# UML

# Понятие диаграммы

Визуальное отображение некоего факта.

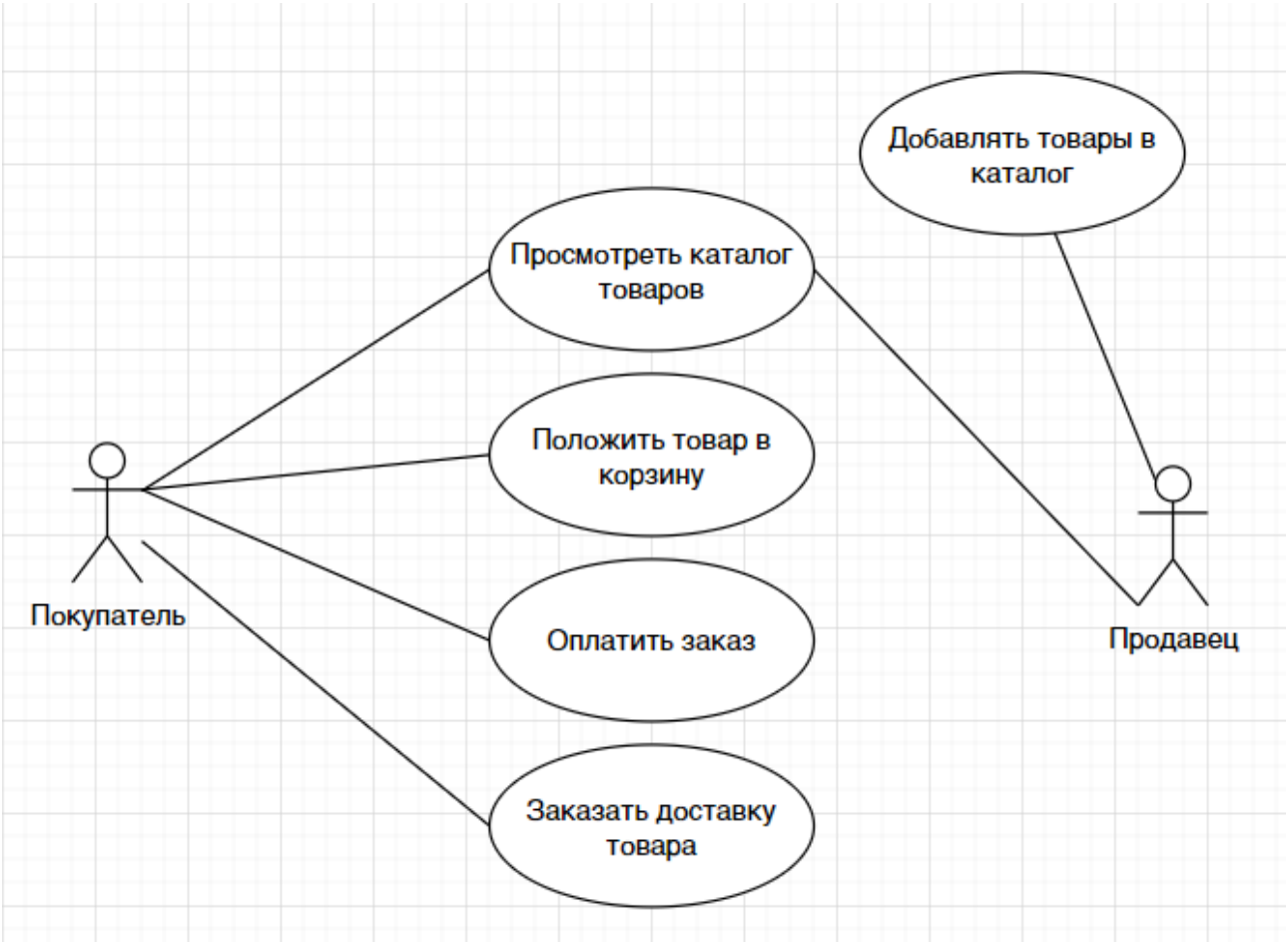
Целью диаграммы UML является описание некоего программного комплекса с разных сторон.

draw.io

# Диаграмма вариантов использования (user case diagram)

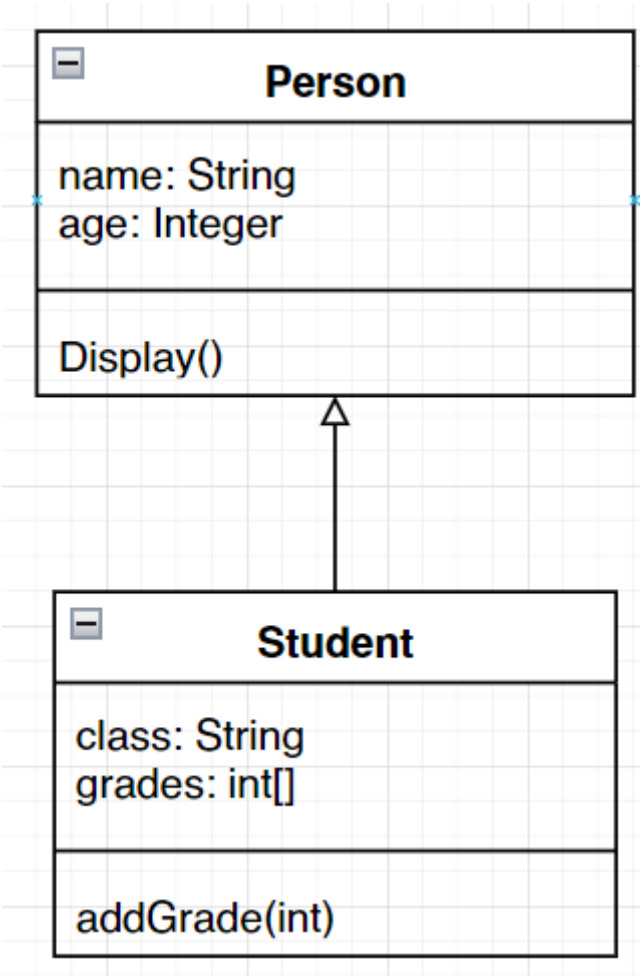
Предоставляет возможность анализировать функционал приложения.

Он очень прост и понятен любому человеку. Он отображает сценарий использования приложения в обычных человеческих терминах.



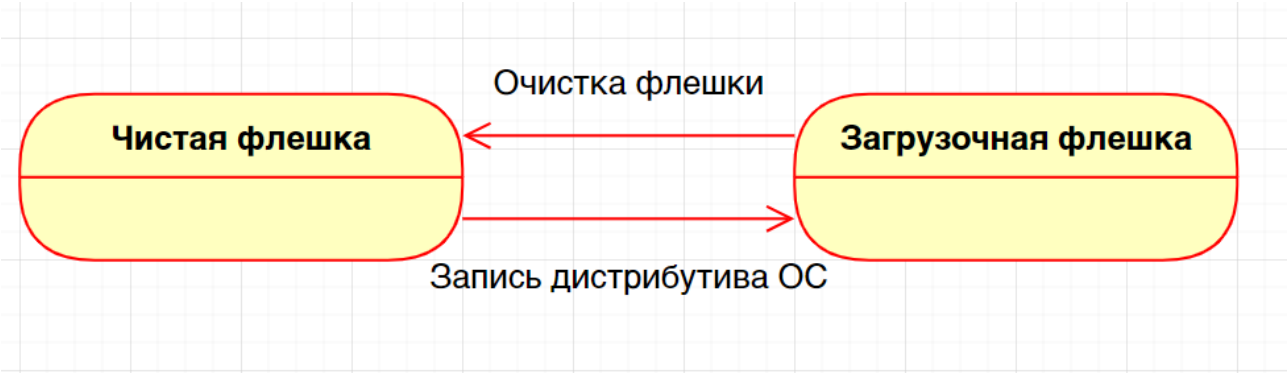

# Диаграмма классов

Описывает структуру программного приложения, показывает классы, их атрибуты, их методы и их взаимодействие.




# Диаграмма состояния

Это совокупность свойств системы. Отображает как система переходит между состояниями.



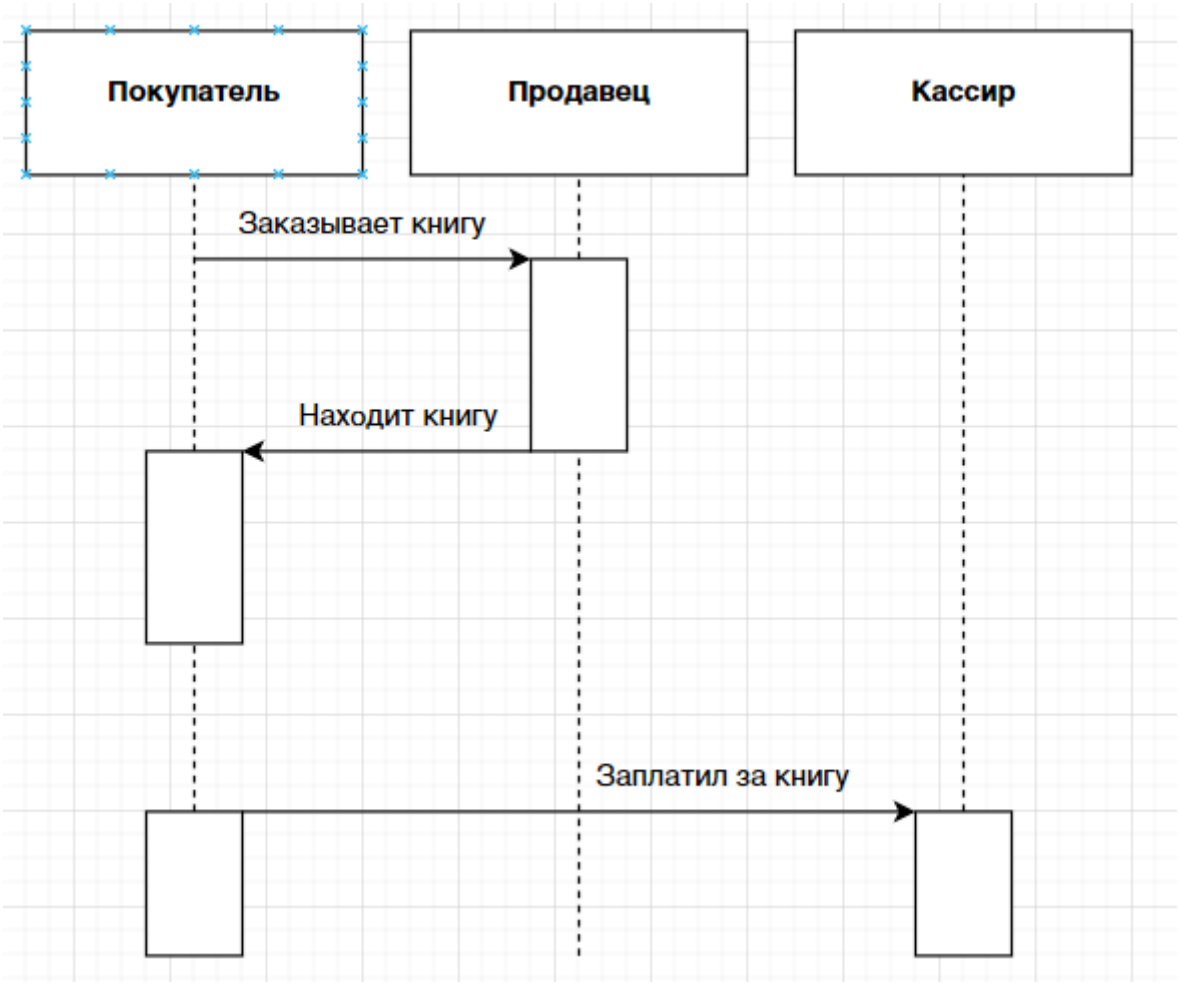

# Диаграмма деятельности

Аналогична уже знакомым блок-схемам. Блок-схемы по своей сути — частный случай диаграмм деятельности.

