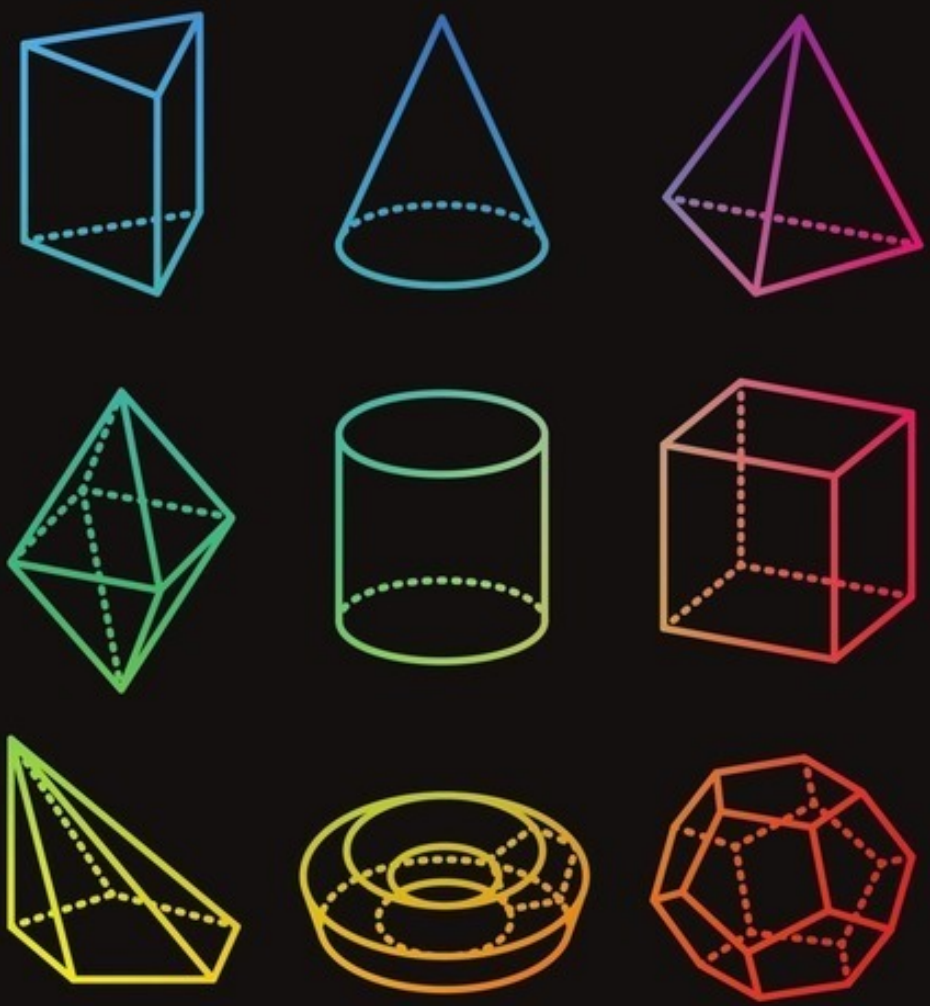
ПРОЕКТИРОВАНИЕ НА UML

Сборник задач



Хританков • Полежаев • Андрианов

Антон Хританков

Проектирование на UML. Сборник задач

«Издательские решения»

Хританков А. С.

Проектирование на UML. Сборник задач /
А. С. Хританков — «Издательские решения»,

ISBN 978-5-44-857954-7

В данном сборнике представлены задачи по проектированию ПО с использованием унифицированного языка моделирования UML 2, принципов и паттернов проектирования. Сборник содержит более 120 задач с несколькими заданиями в каждой по разным разделам UML и проектирования ПО. Для каждого раздела приводятся основные понятия, для задач даны ответы и пояснения по решению.http://www.objectoriented.ru

ISBN 978-5-44-857954-7

© Хританков А. С.
© Издательские решения

Содержание

Проектирование на UML
Сборник задач

Антон Сергеевич Хританков
Валентин Александрович Полежаев
Андрей Иванович Андрианов

© Антон Сергеевич Хританков, 2017
© Валентин Александрович Полежаев, 2017
© Андрей Иванович Андрианов, 2017

ISBN 978-5-4485-7954-7
Создано в интеллектуальной издательской системе Ridero

Проектирование на UML. Сборник задач по проектированию программных систем

Рекомендовано ученым советом ФИВТ МФТИ
к использованию в учебном процессе факультета
при подготовке студентов по направлениям
010400 «Прикладные математика и информатика» и
010600 «Прикладные математика и физика»
Рецензенты:
д.ф.-м.н., профессор, Соколинский Л. Б.,
ведущий разработчик, Колпаков Е. А.

