МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

А.Н. Моисеев, М.И. Литовченко

Основы языка UML

Учебное пособие

Томск
Издательство Томского государственного университета
2023

УДК 004
ББК 32.973-01
М74

Моисеев А.Н., Литовченко М.И.

М74 Основы языка UML : учеб. пособие. – Томск : Издательство Томского государственного университета, 2023. – 96 с.

ISBN 978-5-907572-06-5

В учебном пособии рассматриваются основы языка UML (Unified Modeling Language), который широко используется при проектировании программных приложений, анализе бизнес-процессов, моделировании требований к системам, рефакторинге. Материал каждого раздела иллюстрируется примерами, которые взаимосвязаны на протяжении всего пособия. Каждый раздел снабжен контрольными заданиями и вопросами.

Для студентов и аспирантов, обучающихся по направлениям, связанным с разработкой приложений, а также специалистов, работающих в сфере информационных технологий.

УДК 004
ББК 32.973-01

Рецензенты:

О.А. Змеев, доктор физико-математических наук, профессор;
А.Ю. Дёмин, кандидат технических наук, доцент

ISBN 978-5-907572-06-5

Оглавление

Введение

Зачем моделировать программные системы. Моделирование систем играет большую роль в науке и технике. Важным оно является и для сферы разработки программных приложений. Моделирование позволяет сконструировать и испытать прообраз будущей системы, не создавая ее. Таким образом, мы можем проверить важные свойства, не затрачивая усилий и средств на создание реальной системы.

Моделирование в сфере разработки приложений представляет собой создание «чертежей» будущей системы на бумаге или на компьютере с помощью специализированных программных продуктов. В основном в этой сфере моделирование используется для прямого проектирования будущих приложений, но также возможны варианты, когда строится модель уже существующего приложения (обратное проектирование). Результатом моделирования и в том и в другом случае является комплекс моделей. Каждая из моделей, входящих в этот комплекс, преследует определенные цели и позволяет посмотреть на систему с определенной точки зрения, например, какие структуры и связи необходимо определить или как должны взаимодействовать объекты, чтобы реализовать определенный сценарий работы пользователя с системой.

Так как в современных условиях разработкой программ занимаются целые коллективы, то возникает потребность в некотором средстве, с помощью которого можно обмениваться идеями и информацией о моделях разрабатываемой программной системы. При этом желательно, чтобы все разработчики единообразно понимали представленную информацию. Таким средством общения и стал язык UML – Unified Modeling Language (унифицированный язык моделирования). Стандарт UML (версия 1.1) был принят в 1997 г. и неоднократно дорабатывался. В 2005 г. в качестве стан-

дарта была принята версия UML 2.0. На момент написания данного пособия действующей версией стандарта является 2.5.1.

Язык UML – это средство визуализации моделей программных систем. UML использует графическую нотацию, т.е. позволяет изображать устройство и работу приложения в виде рисунков – диаграмм, что сильно упрощает процесс создания и восприятия моделей приложения. При этом в UML есть практически все средства, необходимые для описания различных аспектов разработки.

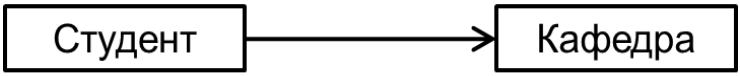
Язык UML является формальным, но допускает расширение и модификацию, а также использование некоторых условностей – так называемых значений по умолчанию. Это делает язык гибким – с помощью UML вы можете создавать как детализированные формальные модели, готовые для генерации кода, так и достаточно неформально визуализировать некоторые идеи относительно устройства и работы программы без детальных подробностей. Первый подход обычно называют построением моделей, а второй – построением эскизов.

Следует обратить внимание, что стандарт UML предоставляет инструменты для визуализации, но не решает вопросы, связанные с тем, когда и какие модели нужно создавать (это называется процессом разработки), а также с тем, как эти модели должны быть устроены (это называется проектированием). Соответственно, данное учебное пособие посвящено самому языку UML и почти не касается вопросов процесса разработки и проектирования.

Из чего состоит UML. Как любой другой язык, UML имеет свой алфавит, грамматику, синтаксис.

В качестве «строительных блоков», или «букв», UML выступают символы обычного языка, цифры, знаки операций, а также графические фигуры: прямоугольники, линии, ромбы, стрелки, овалы и другие примитивы. Из букв строятся «слова», например спецификация класса:

А из «слов» создаются «предложения» – диаграммы UML, например:

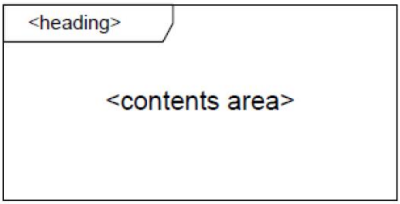


Модели в UML записываются в графическом виде – в виде диаграмм.

Важно!
Каждая **диаграмма** UML так же, как и предложение в обычном языке, **должна выражать определенную мысль**!

Поэтому большинство диаграмм содержит не все элементы модели, а лишь те, которые необходимы для формулировки высказывания. Принцип подавления (сокрытия) элементов является одним из основных в UML: диаграмма не должна включать в себя полное описание представленных элементов, иначе она становится слишком громоздкой и теряется идея, которую автор хотел донести до читателя. Только лишь часть диаграмм, задачей которых является полное формальное представление модели (спецификация), включают полное описание объектов системы.

Диаграммы можно оформлять с помощью рамки и заголовка:



но они могут и отсутствовать.

UML описывает следующие 14 конкретных типов диаграмм.

Диаграммы структуры:

Диаграммы классов.

Диаграммы объектов.

Диаграммы компонентов.

Диаграммы размещения (развертывания).

Диаграммы составных структур.

Диаграммы пакетов.

Диаграммы профилей (профайлов).

Диаграммы поведения:

Диаграммы прецедентов (вариантов использования).

Диаграммы деятельности.

Диаграммы состояний.

Диаграммы взаимодействия:

Диаграммы коммуникации.

Диаграммы последовательности.

Диаграммы обзора взаимодействий.

Временные диаграммы.

В моделях приложений обычно выделяют два самых простых и понятных аспекта – структуру и поведение (статику и динамику). Поэтому самыми понятными и распространенными среди разработчиков типами диаграмм UML являются диаграммы классов, отражающие структурную составляющую, и диаграммы взаимодействия (последовательностей), отражающие динамику системы,

описывающие алгоритмы ее поведения. Описание именно данных типов диаграмм в настоящем пособии представлено более подробно. Остальные типы диаграмм и их элементы описаны кратко – в той мере, насколько этого достаточно для начала работы с ними. Часть диаграмм – диаграммы составных структур, диаграммы профилей, диаграммы обзора взаимодействий и временные диаграммы – рассмотрены кратко, так как они применяются слишком редко и часто имеют специфические области применения.

Контрольные вопросы

1. Для чего нужен язык UML?
2. Из чего состоит UML?
3. Что такое диаграмма UML? Какие виды диаграмм есть в языке?
4. На какие два основных вида делятся диаграммы UML? Поясните, почему именно эти два вида являются важными.
5. Сколько элементов должно быть на диаграмме? Зачем в UML столько типов диаграмм?
6. Что такое прямое и обратное проектирование? Опишите нюансы этих процедур.

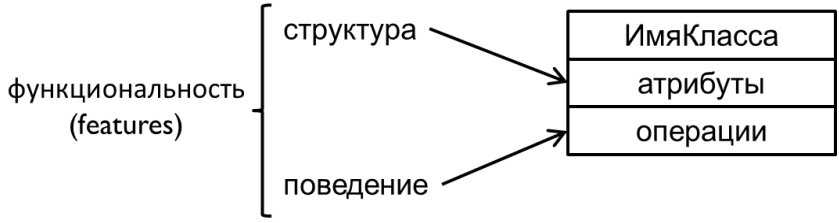
1. Диаграммы классов

Диаграммы классов – это самый простой, понятный и популярный тип диаграмм. Эти диаграммы предназначены для описания типов объектов и отношений между ними.

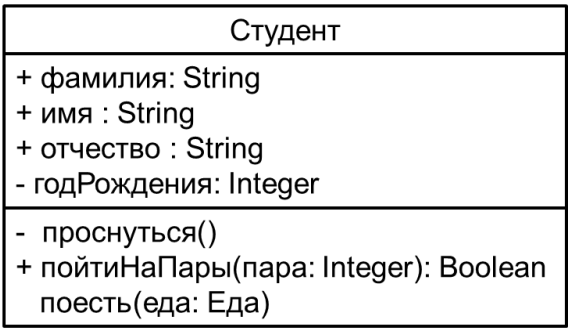
Класс можно представить как описание общих атрибутов, операций, отношений и семантики некоторой совокупности объектов. В это описание входит описание структуры и описание поведения данных объектов. В языках типа C++ и то и другое обычно называют членами класса. В совокупности структура и поведение в UML называется «features» (функциональность). В структуру класса также включают и отношения с другими классами, которые в совокупности с атрибутами называют свойствами класса (properties).

Объекты относятся к определенному классу и называются экземплярам класса (instance). Структура и поведение объектов одного и того же класса полностью совпадают. Объекты одного и того же класса отличаются только конкретными данными и связями, которыми заполнены их свойства.

Класс в UML изображается в виде прямоугольника, разделенного на три секции: имя класса, структура (свойства), поведение (операции).

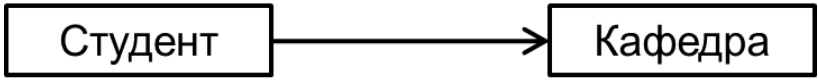


Например:



Любая из секций кроме имени класса может отсутствовать. Любой атрибут, операцию, их элемент можно опустить, если это помогает восприятию и концентрации внимания на главной идее текущей диаграммы. Например, этот класс может выглядеть так:

Зачем может понадобиться такое изображение класса? Например, для того чтобы сконцентрировать внимание на важных связях с другими объектами:



Можно оставить только раздел операций, чтобы продемонстрировать, какие операции могут вызвать другие классы. Можно оставить только раздел атрибутов, чтобы отобразить структуру класса, например его данные, которые необходимо сохранять в базе данных.

Примитивные типы данных. Для описания типов данных используются примитивные типы данных UML или типы данных, определенные пользователем. В UML определены следующие примитивные типы:

– String – строка, состоящая из 0 или более символов;

– Integer – целое число;

– Boolean – логическое значение (true или false);

– Real – вещественное число;

– UnlimitedNatural – неотрицательное целое число (0, 1, 2, …), включая символ «*», означающий неограниченность диапазона. Обычно используется для служебных целей UML – указание кратности экземпляров, ролей и т.п.

Атрибут – описание свойства класса в текстовой форме. Удобнее всего воспринимать атрибут как точку доступа к данным. Полная спецификация атрибута выглядит так:

```
мд имя: тип кратность = знач_по_умолч {ограничения}
```

Обязательно указывать только имя, всё остальное может отсутствовать. UML рекомендует имена атрибутов и операций записывать с маленькой (строчной) буквы (как и имена экземпляров), а имена классов и других классификаторов – с большой (прописной).

«мд» – модификатор доступа к атрибуту. Бывают:

«–» – доступен только внутри класса (private);

«+» – доступен всем (public);

«#» – доступен внутри данного класса и внутри его потомков (protected).

Кратность указывает, сколько экземпляров данных указанного типа хранится в атрибуте. Кратность задается квадратными скобками с указанием конкретного числа или диапазона: [2], [0..2], [*]. Следует понимать разницу между обозначениями [2] и [0..2]: в первом случае атрибут всегда содержит два значения, во втором – может содержать одно, два или ни одного. Обозначение [1..2] не означает, что есть два элемента массива и они нумеруются начиная с единицы, оно означает, сколько значений содержит атрибут (в данном случае 1 или 2). UML не отвечает на вопросы, как элементы нумеруются и, вообще, как они представлены в программе – в виде массива, нескольких полей или ещё каким-либо способом.

Значение по умолчанию определяет значение, которым будет заполнен данный атрибут при создании нового объекта.

Ограничения – перечисленные через запятую свойства и ограничения, наложенные на данный атрибут. Например: {readOnly}, {>0}, {notNull, unique}.

Следует обратить внимание, что доступ к значениям полей объекта, согласно принципу инкапсуляции (сокрытия данных), возможен только через методы этого объекта – так называемые геттеры и сеттеры (от англ. get и set). Многие современные языки программирования предоставляют еще более удобный механизм доступа – свойство (property), которое определяет совокупность get- и set-методов. В таких языках программирования с точки зрения внешних объектов свойства очень похожи на атрибуты UML: главное – чтобы было известна сигнатура атрибута, а то, как он реализован, относится к его внутреннему устройству (реализации). Таким образом, внешним объектам всё равно, хранятся данные атрибута в каком-то поле или вычисляются по запросу, главное – это доступность и сигнатура. Этим принципом следует руководствоваться, когда мы расставляем модификаторы доступа (видимости), т.е. используем public, когда эти данные вообще доступны другим объектам независимо от того, каким образом это обеспечивается.

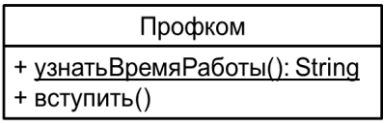
Раздел операций, по сути, содержит только сигнатуры (заголовки) возможных операций (методов) класса. Этим операция и отличается от метода, который обычно имеет конкретную реализацию. Операции определяют интерфейс класса. Они как бы говорят другим классам (точнее, их экземплярам), что можно делать с экземплярами данного класса, не раскрывая при этом деталей реализации. Внешний вид описания операции:

```
мд имя(параметры): тип_возвр_знач
```

Как обычно, обязательным является только имя операции. Рекомендуется оставлять еще () даже в случае отсутствия параметров, чтобы не спутать с атрибутом, особенно если один из разделов описания класса пропущен. Параметры перечисляются через запятую и имеют представление, похожее на описание атрибутов, за исключением отсутствия модификатора доступа (вместо него

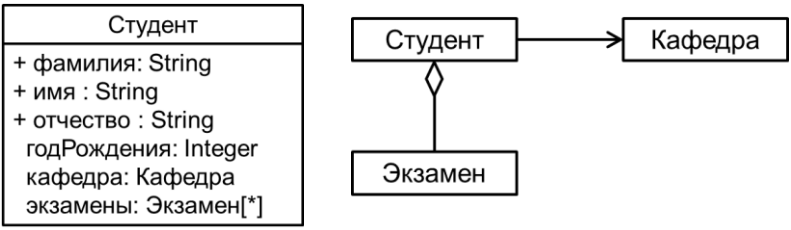
иногда используются обозначения вида параметра: in – входящий, out – выходящий). Тип возвращаемого значения указывает, какого типа значение возвращает данная операция в качестве результата. Если возвращаемый результат отсутствует (void в С-подобных языках), то двоеточие и тип опускают.

Статические члены обозначаются подчеркиванием соответствующей сигнатуры (атрибута или операции):



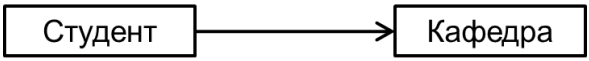
В данном примере, для того чтобы узнать время работы профкома, не нужно создавать экземпляр этого класса, для этого можно обратиться к статической операции `узнатьВремяРаботы()`.

Ассоциация представляет совокупность связей экземпляров данного класса с объектами другого класса (иногда – с другими объектами того же самого класса). Ассоциации могут быть представлены в виде свойств в текстовой форме в разделе атрибутов (левый рисунок) или с помощью специальных обозначений – линий, возможно, со стрелками или ромбами (правый рисунок):



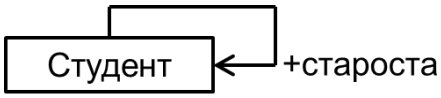
Первую форму не рекомендуется использовать. Исключение могут составлять случаи, когда классы на другом конце ассоциации не очень важны для понимания текущей диаграммы, например, являются типами данных, какими-нибудь примитивными, небольшими, вспомогательными классами, или, в крайнем случае, когда физически не помещаются на диаграмме.

Стрелка на ассоциации показывает направление видимости (навигации). Фактически это может означать, что у экземпляров (объектов) класса, откуда выходит стрелка, есть указатель на один или несколько экземпляров класса, куда она приходит:



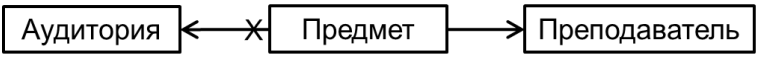
Каждый экземпляр класса Студент может быть связан с объектами класса Кафедра, другими словами, у каждого экземпляра класса Студент может быть указатель на экземпляр(ы) класса Кафедра. В коде это выражается в наличии у класса Студент поля для хранения указателя на объект класса Кафедра.

Ещё один пример:



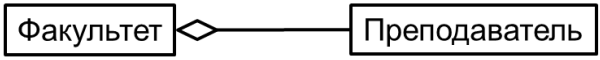
Случай, при котором ассоциация представляет совокупность связей экземпляров класса с объектами того же самого класса. Каждый экземпляр класса Студент может быть связан с экземпляром класса Студент, причём последний будет выполнять роль «старосты» для первого.

Если стрелки не указаны (прямая линия), то, возможно, предполагается двунаправленная ассоциация, либо на данной диаграмме (на данном этапе разработки) видимость пока не специфицирована. Чтобы чётко подчеркнуть, что видимость в каком-либо направлении отсутствует, на соответствующем конце ассоциации ставят крест:



В этом примере экземпляры класса Предмет имеют указатели на объекты Преподаватель, но видимость в обратную сторону явно не запрещена. С другой стороны, класс Предмет имеет ассоциацию, направленную к классу Аудитория, но для самого класса Аудитория экземпляры класса Предмет точно не видны.

Агрегация – ассоциация, в которой экземпляр одного класса включает в себя коллекцию (состоит из) объектов другого класса, обозначая связь типа «часть–целое». Обозначается линией с ромбом на конце. Ромб изображается около класса, являющегося целым (владельцем):



На каждом Факультете есть своя «коллекция» Преподавателей. При уничтожении объекта-целого, т.е. Факультета, объекты-части – коллекция Преподавателей, продолжают существовать. Также элементы коллекции преподавателей для одного объекта Факультет могут образовывать связи с другими экземплярами класса Факультет.

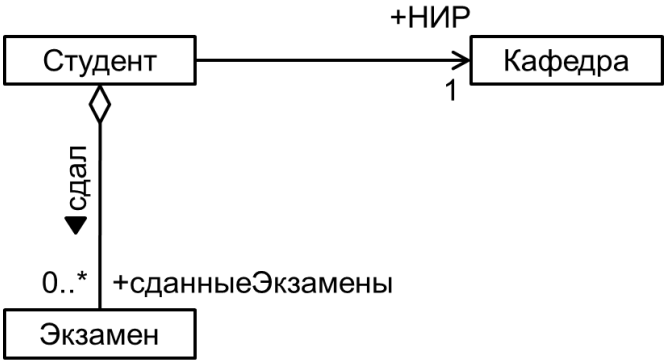
Хотя главный смысл отношения агрегации – это моделирование связей типа «часть–целое», однако на практике значок агрегации часто используется в случаях, когда подразумевается наличие коллекции объектов. Связано это в том числе и с тем, что зачастую очень трудно определить, является ли один объект частью другого.

Композиция – более сильная форма агрегации, когда экземпляры-части не могут существовать вне объекта-целого – их полноправного владельца. В таком случае владелец обычно полностью отвечает за создание, удаление своих элементов и предоставление доступа к ним.

При удалении объекта-целого УчебныйКорпус объекты-части из его коллекции аудиторий не могут существовать сами по себе и тоже уничтожаются.

Композиция также используется для отображения строгой связи «один к одному», при котором экземпляр одного класса также выступает полноправным владельцем объекта другого класса, но в единственном числе:

Как видно из рисунков, около обозначений ассоциаций могут быть подписаны имена ролей классов в ассоциации и кратности этих ролей.



У каждого экземпляра класса Студент имеется коллекция сданныеЭкзамены, состоящая из экземпляров класса Экзамен, и ссылка на экземпляр класса Кафедра, выполняющий роль НИР.

Очень редко используют имя для самой ассоциации – подпись посередине линии, обычно в глагольной форме и с треугольной стрелкой, которая помогает правильно прочитать отношения между экземплярами классов (в примере «сдал» – «студент сдал экзамены» или «экзамены, которые сдал студент»).

Кратность роли (число или диапазон, подписанный на конце линии) означает, сколько экземпляров данного класса участвуют в ассоциации с одним экземпляром класса на противоположном конце. Например, на последней диаграмме только один объект класса Кафедра должен быть в ассоциации с одним объектом класса Студент. То есть каждый студент прикреплен ровно к одной кафедре в этой ассоциации. На противоположном конце (около класса Студент) ничего не указано, значит, мы не уточняем, сколько студентов прикреплены к одной кафедре (может быть и 0, и 1, и 47). В ассоциации Студент–Экзамен кратность роли «сданныеЭкзаме-

ны» обозначена как «0..*», т.е. у одного студента может быть от нуля до бесконечности сданных экзаменов.

