UML – быстрый старт

Михаил Смирнов
www.msmirnov.ru

Издание 2-е, оформленное

Содержание

1.Введение

UML – это **Unified Modeling Language**, как следует из названия – унифицированный язык моделирования. **UML** представляет собой набор соглашений, которые предназначены для облегчения процесса моделирования и обмена информацией в проектной группе. Наличие стандартизированной нотации позволяет сократить время на усвоение информации, упрощает общение и взаимодействие, облегчает документирование.

В этом документе описаны самые основные разделы языка **UML**, которые требуются в повседневной работе.

Разработан документ на основе моего опыта обучения сотрудников основам языка **UML** и представляет собой краткое пособие для изучающих **UML** самостоятельно или быстрой актуализации ранее полученных знаний.

По любым интересующим вас вопросам обращайтесь:

- www.msmirnov.ru
- e-mail: msmirnov@msmirnov.ru
- Skype: michael_e_smirnov
- Блог: http://michaelsmirnov.blogspot.ru
- Facebook: https://www.facebook.com/michaelesmirnov
- LinkedIn: https://www.linkedin.com/in/msmirnov
- YouTube: https://www.youtube.com/user/MichaelSmirnov123

Михаил Смирнов, 2016 г.

2. Основы

UML представляет собой графическую нотацию, которая предназначена для моделирования и описания всех процессов, протекающих в процессе разработки. Основу **UML** представляют диаграммы, которые различаются по типам и предназначены для моделирования различных аспектов разработки.

Все диаграммы можно условно разделить на **поведенческие** и **структурные**. Поведенческие диаграммы отображают процессы, протекающие в моделируемой среде. Структурные диаграммы отображают элементы, из которых состоит система. При этом одни и те же типы диаграмм могут использоваться как для моделирования бизнес-процессов, так и для непосредственного проектирования архитектуры.

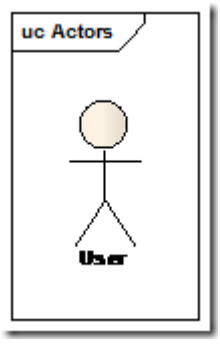
3. Типы диаграмм

3.1 Диаграмма вариантов использования (Use-case diagram)

Диаграмма вариантов использования является отправной точкой в процессе моделирования. Она предназначена для описания взаимодействия проектируемой системы с любыми внешними или внутренними объектами - пользователями, другими системами и т.п.

Основными понятиями при работе с диаграммой вариантов использования являются **Актор** (Actor) и **Вариант использования** (Use case).
Актор – это роль, которую выполняет пользователь или другая система, при взаимодействии с проектируемой системой.

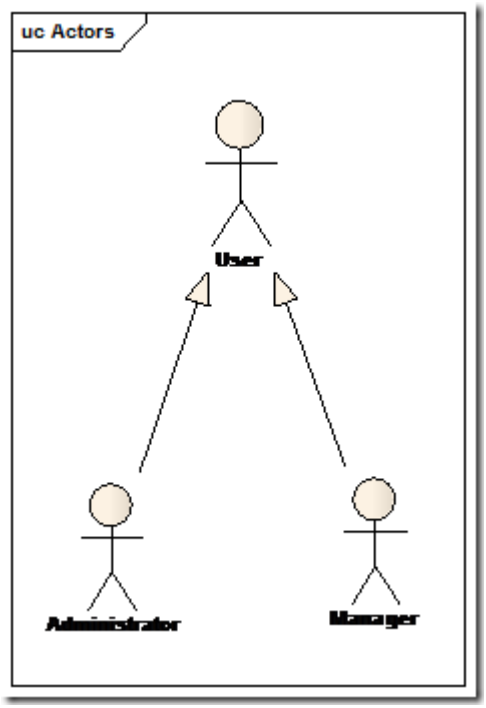
Проектирование диаграммы вариантов использования начинается с определения списка Акторов. На диаграммах Актор обозначается следующим значком:



Каждый Актор обладает уникальным именем.

Друг с другом акторы могут быть связаны различного рода отношениями.

Например, акторы могут наследоваться друг от друга.

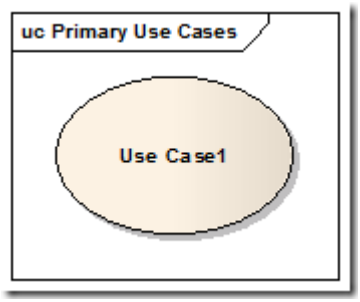


Это означает, что акторы-наследники наследуют характеристики базовых акторов.

Следующим этапом после определения списка акторов является определение списка вариантов использования.

Вариант использования – это конечная единица взаимодействия актора и системы. Совокупность всех вариантов использования полностью определяет поведение системы.

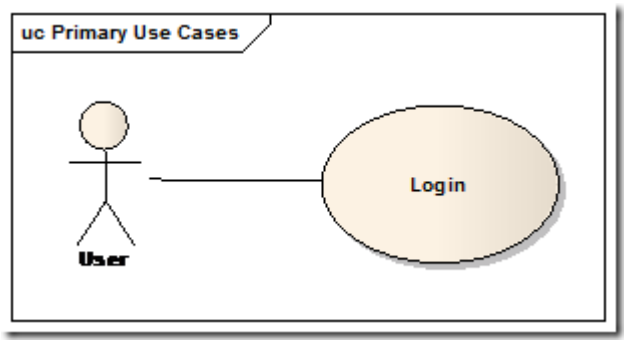
Вариант использования обозначается значком:



Каждый вариант использования относится к каком-либо актору.

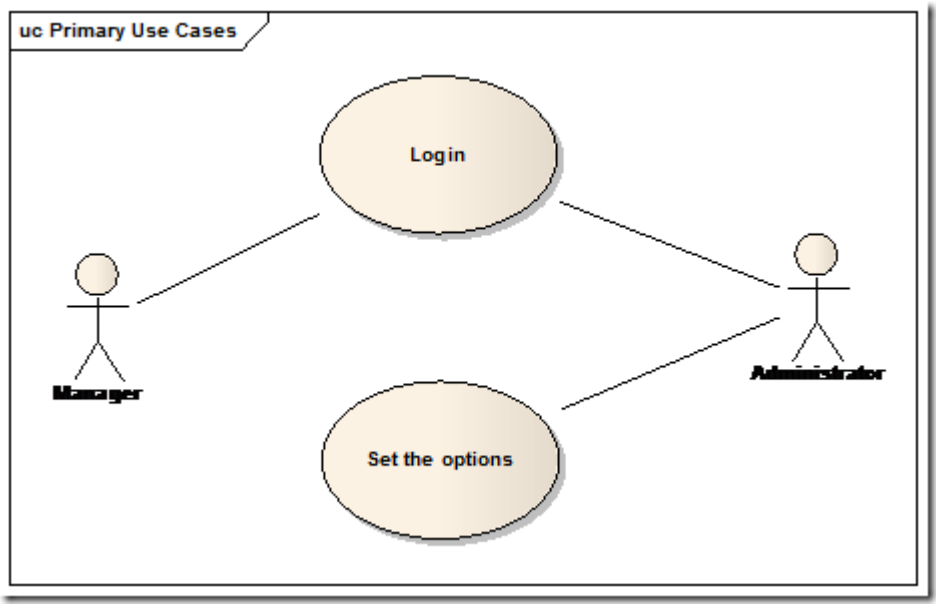
Такое отношение обозначает, что данный актор инициирует данный вариант использования.

Например:



Это означает, что актор User инициирует вариант использования Login.

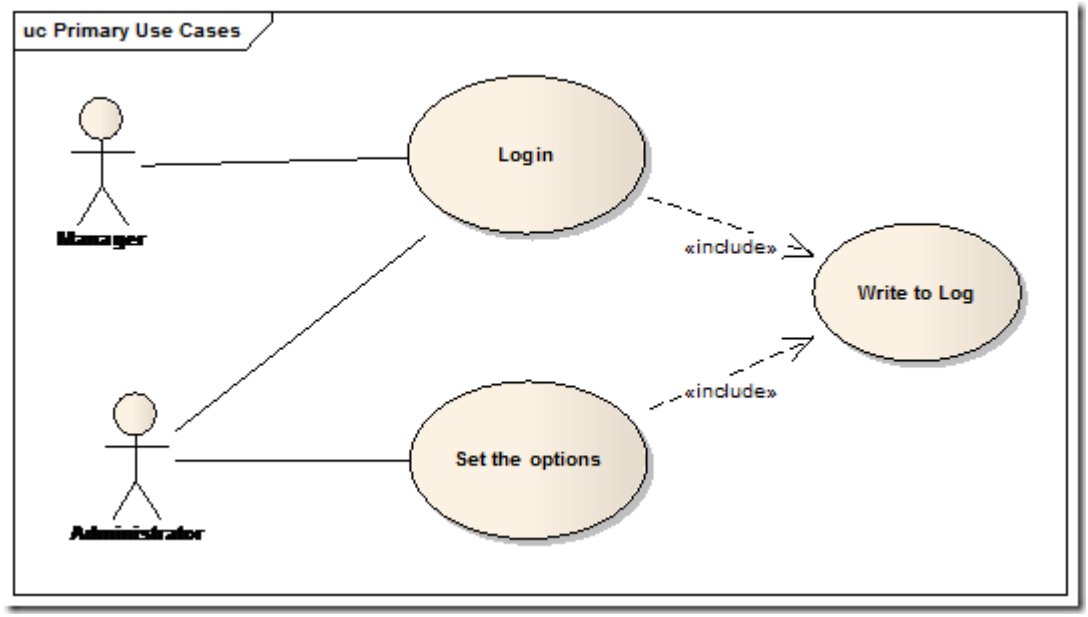
Один и тот же вариант использования может использоваться несколькими акторами, например:



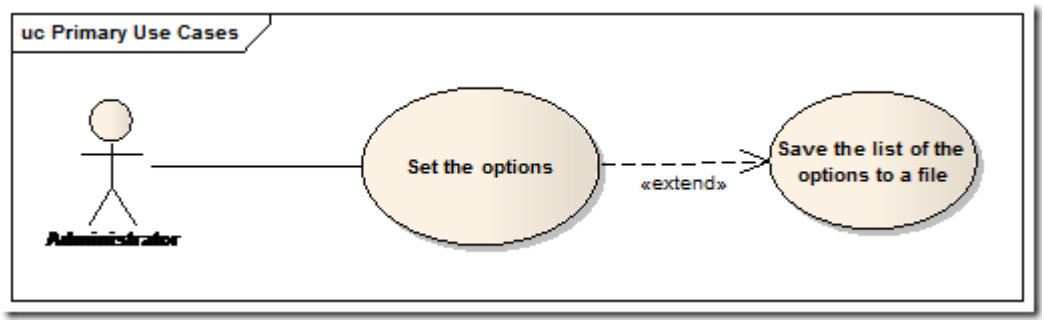
Здесь вариант использования Login используется двумя акторами.

Варианты использования также могут быть связаны друг с другом различными отношениями.

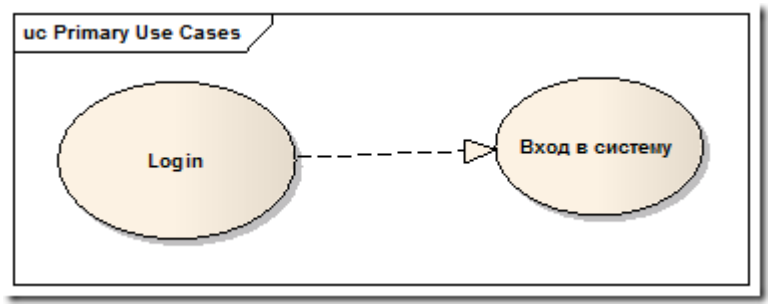
1. «**Включение**» одного варианта использования в другой. Означает, что один вариант использования инициируется в процессе другого. Например:



2. «**Расширение**». Означает, что один вариант использования является дополнением или уточнением другого варианта использования в случае наступления некоторых условий. Например:

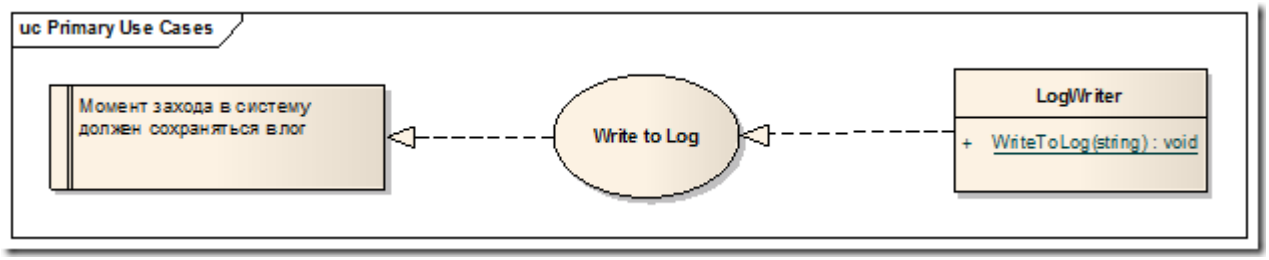


3. «**Реализация**». Означает, что один вариант использования является реализацией другого варианта использования. Например, если один из них описан в терминах бизнес-процессов, а другой – в терминах проектируемой системы. Например:



Кроме того, варианты использования могут быть связаны отношением «Реализация» с требованиями к системе и с классами. При наличии таких связей есть возможность проследить в каких классах реализованы требования и какие классы могут быть затронуты при изменении требований или вариантов использования.

