

Объектно- ориентированные технологии программирования и стандарты проектирования



Факультет информационных технологий
старший преподаватель кафедры ВСиС Глухова Т.М



ПОЛОЦКИЙ
ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ



Диаграмма вариантов использования

Лекция 1



ПОЛОЦКИЙ
ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ



План лекции

- Назначение диаграммы вариантов использования
- Компоненты диаграммы вариантов использования
- Примеры



Диаграмма вариантов использования (ДВИ)...

- = *Диаграмма прецедентов*;
- Описывает функциональное назначение системы, т.е. то, что система будет делать в процессе своего функционирования;
- Является *исходной концептуальной моделью системы* в процессе ее проектирования и разработки.



Кому и в каких случаях нужны сценарии

- — **Разработчикам.** Очень удобно, когда ветвистое требование описано при помощи основного и альтернативного потока событий.
- — **Заказчикам.** Описано человеческим языком, заказчик своевременно может подтвердить, что это именно то, чего он ждет, или поправить.
- — **Тестировщику.** Почти готовый тест-кейс
- — **Всей проектной команде.** Если сценарий нужно согласовать, а на каждом совещании пара-тройка альтернативных вариантов сценария звучит иначе, поможет строго описанный поток событий.



Суть диаграммы прецедентов

- Проектируемая система представляется в виде множества сущностей или *актеров* (действующих лиц), взаимодействующих с системой с помощью так называемых *вариантов использования* (прецедентов).

Таким образом,

- Основными компонентами ДВИ являются:
 - актеры
 - прецеденты
 - отношения



Вариант использования

- = *Прецедент* = *use case*;
- Определяет последовательность действий, которая должна быть выполнена проектируемой системой при взаимодействии ее с соответствующим актером.



Имя ВИ начинается с большой буквы и обозначается оборотом глагола или существительного, обозначающего действие

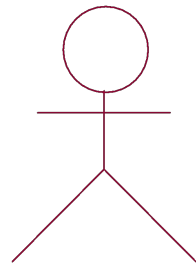
Актер

- = *Actor* = действующее лицо
- Представляет собой внешнюю по отношению к моделируемой системе сущность
- Взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей и решения частных задач.
- Может рассматриваться как некая **роль** относительно конкретного варианта использования.



Актер

Стандартное графическое изображение актера:



Клиент банка

- Актер всегда находится вне системы, его *внутренняя структура* никак не воспринимается.
- Примеры актеров: клиент банка, банковский служащий, продавец, сотовый телефон.



Отношения

- Один актер может взаимодействовать с несколькими вариантами использования и наоборот.
- 2 варианта использования, определенные для одной и той же сущности, не могут взаимодействовать друг с другом, т.к. любой из них самостоятельно описывает законченный вариант использования этой сущности.



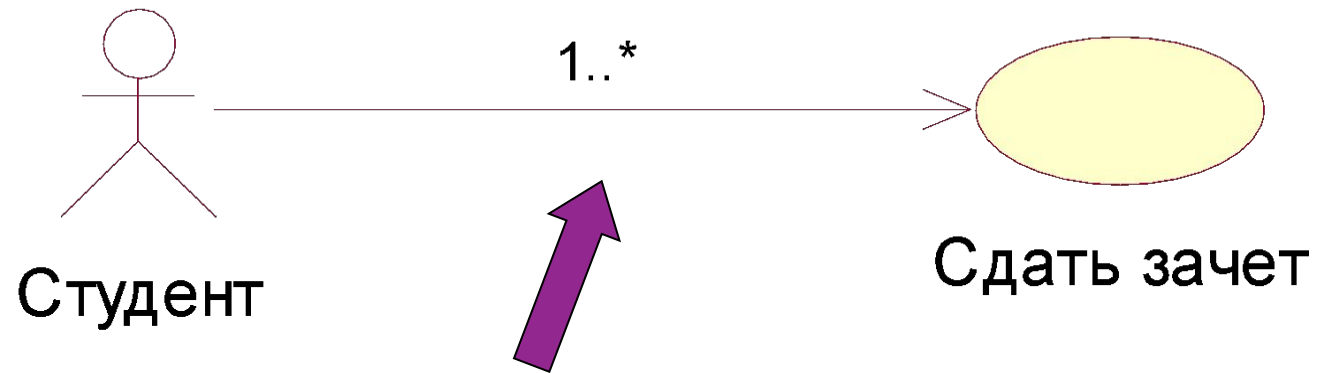
Виды отношений

- 1) **ассоциативное отношение** (отношение ассоциации, association relationship)
- 2) **отношение расширения** (extend relationship)
- 3) **отношение обобщения** (generalization relationship)
- 4) **отношение включения** (include relationship)



Отношение ассоциации

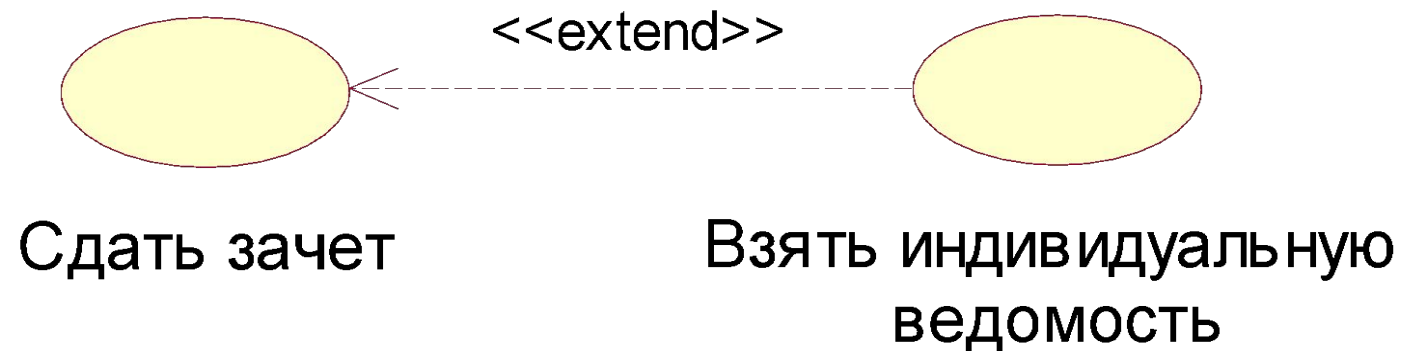
- Отношение между вариантом использования и актером, отражающее *связь* между ними.
- Оно устанавливает, какую конкретную роль играет актер при взаимодействии с экземпляром варианта использования.



Обозначение: в виде прямой линии. Могут быть дополнительные обозначения (кратность связи, направление связи, наименование связи)

Отношение расширения

- Определяет взаимосвязь *базового варианта использования* с некоторым *другим* вариантом использования, функциональное поведение которого задействуется базовым не всегда, а только при выполнении некоторых дополнительных условий.

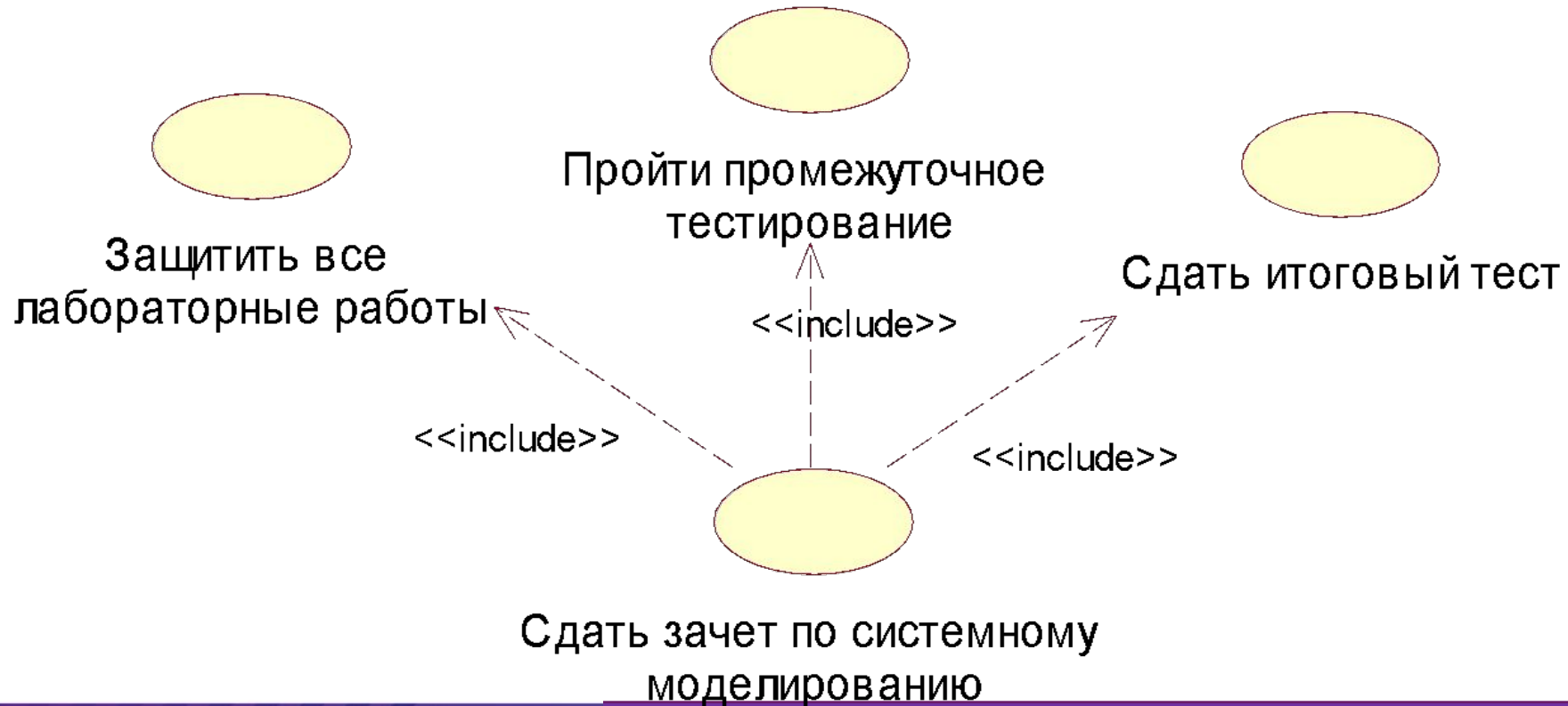


Стрелка указывает на базовый вариант использования!



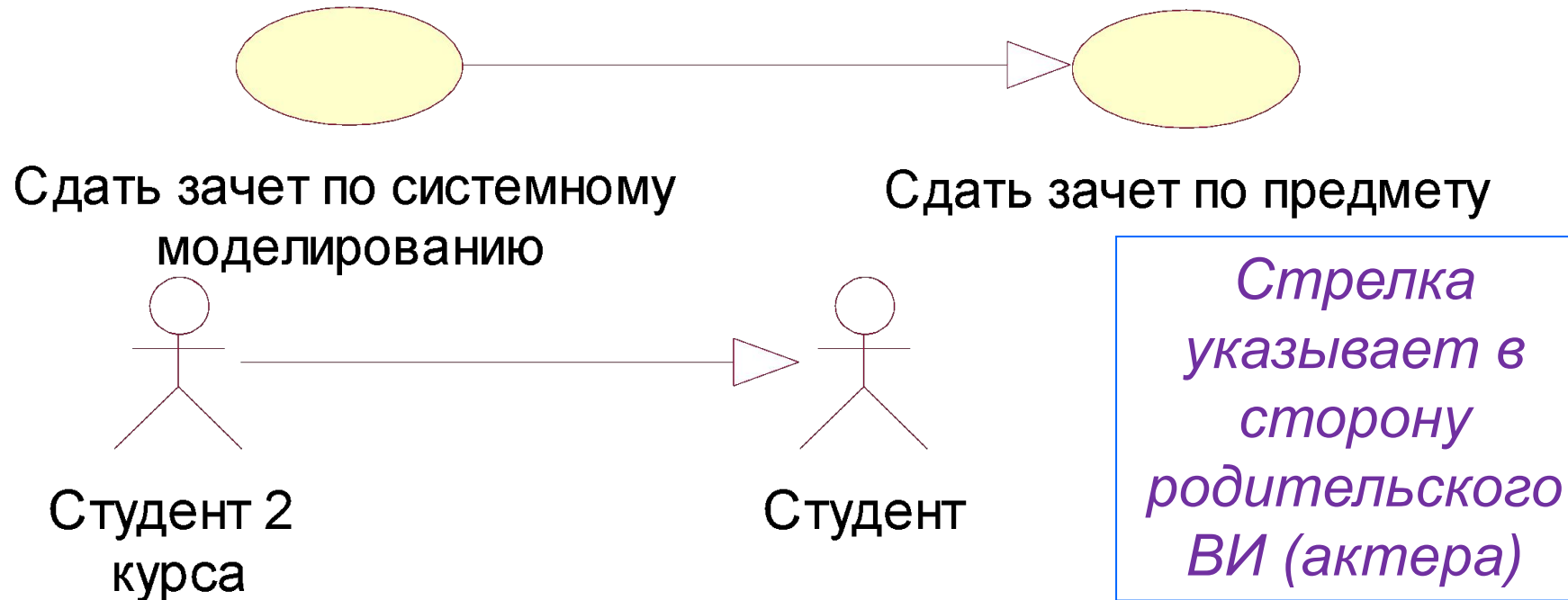
Отношение включения

- Указывает, что некоторое заданное поведение для одного варианта использования *включается в качестве составного компонента* в последовательность поведения другого варианта использования.



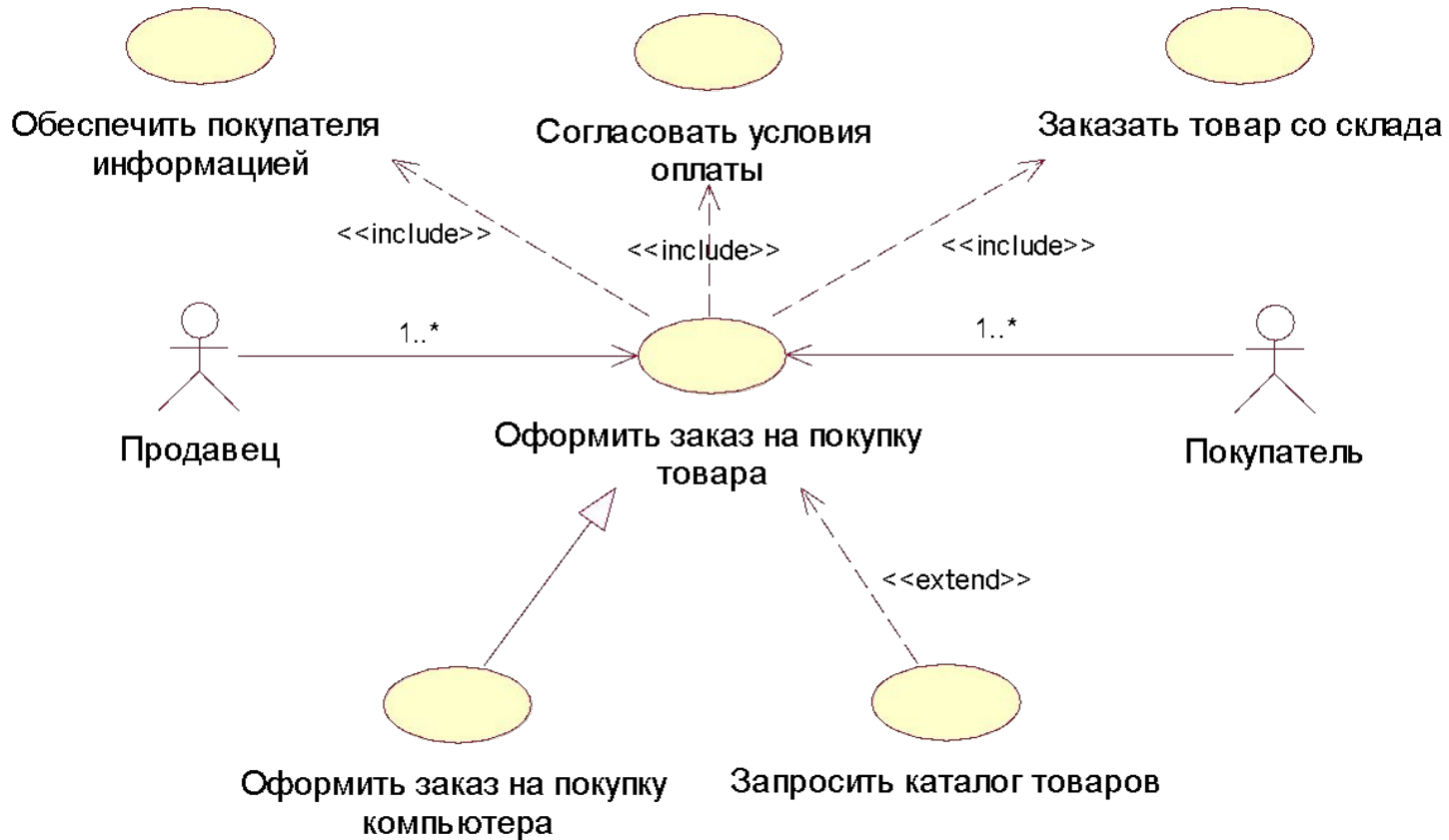
Отношение обобщения

- Служит для указания того факта, что некоторый вариант использования *А* может быть обобщен до варианта использования *Б* (или актер *А* может быть обобщен до актера *Б*).