Предназначены для визуализации алгоритмов.

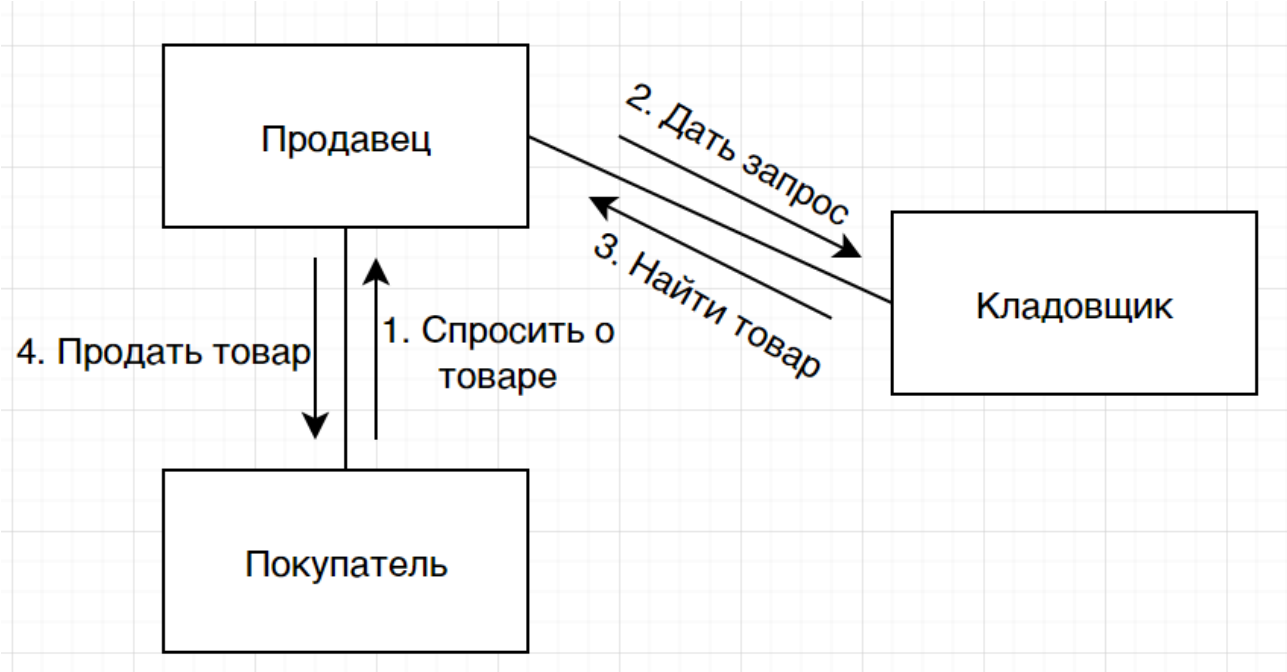

# Диаграмма последовательности

Используется для отображения упорядоченного во времени взаимодействия объектов.



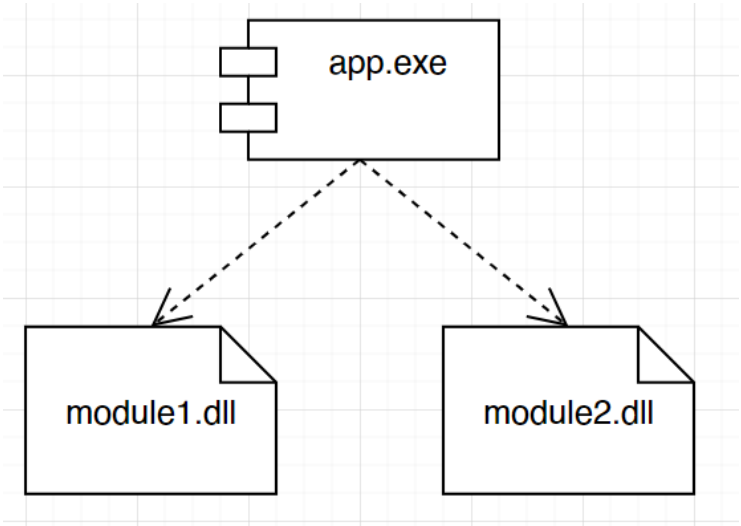

# Диаграмма кооперации

Отображает поток сообщений между объектами в системе. Аналогична диаграмме последовательностей.



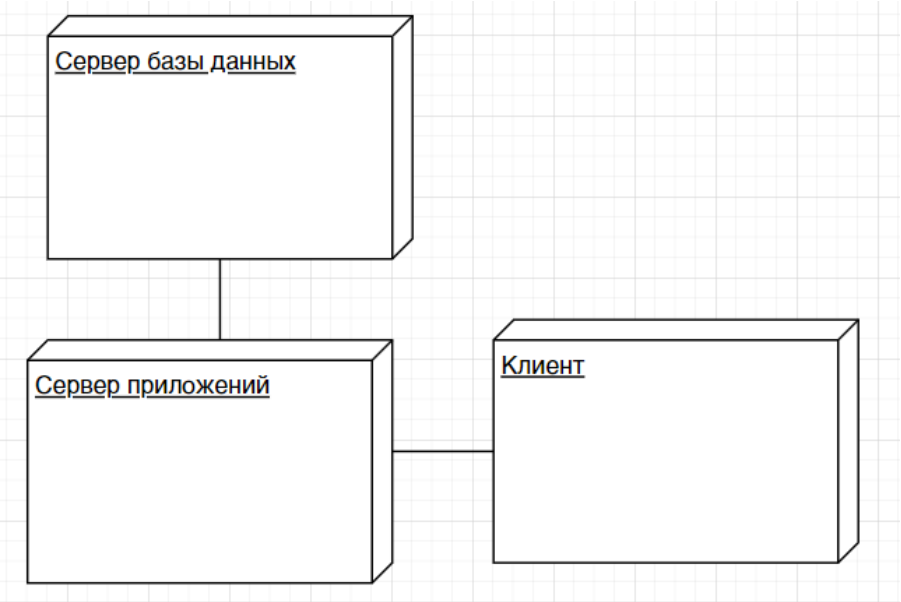

# Диаграмма компонентов

Отображает разбиение программного проекта на множество структурных компонентов. В качестве компонентов для программной системы могут выступать библиотеки, файлы, модули, ассеты и так далее.




# Диаграмма развёртывания

Описывает, что нужно, чтобы программа начала работать с точки зрения пользователя.




# Диаграмма классов

Описывает структуру классов — их атрибуты, методы, интерфейсы.

Цели данного типа диаграмм:

- Описание модели системы типов (словарь системы)
- Отражение простейших кооперaций. Кооперация — это совокупность классов, интерфейсов и т. п., действующих совместно для реализации некоего кооперативного поведения. Эта цель отражает эмерджентность — это свойство систем иметь свойства, не сводимые к свойствам составляющих её элементов.
- Отражение структурного аспекта организации данных

Три точки зрения Мартина Фаулера

- **концептуальная точка зрения**. Диаграммы классов служат для представления понятий изучаемой предметной области
- **точка зрения спецификации.** Имеет значение прежде всего интерфейс. Но ООП рассматривает интерфейсы отдельно от реализации. Когда говорят о проектировании класса, часто имеют в виду проектирование его интерфейсов.
- **точка зрения реализации**. С данной точки зрения, мы проектируем непосредственно классы, как мы будем их реализовывать. Более адекватной ситуации проектирования считается точка зрения спецификации.

## Базовые понятия

### *Класс*

Всякое понятие UML зависит от уровня, где оно используется.

На концептуальном уровне класс — это одно из понятий предметной области.

На уровне моделирования — это некоторый абстрактный тип данных.

На уровне реализации — это уже формальный тип данных, описанный на объектно-ориентированном языке программирования.

**Атрибут —** это данные, которые хранит класс, и которые его некоторым образом характеризуют.

```
видимость имя_атрибута: тип = значение_по_умолчанию
```

- Видимость указывается с помощью знаков +, #, -
  - + public
  - - private
  - # protected
- Имя_атрибута — это последовательность символов, его характеризующая.
- Тип — это спецификация типа атрибута.

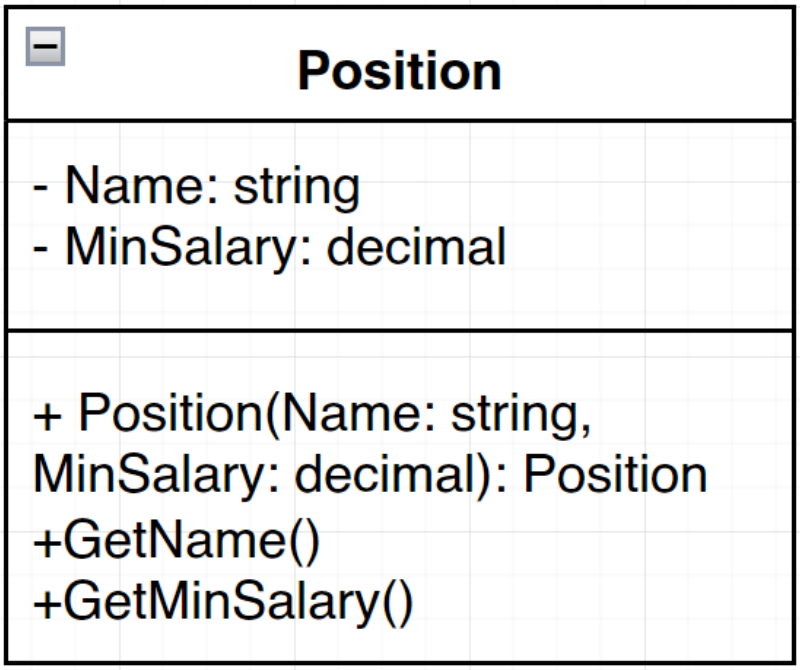
```
# Name: string
- MinimalSalary: decimal = 20000
```

**Операция —** это процессы, которые выполняет класс. Как правило, описанные на концептуальном уровне операции соответствуют методам класса на уровнях спецификации и реализации.

```
видимость имя_операции(список аргументов):
тип_возвращаемого_значения(свойства_операции)
```

- Видимость определяется также, как для атрибутов.
- Имя операции тоже последовательность символов.
- Список аргументов состоит из аргументов, разделённых запятой, каждый из которых записан по такому синтаксису:
  - `направление имя_параметра: тип = значение_по_умолчанию`
  - где направление может принимать значения *in*, *out*, *inout* (*in* чаще всего опускается)
- тип_возвращаемого_значения может состоять из типов данных, разделённых запятой, но обычно возвращается только один.
- свойства_операции — это список разделённых запятой свойств операции, и часто опускается.

```
+GetMinSalary(): decimal
+GetName(): string
+Position(Name: string, MinimalSalary: decimal)
```

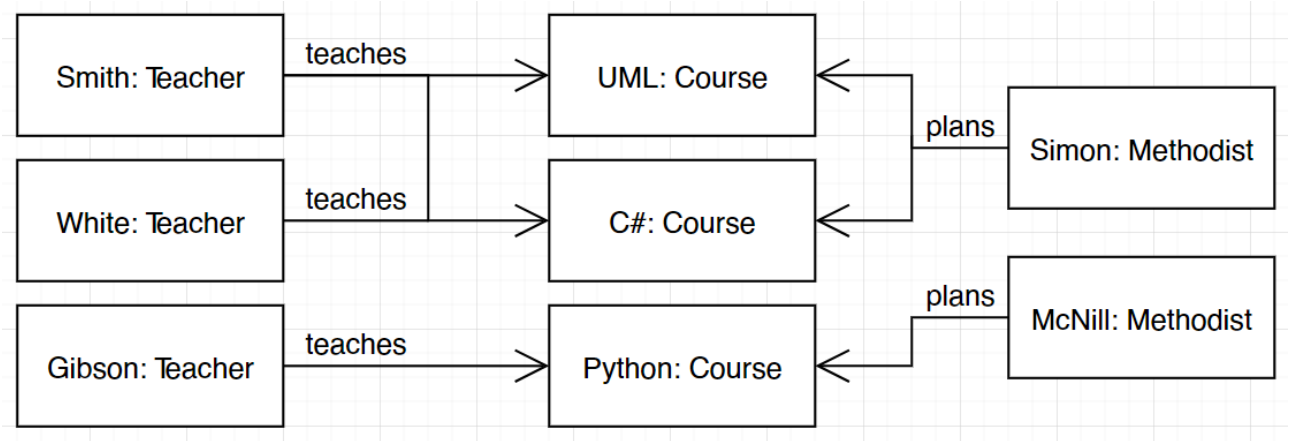



**Объект —** на концептуальном уровне и уровне моделирования, это экземпляр некоего класса. Класс описывает некоторое понятие о некоем предмете проблемной области, и «объект» - это собственно сами объекты, описанные при помощи классов.

