UML. Диаграмма классов

для внутреннего пользования

Диаграмма классов

При проектировании системы информационная составляющая предметной области представляется в виде логической модели уровня сущностей (логическая модель)

Логическая модель описывает сущности (объекты данных) автоматизируемой предметной области и связи между ними

В унифицированном языке моделирования (UML) для представления логической модели можно использовать диаграмму классов

Диаграмма классов (class diagram) - это диаграмма, на которой показано множество классов, интерфейсов, кооперации и отношений между ними. Ее изображают в виде множества вершин и дуг

Назначение и особенности

Назначение диаграммы классов

- Диаграмма классов служит для представления статической структуры модели системы
- Диаграмма классов может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений

Особенности диаграммы классов

- На диаграмме классов не указывается информация о временных аспектах функционирования системы

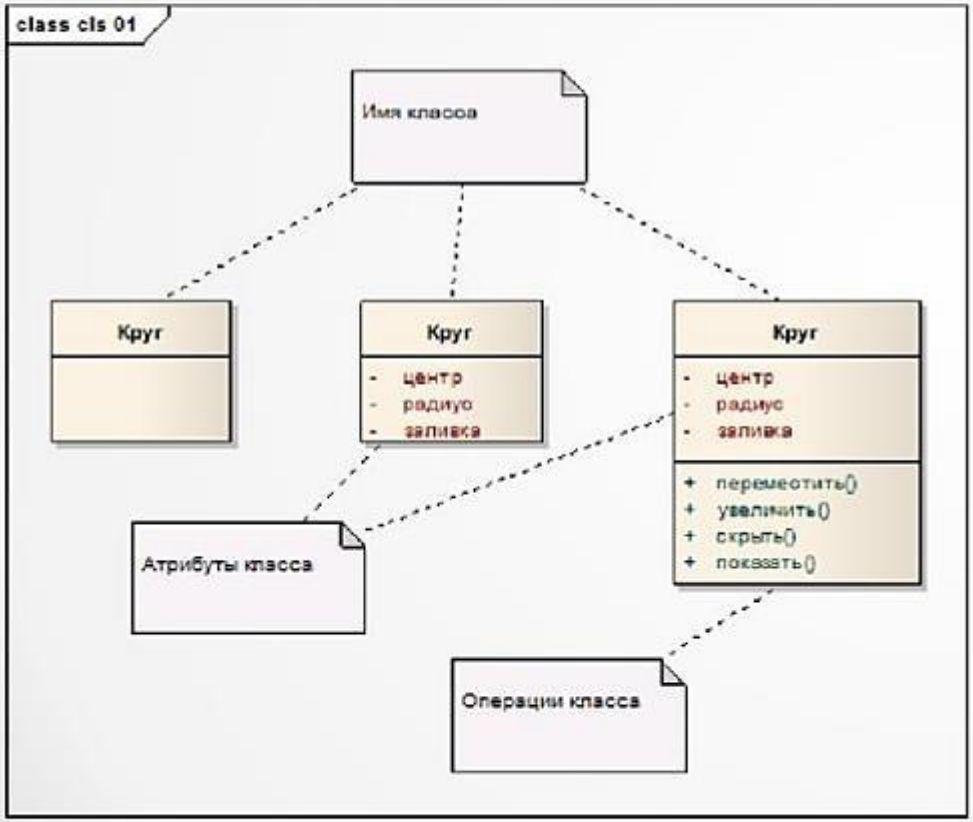
Диаграмма классов

Диаграммы классов могут содержать:

- Классы
- Отношения
- Интерфейсы
- Кооперации

Элементы диаграммы классов

Классы



Класс (class) служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции. В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

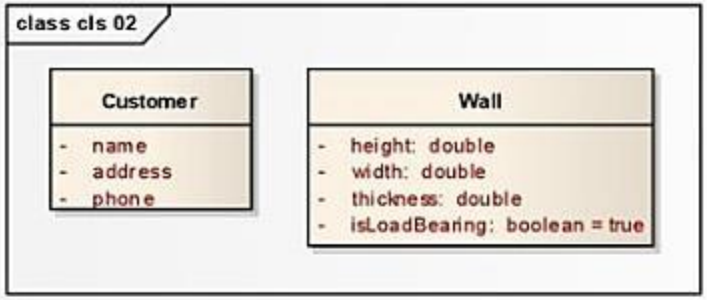
Имя класса

- **Имя класса** должно быть уникальным в пределах проекта, который описывается некоторой совокупностью диаграмм классов (возможно, одной диаграммой). Имя класса указывается в верхней секции прямоугольника полужирным шрифтом и должно начинаться с прописной буквы.

- **Абстрактный класс** – это класс, который не может иметь экземпляров (объектов). Для обозначения его имени используется наклонный шрифт.

Атрибуты класса

Атрибут - это именованное свойство класса, общее для всех его объектов, включающее описание множества значений, которые могут принимать экземпляры этого свойства

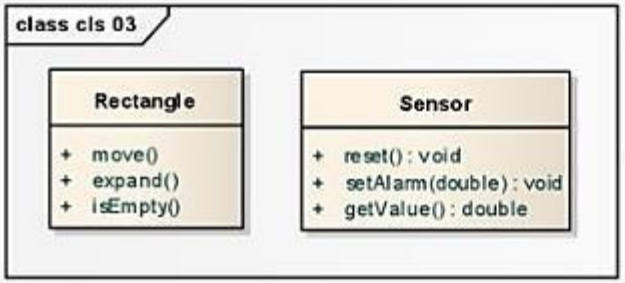


- *Класс может иметь любое число атрибутов или не иметь их вовсе*
- *Атрибут является абстракцией данных объекта или его состояния*
- *В каждый момент времени любой атрибут объекта, принадлежащего данному классу, обладает вполне определенным значением*
- *Атрибуты представлены в разделе, который расположен под именем класса*

При описании атрибута можно явным образом указывать его тип и значение, принимаемое по умолчанию

Операции класса

Операция – это реализация услуги, которую можно запросить у любого объекта класса для воздействия на поведение. Операция - это абстракция того, что может делать объект.



- *У всех объектов класса имеется общий набор операций*
- *Класс может содержать любое число операций или не содержать их вовсе*

- *Как правило (хотя не всегда) обращение к операции объекта изменяет его состояние или его данные*
- *Операции класса изображаются в разделе, расположенном ниже раздела с атрибутами. При этом можно ограничиться только именами.*

Советы по классам

Изображая класс в UML, придерживайтесь следующих правил:

- показывайте только те свойства класса, которые важны для понимания абстракции в данном контексте
- разделяйте длинные списки атрибутов и операций на группы в соответствии с их категориями
- показывайте взаимосвязанные классы на одной и той же диаграмме

Отношения между классами

Помимо внутреннего устройства классов на диаграмме классов указываются различные отношения (связи) между ними. Совокупность типов отношений фиксирована.

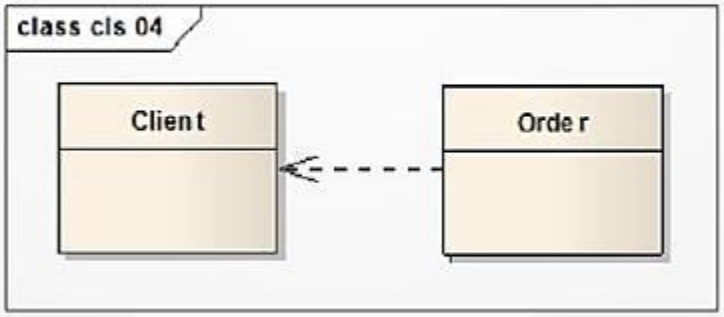
Базовые отношениями в языке UML:

- Отношение зависимости (dependency relationship)
- *Отношение ассоциации (association relationship)*
- Отношение обобщения (generalization relationship)

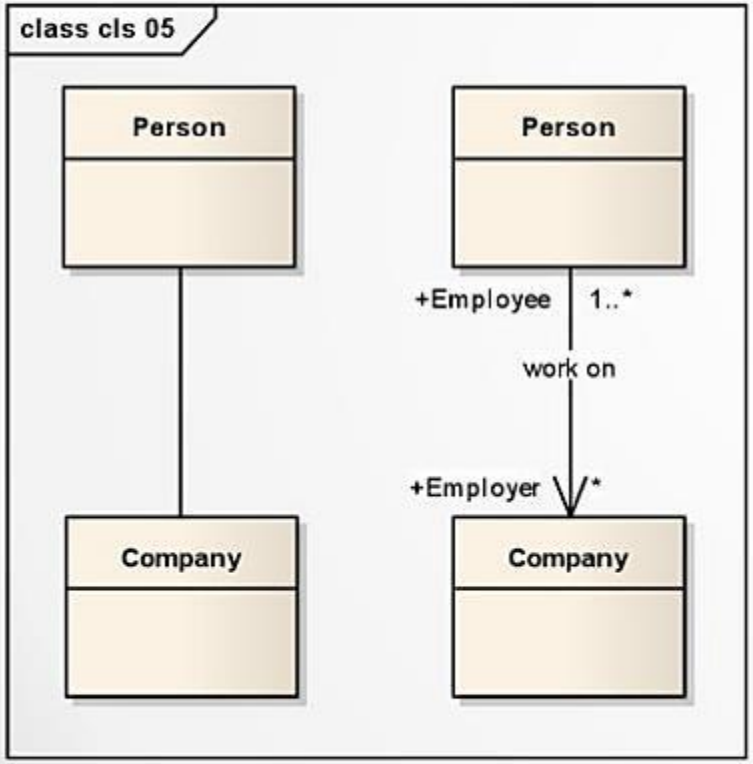
Отношение зависимости

Зависимость (dependency) - отношение использования, согласно которому изменение в спецификации одного элемента может повлиять на другой элемент, его использующий, причем обратное не обязательно.

