**1. Понятия сущности, отношения, диаграммы в UML2. Синтаксис диаграммы.**

UML2 состоит всего из трёх строительных блоков (рис. 2.2):

Сущности – это сами элементы модели.

Отношения связывают сущности. Отношения определяют, как семантически связаны две или более сущностей.

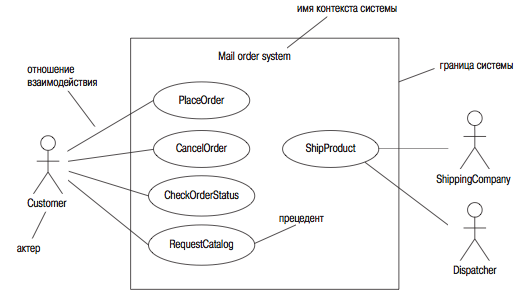
Диаграммы – это представления моделей UML2. Они показывают наборы сущностей, которые «рассказывают» о программной системе и являются нашим способом визуализации того, что будет делать система (аналитические диаграммы) или как она будет делать это (проектные диаграммы).

У каждой диаграммы может быть рамка, область заголовка и область содержимого. Область заголовка – это неправильный пятиугольник, содержащий тип (не обязательно), имя и параметры (не обязательно) диаграммы.

**2. Диаграмма прецедентов. Вид диаграммы. Назначение диаграммы.**

На диаграмме прецедентов контекст модели прецедентов изображается в виде блока с именем контекста. Этот блок является контекстом и представляет границу системы, моделируемую прецедентами. Актёры располагаются вне контекста (они внешние по отношению к системе), а прецеденты, составляющие поведение системы, располагаются внутри контекста (они внутренние по отношению к системе). Это проиллюстрировано на рис. 3.3.

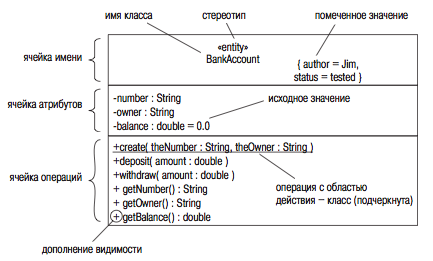
Отношение между актёром и прецедентом обозначается сплошной линией. Это символ ассоциации в UML2. Ассоциация между актером и прецедентом показывает, что актёр и прецедент каким-то образом взаимодействуют.



**Рис. 3.3.** Диаграмма прецедентов

**3. Диаграмма классов. Вид диаграммы. Назначение диаграммы.**

Обязательной частью в визуальном синтаксисе является только ячейка с именем класса. Все остальные ячейки и дополнения необязательны. Для справки на рис. 5.1. показано все.



**Рис. 5.1.** Нотация классов в UML2

В аналитических моделях обычно необходимо показывать только:

* имя класса;
* ключевые атрибуты;
* ключевые операции;
* стереотипы (если они приносят пользу делу).

Обычно *не* показывают следующее:

* помеченные значения;
* параметры операций;
* видимость;
* исходные значения (если только они не значимы для дела).

Используя общую структуру описания классов в языке UML2 возможно построить модели не только классов, но и таблиц баз данных и интерфейсов.

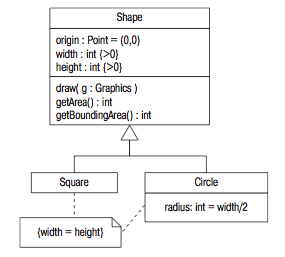
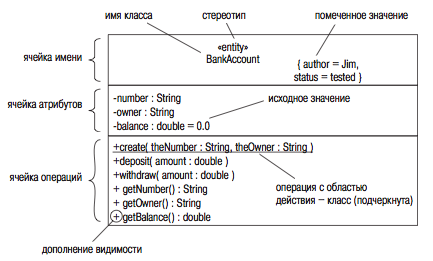


Рисунок 1.Пример диаграммы

**4. Диаграмма классов. Вид диаграммы. Назначение диаграммы. Атрибуты класса. Операции класса.**

Обязательной частью в визуальном синтаксисе является только ячейка с именем класса. Все остальные ячейки и дополнения необязательны. Для справки на рис. 5.1. показано все.