На уровня реализации, объект — это экземпляр реализованного класса, который создан и хранится в памяти компьютера.

В UML существует **диаграмма объектов**. Она не классическая для UML, но иногда нужна для уточнения контекста использования классов и для представления функционального назначения класса в общей системе.

Например, у нас есть система, где есть три класса: Teacher, Course, Methodist. Teacher (Преподаватель) может читать несколько курсов (Course), и один курс могут читать несколько преподавателей, в то время, как над курсом может работать только один методист (Methodist), хотя он может работать над несколькими курсами.



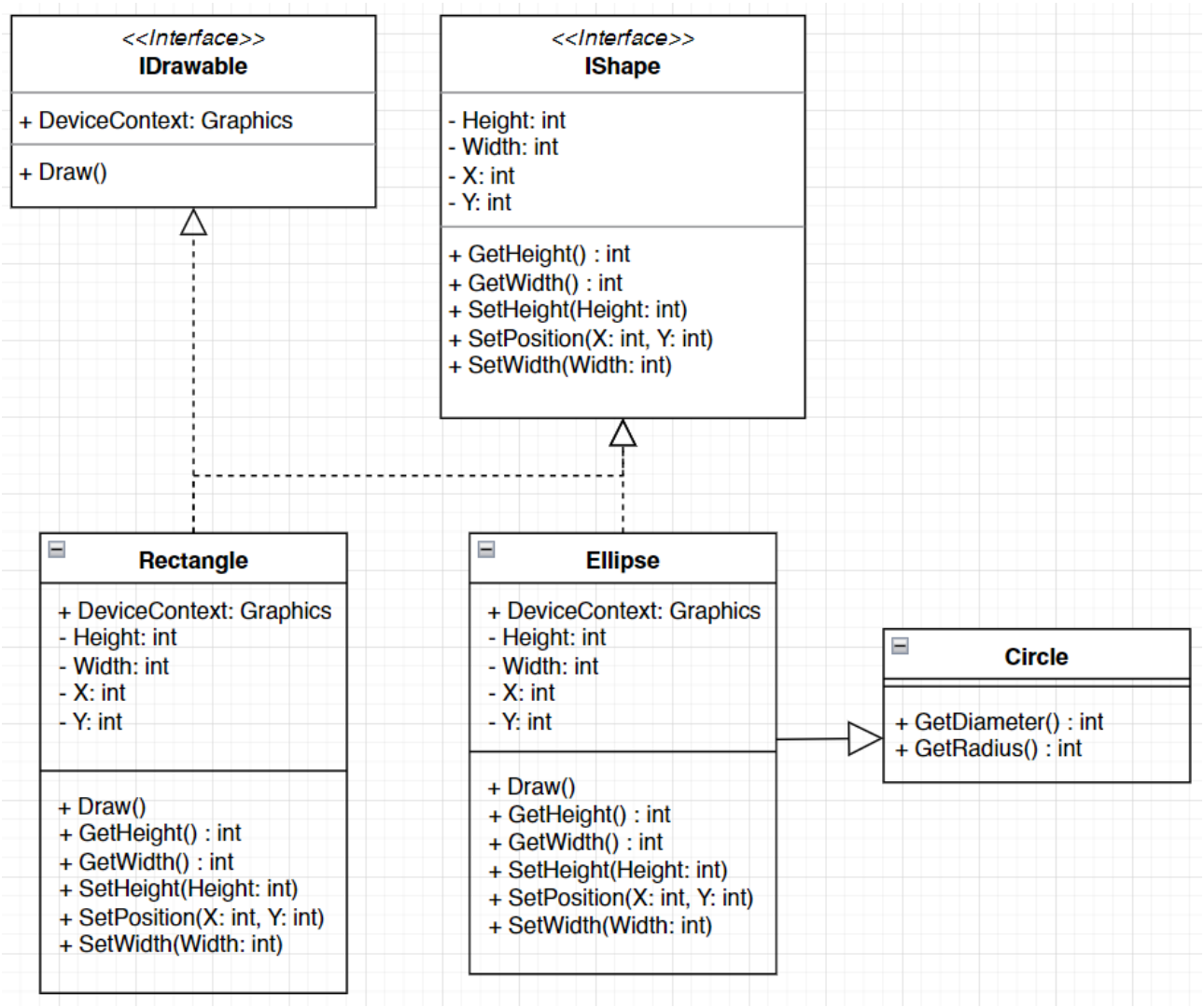

### *Интерфейс*

**Интерфейс** — имеет две стороны, два определения.

С одной стороны, это набор public операций класса, которые в совокупности описывают различные способы использования класса.

С другой стороны — это категория UML, которая содержит описание функциональности некоего понятия. В общем, интерфейс — это именованный набор открытых свойств, выраженный в виде типа.

Основная идея — это внесение разделения между описанием функциональности и реализации.

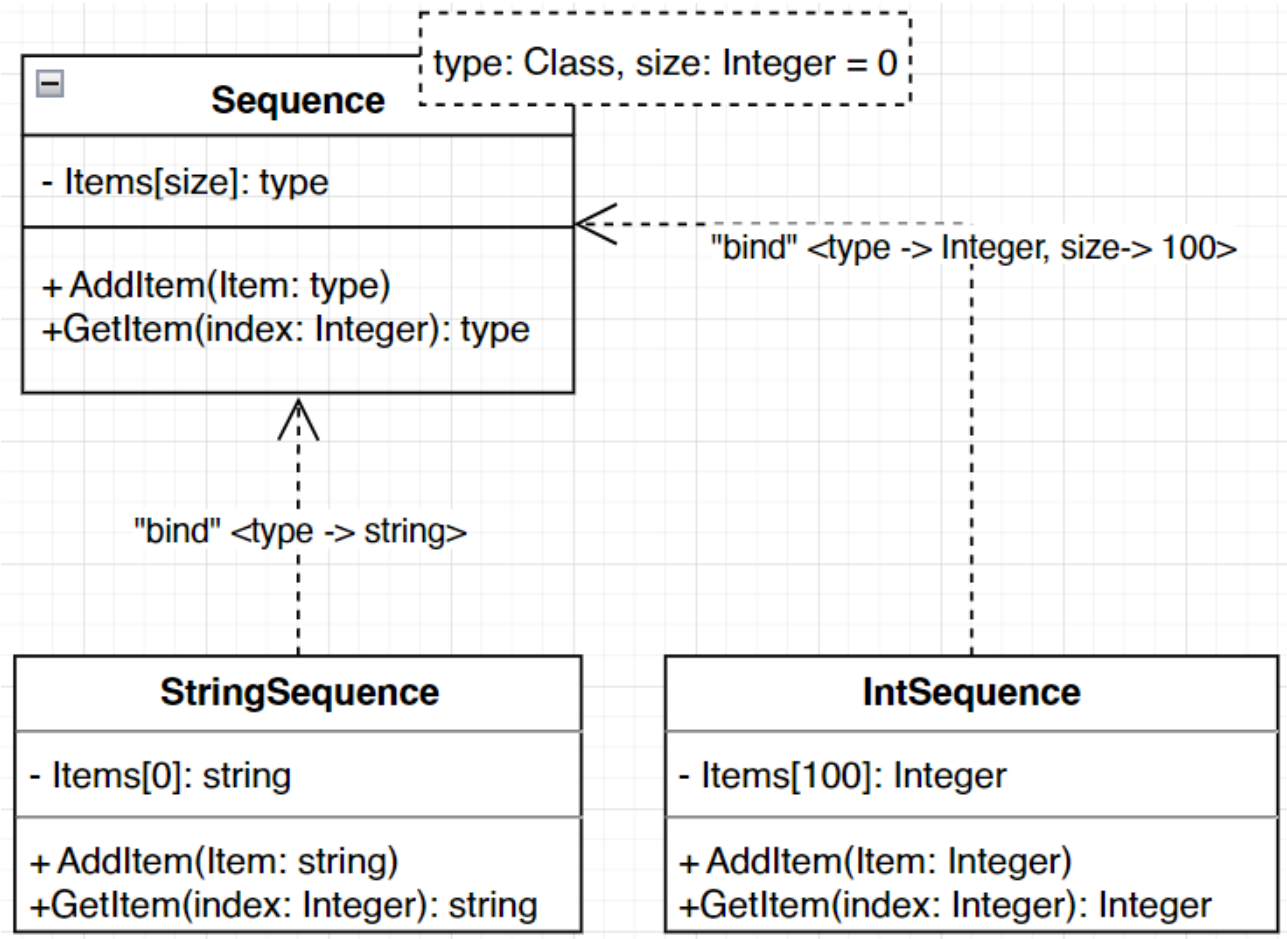



### *Шаблон*

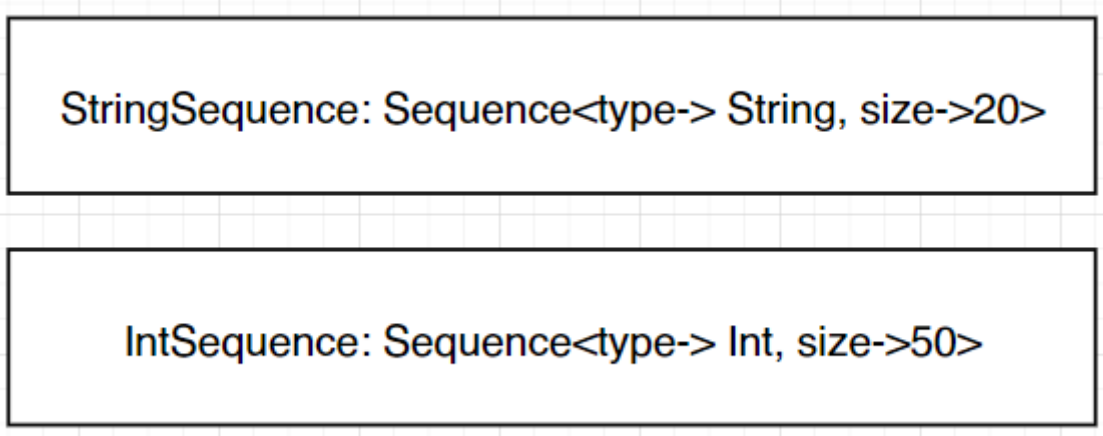
При описании класса иногда можно не указывать напрямую типы принимаемых или возвращаемых операциями значений. Вместо этого используют «структурные нули» или «заполнители» (placeholders). Она заменяются фактическим значением типа при создании нового класса.

**Шаблон** описывается похоже на класс, но в углу в пунктирном контейнере указываются параметры шаблона

```
имя_параметра: тип_параметра = значение_по_умолчанию
```




Связываются с объектами с помощью стереотипов bind. Могут явные (выше) и неявные (ниже).