Графически зависимость изображается пунктирной линией со стрелкой, направленной от зависимого элемента на тот, от которого он зависит.



Отношение ассоциации

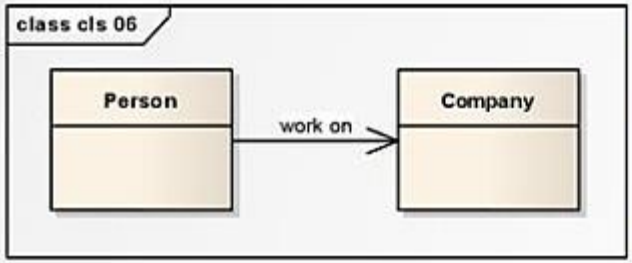


Ассоциация (association) - структурное отношение, показывающее, что объекты одного типа неким образом связаны с объектами другого типа.

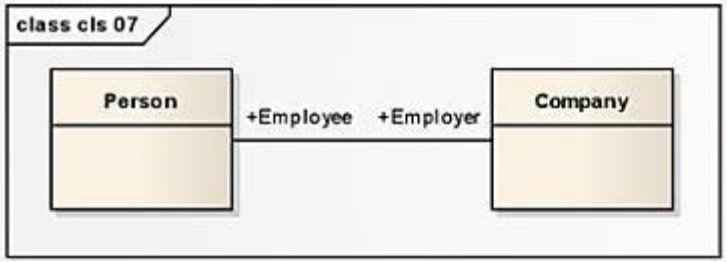
Ассоциация, связывающая два класса, называется бинарной. Можно создавать ассоциации, связывающие сразу несколько классов; они называются n-арными.

У ассоциаций могут быть: имя, роль, кратность и агрегирование.

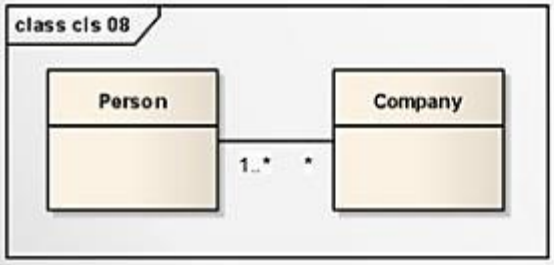
Отношение ассоциации



Имя описывает природу отношения. Чтобы избежать возможных двусмысленностей в понимании имени, указывается направление, в котором оно должно читаться.



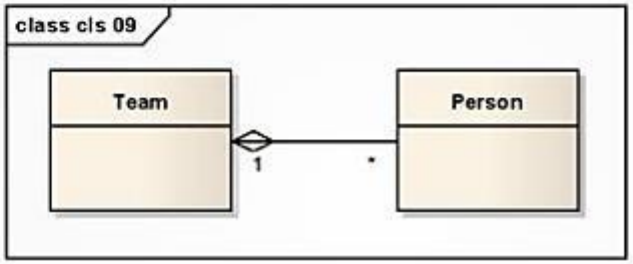
Роль – это «лицо», которым класс, находящийся на одной стороне ассоциации, обращен к классу с другой ее стороны. Один класс может играть в разных ассоциациях как одну и ту же роль, так и различные.



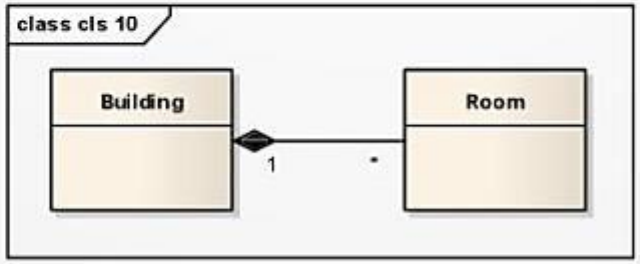
Кратность - количество объектов, которое может быть связано посредством одного экземпляра ассоциации. Кратность записывается либо как выражение, значением которого является диапазон значений, либо в явном виде. Кратность можно задать равной единице (1), можно указать диапазон: «ноль или единица» (0..1), «много» (0..*), «единица или больше» (1..*), «любое число объектов, кроме 2 и 5» (0 . . 1, 3..4, 6..*).

Отношение ассоциации

Агрегирование (aggregation) показывает отношение типа «часть/целое», в котором один класс состоит из нескольких меньших классов



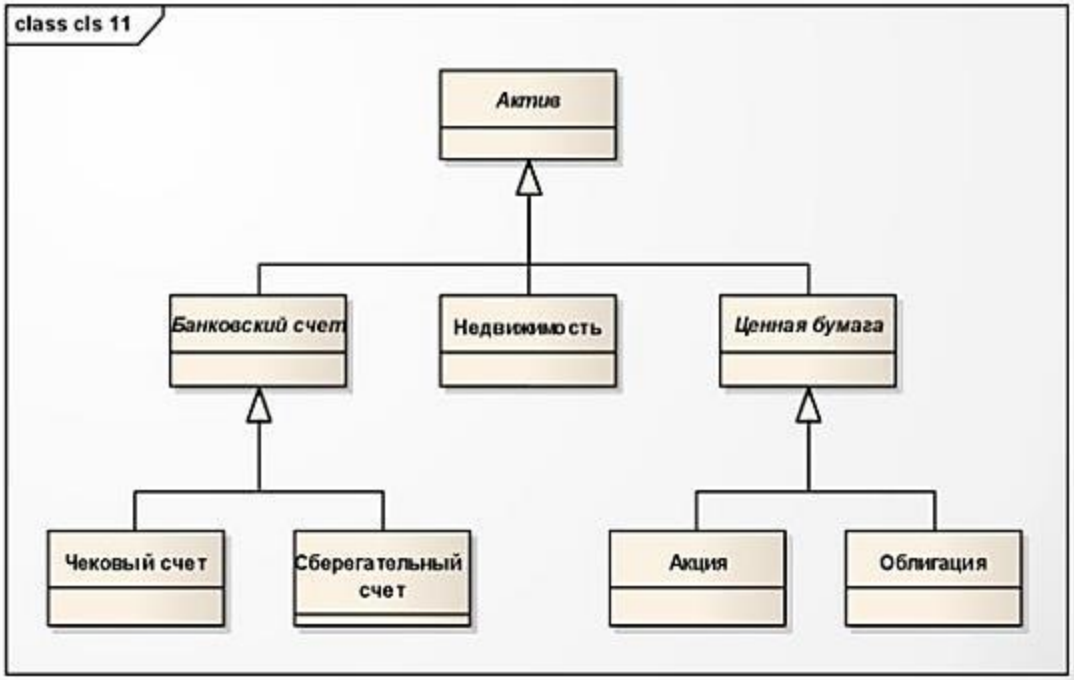
Композиция (composition) служит для выделения специальной формы отношения «часть/целое». Специфика связи заключается в том, что части не могут выступать в отрыве от целого, то есть с уничтожением целого уничтожаются и все его составные части.



Отношение обобщения

Обобщение (generalization) - это отношение между общей сущностью (суперклассом, или родителем) и ее конкретным воплощением (подклассом, или потомком).

Данное отношение описывает иерархию классов и наследование их свойств и поведения. Предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка.



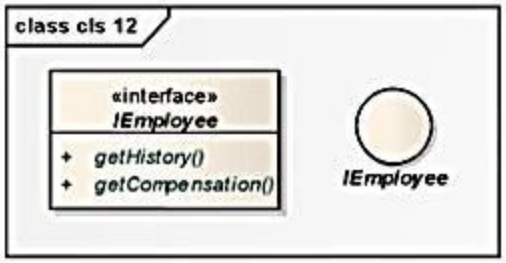
Советы по отношениям между классами

При моделировании отношений в UML соблюдайте следующие правила:

- используйте зависимость, только если моделируемое отношение не является структурным
- используйте обобщение, только если имеет место отношение типа «является»
- избегайте множественного наследования
- иерархия наследования не должна быть ни слишком глубокой (желательно не более пяти уровней), ни слишком широкой
- применяйте ассоциации прежде всего там, где между объектами существуют структурные отношения

Интерфейсы

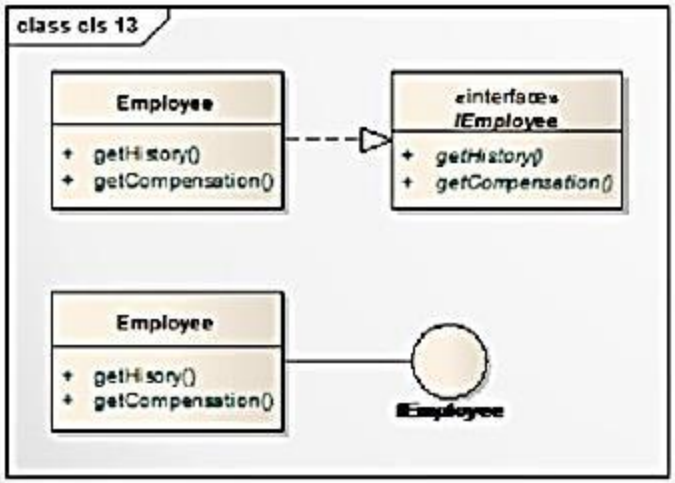
Интерфейс (interface) - набор операций, используемый для специфицирования услуг, предоставляемых классом или компонентом.