**Рис. 5.1.** Нотация классов в UML2

В аналитических моделях обычно необходимо показывать только:

* имя класса;
* ключевые атрибуты;
* ключевые операции;
* стереотипы (если они приносят пользу делу).

Обычно *не* показывают следующее:

* помеченные значения;
* параметры операций;
* видимость;
* исходные значения (если только они не значимы для дела).

Используя общую структуру описания классов в языке UML2 возможно построить модели не только классов, но и таблиц баз данных и интерфейсов.

При использовании ячейки имени (обязательной для любой диаграммы классов) и ячейки атрибутов, опуская ячейку операций, получаем описание таблиц базы данных в нотации диаграммы классов.

При использовании ячейки имени и ячейки операций, опуская ячейку атрибутов, получаем описание интерфейсов, или описание структуры диалогового взаимодействия пользователя с системой, в нотации диаграммы классов.

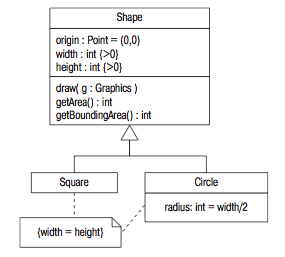
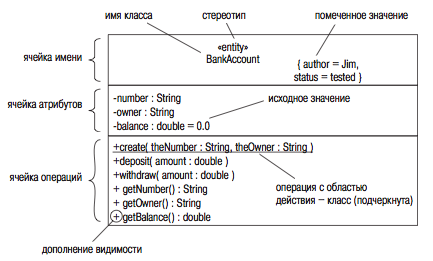


Рисунок 2.Пример диаграммы

**5. Диаграмма классов. Вид диаграммы. Назначение диаграммы. Отношения на диаграмме классов. Исключающая ассоциация между тремя классами.**

Обязательной частью в визуальном синтаксисе является только ячейка с именем класса. Все остальные ячейки и дополнения необязательны. Для справки на рис. 5.1. показано все.



**Рис. 5.1.** Нотация классов в UML2

В аналитических моделях обычно необходимо показывать только:

* имя класса;
* ключевые атрибуты;
* ключевые операции;
* стереотипы (если они приносят пользу делу).

Обычно *не* показывают следующее:

* помеченные значения;
* параметры операций;
* видимость;
* исходные значения (если только они не значимы для дела).

Используя общую структуру описания классов в языке UML2 возможно построить модели не только классов, но и таблиц баз данных и интерфейсов.

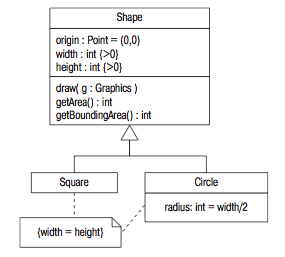


Рисунок 3.Пример диаграммы

Отношения – это семантические (значимые) связи между элементами модели. Отношения – это способ объединения сущностей в UML2.

Обычно одна связь соединяет только два объекта. Однако UML2 допускает соединение нескольких объектов одной связью. Такую связь называют n-арной. Она изображается в виде ромба, от которого отходят линии к каждому из объектов-участников.