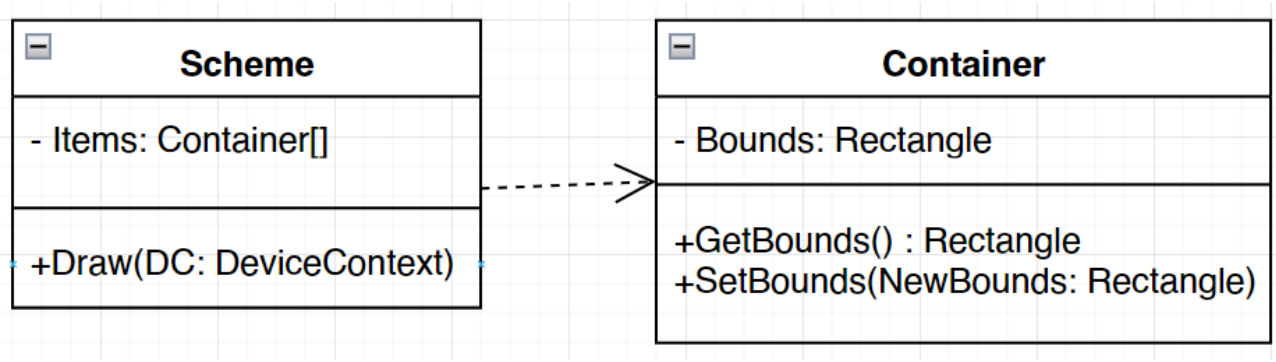

## Отношения между классами (relationships)

Структура — это совокупность составляющих систему элементов вне зависимости от их связей.

Характеры связей между элементами определяет характер процессов, протекающих в системе.

### *Отношение зависимости (dependency relationship)*

Это отношение при котором один элемент (клиент) использует другой элемент (поставщик, supplier), или же зависит от него.



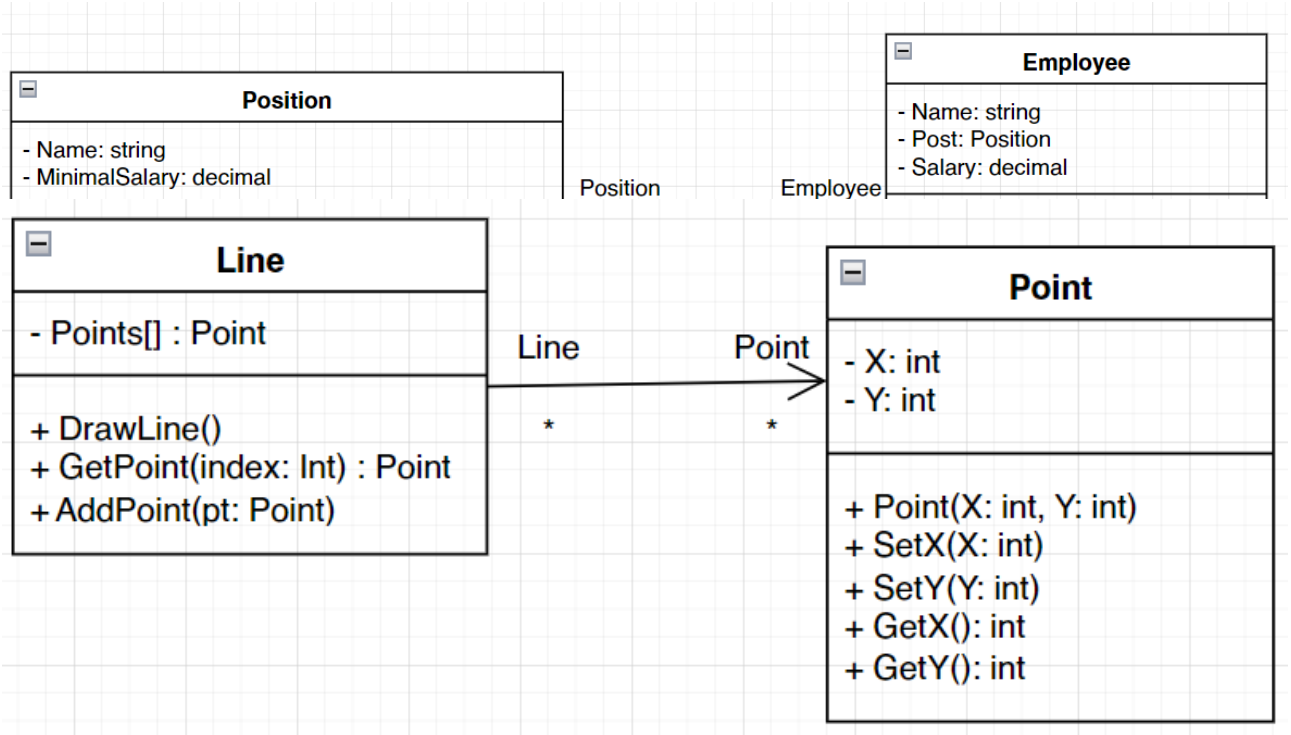

Обычно, они не подписываются, но их можно подписать для определения характера зависимости. Существуют следующие виды:

- абстракция (abstraction, derive, refine, trace) – определяет отношение между двумя или более элементами модели, которые описывают одно и то же понятие на разных уровнях абстракции.
- связь (bind) – связывает аргументы шаблона с параметрами шаблона в объектах.
- реализация (realize) – указывает, что элемент клиент — реализация элемента-поставщика.
- замещение (substitute) – указывает, что элемент-клиент занимает место элемента-поставщика. При этом, интерфейс клиента должен обязательно соответствовать интерфейсу поставщика.
- использование (use, call, create, send, instantiate) – указывает, что один элемент использует другой для реализации той или иной функции.

### *Отношение ассоциации*

Отношение между двумя классификаторами, которое описывает причину отношения и правило, которое определяет отношение. С одной стороны, ассоциация — это структурное отношение, с другой — атрибут, определяющий свойства классификатора.

Обычно ассоциации помечаются метками на концах стрелок. Метки обычно по две на каждый конец — название ассоциации и её кратность. Обычно, * означает сколько угодно связей.



### *Отношение обобщения (generalization)*

Отношение при котором один элемент (дочерний) базируется на другом (родительском).

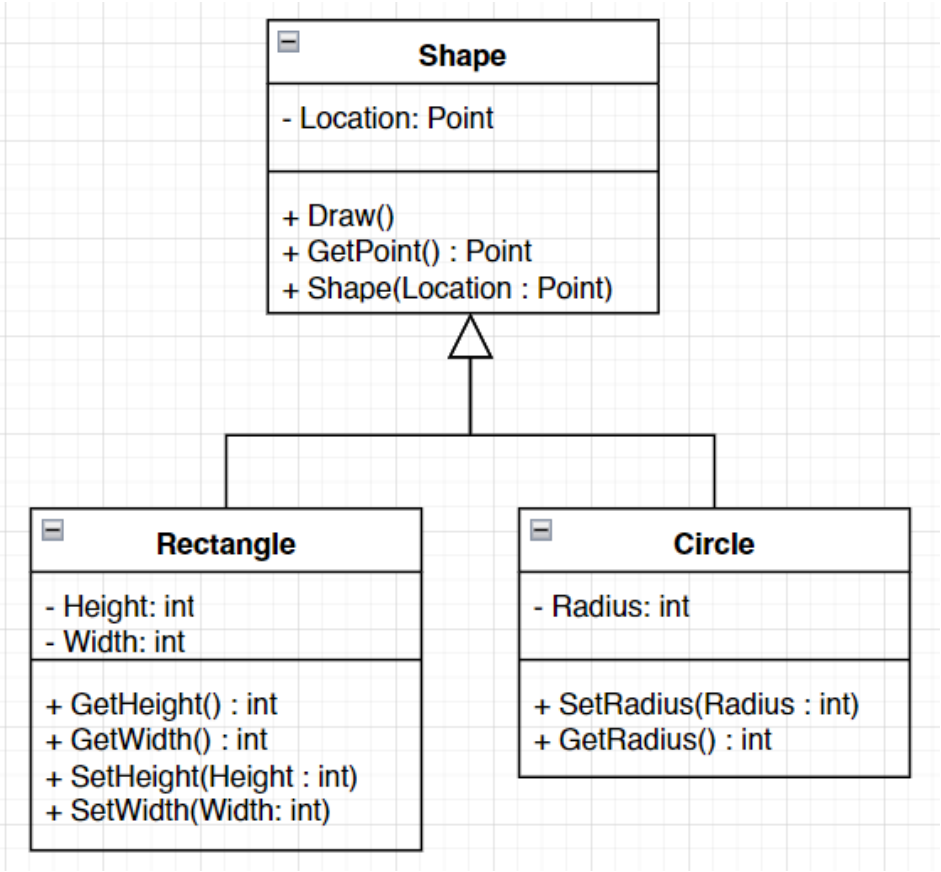
Используется в диаграммах вариантов использования, классов, компонентов и развёртывания.

Отношение обобщения можно использовать только для однотипных классификаторов.

Например, оно может быть между двумя классами, но не между классом и вариантов использования.

Один родительский элемент может иметь несколько дочерних. Один дочерний может иметь несколько родительских, но обычно это нежелательная практика.

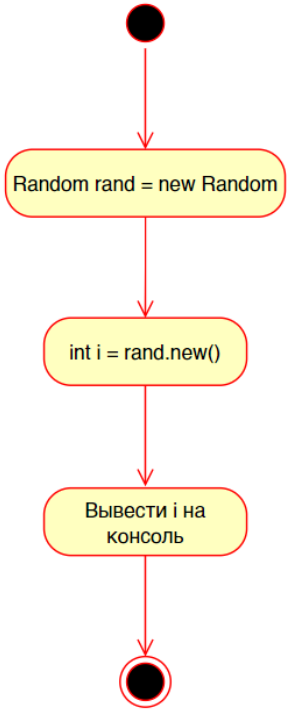
Обобщения не имеют имён. На уровне реализации, соотносятся с наследованием.




# Диаграмма деятельности

Диаграммы алгоритмов представляют из себя частный случай диаграмм деятельности.

## Базовые понятия




**Состояние действия** — это состояние с входным действием и минимум одним исходящим переходом. Условие перехода — завершение действия.

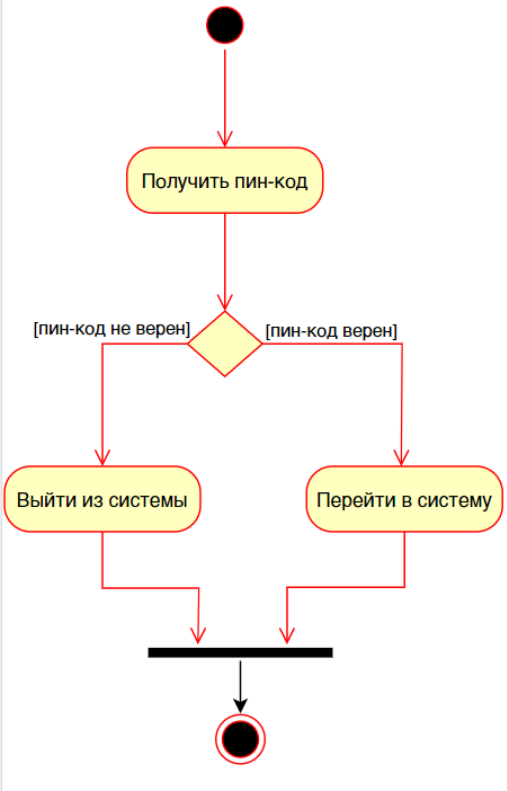
Может содержать код на языке программирования, псевдокод или глагол в побудительном наклонении.

Принято располагать сверху вниз.