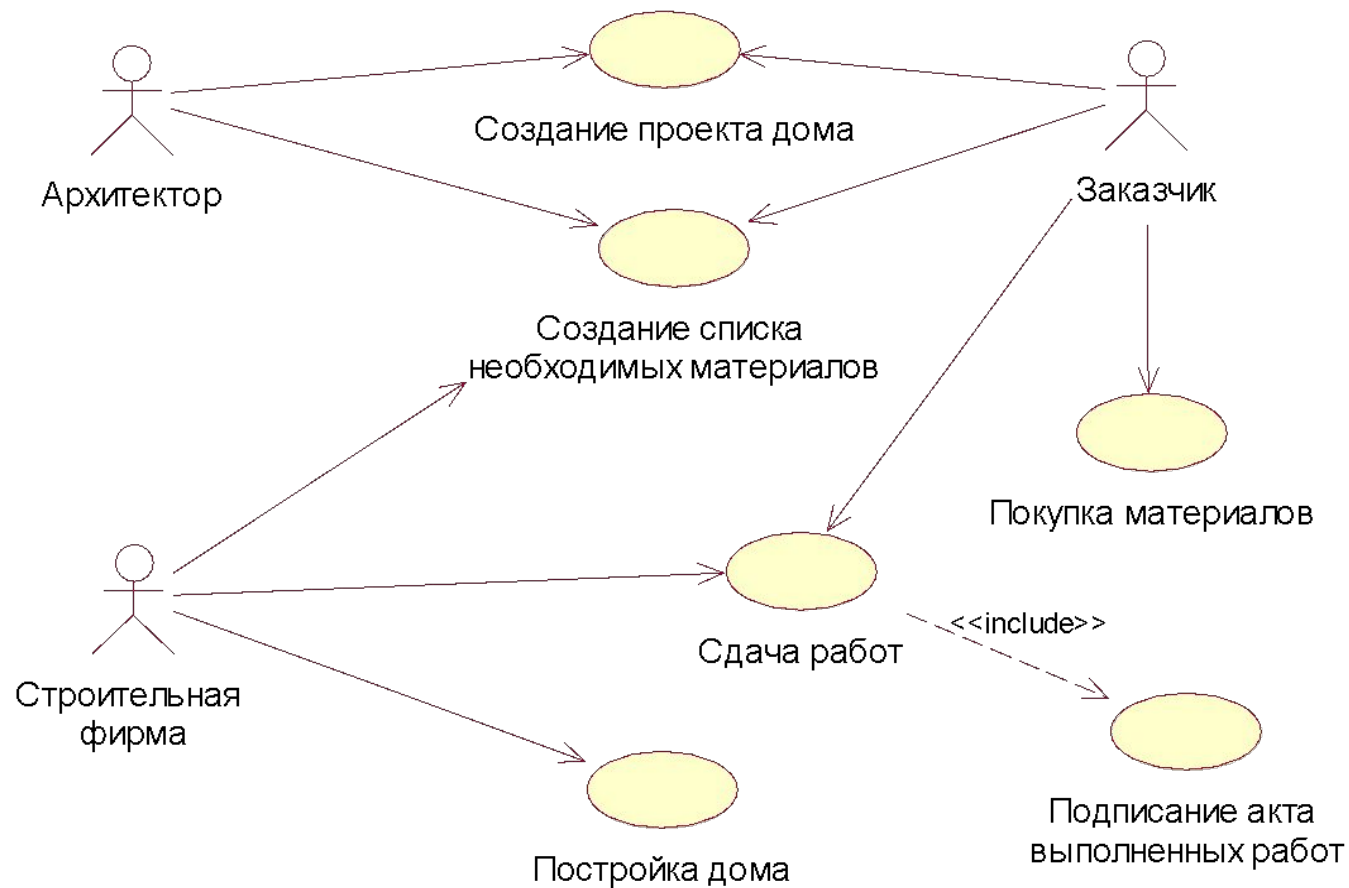
Примеры

- ДВИ процесса оформления заказа на покупку товара



Примеры

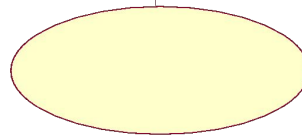
- Диаграмма прецедентов для процесса постройки дома



Примечание как элемент ДВИ

- **Примечание (Note)** в языке UML предназначено для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта.
- Примечание может относиться к любому элементу диаграммы.

Реализовать в форме
печати чека с указанием
реквизитов



Получение справки о
состоянии счета



Пример 1. Разблокировать учетную запись пользователя (простой короткий пример, без альтернативного потока событий):

Действующие лица	Администратор, Система
Цель	Изменить статус учетной записи пользователя на «активный».
Предусловие	Учетная запись пользователя не активна.
Успешный сценарий: <ol style="list-style-type: none">1. Администратор выбирает пользователя и активирует «Разблокировать».2. Система переключает учетную запись пользователя в статус «активный», и посылает сообщение (тут можно сослаться на текст сообщения из списка сообщений, см. примечание ниже) пользователю на email (если «User Account → email» не пусто).	
Результат	Учетная запись пользователя была переведена в статус «активный».

Пример 2. Авторизация пользователя:

Действующие лица	Пользователь, Система
Цели	Пользователь: авторизоваться в системе и начать работать; Система: идентифицировать пользователя и его права.
Успешный сценарий:	
<ol style="list-style-type: none">1. Пользователь запускает систему. Система открывает сессию пользователя, предлагает ввести логин и пароль.2. Пользователь вводит логин и пароль.3. Система проверяет логин и пароль.4. Система создает запись в истории авторизаций (IP адрес пользователя, логин, дата, рабочая станция).5. Система выдает пользователю сообщение по поводу успешной авторизации (ссылка на сообщение).	
Результат	Пользователь успешно авторизирован и может работать с системой.
Расширения:	
*а	Нет доступа к БД. Система выдает сообщение (ссылка на сообщение). Результат: пользователь не может войти.
1а	В настройках безопасности для данного IP адреса существует запрет на вход в систему. Результат: форма логина не предоставляется, система выдает сообщение пользователю (ссылка на сообщение).
2а	Пользователь выбирает: «Напомнить пароль». Вызывается сценарий «Напомнить пароль».
3а	Пользователь с введенными логином и паролем не найден. Результат: отказ в авторизации. Система выдает сообщение (ссылка на сообщение). Переход на шаг 2.
3б	Количество неудачных попыток авторизоваться достигло максимального, установленного в настройках. Результат: пользователь не может войти. Выдается сообщение: (ссылка на сообщение). Вход с IP адреса Пользователя заблокирован на время, установленное в настройках.



Диаграмма классов

Лекция 2





План лекции

- Что такое диаграмма классов
- Компоненты диаграммы классов и их назначение
- Пример диаграммы классов
- Расширение языка UML для построения моделей программного обеспечения и бизнес-систем



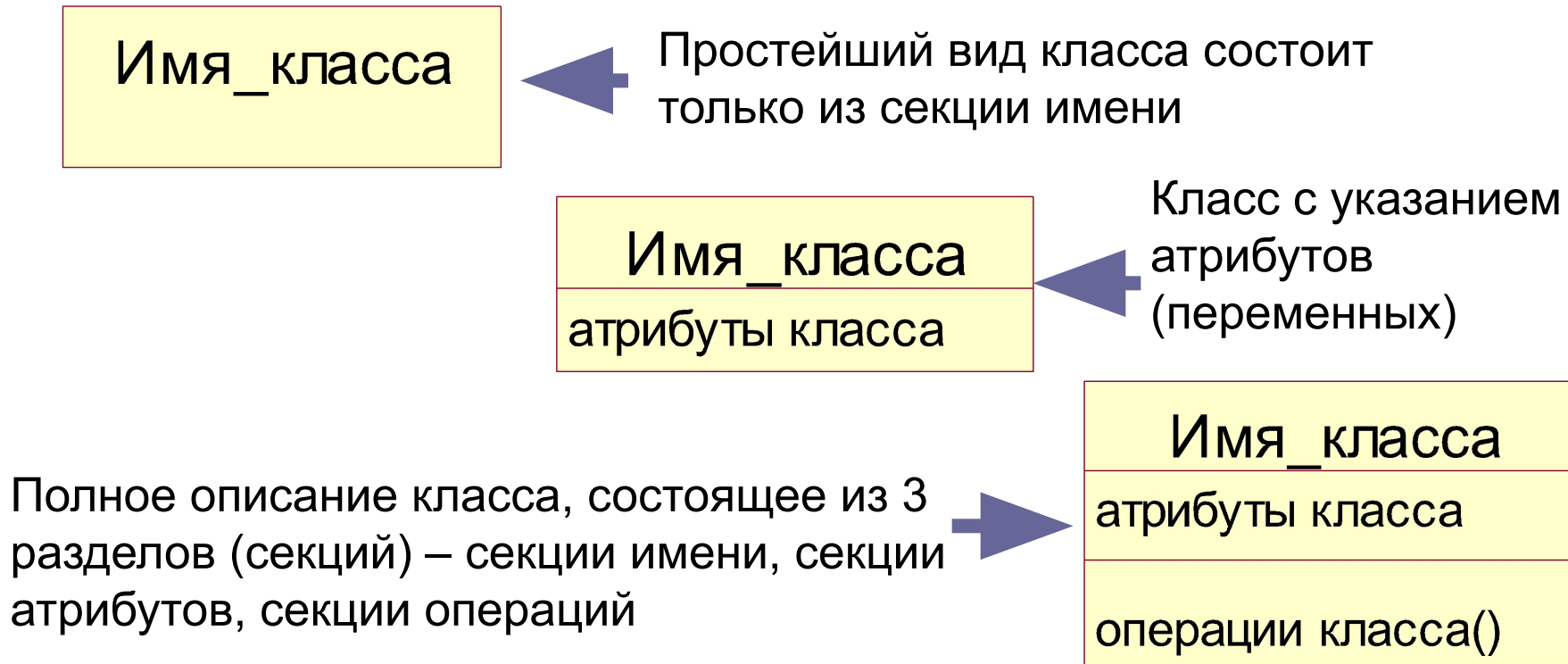
Диаграмма классов

- Является центральным звеном объектно-ориентированного подхода
- Содержит информацию об объектах системы и статических связях между объектами
- Отражает *декларативные знания* о предметной области
- Оперирует понятиями *класса, объекта, отношения, пакета*



Класс

- **Класс** – это множество объектов, которые обладают *одинаковой* структурой, поведением и отношениями с объектами из других классов.



Класс

- **Имя класса** должно быть уникально
- Имя класса должно начинаться с заглавной буквы.
- Класс может не иметь экземпляров или объектов. В этом случае он называется **абстрактным классом**, а для обозначения его имени используется *курсив*



Атрибуты класса

- **Атрибут = свойство**, которое является общим для всех объектов данного класса
- Общий формат записи атрибутов:
*<квантор видимости> <имя атрибута>
[кратность]: <тип атрибута> = <исходное значение> {строка-свойство}*



Атрибуты класса. Квантор видимости

- Квантор видимости может принимать одно из следующих значений: +, #, -, ~.
- «+» - атрибут с областью видимости типа **общедоступный** (public).
- «#» - атрибут с областью видимости типа **защищенный** (protected).
- «-» - атрибут с областью видимости типа **закрытый** (private).
- «~» - атрибут с областью видимости типа **пакетный** (package).



Атрибуты класса.

Имя атрибута

- Представлено в виде *уникальной* строки текста
- Имя атрибута является единственным обязательным элементом в синтаксическом обозначении атрибута
- Должно начинаться со строчной буквы
- По практическим соображениям записывается *без пробелов*



Атрибуты класса.

Кратность атрибута

- ***Кратность атрибута*** характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса.
- Формат: *[нижняя граница .. верхняя граница]*
- ***Примеры: [0..1], [0..*], [1..3,5..7]***



Атрибуты класса. Тип атрибута

- Выражение, определяемое некоторым ТИПОМ ДАННЫХ (например, в зависимости от языка программирования)
- В простейшем случае – *осмысленная строка текста.*
- Пример:
 - цвет: Color*
 - имяСотрудника[1..2]: String;*
 - видимость: Boolean*



Атрибуты класса. Исходное значение

- Служит для задания некоторого начального значения в момент *создания* отдельного экземпляра класса

- Пример:

цвет: Color = (255, 0, 0)

имяСотрудника[1..2]: String = 'Иван Иванов';

видимость: Boolean = истина



Атрибуты класса. Строка-свойство

- Служит для указания **дополнительных свойств атрибута**, которые могут характеризовать особенности изменения значений атрибута в ходе выполнения соответствующей программы.
- Это значение принимается за **исходное значение атрибута**, которое не может быть изменено в дальнейшем.
- Пример:

заработнаяПлата: Currency = \$500 {frozen}

Операции класса

- Представляют собой некоторый сервис, который предоставляет каждый экземпляр класса или объект по требованию своих клиентов.
- Правила записи операций:
<квантор видимости> <имя операции> (список параметров): <выражение типа возвращаемого значения> {строка-свойство}



Операции класса. Список параметров

- *Список параметров* является перечнем разделенных запятой формальных параметров, каждый из которых, в свою очередь, может быть представлен в следующем виде:

*<вид параметра> <имя параметра> :
<выражение типа> = <значение параметра по умолчанию>*



Операции класса. Строка-свойство

- **Строка-свойство** служит для указания значений свойств, которые могут быть применены к данной операции.
- Например, для указания последовательности действий будет использована строка-свойство вида:

{concurrency = ИМЯ} ,

где *ИМЯ* может принимать одно из следующих значений:

- **sequential** (последовательная),
- **concurrent** (параллельная),
- **guarded** (охраняемая)

Операции класса. Примеры

- +нарисовать (форма : Многоугольник = прямоугольник, цветЗаливки : Color = (0, 0, 255));
- -изменитьСчетКлиента (номерСчета : Integer) : Currency;
- #выдатьСообщение() : ('Ошибка деления на ноль').



Отношения между классами

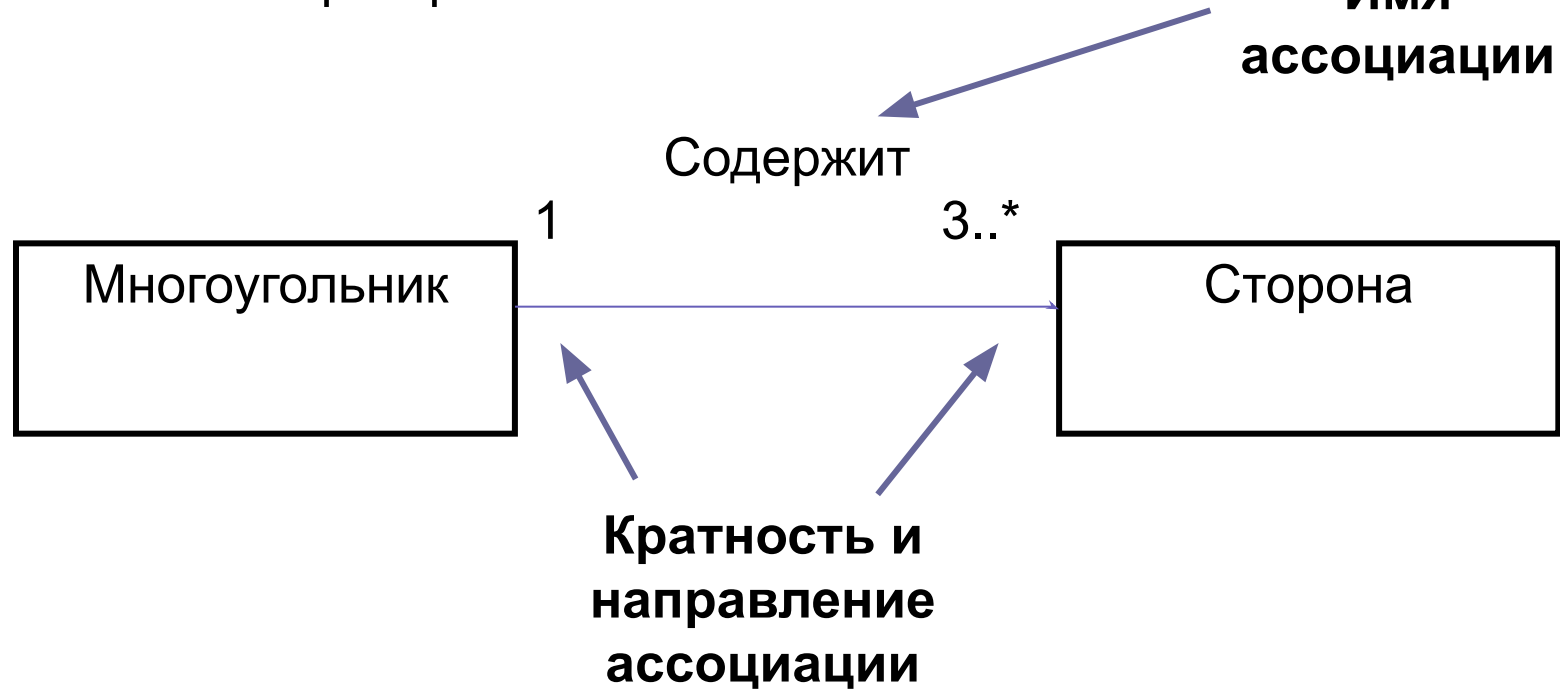
Базовыми отношениями на диаграмме классов являются:

- отношения **ассоциации** (*association*);
- отношения **обобщения** (*generalization*);
- отношения **агрегации** (*aggregation*);
- отношения **композиции** (*composition*);
- отношения **зависимости** (*dependency*);
- отношения **реализации** (*realization*).