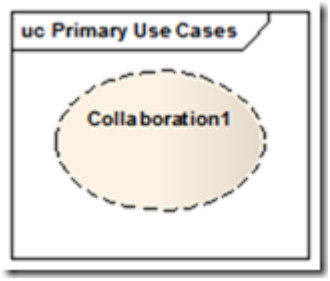
Например:



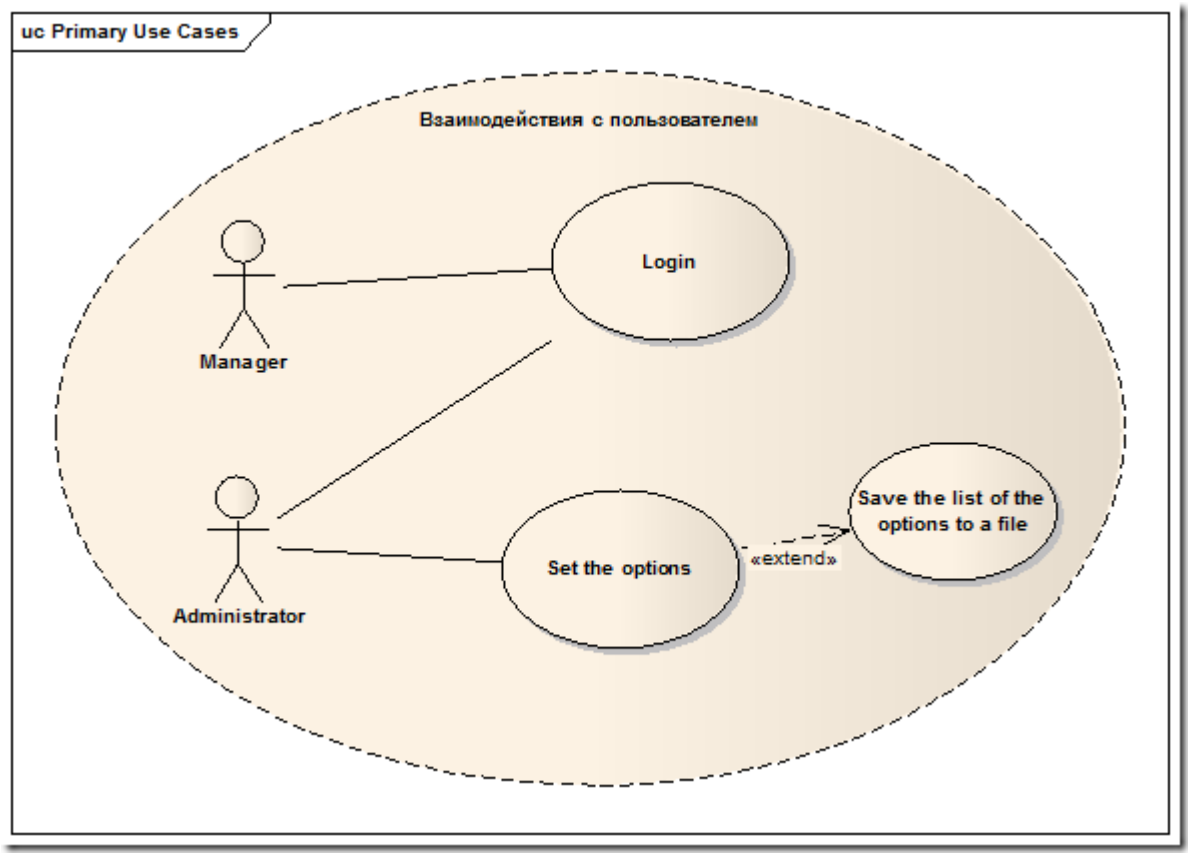
Кроме Акторов и Вариантов использования на диаграмме также могут находиться следующие элементы:

1. “**Collaboration**” – элемент, предназначенный для визуальной группировки объектов – акторов и вариантов использования – по принципу их совместной работы.

   Обозначается значком

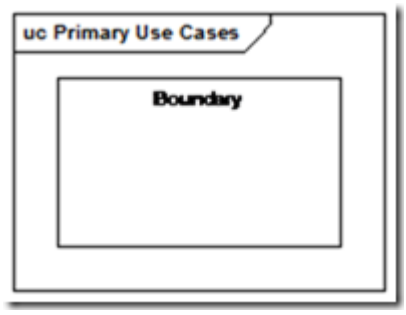


   Например,

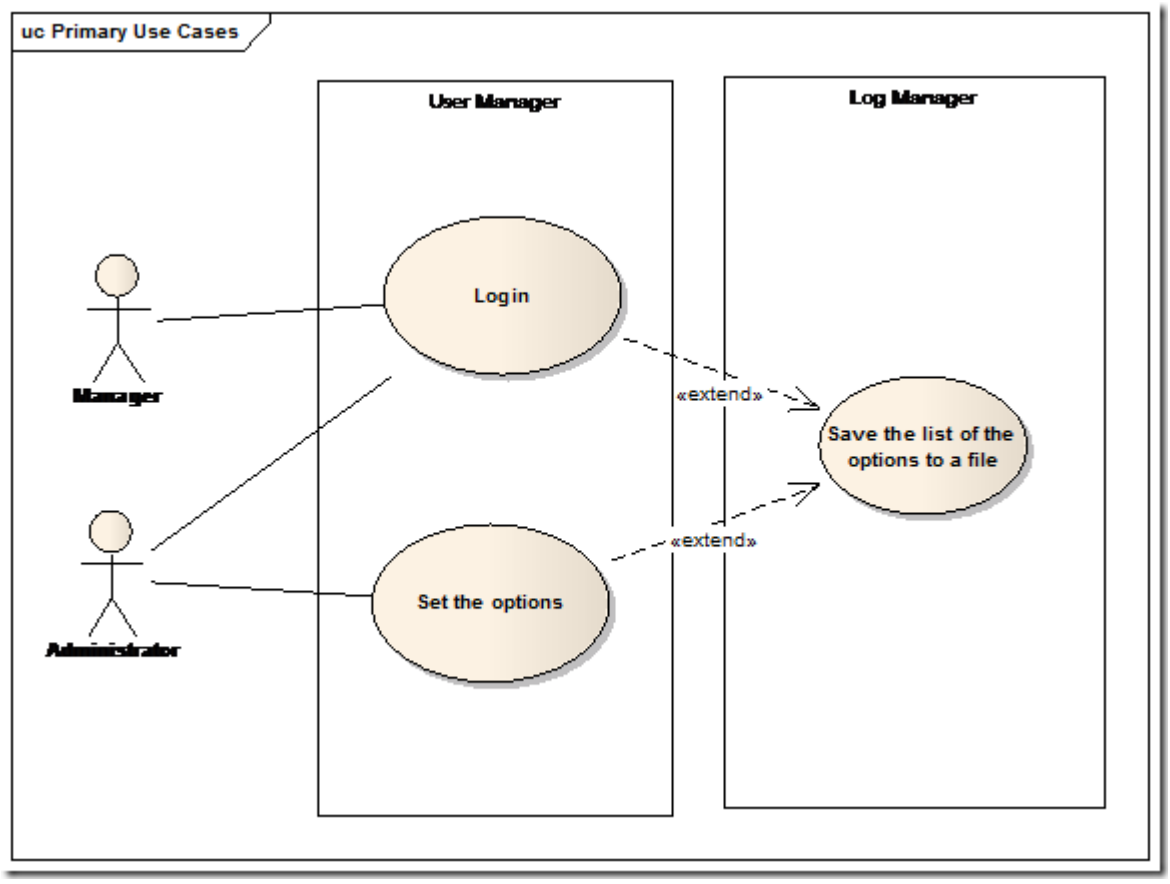


- «**Boundary**» - элемент, предназначенный для визуальной группировки объектов – акторов и вариантов использования – по принципу их распределения на подсистемы или компоненты.

  Обозначается значком

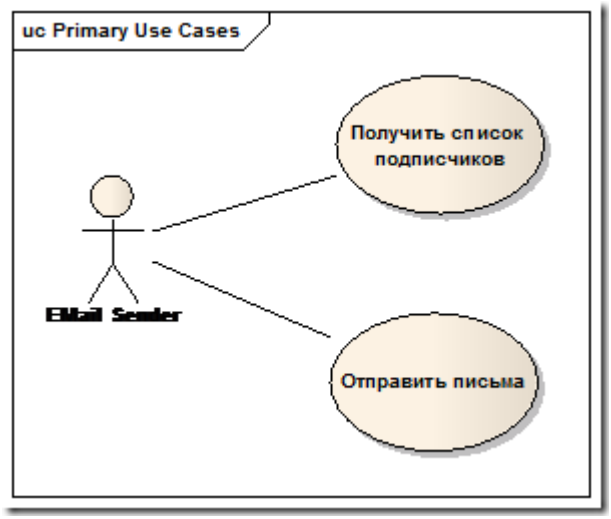


Например:



Среди акторов могут быть не только пользователи, но и внешние системы и внутренние подсистемы.

Пример внутренней подсистемы:



3.2 Диаграмма последовательностей (Sequence diagram)

Диаграмма последовательностей служит основным способом расшифровки последовательности действий в процессе выполнения того или иного варианта использования.

Иными словами, если вариант использования отвечает на вопрос «Что делает актор?», то последовательность отвечает на вопрос «Как работает система при выполнении данного варианта использования?».

Таким образом, диаграмма последовательностей всегда создается в привязке к варианту использования. Каждый вариант использования может содержать несколько диаграмм последовательностей, на тот случай, если они описывают несколько альтернативных вариантов развития событий.

Диаграмма последовательностей, так же, как и вариант использования, может быть реализована как в терминах бизнес-объектов, так и в терминах физических сущностей, таких как компоненты или классы.

Проще всего продемонстрировать суть диаграммы последовательностей на примере:

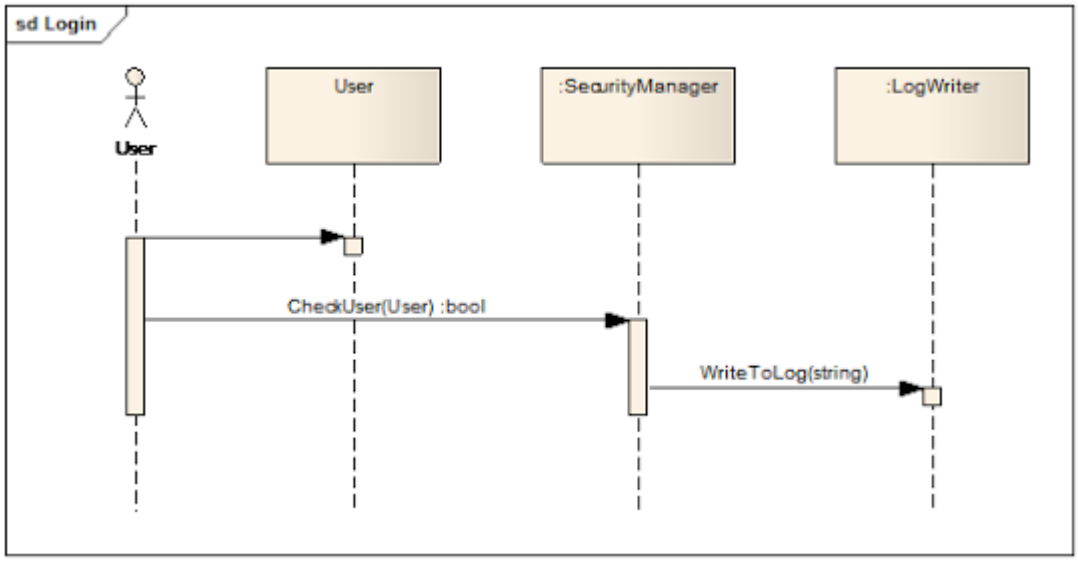


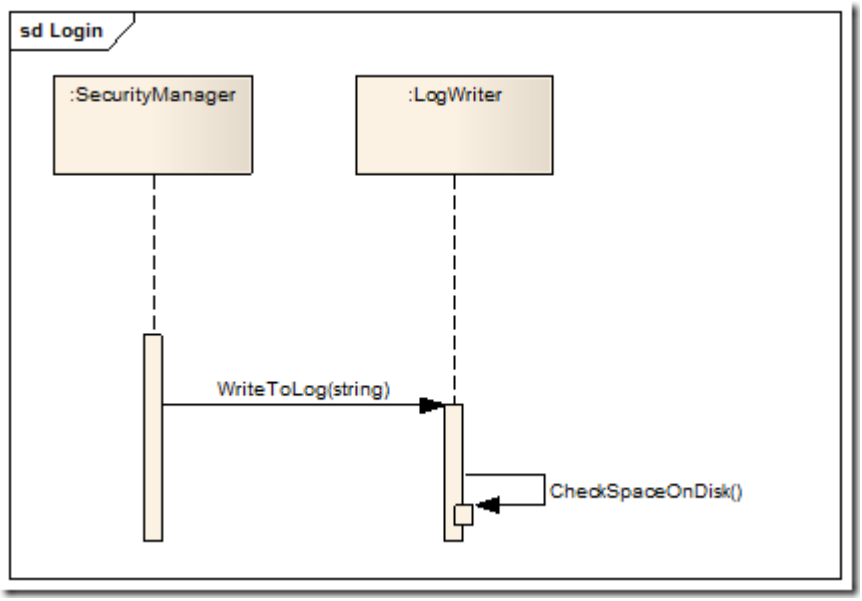
Диаграмма последовательностей всегда начинается с актора, инициирующего процесс. Вверху диаграммы располагаются элементы, классы или компоненты, которые задействованы в процессе работы.