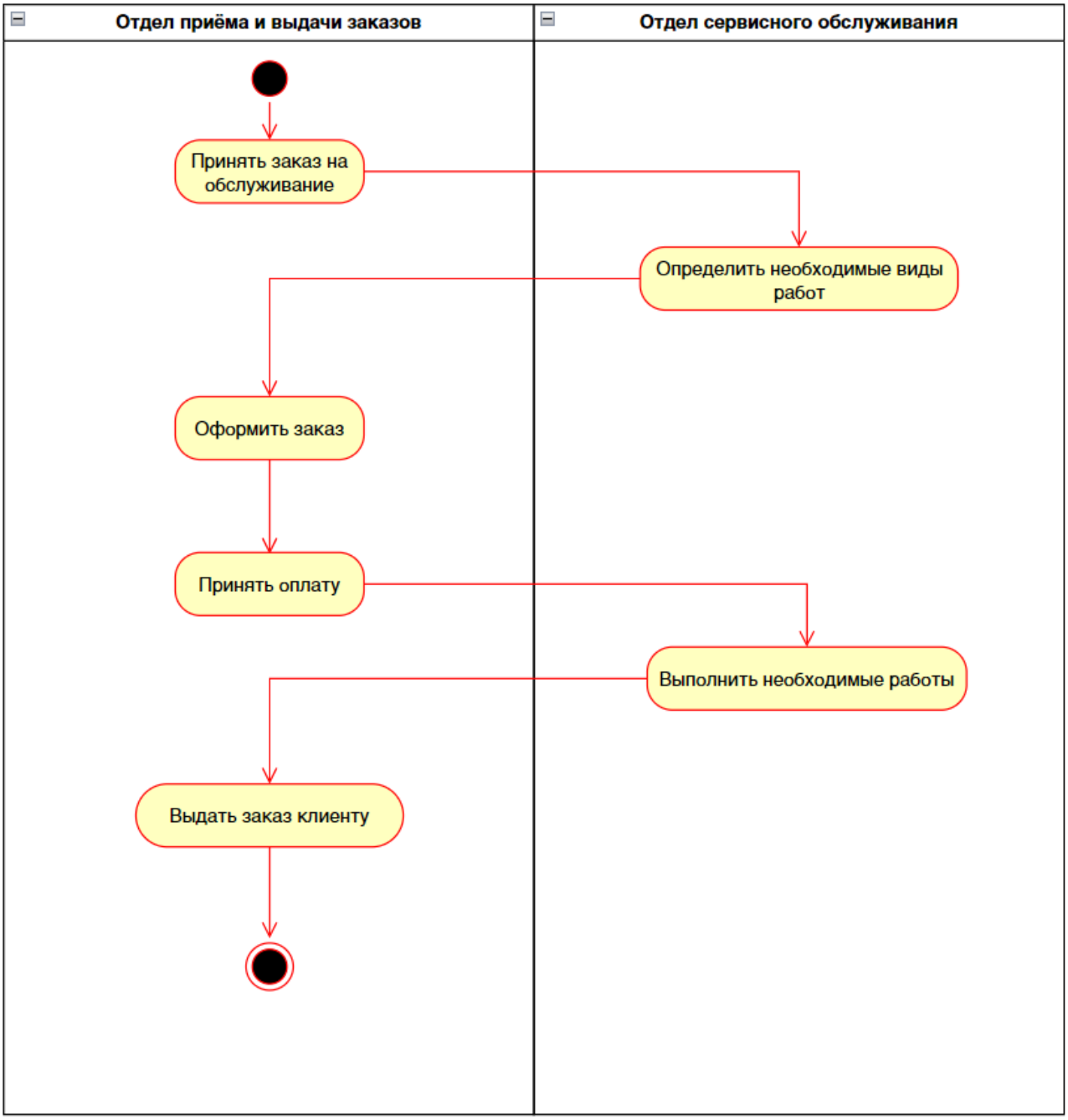
**Переход —** аналогичен переходам в диаграмме состояний. В диаграммах деятельности используются только нетриггерные переходы.

Если из состояния действия есть один переход, он не содержит подписей. Изображается в виде сплошной стрелки, направленной от исходного состояния к целевому. Если их более одного, их помечают сторожевыми условиями. Указываются в прямоугольных скобках возле стрелки. При разделении при каких-то условиях говорят о ветвление.

Графически ветвление изображают, как ромб с одним входящим переходом и минимум двумя исходящими. Вход соединяют либо с верхней, либо с левой вершиной ромба. И также бывает объединение переходов, когда они сходятся к одному действию. Объединение переходов имеет несколько входящих и один исходящий переходы.

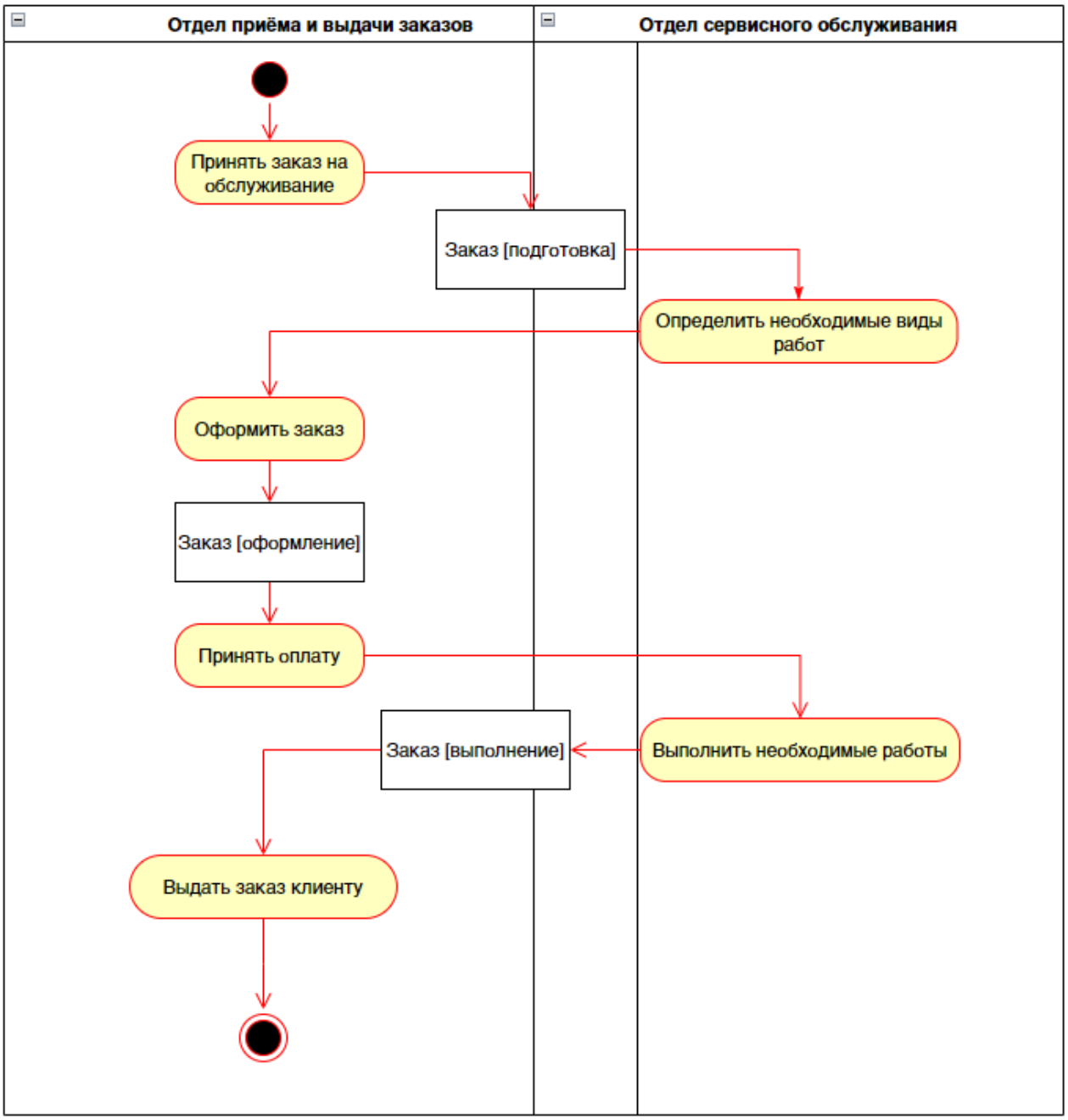



**Дорожка (swimlane) —** отделяют друг от друга части системы, например отделы внутри компании при осуществлении человеческой деятельности.




**Объекты** — это над чем выполняются действия. При этом обычно отображаются состояния объекта, а не сам объект в чистом виде. Записывается, как

`имя_объекта [состояние_объекта]`


Отдел приёма и выдачи заказов
Отдел сервисного обслуживания
Принять заказ на обслуживание
Заказ [подготовка]
Определить необходимые виды работ
Оформить заказ
Заказ [оформление]
Принять оплату
Заказ [выполнение]
Выполнить необходимые работы
Выдать заказ клиенту

Диаграммы последовательности отражают временную природу взаимодействий в системе.

Диаграммы кооперации отражают структурную сторону взаимодействий в системе.

# Диаграммы последовательности

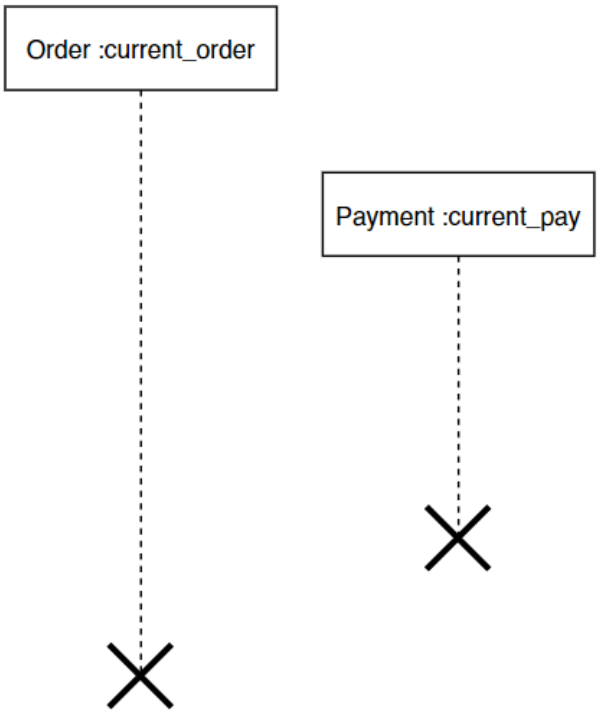
## Базовые понятия

**Объект** — возникают, когда заходит речь о реальном взаимодействии. Взаимодействие подразумевает несколько участников — соответственно, в диаграммах последовательности обычно несколько объектов. Описывается как имя объекта и его класс.

```
Type_Name: Object_Name
```

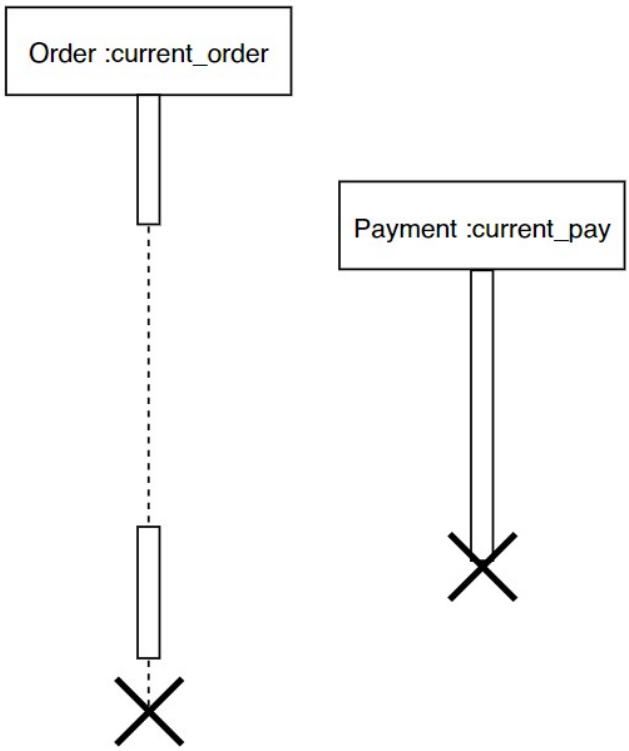
Временная шкала направлена сверху вниз.