Отношение ассоциации

- **Отношение ассоциации** свидетельствует о наличии произвольного отношения между классами.
- На диаграммах ассоциация обозначается сплошной линией.

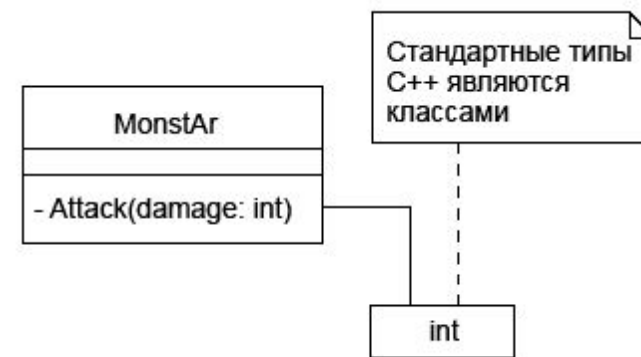


Ассоциация - самый слабый вид связи.

Обычно ассоциация возникает, когда один класс вызывает метод другого или если при вызове метода в качестве аргумента передаётся объект другого класса.

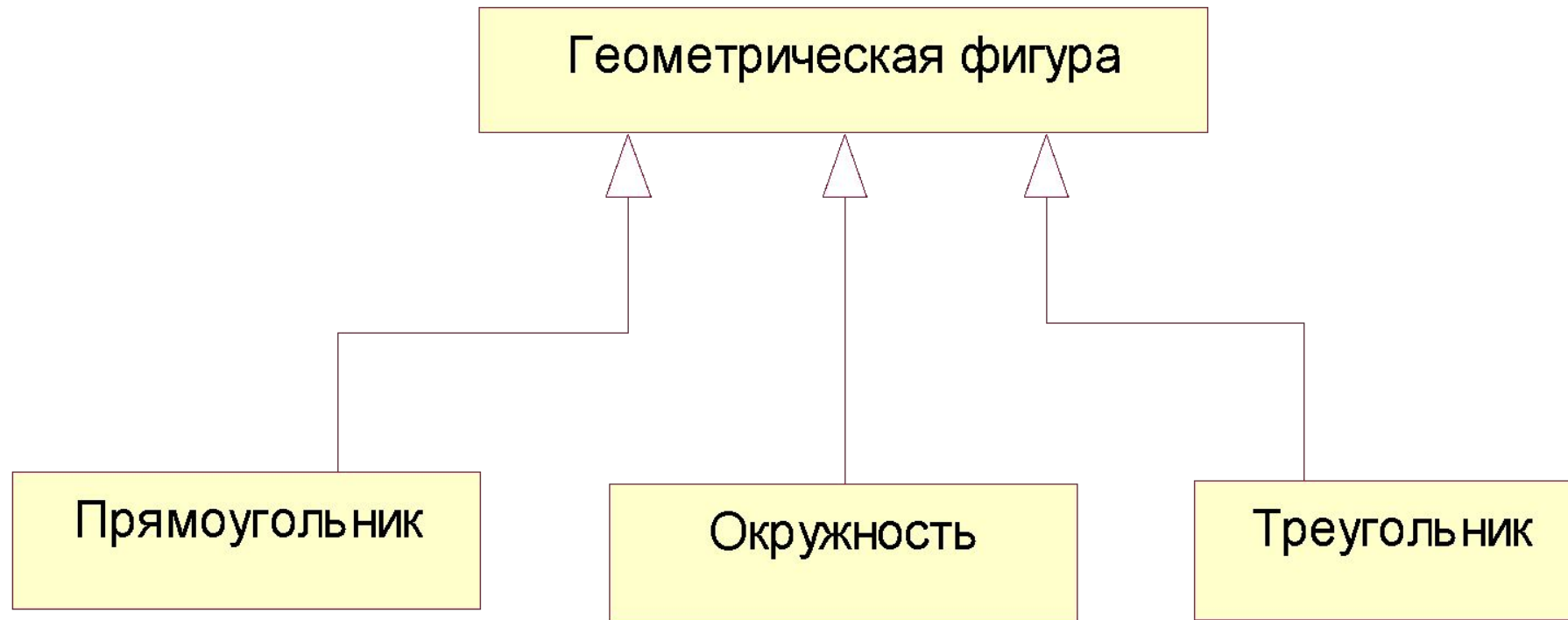
Для примера напишем простой класс:

```
class MonstAr
{
private:
    attack(int damage) // damage - урон
}
};
```



Отношение обобщения

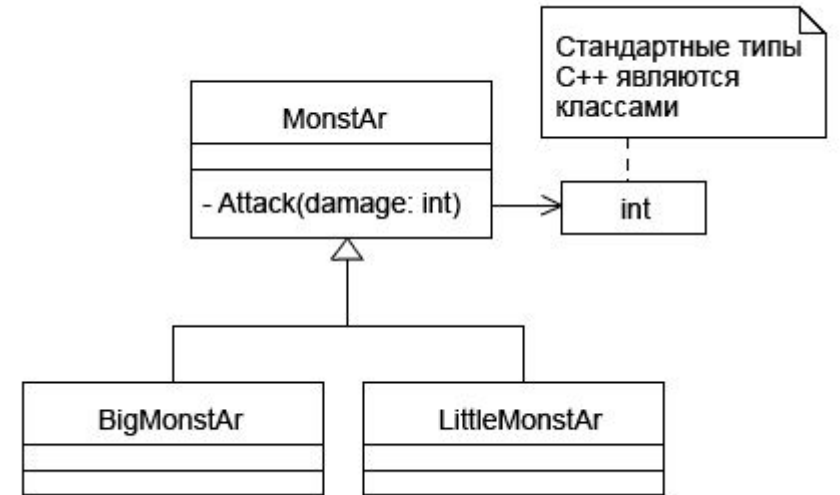
- Является отношением *классификации* между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком)
- При обобщении рисуется сплошная линия. Стрелочка - пустой треугольник.




```
MonstAr
{
private:
    attack(int damage) // damage - урон
    {}
};
```

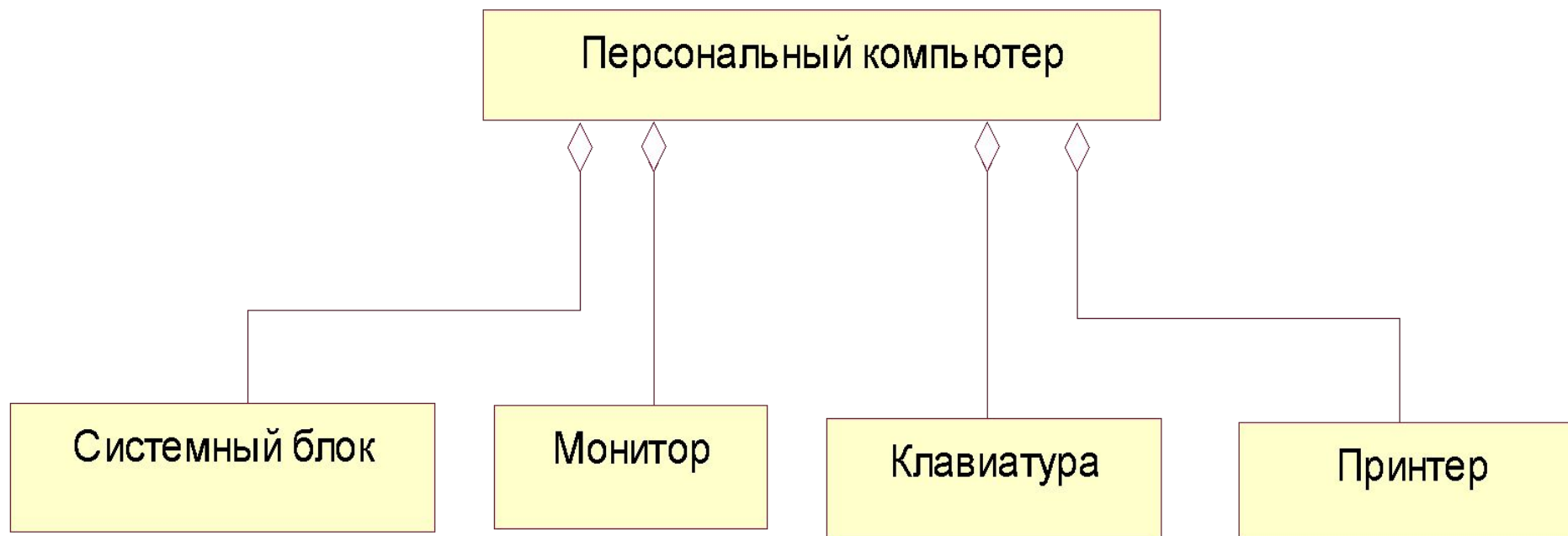
```
BigMonstAr : public MonstAr // большой
{
    // определение класса
};
```

```
SmallMonstAr : public MonstAr // маленький
{
    // определение класса
};
```



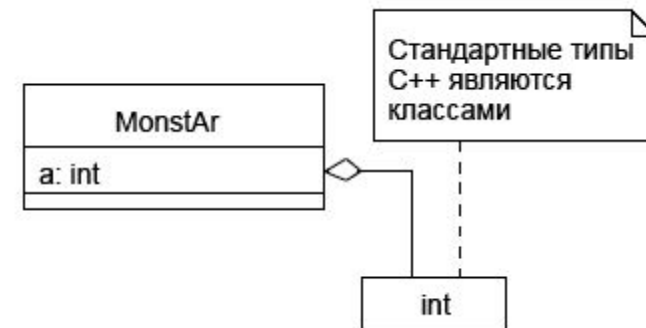
Отношение агрегации

- Смысл: один из классов представляет собой некоторую сущность, которая **включает** в себя в качестве составных частей другие сущности.
- Применяется для представления системных взаимосвязей типа «часть-целое».
- На диаграммах агрегация показывается незакрашенным ромбом.



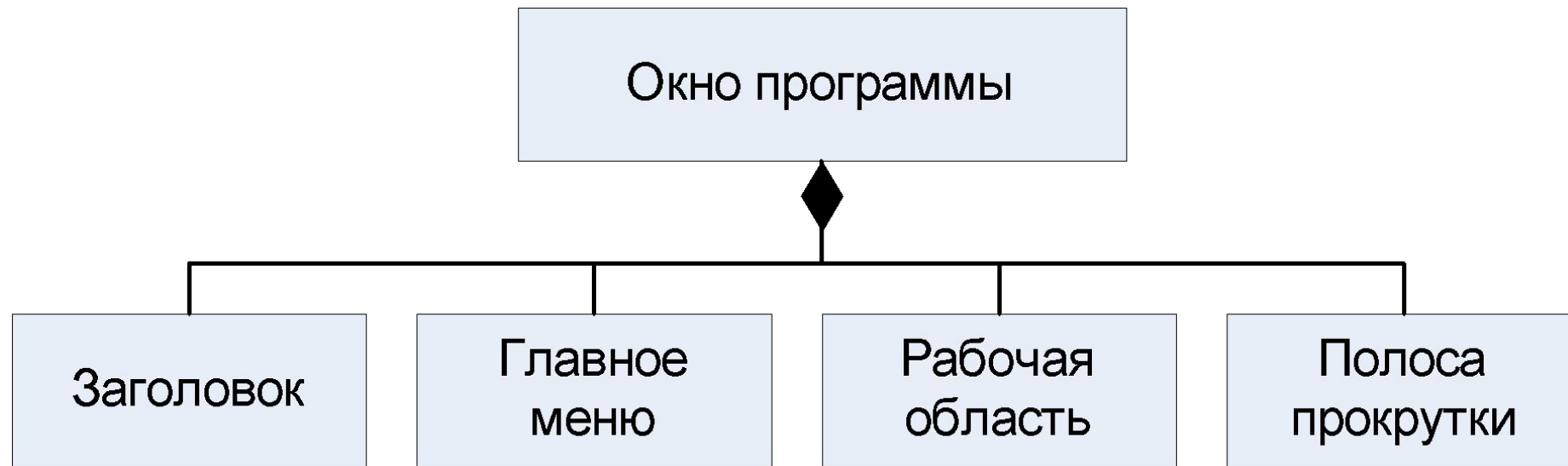
Итак, в UML агрегация отражает связь классов, когда объект одного класса является атрибутом другого. Пример:

```
class MonstAr
{
public:
    int a;
};
```



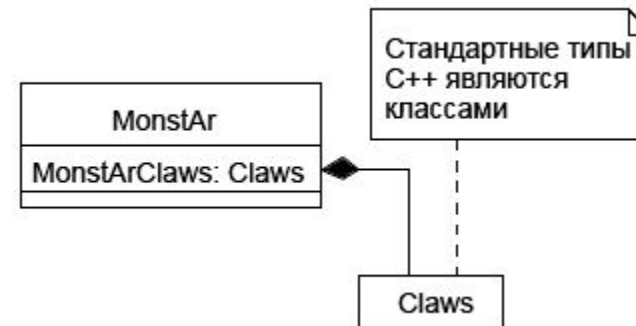
Отношение композиции

- Является частным случаем отношения агрегации.
- Части не могут выступать в отрыве от целого, т.е. с уничтожением целого уничтожаются составные части.



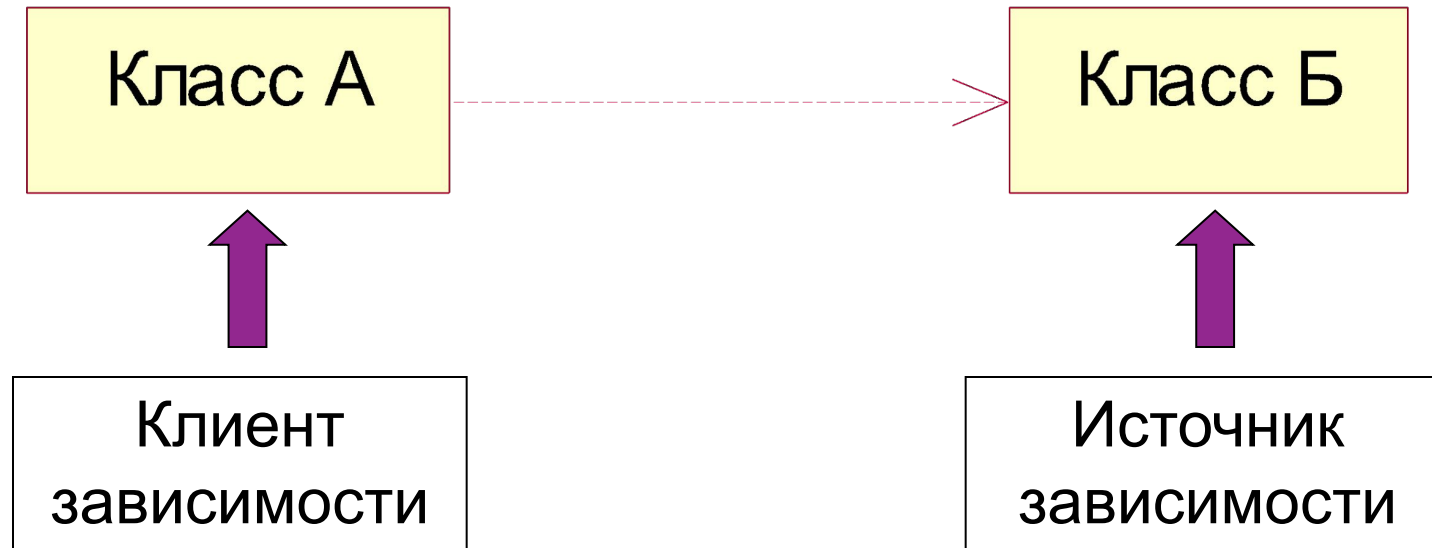
```
class Claws; // claws - когти
```

```
class MonstAr  
{  
public:  
    Claws MonstArClaws;  
};
```



Отношение зависимости

- Используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого элемента.



Также возникает, когда один класс вызывает метод другого или если при вызове метода в качестве аргумента передаётся объект другого класса, влияющего на объект первого класса.



