — внешняя спецификация последовательности действий, которые система может выполнять в процессе взаимодействия с актерами для получения определенного значимого для них результата.

Если каждую выявленную из пожеланий функцию отобразить в виде прецедента, результат которого тождественен выполнению данной функции, то диаграмма вариантов использования будет отображать и документировать не только сами функциональные требования, но и внешних актеров, их инициирующих и использующих их сервисы.

Пожелания заказчика (в виде исходного текста), функции системы и соответствующие им варианты использования фиксируются в трассировочной таблице обеспечивая **сквозную трассировку**:

«пожелание — функция — use case — сценарий — объекты и классы программной системы»,

реализуя таким образом один из основных принципов разработки: *прецедент определяет архитектуру программной системы.*

Трассировочную матрицу можно реализовать простейшим способом в виде таблицы текстового документа MS WORD **,** либо используя возможности CASE-средств.

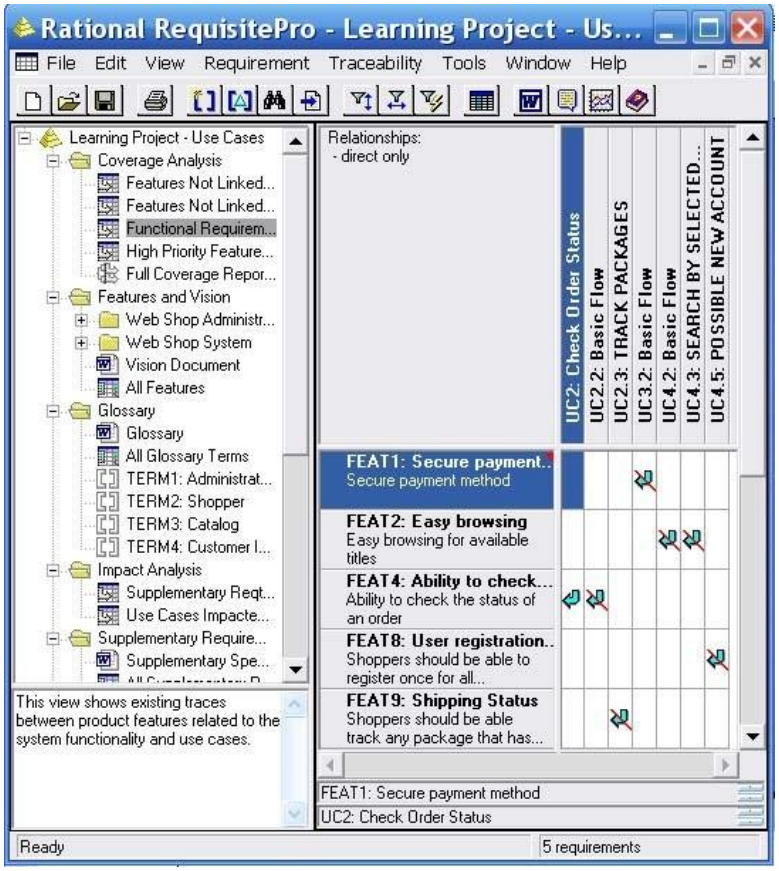
№ пп.	Пожелание Заказчика	№ ф-ции	Наименование функции	Наименование UC
1	Клиент проходит регистрацию у администратора гостиницы, который при помощи своего компьютера вносит в базу данных на выделенном сервере его имя и фамилию, адрес проживания, номер паспорта, реквизиты банка и номер кредитной карточки, дату заселения и дату убытия из отеля.	1.1	Регистрация клиета	регистрация клиента (абстр)
		1.2	Ввести ФИО клиента	ввести ФИО
		1.3	Ввести адрес проживания	ввести адрес
		1.4	Ввести номер паспорта	ввести номер паспорта
		1.5	Ввести адрес банка	Ввести адрес банка
		1.6	Ввести МФО банка	Ввести МФО банка
		1.7	Ввести № счета кредитной карточки	Ввести счет кредитной карточки
		1.8	Ввести дату заселения	Ввести дату заселения
		1.9	Ввести дату убытия	Ввести дату убытия
2	Каждый клиент получает пятиразрядный буквенно-цифровой логин.	2.1	Ввести логин клиента	Ввести логин клиента

Лекция 5. Тема 5

Использование вариантов использования для формализации функциональных требований

Трассировочная матрица (продолжение)

Пример трассировки «функция (feature) – прецедент (use case)», выполненный в Rational RequisitePro.



На практике варианты использования могут быть представлены в различных форматах с разными уровнями детализации:

- ВИ высокого уровня – описывают процессы очень сжато, обычно в двух-трех предложениях. Такой тип описания удобно использовать на начальном этапе формулирования требований к системе.
- Развернутый формат ВИ *(сценарии)* описывает процесс более детально и имеет разделы: Краткое описание, Предусловия (pre-conditions), Основной поток событий, Альтернативный поток событий, Постусловия (post-conditions).

Каждый, не являющимся абстрактным, вариант использования реализует определенное функциональное поведение, которое в свою очередь, является результатом взаимодействия определенного набора объектов программной системы (архитектуры) и внешнего пользователя. Таким образом, можно утверждать, что каждый вариант использования реализуется поведением уникальной, только ему присущей архитектуры.

Лекция 5. Тема 6

Сценарии вариантов использования

Сценарий, поиск объектов в потоке событий

Для перехода от вариантов использования к архитектуре программной системы используются текстовые описания потоков событий, называемые **сценариями** *(scenario)*.

Анализируются имена-существительные в тексте сценария. Некоторые из них будут действующими лицами, другие — объектами, а третьи — атрибутами объекта.

Как правило, каждый вариант использования включает несколько потоков:

- типовой ход событий – основной поток, результат которого ожидаем в данном варианте использования;
- альтернативный поток событий – таких потоков может быть несколько, каждый из них приводит к своему результату. Различные потоки событий возникают в точке ветвления при наличии решающего правила «**если**». В качестве альтернативных сценариев следует предусматривать и «ошибочные» потоки событий, когда актер, являясь физическим лицом, допускает одну из возможных ошибок в действиях.

Как и для вариантов использования, существует *двухэтапный подход* к написанию сценариев. На первом этапе сценарий пишут в произвольной форме с высоким уровнем абстракции. Целью такого сценария будет не выявление объектов, а поверхностное описание потока событий для общего понятия выполняемого функционала и выявления скрытых функций в данном варианте использования.

На втором этапе сценарии уточняют и подробно описывают в табличной форме с указанием начальных и конечных условий, точек ветвления с привязкой к разработанному для этого сценария или для группы сценариев проекту графического интерфейса пользователя GUI (graphical user interface). Данные сценарии используются для поиска объектов в потоке событий.

Лекция 5. Тема 6

Сценарии вариантов использования

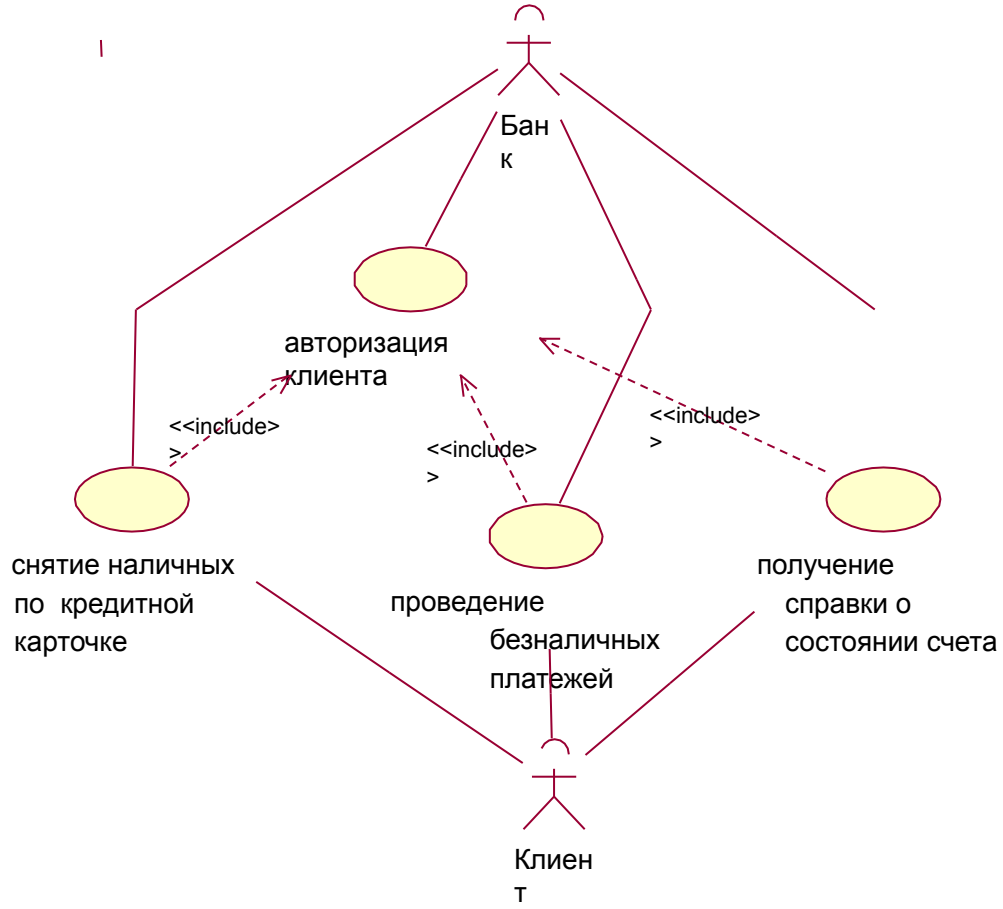
Пример: сценарии для банкомата

Для иллюстрации особенностей спецификации *функциональных требований* на диаграмме вариантов использования рассмотрим представленную здесь Use-Case-модель банкомата.

Процесс функционирования этой системы хорошо знаком владельцам кредитных карточек, поэтому не требует дополнительного описания.

Рассматриваемая система имеет двух актеров, один из них является Клиентом банкомата, а другой -- Банком, который осуществляет выполнение соответствующих транзакций. Каждый из этих актеров взаимодействует с банкоматом, хотя главный актер Клиент, поскольку именно он инициирует функциональность банкомата.