Интерфейс должен иметь уникальное имя.
Интерфейс может включать любое число операций.

Особенности:

- Интерфейсы не содержат атрибутов
- Интерфейсы не содержат реализующих операции методов

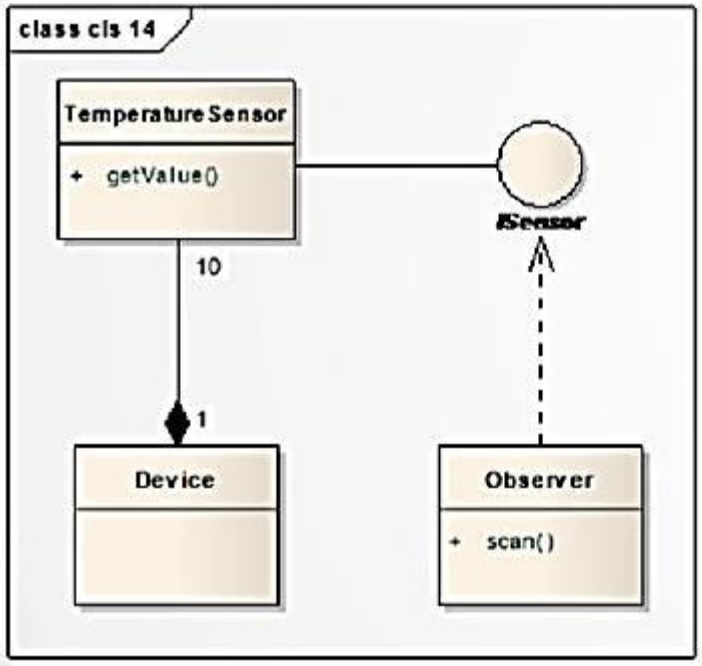


Интерфейс специфицирует контракт класса, но не накладывает никаких ограничений на свою реализацию.

Связь интерфейса с реализующим его элементом можно графически представить двумя способами:

- интерфейс представляют в виде стереотипного класса и связывают его с классом **отношением реализации**. Отношение реализации объединяет отношения обобщения и зависимости
- отношение между интерфейсом и его реализацией изображается кружочком с одной стороны класса.

Интерфейсы



Интерфейсы могут принимать участие в отношениях обобщения, ассоциации и зависимости.

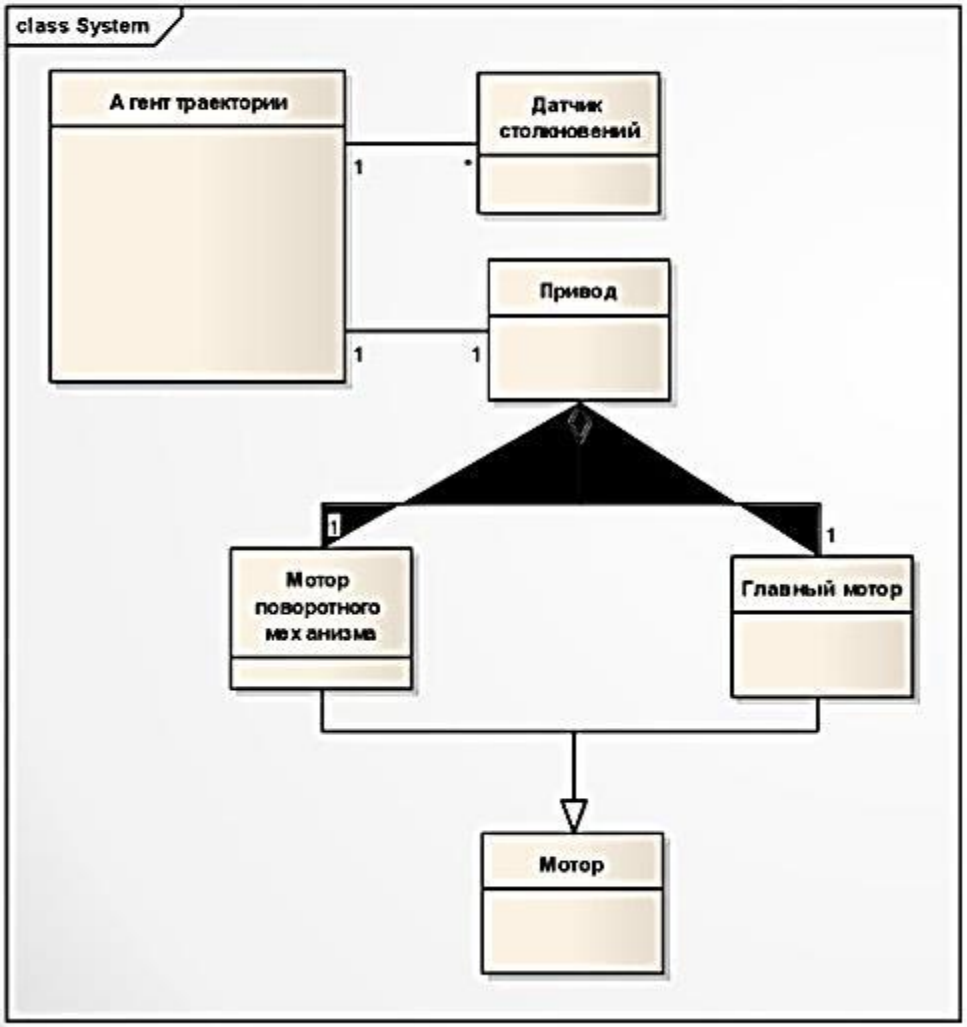
Интерфейс представляет собой стыковочный узел в системе. Он определяет условия контракта, после чего обе стороны - клиент и поставщик - могут действовать независимо друг от друга, полностью полагаясь на взаимные обязательства.

Советы по интерфейсам

Изображая интерфейс на языке UML, руководствуйтесь приведенными ниже правилами:

- используйте сокращенную нотацию, если надо просто показать наличие стыковочного узла в системе
- используйте форму, если надо визуализировать детали самого сервиса

Кооперации



Кооперация (collaboration) - это сообщество классов, интерфейсов и других элементов, которые работают совместно для обеспечения кооперативного поведения, более значимого, чем сумма его составляющих.

Пример

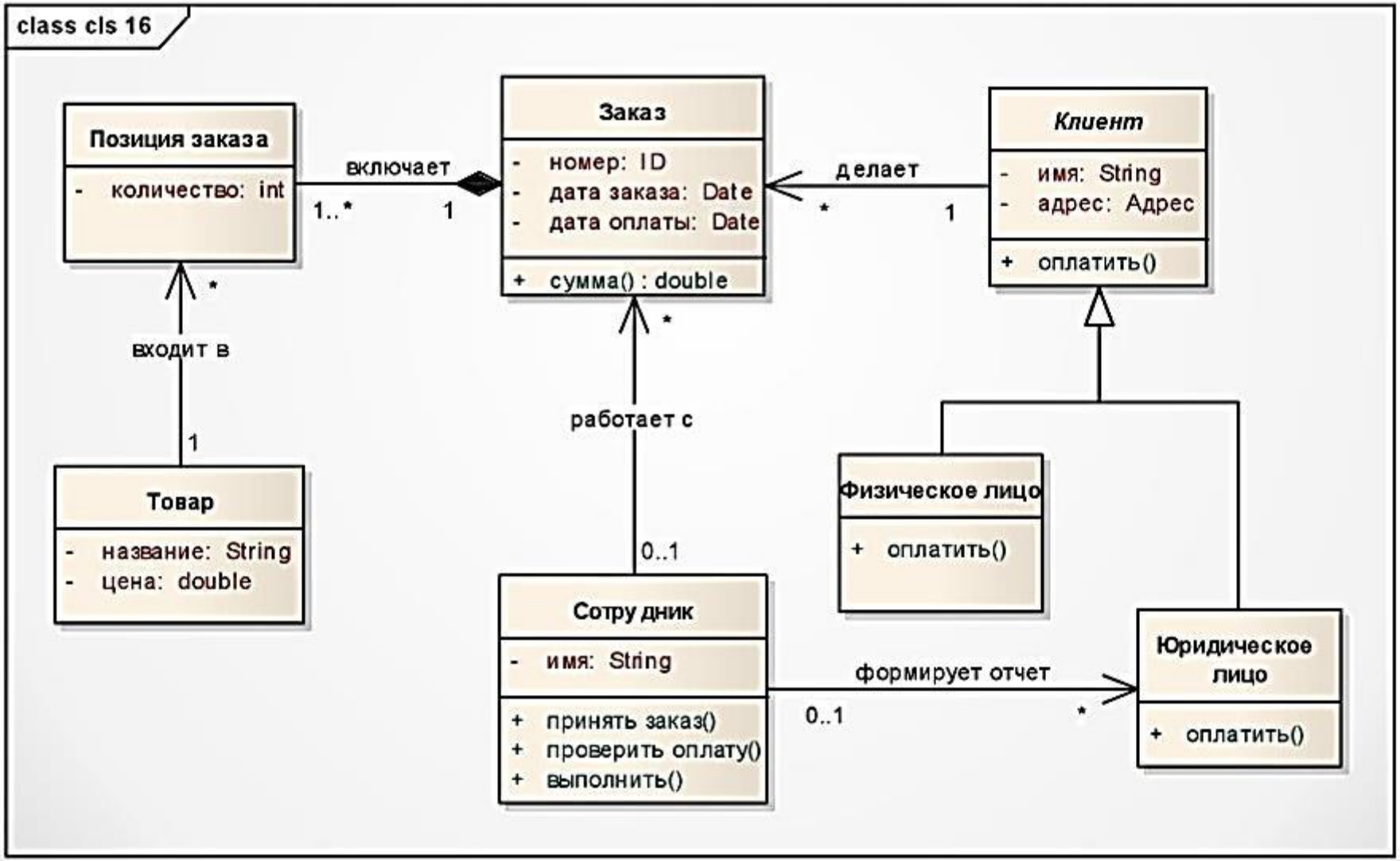

class cls 16
Позиция заказа
- количество: int
включает
1..*
1
Заказ
- номер: ID
- дата заказа: Date
- дата оплаты: Date
+ сумма() : double
делает
*
1
Клиент
- имя: String
- адрес: Адрес
+ оплатить()
*
входит в
1
Товар
- название: String
- цена: double
*
работает с
0..1
Сотрудник
- имя: String
+ принять заказ()
+ проверить оплату()
+ выполнить()
Физическое лицо
+ оплатить()
формирует отчет
0..1
*
Юридическое лицо
+ оплатить()