Спецификация UML2 допускает три разных способа отображения возможности навигации. Самое распространённое обозначение:

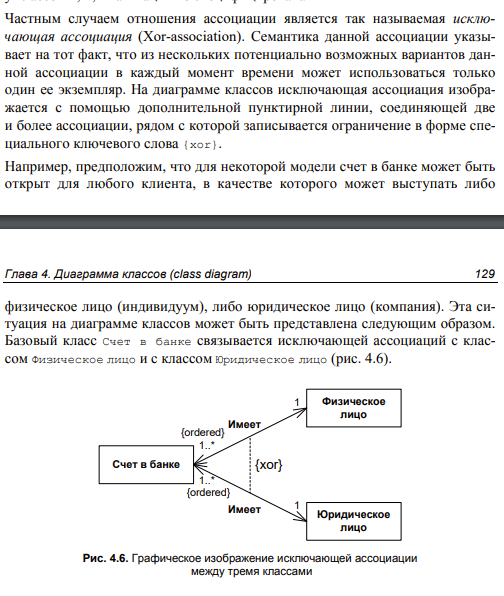
* все кресты опускаются;
* двунаправленные ассоциации изображаются *без* стрелок;
* однонаправленные ассоциации изображаются с одной стрелкой.

**Ассоциации** – это отношения между классами. Аналогично связям, соединяющим объекты, ассоциации соединяют классы. Самое главное: для того чтобы между двумя объектами была связь, между классами этих объектов *должна* существовать ассоциация. Потому что связь – это экземпляр ассоциации. Так же как объект – экземпляр класса.

Семантика базовой, неуточненной ассоциации чрезвычайно проста: ассоциация между классами указывает на то, что между объектами этих классов могут устанавливаться связи. Существуют другие, более конкретные формы ассоциаций (агрегация и композиция.

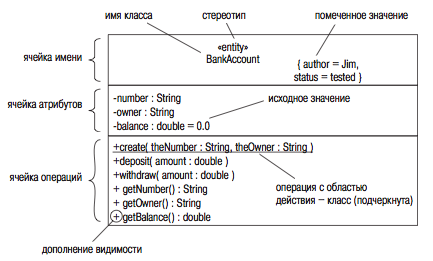
Ассоциации могут иметь:

* имя ассоциации
* имена ролей
* кратность
* возможность навигации



**6. Диаграмма классов. Вид диаграммы. Назначение диаграммы. Отношения агрегации.**

Обязательной частью в визуальном синтаксисе является только ячейка с именем класса. Все остальные ячейки и дополнения необязательны. Для справки на рис. 5.1. показано все.



**Рис. 5.1.** Нотация классов в UML2

В аналитических моделях обычно необходимо показывать только:

* имя класса;
* ключевые атрибуты;
* ключевые операции;
* стереотипы (если они приносят пользу делу).

Обычно *не* показывают следующее:

* помеченные значения;
* параметры операций;
* видимость;
* исходные значения (если только они не значимы для дела).

Используя общую структуру описания классов в языке UML2 возможно построить модели не только классов, но и таблиц баз данных и интерфейсов.

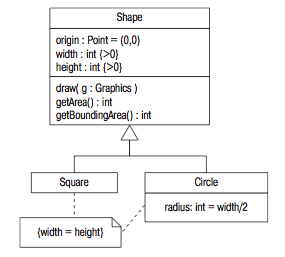
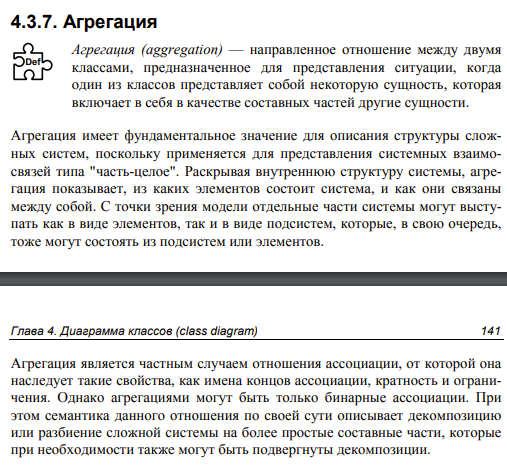
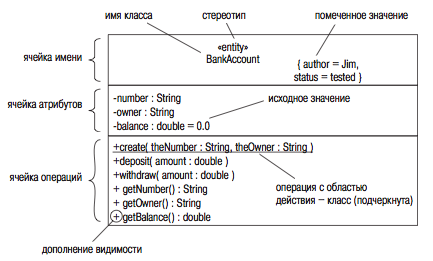


Рисунок 4.Пример диаграммы



**7. Диаграмма классов. Вид диаграммы. Назначение диаграммы. Отношение композиции.**

Обязательной частью в визуальном синтаксисе является только ячейка с именем класса. Все остальные ячейки и дополнения необязательны. Для справки на рис. 5.1. показано все.