Use-Case-модель банкомата



Лекция 5. Тема 6

Сценарии вариантов использования

Пример: сценарии для банкомата

Для текстового описания сценария существуют специальные шаблоны. Каждый сценарий начинается главного раздела, в котором отражаются следующие данные:

Наименование варианта использования	**Снятие наличных по кредитной карточке**
Актеры	Клиент, Банк
Цель исполнения	Получение Клиентом требуемой суммы наличными
Краткое описание	Клиент запрашивает требуемую сумму. Банкомат обеспечивает доступ к счету клиента. Банкомат выдает клиенту наличные.
Тип	Базовый
Начальные условия	Банкомат и устройство чтения находятся в состоянии ожидания. На мониторе отображается окно с приглашением

Начальные условия при подробном описании сценария обычно связываются с соответствующими экранными формами, в которых указываются соответствующие кнопки и поля для ввода данных. Далее, после описания главного раздела описывается типовой поток событий, результат которого и является целью выполнения данного варианта использования.

Лекция 5. Тема 6

Сценарии вариантов использования

Пример: сценарии для банкомата

Типичный ход событий сценария выполнения варианта использования "Снятие наличных по кредитной карточке"	
Действия актеров	**Отклик системы**
1. Клиент вставляет кредитную карточку в устройство чтения банкомата **Точка ветвления №1: Кредитная карточка недействительна**	2.Устройство чтения проверяет кредитную карточку и на мониторе отражается окно для ввода ПИН-кода 3. Банкомат предлагает ввести ПИН-код
4. Клиент вводит персональный PIN-код **Точка ветвления №2: Клиент вводит неверный ПИН-код**	5.Банкомат проверяет ПИН-код 6.Банкомат переходит в окно выбора транзакции и отображает опции меню
7. Клиент выбирает снятие наличных со своего счета	8.Система делает запрос в Банк и выясняет текущее состояние счета клиента 9.Банкомат переходит в окно «Снятие наличных» предлагает ввести требуемую сумму
8.Клиент вводит требуемую сумму 9.Банк проверяет введенную сумму **Точка ветвления №3: Требуемая сумма превышает сумму на счете клиента**	12. Банкомат изменяет состояние счета клиента, выдает наличные и чек
13. Клиент получает наличные и чек	14. Банкомат предлагает клиенту забрать кредитную карточку
15. Клиент получает свою кредитную карточку	16. Банкомат отображает сообщение о готовности к работе

Лекция 5. Тема 6

Сценарии вариантов использования

Пример: сценарии для банкомата

Точки ветвления сценария выполнения варианта использования "Снятие наличных по кредитной карточке"	
Действия актера	**Отклик системы**
Точка ветвления №1. Кредитная карточка недействительна или неверно вставлена	
	3. Банкомат отображает информацию о неверно вставленной кредитной карточке
	4. Банкомат возвращает клиенту его кредитную карточку
5. Клиент получает свою кредитную карточку	6. Банкомат переходит в состояние ожидания.
Точка ветвления №2. Клиент вводит неверный ПИН-код	
	6. Банкомат отображает информацию о неверном ПИН-коде и предлагает ввести его повторно
4. Клиент вводит новый ПИН-код	
Точка ветвления №3. Требуемая сумма превышает сумму на счете клиента	
	12. Банкомат отображает информацию о превышении кредита и предлагает ее ввести повторно
10. Клиент вводит новую требуемую сумму	

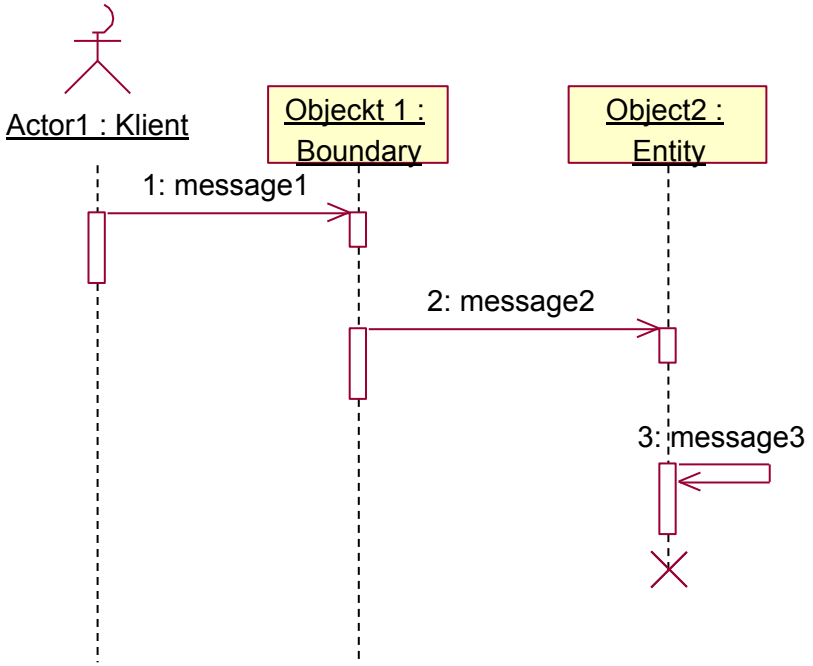
Лекция 6. Тема 7

Диаграммы последовательности (sequence diagram)

Диаграмма последовательности *(sequence diagram)* описывает временную последовательность обмена сообщений между объектами программной системы в одном из потоков событий варианта использования.

Каждый *объект* графически изображается в форме прямоугольника и располагается в верхней части своей *линии жизни* (рис.). Внутри прямоугольника записываются собственное имя *объекта* со строчной буквы и имя класса, разделенные двоеточием. При этом вся запись подчеркивается, что является признаком *объекта*, который, как указывалось ранее, представляет собой экземпляр класса. Крайним слева на диаграмме изображается *объект* -- инициатор моделируемого процесса взаимодействия, часто инициатором процесса выступает актер.

Sequence diagram – канонический вид



Лекция 6. Тема 7

Диаграммы последовательности (продолжение)

Линия жизни объекта *(object lifeline)* -- вертикальная пунктирная линия на диаграмме последовательности, которая обозначает период существования объекта.
Если *объект* существует в системе постоянно, то и его *линия жизни* должна продолжаться по всей рабочей области *диаграммы последовательности* от самой верхней ее части до самой нижней. Отдельные *объекты*, закончив выполнение своих операций, могут быть уничтожены , чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения. Для обозначения момента уничтожения *объекта* в языке UML применяется специальный символ в форме латинской буквы "X". Ниже этого символа пунктирная линия не изображается, поскольку соответствующего объекта в системе уже нет, и этот объект должен быть исключен из всех последующих взаимодействий.

Фокус управления *(focus of control)* -- специальный символ на диаграмме последовательности, указывающий период времени, в течение которого объект выполняет некоторое действие, находясь в активном состоянии.
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 активен.

В отдельных случаях объект может посылать сообщения самому себе, инициируя так называемые **рефлексивные сообщения**. Такие *сообщения* изображаются в форме *сообщения*, начало и конец которого соприкасаются с *линией жизни* или *фокусом управления* одного и того же *объекта*. Подобные ситуации возникают, например, при обработке нажатий на клавиши клавиатуры при вводе текста в редактируемый документ, при наборе цифр номера телефона абонента.

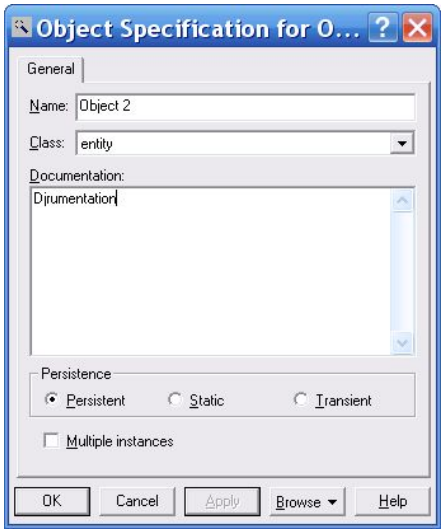
Лекция 6. Тема 7

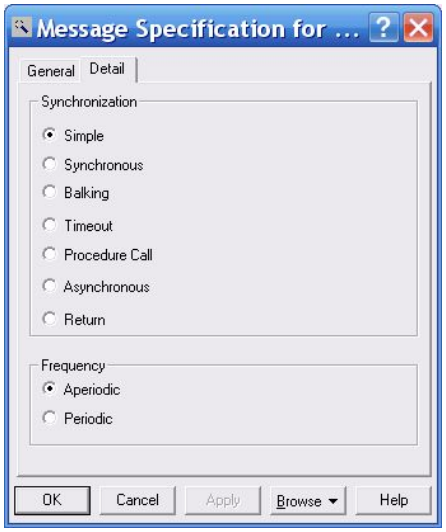
Диаграммы последовательности (продолжение)

Спецификация объектов:

1-- уникальное имя (Name);
2-- класс, к которому объект относится (Class). Класс может и не показываться (убирается флажок Show Class), либо на начальном этапе объект может быть не соотнесен с классом (unspecified)
3 -- устойчивость объекта:

- **Persistent** (Устойчивый) – объект постоянно сохраняется в базе или любым другим способом. Существует даже после прекращения работы программы.
- **Static** (Статичный) – сохраняется в течении всего времени работы программы. Существует и после выполнения отраженных на диаграмме последовательности действий, но не сохраняется после прекращения работы программы
- **Transient** (Временный) – сохраняется в течение очень короткого времени, например, пока не закончится выполнение процессов, определенных в диаграмме последовательности.





Спецификация сообщений:

1 -- именование сообщения можно задать из окна документации дважды щелкнув мышью на сообщении
2 -- установка синхронизации:

- **Simple** (Простое) – выполняется по умолчанию. Означает, что все сообщения выполняются в одном потоке управления;
  Synchronous (Синхронное) – применяется когда клиент посылает сообщение и ждет ответа пользователя;
- **Balking** (С отказом становиться в очередь) – Клиент посылает сообщение серверу Если сервер не может немедленно принять сообщение, оно отменяется;
- **Timeout** (С лимитированным временем ожидания) -- Клиент посылает сообщение серверу, а затем ждет указанное время. Если в течение указанного времени оно не принимается сервером, то затем отменяется;
- **Asynchronous** (Асинхронное) – Клиент посылает сообщение серверу и продолжает свою работу, не ожидая подтверждения о получении;
  - **Procedure Call** (Вызов процедуры);
  - **Return** (Возврат).