Диаграмма состояний

Основные вопросы

- Сущность и назначение диаграммы состояний
- Основные компоненты
- Примеры

Назначение диаграммы состояний

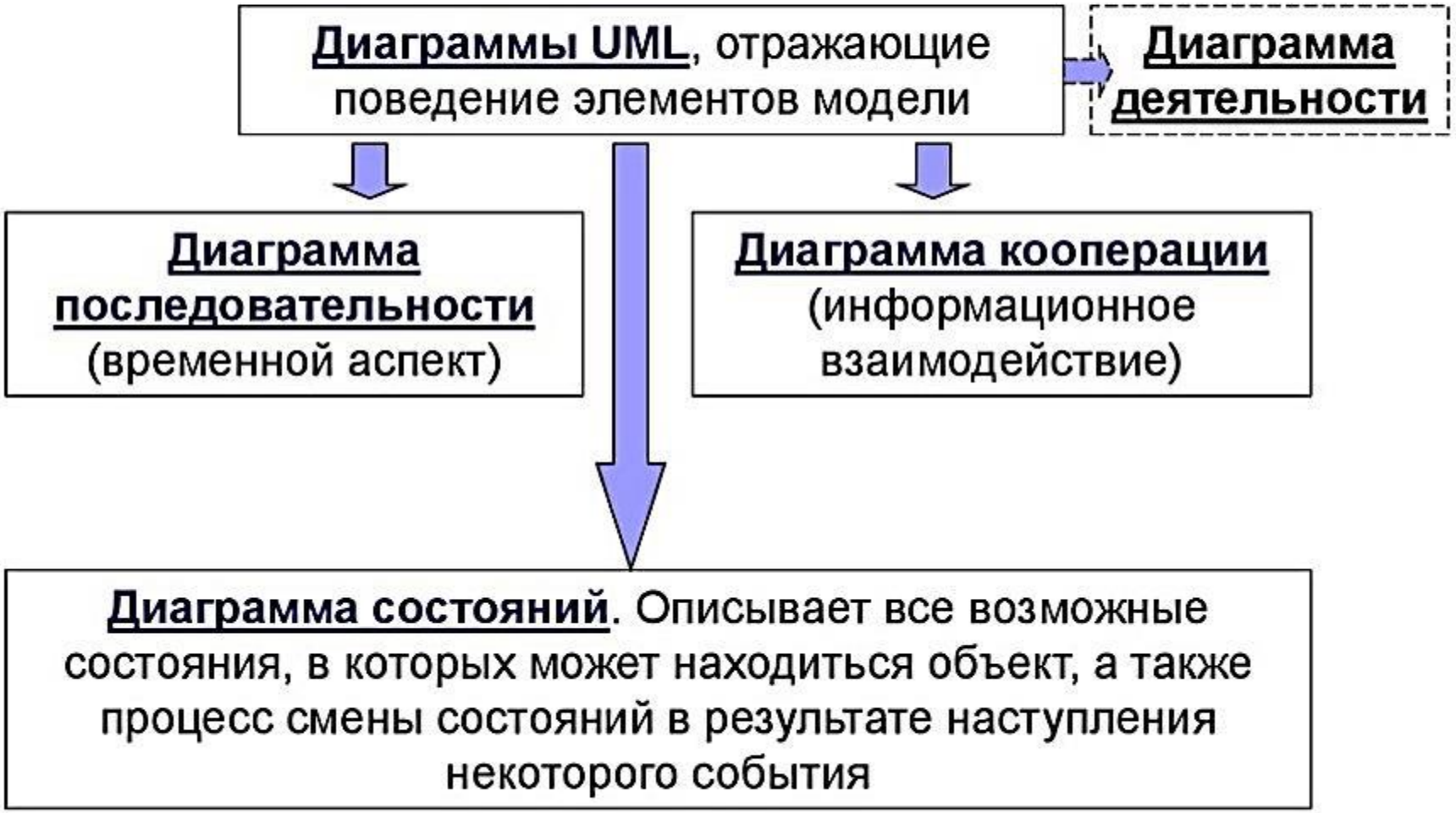


Диаграмма состояний

- Диаграмма состояний (statechart diagram) используется для описания поведения объектов (отдельных экземпляров класса)
- Диаграмма состояний является графом специального вида, который представляет некоторый автомат. Вершинами этого графа являются состояния. Дуги графа служат для обозначения переходов из состояния в состояние.
- Переход объекта из состояния в состояние происходит в результате наступления некоторого события
- Смена состояний происходит мгновенно
- Диаграммы состояний могут быть вложены друг в друга. Состояние на диаграмме может быть описано с помощью другой диаграммы состояний.

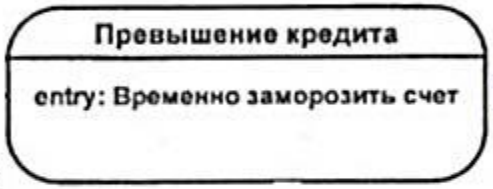
Элементы диаграммы состояний

- Состояние

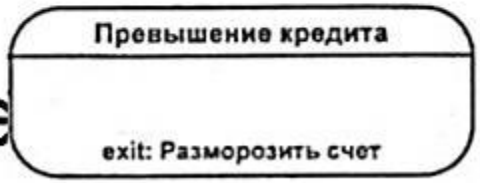
- Начальное состояние

- Конечное состояние

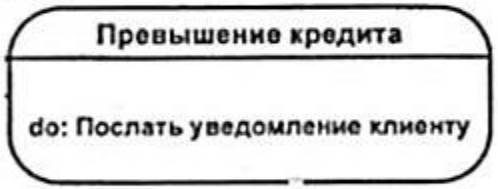
- Входное действие



- Выходное действие

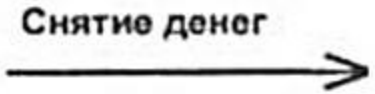


- Внутренняя деятель-ность

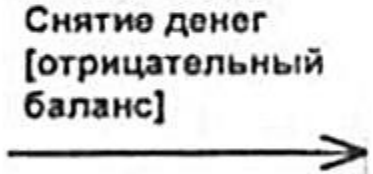


- Переход

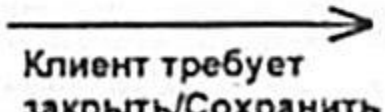
- Событие



- Ограничивающие условия



- Действие



Основные компоненты диаграммы состояний

Основные компоненты диаграммы состояний:

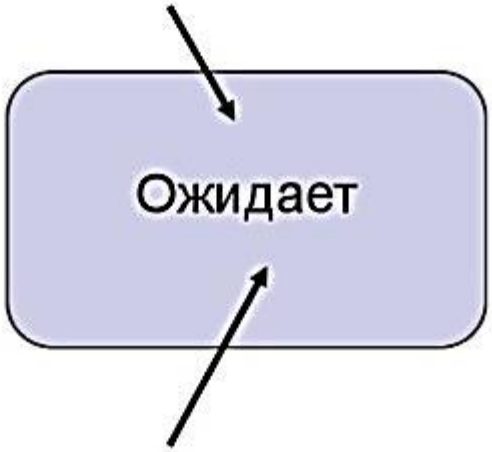
- *состояния;*
- *переходы.*

Состояние

- Состояние (state) объекта определяется набором значений его атрибутов и связей с другими объектами
- Состояние – это некоторый период времени жизни объекта
- Объекты класса имеют конечное число возможных состояний
- Находясь в определенном состоянии объект воспринимает только определенные события, а другие игнорирует
- Находясь в некотором состоянии объект может совершать некоторую деятельность