На самой диаграмме показаны линии жизни каждого из объектов и процесс их взаимодействия.

Взаимодействие объектов показано стрелками. В терминах диаграмм последовательностей такое взаимодействия называется **Сообщение** (**Message**) В приведенном примере под сообщением понимается вызов методов тех или иных классов. При этом, как видно на примере, сообщения могут быть вложены друг в друга, что означает, что один метод вызывается в теле другого.

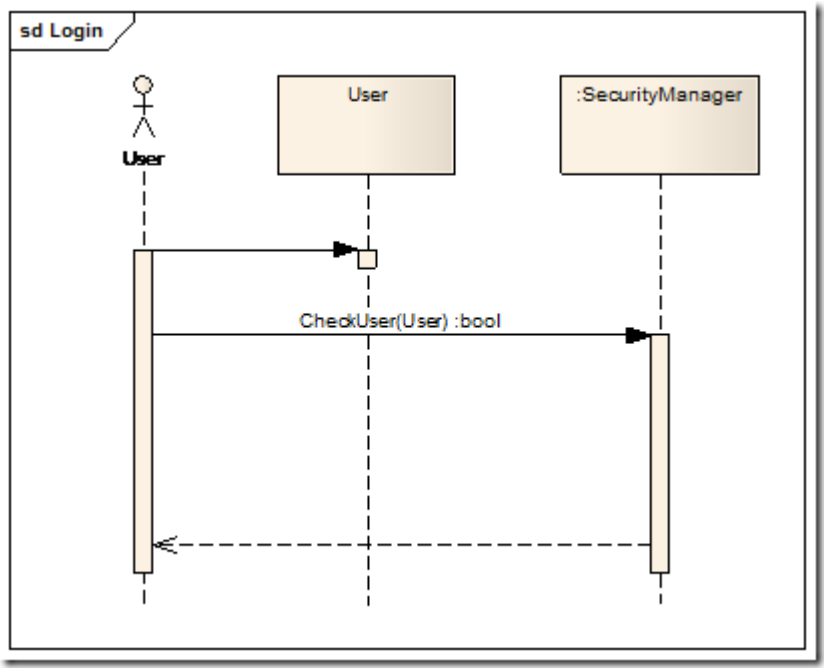
Кроме сообщений, которые вызываются другими объектами, существуют **собственные сообщения**, которые объект вызывает сам у себя (**Self-message**).

Например:



Кроме того, существуют также **обратные сообщения** (**Return message**), которые обозначают передачу некоторой информации вызывающему сообщению. В терминах классов это означает возврат некоторого значения.

Например:

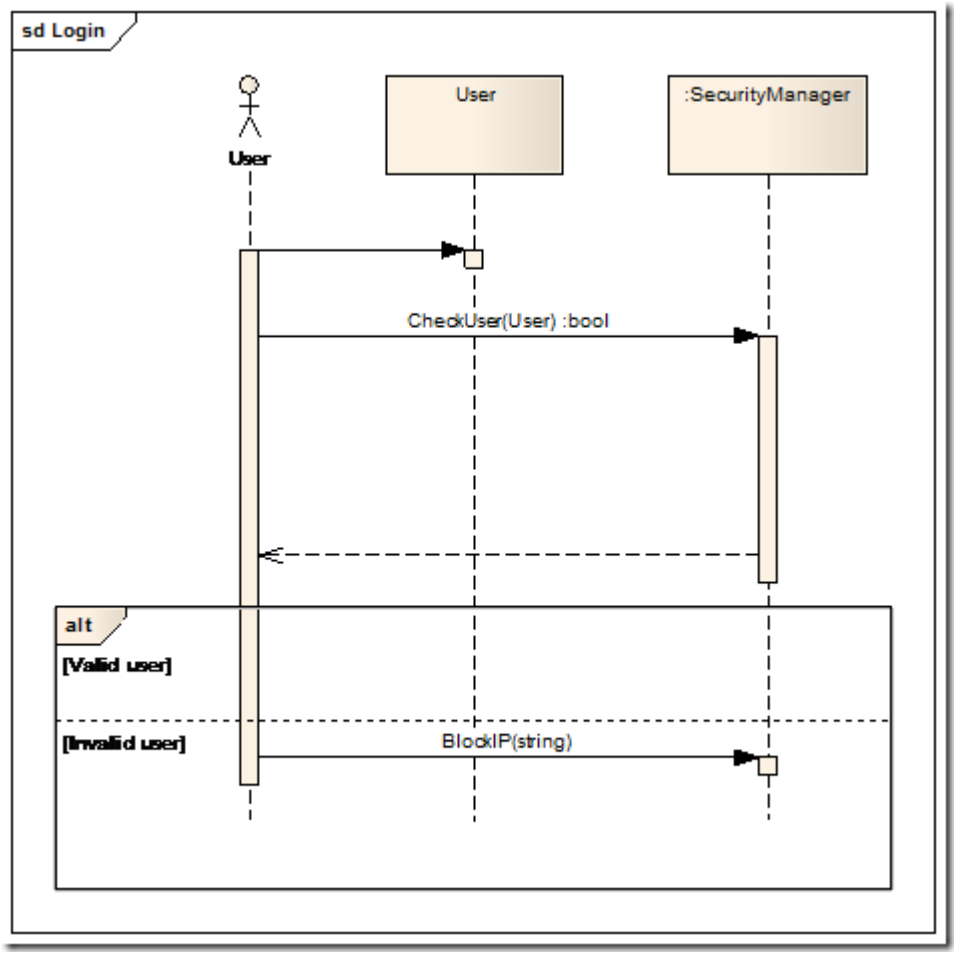


На этой диаграмме метод возвращает результаты своей работы. Собственные сообщения также могут быть обратными.

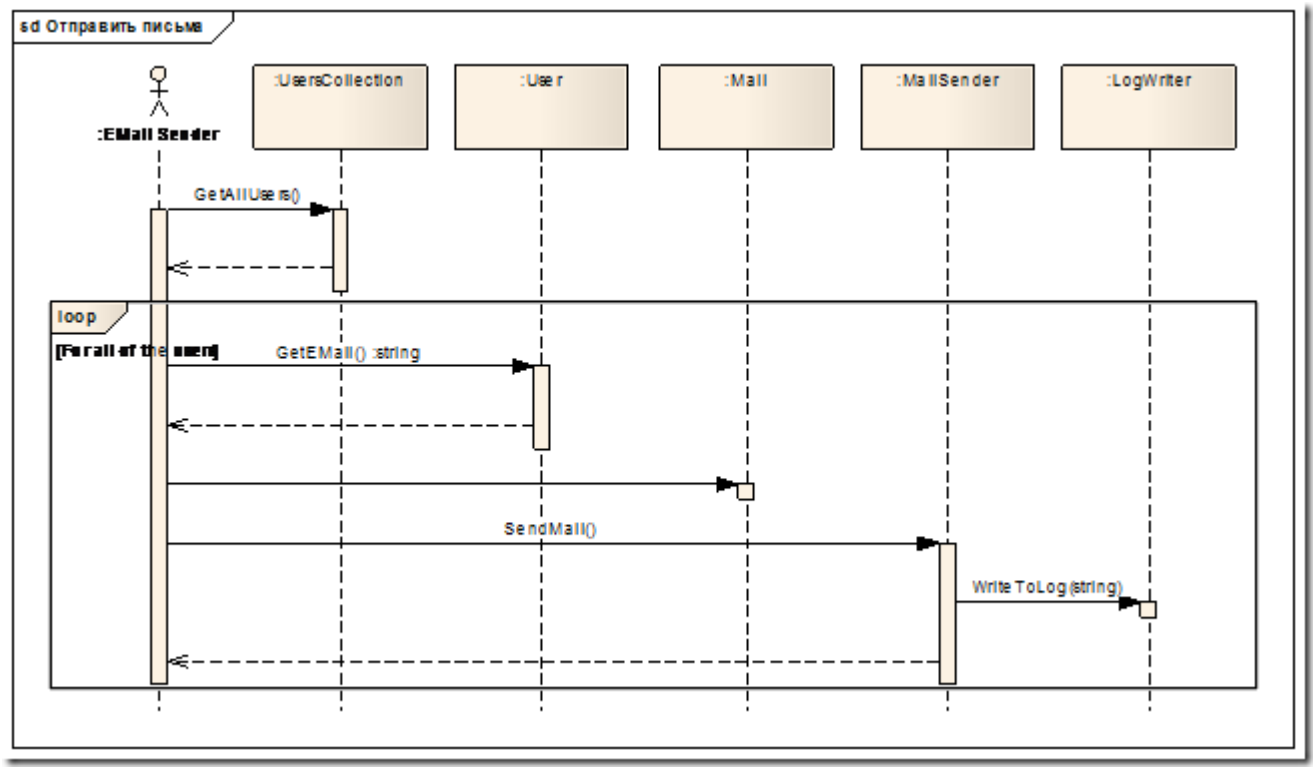
Таким образом, к основным объектам диаграмм последовательностей относятся акторы, объекты-участники процесса, линии жизни и сообщения.

Сообщения могут объединяться в **комбинированные фрагменты** (**Combined fragment**), который предназначен для отображения циклов, ветвлений, критических секций и пр.

Пример ветвления:

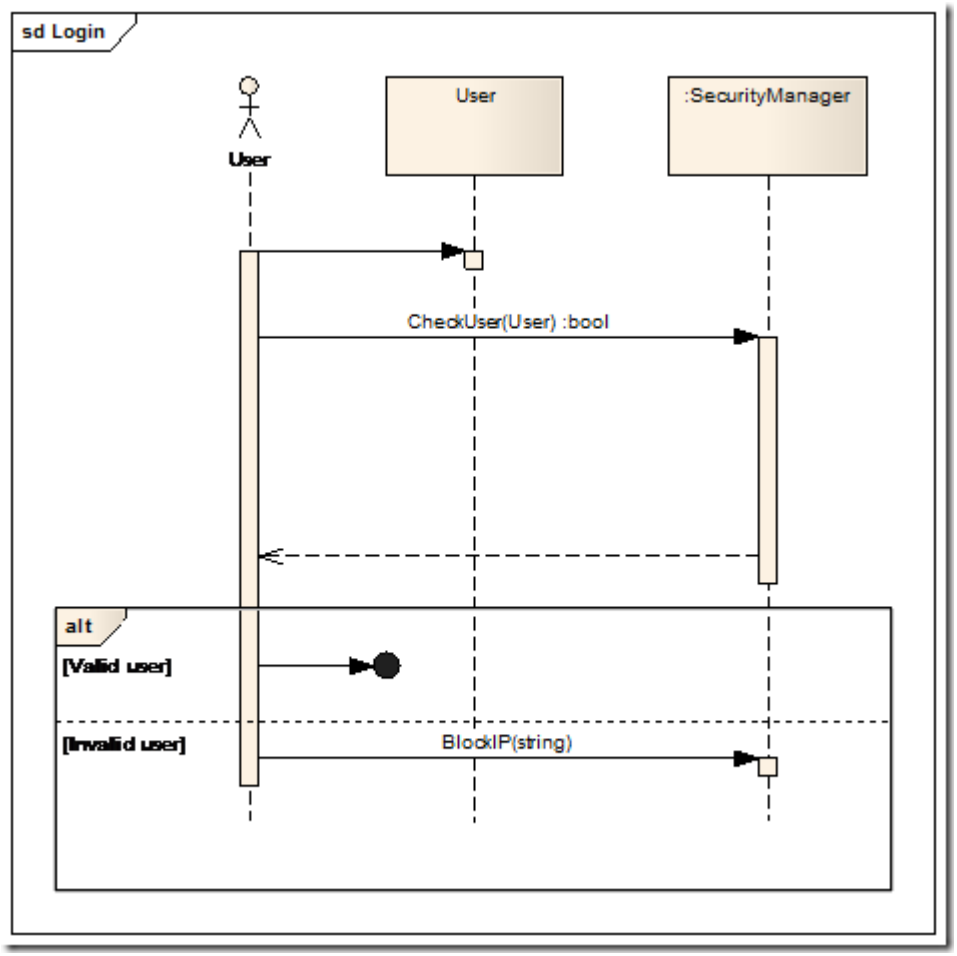


Пример цикла:



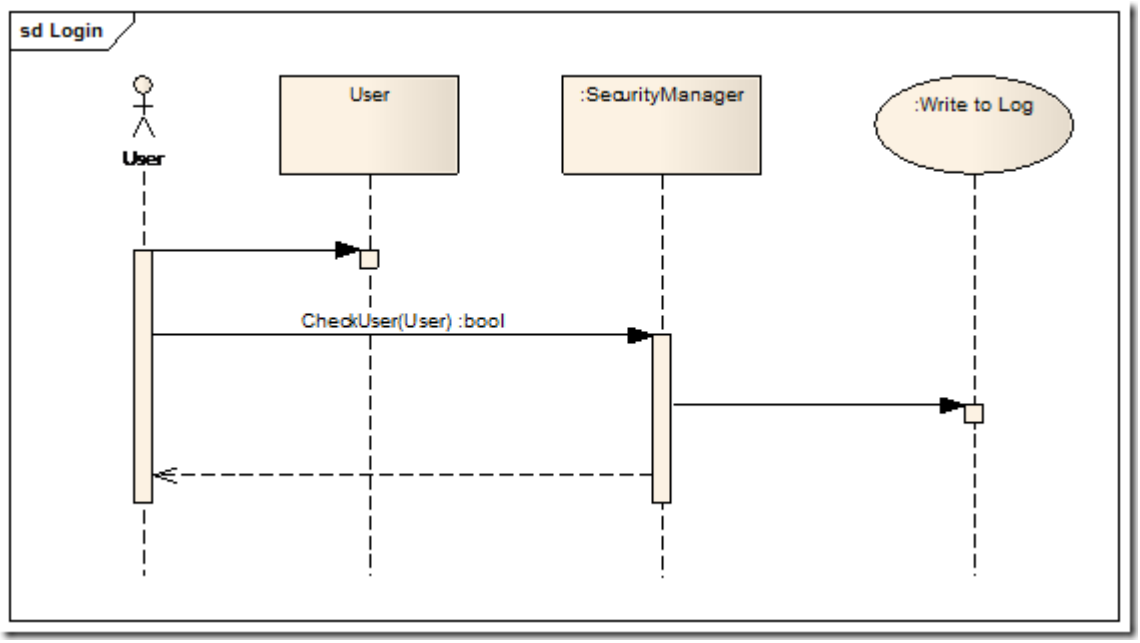
Некоторые сообщения могут заканчиваться Конечными точками (End point), которые означают выход из алгоритма.

Пример:



В качестве участника диаграммы могут выступать не только классы, но и любые другие объекты – например, варианты использования в тех случаях, когда есть необходимость продемонстрировать включение или расширение.

Например:



3.3 Диаграмма классов

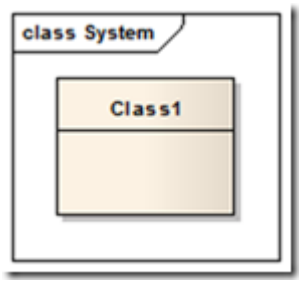
В отличие от двух предыдущих поведенческих диаграмм, диаграмма классов носит структурный характер. Она предназначена для отображения классов разрабатываемого приложения и их взаимосвязей.

Так же, как и предыдущие диаграммы она может быть представлена как в терминах конкретных классов, так и в терминах бизнес-объектов.

Диаграммы классов обычно заполняются параллельно с диаграммами последовательностей в процессе моделирования работы вариантов использования.

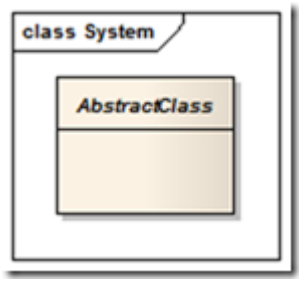
Основным элементом диаграммы классов является класс.

Обозначается значком:

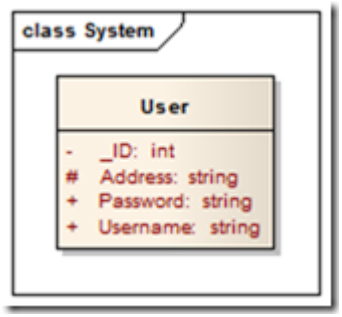


Класс состоит из двух частей – заголовка с именем класса и тела с описанием его полей (**Атрибуты** – в терминах UML) и методов (**Операции** - в терминах UML).

Абстрактные классы отличаются наклонным написанием заголовка:



Под атрибутами класса в терминологии UML понимают его поля. Атрибуты записываются с указанием доступности, имени и типа. Например:



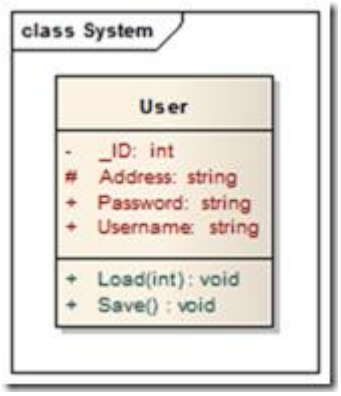
Знак «-» означает, что атрибут является приватным (private).
Знак «+» означает, что атрибут является публичным (public).
Знак «#» означает, что атрибут является защищенным(protected).

После имени следует указание типа атрибута.

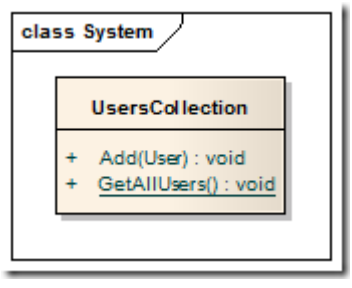
Под **операциями** в терминологии UML понимаются методы, свойства, индексаторы и пр.

Операции также записываются с указанием области видимости, имени и возвращаемого типа. Однако для них также указывается перечень принимаемых значений.

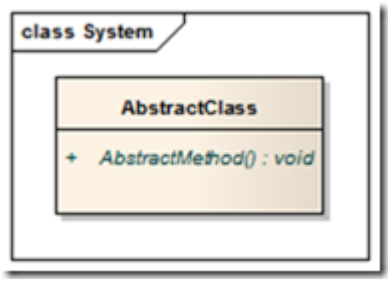
Например:



Статические атрибуты и операции записываются с подчеркиванием, например:

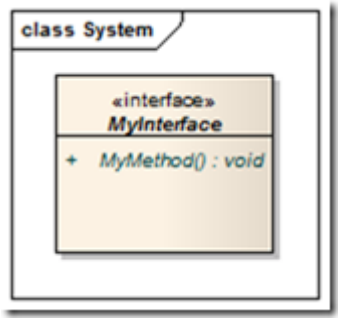


Абстрактные методы записываются наклонным шрифтом, например:



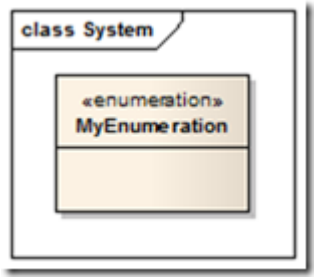
Кроме классов, важным элементом диаграмм классов являются интерфейсы.

Интерфейс обозначается так:

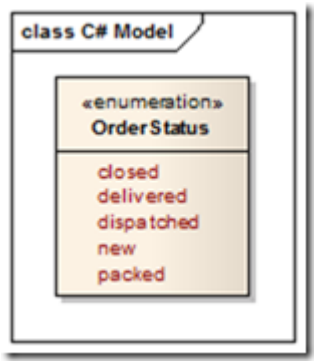


Кроме классов и интерфейсов на диаграмме классов также могут помещаться перечисления.

Перечисление обозначается так:

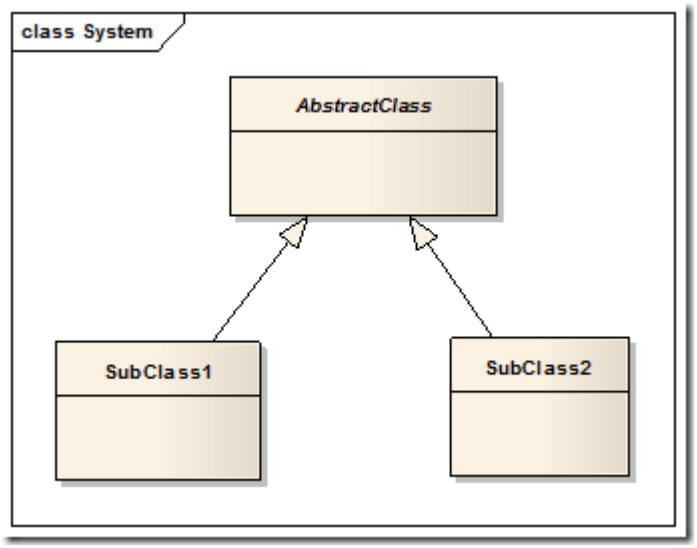


Обычно перечисление указывается с перечнем возможных значений, например:



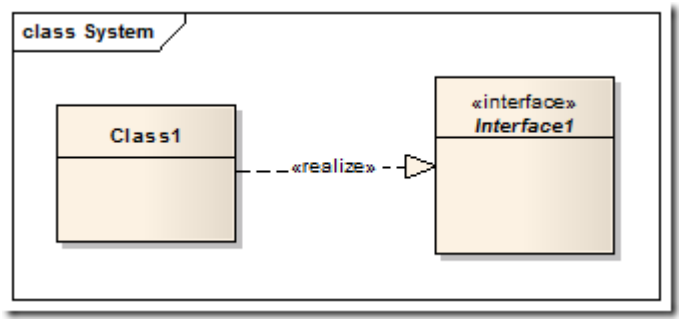
На диаграмме классов отображаются не только классы, интерфейсы и пр., но и их взаимосвязи.
Взаимосвязи бывают различных типов и отображаются, соответственно, по-разному.

Генерализация или **наследование** (**Generalize**) обозначается так:



На примере показано, что два класса являются наследниками абстрактного класса.

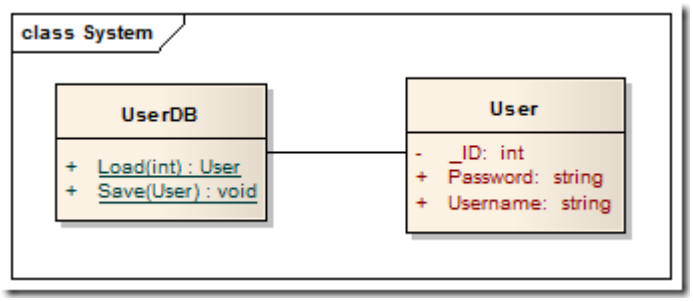
Реализация (**Realize**) – означает, что данный класс реализует данный интерфейс:



Ассоциация (**Associate**). Наиболее широко используемая связь между классами. Она имеет достаточно широкое значение и может означать, например, следующее:

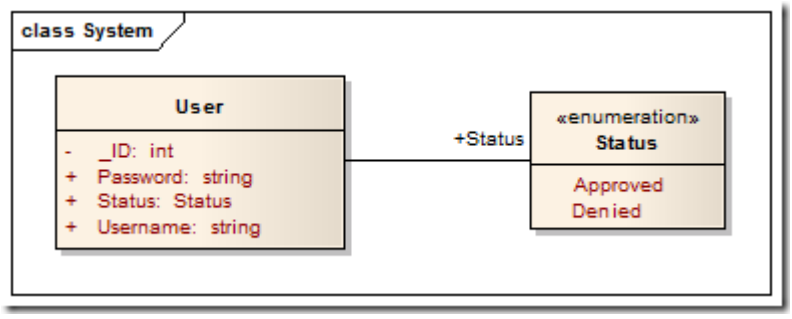
1. Один класс осуществляет взаимодействие с другим каким-либо образом.

   Например:



2. Один класс включает в себя экземпляр другого класса.

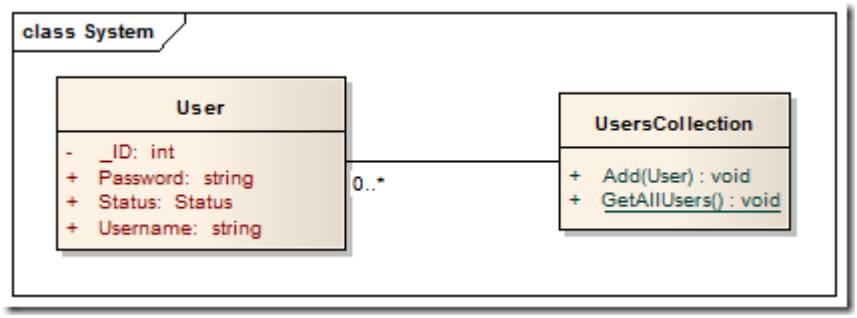
   Например:



Видим, что перечисление Status включено в класс User, при этом имя поля Status, с областью видимости public.

3. Один класс включает в себя несколько экземпляров другого класса.