3 -- установка частоты сообщения

- **Periodic** (Периодические) – сообщение регулярно посылается через определенные промежутки времени
- **Aperiodic** (Апериодические) – сообщение посылается нерегулярно

Лекция 6. Тема 7

Диаграммы последовательности (продолжение)

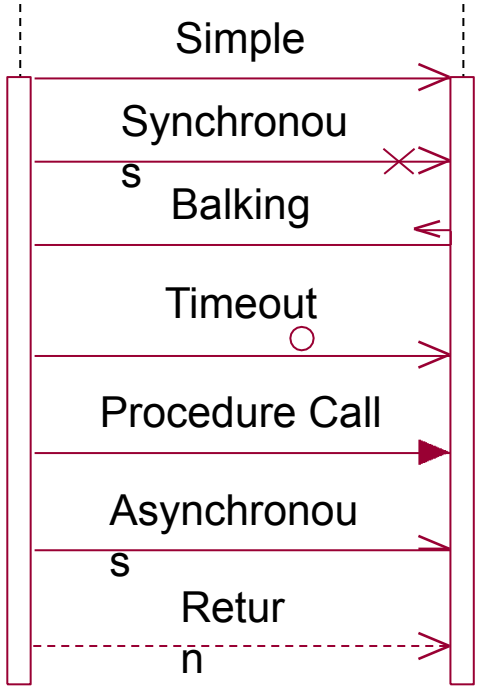
Двухэтапный подход к построению диаграмм

На первом этапе отображается информация высокого уровня, которая нужна конечным пользователям проектируемой системы. Сообщения не соотносятся с операциями, а объекты могут быть не соотнесены с классами. Эти диаграммы позволяют аналитикам и пользователям увидеть как будут развиваться процессы в системе.

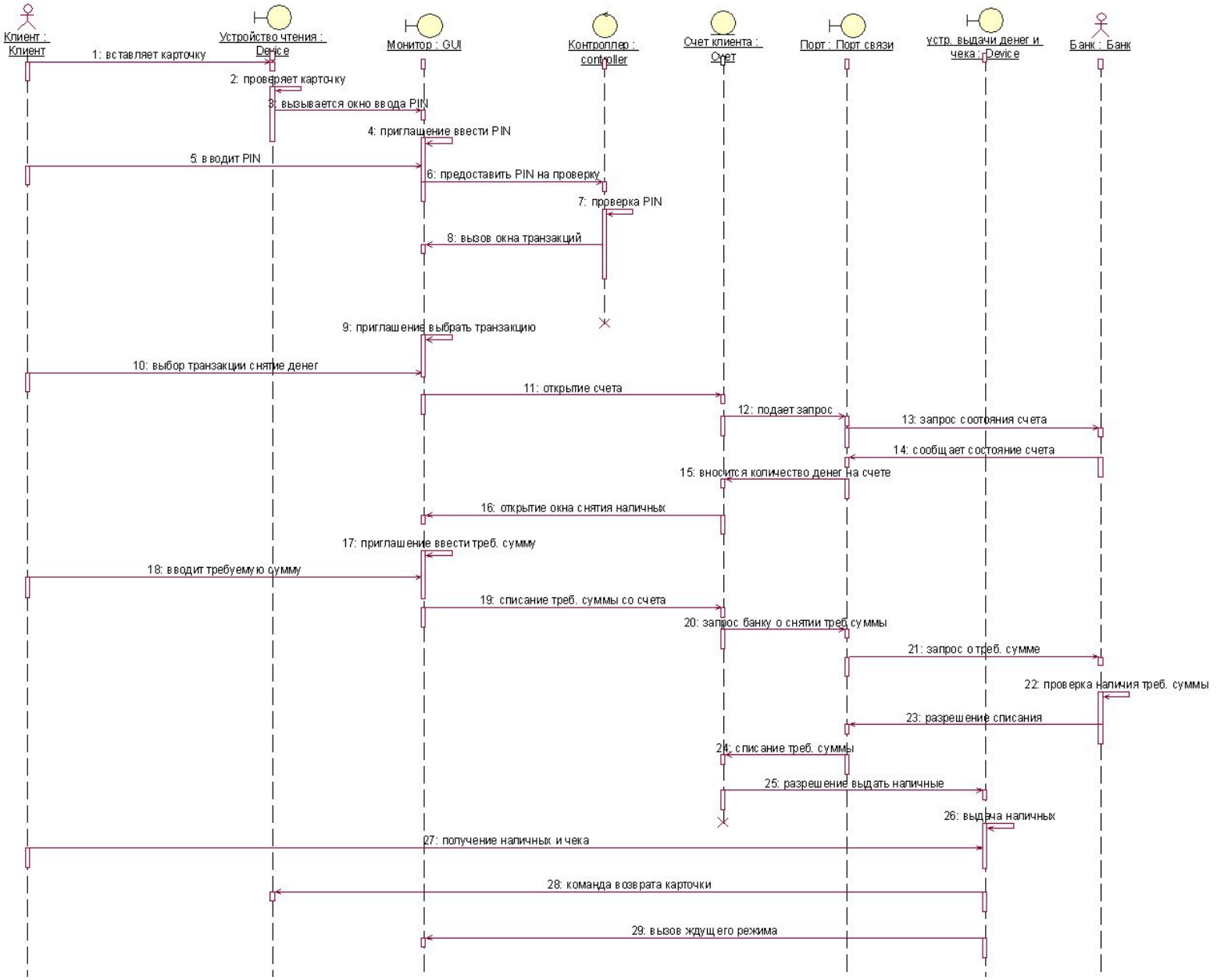
На втором этапе диаграммы детализируются и ими пользуются исключительно разработчики. В начале второго этапа на диаграммы помещают некоторые новые объекты. Как правило на каждой диаграмме последовательности имеется управляющий объект, отвечающий за выполнение последовательностью сценария. Управляющий объект не реализует никаких бизнес-процессов, он лишь посылает сообщения другим объектам и отвечает за координацию их действий.
Такие объекты называют *объектами-менеджерами*. Можно поместить на диаграмму и другие объекты, отвечающие, например, за безопасность, обработку ошибок или за связь с базой данных. Такой подход позволяет отделить бизнес-логику от логики управления или баз данных.

После того как объекты созданы, необходимо соотнести их с классами. Их можно связать с существующими классами или создать новые. Затем нужно соотнести сообщения с операциями, затем перейти к спецификациям объектов и сообщений и задать такие детали как устойчивость объекта, синхронизация и частота сообщений

Вид специфицированных в RR сообщений



Пример sequence diagram. Снятие наличных по кредитной карточке



Лекция 7. Тема 8

Диаграммы кооперации (collaboration diagram)

Диаграммы кооперации (collboration diagram), подобно диаграммам последовательности отображают поток событий в конкретном сценарии варианта использования, однако в отличие от них, больше внимания заостряют на связях между объектами.

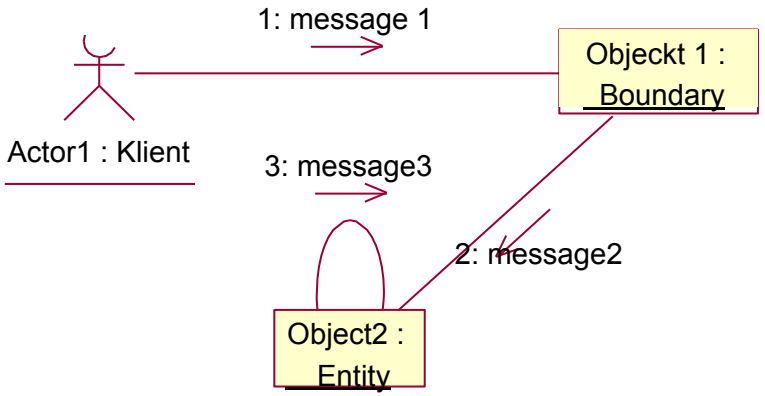


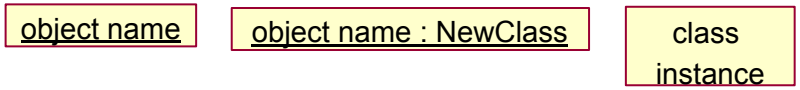
Диаграмма *кооперации* описывает поведение системы на уровне отдельных *объектов*, которые обмениваются между собой *сообщениями*, чтобы реализовать некоторый вариант использования.

На диаграмме *кооперации* размещаются ***объекты*** (object), представляющие собой экземпляры классов, ***связи*** (object link) между ними, которые в свою очередь являются экземплярами ассоциаций и ***сообщения*** (link message)***,*** направление которых показывается стрелками, сообщения специфицируются аналогично sequence diagram имеют имена и порядковые номера.

При этом, на диаграмме показываются только те *объекты*, которые участвуют в реализации моделируемой *кооперации*.

Объекты на диаграммах кооперации могут не соотносится с классами, тогда они изображаются как на рис.а, если объекты соотнесены с классами, то их изображение соответствует рис.б.

рис.а рис.б рис.в



Кроме объектов на collaboration diagram могут показываться **экземпляры классов** (*class instance*) рис.в.

Лекция 7. Тема 8

Диаграммы кооперации (collaboration diagram)

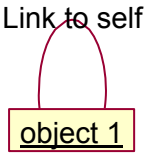
Таблица. Назначение кнопок специальной панели инструментов диаграммы кооперации

Графическое изображение	Всплывающая подсказка	Назначение кнопки
	Selection Tool	Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме
	Text Box	Добавляет на диаграмму текстовую область
	Note	Добавляет на диаграмму примечание
	Anchor Note to Item	Добавляет на диаграмму *связь* примечания с соответствующим графическим элементом диаграммы
	Object	Добавляет на диаграмму *объект*
	Class Instance	Добавляет на диаграмму экземпляр класса
	Object Link	Добавляет на диаграмму *связь*
	Link To Self	Добавляет на диаграмму рефлексивную *связь*
	Link Message	Добавляет на *связь* диаграммы прямое *сообщение*
	Reverse Link Message	Добавляет на *связь* диаграммы обратное *сообщение*
	Data Token	Добавляет на *связь* диаграммы элемент прямого потока данных
	Reverse Data Token	Добавляет на *связь* диаграммы элемент обратного потока данных

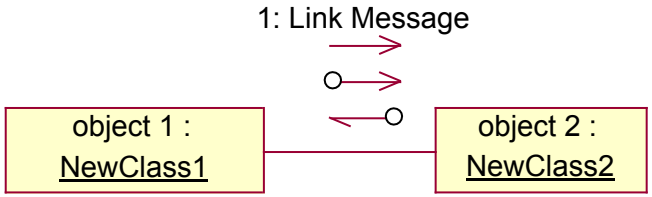
Лекция 7. Тема 8

Диаграммы кооперации (collaboration diagram)

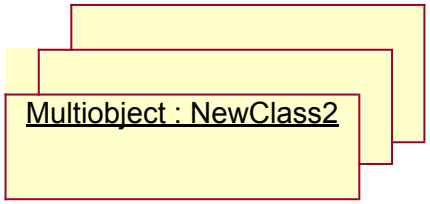
Связь с самим собой (*link to self*) – показывает, что объект может обращаться к своим операциям

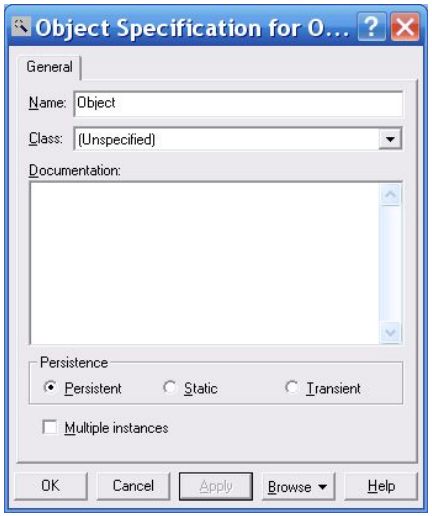


Поток данных (*data flow*) – показывает поток информации между объектами; Обратный поток данных (reverse data flow) – показывает поток информации между объектами в противоположном направлении.