**Линия жизни —** используется для демонстрации периода времени, в который существует объект. Отображается пунктиром, заканчивается символом момента уничтожения.



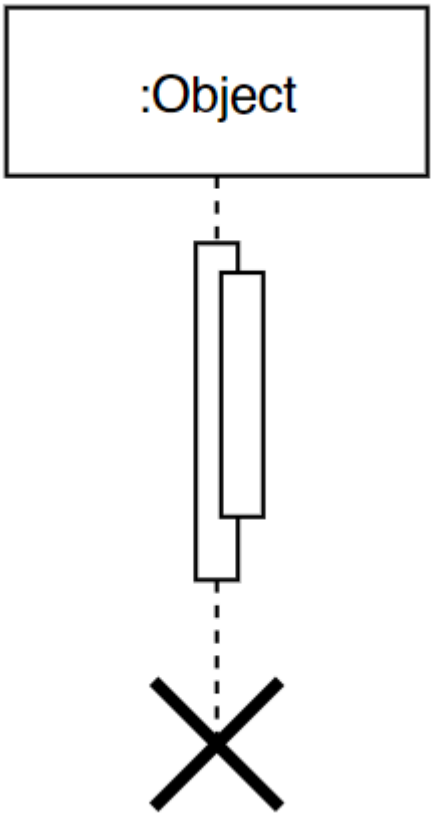

Не все объекты существуют с начала осуществления взаимодействия. Вспомогательные объекты создаются позже, и уничтожаются, когда в них отпадает необходимость.

Фокус управления — это графического отображение активного состояния объекта в пределах его линии жизни.



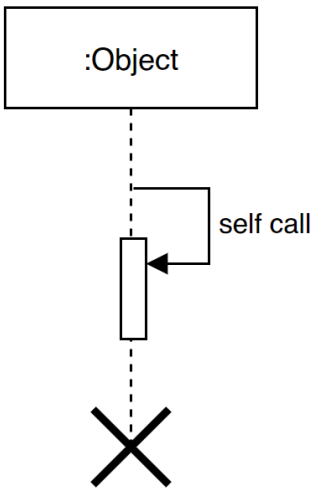

Бывают также ситуации, когда источником активности выступает стороннее по отношению к системе «действующее лицо», например, пользователь. Его пунктирная линии не имеет символа уничтожения, так как деятели могут присутствовать на протяжении всего существования системы.

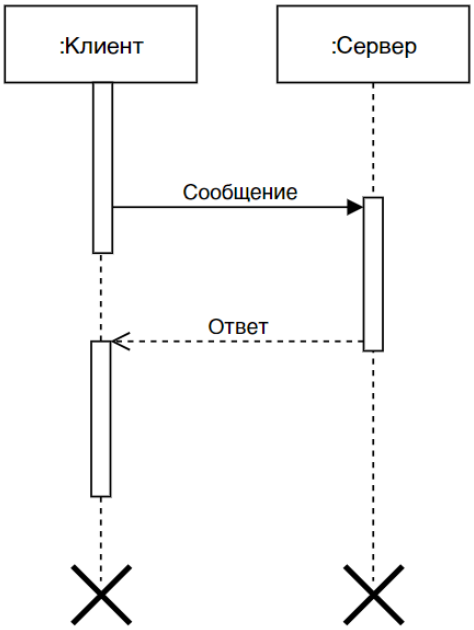
Действующее лицо может иметь имя, а может выступать анонимно.




Для рекурсивных вызовов, когда объект взаимодействует сам с собой, тоже существует обозначение в виде второго прямоугольника на фокусе управления.

Если же объект инициализирует обращение к самому себе не рекурсивного характера (например, вызов объектом собственного метода), то это называется рефлексивным обращением и изображается в виде стрелки на фокусе управления.





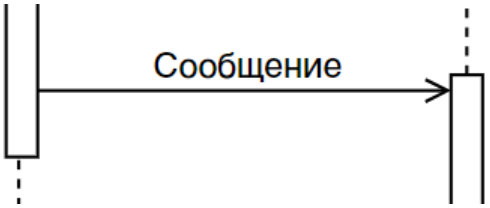

**Сообщения** исходят от клиента к серверу, сообщение носит форму запроса, а ответ — это сообщение о выполнении запрошенных действий. Сообщение — это горизонтальная стрелка от времени у клиента к времени у сервера, а ответ — пунктирная стрелка наоборот.

Принимается допущение, что время передачи сообщения настолько мало по сравнению с временем на исполнение запроса в сообщении, что им можно пренебречь.

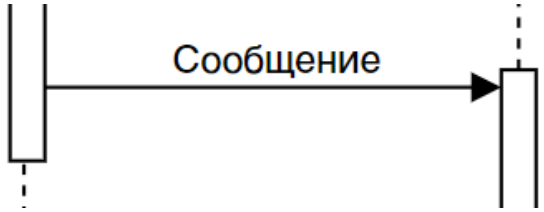
Также, принято считать, что во время передачи сообщения, с соответствующим объектом не может происходить никаких изменений.

Есть различные виды сообщения, для которых существуют разные обозначения.

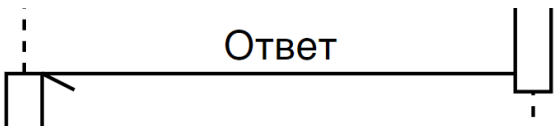
- Синхронное сообщение — это наиболее часто встречаемый вид сообщений. Вызов процедур, операций и вложенных потоков управления. Начало соприкасается с линией жизни или фокусом управления клиента, а конец с линией жизни сервера. Обычно, принимающий объект становится активным.

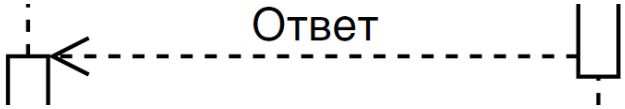



- Сообщение вызова — используется для обозначения не вложенного потока управления. Стрелка от линии жизни или фокуса управления у клиента к фокусу управления у сервера. Один шаг потока управления, обычно асинхронны.




- Асинхронное сообщение — это сообщение между двумя объектами в некоей последовательности. В чистом виде, примером может являться отправка объектом-сервером сообщение о переходе процедуры на новый этап.

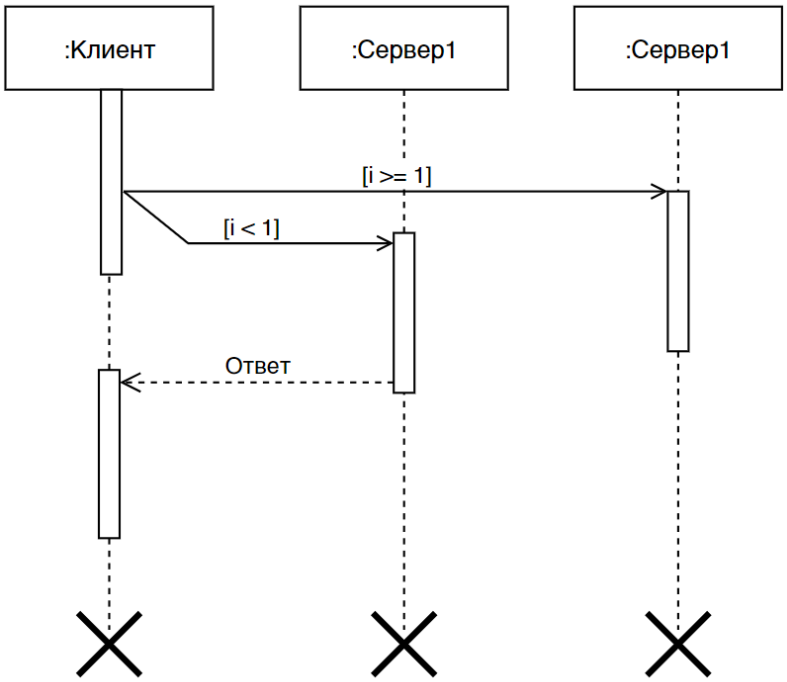



- Сообщение возврата — это сообщение о том, что сервер выполнил действие и вернул какие-

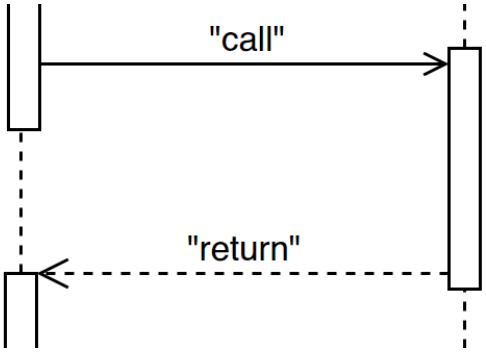
то данные объекту-клиенту. Может быть опущено, когда из контекста ясно, что сервер должен вернуть результат. Каждый вызов имеет парный возврат.

**Ветвление —** возникает, когда поток управления ветвится по условиям. Условия обозначаются у стрелок-сообщений, подразумевается, что они исключают друг друга.




**Стереотипы сообщений —** это некоторое стандартное действие, выполняемое в ответ на сообщение. Указывается в двойных кавычках возле сообщения.

В UML есть стандартный набор стереотипов:

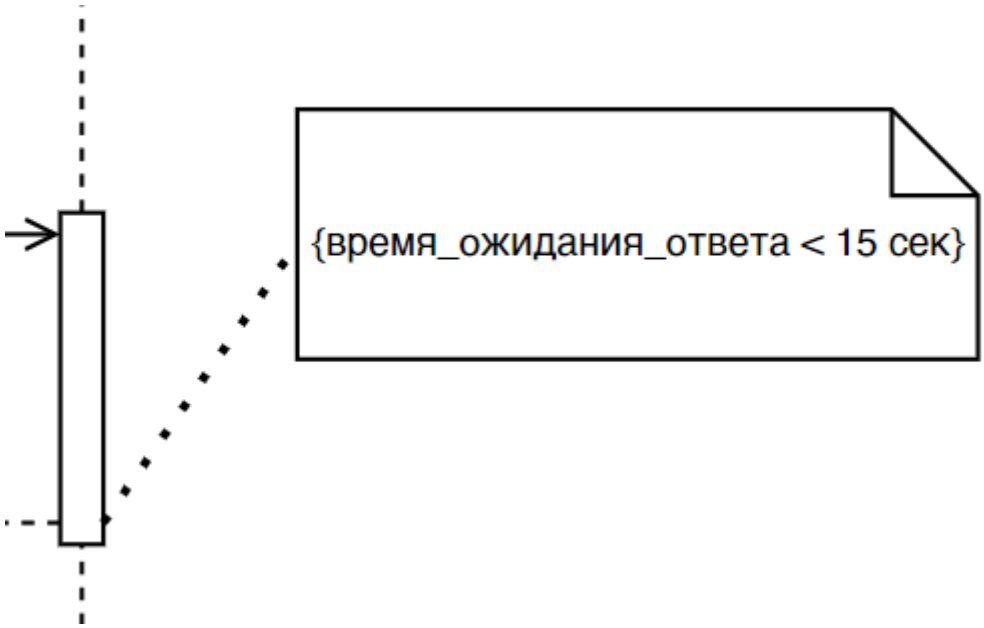



- “call” – вызов процедуры объекта-сервера
- “return” – сообщение, которое возвращает значение завершённой процедуры
- “create” – сообщение, которое требует создания нового объекта
- “destroy” – сообщение, которое требует уничтожения соответствующего объекта
- “send” – отправить принимающему объекту некий сигнал. Отличие сигнала от сообщения в том, что сигнал явно определён в классе отправителе.

Сообщения также могут иметь и указания конкретных операций, которые они инициируют. Тогда указывается имя операции и её аргументы в скобках.