По умолчанию. Чтобы не загромождать диаграмму очевидными или «естественными» в мире ООП отношениями, сам UML и сообщество разработчиков используют некоторые «умолчания», которые вполне однозначно трактуются большинством людей. Особенно удобным является применение этих приемов при построении эскизов, когда не требуется точности и полноты модели. При построении же полной спецификации проекта рекомендуется выносить на диаграммы полные спецификации. В UML и сообществе разработчиков приняты следующие умолчания, когда соответствующий элемент пропущен:

– если указана кратность роли 1 или N, то это означает ровно 1..1 или N..N;

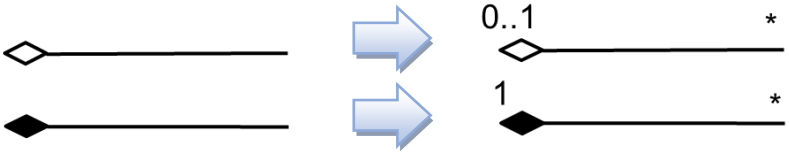
– если указана кратность *, то это означает 0..*;

– имя роли класса совпадает с именем самого класса, кроме первой буквы, которая в роли пишется строчной; видимость соответствующего свойства – private;

– для ассоциации, обозначенной прямой линией, предполагается навигация в обе стороны с кардинальными числами 0..1;

– для ассоциации с навигацией в одну сторону: кратность роли на конце стрелки – 0..1, на противоположной – *;

– для агрегации: кратность роли части – *, целого – 0..1 (для композиции – 1..1);

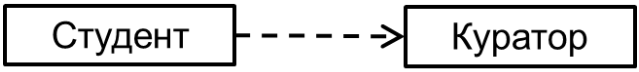


– атрибуты имеют видимость private;

– операции имеют видимость public;

– конструкторы, деструкторы, геттеры и сеттеры свойств не изображаются на диаграммах за исключением особых случаев, когда разработчик хочет подчеркнуть некоторую особенность или приводит полную спецификацию.

Зависимость – отношение между элементами модели, при котором изменения в одном из них могут повлечь изменения в другом (зависимом). Обычно проявляется в том, что объекты одного класса временно используют (зависимость типа «use», которая подразумевается по умолчанию) экземпляры другого, например, принимая их в качестве параметров операций или создавая и уничтожая их в ходе выполнения одного алгоритма (внутри метода). При этом предполагается, что зависимый объект не содержит свойства для указателя на объект, от которого он зависит. Обозначается пунктирной стрелкой, направленной от зависимого объекта к используемому (удобно запомнить, если писать над стрелкой или представлять, что над ней написано «use» – использует):

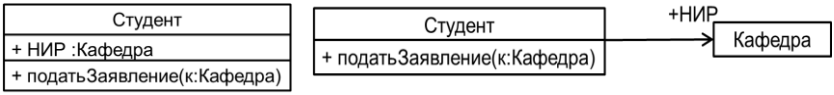


Экземпляр класса Студент не хранит указатель на экземпляр Куратор, а лишь временно его использует в качестве параметра своего метода или создавая временный экземпляр.

Следует отметить, что зависимость определяет отношения между описаниями элементов, например между классами, а не между их экземплярами. Для классов это означает зависимость одного участка кода от другого. Фактически можно считать, что

если в коде одного класса встречается имя другого, то первый класс зависит от второго.

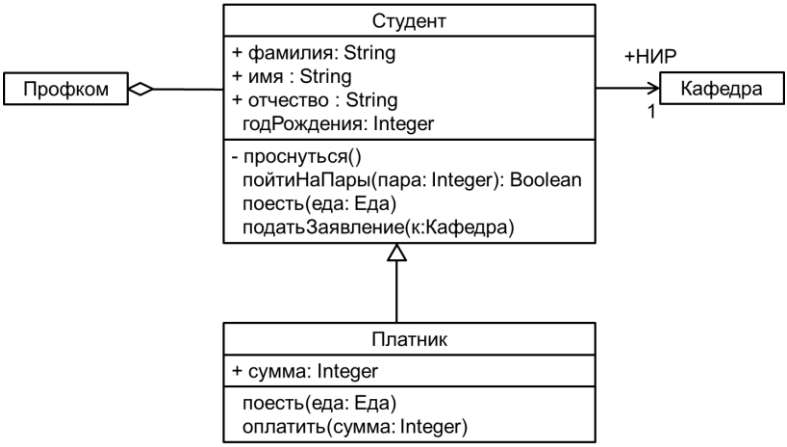
Другие виды отношений между классами – ассоциации, обобщения и т.д. – также являются зависимостями, просто более строгими, специфицированными. Поэтому в случае наличия ассоциаций или обобщений дополнительные зависимости уже не изображаются:



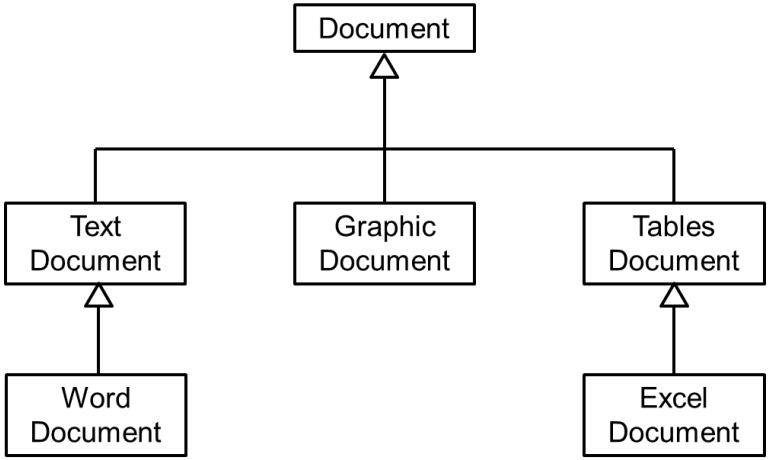
На рисунке слева изображён развернутый вид класса Студент. Атрибут «НИР» является экземпляром класса Кафедра, а следовательно, это связь является ассоциацией. Также имеется операция «податьЗаявление(к:Кафедра)», принимающая экземпляр класса Кафедра в качестве параметра, что свидетельствует о связи зависимость между этими двумя классами. На рисунке справа отображена только более сильная связь – ассоциация. То есть в данном случае нет необходимости (и даже запрещено) изображать еще и зависимость между этими классами в дополнение к ассоциации.

Обобщение (наследование) – отношение между классами типа «предок/потомок» («супертип/подтип»), при котором потомок включает в себя структуру и поведение предка, а его экземпляры могут использоваться везде, где могут использоваться экземпляры предка. При этом класс-потомок может определять новые атрибуты и операции, а также переопределять публичные и защищенные операции предка с целью изменения их реализации. Обобщение обозначается линией с не закрашенной треугольной стрелкой на одном из концов, направленной в сторону класса-предка.

Дочерний класс Платник наследует от класса Студент все свойства, в том числе и ассоциации. Дублирование операции `поесть(еда:Еда)` в классе Платник означает, что в этом классе данный метод переопределен.

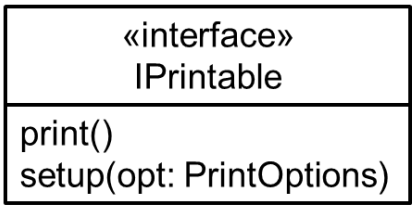


Обобщения рекомендуется изображать вертикально, располагая класс предка вверху. Если требуется показать обобщение нескольких классов (иерархию наследования), то линии обычно соединяют в одну:



Интерфейс – именованная совокупность сигнатур операций. Операции в интерфейсе всегда публичные и не имеют реализации.

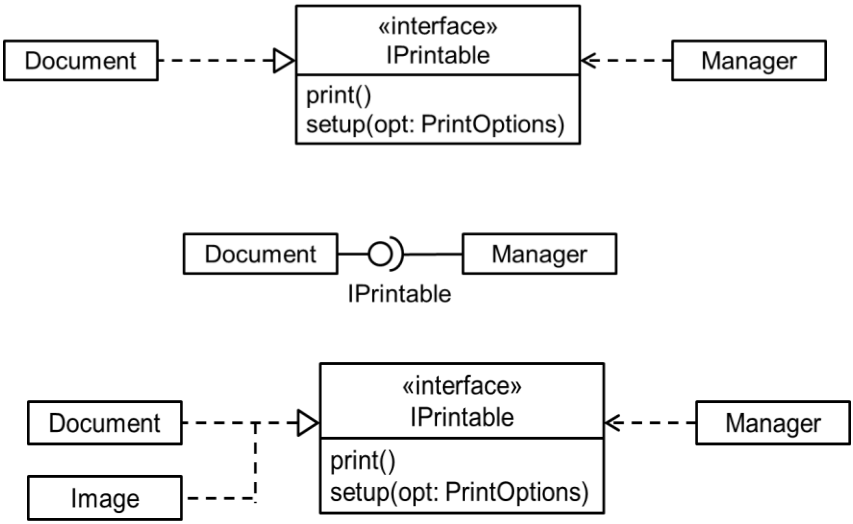
Обозначается так же, как класс (без раздела атрибутов), но перед именем ставится ключевое слово «interface». Рекомендуется начинать имя интерфейса с буквы I:



Интерфейсы используются для выделения общности классов, не входящих в единую иерархию наследования. Интерфейсы поддерживают стиль «контрактов»: они объявляют обязанности – контракты, а реализующие их классы исполняют эти контракты, реализуя каждую операцию интерфейса.

Реализация – обязательство класса реализовать операции, объявленные в интерфейсе. Изображается пунктирной линией со стрелкой в виде незакрашенного треугольника, направленной от класса к интерфейсу, который он реализует. Также применяется так называемый «свернутый» вид интерфейса, когда указывается только его имя, а факт реализации (предоставления) интерфейса классом изображается в виде «кружка на ножке». В этом случае использование этого интерфейса другими классами обычно показывается полукругом на ножке («рюмкой»), допустимо также использовать стандартное обозначение зависимости использования (пунктирная стрелка).

Класс Manager для своей работы требует интерфейс IPrintable (зависит от него), который реализуется классами Document и Image. Объекты класса Manager, требующие данный интерфейс, фактически могут использовать любой из реализующих его классов – Document или Image, однако о самих этих классах они не знают.



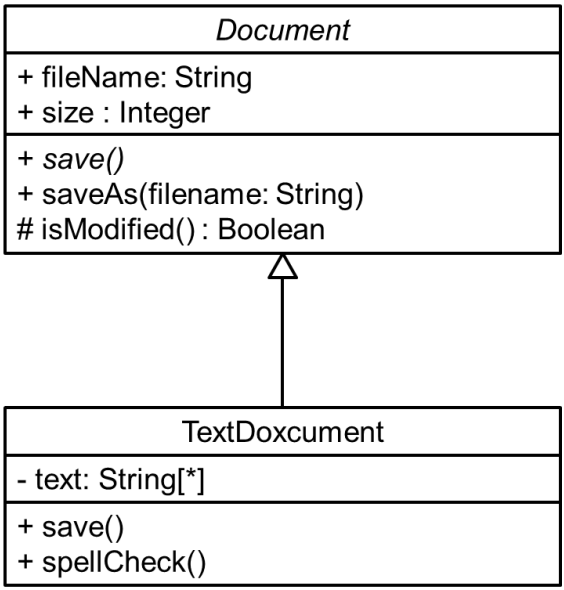
Абстрактные классы и операции. Операция класса, которая не имеет реализации в данном классе, называется абстрактной и обозначается наклонным шрифтом всей сигнатуры. Такая операция не может быть вызвана. Предполагается, что такие операции должны быть реализованы в потомках данного класса.

Класс, экземпляры которого не могут быть созданы, называется абстрактным классом. Обозначается наклонным шрифтом имени класса. Класс, содержащий хотя бы одну абстрактную операцию, автоматически является абстрактным, так как экземпляр такого класса нельзя создавать из-за угрозы вызова абстрактной операции.

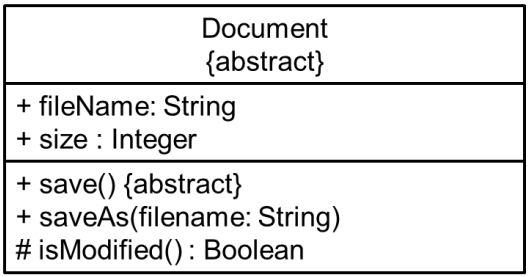
Неабстрактные элементы модели называют конкретными (классами, операциями).

На диаграмме класс Document – абстрактный. Его операция `save()` тоже является абстрактной. У данной операции отсутствует реализация, поэтому ее обязаны реализовать (переопределить) все конкретные потомки абстрактного класса: на диаграмме у класса TextDocument операция `save()` прописана прямым шрифтом.

Примечание: наличие абстрактной операции в классе является достаточным, но не необходимым условием для абстрактности класса. То есть в абстрактном классе может и не быть абстрактных операций.



При изображении диаграмм «от руки» (ручкой на бумаге) неудобно использовать наклонный шрифт для обозначения абстрактных элементов (он, как правило, всегда выглядит наклонным). В этом случае применяют ограничение {abstract}, которое записывается после элемента (имени класса или сигнатуры операции):

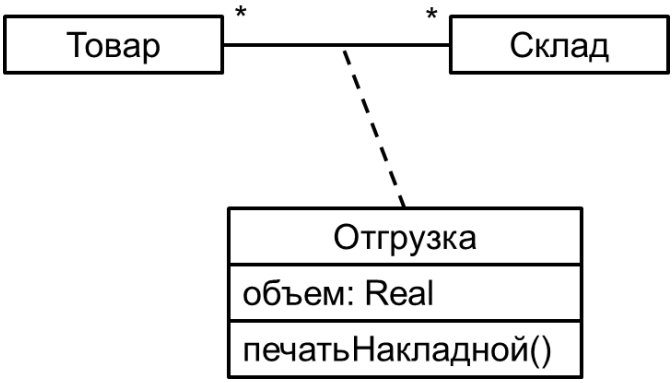


Абстрактные классы и интерфейсы. Часто задается вопрос, когда следует использовать абстрактные классы, а когда – интерфейсы. Ответ на этот вопрос зависит от многих факторов и предпочтений, но в некоторых случаях определяется однозначно:

– абстрактный класс используется, если классы (потомки) имеют общую структуру (атрибуты) и/или реализацию каких-либо алгоритмов (конкретные методы), а также когда абстрактный класс определяет вполне понятный примитив (например, элемент рисунка);

– интерфейс используется, если необходимо предоставить разнотипным классам возможность исполнять одинаковые контракты, реализуя их разными способами.

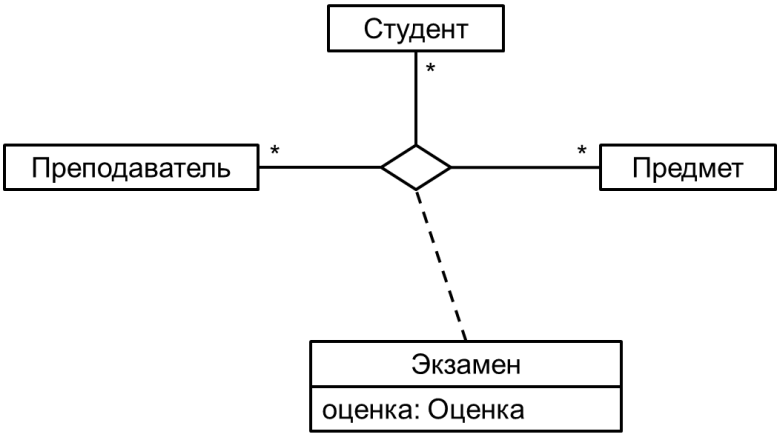
Ассоциативный класс, или класс-ассоциация, представляет сущности, порождаемые экземплярами ассоциаций, т.е. экземпляр ассоциативного класса возникает, когда объекты двух (или более) других классов вступают в связь. Ассоциативные классы выделяют в тех случаях, когда в результате этой связи появляются данные или поведение, которое нельзя отнести ни к одному объекту, вступившему в связь.



В данном примере есть много товаров, которые можно доставлять на разные склады. Для учета этих доставок создан класс Отгрузка, атрибутами которого являются объём отгружаемой про-

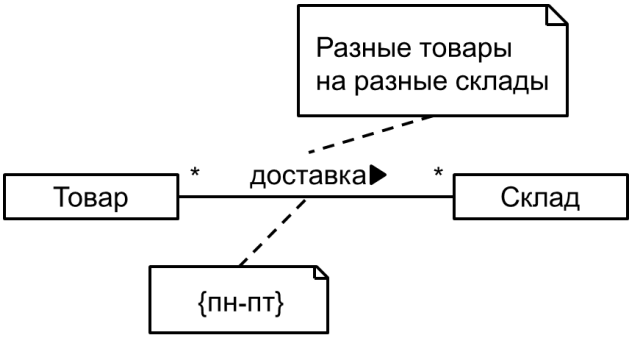
дукции и возможность распечатать накладную. Эту функциональность нельзя отнести ни к объекту Товар, ни к объекту Склад.

Многосторонняя ассоциация – ассоциация между более чем двумя классами. Обозначается ромбом, с которым с помощью прямых линий соединяются классы, участвующие в ассоциации. При таком виде ассоциации очень часто возникает необходимость в ассоциативных классах:



Ключевые слова и стереотипы – обозначения, записываемые в двойных угловых (французских) кавычках перед элементом, с помощью которых определяется вид элемента или специфицируются его особенности. Различие между ключевыми словами и стереотипами заключается лишь в том, что первые определены (зарезервированы) в стандарте UML, а вторые определяются пользователями (авторами диаграмм). Примеры: ключевые слова «interface», «use», стереотипы – всё, что может придумать разработчик, например «table». Значения ключевых слов разъяснены в стандарте UML. Значения стереотипов должен разъяснить разработчик в рамках своей документации.

Примечания используются для написания комментариев или ограничений. Изображаются в виде прямоугольника, один из углов которого загнут наподобие листа бумаги.



Ограничения заключаются в { } и в данном примере представляют собой правила бизнес-логики, которые необходимо соблюдать. В отличие от ограничений комментарии носят пояснительный характер.

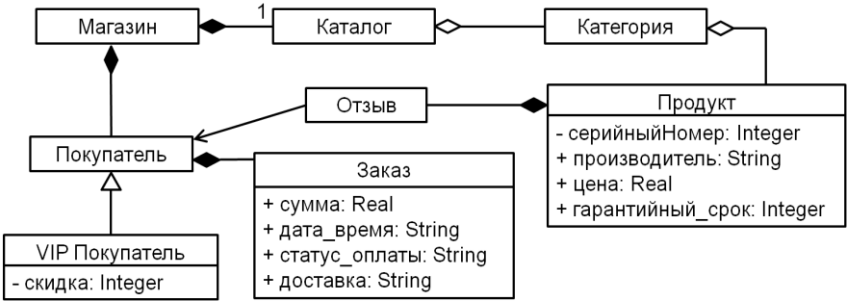
Примечание с помощью пунктирных линий соединяется с элементом или несколькими элементами, к которым относится комментарий или ограничение. Это может быть как «крупный объект», например класс, так и его отдельные элементы – атрибут, операция. Примечание можно не соединять ни с одним элементом диаграммы, тогда считается, что оно относится ко всей диаграмме, либо можно указать элемент, к которому относится примечание прямо в тексте самого примечания.

Многие описанные в данном разделе элементы, например комментарии, ключевые слова, зависимости и др., могут встречаться и на диаграммах других типов.

Далее приводятся учебные примеры диаграмм классов. Для наглядности в примерах опущены или упрощены некоторые особенности объектов и процессов, которые могут встретиться в реальной жизни.

Пример. Требуется спроектировать систему для магазина электроники и бытовой техники, в которой торговля ведется через интернет-магазин и непосредственно в филиале сети.

Сделав анализ предметной области, можно представить соответствующую модель предметной области.



Опишем сущности модели предметной области:

• Магазин – конкретный представитель торговой точки, расположенный по определенному адресу. Магазин связан с сущностью каталог, представленным в единственном экземпляре.

• Каталог – перечень категорий с содержащимися в них товарами. Товары включаются в перечень независимо от наличия в магазине.

• Категория – группа товаров, объединенных по какому-либо признаку.

• Продукт – товар, продаваемый на торговой точке или доступный для заказа в онлайн-системе. Атрибутами продукта являются: серийный номер (скрытый) – номер товара, присваиваемый им на складе для учета; доступные всем атрибуты – производитель, цена, гарантийный срок. У продукта имеется коллекция отзывов.