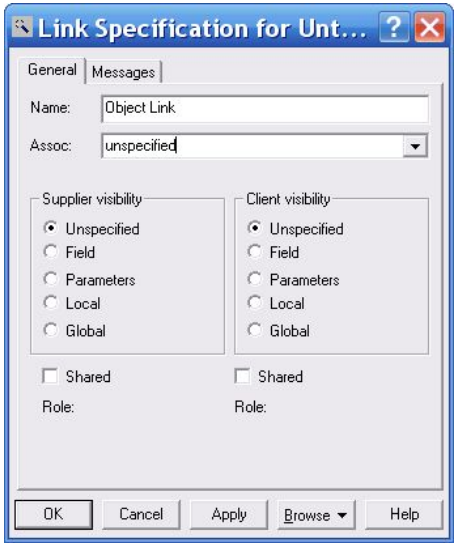
Мультиобъект (*multiobject*) представляет собой множество анонимных *объектов*, которые могут быть образованы на основе одного класса.





Спецификация объектов на диаграмме кооперации полностью аналогична их спецификации на диаграмме последовательности. Указываются имя объекта, класс и устойчивость.

По умолчанию каждая связь на диаграмме считается анонимной.



Спецификация связей включает:

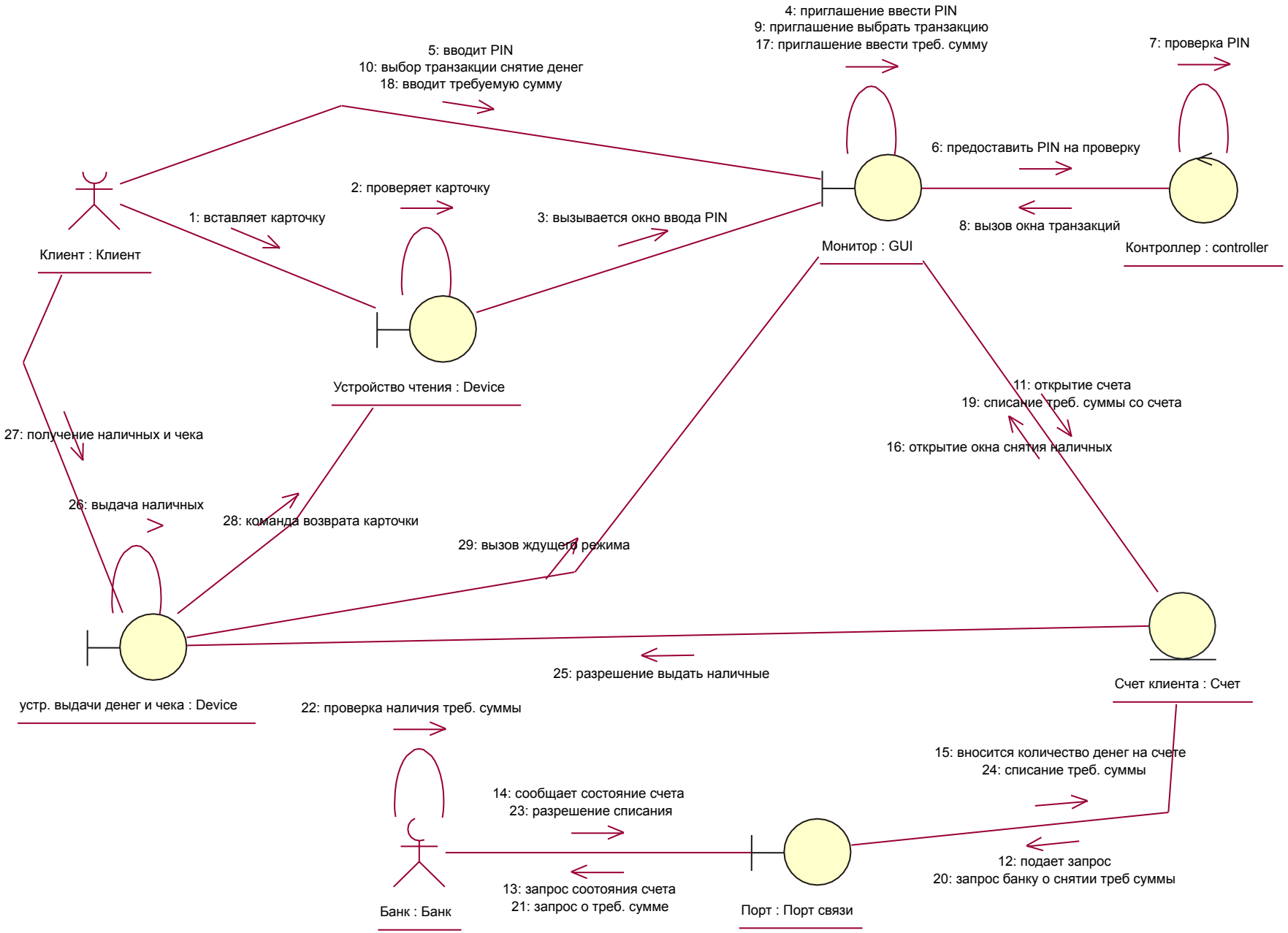
- наименование связи;
  - имя ассоциации;
- видимость соответствующей пары объектов;
  - наличие общих ролей.

Лекция 7. Тема 8

Диаграммы кооперации (collaboration diagram)

Таблица Характеристика свойств синхронизации сообщений		
Название свойства	**Графическое изображение стрелки**	**Назначение свойства**
Simple (Простое)		Данное *сообщение* выполняется в одном потоке управления. Это *свойство* задается добавляемому на диаграмму *сообщению* по умолчанию
Synchronous (Синхронное)		После передачи данного *сообщения* клиент ожидает ответа от объекта-приемника о результате выполнения соответствующей операции
Balking (С отказом)		После передачи данного *сообщения* объект-приемник отказывает клиенту в выполнении соответствующей операции, если он занят выполнением других операций
Timeout (С ожиданием)		После передачи данного *сообщения* объект-приемник может поместить данное *сообщение* в очередь с ограниченным временем ожидания, если он занят выполнением других операций
Procedure Call (Вызов процедуры)		Клиент посылает данное *сообщение* объекту-приемнику и, чтобы продолжить свою работу ожидает, пока вся дальнейшая вложенная последовательность *сообщений* не будет обработана приемником
Asynchronous (Асинхронное)		Клиент посылает данное *сообщение* и продолжает свою работу, не ожидая подтверждения от объекта-приемника о получении этого *сообщения*. При этом соответствующая операция может быть как выполнена, так и не выполнена
Return (Возврат)		Данное *сообщение* посылается клиенту после окончания выполнения вызова процедуры

Пример collaboration diagram. Снятие наличных по кредитной карточке



Лекция 8. Тема 9

Диаграммы деятельности (activity diagram)

Для моделирования процесса выполнения операций в языке UML используются **диаграммы деятельности** (*activity diagram*). С помощью этих диаграмм можно моделировать:

- процессы на предметной области (бизнес-процессы);
- бизнес-логику работы разрабатываемой программной системы;
- потоки событий вариантов использования;
- особенности реализации операций классов

Таблица. Назначение кнопок специальной панели инструментов диаграммы деятельности

Графическое изображение	Всплывающая подсказка	Назначение кнопки
	Selection Tool	Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме
	Text Box	Добавляет на диаграмму текстовую область
	Note	Добавляет на диаграмму примечание
	Anchor Note to Item	Добавляет на диаграмму связь примечания с соответствующим графическим элементом диаграммы
	State	Добавляет на диаграмму состояние
	Activity	Добавляет на диаграмму *деятельность*
	Start State	Добавляет на диаграмму начальное состояние
	End State	Добавляет на диаграмму конечное состояние
	State Transition	Добавляет на диаграмму *переход*
	Transition to Self	Добавляет на диаграмму рефлексивный *переход*
	Horizontal Synchronization	Добавляет на диаграмму горизонтально расположенный символ синхронизации
	Vertical Synchronization	Добавляет на диаграмму вертикально расположенный символ синхронизации
	Decision	Добавляет на диаграмму *символ принятия решения* для альтернативных *переходов*
	Swimlane	Добавляет на диаграмму дорожку
	Object	Добавляет на диаграмму объект (по умолчанию отсутствует)
	Object Flow	Добавляет на диаграмму стрелку потока объектов (по умолчанию отсутствует)
	Business Activity	Добавляет на диаграмму бизнес-деятельность (по умолчанию отсутствует)
	Business Transaction	Добавляет на диаграмму бизнес-транзакцию (по умолчанию отсутствует)

Лекция 8. Тема 9

Диаграммы деятельности (activity diagram)