Аннотация

В данном сборнике представлены задачи по проектированию программных систем с использованием унифицированного языка моделирования UML2, принципов и паттернов проектирования. Сборник содержит более 120 задач с несколькими заданиями в каждой по разным разделам UML и проектирования ПО. Для каждого раздела приводятся основные понятия, для задач даны ответы и пояснения по решению. Приведены рекомендации по составлению проверочных работ с использованием задач сборника по темам проектирования.

Для слушателей курсов по объектно-ориентированному анализу и проектированию программного обеспечения, студентов технических и физико-математических специальностей, преподавателей высших учебных заведений, специалистов по программной инженерии.

Дополнительную информацию и материалы можно найти на сайте книги http://www.objectoriented.ru

При цитировании, используйте следующую информацию о книге.

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.; ил.

ISBN 978-5-4485-7954-7

УДК 004.41+004.02+372.8

ББК 32.973.23—018

(С) Хританков А. С., 2017

(С) Полежаев В. А., 2017

(С) Андрианов А. И., 2017

ОБ АВТОРАХ

Хританков Антон Сергеевич, к.ф.-м.н.
доцент кафедры АТП, Московский физико-технический институт.

Защитил диссертацию в сфере высокопроизводительных вычислений (МФТИ / ИСА РАН). Опыт преподавания более восьми лет, научные интересы: архитектура программного обеспечения, автоматизированные и интеллектуальные методы разработки программ. Опыт работы в индустрии более 12 лет от разработчика ПО до архитектора и руководителя департамента разработки и исследований. Сертифицированный специалист по UML2 (OMG Certified UML Professional Advanced).

Email: anton.khritankov@objectoriented.ru

Полежаев Валентин Александрович
директор по разработке и анализу данных компании Интелиор.

Окончил ВМК МГУ, автор нескольких статей по теме машинного обучения и практике применения предметно-ориентированных методов проектирования. Участвовал в разработке более десяти информационных систем, из них более половины в качестве бизнес-аналитика и архитектора.

Email: valentin.polezhaev@objectoriented.ru

Андрианов Андрей Иванович
руководитель группы морфологии «Аби Продакшн» (ABBYY).

Магистр физ.-мат. наук (МФТИ), в разное время преподавал в МФТИ курсы «Алгоритмы и структуры данных», «Проектирование программных систем», «Машинное обучение». А также «Концепции языков программирования», «Промышленное программирование». Опыт разработки, проектирования архитектуры и управления проектами более 9 лет.

Email: andrey.andrianov@objectoriented.ru

ПРЕДИСЛОВИЕ КО ВТОРОМУ ИЗДАНИЮ

Проектирование – это процесс построения модели объекта, который предполагается разработать или создать. Модели в проектировании играют важную роль: модели используются для уточнения того, что нужно сделать, для прояснения способа реализации, для понимания реализуемости решения и его соответствия требованиям, для передачи знаний и распространения информации о проектируемом объекте, для планирования и ведения работ по реализации.

Во многих инженерных отраслях проектирование занимает важное место и часто регулируется государственными, промышленными стандартами или стандартами уровня предприятия. В сфере разработки программного обеспечение проектирование будущей программной системы происходит как в начале работ, так и по ходу реализации.

Важно понимать, что область знаний проектирования – это отдельная дисциплина, требующая особых навыков работы с моделями, применения методов и практик, которые обычно не используются при реализации создаваемого объекта.

На данный момент в индустрии разработки программного обеспечения сложилось несколько отраслей, каждая из которых использует несколько отличные от других методы проектирования. Среди этого разнообразия в книге уделено больше внимания проектированию прикладных программ, компонентов и приложений с помощью объектно-ориентированных методов и унифицированного языка моделирования UML2.