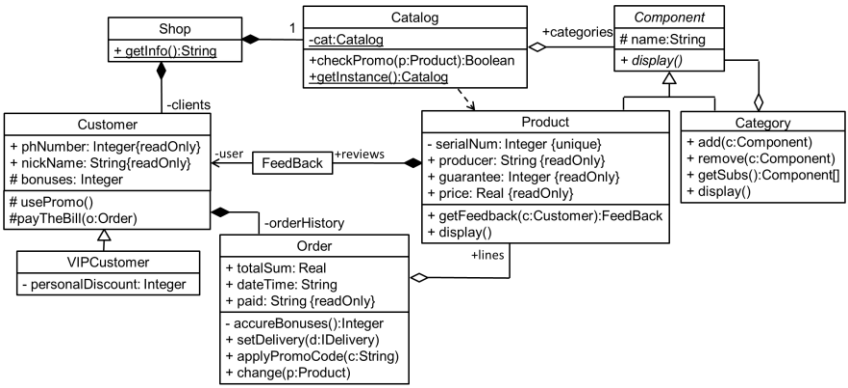
• Отзыв – отзыв об эксплуатации товара конкретным покупателем.

• Покупатель – зарегистрированный пользователь системы онлайн-продаж.

• VIP Покупатель – покупатель, имеющий персональную скидку.

• Заказ – перечень конкретных товаров покупателя, которые он собирается купить в данный момент.

Спроектируем возможную диаграмму классов для вышеописанной модели предметной области. Обратите внимание, что в учебных целях диаграмма представлена неисчерпывающая. Можете самостоятельно внести соответствующие правки в данную диаграмму.



Приведем описание каждого элемента диаграммы:

Класс Shop – описывает сущность магазина:

- -clients: Customer[] – коллекция клиентов;
- -catalog: Catalog – ссылка на один экземпляр класса каталога;
- getInfo(): String – статическая операция получения общей информации и контактов конкретной торговой точки.

Класс Catalog – описывает сущность каталога торговой точки:

- -cat: Catalog – приватный статический экземпляр для стандартного каталога;
- +categories: Component[] – публичная коллекция категорий;
- +checkPromo(p:Product): Boolean – публичная операция для проверки наличия скидки (участия в акции) товара p:Product;
- +getInstance(): Catalog – публичная статическая операция для получения экземпляра каталога.

Класс Component – абстрактный класс для структурирования вложенной иерархии категорий:

- #name: String – наименование категории;
- + *display()* – абстрактная операция для вывода категорий, и товаров в каждой категории.

Класс Category – отдельная категория, содержащая коллекцию входящих в нее товаров и/или других категорий:

- -components: Component[] – содержит ссылки на другие элементы иерархии;

• +add(c:Component) – операция по добавлению категории или товара;

• +remove(c:Component) – операция по удалению категории или товара;

• +getSubs():Component[] – операция для предоставления коллекции components;

• + display() – реализация абстрактной операции родительского класса.

Класс Product – представляет сущность товар, его атрибуты и операции:

• -serialNum: Integer {unique} – уникальный серийный номер товара;

• +producer: String {readOnly} – название производителя, доступен только для чтения;

• +guarantee: Integer {readOnly} – гарантийный срок, доступен только для чтения;

• +price: Real {readOnly} – цена, доступен только для чтения;

• +reviews:Feedback – коллекция отзывов на товар;

• +getFeedback(c:Customer): FeedBack – операция по предоставлению отзыва по использованию товара, c:Customer – ссылка на покупателя, оставляющего отзыв;

• + display() – реализация абстрактной операции родительского класса.

Класс FeedBack описывает такую сущность, как отзыв, оставленный покупателем после приобретения некоторого товара:

• -user – приватная ссылка на покупателя.

Класс Order – описывает сущность заказа покупателя, его атрибуты и операции:

• +totalSum: Real – общая стоимость заказа, суммарная по всем товарам, вхожим в заказ;

• +dateTime: String – дата и время оформления заказа, принятого к исполнению на торговой точке;

• +paid: String {readOnly} – статус оплаты: оплачено полностью, оплата при получении, оплата в кредит; атрибут доступен только для чтения,

• +lines: Product[] – коллекция позиций заказа, состоящая из экземпляров конкретных товаров;

• -accureBonuses(): Integer – операция по начислению бонусов на бонусный счет покупателя за каждый товар при покупке;

• +setDelivery(d: IDelivery) – операция по установке способа доставки заказа покупателю, требующая в качестве аргумента экземпляр класса, реализующий интерфейс IDelivery;

• +applyPromoCode(c: String) – операция применения промокода на скидку к заказу, принимающая в качестве аргумента код в виде строки;

• +change(p: Product) – у покупателя имеется возможность обменять или вернуть один и более товаров из заказа. Товар передается в качестве аргумента операции.

Класс Customer – описывает сущность покупателя:

• -orderHistory: Order[] – приватная коллекция всех совершенных заказов покупателя;

• +phNumber:Integer {readOnly} – атрибут, содержащий номер телефона, доступен только для чтения;

• #bonuses: Integer – количество бонусов на счете покупателя;

• #usePromo() – операция, позволяющая покупателю использовать промокод при оплате заказа;

• #payTheBill(o: Order) – операция по оплате заказа, в качестве аргумента принимается конкретный экземпляр заказа.

Класс VIPCustomer – описывает сущность привилегированного покупателя, получившего персональную скидку. Наследует все атрибуты и операции класса Customer и имеет дополнительный атрибут:

• -personalDiscount: Integer – персональная скидка.

Контрольные вопросы

1. Что такое класс? Для чего предназначены диаграммы классов? Для чего они не предназначены?

2. Как изображаются классы в UML?

3. Что такое атрибут класса? Как он изображается на диаграммах классов?

4. Что такое операция класса? Как она изображается на диаграммах классов?

5. Перечислите основные примитивные типы данных UML. Можно ли на диаграммах UML использовать более сложные типы или типы данных из языка программирования?

6. Что такое ассоциация? Как она изображается на диаграммах UML?

7. Какие виды ассоциаций вы знаете? Что они означают и как изображаются на диаграммах UML?

8. Зачем в UML используются значения или свойства «по умолчанию»? Какие значения и свойства UML «по умолчанию» вы знаете?

9. Что такое зависимость? Чем она отличается от ассоциации? Как зависимость изображается на диаграммах UML?

10. Что такое обобщение? Как оно отображается на диаграммах UML?

11. Что такое интерфейс и реализация? Как они изображаются на диаграммах UML? Чем интерфейс отличается от класса, а реализация – от обобщения?

12. Что такое абстрактные классы и операции? Как они изображаются в UML? Чем абстрактный класс отличается от интерфейса? Когда следует использовать абстрактный класс, а когда – интерфейс?

13. Что такое ассоциативный класс? Как он изображается на диаграммах классов?

14. Что такое многосторонняя ассоциация? Как она изображается на диаграммах классов? Можно ли ее заменить на несколько бинарных ассоциаций?

15. Что такое ключевые слова и стереотипы? Как они изображаются на диаграммах UML? Зачем в UML нужны стереотипы?

16. Как в UML изображаются примечания? Какого рода информацию можно передать с их помощью?

Задания

1. Постройте диаграмму классов, содержащую все основные статические элементы моделирования:

   а) с использованием абстрактного класса и нескольких потомков;

   б) с использованием интерфейса и нескольких его реализаций, а также класса, использующего данный интерфейс.

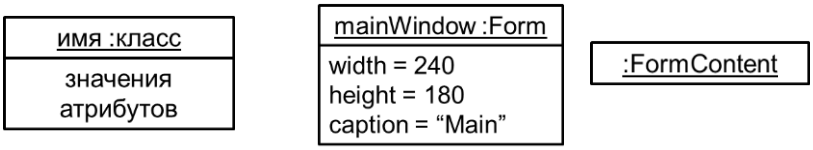
2. По диаграмме классов напишите код на языке программирования.

3. По коду на языке программирования составьте диаграмму классов.

2. Диаграммы объектов

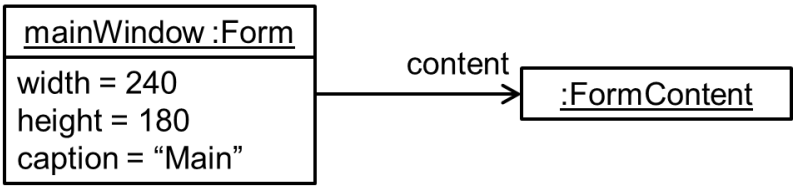
Диаграмма объектов – это возможный снимок объектов системы и связей между ними в произвольный момент времени. Диаграммы объектов отображают только экземпляры классов и значения их свойств.

Объект на диаграмме изображается прямоугольником с двумя секциями: спецификация и значения атрибутов. Спецификация определяет имя и класс экземпляра в формате `имя:Класс`. При этом одна из двух частей спецификации может отсутствовать (символ двоеточия всегда используется с именем класса). Спецификация всегда подчеркивается.

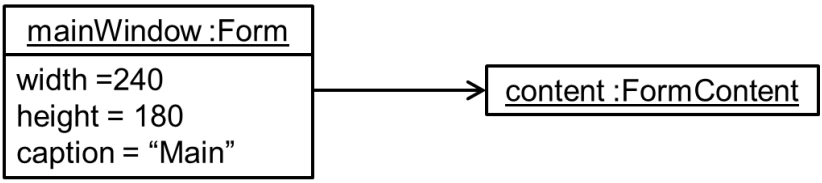


В разделе атрибутов указываются имена атрибутов и их значения. Этот раздел может быть опущен.

На диаграммах объектов также показывают экземпляры ассоциаций, при этом имена ролей следует указывать либо на соответствующем конце связи, либо в качестве имени соответствующего объекта:

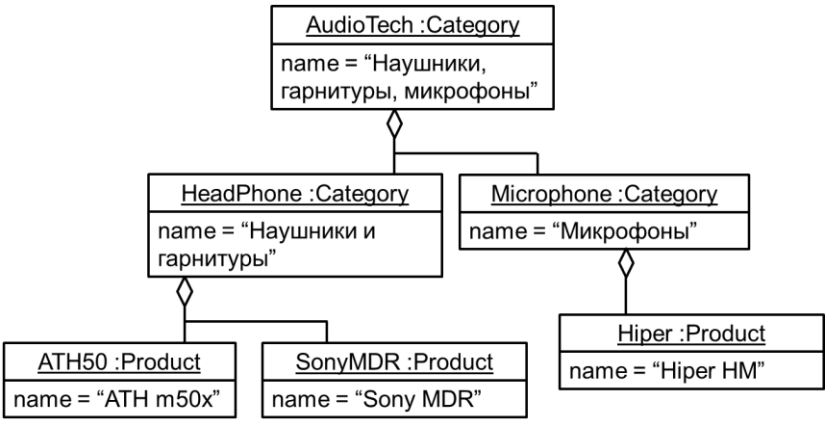


или



Диаграммы объектов применяют как вспомогательные для иллюстрации возможных сложных моментов структуры системы. Поэтому такие диаграммы применяются только в паре с диаграммами классов.

Пример. Необходимо представить иерархическую структуру категорий товаров и их вложенности друг в друга для системы онлайн-покупок.



Диаграммы объектов отображают срез системы для текущего момента времени по объектам. Применяются для того, чтобы увидеть неочевидные взаимосвязи объектов между собой, которые трудно увидеть на диаграмме классов.

На данной диаграмме представлена иерархическая структура одной из категорий товаров. Здесь прослеживается следующая вложенность: категория AudioTech включает категории

HeadPhone, Microphone. В последних двух категориях содержатся объекты типа Product, они являются непосредственно товарами, входящими в категории, «листовыми» объектами, не содержащими ссылки на другие категории или товары. Это можно заметить также на диаграмме классов разделом выше.

Контрольные вопросы

1. Для чего используются диаграммы объектов? Что на них изображается?
2. Как изображаются экземпляр класса и экземпляр ассоциации на диаграммах UML?
3. Как вы думаете, насколько часто приходится создавать диаграммы объектов в реальных проектах? Почему?

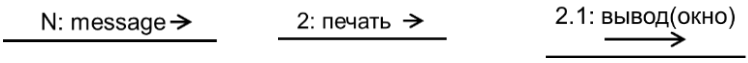
Задание

Составить диаграмму объектов, *полезную* для вашего проекта.

3. Диаграммы коммуникаций

Диаграммы коммуникаций отображают связи между участниками и порядок передачи сообщений между ними. В качестве участников (lifeline) могут выступать различные элементы модели, которые могут посылать и/или принимать сообщения: экземпляры классов, компоненты, актеры и т.п. Участник на диаграмме коммуникаций изображается значком, соответствующим классификатору данного элемента. Чаще всего это прямоугольник. Спецификация участника задается аналогично спецификации объекта (`имя : Класс`), но не подчеркивается.

Сообщения изображаются следующим образом: два участника соединяются линией (коммуникации), над линией или около нее изображается спецификация сообщения – пишется порядковый номер сообщения в алгоритме, символ «:», имя сообщения (возможно, с параметрами) и стрелка, указывающая направление, куда посылается сообщение. В качестве параметров сообщений могут выступать только фактические параметры (конкретные значения или имена других участников). Стрелка может изображаться в той же строке, что и сообщение, или ниже.



Пример диаграммы:

Нумерация сообщений используется для того, чтобы было понятно, в какой последовательности выполняется изображенный алгоритм. Стандарт UML утверждает вложенную нумерацию: 1, 1.1, 1.1.1 и т.д., для различных потоков управления. Например, если операция `b()` первой вызывается внутри операции `1.3: a()`, то она должна нумероваться как 1.3.1. На практике такая нумерация применяется не всегда, так как диаграммы коммуникации часто изображают логику взаимодействия, для которой подчиненность потоков управления не играет особой роли, поэтому разработчики зачастую используют обычную сквозную нумерацию: 1, 2, 3.

На диаграммах коммуникации можно показать посылку сообщений в цикле и ветвления. Цикл изображается символом «*» перед порядковым номером сообщения. Такое обозначение используется, чтобы показать, что данная операция применяется к каждому элементу некоторой коллекции.

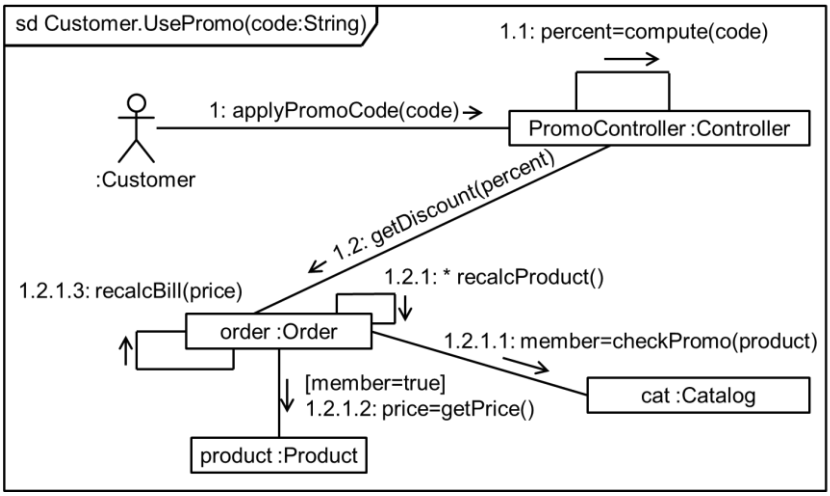
Цикл вида `*[условие]` выполняется с одним и тем же объектом до тех пор, пока условие истинно. Вместо условия в квадратных скобках можно указать конкретное число итераций или даже переменную цикла. Например:

```
*[обработано < 12] 1.4: начислить()
*[2 раза] 1.4: сигнал()
*[i:=1..n] 1.4: проверить()
```

Ветвление реализуется с помощью сторожевых условий вида `[условие]`. Указанное сообщение посылается, если условие выполнено. Соответствует ветвлению if без раздела else.

Диаграммы коммуникаций используются для того, чтобы отобразить не очень сложные взаимодействия, обычно это наброски взаимодействий (алгоритмов). Основное внимание на данных диаграммах сосредоточено на связях объектов и их совместных действиях для достижения результата.

Пример. Задача: описать процесс применения промокода покупателем. Промокод применяется только к тем продуктам, которые в данный момент принимают участие в акции.



На диаграмме изображен сценарий UsePromo объекта Customer. Первым действием Customer посылает сообщение applyPromoCode(code) объекту PromoController класса Controller, в качестве аргумента передавая code. PromoController на основе code вычисляет percent собственным методом compute(code), после чего посылает сообщение getDiscount(percent) объекту order одноименного класса.

В процессе выполнения метода getDiscount(percent) объект order последовательно выполняет несколько действий. В шаге 1.2.1 * показывает, что вложенные в него сообщения (1.2.1.1–1.2.1.3) выполняются в цикле для всех объектов Product в заказе order:

1.2.1.1. Сначала производится посылка сообщения checkPromo(product) объекту cat класса Catalog.

1.2.1.2. Результат member на следующем шаге выступает сторожевым условием. При значении member = true объект order запрашивает price у объекта product.

1.2.1.3. После этого order пересчитывает сумму внутри собственного метода recalcBill(price).

Диаграммы анализа. Диаграммы коммуникаций активно используются на начальных этапах разработки приложения для анализа взаимодействий. К сожалению, в стандарт UML 2 не вошел специальный набор стереотипов для классификаторов участников, предлагаемый Унифицированным Процессом разработки и активно применяемый многими разработчиками. Эти стереотипы называются классами анализа, а диаграммы классов и коммуникаций, построенные с использованием этих стереотипов, – диаграммами анализа. Стереотипы и диаграммы анализа позволяют на ранних стадиях разработки сосредоточиться на простых и высокоуровневых реализациях сценариев взаимодействия пользователя с системой. На данных диаграммах изображают следующих участников:

Актёр (actor) – объект, воздействующий на систему извне, ожидающий результата своих действий.

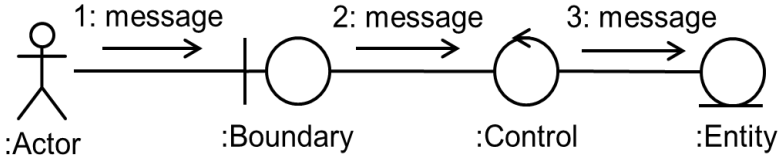
Граничный класс/объект (boundary) – объект, с которым непосредственно взаимодействует актер.

Управляющий класс/объект (control) – объект, выполняющий основные действия (управляющий сценарием).

Сущность (entity) – объект, представляющий данные или сущность для манипуляций.

Замечание: актёр взаимодействует только с граничным классом, который в свою очередь передает и получает данные только от управляющего класса. А управляющий класс манипулирует дан-

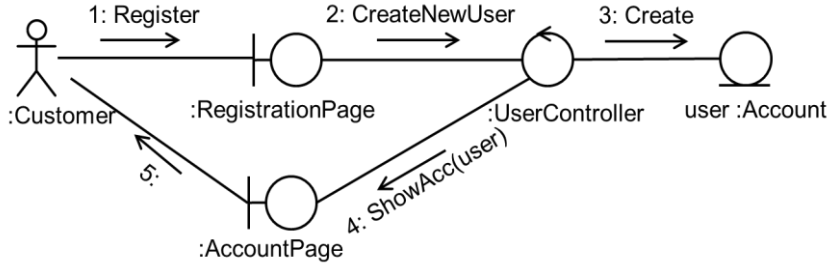
ными, которые хранятся в некоторых сущностях. Сам объект сущности обычно не отправляет сообщения, а выступает в качестве хранилища данных. Ниже представлена типовая схема взаимодействия вышеупомянутых объектов.



Диаграммы коммуникаций, составленные с помощью этих стереотипов, демонстрируют общий взгляд на то, какие объекты могут быть задействованы в том или ином сценарии и как они взаимодействуют для получения результата актёром. Набор диаграмм анализа составляет модель анализа проекта. Как правило, в этой модели много диаграмм коммуникаций и немного диаграмм классов анализа, каждая из которых обычно составлена на основе нескольких диаграмм анализа коммуникаций.

Пример. Описать взаимодействия пользователя и системы онлайн-покупок через web-интерфейс.

На первой диаграмме изображен процесс регистрации пользователя на сайте для совершения дальнейших покупок.



Первым действием объект Customer вводит данные на странице регистрации – граничный класс RegistrationPage. Граничный класс посылает сообщение (2) соответствующему контроллеру UserCon-

troller о том, что необходимо создать нового пользователя. Управляющий объект создает (3), например, в базе данных новую сущность user класса Account, а затем отображает (4) аккаунт нового созданного user на соответствующей странице AccountPage. Сообщение (5) на этой диаграмме возвращает управление объекту Customer и подразумевает, что он увидит именно страницу AccountPage.

Контрольные вопросы

1. Для чего используются диаграммы коммуникаций? Кто такие участники? Какие сущности могут выступать в качестве участников?
2. Как изображается участник на диаграмме коммуникации?
3. Как изображается сообщение на диаграмме коммуникаций?
4. Как изобразить цикл и ветвление на диаграмме коммуникаций?
5. Для чего используются диаграммы анализа?
6. Какие основные виды участников изображаются на диаграммах анализа? Каковы их роли и особенности взаимодействия?