   Например:



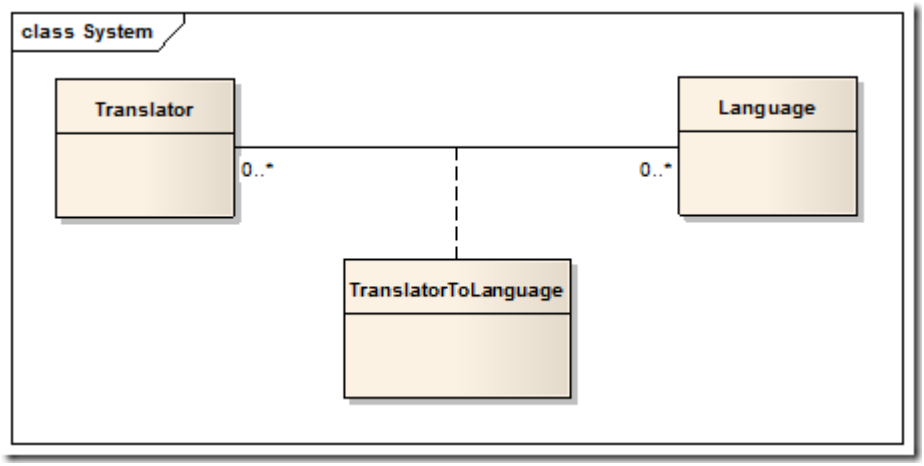
   Видим, что коллекция UsersCollection может включать от нуля до бесконечности объектов User.

   Возможные значения количества объектов:

   - «*» или «0..*» - любое количество объектов
   - «0..n» - любое количество объектов, но не больше n, например «0..5»
   - «n» - точное количество объектов, например «1», «5»
   - «n..*» - любое количество объектов, но не меньше n, например«1..*» - хотя бы один.

Если Ассоциация не может быть реализована без дополнительных классов (например, отношение «многие-ко-многим»), то она реализуется при помощи дополнительных классов ассоциации, которые предназначены решить эту задачу.

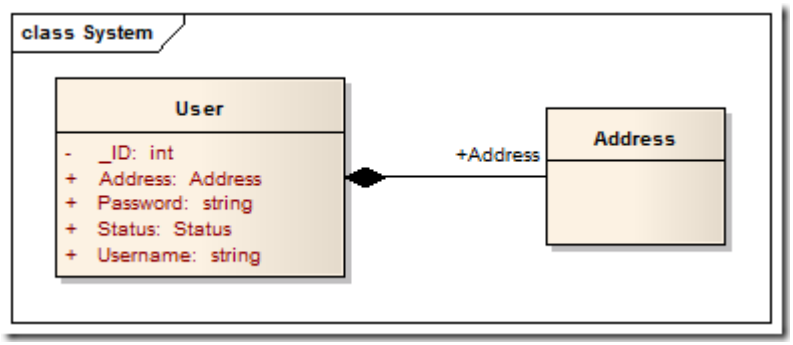
Обозначается это следующим образом:



В данном примере вы видите, что для того, чтобы установить связь «многие-ко-многим» между переводчиками (класс Translator) и языками (класс Language) используется вспомогательный класс TranslatorToLanguage.

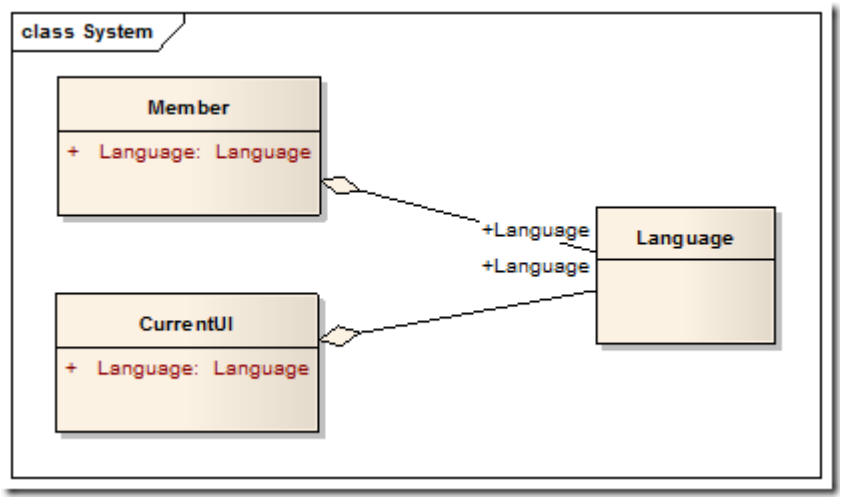
Композиция (**Compose**) - означает, что объекты одного класса могут быть включены в объекты другого класса и при этом этот вложенный объект может находиться только в одном объекте-контейнере. Если объект контейнер удаляется, то вложенный объект тоже удаляется.

Например:



Агрегация (**Aggregate**) - означает, что объекты одного класса могут быть включены в объекты другого класса и при этом этот вложенный объект может находиться в нескольких объектах-контейнерах. Если объект контейнер удаляется, то вложенный объект не удаляется.

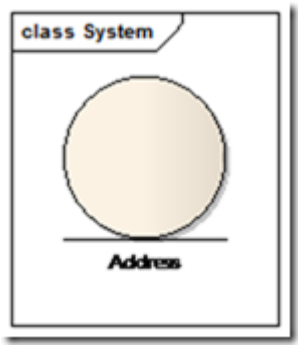
Например:



Для большей наглядности кроме стандартного значка класса, есть также дополнительные значки, которые применяются для некоторых типов классов. Я привел здесь три наиболее часто используемых обозначения, которые применяются для работы с шаблоном «Модель-Представление-Контроллер»

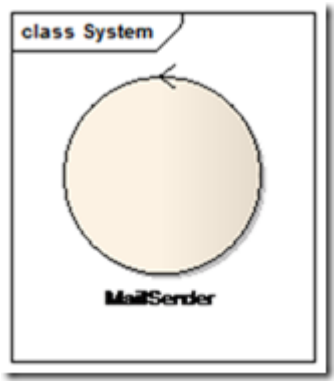
1. **Класс-сущность** (**Entity**) – обычно применяется для обозначения классов, которые хранят некую информацию о бизнес-объектах.

   Обозначается:



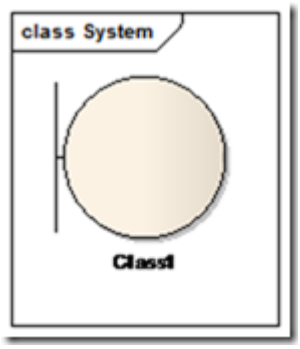
2. **Класс-контроллер** (**Controller**) – обычно используется для классов, которые используются для выполнения некоторых операций над объектами.

   Обозначается:



3. **Класс-Разграничитель** (**Boundary**) – обычно используется для классов, отделяющих внутреннюю структуру системы от внешней среды. Это могут быть WebServices, пользовательский интерфейс и пр.

Обозначается:



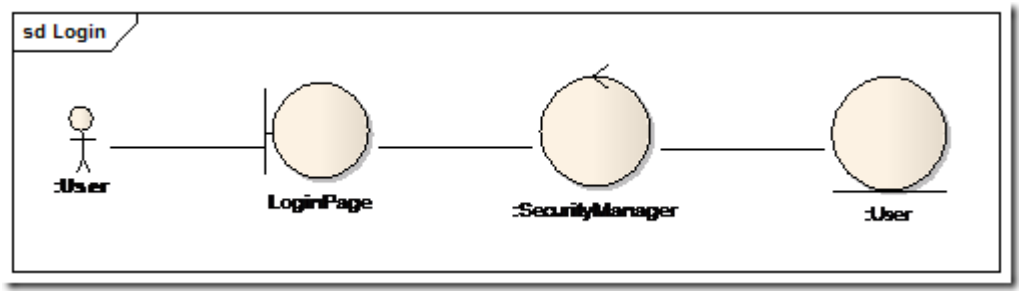
3.4 Диаграмма коммуникаций

Данный тип диаграмм также относится к поведенческим диаграммам и предназначен для описания коммуникаций между элементами системы. Диаграмма коммуникаций позволяет наглядно представить какие элементы системы задействованы при выполнении некоторой задачи и каким образом организовано их взаимодействие.

Диаграмма коммуникаций, так же, как и диаграмма последовательностей, обычно создается в привязке в варианту использования, который она описывает. Отличие от диаграммы последовательностей состоит в том, что диаграмма коммуникаций предназначена для отображения взаимодействия между инстанцированными объектами, в то время как диаграмма последовательностей описывает функционирование и структуру классов.

Обозначения, которые используются для отображения объектов на диаграмме коммуникаций – те же, что и для классов на диаграмме классов.

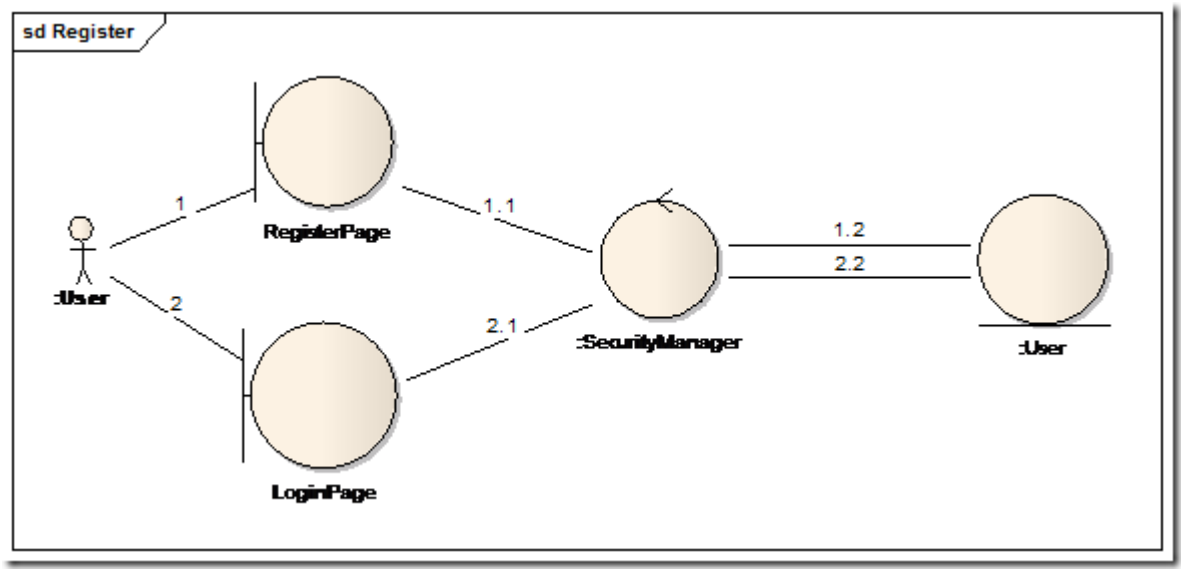
Проще всего привести описание диаграммы коммуникаций на примере:



На примере видим, что актор User взаимодействует с экземпляром страницы LoginPage, который в свою очередь работает с классом SecutiryManager, который оперирует объектами типа User. Элементы диаграммы коммуникаций могут быть связаны отношением «Ассоциация». Как я уже указывал выше Ассоциация имеем достаточно широкое значение и может трактоваться по-разному (см. Диаграммы классов).

Однако, в отличие от диаграмм последовательностей, где порядок инициации сообщений определяется шкалой времени, на диаграммах коммуникаций используется нумерация ассоциаций, которая определяет этот порядок.

Например:



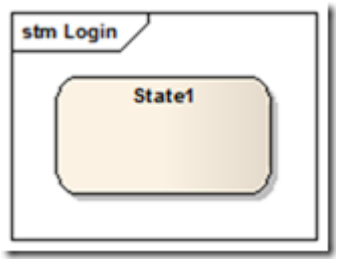
3.5 Диаграмма состояний (State diagram или State Machine)

Диаграммы состояний обычно применяются для иллюстрации того, как какой либо один элемент (обычно, один инстанцированный класс) переходит между различными своими состояниями. Диаграммы состояний также могут применяться как для описания состояний классов, так и бизнес-объектов.

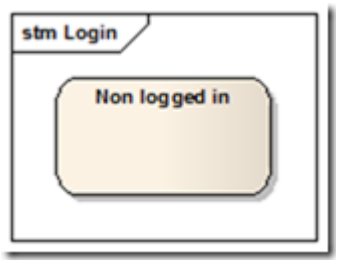
Обычно диаграммы состояний имеют вспомогательную функцию и создаются в дополнение к другим поведенческим диаграммам. Диаграммы состояний состоят из элементов двух основных типов: Состояний и Переходов.

Элемент **Состояние** (**State**) отображает состояние объекта или процесса в какой-либо момент времени.

Обозначается значком:



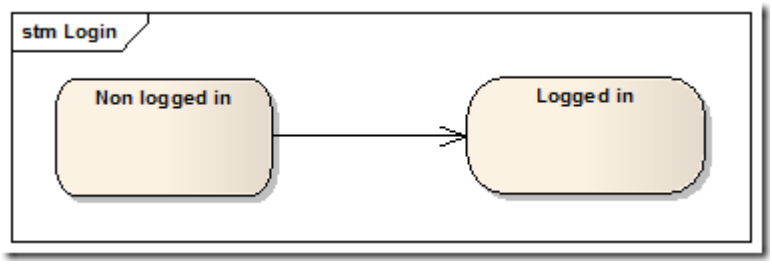
Название Состояния является его описанием, т.е., например, состояние



означает, что пользователь находится в «незалогиненном» состоянии.