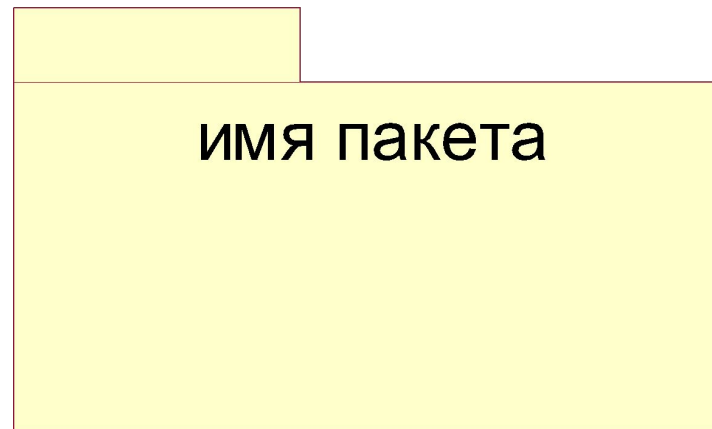
Отношение реализации

- **Отношение реализации** свидетельствует о реализации классом некоторого интерфейса.
- На диаграмме реализация показывается пунктирной линией и незакрашенной стрелочкой

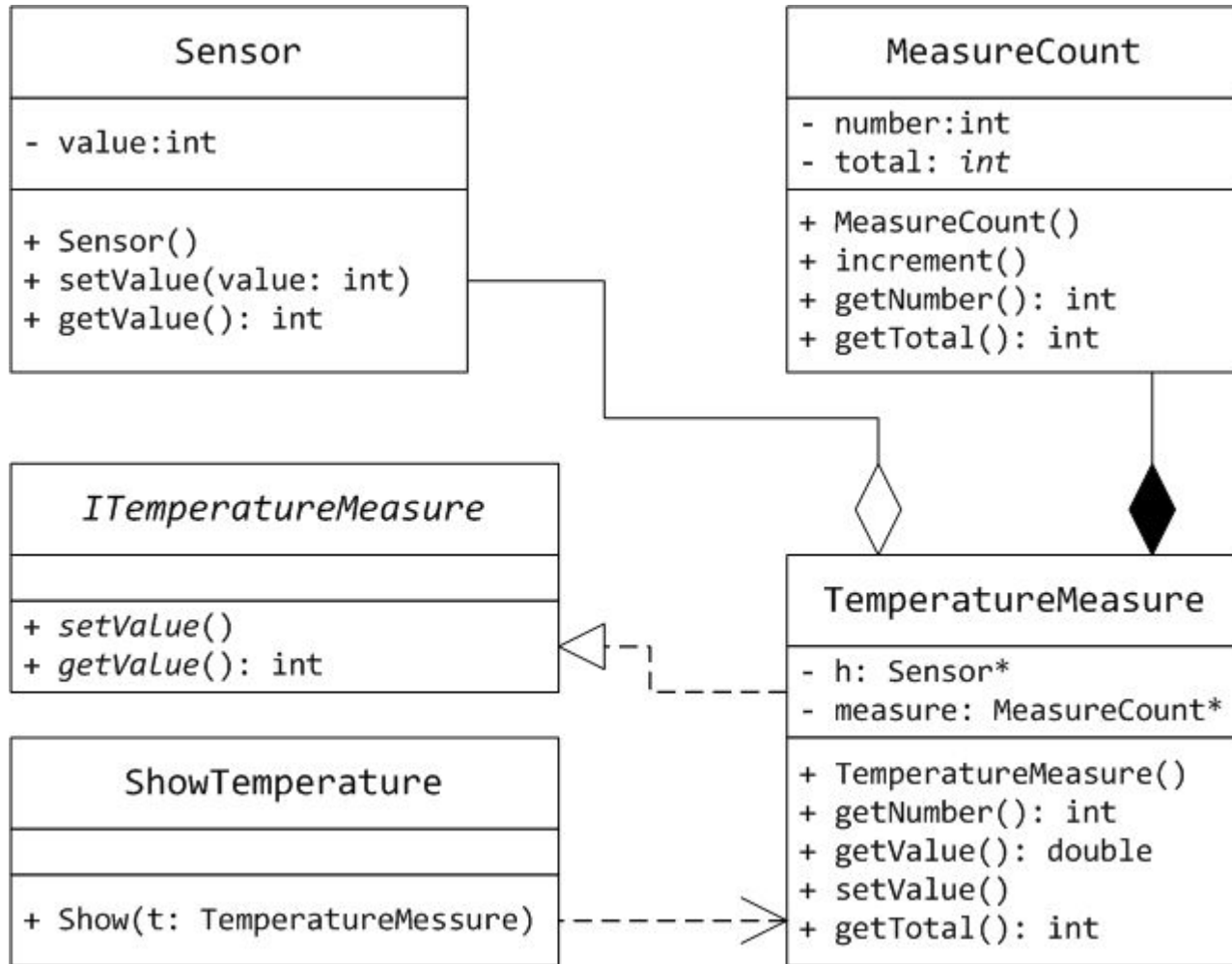


Пакеты

- служат для **группировки** элементов модели
- Любой пакет владеет своими элементами
- любой элемент может принадлежать *только одному пакету*



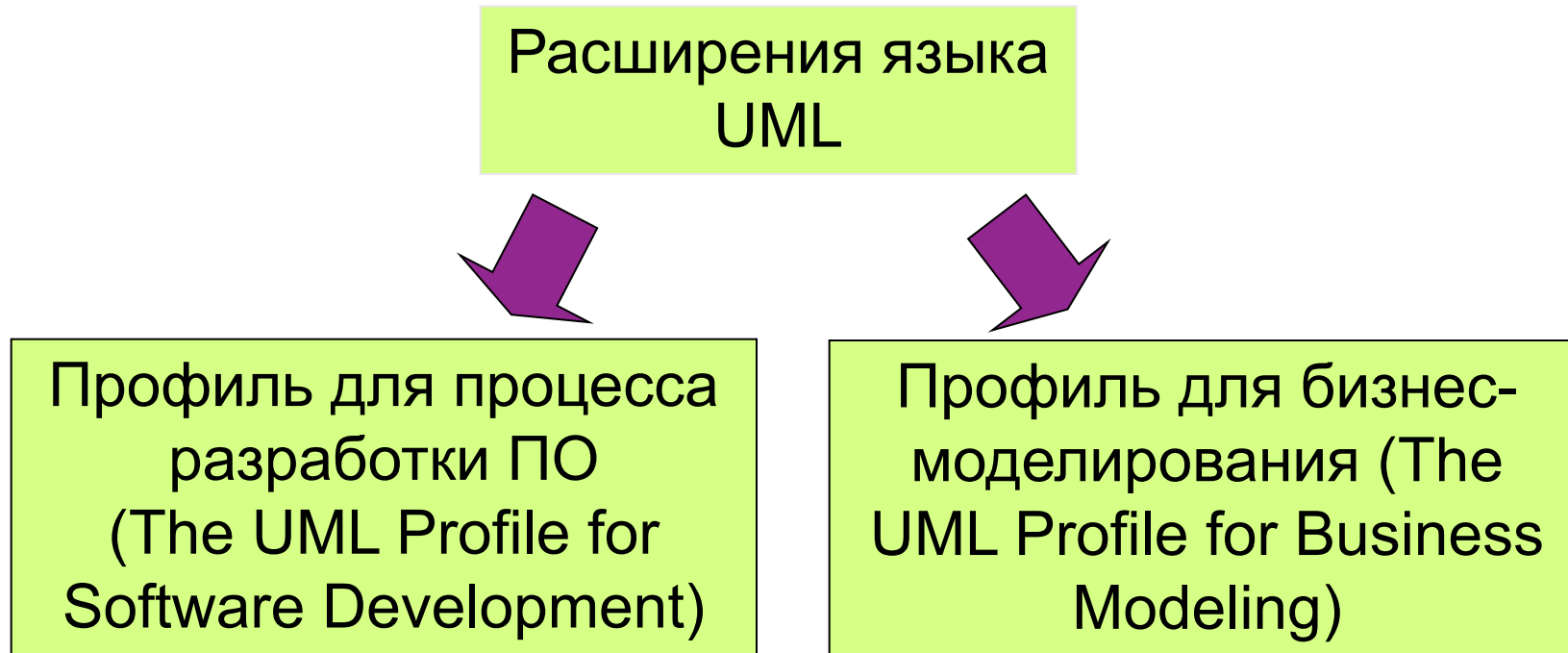
Пример диаграммы классов



На диаграмме классов основным классом является класс `TemperatureMeasure`, который и является измерителем температуры. В качестве измеренного значения формируется среднее арифметическое всех измерений - сумма всех измерений, деленная на их количество. Для получения измерений и их суммирования используется класс `Sensor` (в качестве датчика температуры). В консольной задаче сами измерения передаются в этот класс для суммирования. Класс состоит в отношении агрегации с основным классом `TemperatureMeasure`: мы сначала создаем объект класса `Sensor`, а потом передаем его в качестве параметра конструктора классу `TemperatureMeasure`, чтобы использовать его в качестве части класса.

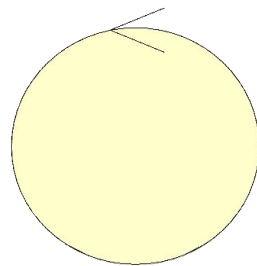
Количество измерений формируется классом `MeasureCount`, который содержит статическое свойство `total` для подсчета общего измерений, а также свойство `count` для подсчета количества измерителей конкретного объекта `TemperatureMeasure`. Класс `MeasureCount` находится в отношении композиции с классом `TemperatureMeasure`: объект `MeasureCount` создается непосредственно при создании объекта `TemperatureMeasure` (в его конструкторе). Класс `ITemperatureMeasure` представляет собой интерфейс класса `TemperatureMeasure` и является своего рода **поставщиком** в отношении реализации. Наконец, класс `ShowTemperature` находится в отношении зависимости с классом `TemperatureMeasure`, поскольку реализация единственного метода `Show` класса `ShowTemperature` зависит от структуры класса `TemperatureMeasure`.

Расширения языка UML



Профиль для процесса разработки ПО

- **Управляющий класс (control)** – отвечает за координацию действий других классов.

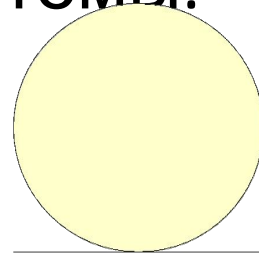


NewClass



Профиль для процесса разработки ПО

- **Класс-сущность (entity)** содержит информацию, которая должна храниться **постоянно** и не уничтожаться с уничтожением объектов данного класса или прекращением работы моделируемой системы.

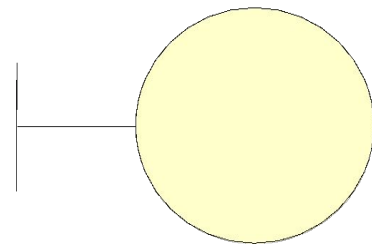


NewClass2



Профиль для процесса разработки ПО

- **Граничный класс (boundary)** – располагается на границе системы с внешней средой, но является составной частью системы.

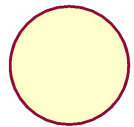


NewClass3

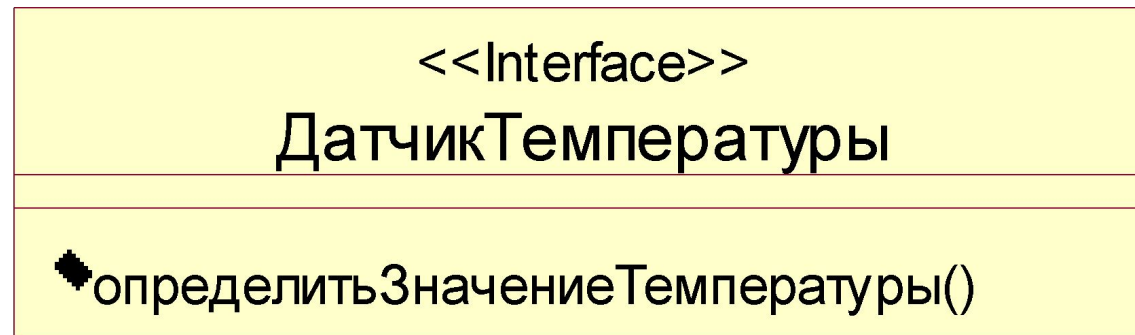


Интерфейс (interface)

- в контексте языка UML является специальным случаем класса, у которого имеются только операции и отсутствуют атрибуты.



ДатчикТем
пературы





Диаграммы взаимодействия

Лекция 3





План лекции

- Что такое диаграммы взаимодействия?
- Виды диаграмм взаимодействия
- Основные компоненты (для каждого вида диаграмм)
- Примеры
- Общее и различное между видами диаграмм взаимодействия



Что такое диаграммы взаимодействия?

- **Диаграмма классов** представляет собой логическую модель статического представления моделируемой системы
- Однако ***элементы системы*** всегда ***взаимодействуют между собой***
- В языке UML это взаимодействие элементов рассматривается в информационном аспекте, т.е. объекты обмениваются некоторой информацией.
- => **Диаграммы взаимодействий являются моделями, описывающими поведение взаимодействующих групп объектов.**

Виды диаграмм взаимодействия

Существуют 2 вида диаграмм взаимодействий:

- 1) **диаграммы последовательности действий** – *sequence diagram*;
- 2) **диаграммы кооперации** (кооперативные диаграммы) – *collaboration diagram*



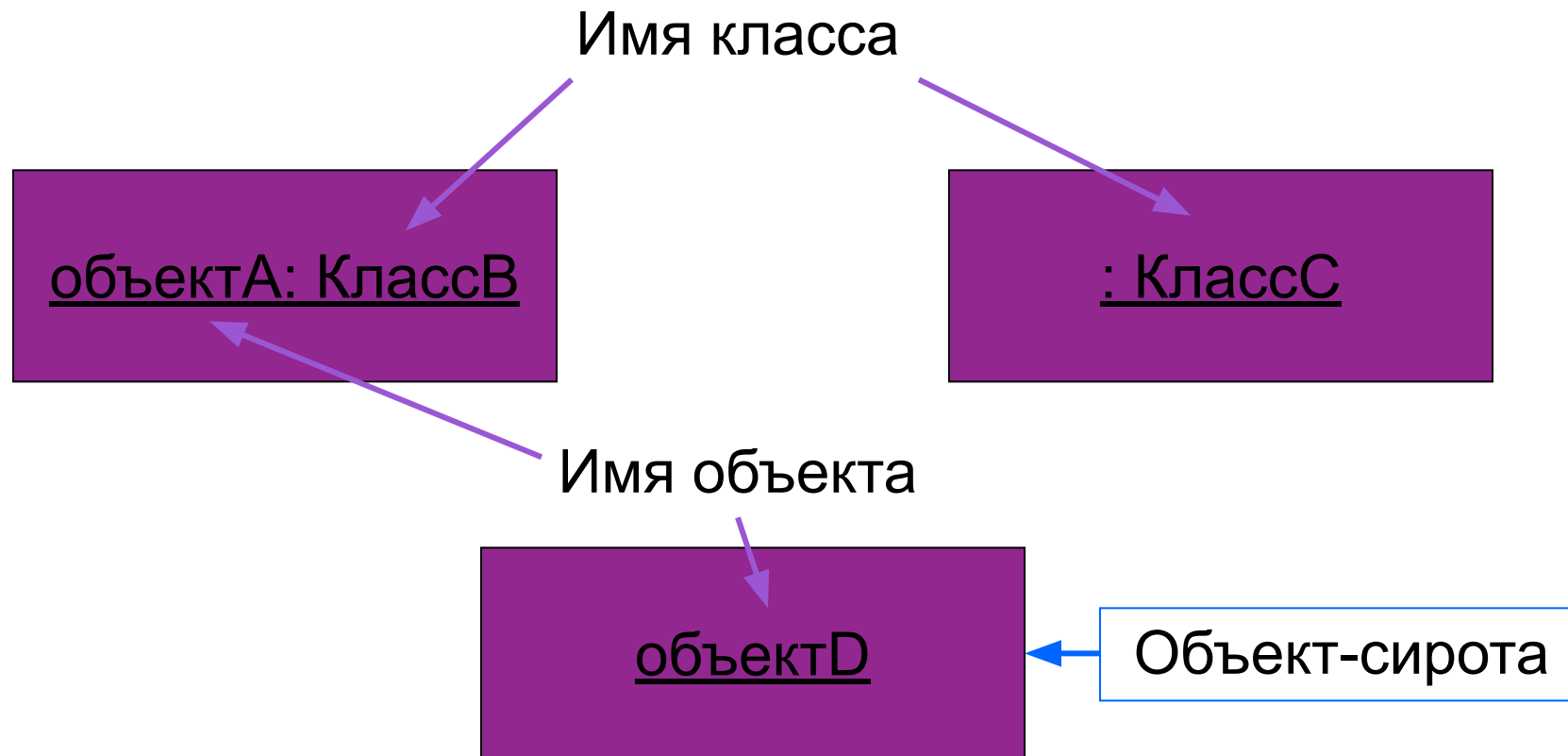
Диаграммы последовательности действий

- Диаграммы последовательности действий отображают взаимодействие объектов, упорядоченное по времени.
- Основными компонентами диаграмм последовательности действий являются:
 - **Объекты;**
 - **Линия жизни;**
 - **Сообщения.**