Овладение этими методами позволит в дальнейшем с легкостью освоить и другие сферы проектирования программных систем, и другие языки моделирования.

В данной книге собрано более сотни задач по проектированию. Авторы приложили все возможные усилия к тому, чтобы задачи помогли читателю понять смысл и освоить те или иные концепции, принципы и методы проектирования. Рассматриваемый перечень тем примерно соответствует программе курса по проектированию программных систем, читаемом авторами в Московском физико-техническом институте на протяжении уже более восьми лет и рекомендациям ACM/IEEE по составу учебных программ по проектированию программного обеспечения.

По сравнению с первым изданием книгу пополнили задачи, предлагавшиеся студентам на контрольных работах по проектированию программного обеспечения, вошел новый раздел по предметно-ориентированному проектированию, добавлены рекомендации по составлению проверочных и контрольных работ по отдельным темам проектирования, а также вошло множество задач разной сложности, которые авторы сочли интересными и важными для освоения дисциплины проектирования.

По сравнению с первым изданием в книге изменился состав авторов, тем не менее, важно отметить вклад Штукатурова А.Н, по согласованию с которым во второе издание вошли подготовленные им задачи 2.5, 3.7, 3.9, 4.4, 5.5, 5.6, 7.5, 8.9. Раздел §9 подготовлен Полежаевым В. А., задачи 6.2, 6.4, 2.3, 2.8, 3.10, 7.11, 7.12, 1.3, 1.4, 2.1, 7.1, 8.1, 3.2 предложены Андриановым А. И., остальные задачи, теоретическая справка и примеры решения задач составлены Хританковым А. С.

Апрель 2017

ПРЕДИСЛОВИЕ К ПЕРВОМУ ИЗДАНИЮ

Предлагаемая читателю книга является сборником задач по курсу проектирования программных систем, преподаваемому авторами в Московском физико-техническом институте.

Сборник включает задачи по проектированию и моделированию с помощью языка UML2. Каждая задача имеет условие, в котором описана заготовка модели, и несколько вопросов к ней. Часть вопросов направлена на уточнение и расширение заготовки модели. Другая часть проверяет понимание смысла построенной модели. Предполагается, что вопросы к задаче будут решаться по порядку.

Задачи сборника направлены на развитие навыков изложения проектных решений средствами UML и устроены таким образом, чтобы в наибольшей степени обеспечить единственность решения. В сложных случаях к задачам даны пояснения и указания по решению.

В сборник включены задачи по объектно-ориентированному моделированию предметной области, в том числе задачи на выделение классов, задачи по моделированию структур времени выполнения и размещения компонентов программных систем. Моделированию поведения систем в сборнике посвящены разделы описания взаимодействий, моделирования с помощью конечных автоматов и представления деятельности.

Разделы сборника снабжены пояснениями по основным понятиям. В конце сборника приведены ответы к задачам с пояснениями, приводятся примеры решения некоторых задач.

Сборник может быть использован при проведении семинаров по курсам объектно-ориентированного моделирования и программирования, проектирования программных систем, а также специализированных курсов по языку UML. Задачи сборника могут предлагаться в качестве примеров, демонстрирующих и поясняющих основные понятия и особенности языка, могут входить в задания для самостоятельной работы, проверочные и контрольные работы, использоваться для самостоятельного изучения методов проектирования и языка UML.

Задачи составлены авторами на основе собственного опыта преподавания, адаптированы из профессиональной практики, проведенных контрольных и проверочных работ, предлагаемых студентам проектов.

Июнь 2012

ГЛАВА 1. ОСНОВЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО МОДЕЛИРОВАНИЯ

Краткая история UML. Унифицированный язык моделирования UML появился в результате объединения нескольких подходов к моделированию в середине 1990-х годов. В отличие от предыдущих попыток, в создании языка участвовали авторы этих подходов, а вследствие стандартизации через организацию OMG, участвовали также заинтересованные компании и исследовательские коллективы из разных отраслей.

Поэтому разные части UML отражают потребности в моделировании в разных отраслях и объединение их в одном языке и на основе одной базовой системы понятий позволяет говорить как раз об унифицированном языке моделирования.