Бывает случаи, когда надо указать временные ограничения при отправке сообщений. Либо на выполнение операции, либо на саму отправку сообщения. Указываются в специальном объекте. Обычно синтаксис такой:

```
{[имя_объекта.]название_ограничения < время}
```




# Диаграммы кооперации

Используются для отображения особенностей взаимодействия объектов. Но в отличие от диаграмм последовательностей, используются для изображения структурного аспекта взаимодействий.

## Базовые понятия

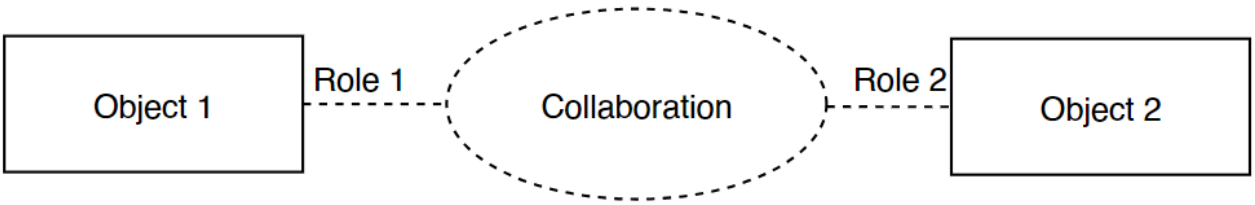
**Кооперация (collaboration) –** это несколько объектов, взаимодействующих с определённой целью.

Кооперация имеет два уровня отображения:

- Уровень спецификации — отображаются роли классификаторов и ассоциаций, существующих в рамках определённого взаимодействия. Используются классы и ассоциации.
- Уровень примеров — это экземпляры и связи, которые образуют роли в рамках кооперации. Используются объекты и связи.

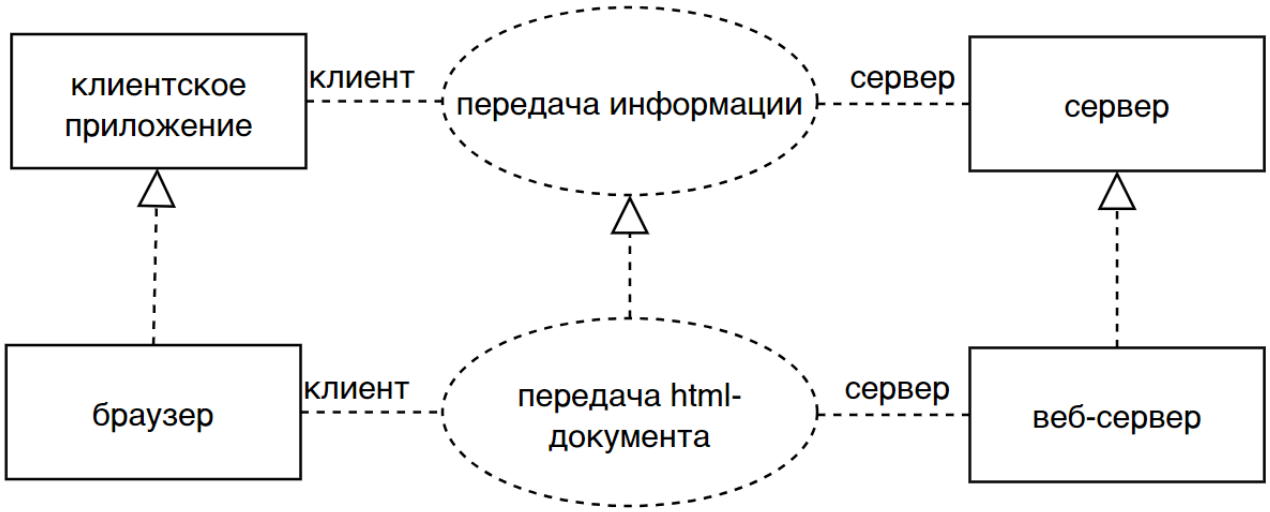
Кооперация уровня спецификации отображается в виде пунктирного эллипса.

Она связана с конкретным вариантом использования и описывает детали его реализации.



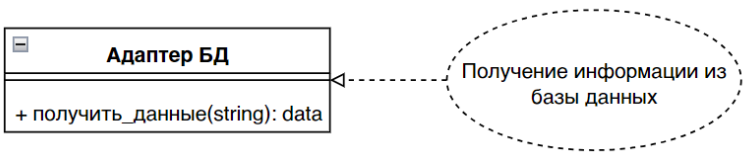
Участники кооперации соединяются пунктирными линиями. В качестве участников могут выступать, как объекты, так и классы. Возле соединительной линии нужно указывать роли (соответствуют именам элементов в контексте взаимодействия).

Текст внутри прямоугольника называется ролью классификатора. Как правило, это только имя класса, но можно указывать секции атрибутов и операций.



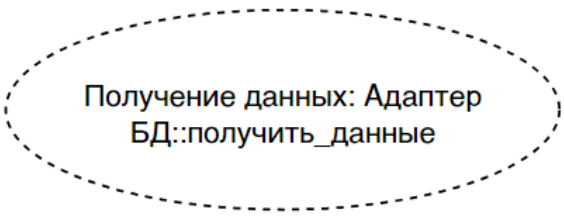

Если кооперация подразумевает наличие обобщённого представления другой кооперации, то есть является частным случаем, это отображается отношением обобщения.

Если надо явно отобразить, что кооперация является реализацией некоей операции или классификатора, есть два способа:




1. Соединить кооперацию пунктирной линией с соответствующим классом
2. Указать две точки после имени кооперации, затем спецификацию того, реализацией какой операции какого класса является кооперация

```
имя_кооперации:
имя_классификатора::имя_операции_классификатора
```




Описание кооперацией начинается с уровня спецификации, на которой указываются обобщения и реализации. Затем, каждая кооперация может быть описана на уровне примеров, которые раскрывают особенности внутренней структуры.

На диаграмме кооперации уровнея спецификации могут присутствовать именованные классы, с указанием роли класса, а также анонимные классы, у которых указана только роль.

**Объекты —** понимается отдельный экземпляр класса. Графически изображается, как прямоугольник с подчёркнутым текстом — это отличает его от классов визуально.

`имя_объекта / имя_роли_классификатора:имя_классификатора`

Имя роли опускается, если существует всего одна роль, в которой могут выступать объекты данного классификатора.

Возможные варианты сигнатур имени объекта:

:А — Анонимный объект, экземпляр А

/Роль — Анонимный объект, исполняющий Роль

/Роль:А — Анонимный объект, экземпляр А, исполняющий Роль

А/Роль — Объект с именем А, играющий Роль

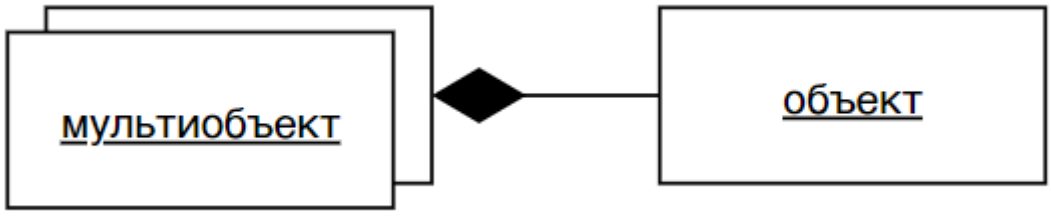
А:Б — Объект с именем А, экземпляр класса Б

А/Роль:Б — Объект с именем А, экземпляр класса Б

А — Объект с именем А

А: — Объект неопредёлонного класса с именем А

**Мультиобъект —** это множество объектов. Используют, чтобы показать отправку сигнала всем объектам в множестве.




Если объект входит в множество, это можно показать отношением композиции.

### *Активные объекты*

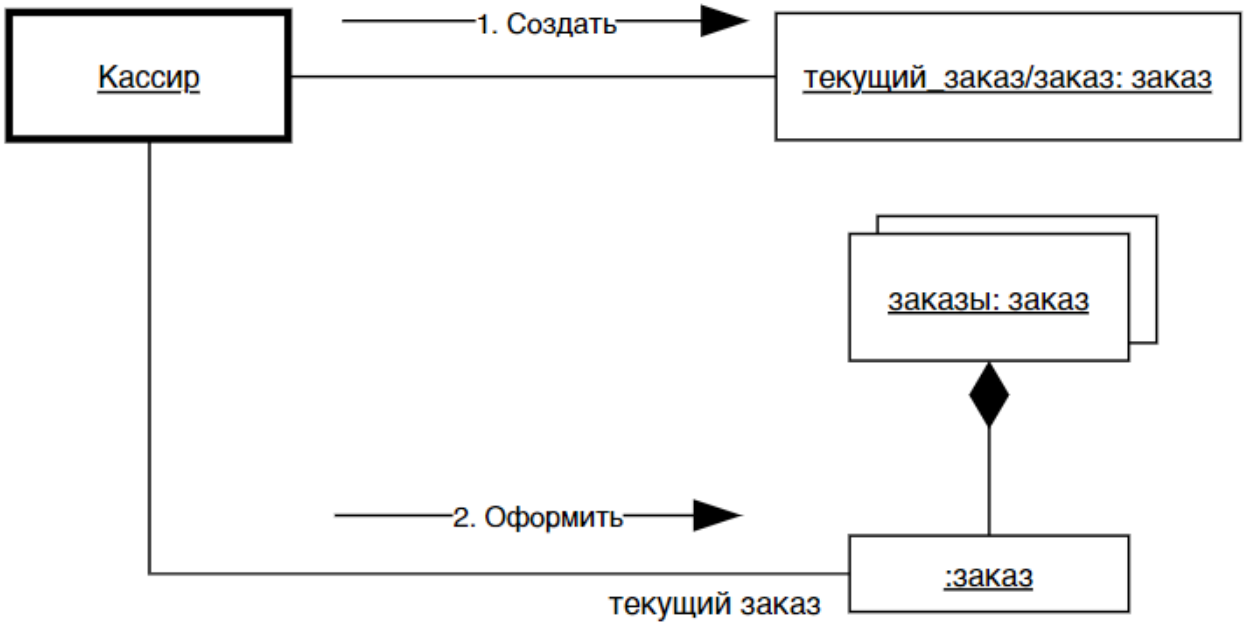
Все объекты в рамках UML можно разделить на **активные** и **пассивные**.

Пассивные выполняют операции только над данными и не могут выполнять их над другими объектами. Они имеют возможность отправлять сигналы другим объектам.

Каждый активный объект имеет нить или поток управления, и может управлять другими объектами. При этом предполагается, что поток управления выполняется параллельно с другими потоками управления в рамках процесса.

Активные объекты обычно изображаются с широкими краями, иногда маркируются как `[active]`.

Активный объект может инициировать один поток управления и представляет собой исходную точку этого потока. Потоки могут быть пронумерованы в порядке создания.

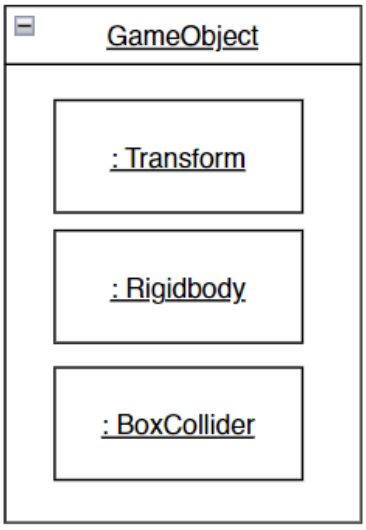



## Составные объекты

Также называются объекты-контейнеры.

Отображают объекты, которые имеют собственную структуру.

Это экземпляр составного класса, то есть связанного отношением композиции или агрегации со своими компонентными частями. Графически, указывается как обычный объект, только вместо атрибутов указаны прямоугольники его компонентных классов.