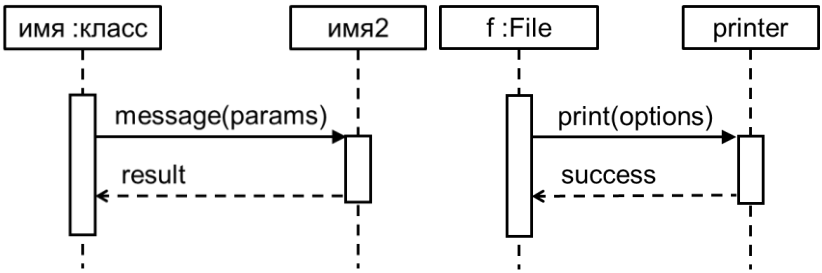
Задание

Выберите любое приложение или веб-сайт и на основе наблюдаемых сценариев взаимодействия пользователя с приложением составьте несколько диаграмм анализа.

4. Диаграммы последовательностей

Диаграммы последовательностей – более строгое (более формальное) описание алгоритмов в UML. На этих диаграммах, в отличие от диаграмм коммуникаций, основное внимание сконцентрировано не на связях объектов, а на последовательности сообщений. У диаграмм последовательностей имеется временная шкала, которая неявно проходит на диаграмме сверху вниз. Таким образом, любое сообщение B, расположенное на диаграмме визуально ниже сообщения A, в программе будет выполнено позже, чем A. Такой порядок задает строгую последовательность сообщений между объектами и таким образом формирует алгоритм описываемого сценария.

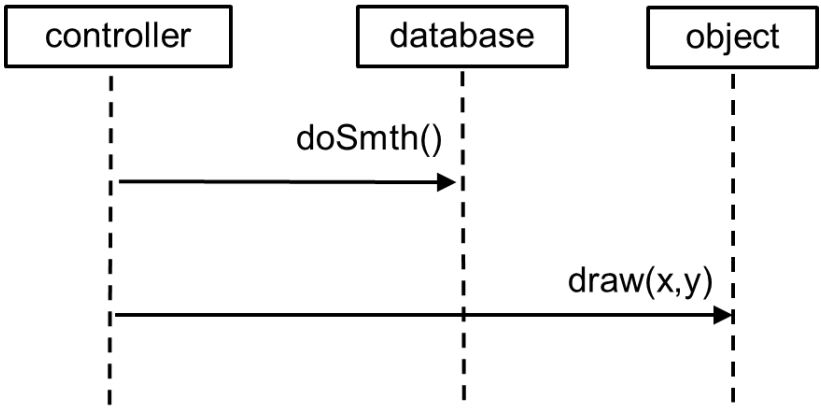
Основными элементами диаграмм последовательностей так же, как и диаграмм коммуникаций, являются участники (lifelines) и сообщения. Сообщения на диаграммах последовательностей обычно не нумеруют, а для участников изображают линию жизни в виде вертикальной пунктирной линии, которая обозначает временной период существования участника (напомним, что время на этих диаграммах течёт вертикально сверху вниз).



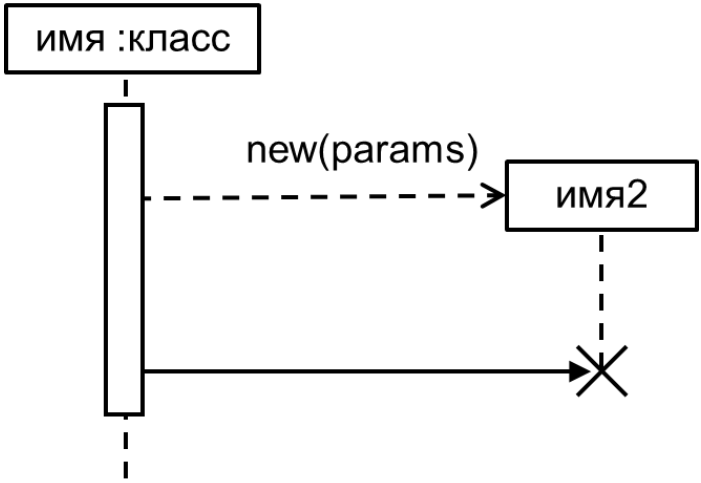
Объект f типа File вызывает операцию print у объекта printer. Имя «printer», скорее всего, говорит о том, что это объект одноименного класса Printer. При вызове передается параметр «options»

(поле объекта f: File), возвращаемое значение сохраняется в переменную (поле) success объекта f.

С другой стороны, сообщения на диаграммах последовательностей представлены более богатым набором. Кроме обычных синхронных сообщений (message(params) на рисунке) встречаются сообщения возврата (result на рисунке), которые, как правило, используются, чтобы продемонстрировать возврат управления в точку вызова и показать возвращаемое значение (result), если оно есть. Синхронные сообщения изображаются сплошной линией со стрелкой в форме закрашенного треугольника. Сообщения возврата изображаются пунктирными линиями с обычными стрелками. Фокус управления (активность) на диаграммах последовательностей изображается прямоугольником, нанесенным поверх пунктирной линии участника. Если возвращаемых значений нет и по диаграмме очень легко понять общий алгоритм передачи управления между участниками, то сообщения возврата можно не изображать. Более того, в таких случаях очень часто не изображают и прямоугольники, обозначающие активность. Например, такой подход можно использовать, когда все вызовы происходят из одного объекта и возвращаемые значения либо отсутствуют, либо не обозначены:



Сообщение создания объекта изображается пунктирной линией с обычной стрелкой, приходящей прямо к прямоугольнику создаваемого объекта. Сообщение при этом можно не именовать или, если необходимо, показать передаваемые параметры, обозначить каким-либо понятным образом, например `new` или `create`.

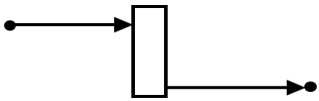


Новый созданный участник размещается на временной оси (вертикально) строго в том месте, где был создан. Таким образом, в самой верхней позиции на диаграмме размещают только участников, которые существовали ещё до начала изображенного сценария.

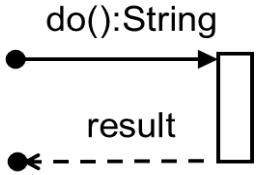
Для обозначения окончания существования участника используется большой крест, поставленный на линии жизни. При этом сама линия жизни участника дальше не продолжается. Если уничтожение происходит не автоматически (например, по завершении сценария), а принудительно, то к этому кресту проводится сообщение от того объекта, который уничтожает данного участника. Для тех же участников, которые продолжают существовать после выполнения сценария, линии жизни должны доходить до самой нижней части диаграммы.

Асинхронные сообщения означают, что посылка сообщения не вызывает ожидания ответа от адресата в точке вызова. Асинхронные сообщения обозначаются сплошной линией с обычной стрелкой на конце.

Также на диаграммах последовательностей можно часто встретить так называемые найденные и потерянные сообщения. Обозначаются они сплошной стрелкой, на одном из концов которой поставлена жирная точка, и этот конец не соединен ни с одним участником. Такие обозначения используются, чтобы изобразить сообщения, приходящие от участников или уходящие к участникам, не изображенным на диаграмме.

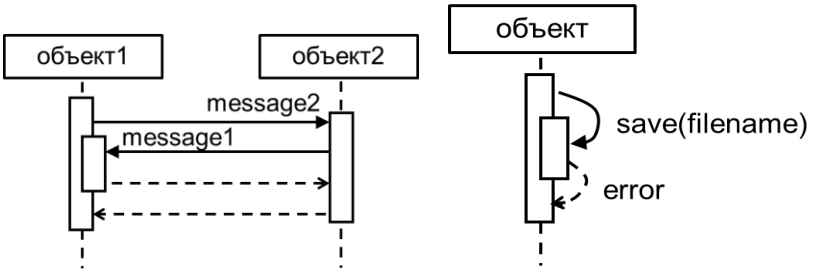


В частности, с помощью найденных сообщений удобно моделировать входящие сообщения сценария (первое сообщение), а с помощью потерянных – соответствующий возврат управления из сценария:

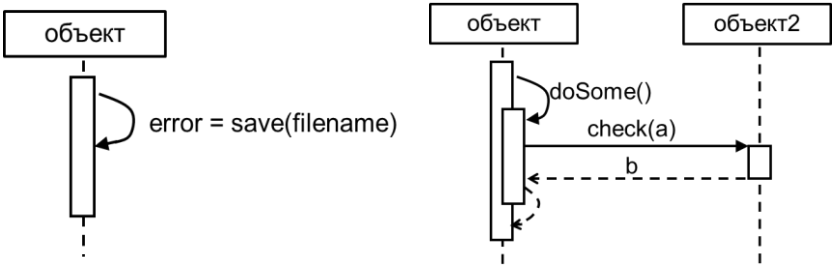


Если во время выполнения последовательности фокус управления передается объекту, уже инициировавшему взаимодействие (объекту с активным фокусом управления), то этот факт изображается с помощью вложенной активности. При этом инициатором вложенной активности у себя может выступать сам участник. Это происходит, например, при вызове внутри одного метода объекта

другого метода этого же объекта. Изображаются вложенные активности наложением нового прямоугольника активности на уже имеющийся.



Изображение вложенных активностей рекомендуется использовать, только если эти вложенные активности порождают дополнительные последовательности, изображенные на этой же диаграмме. В противном случае достаточно воспользоваться обычными сообщениями.

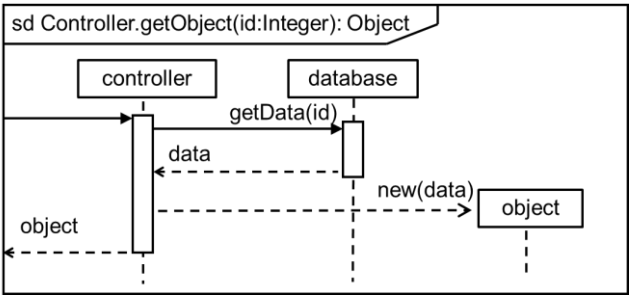


Например, у указанного метода save(filename) нет внутренних вызовов к другим объектам (по крайней мере, они не показаны на данной диаграмме), тогда вложенную активность можно опустить, а возвращаемое значение написать на входящей стрелке в качестве присваемого значения error. Напротив, у метода doSome() внутри передается управление объекту2 посредством вызова метода check(a), поэтому вложенную активность необходимо изобразить.

Более сложные элементы диаграмм последовательностей изображаются с помощью так называемых фрагментов (комбинирован-

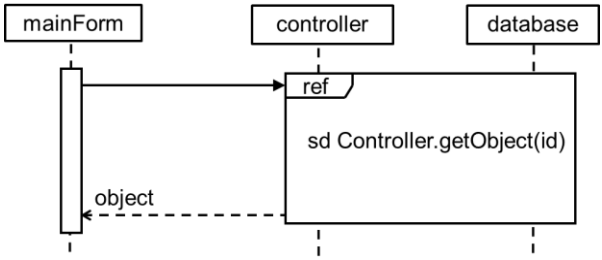
ных фрагментов). К таким элементам относятся: взаимодействия (interactions), ветвления, циклы и тому подобные конструкции.

Взаимодействие представляет собой сценарий, который может быть вложен в другой сценарий. Это очень похоже на вызов подпрограммы в языках программирования. Интерфейс и тело взаимодействия описываются в виде отдельной диаграммы последовательностей, которая в этом случае обязательно имеет рамку и формальный заголовок, с помощью которого определяется имя взаимодействия и его формальные параметры.

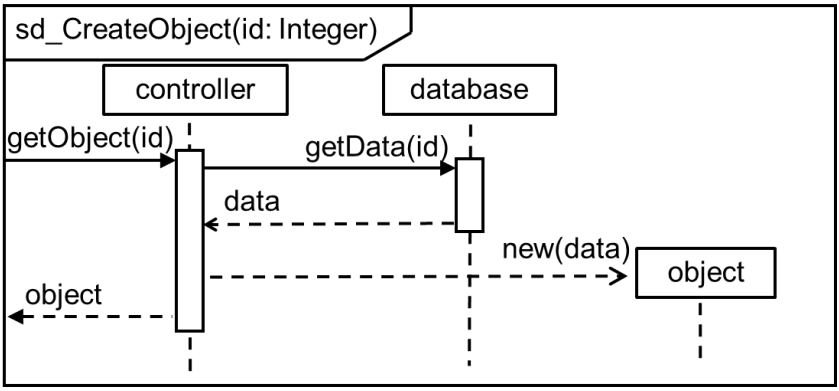


(Обратите внимание на префикс sd – Sequence Diagram, такой префикс рекомендуется использовать в названиях диаграмм взаимодействия.)

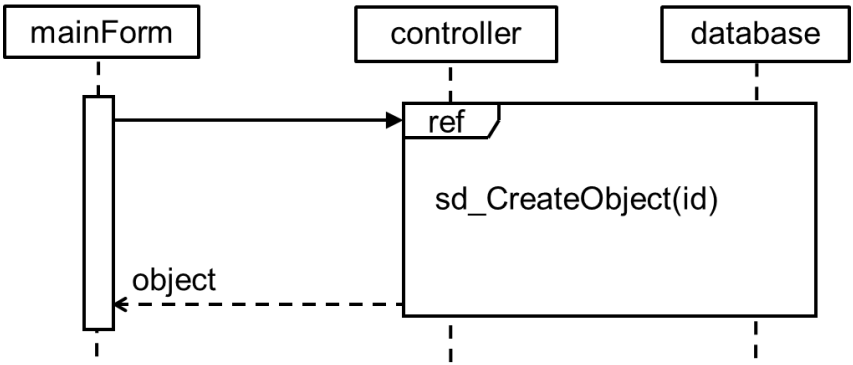
Вызов взаимодействия на другой диаграмме обозначается с помощью такой же рамки, в заголовке которой указано ключевое слово ref, а вместо тела – имя и параметры взаимодействия. При этом взаимодействие должно по возможности захватить линии жизни всех задействованных в нем участников, которые имеются на данной диаграмме.



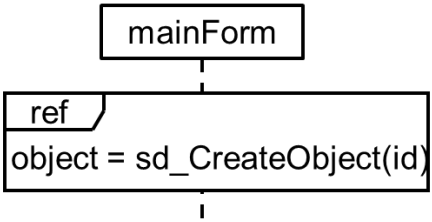
Для большего удобства использования взаимодействий применяют так называемые шлюзы – точки входа и выхода из взаимодействия. Шлюз представляет собой просто точку на рамке взаимодействия, из которой выходит поступающее сообщение или в которую уходит возвратное сообщение.



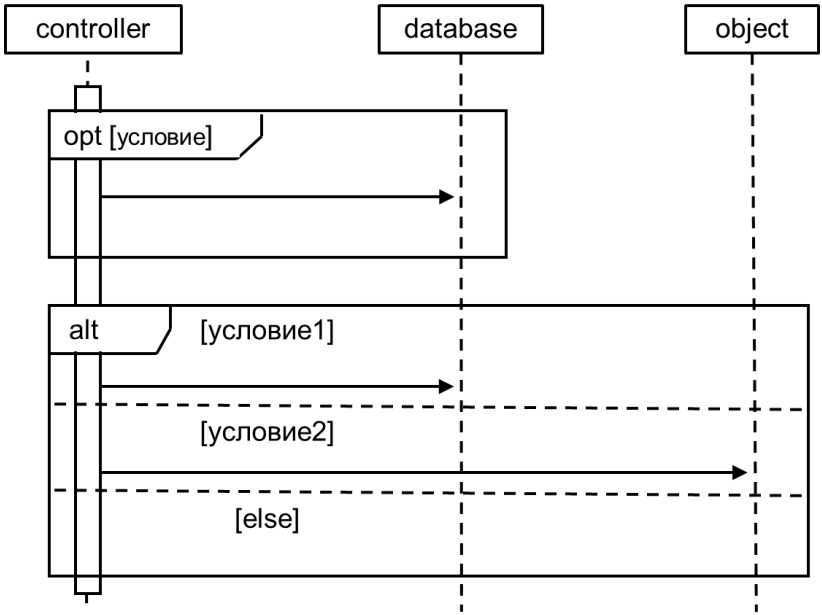
В этом случае само взаимодействие удобно именовать без привязки к конкретному исполнителю и легко использовать возвращаемые значения. Например, вот так:



Или даже так:



Ветвление в диаграммах последовательностей представлено двумя основными типами фрагментов: opt и alt. Фрагмент типа opt представляет собой простое ветвление без альтернатив – аналог конструкции `if … then …` в языке программирования. Рамка фрагмента должна охватывать все сообщения, которые должны выполняться, если сторожевое условие верно, и линии жизни всех объектов, задействованных в этом. Сторожевое условие записывается в заголовке фрагмента в квадратных скобках сразу после ключевого слова opt.



Комбинированный фрагмент alt представляет на диаграммах последовательностей ветвления с несколькими альтернативами – аналогично вложенной конструкции `if … then … else if … then … else …` или конструкции `switch` в языке программирования. Аналогично фрагменту opt рамка фрагмента alt должна охватывать все задействованные в нем сообщения и объекты. При этом обычно в заголовке фрагмента указывается лишь ключевое слово alt, а сторожевые условия для каждой альтернативы указываются в начале соответствующей секции. Данный фрагмент делится с помощью горизонтальных пунктирных линий на секции, соответствующие альтернативам ветвления. В качестве сторожевого условия для последней секции допускается использовать ключевое слово [else], которое означает, что данная секция будет выполнена, если ни одно указанное выше условие не является истинным.

Циклы на диаграммах последовательностей изображаются с помощью фрагмента loop. Так же как и прочие фрагменты, фрагмент loop должен охватывать все сообщения и участников, задействованных в исполнении цикла. В заголовке данного фрагмента после ключевого слова loop указываются параметры и условия выполнения цикла. Например:

– loop или loop * означают бесконечный цикл (как его прервать, будет описано ниже);

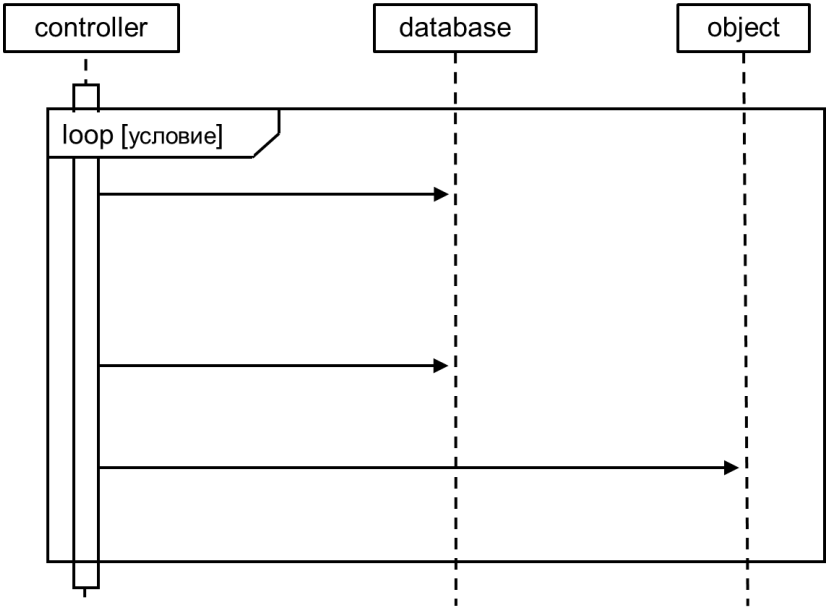
– loop (N) означает цикл, который будет выполнен ровно N раз;

– loop (N, M) цикл будет выполнен минимум N, максимум M раз;

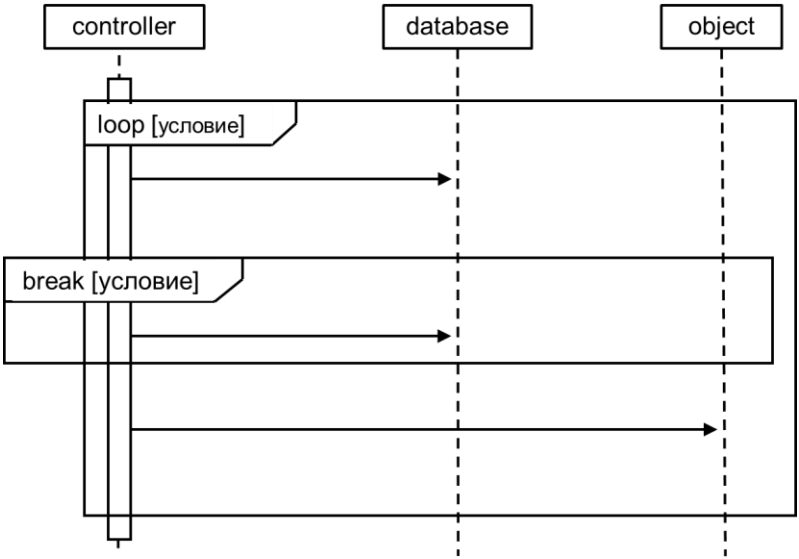
– loop [условие] выполняется, пока условие истинно;

– loop [for each объект in коллекция] будет выполнен для каждого объекта из указанной коллекции. При этом объект обязательно должен быть представлен на диаграмме и вовлечен во взаимодействие. Коллекция может быть либо представлена в явном виде, либо являться свойством одного из участников и определяться однозначно (иметь уникальное имя или являться свойством первого объекта взаимодействия).