В данной книге унифицированный язык моделирования выбран как основной для выражения проектировочных решений. При решении задач стоит ориентироваться на версию UML 2.4.1, которая была стандартизирована международной организацией по стандартизации ISO как ISO/IEC 19505—1:2012 и ISO/IEC 19505—2:2012.

Уровни использования UML. Выделяют несколько уровней владения и использования UML и моделей в целом при проектировании программных систем. На уровне эскиза модели используются для пояснения решений, неформального общения, документирования и не обладают полнотой, строгостью и могут быть несогласованными. На уровне спецификации модель используется как чертеж или план реализации, в соответствии с которым разрабатывается программная система.

Модели и диаграммы на этом уровне должны следовать нотации языка и быть **согласованными** (**well-formed**). Согласованность модели означает соответствие правилам использования языка UML2, определенным в его спецификации (метамодели) [4]. Например, если на диаграмме показан элемент модели, то в модели также должны быть определены все элементы, используемые показанным.

На исполняемом уровне модель представляет собой достаточное описание системы для ее воплощения автоматическими средствами. В этом случае исходный код системы может не сохраняться вовсе и быть промежуточным этапом получения работающей программной системы из исходных моделей.

В данном сборнике задач следует ориентироваться на использование UML на уровне спецификации. В то же время часть задач предполагает владение языком на исполняемом уровне.

Решение задач. Прежде чем приступить к решению задач стоит ознакомиться с рекомендуемой литературой для ознакомления с методами проектирования и нотацией. В помощь читателю в начале каждого раздела приводится краткая справка по используемым в задачах раздела понятиям и демонстрируется нотация языка.

Все задачи построены по единому принципу. В условии дается заготовка модели. Это может быть диаграмма или текстовое описание. В текстовом описании названия элементов модели приведены *курсивом* для облегчения их восприятия. Далее приводится несколько заданий или вопросов к условию. В качестве решения задания нужно указать по шагам ход рассуждения от условия или предыдущего задания к достижению условий, указанных в задании. Для ответа на вопрос следует привести рассуждение в обоснование полученного ответа и сам ответ. Задания и вопросы к задачам следует выполнять по порядку. Решение следующего задания может зависеть от решения предыдущего. Ответом на задание будет фрагмент диаграммы или нескольких диаграмм с представлением изменений, требуемых в данном задании.

При решении следует руководствоваться условием задачи, знаниями методов решения, нотацией и значением понятий языка моделирования. Часть задач составлена на основе реальных проектов разработки программного обеспечения, другую часть составляю учебные задачи. Такие задачи могут вызывать ассоциации с похожими ситуациями или реальными объектами. В этом случае следует придерживаться условия задачи. Если не указано иное, решение задач не предполагает каких-либо специальных знаний в специализированных областях. При необходимости дается сноска, где можно получить дополнительную информацию.

Задачи и задания повышенной сложности отмечены звездочкой (*), для некоторых задач приведено решение, в этом случае указана страница, на которой оно расположено (см. решение в §11). Перед тем, как приступить к решению задач рекомендуется ознакомиться с примерами решения и понять порядок ведения рассуждения и степень его детальности.

При составлении задач уделялось особое внимание тому, чтобы решение было единственным. При необходимости в заданиях к задачам даются указания по предполагаемому способу решения. Впрочем, вполне возможно, что читатель сможет предложить более удачные или лаконичные решения по некоторым задачам. Возможность существования лучшего решения следует учитывать и не требовать однозначного совпадения решения с ответами, приводимыми авторами сборника.

§1. КЛАССЫ И ОБЪЕКТЫ

ОСНОВНЫЕ ПОНЯТИЯ

Пространство имен (namespace) – это именованный элемент модели, который может содержать другие именованные элементы. Принадлежность пространству имен показывается отношением **включения в пространство имен (membership)**. **Полностью квалифицированное имя (fully-qualified name)** элемента в модели состоит из последовательности имен всех вложенных пространств имен, в которые включен элемент.

Классификатор (classifier) – это пространство имен в модели, указывает на общие некоторому множеству объектов черты. Черты классификатора могут быть поведенческими, структурными или соединительными.