Состояние

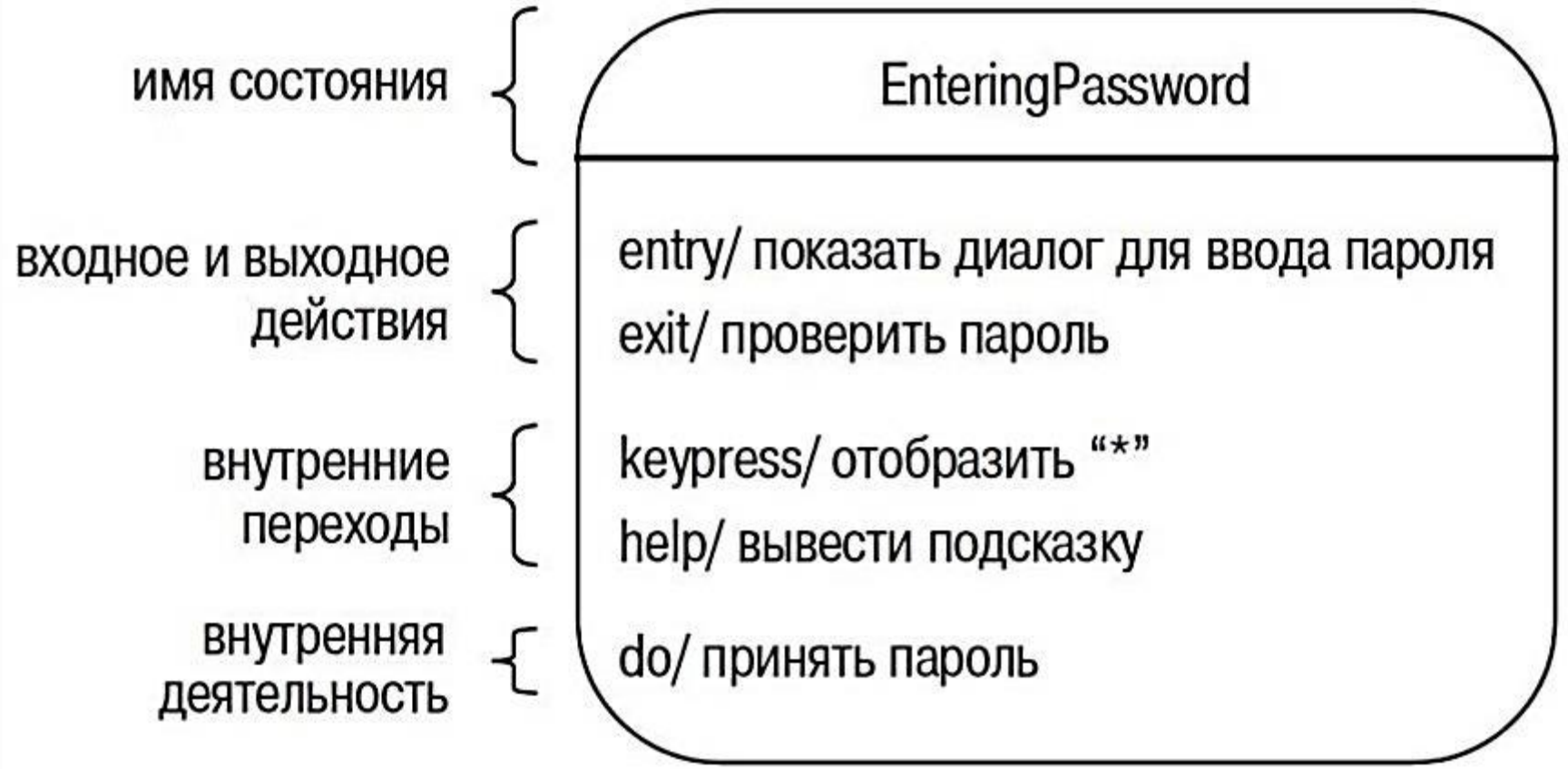
Имя состояния – законченное предложение, начинается с заглавной буквы



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

Секция имени

Список внутренних действий



синтаксис действия: имяСобытия/некотороеДействие

синтаксис деятельности: do/некотораяДеятельность

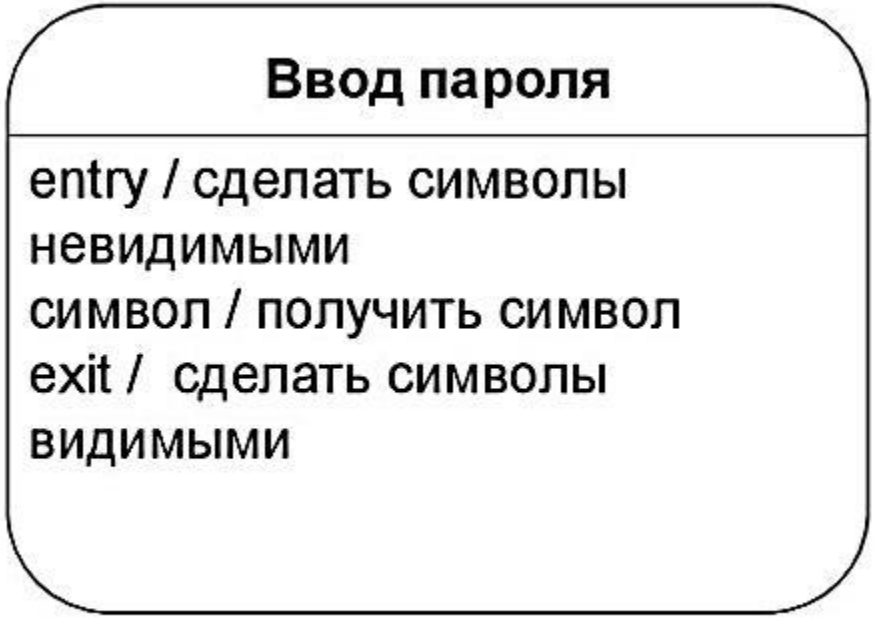
Примеры состояний

Список внутренних действий

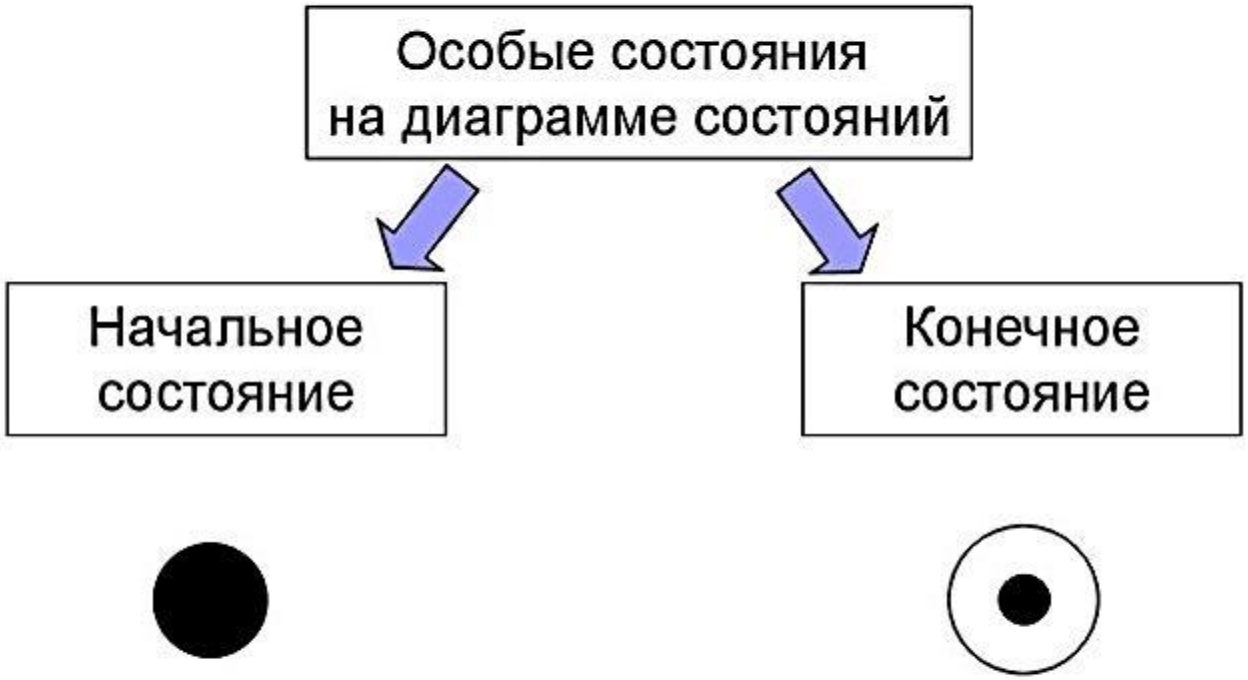
- Формат:

<метка действия ‘/’ выражение действия>

- Перечень меток в языке UML фиксирован:
  - **entry**;
  - **exit**;
  - **do**;
  - **include**.



Начальное и конечное состояние



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

Особые состояния

- Начальное состояние (initial state) – псевдосостояние, с которого начинается диаграмма состояний. Оно соответствует моменту создания объекта.
- Реально объект никогда не находится в начальном состоянии, а сразу переходит в следующее состояние
- Конечное состояние (final state) означает уничтожение объекта
- На диаграмме может быть несколько конечных состояний

Начальное состояние

Конечное состояние

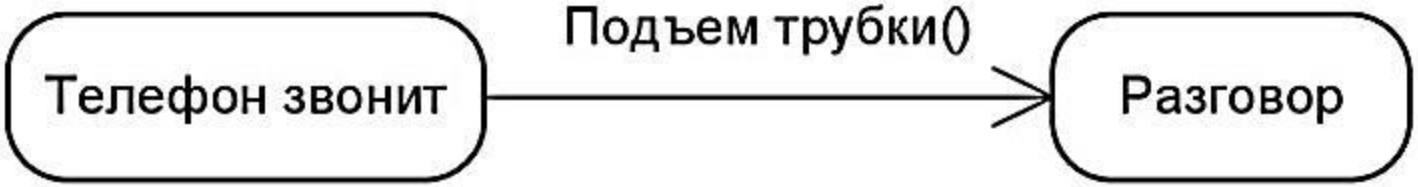
Переход

- 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:

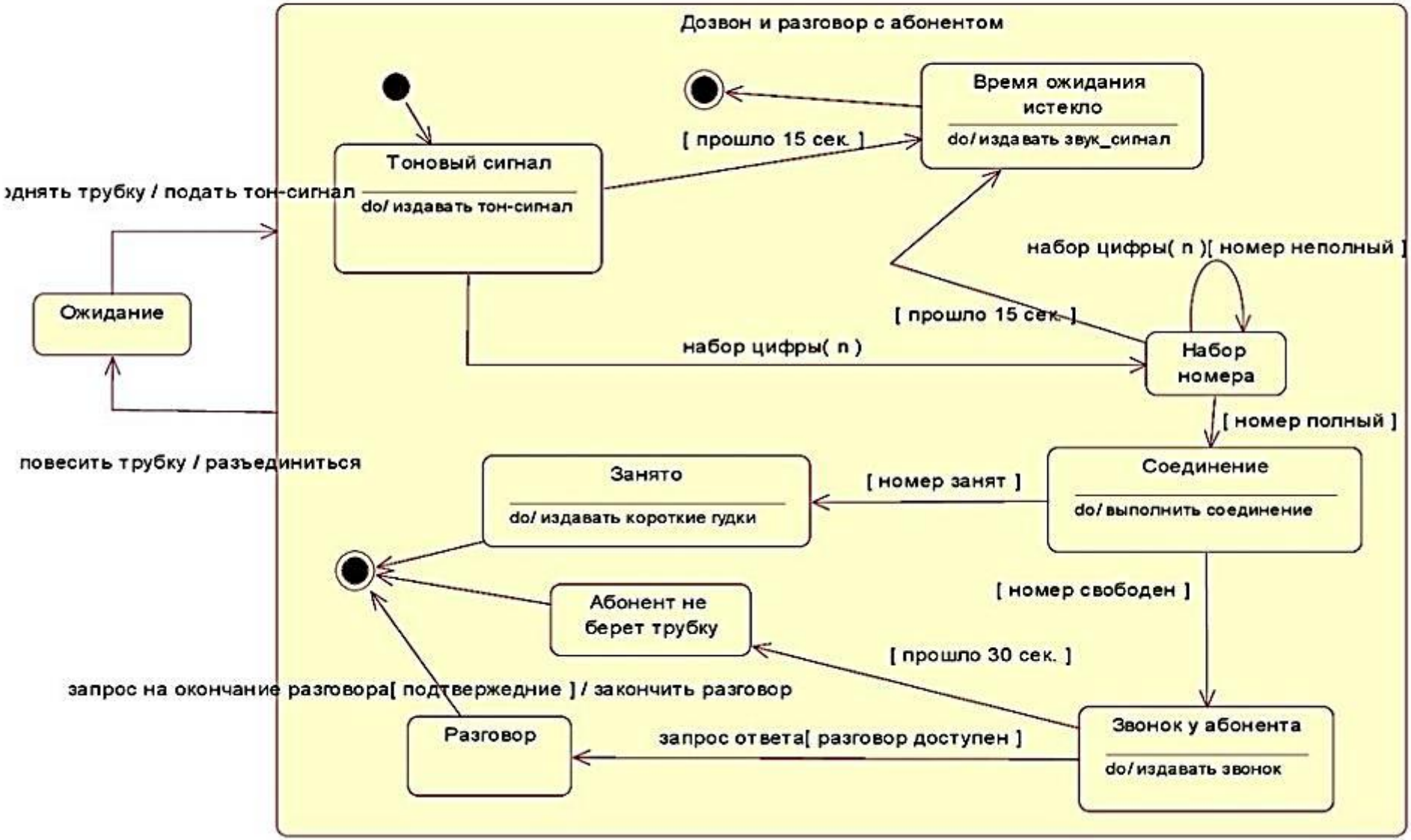
<событие> [<условие>] / <действие>

Переходы

- Переход (transition) – это мгновенная смена одного состояния (исходного) на другое состояние (целевое)
- Переход запускается, когда происходит связанное с ним событие
- Переход может запускаться автоматически, когда заканчиваются действия, выполняемые в состоянии
- Исходное и целевое состояния могут совпадать
- Одно событие может запускать переходы во множестве объектов
- Например, из состояния “звонит” телефон переходит в состояние “разговор” при подъеме трубки



Пример диаграммы состояний



Действие, событие, условие

- Действие
- Действие (action) – это атомарное и, как правило, быстрое вычисление
- Действие может представлять собой последовательность более простых действий
- Действия могут выполняться:
  - При переходе из одного состояния в другое состояние
  - При входе в состояние (entry)
  - При выходе из состояния (exit)
  - При нахождении объекта в состоянии (do)
- В Microsoft Visio do-действия описываются внутренними переходами (internal transitions). Внутренний переход не меняет состояния и ему соответствует некоторое действие.

Событие

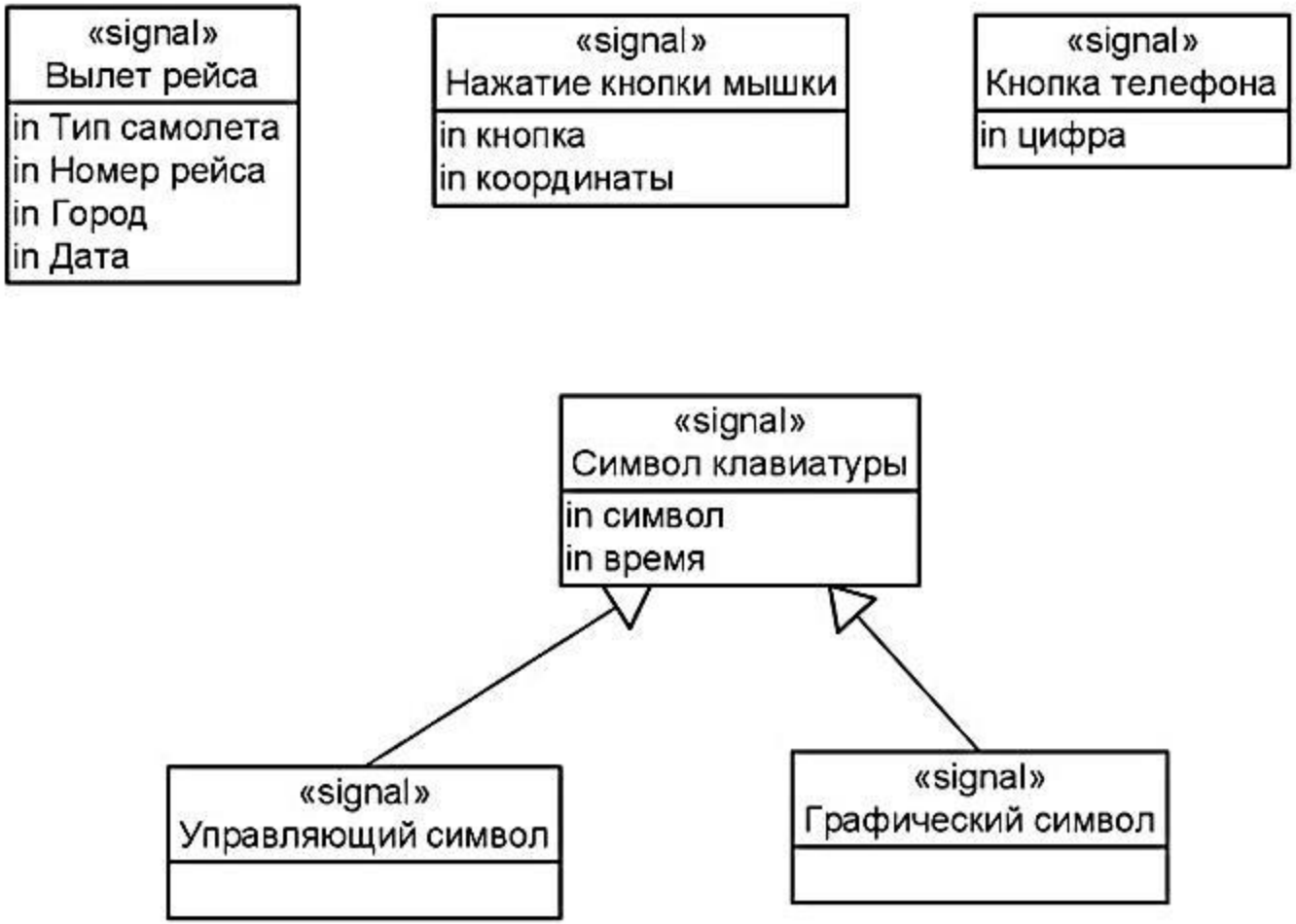
- Событие (event) – некоторое явление, которое имеет определенное положение во времени и пространстве

  Например, поступление денег на счет, нажатие кнопки, изменение температуры
- Событие происходит в некоторый момент времени и не имеет продолжительности
- Событие влияет на состояния объектов системы
- Конкретное событие называется экземпляром события и может иметь свои собственные параметры
- События бывают четырех видов:
  - Событие сигнала
  - Событие вызова
  - Событие изменения
  - Событие времени

Событие сигнала (signal event)

- Сигнал (signal) – это некоторая сущность, которая служит для передачи информации между объектами
- Сигнал имеет имя и набор параметров (атрибутов)
- Сигнал можно описать как класс со стереотипом “signal”
- Между сигналами могут быть отношения обобщения.
- Сигналы-потомки наследуют параметры своих предков и реализуют такие же переходы, что и их предки
- В Microsoft Visio сигнал нужно обязательно описать как класс, чтобы использовать его на диаграмме состояний

Примеры событий сигналов



Событие вызова (call event)