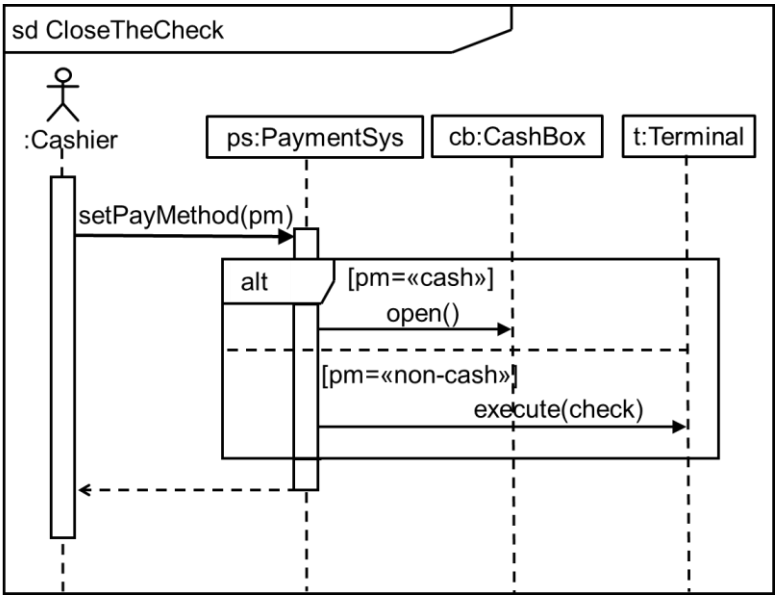
Указанные способы обозначения могут комбинироваться, например, loop (1, *) [условие] выполнится один раз, а затем будет выполняться, пока условие истинно.



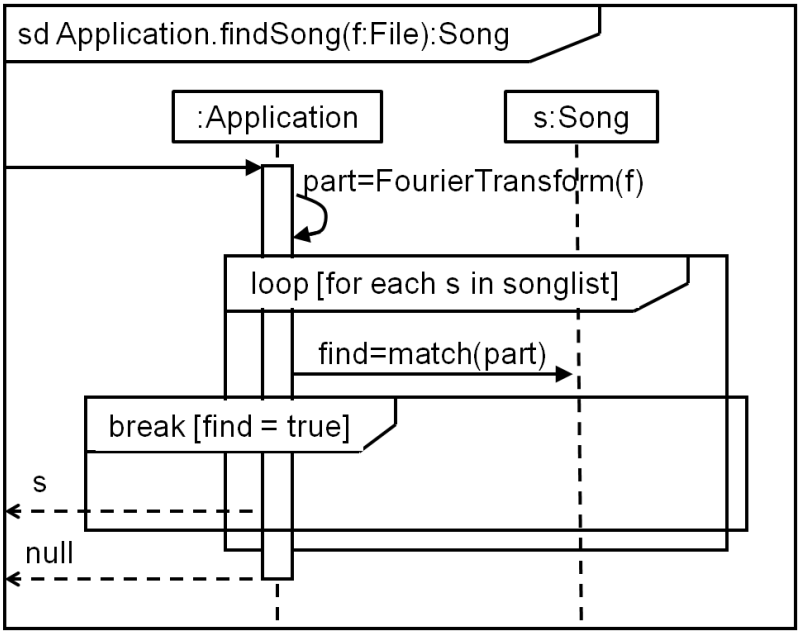
Для того чтобы представить точки и условия прерывания в комбинированных фрагментах, используется специальный фрагмент break, имеющий собственное сторожевое условие, при выполнении которого все другие фрагменты, охваченные фрагментом break, прерываются. При этом выполняются сообщения, находящиеся во фрагменте break, и происходит выход из прерванной конструкции. Рамка фрагмента break всегда изображается выходящей за пределы прерываемого им фрагмента по горизонтали (слева).



Примеры диаграмм последовательностей с ветвлениями, циклами и прерыванием изображены ниже.



На данной диаграмме кассир устанавливает (setPayMethod) в систему оплаты ps типа PaymentSys способ оплаты pm. Объект ps проверяет значение полученной переменной pm: если pm – наличные, то открывает кассу (операция open объекта cb типа CashBox), если безнал, то перенаправляет запрос на исполнение терминалу (операция execute объекта t типа Terminal).



На рисунке изображена последовательность действий для операции findSong объекта типа Application. Сначала для переданного файла f выполняется операция FourierTransform(f). Полученный результат part в цикле сравнивается с каждым объектом s типа Song посредством вызова операции match(part). Если результат сравнения успешен (find = true), то цикл прерывается и возвращается объект s. В противном случае при достижении конца цикла операция findSong вернет указатель null.

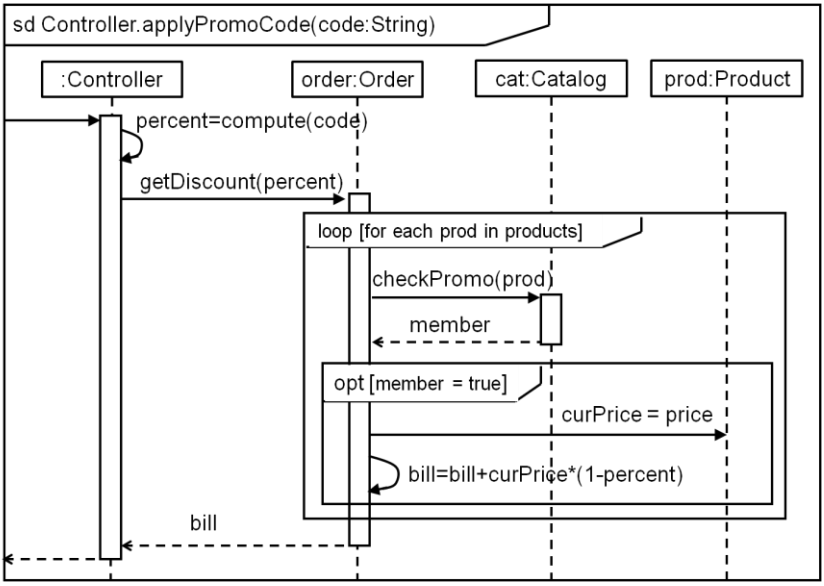
В UML 1 такие конструкции, как фрагменты, отсутствовали, поэтому для обозначения ветвлений и циклов разработчики часто использовали обозначения, аналогичные диаграммам коммуникаций: [сторожевое условие] для ветвлений и * для циклов.

Кроме указанных типов фрагментов в UML существуют и другие – для организации параллельного выполнения, определения критических секций, неверных взаимодействий и так далее, рассмотрение которых не входит в задачи данного пособия.

Как уже говорилось ранее, диаграммы коммуникаций в основном используются для создания легких эскизов на начальных этапах анализа и проектирования. В противовес им диаграммы последовательностей обычно применяются на поздних стадиях фазы проектирования и на фазе реализации процесса разработки, так как позволяют более формально описать практически все особенности алгоритма и таким образом предоставляют возможность автоматической генерации соответствующего кода.

Из любой диаграммы коммуникаций можно автоматически получить диаграмму последовательностей (и многие средства позволяют это сделать), но не наоборот, так как синтаксис диаграмм последовательностей намного мощнее синтаксиса диаграмм коммуникаций.

Пример. Детализируем последовательность действий применения промокода для скидки класса Controller операции applyPromoCode(code:String): Boolean. Покупатель вводит промокод на сайте. На стороне сервера проверяется, имеются ли товары в корзине покупателя, на которые действует промокод. Если товар участвует в акции, то его стоимость пересчитывается. Покупателю возвращается итоговая стоимость заказа с учетом скидки.



На диаграмме изображена последовательность сценария applyPromoCode класса Controller. Первым действием объект Controller вычисляет значение переменной percent на основе полученного code. Затем вызывает операцию getDiscount() у объекта order одноименного класса, передавая в качестве параметра вычисленную переменную percent. При выполнении данной операции у объекта order в цикле по всем элементам prod коллекции products запрашивается операция checkPromo() – проверка на участие в акции, результат которой сохраняется в переменную member. (Обратите внимание, что коллекция products – атрибут класса Order.)

Далее в том же цикле переменная member проверяется на соответствие условию равенства значению true. При выполнении условия у объекта prod типа Product запрашивается его атрибут price, значение которого сохраняется в переменную curPrice класса Order. Затем атрибут bill класса Order пересчитывается на основе формулы, приведенной на диаграмме.

После окончания цикла объект order передает объекту Controller пересчитанную переменную bill.

Контрольные вопросы

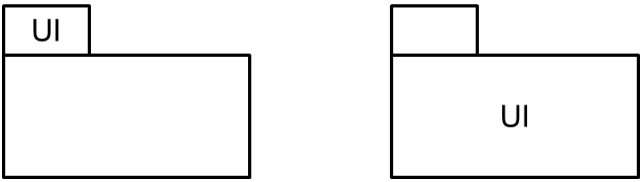
1. Для чего используются диаграммы последовательностей? Чем их применение отличается от диаграмм коммуникаций?
2. Как изображаются участники на диаграммах последовательностей? Что такое линия жизни участника и как она изображается? Что такое активность на диаграмме последовательностей?
3. Как на диаграмме последовательностей показать точки создания и уничтожения объектов?
4. Что такое синхронные, асинхронные, возвратные, найденные и потерянные сообщения? Как они изображаются? Что такое шлюзы?
5. Что такое вложенная активность? Как ее изобразить на диаграмме последовательностей? Когда ее нужно изображать, а когда не стоит?
6. Что такое элемент «взаимодействие» для диаграмм последовательностей, как его задать и использовать?
7. Как изобразить ветвление на диаграмме последовательностей?
8. Как изобразить цикл на диаграмме последовательностей? Как изобразить прерывание цикла?

Задания

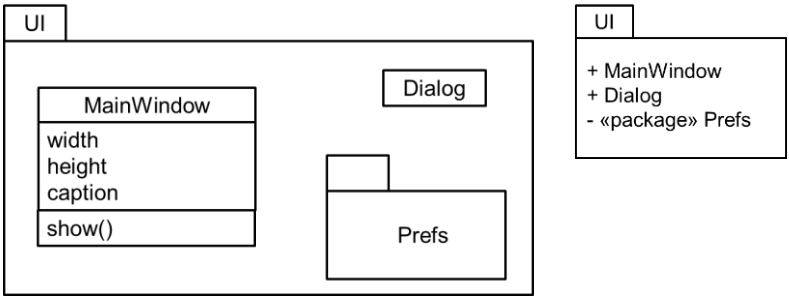
1. Воспользуйтесь диаграммой классов из примера к главе 1. Для одной из операций постройте диаграмму последовательностей:
   а) количество взаимодействующих объектов не менее 3;
   б) диаграмма должна содержать фрагмент (цикл или условие);
   в) один объект совершает вызов собственной операции.
2. По диаграмме последовательностей напишите код на языке программирования.
3. По коду на языке программирования составьте диаграмму последовательностей.

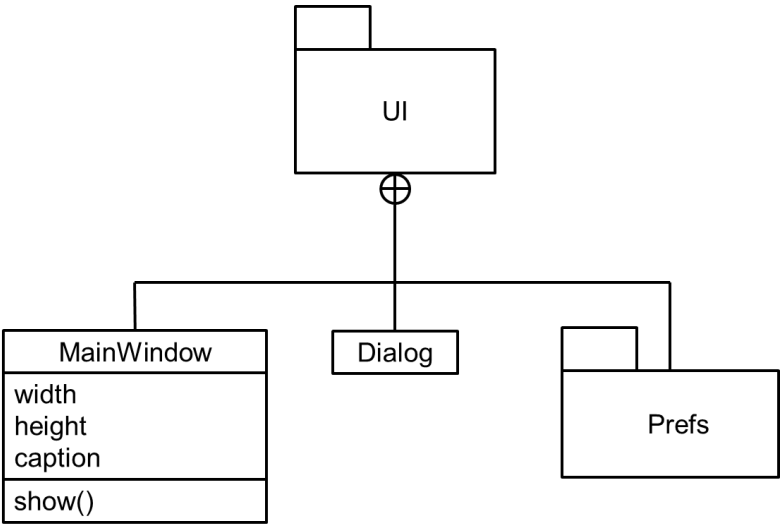
5. Диаграммы пакетов

Пакет – группирующая сущность UML. Позволяет объединить элементы модели в одну группу и продемонстрировать общие свойства этой группы. Пакет изображается в виде папки – прямоугольника, сверху которого нарисован еще один маленький прямоугольник («корешок»). Имя пакета пишется либо на корешке, либо в большом прямоугольнике.



Содержимое пакета обычно изображается внутри него в виде диаграммы или списком. Допускается также показать включение элементов с помощью специального обозначения – сплошной линии, на конце которой со стороны пакета изображен кружок с символом «+» внутри. Часто обозначения внутренних элементов пакета имеют модификатор видимости, указывающий на доступность соответствующего элемента по отношению к внешним пакетам.



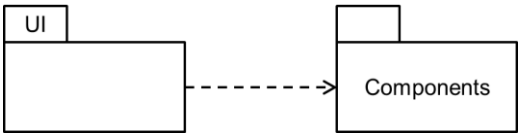


В UML допускается, чтобы каждый элемент входил только в один пакет модели. Это касается и самих пакетов, при этом не допускается самовключений и циклических включений. Принадлежность элемента некоторому пакету можно также кратко обозначить символом «::»:

```
UI::Dialog.
```

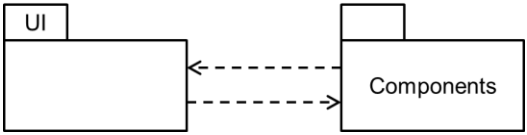
Так как в разных пакетах могут встречаться элементы с одинаковыми именами, то отличить один элемент от другого можно по полному наименованию элемента с указанием всех вложенных пакетов, к которым он относится.

Диаграммы пакетов изображают пакеты, их содержимое и зависимости между ними. Зависимости между пакетами бывают различных видов (стереотипов), но наиболее распространенной является зависимость использования «use», которая подразумевается по умолчанию. Зависимость между пакетами означает, что хотя бы один элемент одного пакета зависит хотя бы от одного элемента другого пакета.



На данной диаграмме изображена зависимость пакета UI от пакета Component. Это означает, что в пакете UI есть элемент или элементы, которые зависят (используют) какой-то элемент (или элементы) пакета Components.

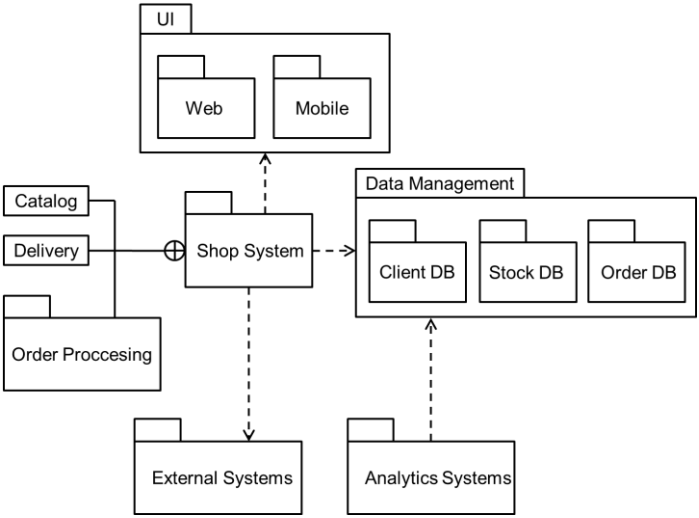
Диаграммы пакетов отражают общее устройство (структурную составляющую архитектуры) системы. Очень важно следить и стараться избегать взаимных и циклических зависимостей между пакетами:



Так делать не надо!

В языках программирования пакетам обычно соответствуют пространства имен.

Пример. Спроектировать схему взаимодействующих логических частей системы онлайн-магазина.



Для примера на диаграмме показаны составляющие элементы пакета ShopSystem: пакет OrderProcessing и классы Catalog и Delivery. Главная по функциональности часть ShopSystem взаимодействует с пакетом графического интерфейса UI для веб- и мобильной вёрстки, а также использует внешние системы, например, для доставки заказов и сервис регистрации клиентов. Все необходимые данные по складу, товарам, заказам, покупателям и прочему хранятся в пакете DataManagement, доступ к которому имеют ShopSystem и AnalyticsSystem для формирования статистики по совершенным заказам и для прогнозирования роста или упадка спроса на товары определенной категории.

Контрольные вопросы

1. Что такое пакет в UML? Как он изображается?
2. Как показать содержимое пакета? Как показать принадлежность элемента пакету?
3. Что означает зависимость между пакетами? Приведите пример.

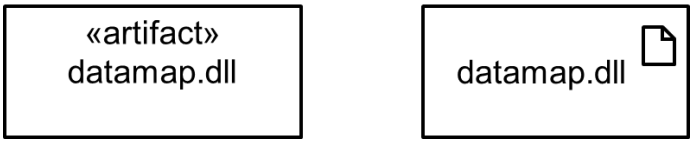
Задание

Составьте диаграмму пакетов для своего проекта. Проанализируйте отсутствие взаимных зависимостей.

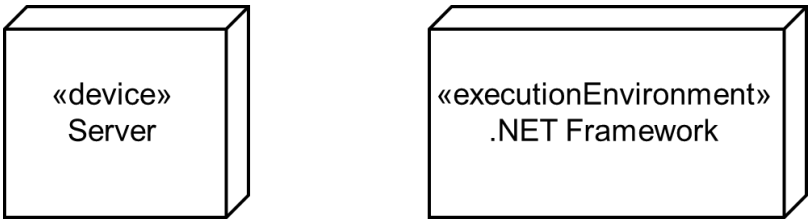
6. Диаграммы развертывания (размещения)

Диаграммы развертывания UML представляют физическое размещение артефактов системы на ее узлах.

Артефакт – элемент информации, который используется или произведен в процессе разработки программного обеспечения или функционирования системы. Обычно на диаграммах развертывания изображают только файлы, необходимые для исполнения приложения: исполняемые файлы .exe, библиотеки .dll, сценарии (скрипты), файлы с данными (в том числе и база данных) или настройками и т.п. Артефакт изображается в виде классификатора (прямоугольника) с ключевым словом «artifact» либо с иконкой артефакта (маленький лист бумаги) в правом верхнем углу.

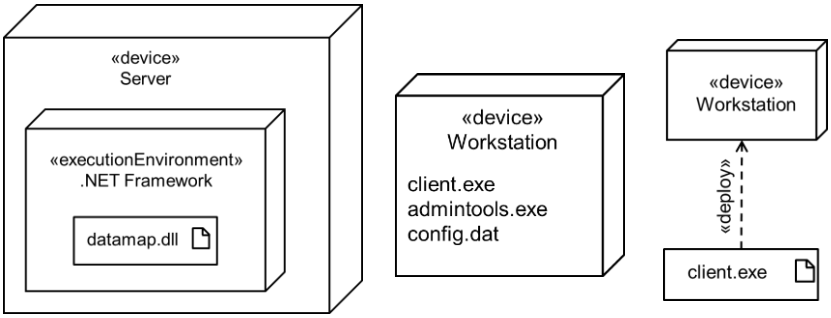


Узел – вычислительный ресурс, в котором могут располагаться и исполняться артефакты системы. Изображается узел в виде прямоугольного параллелепипеда.



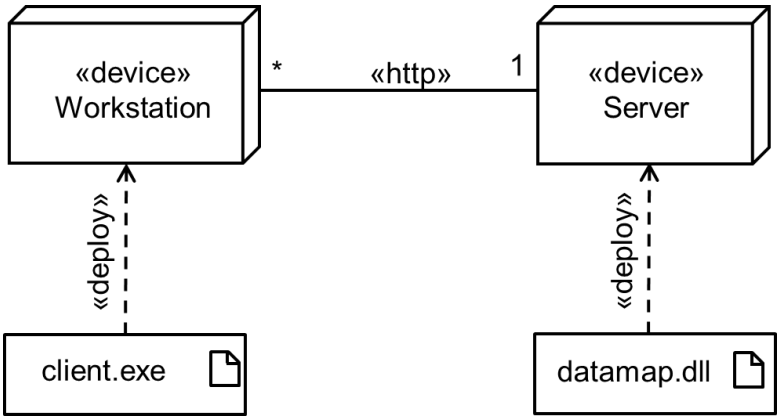
Различают два вида узлов: устройства (помечаются ключевым словом «device») и среды исполнения («executionEnvironment»). На диаграммах развертывания устройства и среды исполнения могут в себе содержать среды исполнения и артефакты. Размещение

артефактов можно также показать в виде списка или с помощью зависимостей «deploy», проведенных от артефакта к узлу.