Класс (class) – это классификатор, который описывает некоторую концепцию моделируемой области. Черты класса могут быть различных видов, наиболее часто для описания функциональности класса используются операции (operation), а для описания хранимых данных или связей с другими классами – **свойства (property)**. Если типом свойства является примитивный тип или тип данных, свойства показывают как атрибуты, класса иначе как часть ассоциации.

Операция (operation) – черта поведения интерфейса, класса или типа данных. Операция задается именем, набором параметров, типом возвращаемого значения и его кратностью. Каждый параметр операции может иметь имя, тип, кратность. В программировании операции будет соответствовать сигнатура метода.

Обратите внимание, что определение операции в классе не влечет определение ее реализации в этом классе. Понятие метода в UML2 обозначает реализацию операции алгоритмом, который не описывается средствами UML или не уточняется в модели. В последнем случае, такую реализацию операции называют нечетким поведением (opaqueBehavior).

Интерфейсом (interface) называют особый вид классификатора, который определяет способ взаимодействия с экземпляром класса, реализующего интерфейс. Интерфейс обычно включает операции, но может включать и свойства. В последнем случае наличие указанных свойств является обязательным для реализующего интерфейс класса.

Экземпляр класса (instance) – это элемент модели с описанием, возможно неполным, объекта, которому в системе приписаны черты данного класса. Для того чтобы указать значения свойствам класса в экземпляре используют слоты.

Связью (link) называется экземпляр ассоциации, соединяющий экземпляры классов. В языке программирования однонаправленной связи соответствует типизированный указатель или ссылка.

Ассоциация (association) – это типизированное отношение между классами, которое указывает на логическую связь между ними. Ассоциация имеет два или более полюсов, по одному у каждого связанного класса. Название полюса обычно указывает на роль, которую класс играет в ассоциации.

Обобщение (generalization) является направленным отношением от более специализированного классификатора к более общему. Специализированный, или дочерний, классификатор наследует черты более общего, или родительского, классификатора. Отношение обобщения уточняется отдельно для каждого вида классификатора, в том числе для классов и интерфейсов.

Украшениями (adornments) называются свойства полюса ассоциации, уточняющие роль участвующего в ассоциации класса. С помощью украшений указываются направление навигации, вид композиции и другие свойства полюса.

Типом данных (data type) называется классификатор, экземпляры которого не обладают индивидуальностью и, при совпадении значений свойств, взаимозаменяемы. **Простыми (primitive)**, или примитивным типами данных, являются предопределенные типы: целое *Integer*, строка *String*, логический тип *Boolean*, числа с плавающей запятой *Real* и неограниченные натуральные числа *UnlimitedNatural*, которые используются для моделирования неопределенного количества элементов, например, экземпляров класса, участвующих в ассоциации.

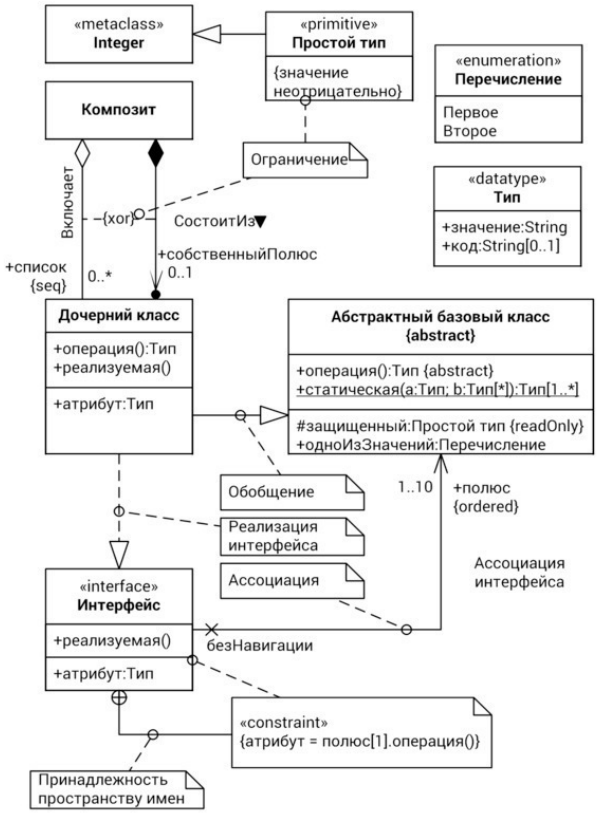


Рис. 1. Основная нотация диаграмм классов

Рис. 2. Нотация диаграмм экземпляров

Ограничением (constraint) называется логическое выражение об ограничиваемых элементах модели, вычисляемое в контексте какого-либо элемента. Если выражение ложно, то модель считается противоречивой (ill-formed).