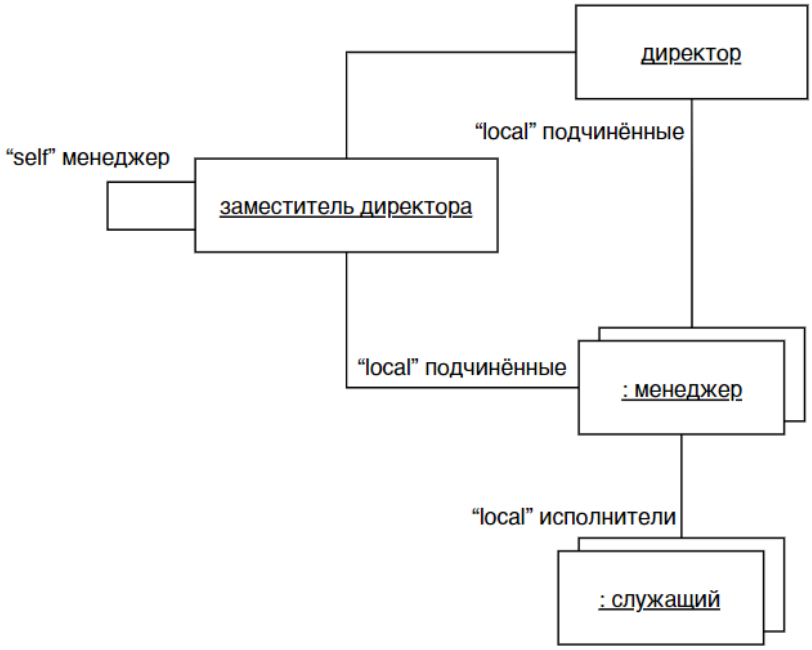



## Связи

Ассоциации тоже могут иметь экземпляр. Экземпляр ассоциации называется **связь (link).** Может существовать между двумя и более объектами. Бинарная связь (между двумя объектами) отображается как сплошная линия. На обоих концах связи могут быть указаны роли объектов, а посередине линии — имя данной ассоциации.

Связи не имеют имён и кратности. Для отдельных случаев, могут быть обозначены.

**Стереотипы связей**




Указывают на особенности реализации связи.

- “association” (ассоциация) — не указывается, так как подразумевается по умолчанию.
- “parameter” (параметр метода) — соответствующий объект выступает только в роли параметра некоего метода.
- “local” (локальная переменная) — область видимости переменной ограничена только методом соседнего объекта
- “global” (глобальная переменная) — область видимость распространяется на всю диаграмму кооперации
- “self” (рефлексивная связь) — объект связан сам с собой. Графически отображается петлёй

## Сообщения

На диаграмме кооперации сообщения указывают, что объекты передают друг другу информацию. Предполагается, что при получении сообщения объект будет выполнять какое-то действие.

Отображается стрелкой от объекта-отправителя к объекту-получателю.

1. ——▶ Используется, когда сообщение ининициирует вызов процедуры или потока исполнения. Как правило, синхронны.
2. ——> Используется для пустых потоков управления. Один отдельный «шаг» в последовательности потока управления. Чаще всего асинхронны.

3. ——\ Используется, когда сообщение инициирует асинхронный поток управления. Сообщение данного типа отправляется в произвольный момент времени, который изначально не известен, и чаще всего активными объектами. Чаще всего инициируется актёрами и являются стартовыми в потоке управления.

4. - - - -> Используется для обозначения возврата из вызова процедуры. Как правило, не изображаются, потому что наличие такого сообщения подразумевается из вызова процедуры.

### *Формат записи сообщений*

```
предшествующее_сообщение[сторожевое_условие]
выражение_последовательности

возвращаемое_значение имя_сообщения список_аргументов
```

Предшествующие сообщения выражаются в виде списка из разделённых запятой номеров сообщений, указанный перед наклонной чертой. Указывается потому что данное сообщение может быть передано только когда будут выполнены указанные в списке сообщения.

Сторожевое условие — это логическое выражение, которое предназначено для синхронизации отдельных потоков управления.

Выражение последовательности — это список терминов последовательности, разделённый точками, после него идёт двоеточие

Каждый термин записывается так:

[целое_число|имя][символ_рекурентности]

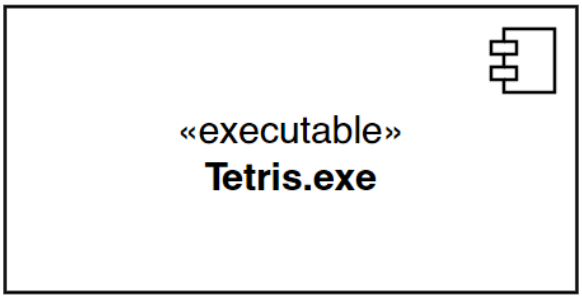
Символ рекурентности просто указывает выполнять условно или итеративно.

# Диаграмма компонентов

Позволяет показать структуру приложения начиная от объектов и классов  до исполняемых файлов и библиотек.

## Базовые понятия

- Компоненты — основной строительный блок в диаграмме
- Интерфейсы — это интерфейсы доступа к компонентам, также помогает отобразить необходимость в классе, поддерживающим специфичный интерфейс.
- Зависимости — показывают когда один объект использует другой. Также называются связью.

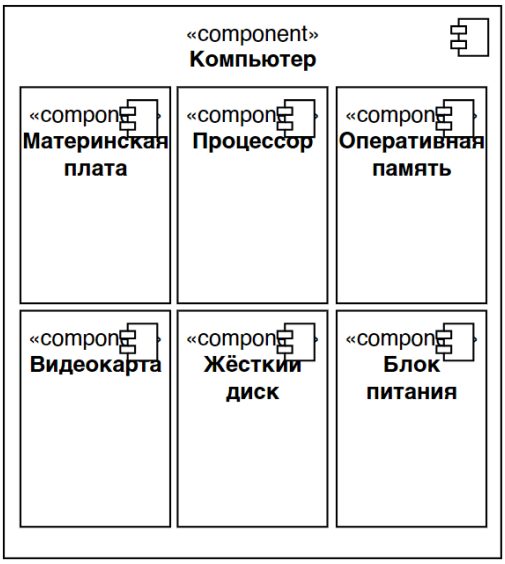


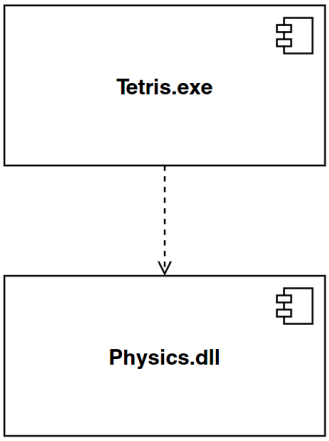

В кавычках указан тип, а под ними — имя.

Иногда вместо имени компонента не пишут ничего или пишут Component.

- Скомпилированный код записывают названием файла с его расширением: Tertris.exe, MessageWindows.dll
- Файлы — когда приложение использует их, как внешнее хранилище, также пишутся с расширением: Config.ini, PlayerData.xml
- Страницы — также имеют расширение: Index.php, Main.html
- Классы — можно подписать слово *class* перед названием класса: class Dispatcher
- Объекты классов — сначала пишется название объекта, затем через двоеточие — его тип: GameManager: Singleton

Компоненты могут находиться отдельно, а могут включаться друг в друга.



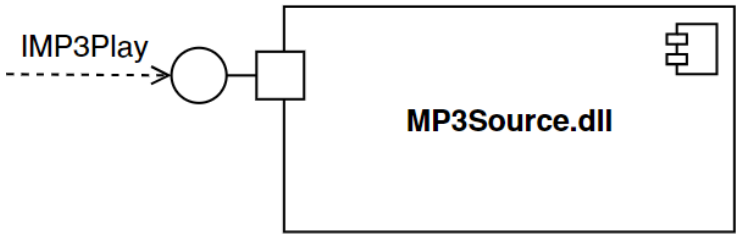




Зависимость рисуется пунктирной линией от зависимого объекта к не зависимому.

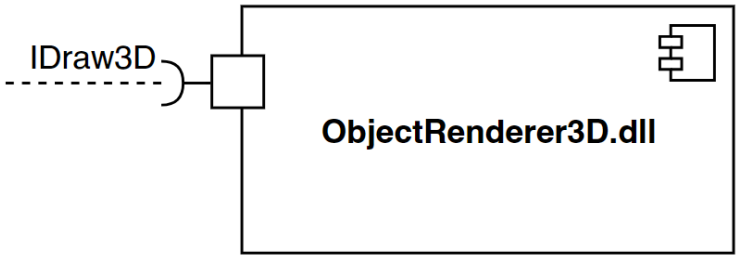
Не рисуется, когда независимый компонент является частью зависимого.

Иногда могут быть подписаны для большей ясности.

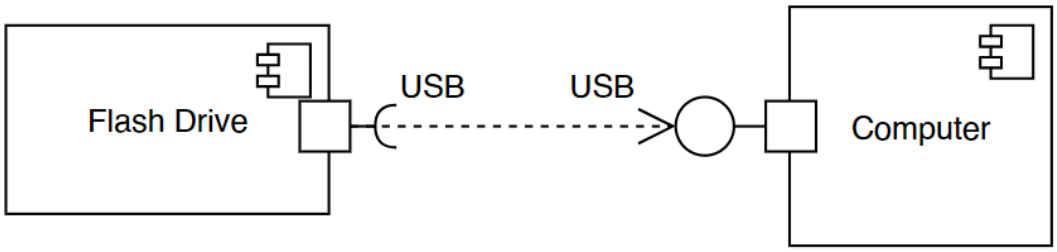
Интерфейсы — это часть диаграммы компонентов, которая показывает, что объект предоставляет для доступа к себе интерфейс.




Когда объект требует для работы другой объект, поддерживающий интерфейс, то изображение другое.




Таким образом, часто зависимость ободоюдная.




# Диаграмма развёртывания

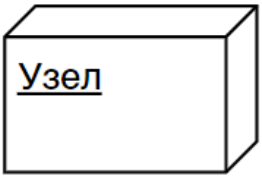
Используется для физического представления развёртываемой системы. Какие физические устройства и связи между ними нужны приложению.

Иногда может быть заменена диаграммой компонентов.

Основная цель — найти слабые или не защищённые места в системе.

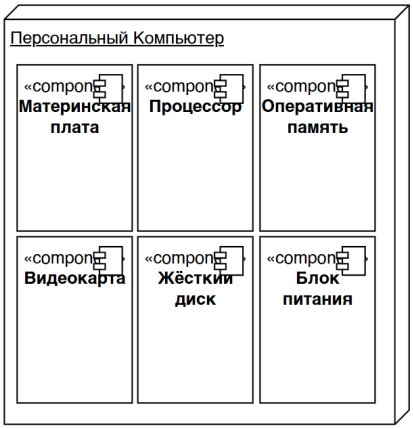
## Базовые понятия

**Узлы** и **соединения.**

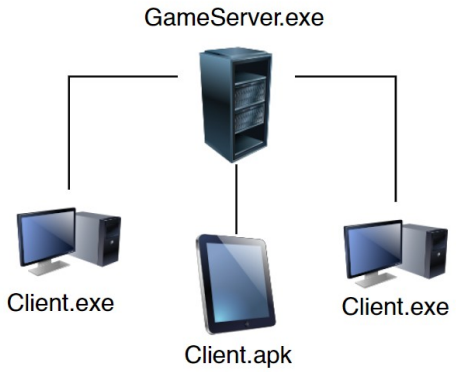



**Узел —** это вычислительная единица — это может быть устройство с процессором и электронной памятью. Изображается, как куб.

Может быть соединён с диаграммой компонентов, она рисуется на передней части куба.




Различные среды рисования UML предоставляют различные изображения для отображения узлов.




Узлом может являться человек, что используется в бизнес-процессах.

Как правило соединения — это простые линии без стрелок. Указывают просто, что объекты имеют друг к другу отношение. Допускается подписывание линий.