Переход (**Transition**) – элемент, отображающий путь перехода из одного состояния в другое.

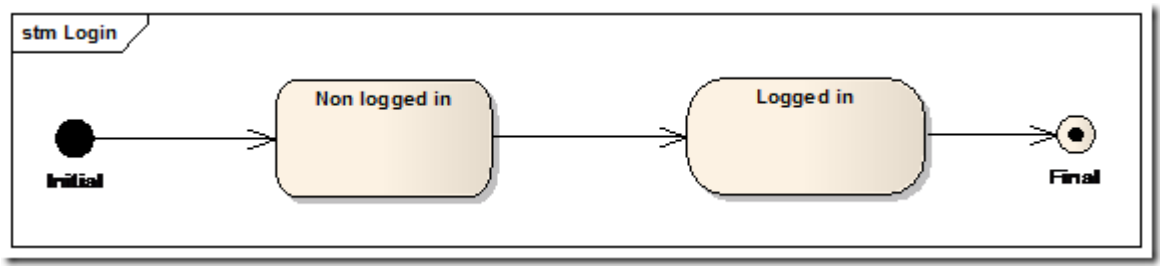
Например:



Для обозначения начала и конца всего процесса переходов используются псевдо-состояния: **Инициирующее** (**Initial**)

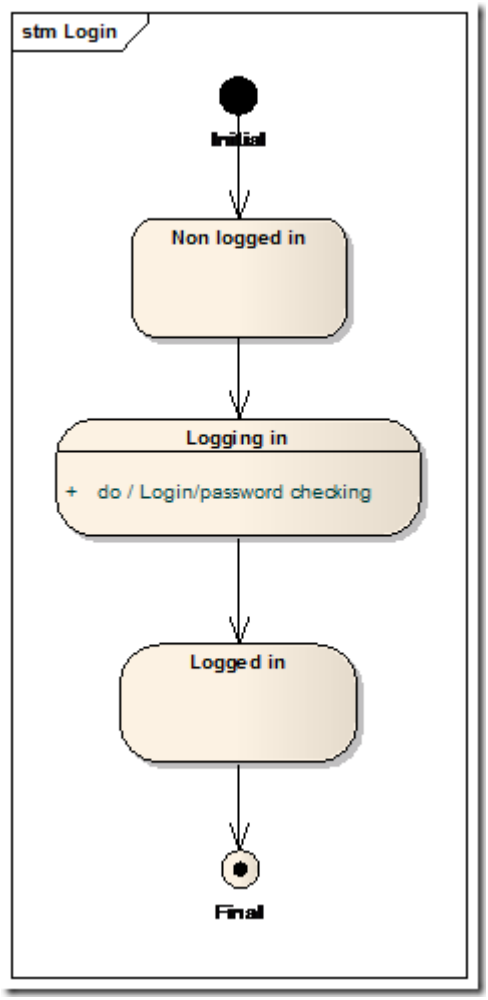
и **Финализирующее** (**Final**)

Применятся они могут, например, так:



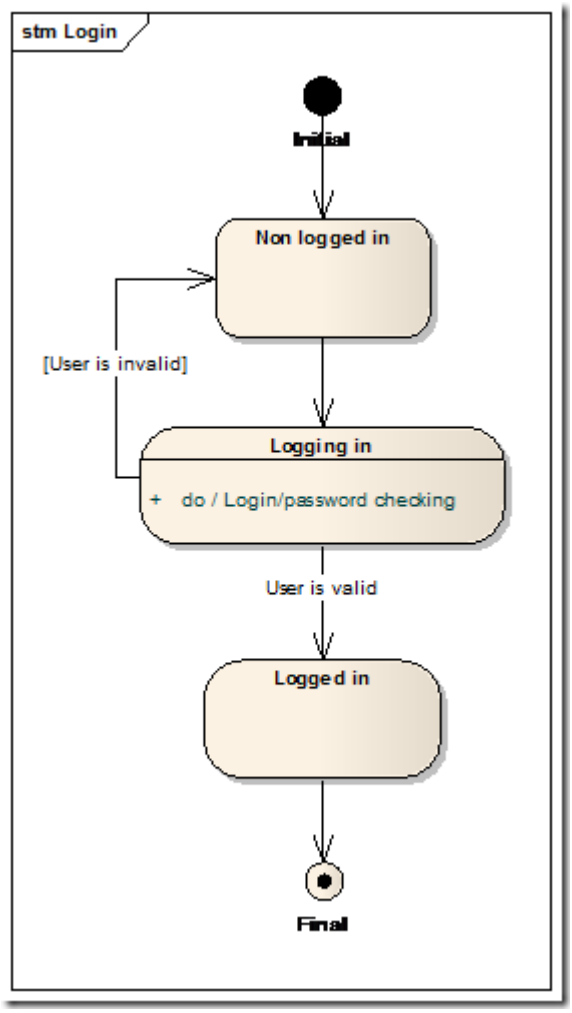
Если Состояние сопряжено с некоторой деятельностью, то это тоже отображается на диаграмме.

Например:



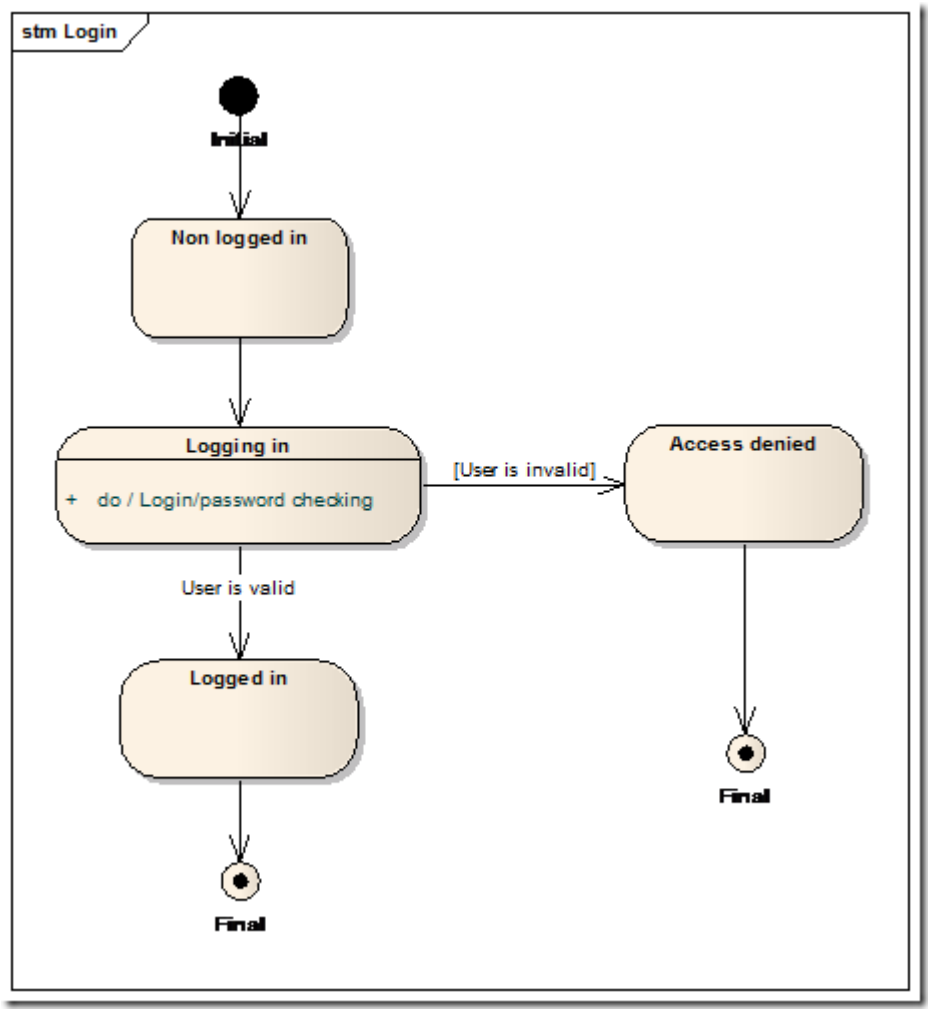
Если из одного Состояния возможно несколько переходов в несколько других различных состояний, то это тоже отображается на диаграмме. Как правило, в этом случае переходы именуются по своему смыслу. Например, по результатам деятельности.

Например:



На примере видим, что после проверки логина и пароля пользователь может быть принят или отвергнут.

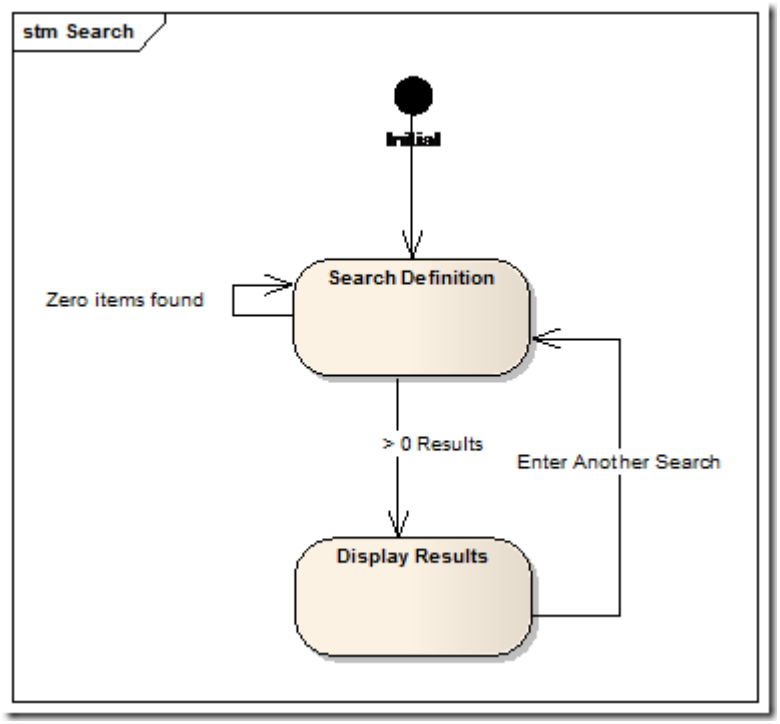
Другой вариант того же самого примера:



Этот пример отличается от предыдущего тем, что после ввода неверных данных пользователю больше не предлагается ввести свои данные.

Переход также может осуществляться между одним и тем же состоянием.

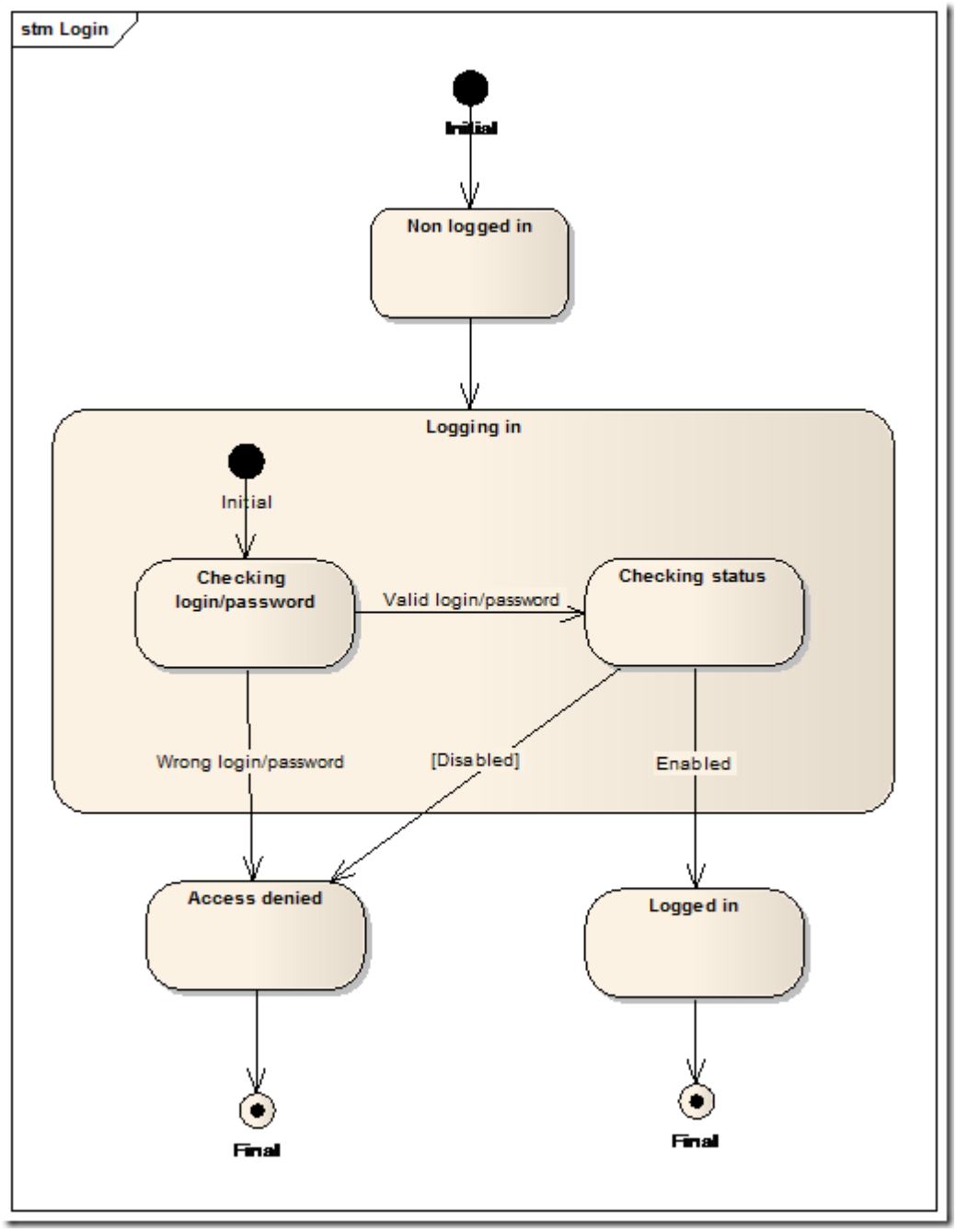
Например:



Кроме обычных Состояний (State) существуют также **Суперсостояния** (**State Machine**), которые могут включать в себя другие состояния и переходы.