Примеры нотации указанных выше элементов модели приведены на рис. 1 и рис. 2.

ЗАДАЧИ

1.1. Абстрактный класс *Account* имеет два дочерних класса: счет физического лица *PersonalAccount* и юридического *CompanyAccount*. При решении задачи используйте диаграммы классов.

а. Добавьте класс *Person* с общедоступным атрибутом *FullName* строкового типа и свяжите его с классом *PersonalAccount* ассоциацией *Owns* с полюсом *owner* у *Person* и навигируемым полюсом *account* у *PersonalAccount*.

б. Аналогично для счета юридического лица добавьте владельца *Company*, свяжите анонимной ассоциацией с *CompanyAccount* и укажите подходящие названия полюсов.

в. Добавьте класс адреса *Address* с атрибутами строкового типа *street*, *city* и целочисленным положительным *building*. Укажите с помощью новых анонимных ассоциаций, что *Person* может иметь адрес регистрации *registeredAt*, фактический адрес *actual*, в то время как компания связана с одним юридическим адресом *legalAddress* и может иметь почтовый адрес *postAddress*.

1.2. Интерфейс *Stack* определяет операции помещения в стек *push* с параметром *obj* типа *Element*, операцию получения элемента из стека *pop* с возвращаемым значением типа *Element*. При решении задачи используйте диаграммы классов.

а. Добавьте в интерфейс *Stack* операции очистки стека *reset*, которая не имеет параметров, статическую операцию создания нового стека *createNew* с возвращаемым значением типа *Stack*.

б. Покажите, что интерфейс *Stack* зависит от типа данных *Element*.

в. Добавьте класс *ListStack*, который реализует интерфейс *Stack*. Покажите реализуемые классом операции интерфейса.

г. Добавьте в класс *ListStack* частное структурное свойство *arr* типа *Element* с кратностью больше нуля, значения которого упорядочены и могут повторяться.

д. Добавьте частный целочисленный атрибут *increment* только для чтения и защищенную операцию изменения размера *resize* с целочисленным параметром *newSize*.

е. Покажите на диаграмме экземпляров экземпляр *stack* класса *ListStack*, свойство *arr* которого содержит элемент *first* типа *Element* первым и *second* того же типа вторым. Укажите, что атрибут *increment* экземпляра *stack* равен *10*.

1.3. В пространстве имен *Time* расположены перечисления *Month*, *DayOfWeek*, а также классы *Date* и *Period*. При решении задачи используйте диаграммы классов.

а. Укажите, что перечисление *Month* может принимать значения: *Jan*, *Feb*, *Mar*, *Apr*, *May*, *Jun*, *Jul*, *Aug*, *Sep*, *Oct*, *Nov*, *Dec*.

б. Укажите, что перечисление *DayOfWeek* может принимать значения: *Mon*, *Tue*, *Wed*, *Thu*, *Fri*, *Sat*, *Sun*.

в. Добавьте классу *Date* частные атрибуты *year*, *month*, *dayOfMonth* типа *Integer*, а также общедоступные операции:

– получения года *getYear* типа *Integer*; – получения месяца *getMonth* типа *Month*; – получения дня *getDayOfMonth* типа *Integer*; – получения дня недели *getDayOfWeek* типа *DayOfWeek*.

г. Добавьте классу *Date* общедоступную статическую операцию *now ()* типа *Date*.

д. Добавьте классу *Period* общедоступную статическую операцию *between*. У операции два аргумента: *from* и *to*. Оба аргумента имеют тип *Date*. Операция возвращает значение типа *Period*

е. Добавьте классу *Date* операцию *plus* с аргументом *delta* типа *Period*. Результат операции – значение типа *Date*.