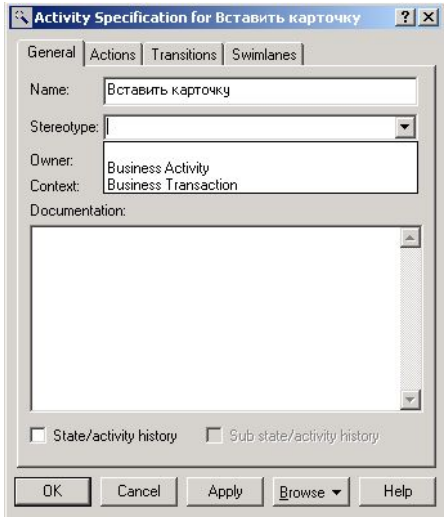
Состояния деятельности и действия

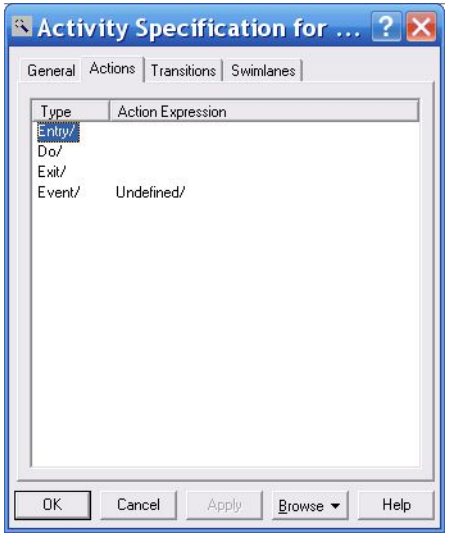
Инструмент **Activity** – моделирует **действие**, **состояние действия** (*activity, action state*) – состояние которое характеризуется некоторым действием и по крайней мере одним выходящим из состояния переходом. Переход предполагает, что входное действие уже завершилось. Состояние деятельности не может иметь внутренних переходов, поскольку оно является элементарным. Деятельность, описанная в состоянии деятельности, не может быть прервана никакими внешними событиями. Состояние действия моделирует один шаг выполнения алгоритма (процедуры) или потока управления.
Выражение действия может быть записано на естественном языке. Рекомендуется в качестве имени простого действия использовать глагол с пояснительными словами.

Спецификация Activity:

- наименование действия;
- стереотип Business Activity;
- описание;
- историческое состояние.

Открыв вкладку Activity можно специфицировать моменты наступления действия: **On Entry** – действие происходит при входе в данный тип активности; **On Exit** – действие происходит при выходе из данного типа; **Do** – действие происходит между действиями входа и выхода; **On Event** – действие происходит в ответ на определенное событие.





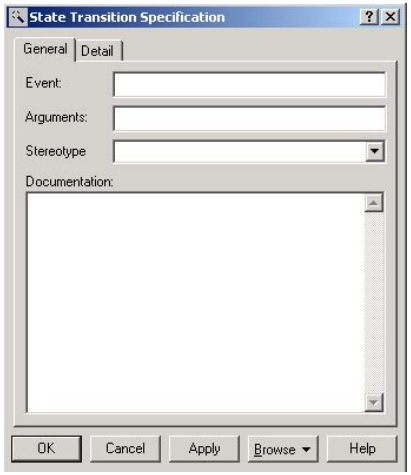
Лекция 8. Тема 9

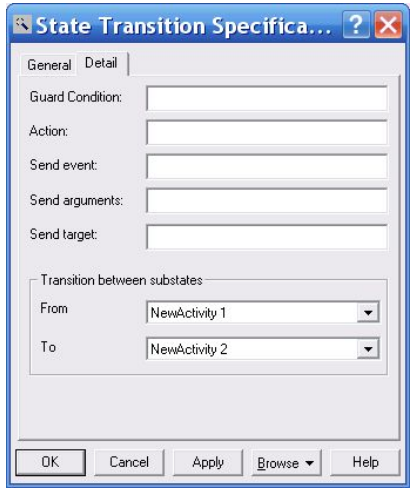
Диаграммы деятельности (activity diagram)

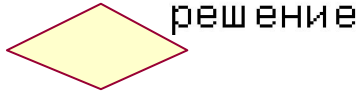
Инструмент **State Transition –** переход по завершении одного определенного действия в начало другого или из одного состояния в другое.

Обладает спецификациями, в которые входят: **Event** (событие, это то, что вызывает переход), **Arguments** (аргументы), **Stereotype** (стереотип) и **Documentation**.

На вкладке Detail вводятся дополнительные **Guard Condition** ограждающие условия (определяют, когда переход может быть выполнен, а когда нет), действия и посылаемые события. Для отображения события на диаграмме можно использовать как имя операции, так и обычную фразу.







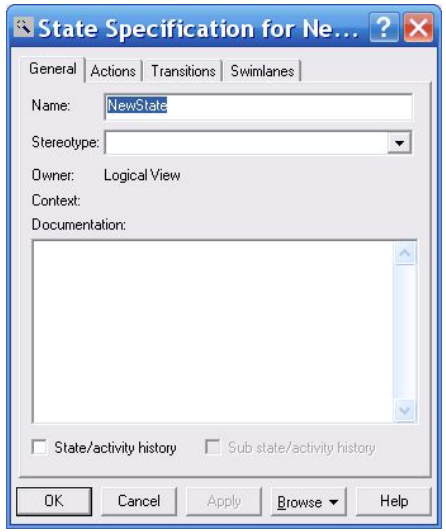
Decision (решение) показывает зависимость дальнейшей работы от внешних условий или решений.

Если из *состояния действия* выходит единственный переход, то его можно никак не помечать. Если же таких переходов несколько, то при моделировании последовательной деятельности запускается только один из них. В этом случае для каждого из таких переходов должно быть явно записано собственное граничное условие в прямых скобках. При этом для всех выходящих из некоторого состояния деятельности переходов должно выполняться требование истинности только одного из них. Подобный случай встречается тогда, когда последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения промежуточного результата. Такая ситуация получила название ветвления, а для ее обозначения применяется специальный символ **Decision** -- *решение*.

Лекция 8. Тема 9

Диаграмма деятельности (activity diagram)

Инструмент **State** – состояние предназначен для обозначения ситуации ожидания, например, в течение жизни объекта, когда объект ожидает некоторое событие или находится в некотором состоянии. Спецификация State аналогична спецификации Activity.



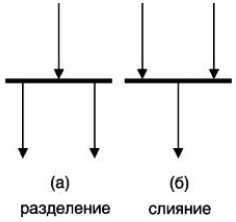
Начальное состояние

Конечное состояние

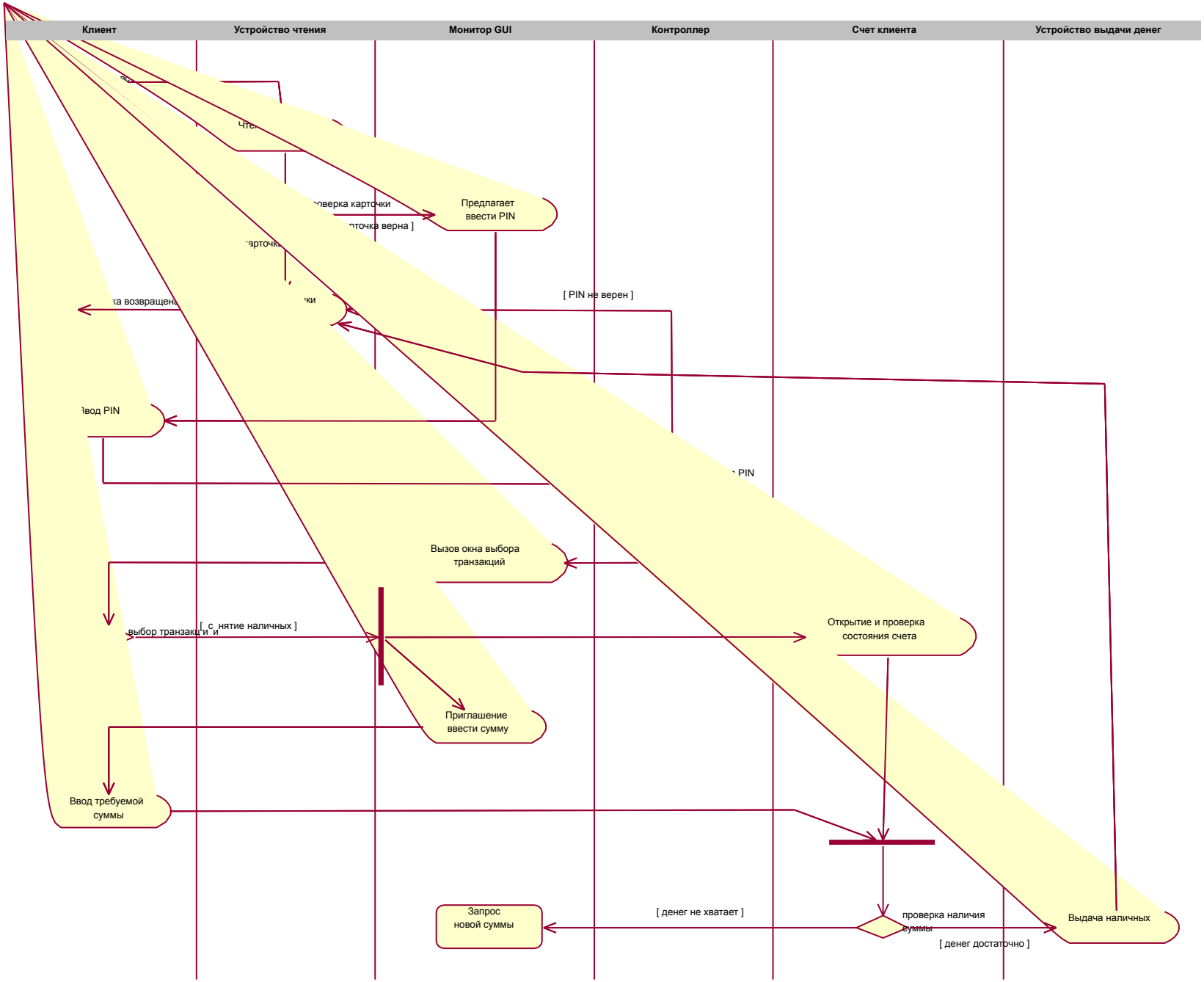
Инструмент **Start State** – начальное состояние, в котором система находится сразу после включения, либо объект сразу после своего создания, либо начальные условия use case. Начальное состояние обязательно. На диаграмме может быть только одно начальное состояние.

Инструмент **Stop State** – конечным состоянием называется то в котором система находится непосредственно перед выключением, либо объект перед уничтожением, либо use case перед завершением. Конечное состояние не является обязательным, их может быть сколько угодно.