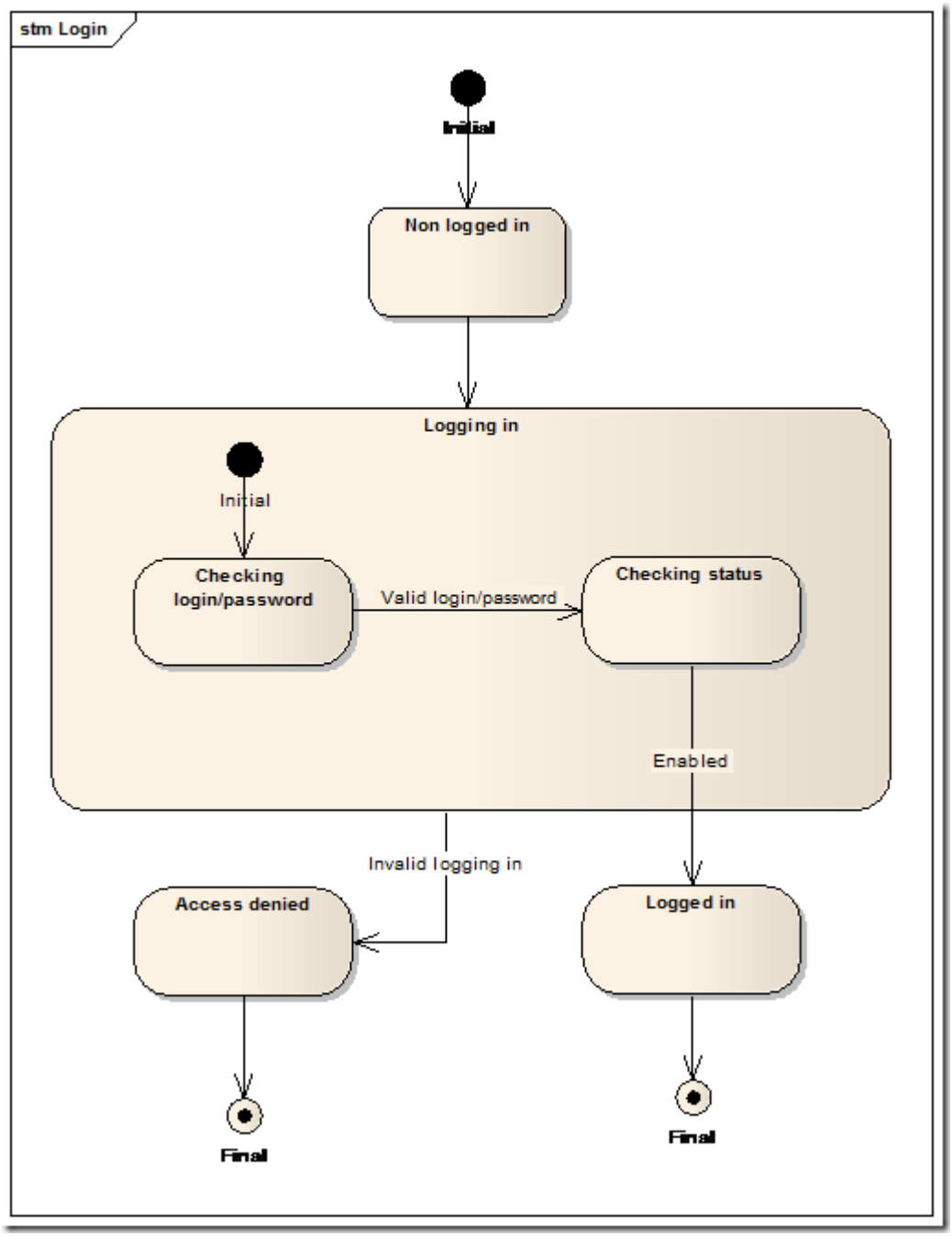
При этом контекст Суперсостояния является актуальным для всех элементов, находящихся внутри этого Суперсостояния.

Например:



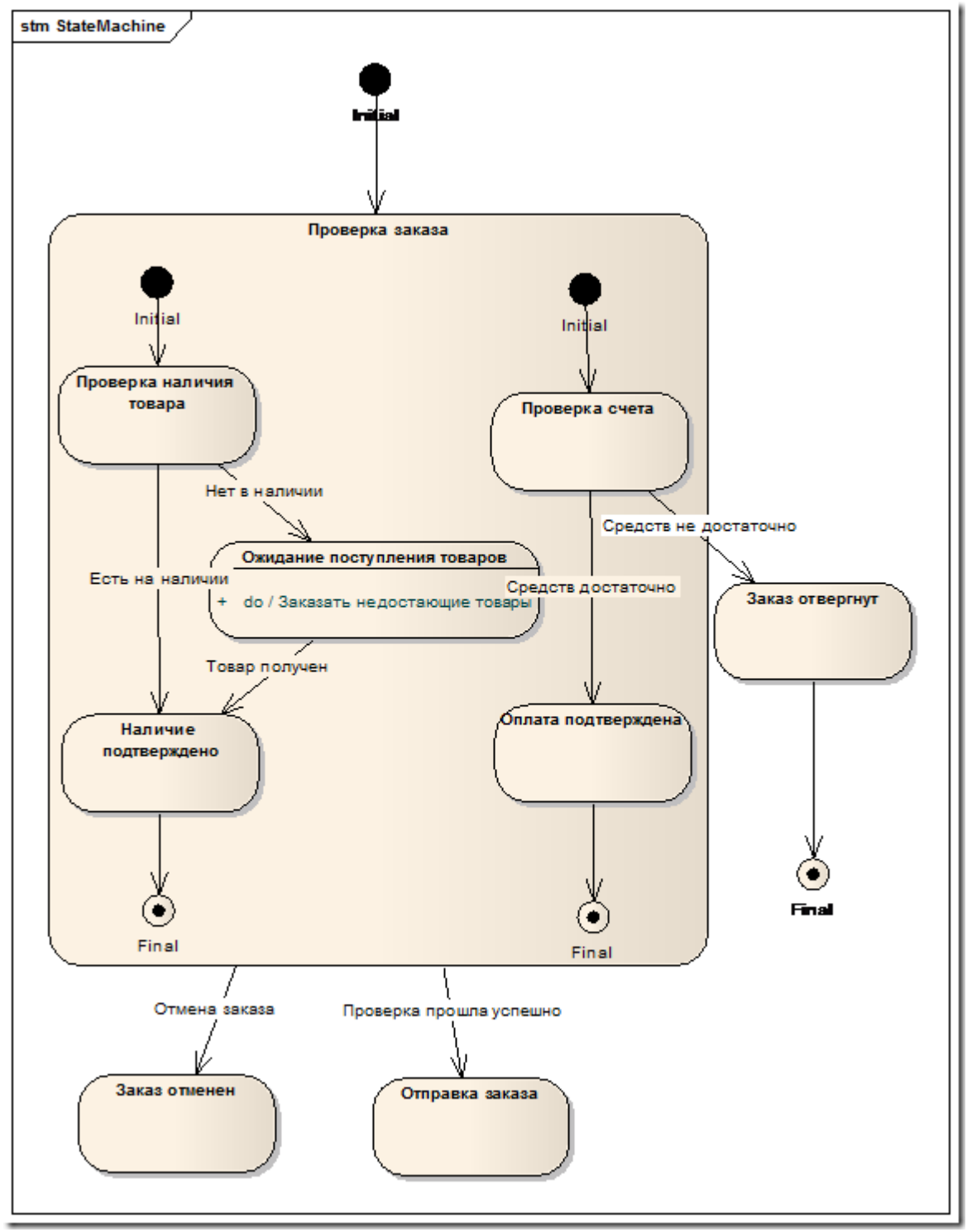
Суперсостояние само по себе может иметь переходы в другие состояния. В этом случае считается, что из любого Состояния внутри Суперсостояния может быть осуществлен такой переход.

Например:

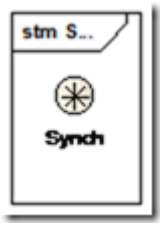


На одной диаграмме состояний может быть отображено несколько одновременных состояний одного и того же объекта, если эти состояния изменяются параллельно друг другу.

Например (из книги Фаулера):

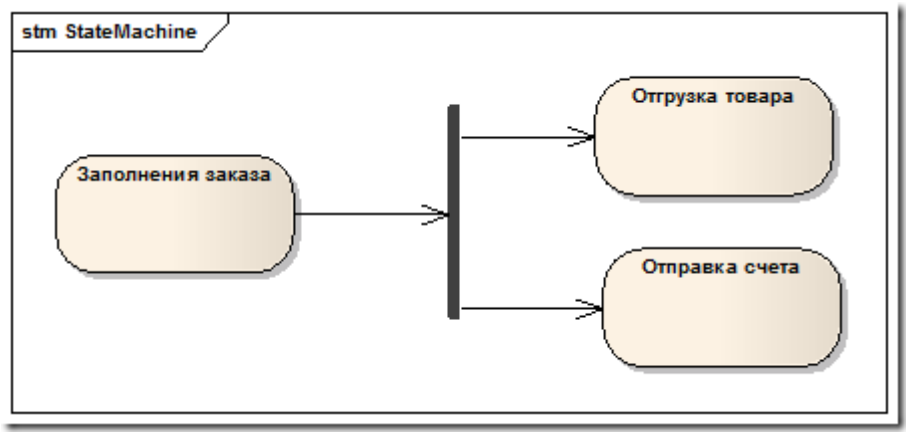


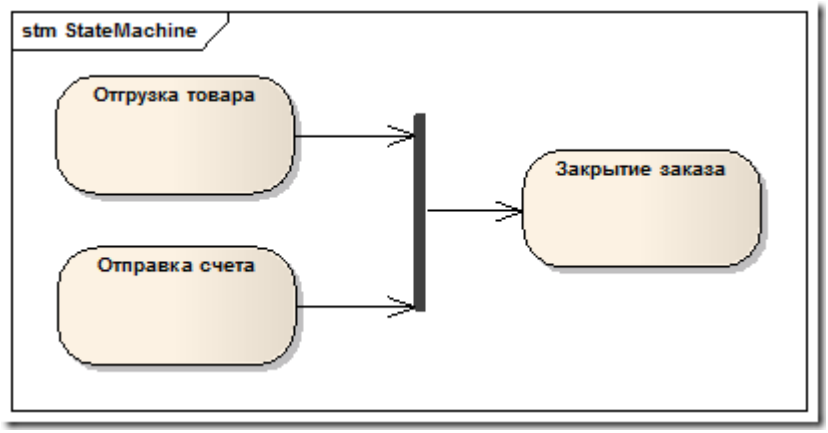
Если несколько параллельных потоков переходов должны быть синхронизироваться в какой-то момент, то для это используется псевдо-состояние **Synch**



Существует также дополнительный элемент Fork/Join – используется для разбивки или объединения нескольких потоков состояний.