- Событие вызова – это вызов операции объекта
- С точки зрения вызывающего объекта (объекта-отправителя) такой вызов не отличим от обычного обращения к операции, которая реализуется методом
- Объект-получатель сам определяет как реализовать вызываемую операцию (метод или событие). Реализация в виде события означает переход из одного состояния в другое состояние.
- Параметры операции совпадают с параметрами события вызова
- В отличие от обычного обращения к операции событие вызова допускает параллельную работу объекта-отправителя и объекта-получателя
- Примеры событий:
  - Показать на форме список студентов курса (номер курса)
  - Зажечь лампочку на кнопке телефонного аппарата (номер телефона, номер лампочки)
  - Положить на счет некоторую сумму (номер счета, сумма)

Событие изменения (change event)

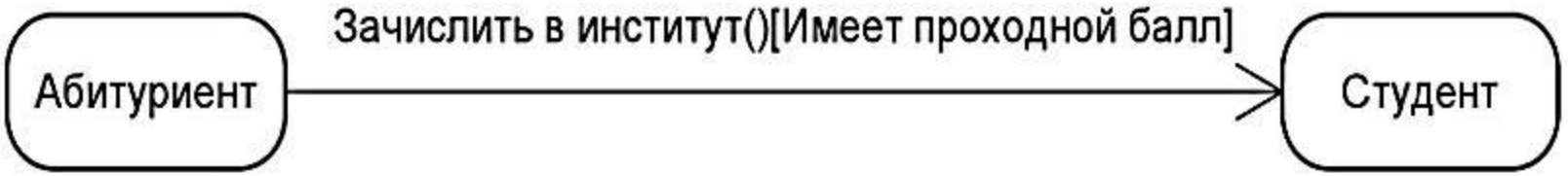
- Событие изменения наступает, когда некоторое логическое выражение принимает значение истина.
- Некоторое логическое выражение постоянно проверяется и, когда оно меняет свое значение с FALSE на TRUE, происходит событие изменения
- Примеры событий
  - when (температура в комнате < минимально допустимая)
  - when (давление в шинах < минимально допустимое)
  - when (давление газа > максимально допустимое)

Событие времени (time event)

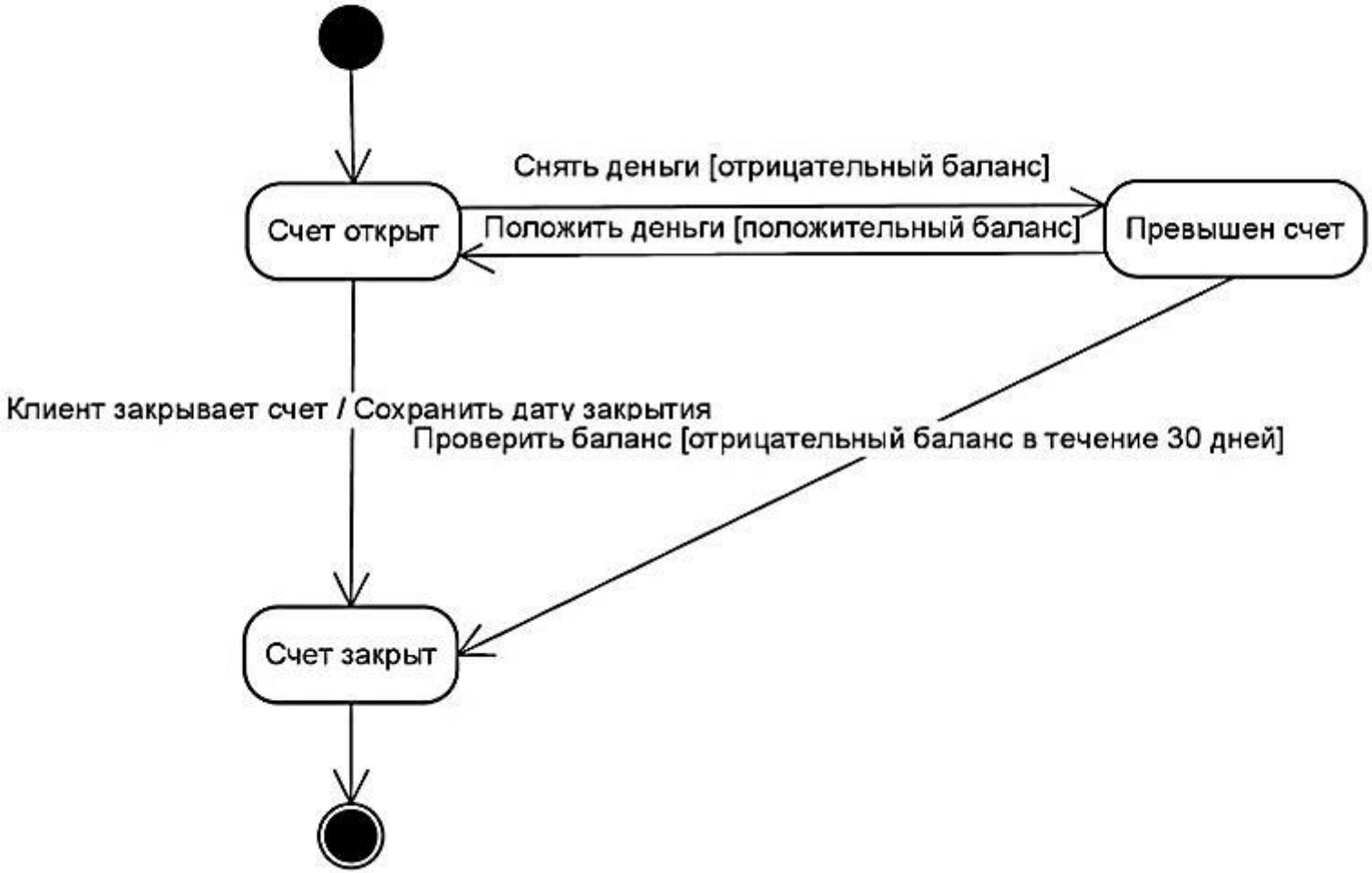
- Событие времени – это событие, которое наступает в определенный момент времени или по окончанию некоторого интервала времени
- Примеры событий
  - when (дата = 1 января текущего года)
  - after (10 секунд)

Условие

- Сторожевое условие (guard condition) – это логическое условие, которое должно быть истинным, чтобы переход был осуществлен
- Сторожевое условие проверяется в момент возникновения события



Пример диаграммы



Действия в состояниях

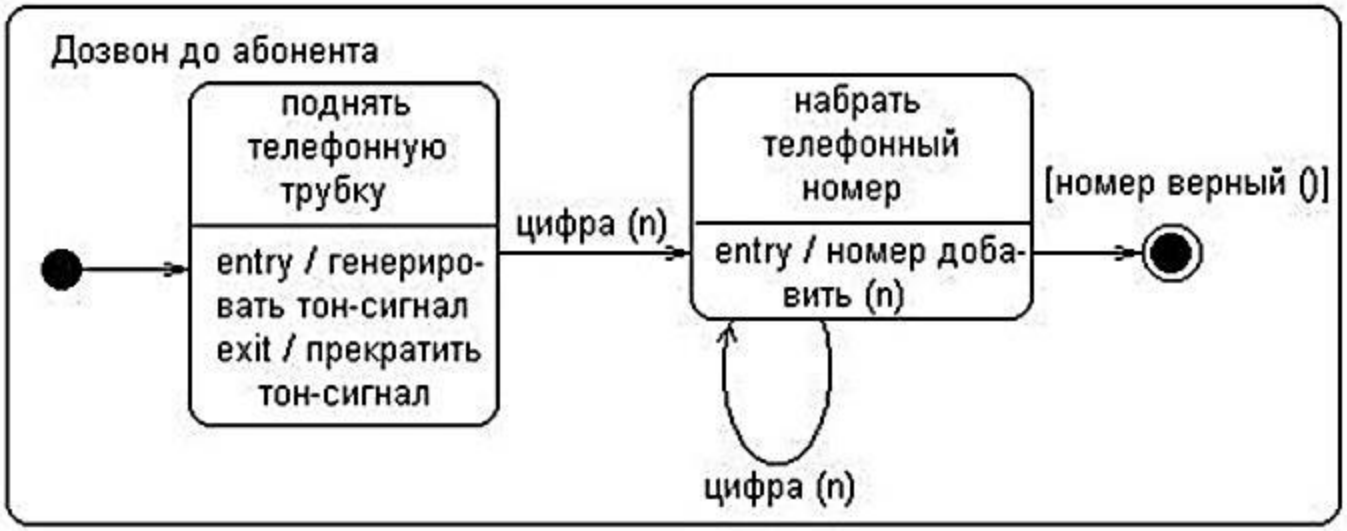
- Состояние “Превышен счет”
  - Entry/Временно заморозить счет
  - Do/Послать уведомление клиенту
  - Exit/Разморозить счет
- Состояние “Счет закрыт”
  - Entry/Выдать кредитную карточку

Составное состояние

- Составное состояние (composite state) - такое сложное состояние, которое состоит из других вложенных в него состояний. Последние будут выступать по отношению к первому как подсостояния (substate). Хотя между ними имеет место отношение композиции.