**Рис. 5.1.** Нотация классов в UML2

В аналитических моделях обычно необходимо показывать только:

* имя класса;
* ключевые атрибуты;
* ключевые операции;
* стереотипы (если они приносят пользу делу).

Обычно *не* показывают следующее:

* помеченные значения;
* параметры операций;
* видимость;
* исходные значения (если только они не значимы для дела).

Используя общую структуру описания классов в языке UML2 возможно построить модели не только классов, но и таблиц баз данных и интерфейсов.

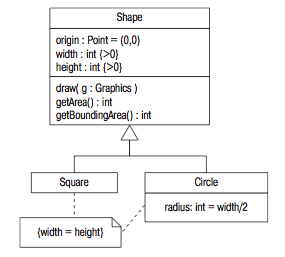
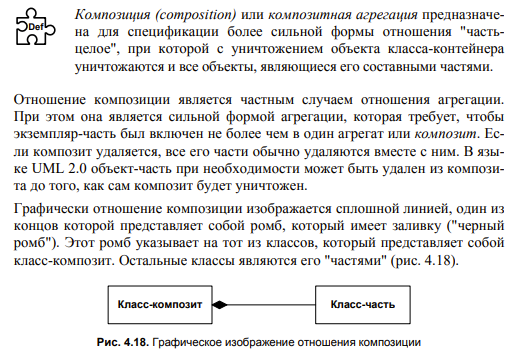
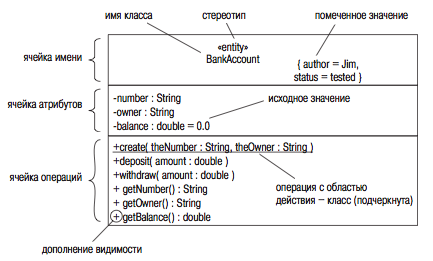


Рисунок 5.Пример диаграммы



**8. Диаграмма классов. Интерфейсы**.

Обязательной частью в визуальном синтаксисе является только ячейка с именем класса. Все остальные ячейки и дополнения необязательны. Для справки на рис. 5.1. показано все.



**Рис. 5.1.** Нотация классов в UML2

В аналитических моделях обычно необходимо показывать только:

* имя класса;
* ключевые атрибуты;
* ключевые операции;
* стереотипы (если они приносят пользу делу).

Обычно *не* показывают следующее:

* помеченные значения;
* параметры операций;
* видимость;
* исходные значения (если только они не значимы для дела).

Используя общую структуру описания классов в языке UML2 возможно построить модели не только классов, но и таблиц баз данных и интерфейсов.

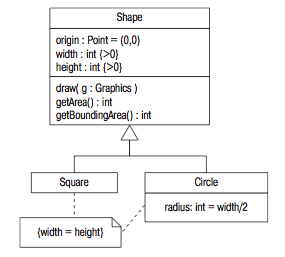


Рисунок 6.Пример диаграммы

Интерфейс определяет именованный набор открытых свойств.

Главная идея, лежащая в основе интерфейсов, – разделение *описания* функциональности (интерфейс) от ее *реализации* классификатором, таким как класс или подсистема. Создать экземпляр интерфейса невозможно. Он просто объявляет контракт, который может быть реализован классификаторами. Все, что реализует интерфейс, принимает и соглашается следовать определяемому интерфейсом контракту.