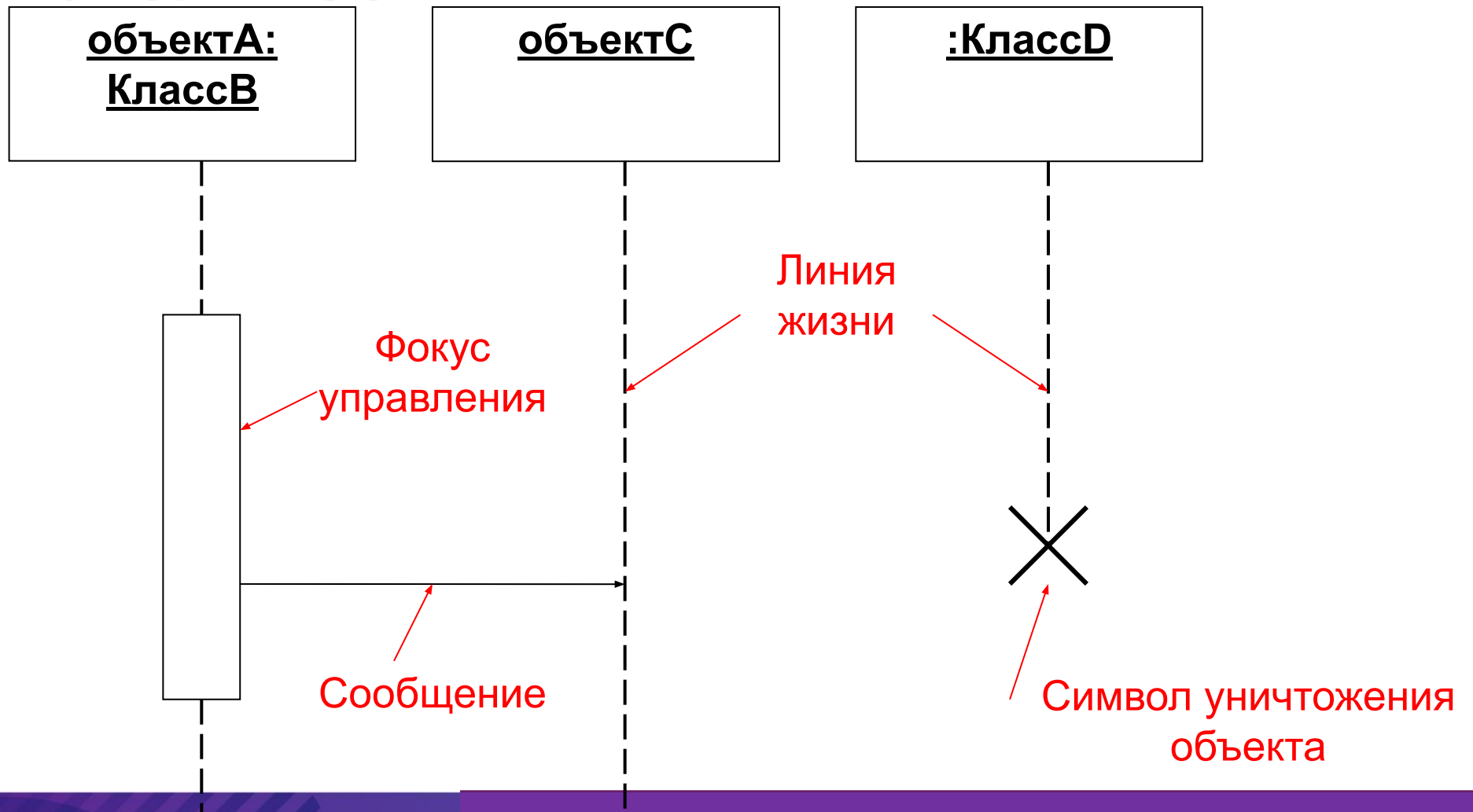


Объекты

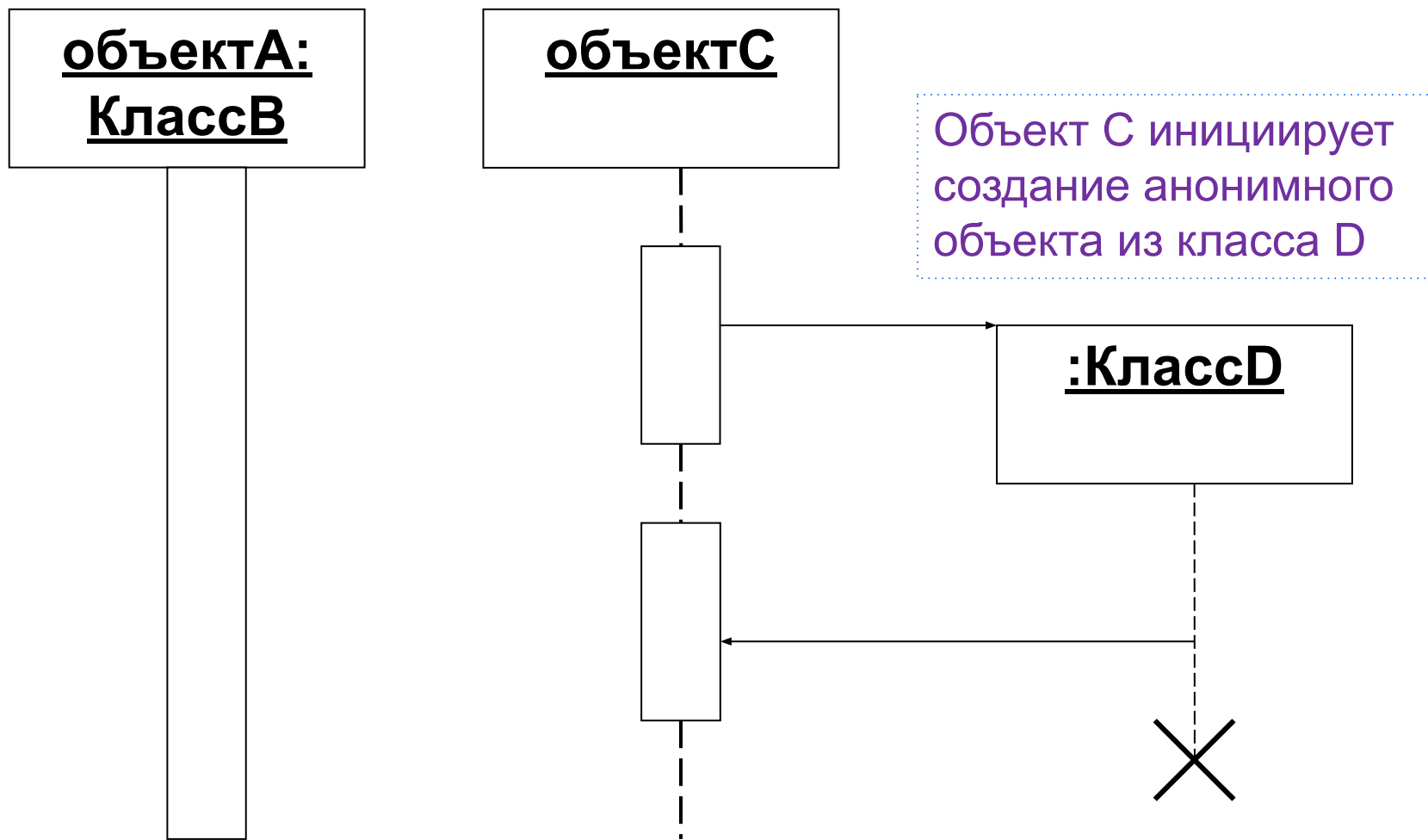
- Объект – экземпляр класса.



Графические элементы диаграммы последовательности



Линия жизни и фокус управления



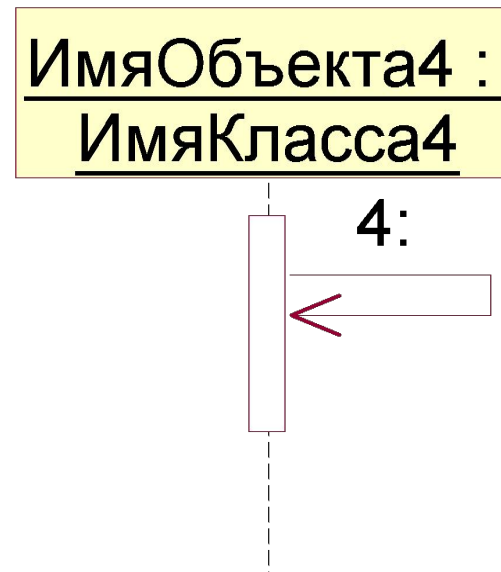
Сообщение

- Представляет собой законченный фрагмент информации, который отправляется одним объектом другому;
- Прием сообщения инициирует выполнение определенных действий;
- **разновидности сообщений**

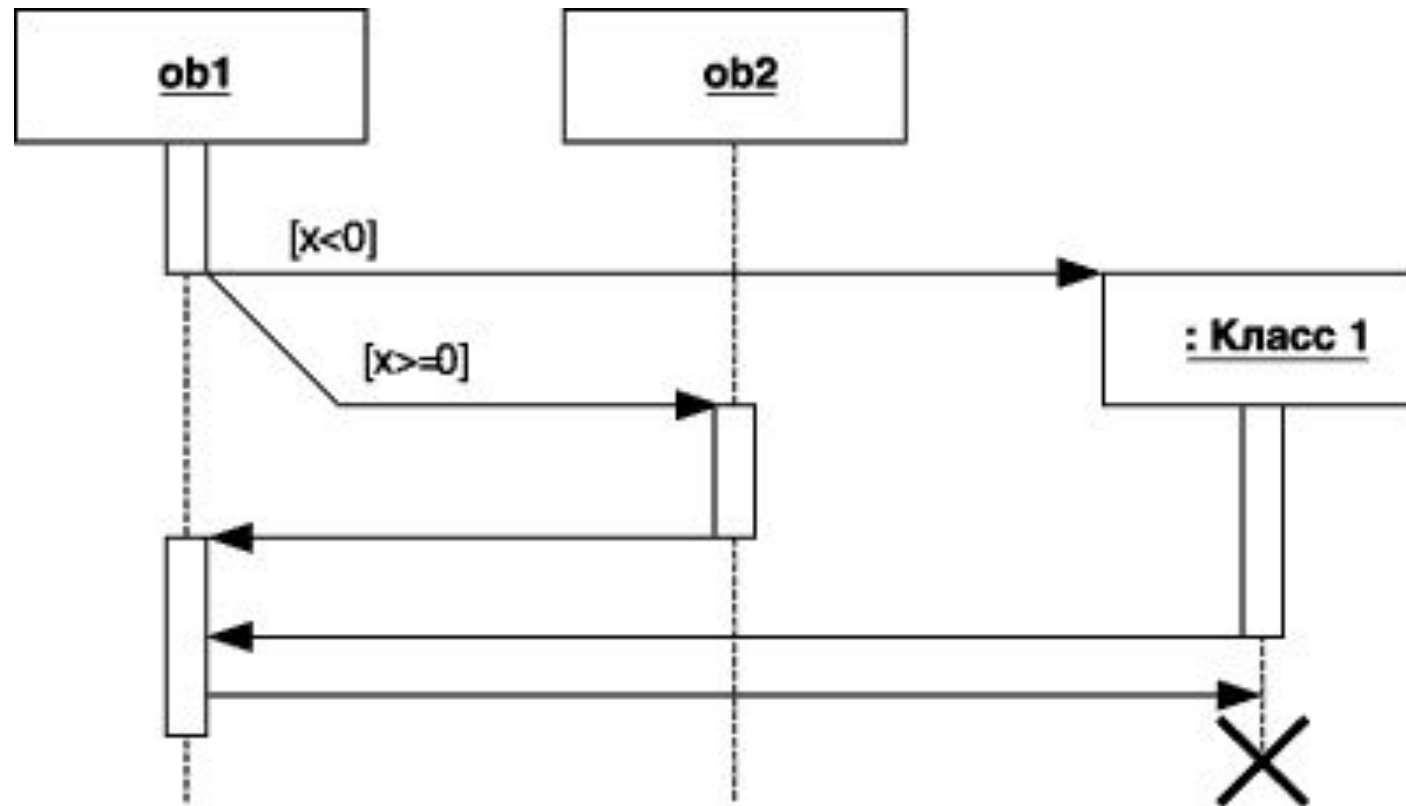


Сообщение

- Сообщение, отправленное самому себе – рефлексивное (саморегулирование).



Ветвление потока



Пример диаграммы последовательности

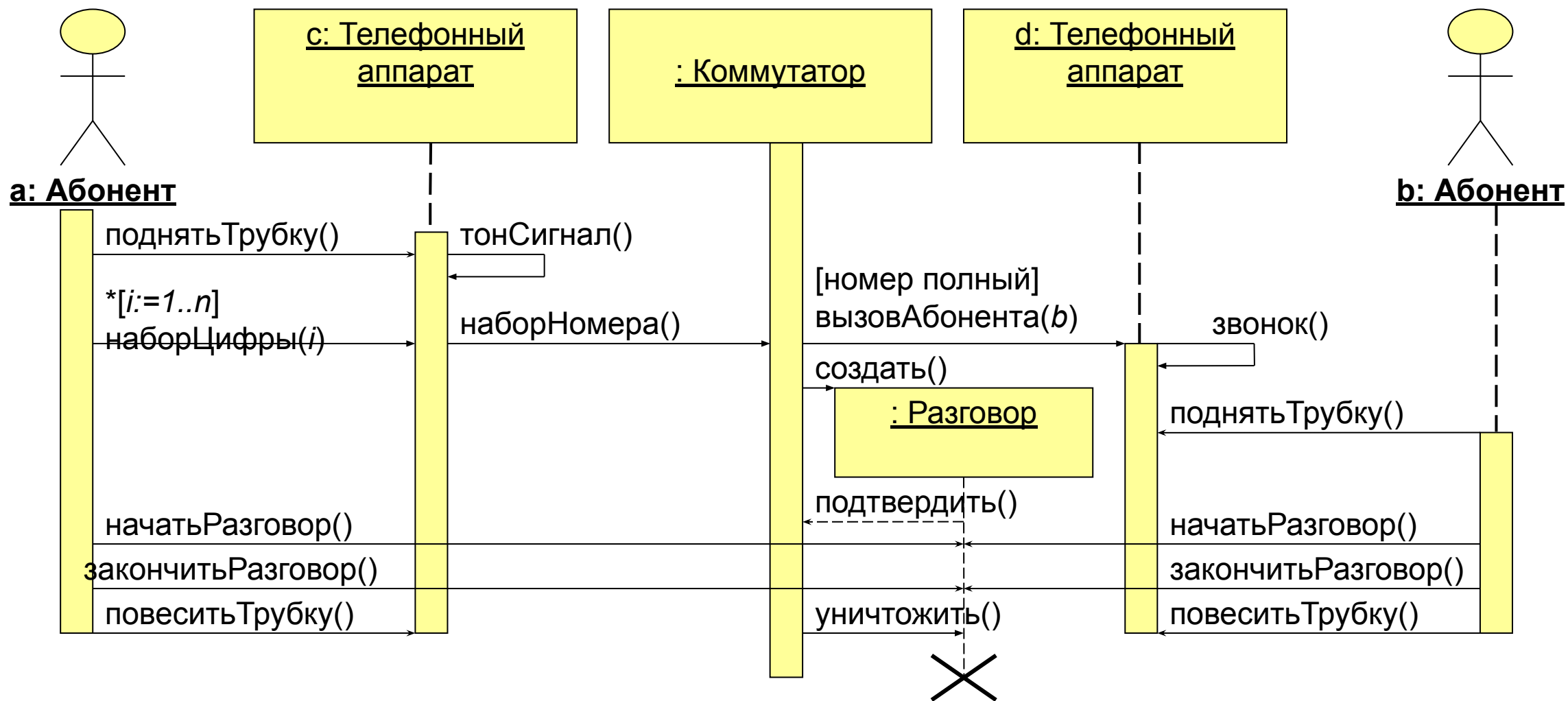


Диаграмма кооперации

- *Поведение системы* описывается на уровне отдельных объектов, которые обмениваются между собой сообщениями, чтобы достичь определенной цели или реализовать некоторый вариант использования.
- **Кооперация**. (*collaboration*) - служит для обозначения множества взаимодействующих с определенной целью объектов в общем контексте моделируемой системы.