Последовательные подсостояния

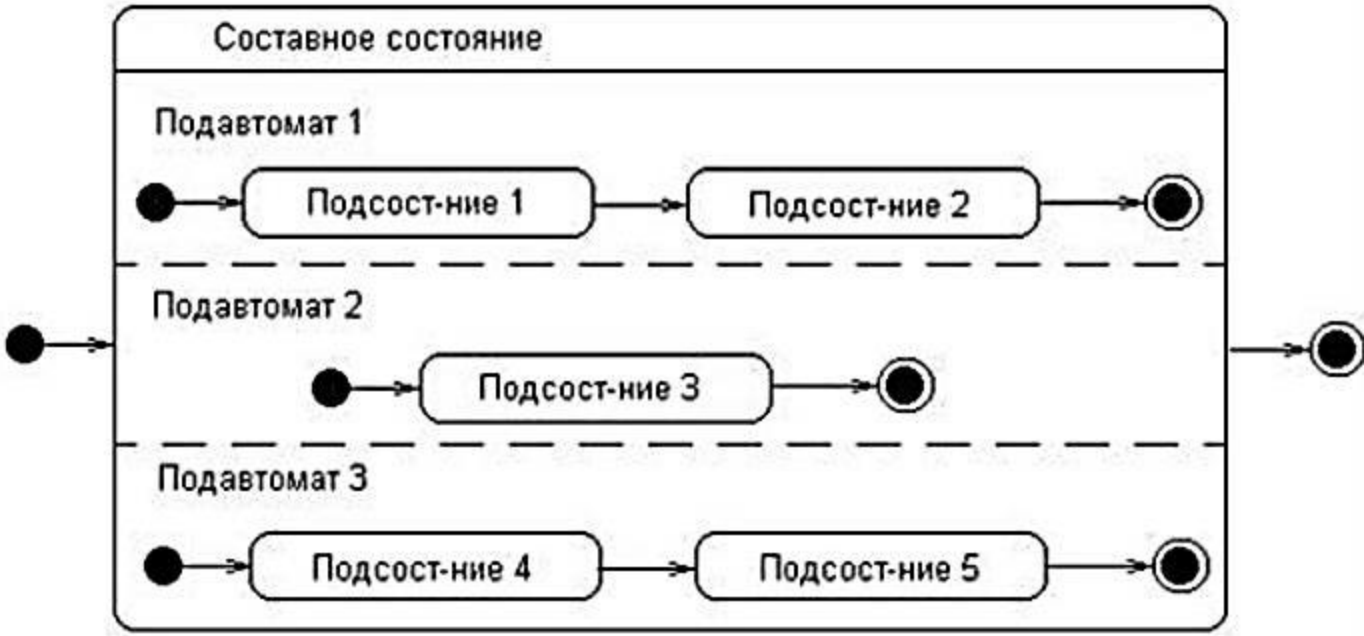
- Последовательные подсостояния (sequential substates) используются для моделирования такого поведения объекта, во время которого в каждый момент времени объект может находиться в одном и только одном подсостояний.



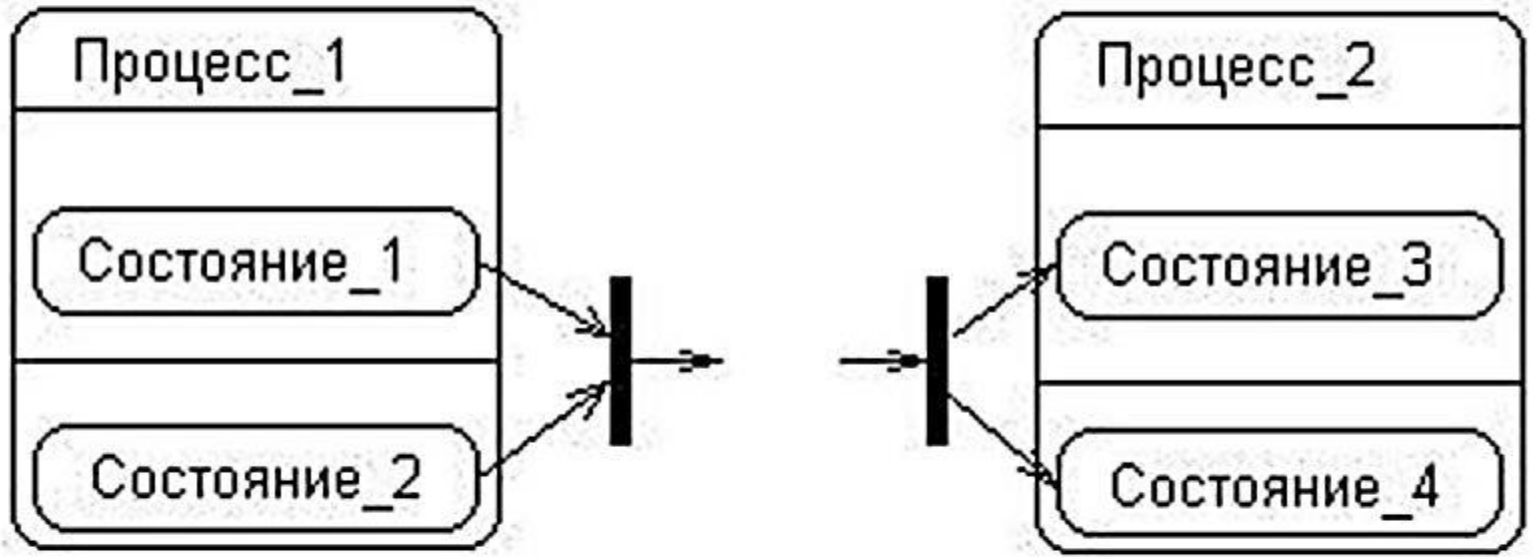
- составное состояние с двумя вложенными последовательными подсостояниями

Параллельные подсостояния

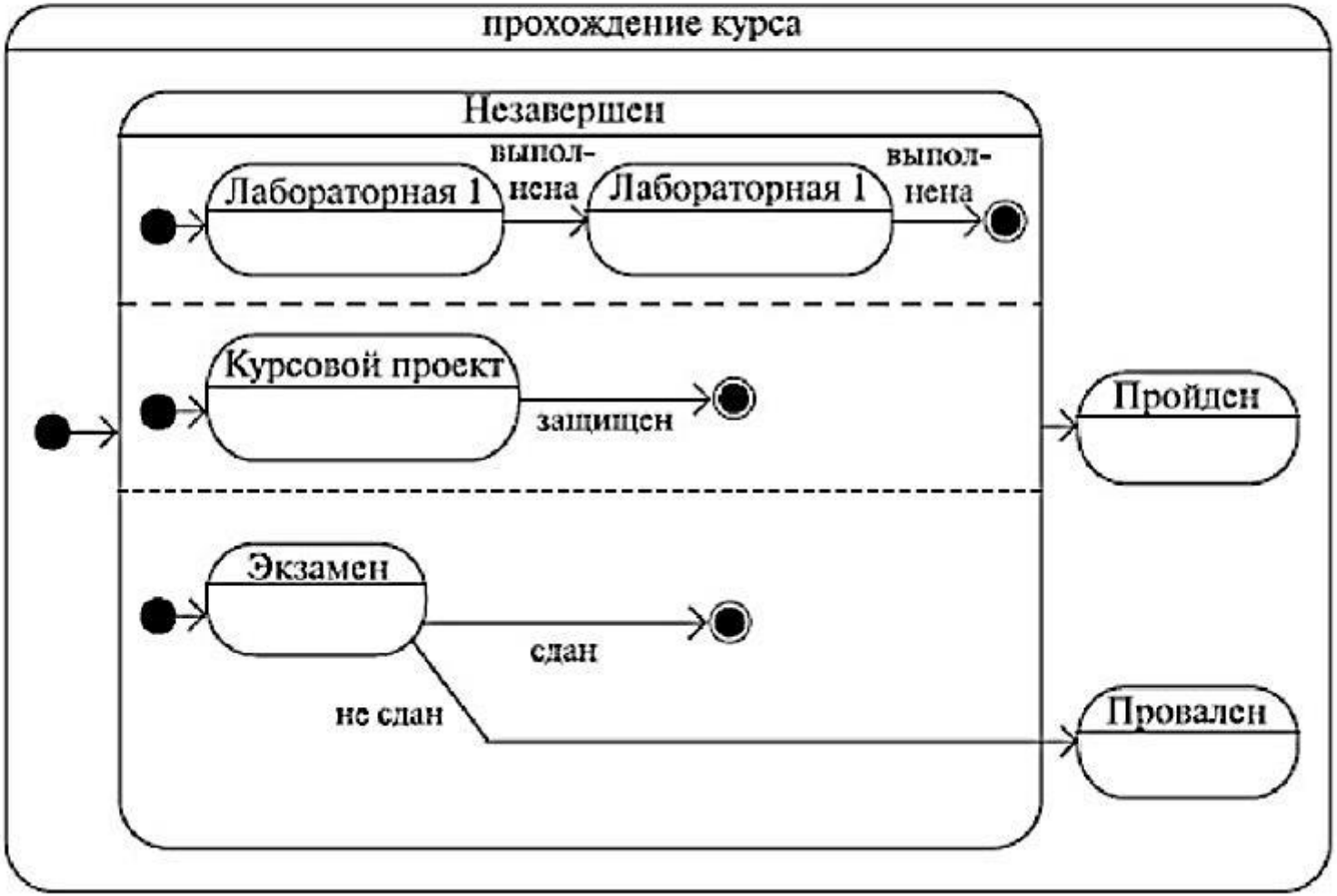
- Параллельные подсостояния (concurrent substates) позволяют специфицировать два и более подавтомата, которые могут выполняться параллельно внутри составного события.



Переход между параллельными состояниями



Составное состояние с параллельным выполнением



Построение Диаграмм Состояния

- Определение содержания
- Выявление исходного, конечного и устойчивых состояний.
- Определение порядка перехода через устойчивые состояния
- Выявление событий, действий и условий, связанных с транзакциями
- Построение диаграммы