В интерфейсах также должны присутствовать описания семантики их возможностей (обычно в виде текста или псевдокода) как руководства для тех, кто будет их реализовывать.

Важно помнить, что интерфейс определяет только *описание* своих возможностей и что он *никогда* не заключает в себе какой-либо конкретной реализации.

**9. Диаграмма компонентов. Вид диаграммы. Назначение диаграммы.**

Компонент – модульная и замещаемая часть системы, инкапсулирующая ее содержимое.

Компонент как чёрный̆ ящик, внешнее поведение которого полностью определяется его предоставляемыми и требуемыми интерфейсами. Поэтому один компонент может быть заменён другим, поддерживающим тот же протокол.

Компоненты могут иметь атрибуты и операции и участвовать в отношениях ассоциации и обобщения. Компоненты – это структурированные классификаторы. У них может быть внутренняя структура, включающая части и соединители.

На диаграмме компонентов могут быть показаны компоненты, зависимости между ними и то, как компонентам назначаются классификаторы. Компонент отображается в виде прямоугольника со стереотипом «component» (компонент) и/или пиктограммой компонента в верхнем правом углу. У компонентов могут быть предоставляемые и требуемые интерфейсы и порты.

Компонент может иметь внутреннюю структуру. Части можно показать вложенными внутрь компонента или находящимися снаружи и соединёнными с ним отношением зависимости. Обе формы синтаксически эквивалентны, хотя первая нотация более наглядна.

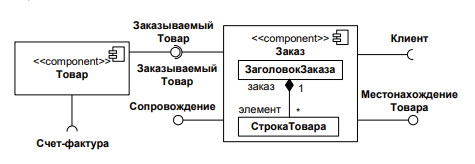


Рисунок 7. Пример

Основное назначение логического представления состоит в анализе структурных и функциональных отношений между элементами модели системы.

**10. Диаграмма компонентов. Вид диаграммы. Назначение диаграммы. Собирающие соединители. Порты.**

Компонент – модульная и замещаемая часть системы, инкапсулирующая ее содержимое.

Компонент как чёрный̆ ящик, внешнее поведение которого полностью определяется его предоставляемыми и требуемыми интерфейсами. Поэтому один компонент может быть заменён другим, поддерживающим тот же протокол.

Компоненты могут иметь атрибуты и операции и участвовать в отношениях ассоциации и обобщения. Компоненты – это структурированные классификаторы. У них может быть внутренняя структура, включающая части и соединители.

На диаграмме компонентов могут быть показаны компоненты, зависимости между ними и то, как компонентам назначаются классификаторы. Компонент отображается в виде прямоугольника со стереотипом «component» (компонент) и/или пиктограммой компонента в верхнем правом углу. У компонентов могут быть предоставляемые и требуемые интерфейсы и порты.

Компонент может иметь внутреннюю структуру. Части можно показать вложенными внутрь компонента или находящимися снаружи и соединёнными с ним отношением зависимости. Обе формы синтаксически эквивалентны, хотя первая нотация более наглядна.

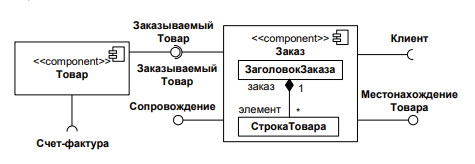
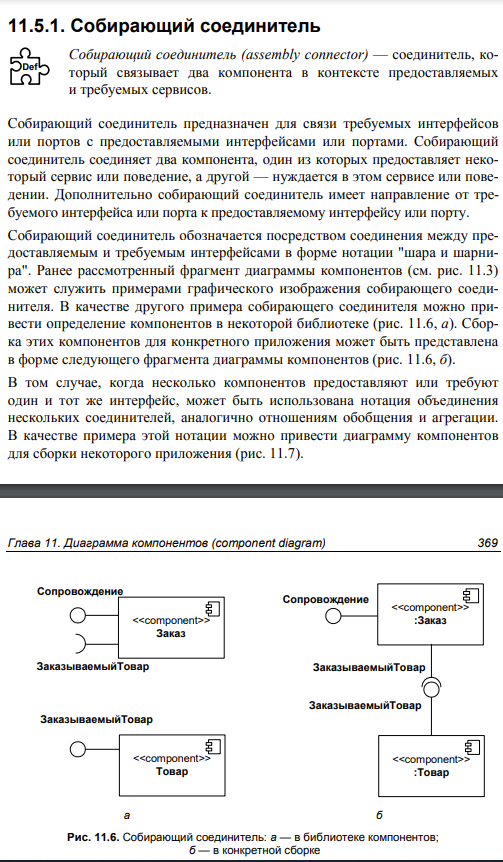