Основные компоненты

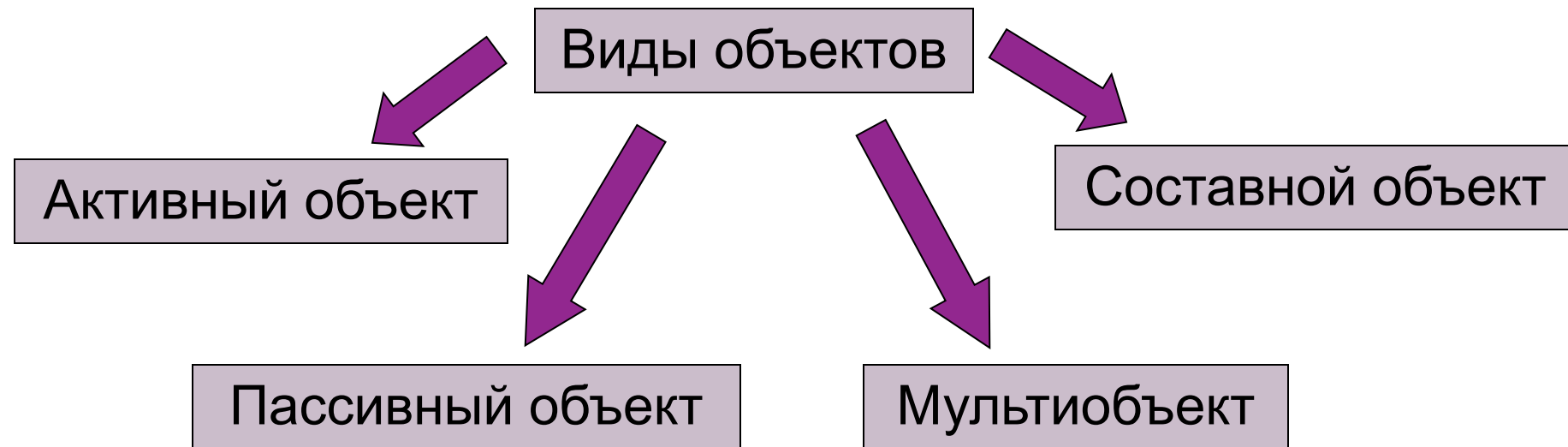
Основные компоненты диаграммы кооперации:

- **объекты;**
- **связи;**
- **сообщения.**



Объекты

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



Мультиобъект

- Представляет собой множество объектов, которые могут быть образованы на основе класса.



: Мультиобъект



Активный объект

- В контексте языка UML объекты делятся на активные и пассивные.



- *Активный объект* имеет свой собственный поток управления и может инициировать деятельность по управлению другими объектами.

Составной объект

- Предназначен для представления объекта, имеющего сложную структуру и внутренние потоки управления.

а: Графическое окно

: Строка заголовка

: Полоса прокрутки

: Рабочая область



Сообщение

- В общем смысле под *сообщением* понимается законченный фрагмент информации, посылаемый одним объектом другому.



1)



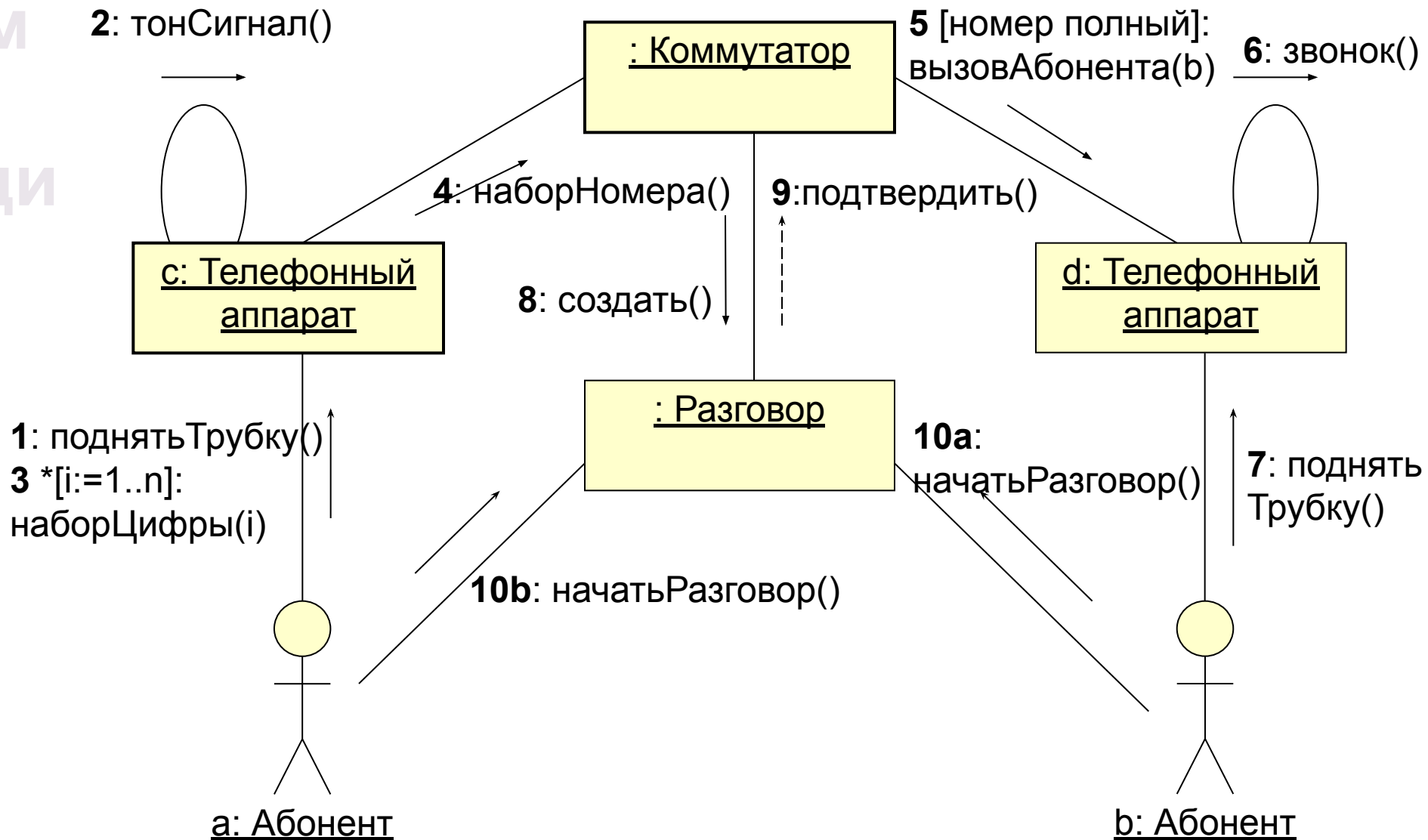
2)



3)



пример
диаграмм
ы
коопераци
и



Основное отличие между диаграммой последовательности и кооперации

- На **диаграмме кооперации** изображаются только такие отношения между объектами, которые играют роль информационных каналов при взаимодействии.
- На **диаграмме кооперации** не указывается время в виде дополнительного измерения.
- Таким образом, в диаграмме последовательности делается акцент на временной аспект, в диаграмме кооперации – на статическое взаимодействие объектов системы.





Диаграммы состояний, деятельности, компонентов, развертывания

Лекция 4



ПОЛОЦКИЙ
ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ



План лекции

- Назначение диаграммы состояний,
- Назначение диаграммы деятельности
- Назначение диаграммы компонентов
- Назначение диаграммы развертывания



Диаграмма состояний

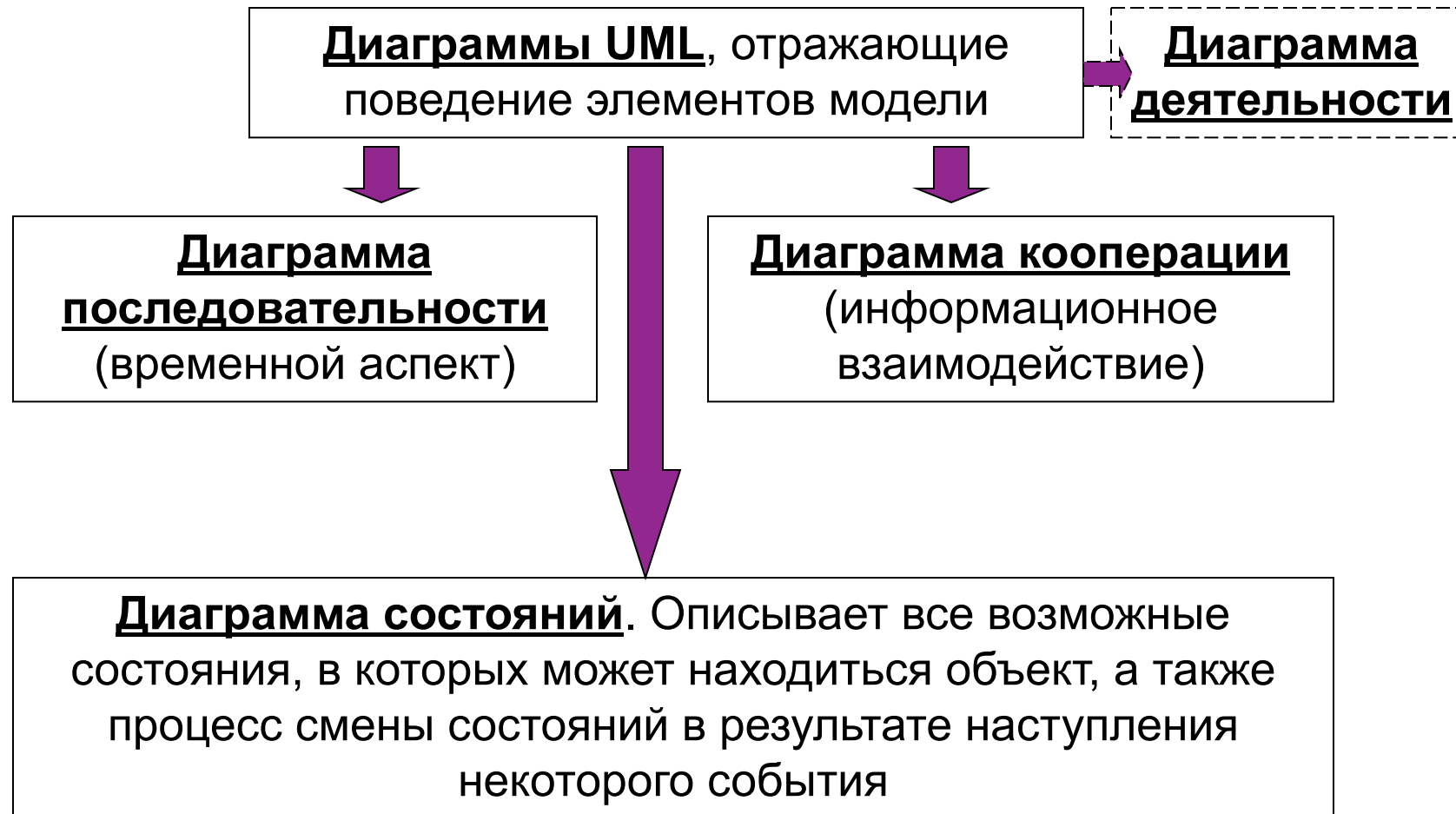


Основные вопросы

- Сущность и назначение диаграммы состояний
- Основные компоненты
- Пример



Назначение диаграммы состояний



Основные компоненты диаграммы состояний

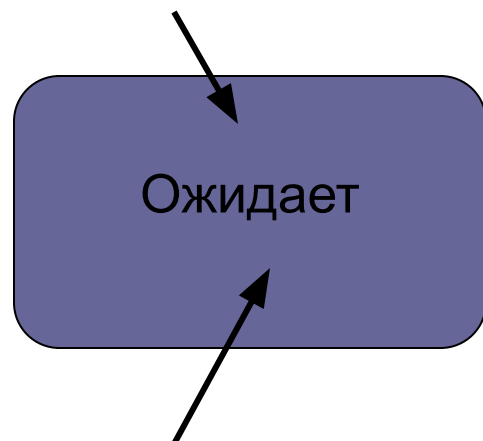
Основные компоненты диаграммы состояний:

- *состояния;*
- *переходы.*



Состояние

Имя состояния – законченное предложение, начинается с заглавной буквы



В качестве имени состояния используют глагол (звонит) или причастие (занят)

Секция имени



Список внутренних действий



Список внутренних действий

- Формат:

<метка действия '/' выражение действия>

- Возможные метки:

- **entry;**
- **exit;**
- **do.**

Ввод пароля

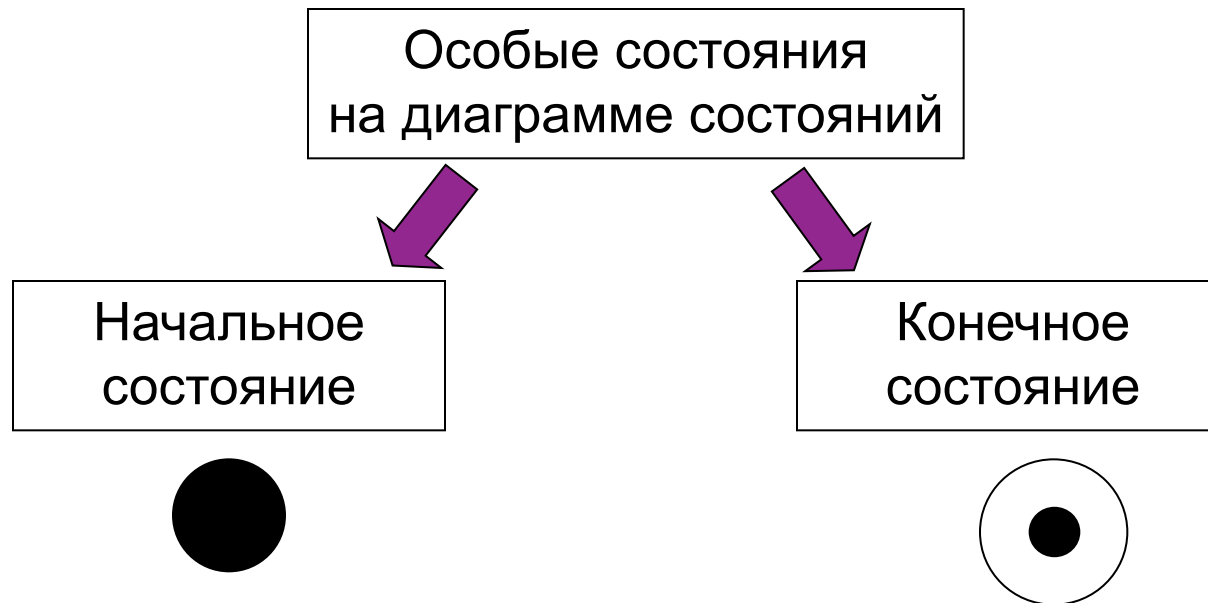
entry / сделать символы
невидимыми

символ / получить символ

exit / сделать символы
видимыми



Начальное и конечное состояние



Начальное состояние указывается обязательно и оно должно быть одно. Конечных состояний может или не быть, или может быть несколько.



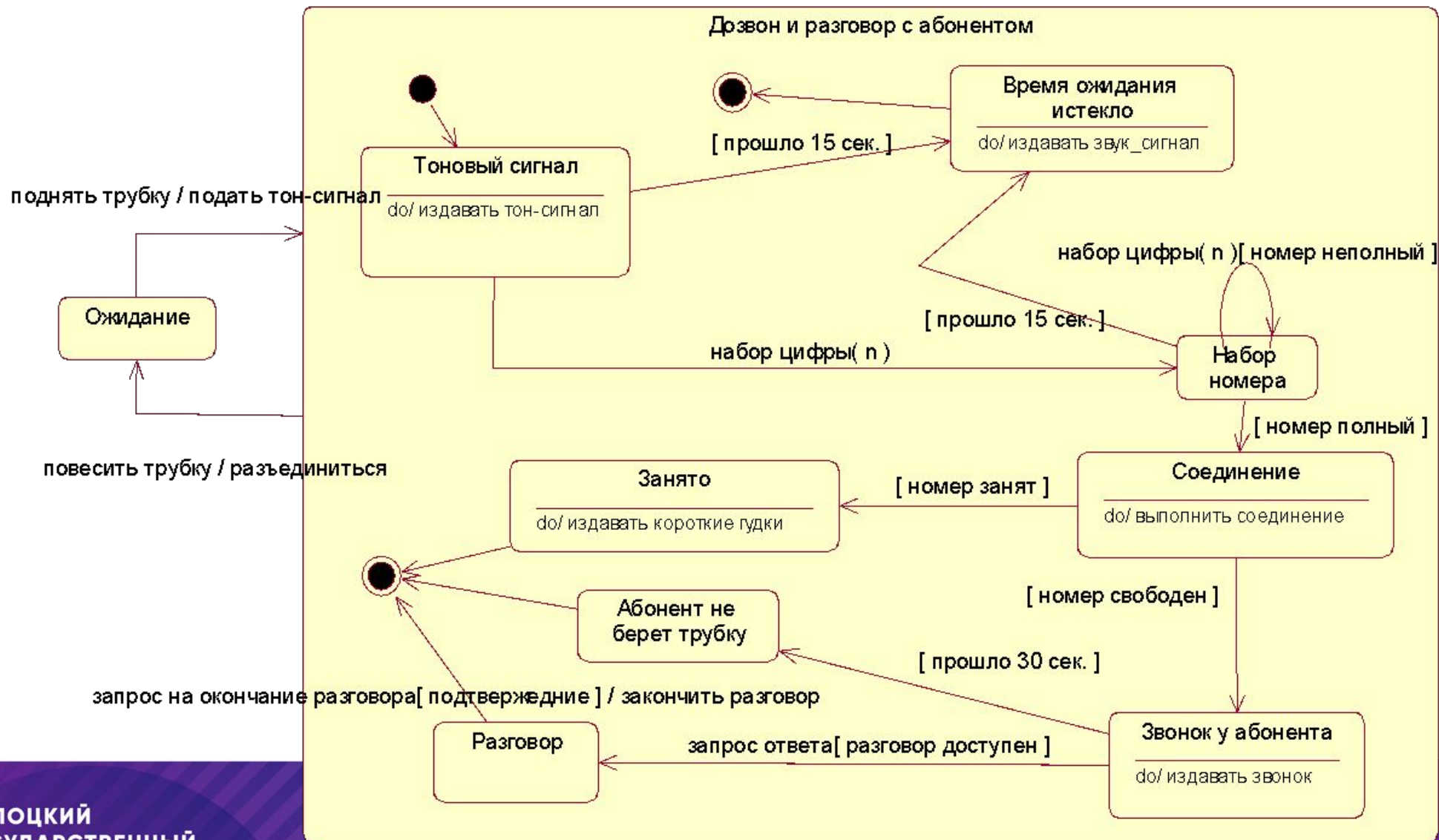
Переход

- Переход – отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим.
- Синтаксическая метка перехода состоит из трех частей, каждая из которых является необязательной:

<событие> [<условие>] / <действие>

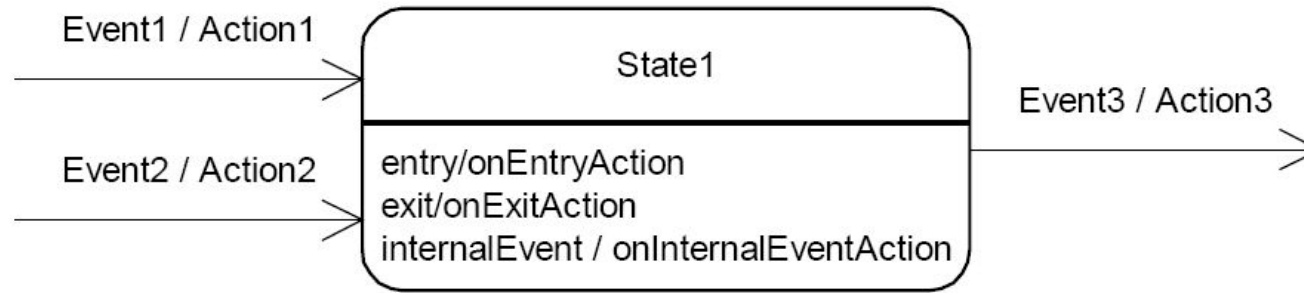


Пример диаграммы состояний



Эквивалентные переходы

Исходный фрагмент



Эквивалентный фрагмент

internalEvent / onInternalEventAction

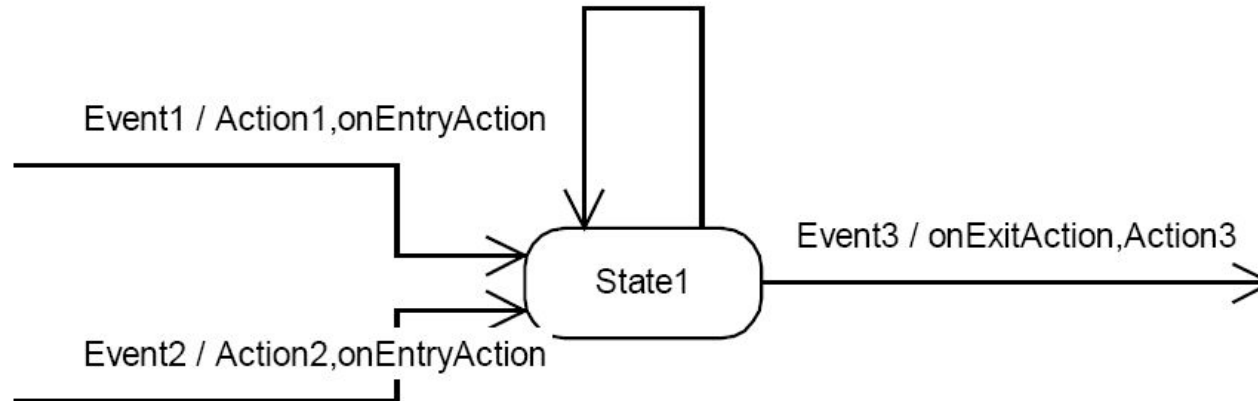


Диаграмма деятельности



Основные вопросы

- Определение и назначение диаграммы деятельности
- Понятие действия
- Основные компоненты диаграммы деятельности
- Пример



Диаграмма деятельности

- Отражает динамику системы и представляет собой схемы потоков управления в системе от действия к действию, а также параллельные действия и альтернативные потоки .
- В контексте языка UML **деятельность** представляет собой некоторую совокупность отдельных вычислений, выполняемых автоматом.



Компоненты диаграммы деятельности

Основные элементы диаграмм деятельности:

- деятельность (действие)
- переход
- элемент выбора
- линия синхронизации (линейка синхронизации).



Действие (деятельность)

- **Действие** - исполнение определенного поведения в потоке управления системой

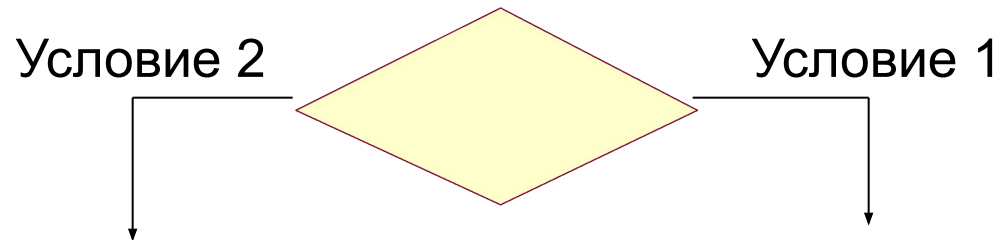
Имя может быть записано на
естественном языке

... или на языке
программирования



Элемент выбора

- Элементы выбора позволяют задавать альтернативные пути потока управления.



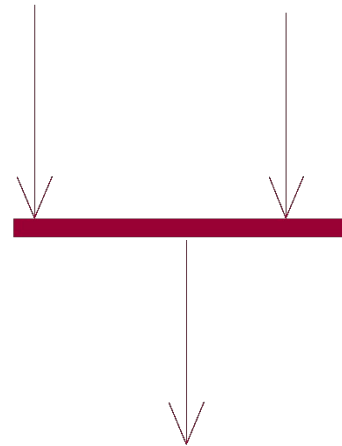
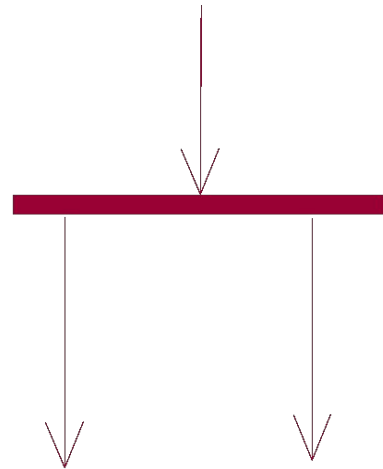
Условие – логическое выражение, которое может принимать значение true или false

Пример ветвления переходов



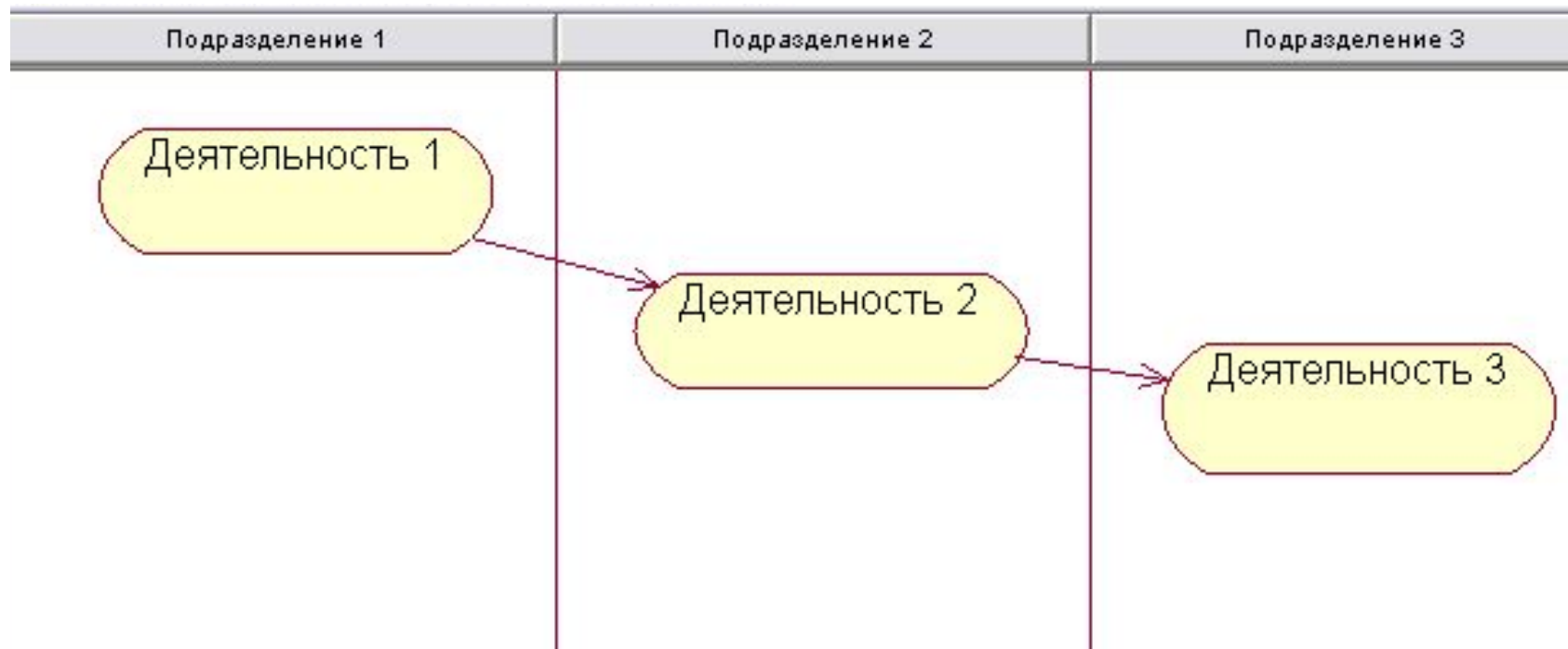
Линии синхронизации

- Линии перехода могут иметь несколько входящих линий и 1 исходящую, либо 1 вход и несколько выходов.



Дорожки (Swimlane)

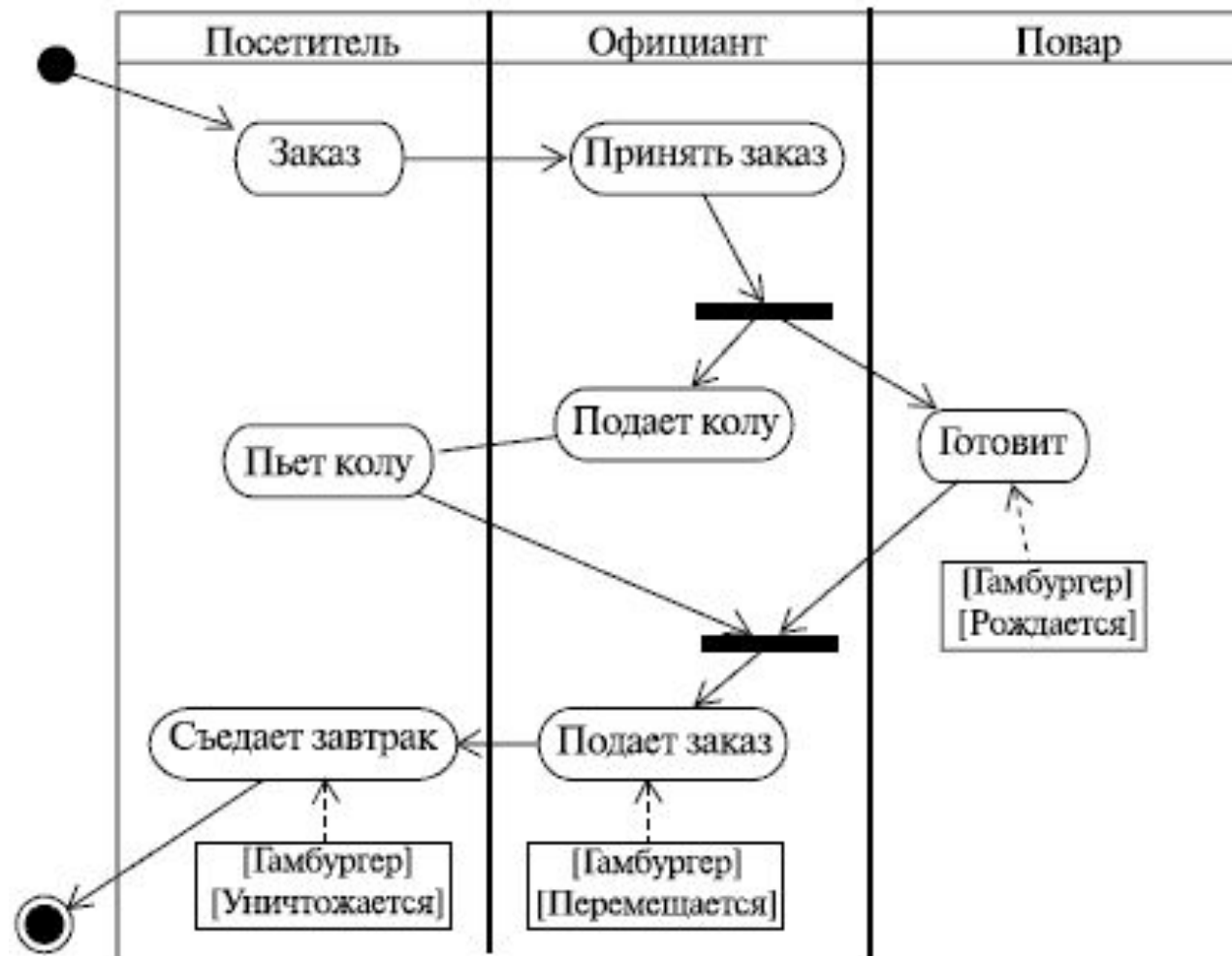
Группа действий между дорожками выполняется соответствующим подразделением