Например:

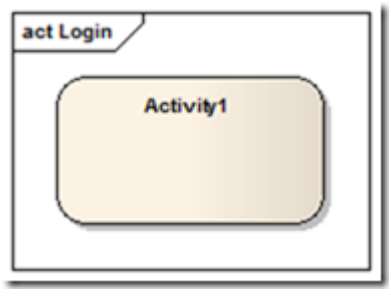




3.6 Диаграмма деятельности (Activity diagram)

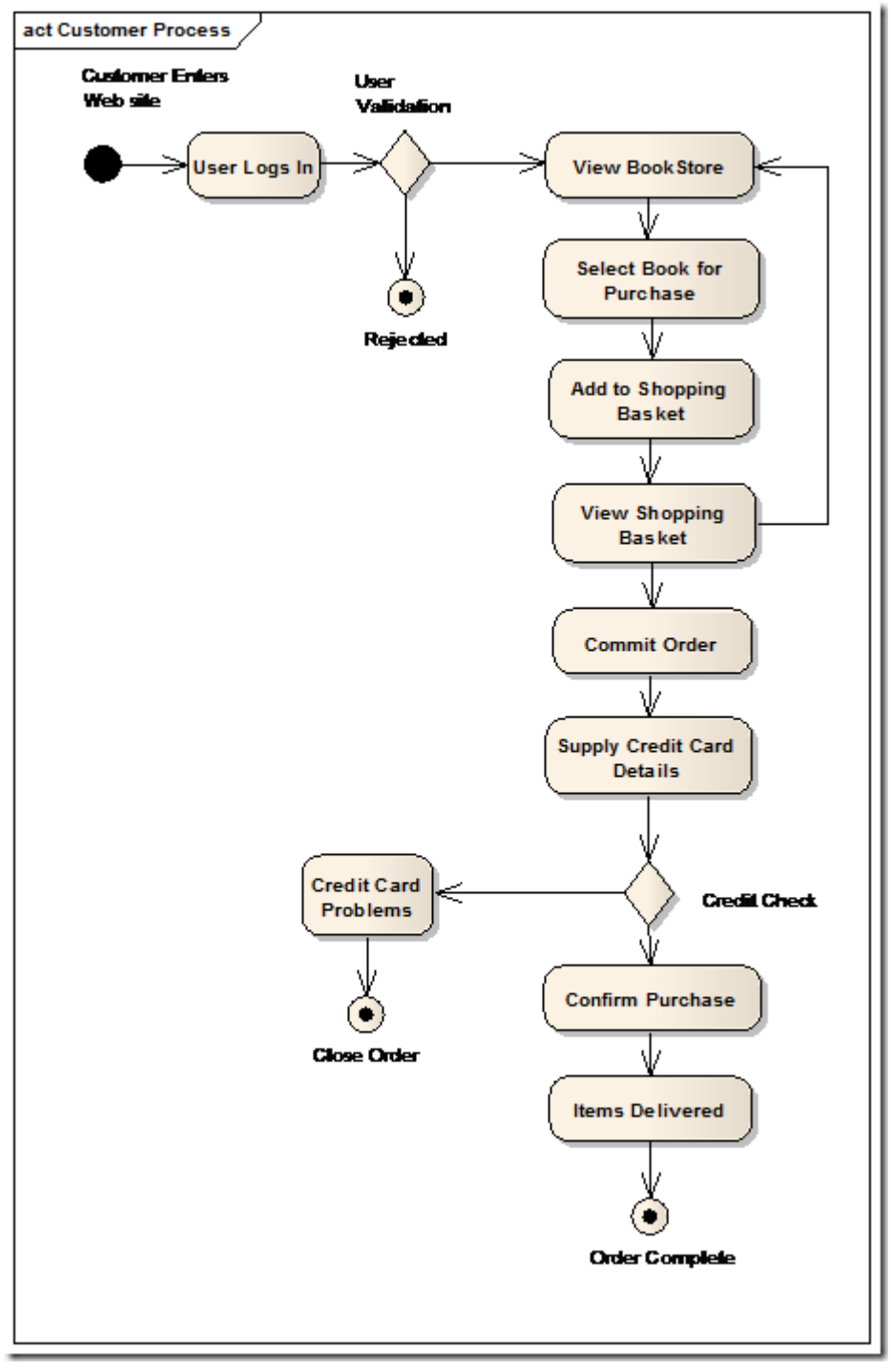
Диаграммы деятельности относятся к диаграммам, описывающим поведение системы. Они во многом родственны диаграммам состояний и имеют множество сходных элементов, но выполняют несколько другую функцию. Диаграммы деятельности предназначены для описания потоков и последовательностей выполнения работ по реализации некоторого варианты использования. В отличие от диаграмм состояний, диаграммы деятельности принимают во внимание не состояние некоторого объекта, а потоки деятельности.

Обычно для диаграммы деятельности используются для описания сложных алгоритмов, бизнес-процессов, вариантов использования и пр. Диаграммы деятельности могут быть выражены как в терминах объектов системы, так и в терминах бизнес-объектов. Основным объектом диаграмм активностей является **Активность** (**Activity**), которая обозначается следующим значком:



Диаграммы активностей имеют те же элементы, что и диаграммы состояний, а именно: псевдо-активности начала и конца потоков, переходы, Fork/Join, Суперактивности.

Пример:



На примере показан поток работ по работе с клиентом. Здесь видно, что диаграмма не посвящена состояниям только одного элемента или объекта – вместо этого она отражает поток

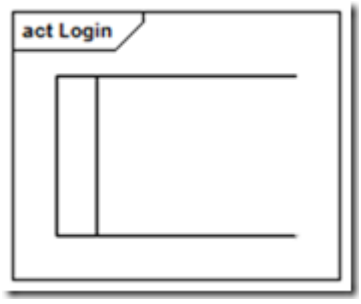
деятельностей, который затрагивает различные элементы или подсистемы.

Кроме того, на этой диаграмме присутствует новый элемент **Решение** (**Decision**) –

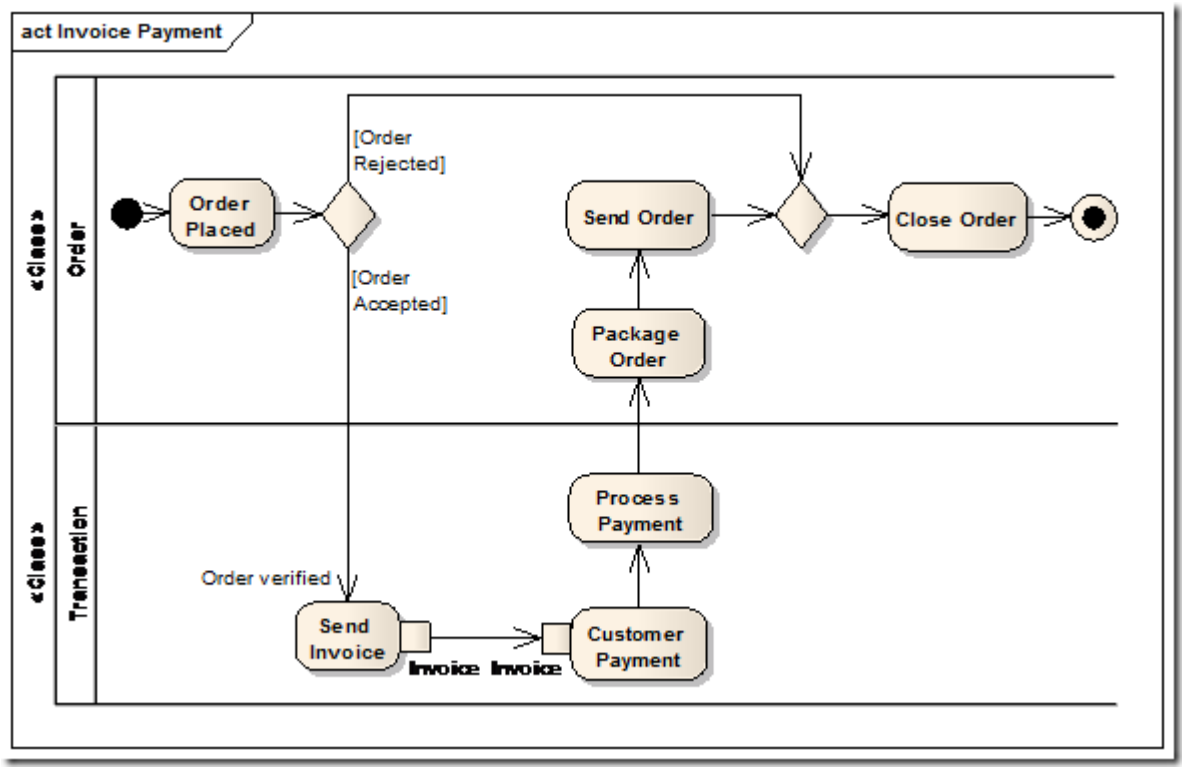
Этот элемент знаком нам по блок-схемам и обозначает момент ветвления или соединения потоков. Отличие от Fork/Join состоит в том, что процесс продолжается только по одной ветке, в то время как после Fork/Join – по всем веткам параллельно.

Существенным дополнением диаграммы деятельности является элемент **Partition**, который сам по себе логического значения не имеет и предназначен для визуальной группировки работ по какому-либо признаку.

Обозначается значком:



Например:

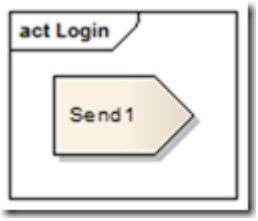


На примере видно, что потоки работ условно сгруппированы в работу по обработке заказа и по проведению платежа.

Кроме обычных активностей, есть несколько дополнительных:

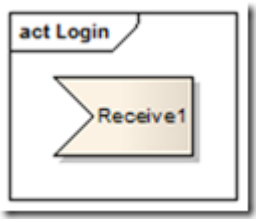
1. **Отправка** (**Send**) – используется для отображения запросов к каким-либо внешним источникам.

   Обозначается:

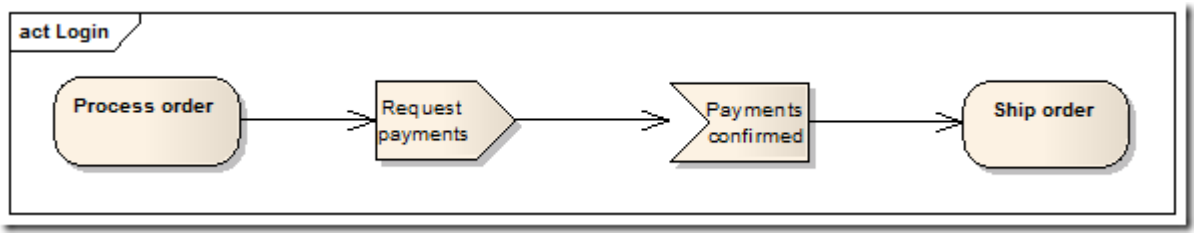


2. **Получение** (**Receive**) – используется для получения информации от каких-либо внешних источников.

   Обозначается:



Например:



4. Пакеты (Packages)

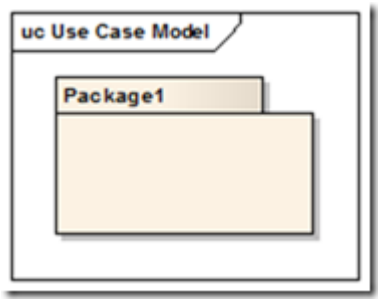
Пакеты – это способ группировки любых элементов языка UML – диаграмм, вариантов использования, классов и пр.

Обычно пакеты представляют собой:

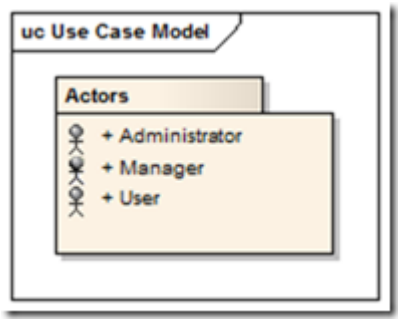
- подсистемы, если речь идет о системе
- компоненты или слои, если речь идет о классах
- наборы бизнес-логики, если речь идет о вариантах использования и т.п.

Конкретный способ группировки элементов в пакеты остается на усмотрение проектировщика.

Пакет обозначается следующим образом:

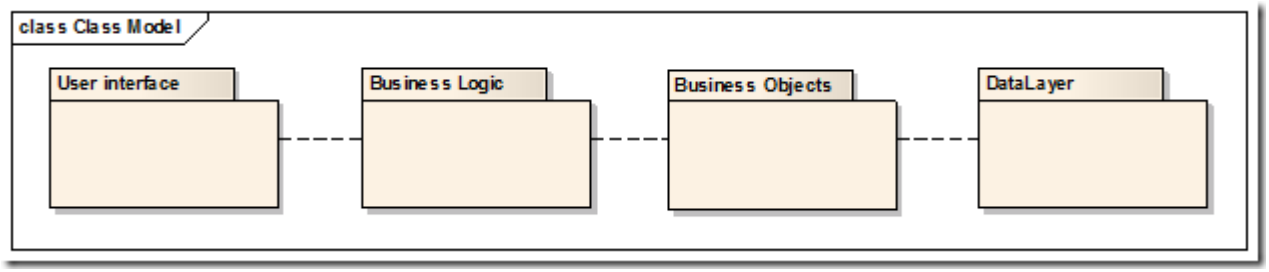


Например, может существовать пакет Акторов -



в котором находятся диаграммы с перечнем Акторов.

Пакеты также могут соединяться различными видами ассоциаций для того, чтобы продемонстрировать их взаимозависимость. Например:



Пакеты могут включать в себя не только объекты, но и диаграммы различных типов. Т.е. в каждом пакете могут быть свои диаграммы вариантов использования, последовательностей, классов и пр.