Инструмент **Synchronization** – синхронизация позволяет определить независимо выполняемые переходы. При этом переходы могут разделяться на несколько выполняемых независимо (разделение) или, наоборот, несколько входящих переходов будут сливаться в один исходящий.



Инструмент **Swimlanes** (плавательные дорожки) -- позволяет моделировать последовательность действий различных объектов и связи между ними. При помощи этого элемента можно моделировать бизнес-процессы организации, отражая на диаграмме различные подразделения и объекты, играющие важные роли в модели бизнеса. Позволяет показать, кто выполняет те или иные роли в процессе.


Клиент
Устройство чтения
Монитор GUI
Контроллер
Счет клиента
Устройство выдачи денег
Предлагает ввести PIN
[PIN не верен]
Вызов окна выбора транзакций
Открытие и проверка состояния счета
Приглашение ввести сумму
Ввод требуемой суммы
Запрос новой суммы
[денег не хватает]
проверка наличия суммы
[денег достаточно]
Выдача наличных

Лекция 9. Тема 10

Диаграмма классов (class diagram)

Диаграмма *классов* (class diagram) является основным логическим представлением модели и содержит детальную информацию о внутреннем устройстве объектно-ориентированной программной системы или, об архитектуре программной системы.
На диаграмме *классов* представлена совокупность декларативных или статических элементов модели, таких как *классы* с *атрибутами* и *операциями*, а также связывающие их отношения.

Класс (class) — абстрактное описание множества однородных объектов, имеющих одинаковые *атрибуты*, *операции* и отношения с объектами других *классов*.

Класс может иметь или не иметь экземпляров или объектов. В зависимости от этого в языке UML различают *конкретные* и *абстрактные классы*.

Конкретный класс (*concrete class*) — *класс*, на основе которого могут быть непосредственно созданы экземпляры или объекты.

Абстрактный класс (*abstract class*) **—** *класс*, который не имеет экземпляров или объектов.

Имя класса должно быть уникальным, записывается по центру секции *имени* полужирным шрифтом и должно начинаться с заглавной буквы. Рекомендуется в качестве *имен классов* использовать существительные, записанные без пробелов, взятые из словаря предметной области.

Для обозначения *имени абстрактного класса* используется наклонный шрифт (курсив). В языке UML принято общее соглашение о том, что любой текст, относящийся к абстрактному элементу, записывается курсивом.

Лекция 9. Тема 10

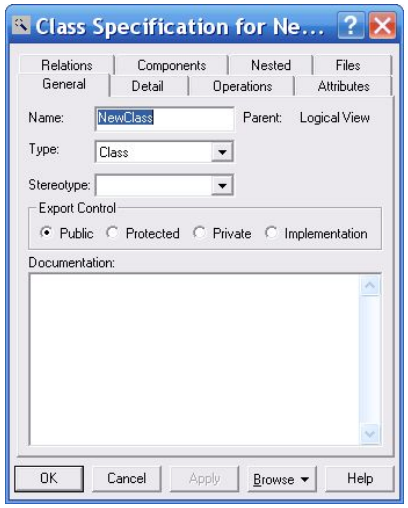
Диаграмма классов (class diagram)

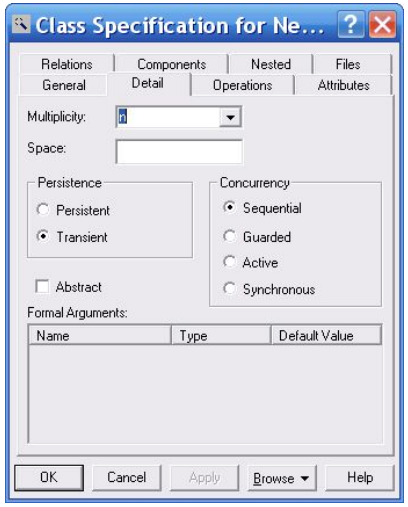
Назначение кнопок специальной панели инструментов для диаграммы классов

Графическое изображение	Всплывающая подсказка	Назначение кнопки
	Selection Tool	Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме
	Text Box	Добавляет на диаграмму текстовую область
	Note	Добавляет на диаграмму примечание
	Anchor Note to Item	Добавляет связь примечания с соответствующим графическим элементом диаграммы
	Class	Добавляет на диаграмму *класс*
	Interface	Добавляет на диаграмму *интерфейс*
	Unidirectional Association	Добавляет на диаграмму направленную *ассоциацию*
	Association Class	Добавляет на диаграмму *ассоциацию класс*
	Package	Добавляет на диаграмму пакет
	Dependency or Instantiates	Добавляет на диаграмму отношение зависимости
	Generalization	Добавляет на диаграмму отношение обобщения
	Realize	Добавляет на диаграмму отношение реализации

Лекция 9. Тема 10

Диаграмма классов (class diagram)





Спецификации класса

В окне **General** (Общие) указываются:
Name -- имя класса,
Type -- тип класса,
Stereotype – стереотип,
Export Control – видимость,
Documentation – описание.

В окне **Detail** (Подробно) указываются:
Multiplizity – множественность класса показывает сколько у данного класса должно быть экземпляров,
Space – требования к хранению класса указывают сколько требуется памяти для его хранения,
Persistence – устойчивость класса,
Concurrency – параллелизм, описывает поведение класса при наличии нескольких потоков управления,
Abstract – указатель абстрактного класса,
Formal Arguments – аргументы класса.

Спецификация классов по типам:
Регулярные (Class, по умолчанию); **Параметризованные** (parameterized) – классы шаблоны. Применяется для создания семейства других классов;
Классы-наполнители (instantiated class) – это параметризированный класс, аргументы которого имеют фактические значения;
Утилиты классов (class utility) – это совокупность операций, например математических;
Утилиты параметризованных классов (parameterized class utility) – шаблон для создания утилит классов;
Утилиты классов наполнителей (instantiated class utility) – утилита параметризованного класса параметры которой имеют фактические значения;
Метаклассы (metaclass) – классы экземпляры которых тоже являются классами.

Лекция 9. Тема 10

Диаграмма классов (class diagram)

Спецификация видимости классов:

- **Public** (открытый) – виден всем классам системы;
- **Protected** (защищенный) – не виден всем классам, за исключением подклассов данного класса;
- **Private** (закрытый) – не доступен для всех классов, за исключением самого себя;
- **Implementation** (пакетный) -- виден из классов того же пакета**.**

Спецификация устойчивости класса:

- **Persistent** (Устойчивый) – сохраняется и после завершения работы приложения (данные хранятся в базе);
- **Transient** (Временный) – не будет храниться после окончания работы приложения.

Спецификация по параллелизму:

- **Sequential** (Последовательный) – (по умолчан.) класс ведет себя нормально при наличии одного потока управления, в присутствии нескольких потоков поведение не гарантируется;
- **Guarded** (Ограждающий) – при наличии нескольких потоков будет вести себя как ожидается но чтобы классы различных потоков не мешали друг другу они должны взаимодействовать;
- **Active** (Активный) – класс будет иметь собственный поток управления;
- **Synchronous** (Синхронный) – класс может самостоятельно обрабатывать взаимные исключения.

Лекция 9. Тема 10

Диаграмма классов (class diagram)

Атрибуты класса

Атрибут *(attribute)* — содержательная характеристика *класса*, описывающая множество значений, которые могут принимать отдельные объекты этого *класса*.
Атрибут класса служит для представления отдельного свойства или признака, который является общим для всех объектов данного *класса*. *Атрибуты класса* записываются во второй сверху секции прямоугольника *класса*.

В языке UML принята определенная стандартизация записи *атрибутов класса*. Каждому *атрибуту класса* соответствует отдельная строка текста, которая состоит из *квантора видимости атрибута*, имени *атрибута*, его *кратности*, типа значений *атрибута* и, возможно, его исходного значения.
Общий формат записи отдельного *атрибута класса* следующий:

<квантор видимости> <имя атрибута> [кратность] :<тип атрибута> = <исходное значение> {строка-свойство}.

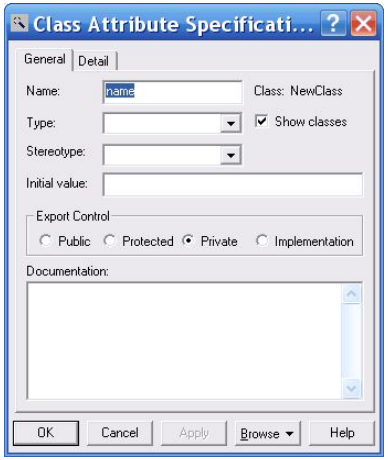
Видимость (*visibility*) — качественная характеристика описания элементов *класса*, характеризующая потенциальную возможность других объектов модели оказывать влияние на отдельные аспекты поведения данного *класса*.
Видимость в языке UML специфицируется с помощью ***квантора видимости*** *(visibility)*, который может принимать одно из 4-х возможных значений и отображаться при помощи специальных символов (Таблица ниже). *Квантор видимости* может быть опущен. Его отсутствие означает, что видимость *атрибута* не указывается.

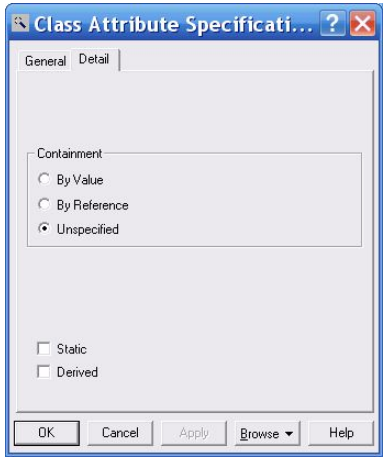
Имя атрибута представляет собой строку текста, которая используется в качестве идентификатора соответствующего *атрибута* и поэтому должна быть уникальной в пределах данного *класса*. Имя *атрибута* - единственный обязательный элемент синтаксического обозначения *атрибута*, должно начинаться со строчной (малой) буквы и не должно содержать пробелов.