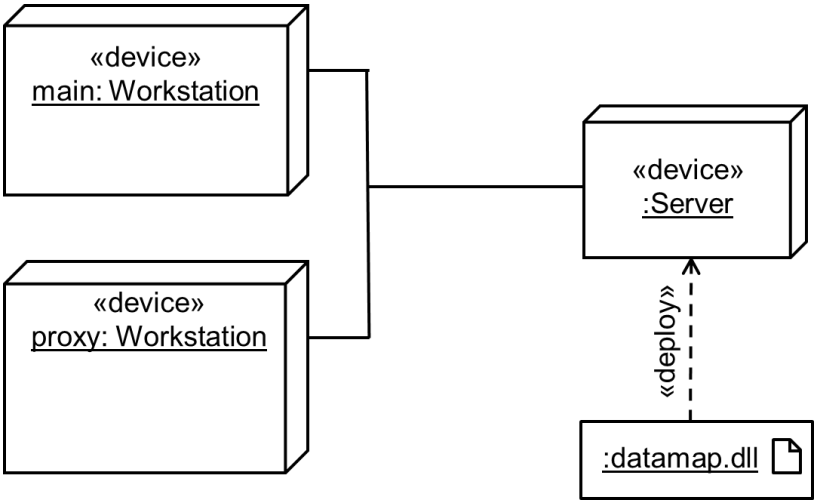
Таким образом, диаграммы развертывания показывают, какие компоненты разработанного и используемого программного обеспечения на каких устройствах и в каких средах исполнения должны быть размещены.

Узлы связывают между собой линиями взаимодействия (Communication Path), на которых можно показать кратность связи и протокол, по которому осуществляется связь.



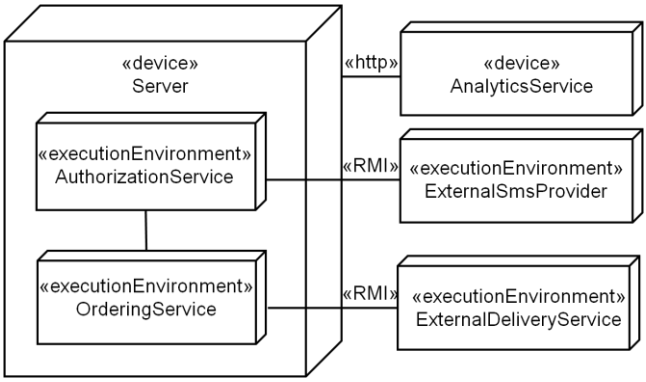
Обычно в одном проекте присутствует одна диаграмма развертывания, либо ее вообще нет, если развертывание тривиально (например, разрабатывается локальное приложение).

Диаграммы развертывания, как правило, рисуются на уровне классификаторов (типов), но также развертывание можно изображать на уровне экземпляров узлов и артефактов. Такие диаграммы демонстрируют конкретику размещения и используются непосредственно перед самим физическим развертыванием системы или для того, чтобы задокументировать топологию имеющегося размещения.



В данном примере показано, что конфигурация системы подразумевает два экземпляра узлов Workstation (main и proxy) и один сервер, на котором расположен файл datamap.dll.

Пример. Представить физическую схему размещения функциональных частей системы.



Модули авторизации в системе заказов и модуль оформления заказов расположены на одном сервере, который задействует внешний сервис sms-провайдеров для авторизации в системе по номеру телефона клиента, внешний сервис доставки, а также сервер аналитики и статистики.

Контрольные вопросы

1. Для чего используются диаграммы развертывания?
2. Что такое артефакт? Как он изображается на диаграмме развертывания? Какие артефакты стоит изображать на диаграммах развертывания?
3. Что такое узлы? Как они изображаются на диаграмме развертывания? Какие виды узлов бывают и как они связаны с артефактами и другими видами узлов?
4. Для чего используются диаграммы развертывания, построенные на уровне классификаторов, а для чего – на уровне экземпляров?

Задание

Составьте диаграмму развертывания для вашего проекта.

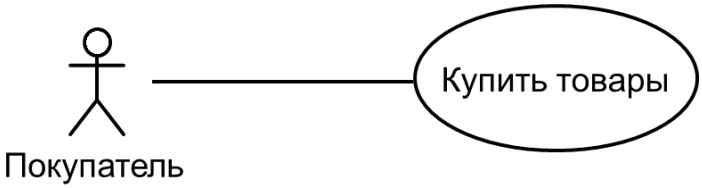
7. Варианты использования (прецеденты)

Вариант использования или прецедент (use case) – это единица поведения системы, результат которой значим для некоторого заинтересованного лица. Варианты использования применяются для моделирования требований к системе, а их сценарии – для моделирования поведения системы.

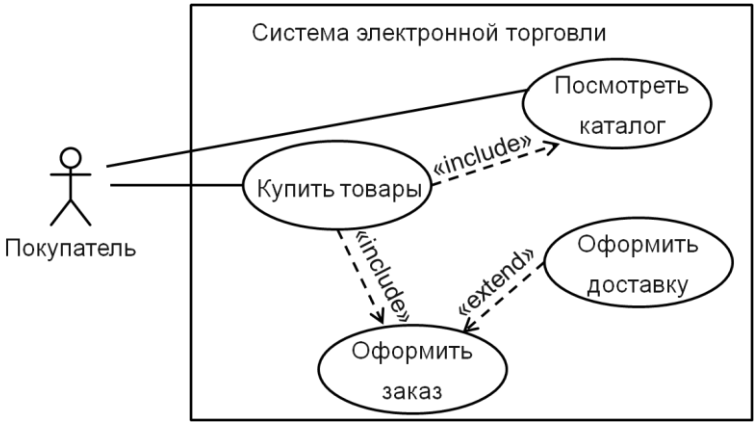
С помощью вариантов использования моделируются так называемые функциональные требования – функции системы. Эти требования отличаются от нефункциональных тем, что предполагают какую-то работу системы, некоторый сценарий, по которому пользователь (действующее лицо) взаимодействует с системой для получения результата. Например, пусть разрабатывается программная система для электронной торговли. Типовое заинтересованное лицо – это покупатель. Он хочет купить товары в нашем магазине. Купить товары – это его цель, т.е. результат, который он хочет достичь. С другой стороны, процесс покупки товаров подразумевает некоторый сценарий взаимодействия пользователя (покупателя) с системой. Покупатель в этом сценарии – не только заинтересованное, но и действующее лицо. Именно он инициирует сценарий покупки товара и взаимодействует с системой, пока этот сценарий не будет завершен (например, товары будут куплены).

В UML действующих лиц называют актёрами (actor). Это могут быть не только пользователи, но и другие компьютерные системы, взаимодействующие с данной системой.

Изображаются актёры в виде человечка, под которым подписывают его роль, варианты использования изображаются в виде овалов (эллипсов), внутри которых пишется название сценария. Рекомендуется присваивать вариантам использования имена в глагольной форме. Связывают актёров с вариантами использования, которыми они могут воспользоваться (которые они могу инициировать), с помощью ассоциации.



Часто, чтобы подчеркнуть границы системы (что она делает, а чего не делает, что в нее входит, а что – нет), рисуют прямоугольную рамку с названием системы.



Если у системы всего один актёр, то его можно не изображать, очертив лишь функциональность системы.

Как уже говорилось, каждый вариант использования предполагает некоторый сценарий взаимодействия актёра с системой. Например, сценарий «Оформить заказ» в свободной форме можно описать так:

> Покупатель оформляет заказ сформированной корзины с отобранными товарами. Сначала покупатель выбирает способ получения: самовывоз из пункта выдачи или доставка на дом. После ввода данных о месте, куда должны доставить заказ, покупатель заполняет страницу оплаты: заполняет данные получателя, а также спо-

соб оплаты – банковской картой онлайн или при получении наличными/картой. При желании можно воспользоваться персональным промокодом на скидку. При выборе способа оплаты «банковской картой онлайн» система перенаправляет пользователя на страницу авторизации карты. Пользователь вводит данные карты, и при успешной оплате система выводит информацию об окончании формирования заказа и сроках для его получения. При выборе способа «оплата при получении» система отображает пользователю информацию, что заказ сформирован.

Для описания сценариев вариантов использования применяются различные методики и нотации: одно- и двухколоночные таблицы (не UML) или диаграммы коммуникаций, последовательностей, деятельности (UML). Один вариант использования может подразумевать под собой множество сценариев, которые объединены общей целью («Купить товары»), даже если эта цель в каких-то из них не достигается (например, товара нет или не хватает денег на счете). Всегда существует типовой сценарий (обычно один), по которому должно происходить взаимодействие в большинстве случаев. Он называется основным или главным успешным сценарием. Все остальные сценарии варианта использования являются вспомогательными или альтернативными сценариями. Они называются расширениями. Вот пример основного сценария «Оформить заказ» с предполагаемыми расширениями.

Вариант использования: «Оформить заказ».

Описание: пользователь хочет оформить товары в корзине к заказу, оплатить и получить заказ.

Предусловие: Покупатель находится на странице с корзиной. В корзину добавлен один и более товаров.

Основной сценарий:

1. Покупатель выбирает команду «Перейти к оформлению».

2. Система открывает страницу «Оформление заказа. Шаг 1 из 2».

Примечание: по умолчанию способ доставки – «самовывоз».

3. Покупатель выбирает магазин – пункт выдачи заказа для самовывоза.

4. Система отображает информацию о магазине.

5. Покупатель выбирает команду «Заберу отсюда».

6. Система отображает страницу оформления заказа с выбранным пунктом выдачи.

7. Покупатель выбирает команду «Далее».

8. Система открывает страницу «Оформление заказа. Шаг 2 из 2».

Примечание: по умолчанию способ оплаты – «банковской картой».

9. Покупатель вводит информацию о получателе. Нажимает «Оплатить».

10. Система перенаправляет запрос на страницу для ввода реквизитов карты.

10.1. Оплата успешна, система открывает страницу заказа со статусом «сформирован» и примерными датами его получения в пункте выдачи.

10.2. Оплата не прошла. Переход на шаг 8.

Альтернативный сценарий 1:

1. **Заменяет шаги 3–7 Основного сценария.** Начинается, когда на шаге 3 Покупатель выбирает способ получения «Доставка».

1.1. Система открывает страницу «Доставка».

1.2. Покупатель вводит данные об адресе доставки: улица, дом, квартира, подъезд, этаж, комментарии курьеру (опционально).

1.3. Система проверяет корректность ввода адреса, отображает данные для заполнения «Дата и время доставки».

1.4. Покупатель выбирает дату и время доставки из предложенных вариантов. Переход к шагу 8 Основного сценария.

Альтернативный сценарий 2:

2. **Заменяет шаги 9–10 Основного сценария.** Начинается, когда на шаге 9 Пользователь выбирает способ оплаты «при получении».

2.1. Система отображает страницу для способа оплаты «при получении».

2.2. Покупатель выбирает команду «Заказать».

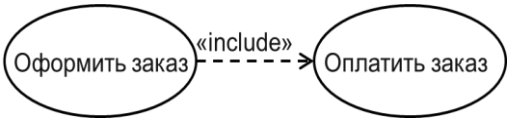
2.3. Система отображает страницу заказа со статусом «сформирован».

Диаграмма вариантов использования не несет особой смысловой нагрузки относительно конструирования системы, а исполь-

зуется для описания функций системы и представления действующих лиц. Применяется, когда есть связи (зависимости и обобщения) между вариантами использования или между актерами (обобщения).

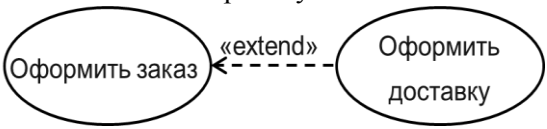
На диаграмме вариантов использования между самими вариантами использования возможны следующие отношения:

– включение – означает что сценарий одного варианта использования (А) включает в себя (в обязательном порядке) сценарий другого варианта использования (В). Изображается пунктирной стрелкой с ключевым словом «include» от варианта использования А к варианту использования В:



Невозможно оформить заказ, не оплатив его. Поэтому сценарий варианта использования «Оформить заказ» обязательно включает в себя полный сценарий «Оплатить заказ»;

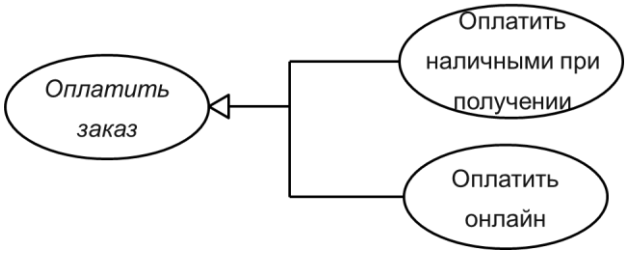
– расширение – означает, что сценарий одного варианта использования (А) может быть (т.е. необязательно) запущен во время исполнения сценария другого варианта использования (В). Изображается пунктирной стрелкой с ключевым словом «extend» от варианта использования А к варианту использования В:



При оформлении заказа может возникнуть потребность воспользоваться доставкой. Это действие необязательное, поэтому сценарий «Оформить заказ» может быть расширен действиями сценария «Оформить доставку»;

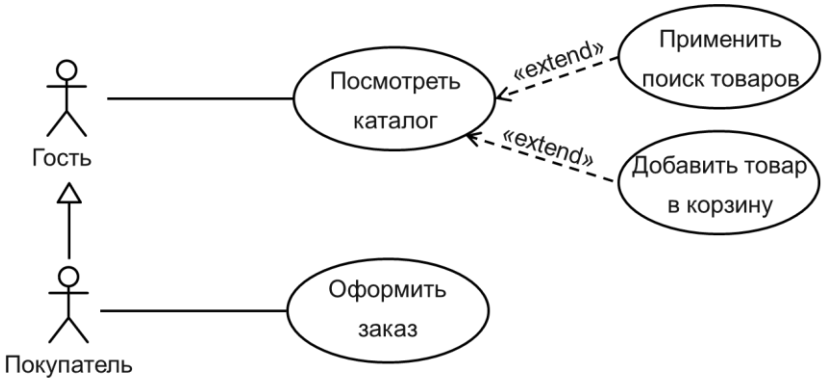
– обобщение – означает, что один вариант использования (А) реализует сценарий для функции системы, описанной с помощью другого варианта использования (В). Изображается сплошной ли-

нией со стрелкой в виде треугольника, проведенной от варианта использования A к варианту использования B. При этом считается, что вариант использования B является абстрактным – он лишь определяет функцию, но не имеет сценария, а следовательно, и не может иметь экземпляров.



Вариант использования «Оплатить заказ» является абстрактным, т.е. не имеет сценария, но может предполагать общие шаги для конкретных реализаций «Оплатить наличными при получении», «Оплатить онлайн».

Между актёрами на диаграммах вариантов использования также возможны отношения, причем только одного вида – обобщение. С помощью обобщения актёров показывают, что актёр-потомок может участвовать во всех прецедентах, что и актёр-предок, и, кроме того, может иметь и ассоциации с другими вариантами использования.

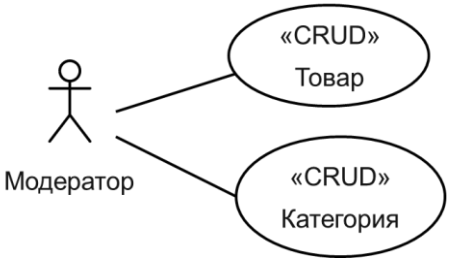


Незарегистрированный пользователь (Гость) также может воспользоваться системой интернет-магазина, но с ограниченной функциональностью – искать и добавлять товары в корзину, но оформить данный заказ может только зарегистрированный пользователь (Покупатель). Таким образом, получается, что актёр Покупатель расширяет функциональность актера Гость, т.е. он может выполнять те же функции, что и Гость («Посмотреть каталог»), но и еще что-то («Оформить заказ»), что актёру Гость не доступно.

Важно! Нужно различать роли актёров и обобщение действий. Кассир точно так же может «Купить товар», как и Покупатель. Однако при этом кассир – физически один и тот же человек – будет выступать уже в роли покупателя, поэтому обобщать этих актёров не имеет смысла. Обобщение возможно для близких по смыслу и функциональности ролей.

Замечание. Варианты использования «Зарегистрироваться», «Авторизоваться» и тому подобные, как правило, не отображаются на диаграммах вариантов использования, так как у конечного пользователя нет цели просто зарегистрироваться в системе или войти в нее, это скорее вынужденная мера, навязанная системой.

CRUD. В информационных системах почти всегда присутствует функциональность просмотра, редактирования и удаления объектов некоторых классов.



Чтобы отразить эти функции на диаграмме вариантов использования, но при этом не плодить множество примитивных прецедентов, используют обозначение «CRUD» в имени варианта использования или в качестве его стереотипа. Аббревиатура CRUD

расшифровывается как Create, Retrieve, Update, Delete – создание, чтение (просмотр), обновление (редактирование), удаление.

Для системы онлайн-торговли Модератор должен поддерживать в актуальном состоянии список товаров и их категории, отслеживать и вносить все изменения.

Хочется обратить внимание, что варианты использования – мощный механизм моделирования функций и сценариев, т.е. рекомендуется их применять всегда. С другой стороны, диаграммы вариантов использования следует применять только для отображения отношений между вариантами использования или между актерами, т.е. только в случаях, если есть такие отношения и диаграмма помогает их понять. В простейших же ситуациях рекомендуется пользоваться обычным текстовым представлением вариантов использования (списки).

Пример. Приведем возможный вариант общей диаграммы вариантов использования для описанной системы интернет-магазина.

Для данной системы предусмотрено три актёра: модератор, гость и пользователь.

Модератор – администратор системы, должен поддерживать в актуальном состоянии список товаров и их категории, отслеживать и вносить все изменения (CRUD – create, retrieve, update, delete).

Гость – незарегистрированный пользователь, который может посмотреть каталог товаров и при желании применить поиск/фильтрацию в каталоге, а также добавить товар в корзину.

Описанная функциональность расширяется для зарегистрированного Пользователя сценарием «Оформить заказ», который включает в себя «Оплатить заказ». Оплата может производиться двумя способами: наличными при получении или онлайн в системе. При оформлении заказа опциональными действиями являются «Применить промокод на скидку» и «Оформить доставку».

Контрольные вопросы

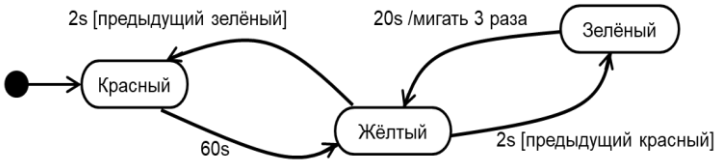
1. Что такое вариант использования?
2. Кто такой актёр?
3. Как варианты использования и актёры изображаются на диаграммах UML?
4. Приведите пример сценария варианта использования.
5. Какие отношения между вариантами использования изображаются на диаграммах UML?
6. Какие отношения между актёрами изображаются на диаграммах UML? Какие особенности следует учитывать при моделировании обобщения между актёрами?
7. Что такое CRUD?

Задание

Придумайте свою систему или проанализируйте имеющиеся приложения или веб-сайты: выделите функциональные требования, оформите их в виде диаграммы вариантов использования. Для нескольких прецедентов напишите сценарии выполнения.

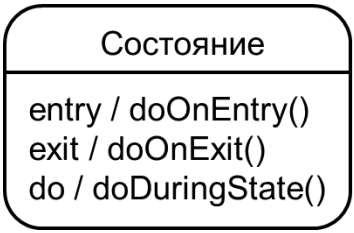
8. Диаграммы состояний

Диаграмма состояний (State Machine diagram) отображает состояния одного объекта или системы (подсистемы) в течение его/ее жизни. Основные элементы диаграммы состояний – это состояния (изображаются прямоугольниками с закругленными углами или боковыми сторонами) и переходы между ними (стрелками).



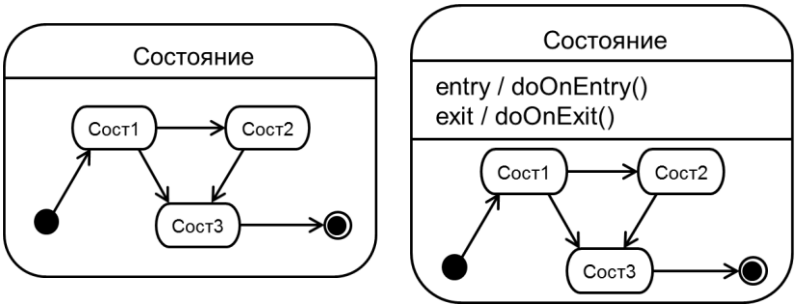
Состояния могут изображаться в упрощенном виде – только идентификатор состояния:

либо в развернутом – с указанием обработчиков в формате «событие/обработчик», которые указываются следующим образом:

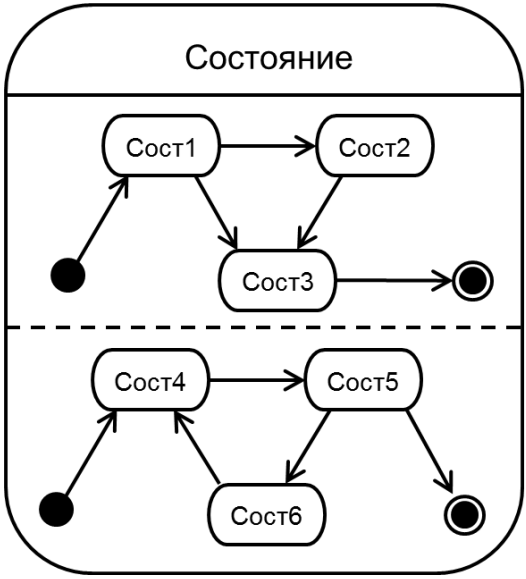


Для событий в UML определены три ключевых слова: «entry», «exit», «do», которые означают, что соответствующий обработчик будет выполнен на входе в состояние, на выходе из состояния либо будет выполняться всё время, пока объект находится в данном состоянии. Разработчики могут определять собственные события.