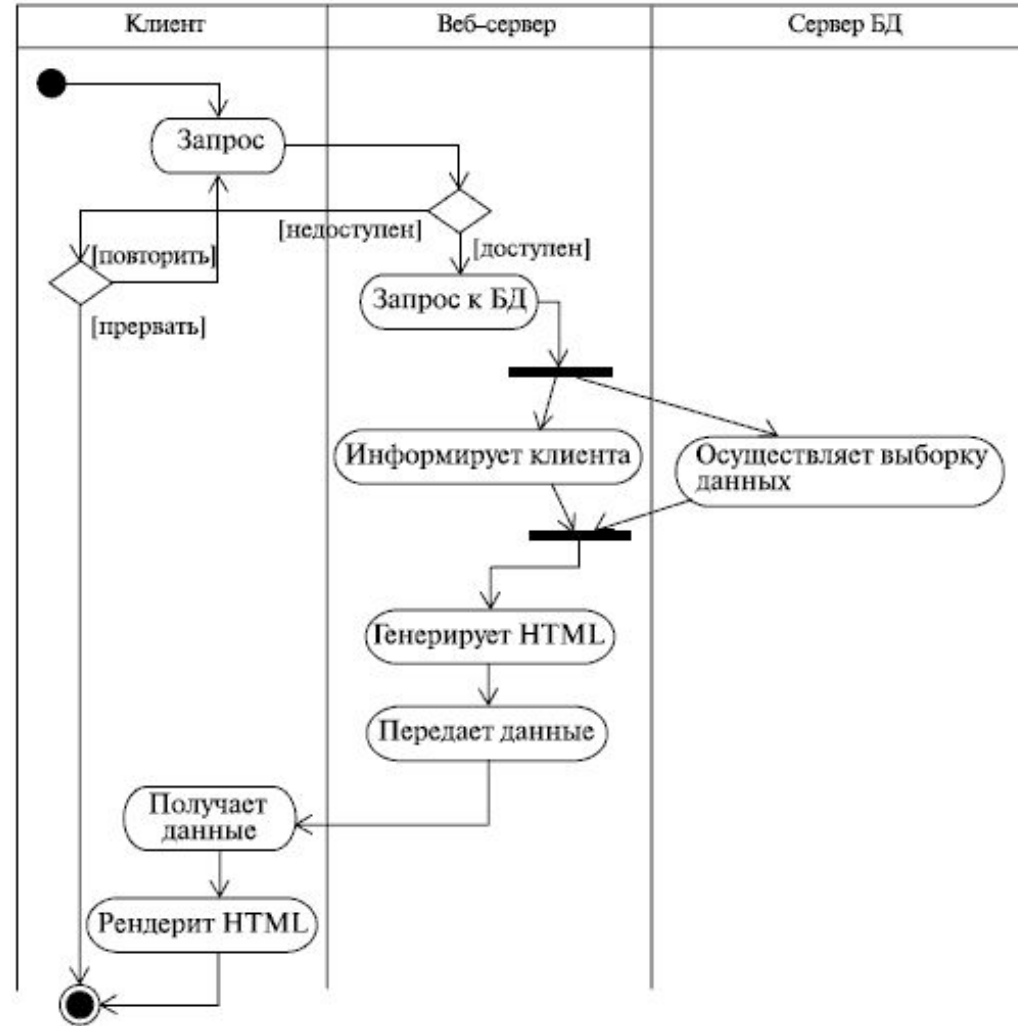
Пример диаграммы деятельности



Пример диаграммы деятельности



Пример диаграммы деятельности



Диаграммы реализации



Основные вопросы

- Виды и назначение диаграмм реализации
- Основные компоненты
- Примеры



Виды диаграмм реализации

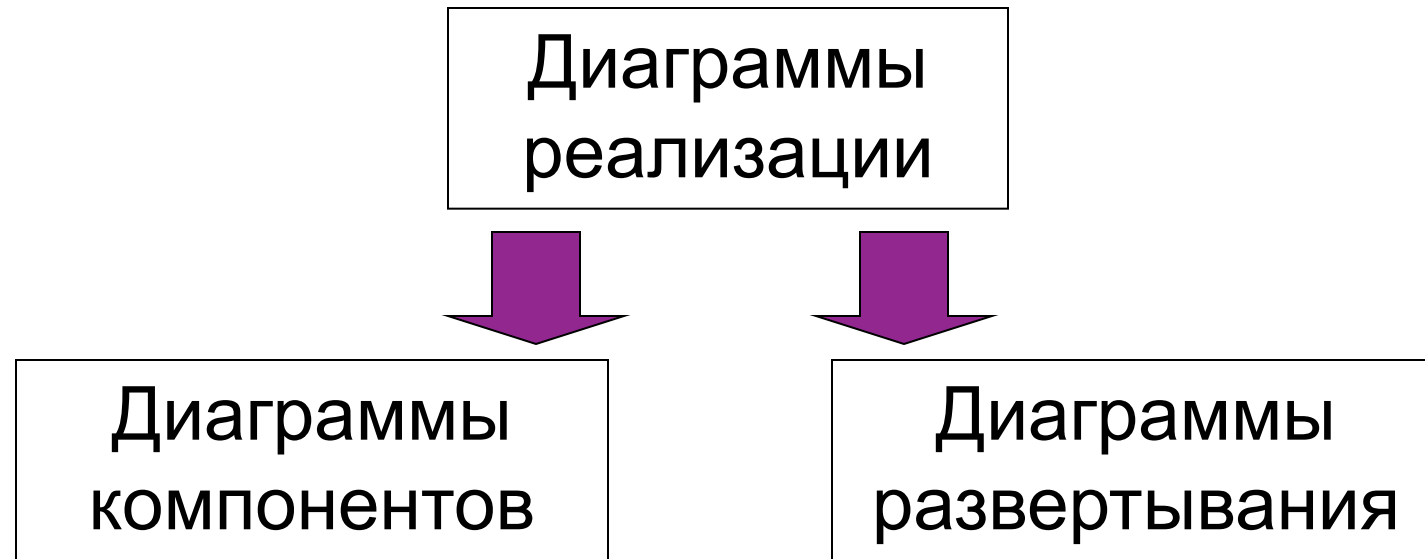


Диаграмма компонентов

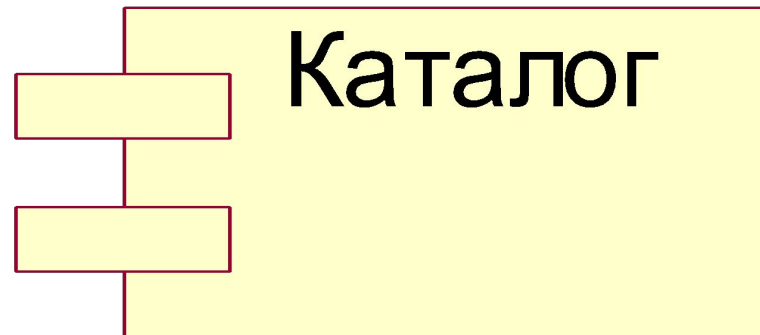
Диаграмма компонентов разрабатывается для следующих **целей**:

- визуализация общей организации **структуры исходного кода** программы;
- спецификация исполнимого варианта **программной системы**;
- представление концептуальной и физической **схем баз данных**.

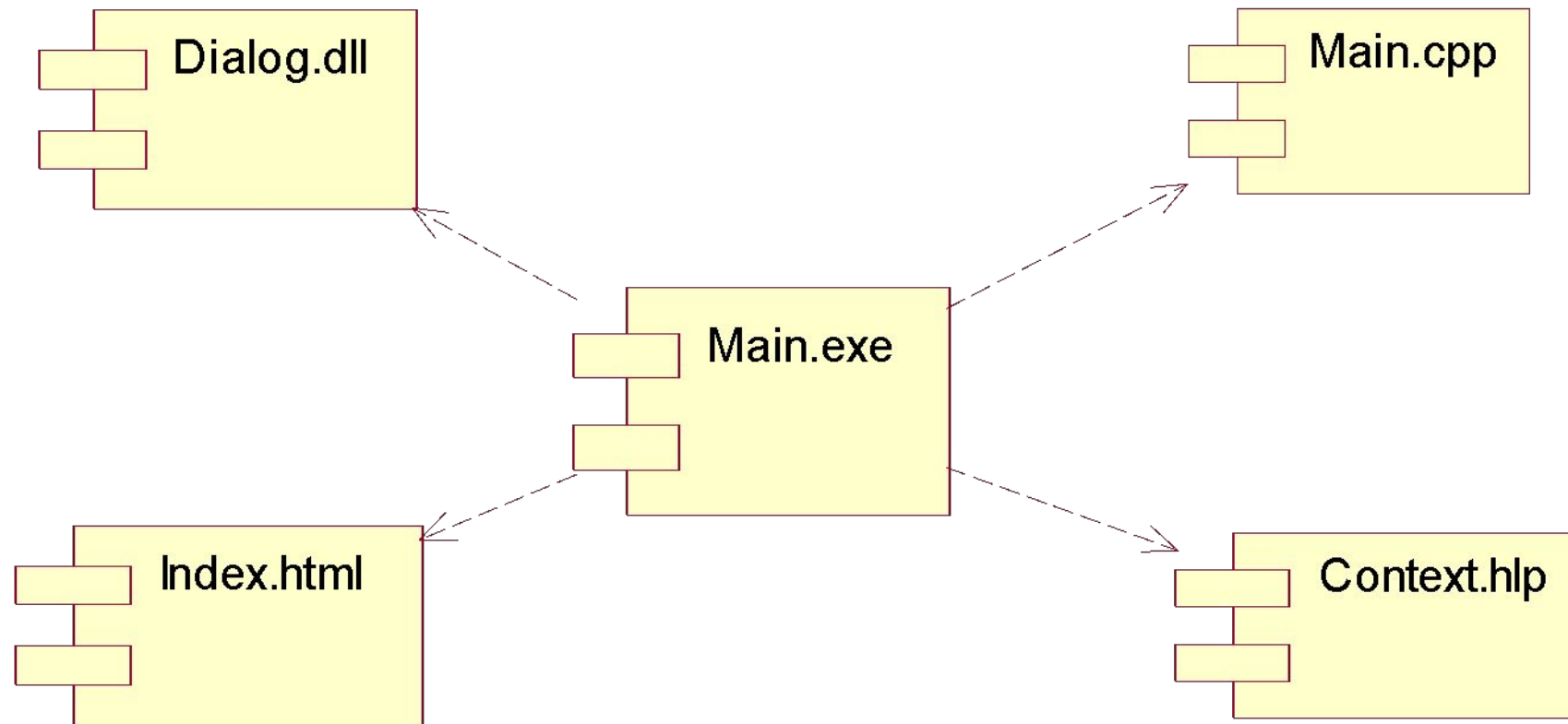


Компонент

- Служит для обозначения элементов физического представления модели и может реализовывать некий набор интерфейсов.



Пример диаграммы компонентов



Пример диаграммы компонентов

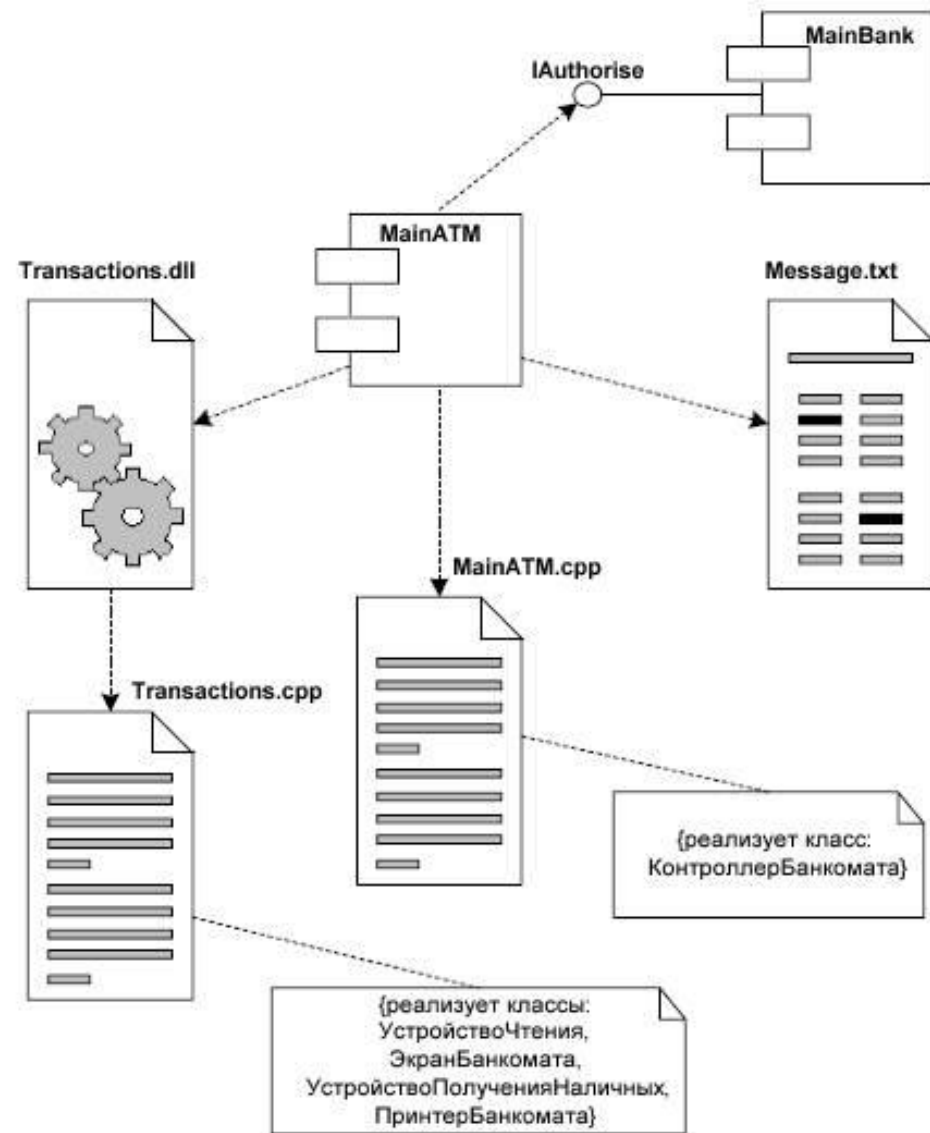


Диаграмма размещения

- *Диаграмма размещения = диаграмма развертывания = диаграмма внедрения*
- **Цели** построения диаграммы развертывания:
 - указать размещение исполнимых компонентов программной системы по отдельным физическим узлам;
 - показать физические связи между всеми узлами реализации системы на этапе ее исполнения;
 - выявить узкие места системы и реконфигурировать ее топологию для достижения наилучшей производительности.

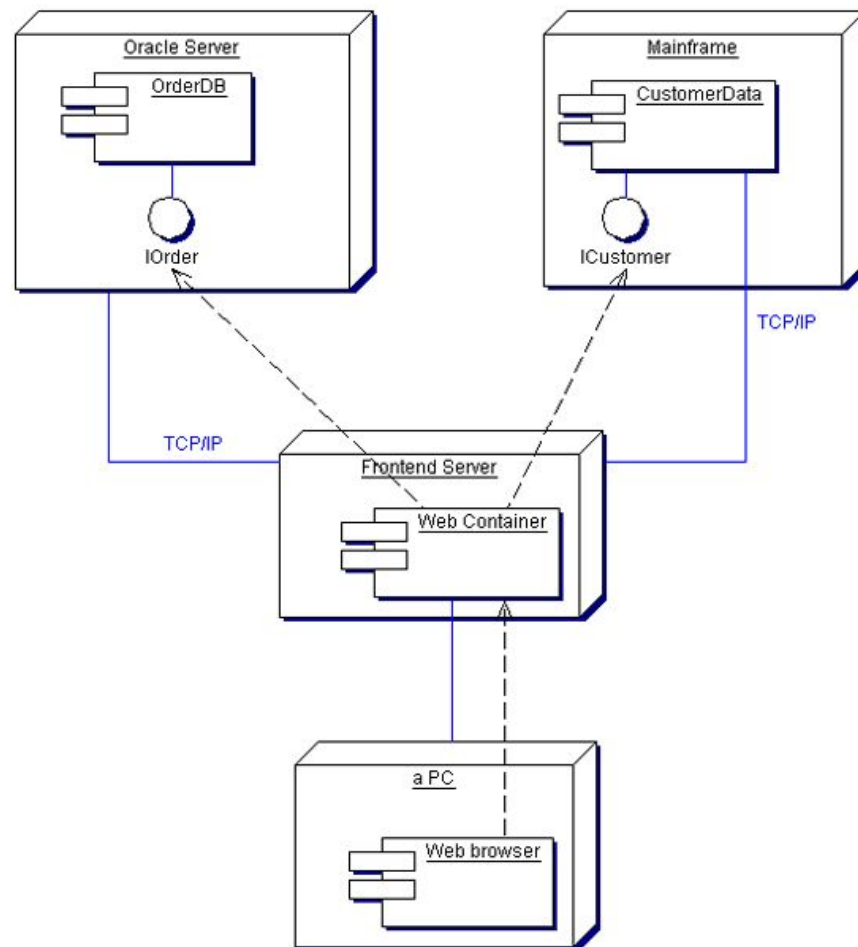


Диаграмма размещения

- **Узел** представляет собой некоторый физически существующий элемент системы, обладающий некоторым вычислительным ресурсом.
- Диаграмма размещения показывает наличие физических соединений – маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.



Пример диаграммы развертывания



Пример диаграммы развертывания

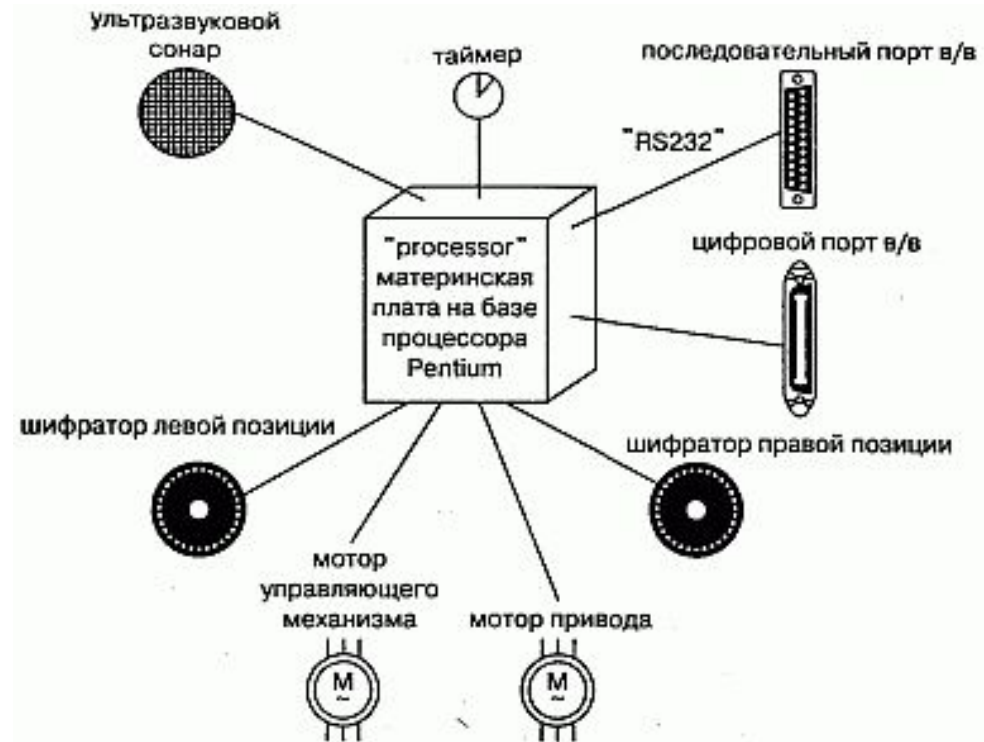


Рис. 30.2. Моделирование встроенной системы