Лекция 9. Тема 10
Спецификация атрибутов

Пиктограммы видимости атрибутов классов

Графическое изображение	Текстовый аналог	Назначение пиктограммы
	Public	Общедоступный или открытый. В нотации языка UML такому *атрибуту* соответствует знак «+»
	Protected	Защищенный. В нотации языка UML такому *атрибуту* соответствует знак «#»
	Private	Закрытый. В нотации языка UML такому *атрибуту* соответствует знак «-»
	Implementation	Реализация. В нотации языка UML такому *атрибуту* соответствует знак «~»





Спецификация атрибутов:
Name -- имя атрибута;
Type -- тип атрибута (boolean, byte, currency, data) показывает тип данных;
Stereotype – стереотип;
Initial value – начальное значение;
Export Control – управление экспортом или видимость;
Documentation – описание.
На вкладке **Detail** в группе выбора **Containment** (Локализация) можно специфицировать условия хранения *атрибута* у объектов выбранного класса.

Далее можно определить атрибут как статичный, выставив отметку в строке выбора **Static.** Статичный атрибут по определению имеет одно и тоже значение для всех объектов рассматриваемого класса. Наконец, на вкладке **Detail** можно определить атрибут как **производный**, выставив отметку в строке выбора **Derived**. Значение производного *атрибута* по определению может быть вычислено на основании значений других *атрибутов* этого или другого класса.

Лекция 9. Тема 10

Диаграмма классов (class diagram)

Операции класса

Операция *(operation)* -- это сервис, предоставляемый каждым экземпляром или объектом *класса* по требованию своих клиентов, в качестве которых могут выступать другие объекты, в том числе и экземпляры данного *класса*.

На практике клиент обычно совершает над объектами операции пяти видов. Из них наиболее распространены следующие три вида операции.

- **Модификатор:** операция, изменяющая состояние объекта.
- **Селектор:** операция, имеющая доступ к состоянию объекта, но не изменяющая его.
- **Итератор:** операция, обеспечивающая доступ ко всем частям объекта в строго определенном порядке.

Два остальных вида операций носят общий характер. Они образуют инфраструктуру, необходимую для создания и уничтожения экземпляров класса.

- **Конструктор:** операция, создающая объект и/или инициализирующая его состояние.
- **Деструктор:** операция, стирающая состояние объекта и/или уничтожающая сам объект.

Операции класса записываются в третьей сверху секции прямоугольника *класса*, которую часто называют секцией *операций*. Совокупность *операций* характеризует функциональный аспект поведения всех объектов данного *класса*. Запись *операций класса* в языке UML также стандартизована и подчиняется определенным синтаксическим правилам. При этом каждой *операции класса* соответствует отдельная строка, которая состоит из *квантора видимости операции*, имени *операции*, выражения типа возвращаемого *операцией* значения и, возможно, строка-свойство данной

Лекция 9. Тема 10

Диаграмма классов (class diagram)

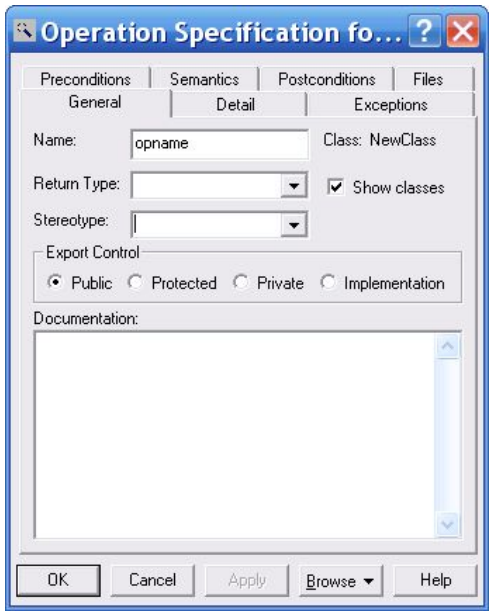
Операции класса

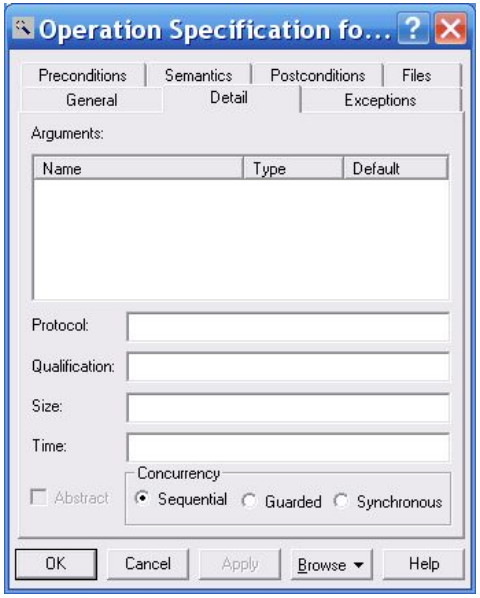
Квантор видимости, как и в случае *атрибутов класса*, может принимать одно из четырех возможных значений: общедоступный (public), защищенный (protected), закрытый (private), пакетный (package).

Квантор видимости для *операции* может быть опущен. В этом случае его отсутствие просто означает, что видимость *операции* не указывается. Вместо условных графических обозначений также можно записывать соответствующее ключевое слово: public, protected, private, package.

Имя операции представляет собой строку текста, которая используется в качестве идентификатора соответствующей *операции* и поэтому должна быть уникальной в пределах данного *класса*. Имя *операции* - единственный обязательный элемент синтаксического обозначения *операции*, должно начинаться со строчной (малой) буквы, и, как правило, записываться без пробелов.

Диаграмма классов (class diagram)





Лекция 9. Тема 10 Спецификация операций

Name – имя операции;
Return Type -- тип возвращаемого результата (boolean, byte, currency, data) показывает тип данных;
Stereotype – стереотип;
Export Control – управление экспортом или видимость;
Documentation – описание.

На вкладке **Detail** в поле **Protocol** (Протокол) можно специфицировать порядок выполнения *операций* класса, например, указать, что одна *операция* не может быть вызвана раньше другой. Соответствующий текст в данное поле вводится с клавиатуры и попадает в генерируемый код в форме комментария.

В поле **Qualification** (Квалификация) можно уточнить детали реализации *операции*, связанные с конкретным языком программирования. Соответствующий текст также вводится в данное поле с клавиатуры и попадает в генерируемый код в форме комментария.

Далее на этой же вкладке в полях **Size** (Размер) и **Time** (Время) можно специфицировать предполагаемый объем памяти и время, необходимое для выполнения *операции*. Соответствующая информация попадает в генерируемый код в форме комментария.

В группе выбора **Concurrency** (Параллельность) можно специфицировать условия на возможность параллельного выполнения данной *операции*.

Диаграмма классов (class diagram)

Лекция 9. Тема 10 Спецификация операций

Для выбора могут быть использованы следующие свойства:

- **Sequential** (Последовательная) - свойство по умолчанию, которое означает, что данная *операция* класса может быть выполнена только при наличии одного потока управления, т. е. соответствующая *операция* класса должна выполняться последовательно. При наличии нескольких потоков управления выполнение данной *операции* класса не гарантируется.
- **Guarded** (Безопасная) - означает, что при наличии нескольких потоков управления выполнение данной *операции* класса гарантируется только в том случае, когда обеспечено взаимодействие объектов друг с другом в различных потоках.
- **Synchronous** (Синхронная) - означает, что выполнение данной *операции* класса гарантируется при наличии нескольких потоков управления. При этом нет необходимости во взаимодействии объектов в различных потоках управления, поскольку данная *операция* класса будет выполняться в отдельном потоке управления вплоть до своего завершения.

Диаграмма классов (class diagram)

Лекция 10. Тема 11 Отношения на диаграмме классов

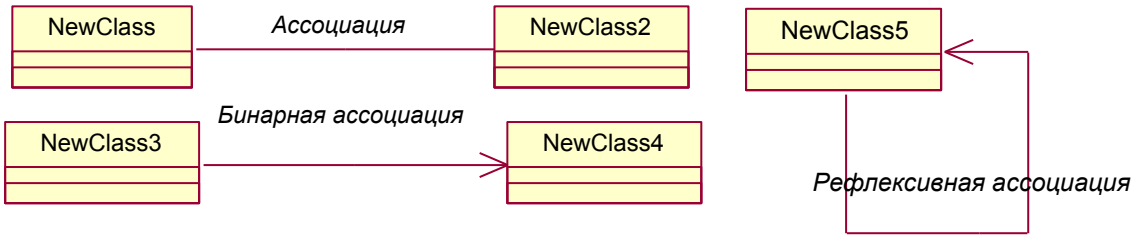
Кроме внутреннего устройства классов важную роль при разработке проектируемой системы имеют различные отношения между классами, которые также могут быть изображены на диаграмме классов. Совокупность допустимых типов таких отношений строго фиксирована в языке UML и определяется самой семантикой этих отношений. Базовые отношения, изображаемые на диаграммах классов:

- Отношение *ассоциации* (association relationship)
- Отношение *обобщения* (generalization relationship)
- Отношение *агрегации* (aggregation relationship)
- Отношение *композиции* (composition relationship)

Каждое из этих отношений имеет собственное графическое представление, которое отражает семантический характер взаимосвязи между объектами соответствующих классов.

Отношение ассоциации

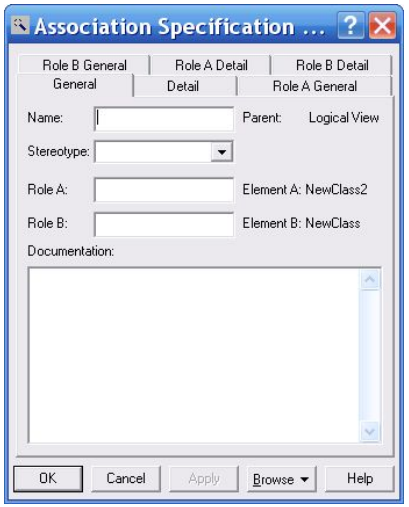
Ассоциация (association) или **бинарная ассоциация** (binary association) -- семантическое отношение между двумя и более классами, которое специфицирует характер связи между соответствующими экземплярами этих классов. Отношение *ассоциации* соответствует наличию произвольного отношения или взаимосвязи между классами.

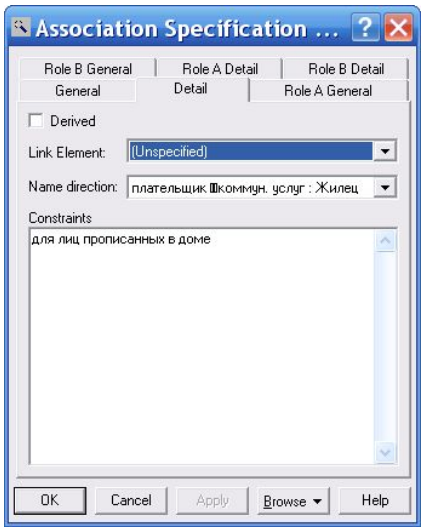


Ассоциация связывает два различных класса и может быть ненаправленным (симметричным) или направленным отношением. Частный случай бинарной *ассоциации* - **рефлексивная ассоциация**, которая связывает класс с самим собой.