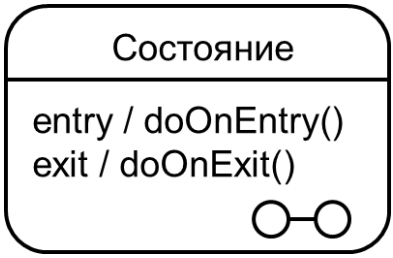
Кроме того, композитные состояния, состоящие из подсостояний со своими переходами, могут изображаться содержащими внутри себя целую диаграмму своих подсостояний:



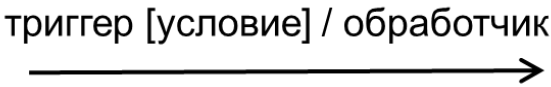
а также диаграмму с несколькими параллельными потоками в случае одновременного (параллельного) изменения нескольких цепочек вложенных подсостояний:



либо в свернутом виде, чтобы подчеркнуть, что данное состояние является композитным:



Переходы между состояниями изображаются сплошными стрелками с обозначением триггера, сторожевого условия и обработчика:



причем любая часть подписи может отсутствовать, в том числе может отсутствовать и вся подпись, но обычно имя триггера присутствует почти всегда. Триггер обозначает, какое событие приводит к данному переходу, сторожевое условие говорит, при каком условии возможен переход, а обработчик – какая операция выполняется в момент перехода.

Считается, что переходы на диаграммах состояния происходят мгновенно и не могут быть прерваны, даже когда выполняется обработчик.

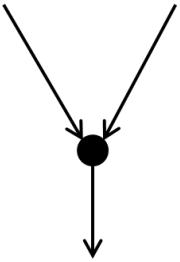
Также важными элементами диаграмм состояний являются так называемые псевдосостояния. Псевдосостояния отличаются от обычных состояний тем, что объект не находится в этих состояниях, но переходы в них или из них осуществляются. Ниже перечислим основные применяемые псевдосостояния:

Начальное состояние используется для обозначения точки входа на диаграмму или, другими словами, начала работы машины состояний. Должно присутствовать на диаграмме в единственном числе

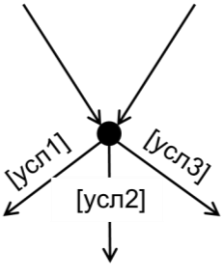
Конечное состояние используется для обозначения точки выхода из диаграммы или, другими словами, окончания работы машины состояний. На диаграмме может быть несколько конечных состояний

Точка входа используется для обозначения точки входа в композитное состояние

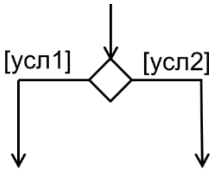
Точка выхода используется для обозначения точки выхода из композитного состояния



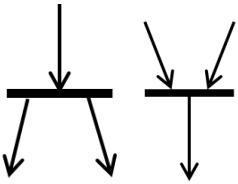
Соединение (junction) используется для соединения потоков перехода в единый переход (например, чтобы обозначить одинаковые триггер, сторожевое условие, обработчик или соединение потоков после ветвления)



Псевдосостояние соединения может также использоваться и для разветвления при указании соответствующих сторожевых условий переходов

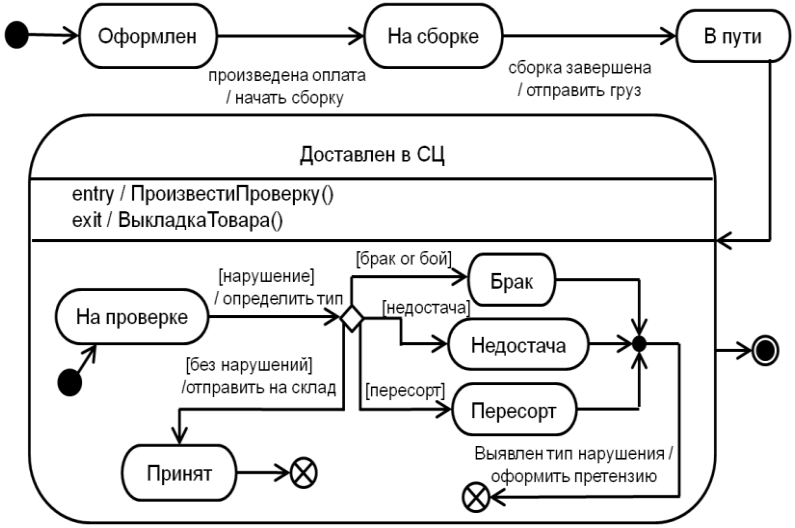


Ветвление (choice) используется, чтобы обозначить возможные пути переходов при различных условиях



Разветвление (fork) и объединение (join) применяются для распараллеливания и соединения потоков одновременного изменения состояний

Пример. Товары для последующей реализации в торговой точке закупаются оптом у поставщиков. Изначально оформляется заявка с перечислением необходимых товаров и их количеством, согласовывается цена покупки-продажи. После того как заказчиком произведена оплата, поставщик отправляет заказ на сборку. По завершении сборки происходит погрузка и транспортировка груза до заказчика. Груз поставляется в сортировочный центр (СЦ), где проверяется соответствие номенклатуры поставки и заявки, а также качества товара. Если нарушений не выявлено, груз передается на склад, а далее – по запросу на торговую точку. В случае выявления нарушения – брак, недостача, пересорт – оформляется претензия на имя поставщика о несоответствии номенклатуры и качества груза.



На диаграмме представлены переходы состояний возможного груза – от оформления заявки до поставки товаров. Из начального состояния переходим в состояние «Оформлен». Триггер «произведена оплата» запускает обработчик «начать сборку», и объект переходит в состояние «На сборке». По завершении сборки запускается обработчик отправки груза – переход в состояние «В пути».

Далее переход в композитное состояние «Доставлен в СЦ», при входе в которое необходимо выполнить действие «Произвести Проверку()». Груз сверяется с номенклатурой поставки, и если нарушений нет, груз отправляется на склад – переход в состояние «Принят». При успешном выходе из данного композитного состояния выполняется действие «ВыкладкаТовара()».

При выявлении нарушений, определяется его тип – отдельные состояния «Брак», «Недостача», «Пересорт», переходы из этих состояний сливаются в единый узел соединения по общему триггеру и обработчику, так как необходимо оформить определенную претензию на свой тип нарушения.

Контрольные вопросы

1. Для чего применяются диаграммы состояний UML?
2. Как изображаются состояния, какой смысл они имеют?
3. Что такое композитное состояние? Как оно изображается?
4. Приведите пример композитного состояния.
5. Приведите пример композитного состояния с параллельными потоками.
6. Что такое переход на диаграмме состояний, как он обозначается? Назовите его основные элементы.
7. Что такое псевдосостояния? Почему они так называются?
8. Как обозначаются начальное и конечное псевдосостояния, что они означают? Приведите пример с данными псевдосостояниями.
9. Как обозначаются псевдосостояния точек входа и выхода, что они означают? Приведите пример с данными псевдосостояниями.
10. Как обозначаются псевдосостояния ветвления (choice) и соединения (junction), что они означают? Приведите пример с данными псевдосостояниями.
11. Как обозначаются псевдосостояния разветвления (fork) и объединения (join), что они означают? Приведите пример с данными псевдосостояниями.

Задание

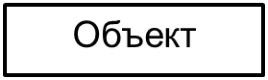
Составить диаграмму состояний для своего проекта. Обратите внимание, что довольно часто путают диаграммы состояний и деятельности. Помните: на диаграмме состояний отображены изменения состояний одного объекта вашей системы.

9. Диаграммы деятельности

Диаграммы деятельности UML используются для изображения алгоритмов и потоков передачи данных. Эти диаграммы во многом похожи на обычные блок-схемы, но в отличие от последних допускают параллельные потоки управления.

Основные элементы диаграмм деятельности – это действия, узлы управления, объекты, потоки передачи данных и управления.

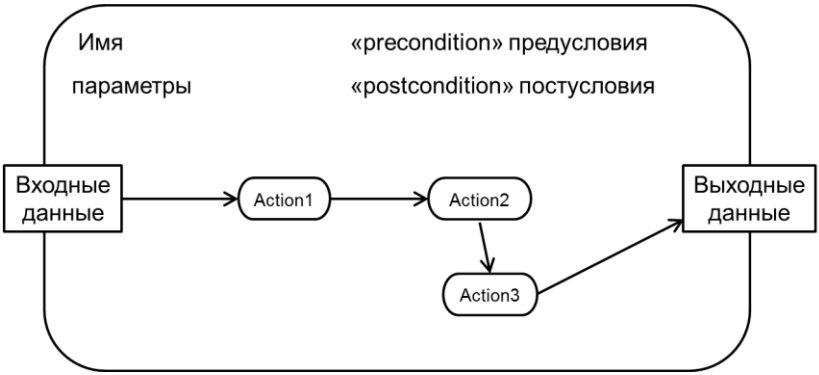
Действие (action) изображается в прямоугольнике с закругленными краями. Обозначает некоторое поведение (behavior), действие, выполняемое в ходе исполнения изображенной на диаграмме деятельности



Объект изображается в виде прямоугольника и обозначает некую сущность (обычно – переноса данных), которая задействована в процессе исполнения алгоритма. Рекомендуется использовать нотацию «имя :тип» для обозначения экземпляров классов, если изображаемые объекты действительно являются экземплярами классов, существующих в проекте

Потоки управления и передачи данных изображаются в виде стрелок. Они обозначают передачу управления и данных (объекта) от одного шага алгоритма (действия) к другому (действию).

Весь алгоритм, представленный на диаграмме деятельности, изображает некий сценарий, который называется деятельностью и который может быть изображен в виде действия, например, на других диаграммах. Поэтому для обозначения полной спецификации деятельности применяют следующую нотацию.



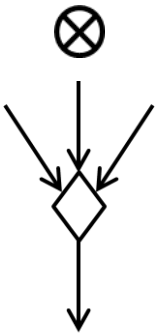
Здесь имя – название деятельности, параметры – возможные параметры (переменные), участвующие в деятельности, пред- и постусловия – некоторые логические выражения, которые должны или будут иметь истинное значение соответственно перед началом исполнения деятельности либо по ее окончании. В качестве входных и выходных данных используются объекты, причем их может быть несколько как на входе, так и на выходе.

Узлы управления на диаграммах деятельности позволяют задавать правила передачи потоков управления. Основные типы узлов управления:

Начальный узел используется для обозначения точки входа или, другими словами, начало алгоритма. Должен присутствовать на диаграмме в единственном числе

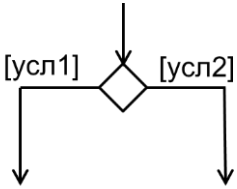
Конечный узел используется для обозначения точки выхода из диаграммы или, другими словами, – окончания алгоритма. На диаграмме может быть несколько конечных узлов

Узел соединения используется в тех случаях, когда затруднительно провести линию для потока управления. В этом случае стрелка передачи потока от источника изображается входящей в узел соединения с определенным именем, а в другом месте диаграммы снова изображается узел соединения с таким же именем, из которого линия передачи управления идет к действию-приемнику

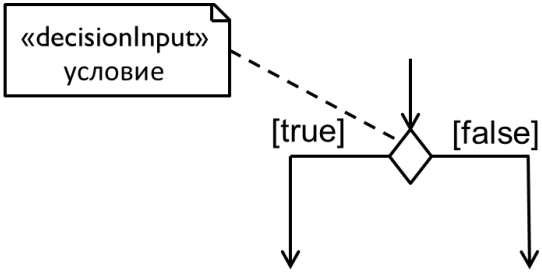


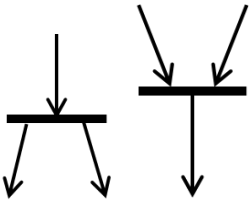
Конечный узел потока обозначает, что в этой точке завершается выполнение данного потока (в случае наличия параллельных потоков), но не всего алгоритма

Узел слияния (merge) используется для соединения потоков управления (например, после ветвления)



Узел решения (decision) используется, чтобы обозначить возможные пути переходов при различных условиях. Если условие ветвления достаточно сложное, применяют специальное обозначение для ограничения с ключевым словом «decisionInput», в котором и записывают это условие подробно:

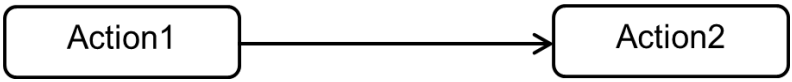




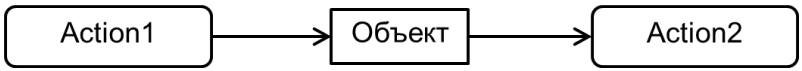
Узлы разветвления (fork) и объединения (join) применяются для распараллеливания и синхронного соединения потоков управления

Передачу фокуса управления от одного действия к другому изображают с помощью стрелки передачи управления. При этом стрелка может иметь различные подписи – имя, условия, ограни-

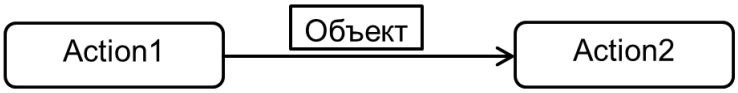
чения и т.п., которые мы не рассматриваем в рамках данного материала.

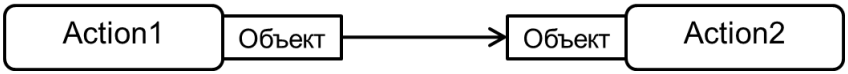


Передача данных обозначается двумя стрелками от одного действия к другому, между которыми вставлен соответствующий объект передачи данных:

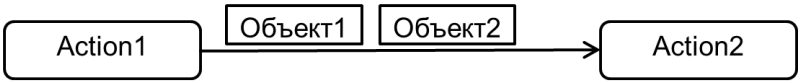


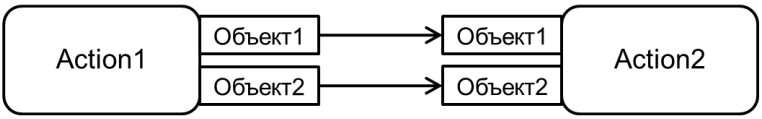
Объект передачи также может быть изображен над стрелкой или продублирован на ее концах:



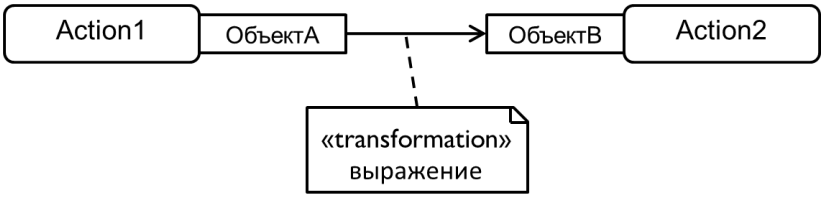


С помощью таких способов изображения можно показать сразу несколько объектов, которые передаются от действия к действию:



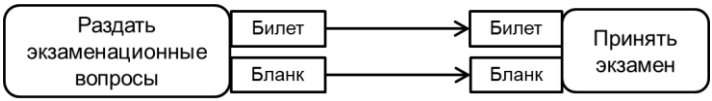


или даже преобразование данных с помощью некоторого выражения:

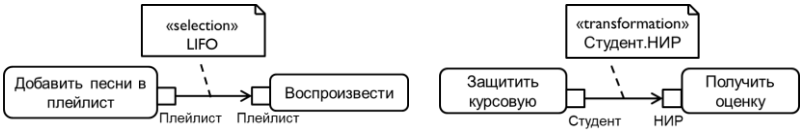


В этом случае на концах линии передачи данных будут изображены разные объекты, а сама линия передачи должна быть помечена пояснением с ключевым словом «transformation» и описанием того, каким образом первый объект будет преобразован во второй (или как вычислить второй объект на основе данных первого).

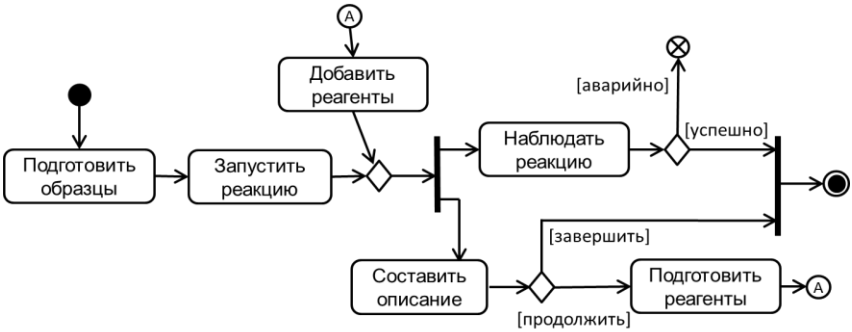
Приведем несколько примеров при переходе от одной деятельности к другой с участием объектов передачи данных. При проведении экзамена учащимся раздаются билет с вопросами и бланк ответов.



Пример ограничения «selection» (слева), обозначающего, как выбираются объекты (в данном примере объекты – песни плейлиста – будут воспроизводиться в порядке LIFO). Справа – пример преобразования «transformation»: объект НИР получен из атрибута НИР объекта Студент.



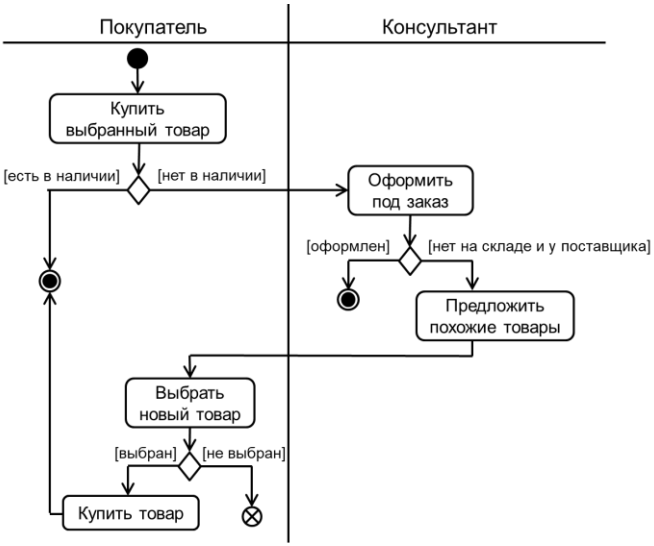
На следующей диаграмме приведен пример с параллельными потоками управления: при наблюдении реакции можно попутно составлять описание эксперимента и готовить следующие реагенты. Также использован узел соединения, по которому осуществляется переход от подготовки реагентов к их добавлению и новому протеканию реакции.



Плавательные дорожки. На диаграммах деятельности очень часто используются так называемые разделы или «плавательные дорожки» (partitions, regions, swimlanes), которые делят диаграмму на несколько частей и обозначают некоторые специфические условия выполнения действий, попавших в тот или иной раздел. Чаще всего технология «плавательных дорожек» используется, когда диаграммы деятельности изображают бизнес-процессы, шаги которых выполняются разными исполнителями. В этом случае каждому исполнителю отводится своя дорожка, и все его действия располагаются именно в этой дорожке. Пример применения «плавательных дорожек» представлен ниже.

Пример. Опишем бизнес-процесс выбора товара покупателем очно в магазине.

Если выбранный товар есть в наличии в магазине, то покупатель оформляет покупку. Если выбранного товара в наличии нет, то покупатель обращается к консультанту, чтобы оформить доставку товара под заказ. Если на складе в наличии товара нет и от поставщика не ожидается поставок, то консультант предлагает рассмотреть другие товары со схожими характеристиками. Если в подборке товар заинтересует покупателя – он оформляет покупку.



Контрольные вопросы

1. Для чего применяются диаграммы деятельности?

2. Назовите основные элементы диаграмм деятельности, их назначение и обозначение.

3. Перечислите типы узлов управления на диаграммах деятельности.

4. Приведите пример использования узлов решения (decision) и слияния (merge).

5. Приведите пример использования узлов разветвления (fork) и объединения (join).

6. Как обозначается передача данных от действия к действию? Приведите примеры возможных обозначений.