Рисунок 7. Пример

Основное назначение логического представления состоит в анализе структурных и функциональных отношений между элементами модели системы.



Порт (port) — свойство классификатора, которое специфицирует отдельную точку взаимодействия между этим классификатором и его окружением или между классификатором и его внутренними частями.

**11. Диаграмма развёртывания. Вид диаграммы. Назначение диаграммы.**

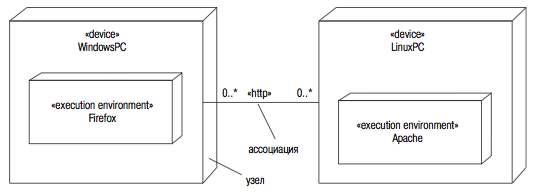
Диаграмма развёртывания проецирует программную архитектуру на аппаратную архитектуру.

Диаграмма развёртывания определяет физическое оборудование, на котором будет выполняться программная система, а также описывает, как программное обеспечение развёртывается на это оборудование.

Диаграмма развёртывания проецирует программную архитектуру, созданную при проектировании, на исполняющую ее физическую архитектуру системы. В распределённых системах она моделирует распределение программного обеспечения по физическим узлам.

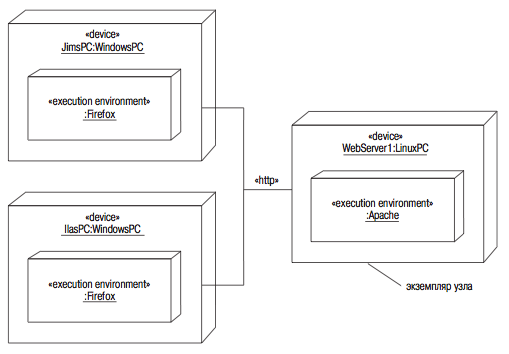
Существует две формы диаграмм развертывания.

1. Дескрипторная форма (descriptor form) – содержит узлы, отношения между узлами и артефакты. Узел представляет тип оборудования (например, ПК). Аналогично артефакт представляет тип физического программного артефакта, например Java JARфайл.
2. Экземплярная форма (instance form) – включает экземпляры узлов, отношения между экземплярами узлов и экземпляры артефактов. Экземпляры узлов представляют конкретную, идентифицируемую часть оборудования (например, ПК Джима). Экземпляр артефакта представляет конкретный экземпляр типа программного обеспечения, например, конкретный JAR файл. Если детали конкретных экземпляров неизвестны (или неважны), могут использоваться анонимные экземпляры.



**Рис. 4.6.** Дескрипторная форма диаграммы развёртывания

Дескрипторная форма диаграмм развёртывания хороша для моделирования типа физической архитектуры, а экземплярная форма – для моделирования фактического развёртывания этой архитектуры на конкретном сайте.



**Рис. 4.7.** Экземплярная форма диаграммы развёртывания

Для стереотипов можно придумать собственные пиктограммы, напоминающие реальное оборудование, и затем использовать эти символы на диаграмме развёртывания. Такой подход упрощает восприятие диаграммы.