Лекция 10. Тема 11 Отношения на диаграмме классов

Спецификации ассоциаций





Окно General (Общие)

Имя (Name) -- необязательный элемент ее обозначения, задается в случае, если ассоциация не очевидна, записывается с большой буквы.

Стереотип (Stereotype) – необязательный элемент.

Имя роли (Role) -- строка текста рядом с концом *ассоциации* для соответствующего класса. Она указывает на специфическую роль, которую играет класс, являющийся концом рассматриваемой *ассоциации*. Имя роли не обязательный элемент обозначений и может отсутствовать на диаграмме.

Окно Detail (Подробно)

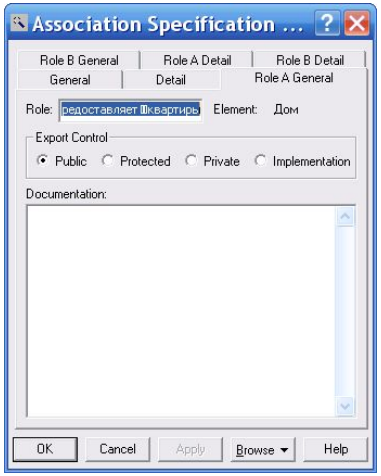
Производная (Derived) – наклонная черта перед именем связи, показывает, что она является производной от другого элемента.

Ограничение (Constraints) – условие ограничивающие действие связи.

Окно Role A Detail (Роль А подробно)

Кратность (Multiplicity) -- означает потенциальное число отдельных экземпляров класса, которые могут иметь место, когда остальные экземпляры или объекты классов фиксированы.

Спецификации ролей



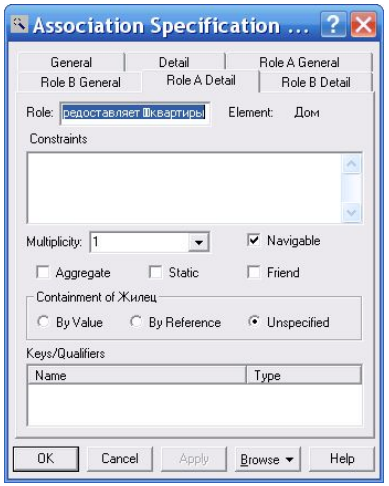
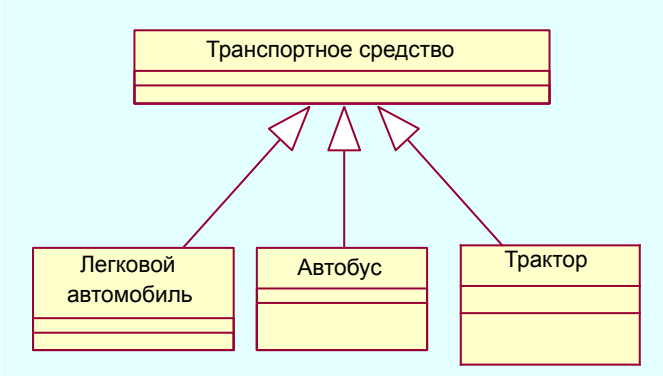
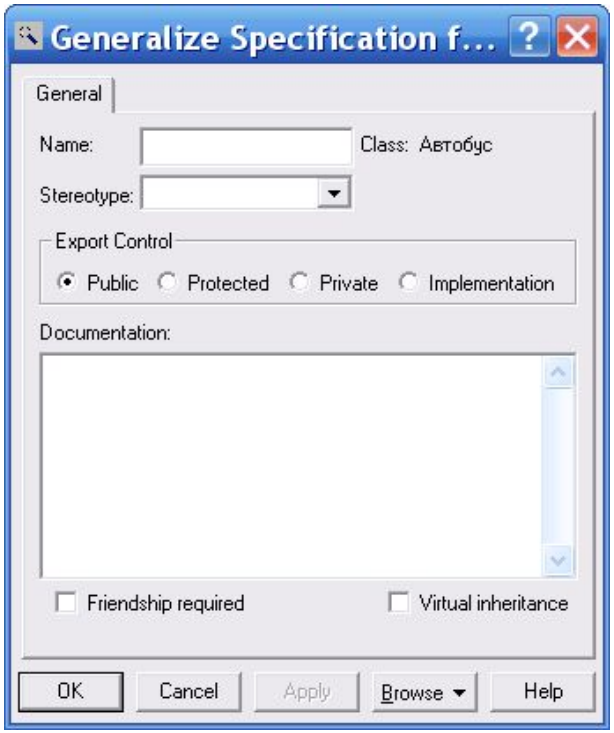


Диаграмма классов (class diagram)

Лекция 10. Тема 11 Отношения на диаграмме классов





Отношение обобщения

Обобщение (generalize) – отношение между общим (родителем) и частным (предком). Применительно к диаграмме классов данное отношение описывает иерархическое строение классов и *наследование* их свойств и поведения.

Наследование (inheritance) -- специальный концептуальный механизм, посредством которого более специальные элементы включают в себя структуру и поведение более общих элементов.

Согласно одному из главных принципов методологии ООАП -- *наследованию*, класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет собственные свойства и поведение, которые могут отсутствовать у класса-предка.

Родитель, предок (parent) -- в отношении обобщения более общий элемент. **Потомок** (child) - специализация одного из элементов отношения обобщения, называемого в этом случае родителем.

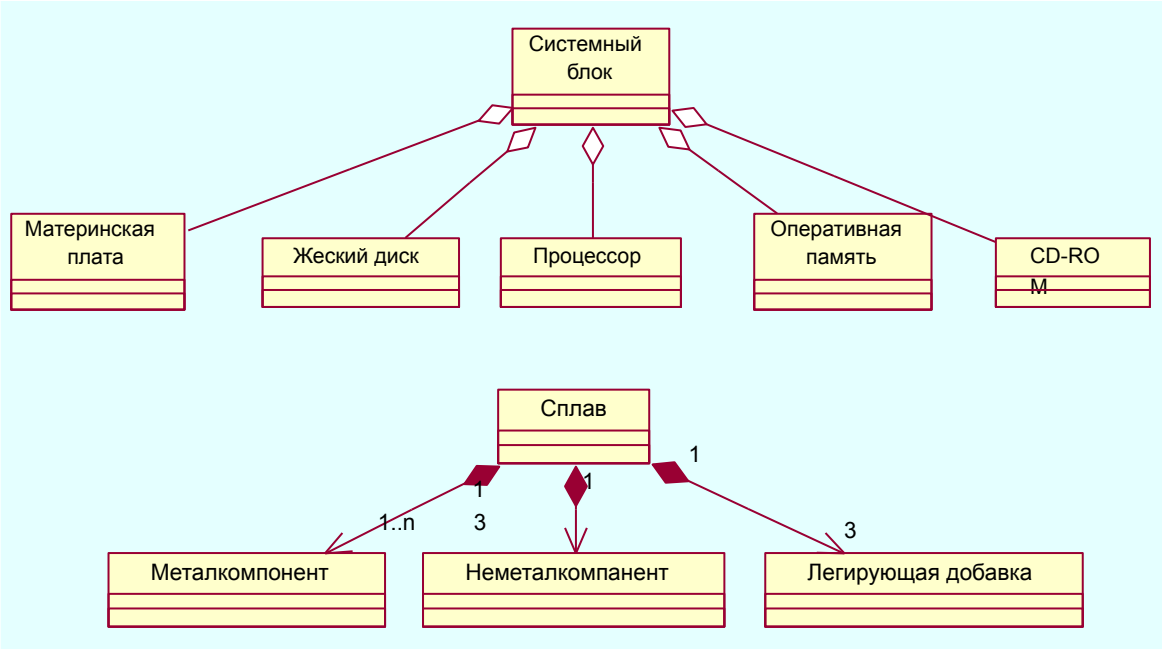
От одного класса-предка одновременно могут наследовать несколько классов-потомков

Диаграмма классов (class diagram)

Лекция 10. Тема 11 Отношения на диаграмме классов

Отношение агрегации

Агрегация (aggregation) -- специальная форма ассоциации, которая служит для представления отношения "часть-целое" между агрегатом (целое) и его составной частью. Отношение *агрегации* имеет место между несколькими классами в том случае, если один из классов представляет собой сущность, которая включает в себя в качестве составных частей другие сущности.



Принципиальное отличие агрегации от обобщения заключается в том, что части целого никак не обязаны наследовать его свойства и поведение, поскольку являются самостоятельными сущностями. Более того, части целого обладают собственными атрибутами и операциями, которые существенно отличаются от атрибутов и операций целого.

Спецификации отношения агрегации – полностью аналогичны спецификациям ассоциации.

Композиция (composition) **--** отношение *композиции* -- частный случай отношения *агрегации*. Служит для спецификации более сильной формы отношения "часть-целое", при которой составляющие части тесно взаимосвязаны с целым. Особенность этой взаимосвязи заключается в том, что части не могут выступать в отрыве от целого, т.е. с уничтожением целого уничтожаются и все его составные части. Для установки композиции на вкладке **Role Detail** устанавливается опция **By Value**.

№ п.п.	Наименование артефакта	Тип артефакта	Область аналитики
1	Глоссарий предметной области	Текст. документ	Бизнес.анализ
	Сущности предметной области	Class diagram	
	Бизнес-актеры и бизнес-функции	Use Case diagram	
	Бизнес-прецессы	Activity diagram	
2	Фиксация пожеланий заказчика, выявление функций системы	Трассировочная матрица	Анализ требований
	Спецификация функциональных требований	Текст. документ	
	Спецификация нефункциональных требований	Текст. документ	
	Модели вариантов использования	Use Case diagram	
	План итераций	Текст.документ Use Case diagram	
	Сценарии вариантов использования	текст	
	Прототипы экранных форм	Скрин-шоты	
	Представление Use Case-процессов	Activity diagram	
3	Модели взаимодействия объектов программной системы	Sequence diagram Collaboration diagram	Системный анализ
	Модели классов анализа	Class diagram	
	Модель размещения	Deployment diagram	

Диаграммы UML 2