Задание

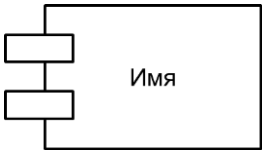
Отобразите значимые процессы своей проектируемой системы с помощью диаграмм деятельности:

а) используйте узлы решения, разветвления и другие;

б) для одного из описываемых процессов отобразите плавательные дорожки.

10. Диаграммы компонентов

Компонент – это логическая замещаемая часть системы, которая соответствует некоторому набору интерфейсов и обеспечивает их реализацию. Обозначается таким же образом, как и другие классификаторы, – прямоугольником, возможно, с несколькими секциями. В первой секции, которая не может быть опущена, указывается ключевое слово «component» и имя компонента. Также часто ставят знак компонента, использовавшийся в UML 1:



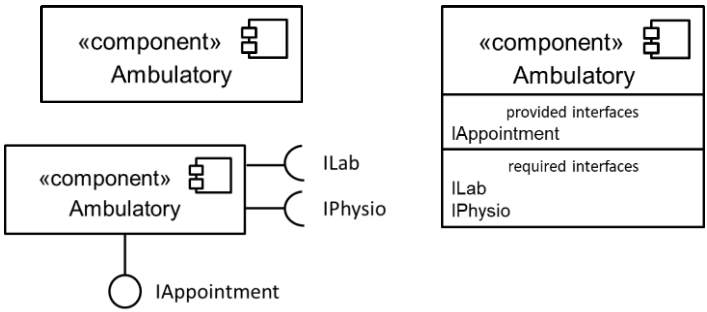
Так выглядит компонент в UML 2:

Многие разработчики опускают ключевое слово «component», оставляя лишь значок и используя его для указания стереотипа.

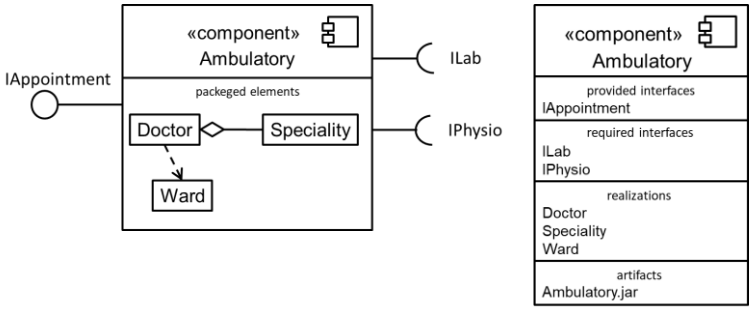
С помощью компонентов моделируют продукты компонентно-ориентированной разработки. Обычно компонент является логической реализацией некоторой группы функций системы и реализуется в виде одного или нескольких артефактов.

На диаграмме компонент может быть представлен в двух основных видах: как черный или как белый ящик в зависимости от назначения самой диаграммы – показать устройство компонента или продемонстрировать его связи с другими частями системы.

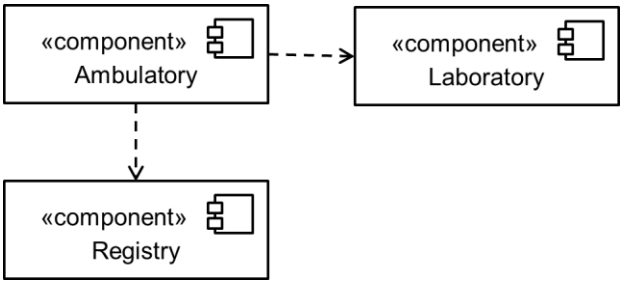
Компонент в виде черного ящика представляется с помощью имени или более расширенно – с указанием предоставляемых (реализуемых) и требуемых (использующихся) интерфейсов:



Изображая компонент как белый ящик, мы показываем его устройство – кроме предоставляемых и требуемых интерфейсов указываем классы, которые реализует данный компонент, и/или артефакты, которые реализуют сам компонент:

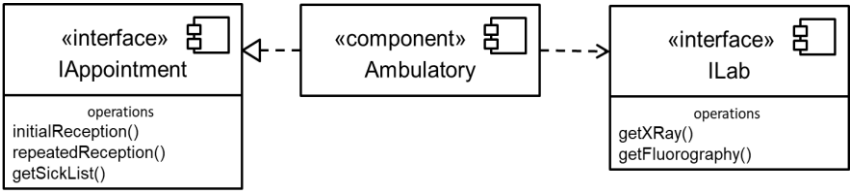


Диаграммы компонентов раскрывают связи между компонентами и интерфейсами. В упрощенном виде эти связи изображают в виде зависимостей:

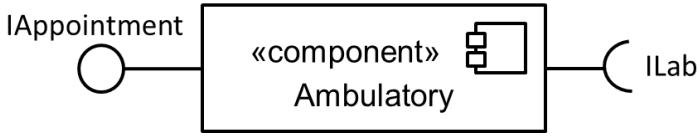


Для работы компонента Ambulatory потребуется наличие компонентов Registry и Laboratory.

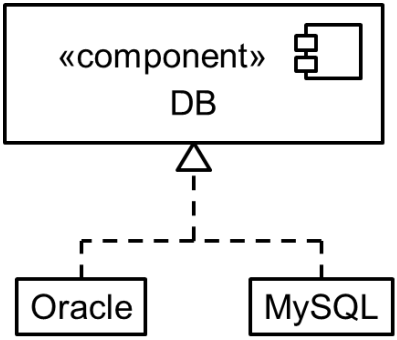
Расширенное представление может включать описание предоставляемого (IAppointment) и требуемого (ILab) интерфейсов для компонента Ambulatory.



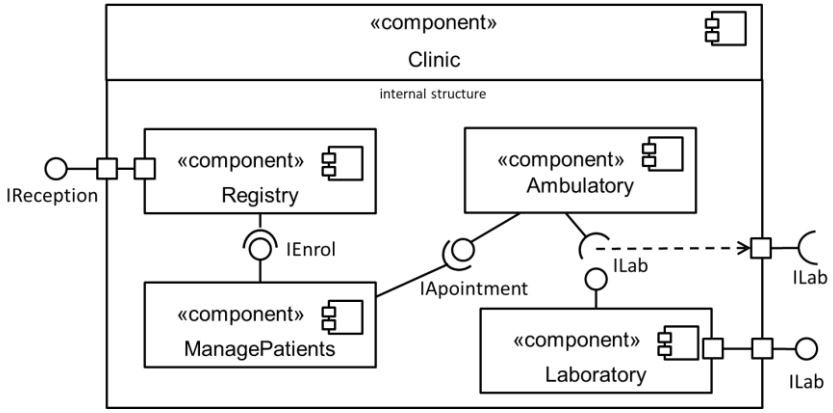
или



Так изображают классы, с помощью которых реализуется интерфейс:

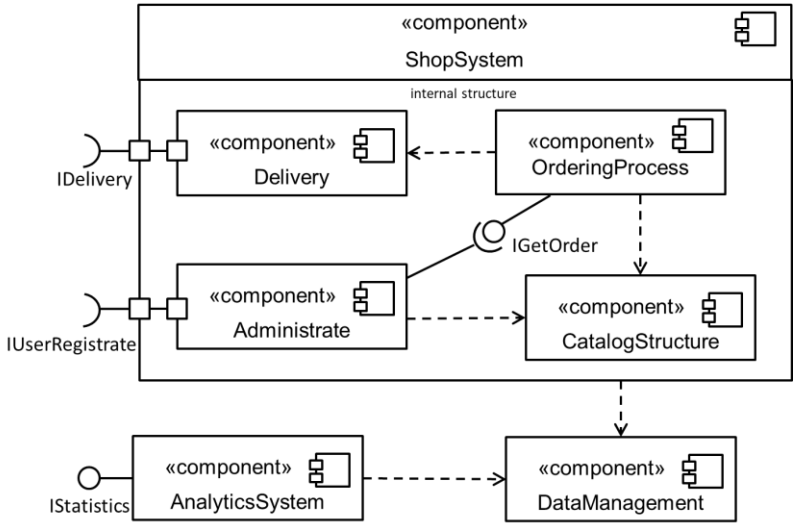


Внутренняя структура составного компонента может быть представлена следующим образом:



На данной диаграмме маленькими квадратиками на границах компонентов показаны так называемые порты, с помощью которых уточняются внутренние элементы, которые реализуют или требуют соответствующие интерфейсы.

Пример. Разбить систему онлайн-торговли на структурные компоненты, указать связи между составляющими.



Основной компонент ShopSystem требует интерфейсы доставки IDelivery и регистрации пользователя IUserRegistrate. Интерфейсы используются в конкретных элементах основного компонента – Delivery и Administrate. Компонент Administrate содержит логику работы CRUD пользователей системы, а также CRUD элементов структуры каталога CatalogSructure. Для оформления заказа реализован интерфейс IGetOrder компонентом OrderingProcess, которому для работы необходимы компоненты CatalogStructure и Delivery.

Также основной компонент ShopSystem задействует компонент с данными DataManagement, который используется компонентом аналитики и статистики AnalyticsSystem.

Контрольные вопросы

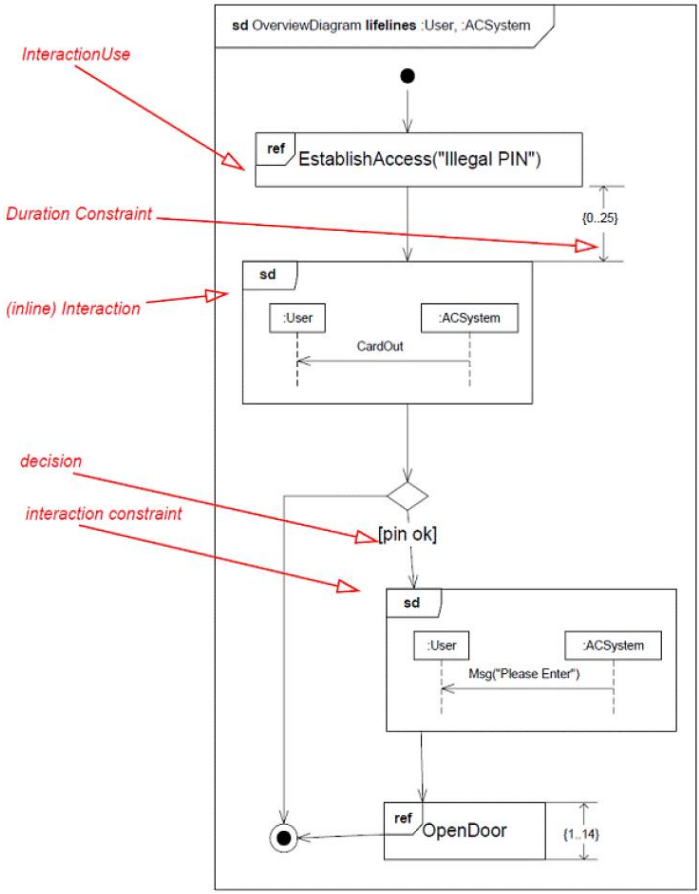
1. Что такое компонент? Как он обозначается в UML?
2. Для чего используются диаграммы компонентов?
3. Что такое предоставляемый и требуемый интерфейсы? Как они обозначаются?
4. Подумайте, чем компонент отличается от пакета и артефакта, а диаграмма компонентов от диаграмм пакетов и развертывания.

Задание

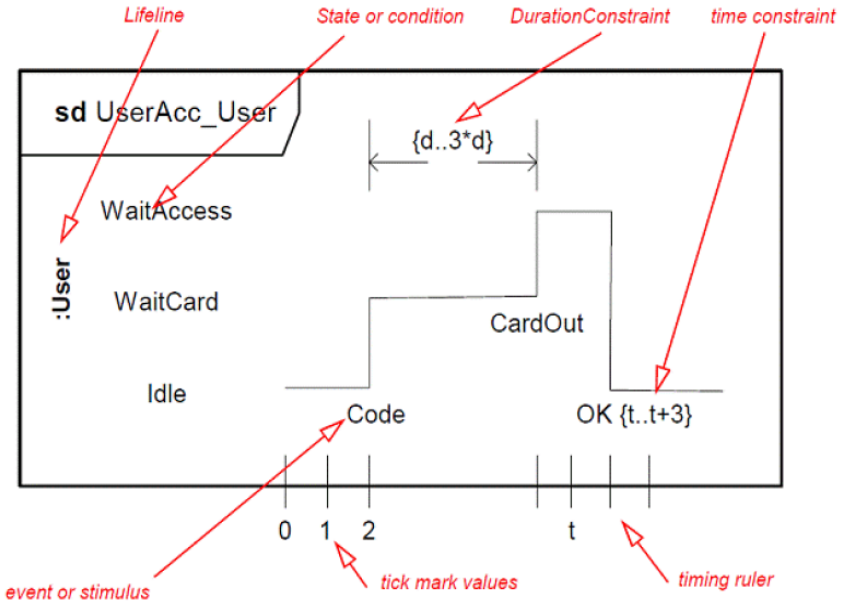
Представьте вашу систему в виде взаимосвязанных отдельных «готовых модулей» – компонентов. Отследите отсутствие циклических зависимостей между компонентами. Уделите внимание предоставляемым и требуемым интерфейсам как внутри приложения, так и по отношению к внешним системам.

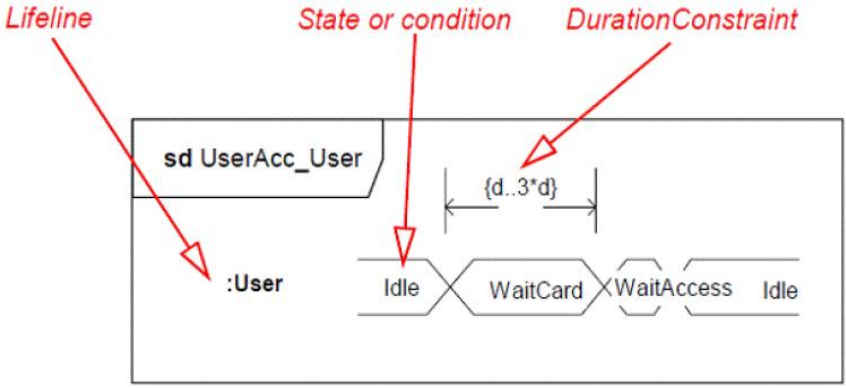
11. Другие типы диаграмм

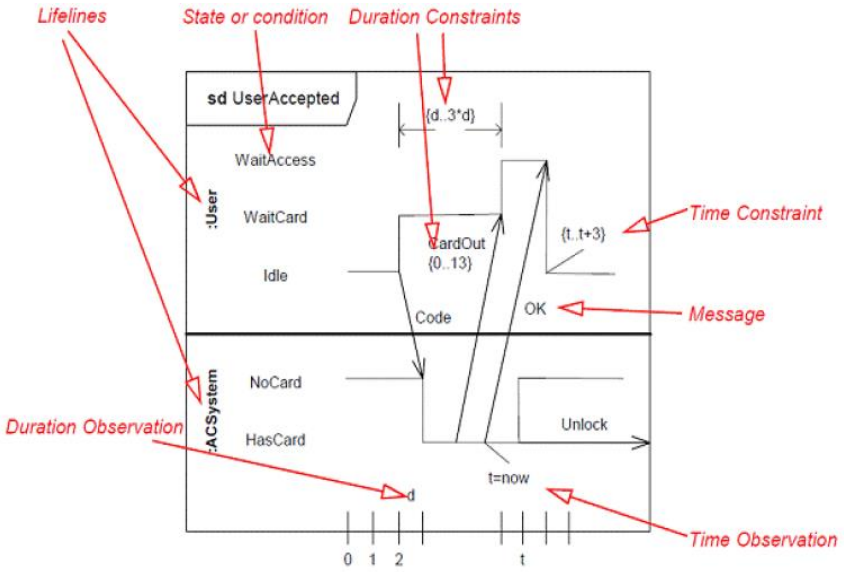
Диаграммы обзора взаимодействий – это, по сути, то же самое, что и диаграммы деятельности, но на них вместо действий узлами являются взаимодействия (interactions), подробно описанные в разделе про диаграммы последовательностей. Пример диаграммы обзора взаимодействий с пояснениями, заимствованный из описания стандарта UML 2.5, представлен ниже:



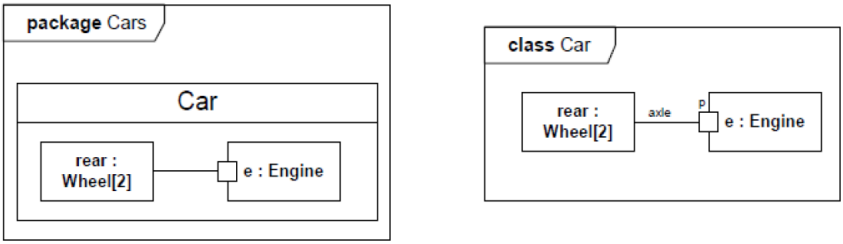
Временные диаграммы изображают процессы, происходящие во времени. На таких диаграммах представлены состояния объектов, действия и события, меняющие эти состояния. Примеры временных диаграмм с пояснениями, заимствованные из описания стандарта UML 2.5, приведены ниже:





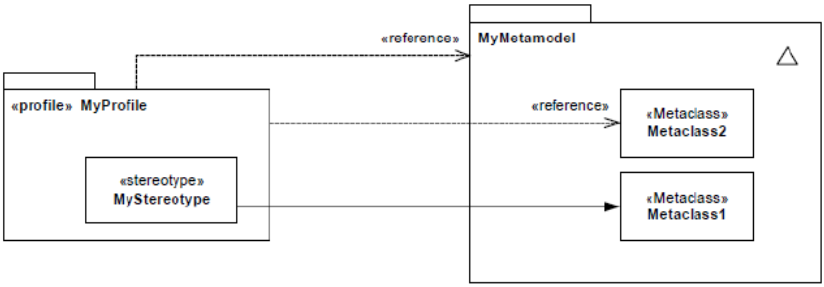


Диаграммы составных структур (или компоновки структуры) используются для подробного описания устройства классификаторов, таких как классы, кооперации и т.п. На рисунке (взят из стандарта UML 2.5) изображены два эквивалентных способа изображения внутреннего устройства объекта Car с помощью диаграмм составной структуры:



Диаграммы профилей предназначены для отображения структуры и связей между профилями UML. Профиль UML – это неко-

торая метамодель, описывающая и/или расширяющая язык. Сама базовая метамодель UML является профилем. Диаграммы анализа и их элементы можно считать тоже профилем UML – они дополняют язык новыми стереотипами и правилами конструирования модели. Ниже приведен пример простой диаграммы профилей (взят из описания стандарта UML 2.5):



Диаграммы профилей применяются обычно только в тех случаях, когда проект затрагивает или целиком посвящен модификации базовых конструкций языка UML.

Литература

1. Welcome to UML Web Site! URL: http://www.uml.org

2. Фаулер М. UML. Основы. 3-е изд. СПб. : Символ-Плюс, 2004. 192 с.

3. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. 2-е изд. М. : ДМК Пресс, 2006. 496 с.

4. Арлоу Д., Нейштадт А. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. 2-е изд. : пер. с англ. СПб. : Символ-Плюс, 2007. 615 с.

5. Иванов Д.Ю., Новиков Ф.А. Унифицированный язык моделирования UML : [учеб. пособие]. СПб. : Изд-во Политехн. ун-та, 2011. 229 с.

6. Леоненков А. Самоучитель UML. СПб. : БХВ-Петербург, 2004. 427 с.

7. Хританков А.С., Полежаев В.А., Андрианов А.И. Проектирование на UML. Сборник задач по проектированию программных систем. 2-е изд. Екатеринбург : Издательские решения, 2017. 240 с.

8. Рамбо Дж., Блаха М. UML 2.0. Объектно-ориентированное моделирование и разработка. СПб. : Питер, 2007. 544 с.

9. Мартин Р., Мартин М. Принципы, паттерны и методики гибкой разработки на языке C# : пер. с англ. СПб. : Символ-Плюс, 2011. 768 с.

10. Учебник по UML Coderlessons.com. URL: https://coderlessons.com/tutorials/kompiuternoe-programmirovanie/uchebnik-uml/uchebnik-uml

Учебное издание

Александр Николаевич МОИСЕЕВ
Марина Игоревна ЛИТОВЧЕНКО

Основы языка UML

Учебное пособие

Редактор Н.А. Афанасьева
Компьютерная верстка А.И. Лелоюр
Дизайн обложки Л.Д. Кривцовой

Подписано к печати 27.06.2023 г. Формат $60\times84^{1}/_{16}$.
Бумага для офисной техники. Гарнитура Times.
Печ. л. 6. Усл. печ. л. 5,6. Тираж 24 экз. Заказ № 5483.

Отпечатано на оборудовании
Издательства Томского государственного университета
634050, г. Томск, пр. Ленина, 36
Тел. 8+(382-2)–52-98-49
Сайт: http://publish.tsu.ru
E-mail: rio.tsu@mail.ru

ISBN 978-5-907572-06-5