1.4. Класс *MyWindow* уточняет абстрактный базовый класс *Window*. *MyWindow* состоит (композиция) из кнопки класса *Button* и надписи класса *Label*. Отобразите на диаграмме классов.

а. Класс *Label* имеет частный атрибут *text* типа *String* и общедоступную операцию *setText* с параметром *text* типа *String*.

б. Композиция между *MyWindow* и *Button* называется *HoldsButton*. Полюс со стороны кнопки имеет имя *okButton*, защищенную видимость, кратность *1*. Композиция между *MyWindow* и *Label* называется *HoldsLabel*. Украшения полюса со стороны *Label*: название *textLabel*, частная видимость, кратность *1*.

в. Для реакции на события кнопки реализован паттерн Слушатель (Listener) следующим образом. Класс *Button* предоставляет операцию *setClickListener* с единственным параметром *l* типа *IClickListener*. Интерфейс *IClickListener* содержит единственную операцию *onClick* без параметров.

г. Класс *MyWindow* реализует интерфейс *IClickListener* для реакции на нажатие кнопки. Отобразите на диаграмме, что между классом *Button* и *MyWindow* есть ассоциация с именем *NotifyListener* с направлением от кнопки к окну. Укажите, что полюс со стороны окна называется *listener*, имеет тип *IClickListener*, множественную кратность и частную видимость.

д. И *Label* и *Button* имеют строковый атрибут *text*, который можно менять с помощью метода *setText*. Вынесите общий атрибут и метод в абстрактный базовый класс *TextWidget*.

е. Отобразите на диаграмме объектов, как в процессе выполнения объекты связаны между собой: объект *window* класса *MyWindow* связан с кнопкой *button* класса *Button* и с надписью *label* класса *Label*.

1.5. (см. решение в §11) Интерфейс доступа к коллекции элементов *Collection* обобщает интерфейс работы со списками *List*. Абстрактный класс *BaseCollection* реализует интерфейс *Collection*, абстрактный класс *BaseList* является потомком *BaseCollection* и реализует интерфейс *List*, оставляя операции по хранению данных дочерним классам.

а. Используя наследование, добавьте в модель класс *ArrayList*, реализующий операции со списками с помощью массива.

б. Пусть интерфейс *List* содержит операцию *get* получения элемента списка по заданной позиции *k*. Укажите, в каких классах должна быть объявлена данная операция, чтобы модель была согласованной. Ответ поясните.

в. Пусть интерфейс *Collection* содержит операцию *add* добавления элемента *obj*. Укажите, в пространстве имен каких классов может присутствовать поведение, реализующее операцию *add*. Ответ поясните.

1.6. Класс *Collections* содержит общедоступную статическую операцию *addAll* с возвращаемым значением типа *boolean*. Первый параметр операции называется *coll* и имеет тип *Collection*, второй параметр называется *elements* и имеет тип *Object* и кратность больше нуля.

а. Добавьте в класс *Collections* статический атрибут *empty* типа *Collection,* предназначенный только для чтения.

б. Реализуйте в классе *Collections* операцию *addAll* с помощью нечеткого поведения (метода), используя операцию добавления элемента *insert (e: Object)* класса *Collection*. *Указание*. Алгоритм реализации можно показать как псевдокод в комментарии в формате *{method = {<language>} <method body>}*.

1.7. Узел дерева *Node* может иметь несколько дочерних *child* узлов того же класса *Node*.

а. Приведите пример бинарного дерева, состоящего из семи узлов *Node*.

б. Постройте модель дерева, в котором каждый узел имеет от двух до четырех дочерних узлов.

в. Разработайте модель дерева, узлы которого могут быть двух видов: узел *Red* и узел *Black*. *Указание.* Вид узла может изменяться, при этом следует считать, что поведение узла не изменяется при смене типа.

1.8. У абстрактного класса заказа *Reservation* имеется два потомка: одиночный *Single* и подписка *Subscription*. *Single* связан с одним билетом *Ticket* ассоциацией бронирован *reserved*, *Ticket* может быть связан той же ассоциацией не более чем с одним *Single*.

а. Свяжите подписку с билетами в количестве от трех до шести включительно. Билет не обязательно связан с подпиской.

б. Как с помощью ограничений указать, что билет не может быть одновременно связан и с подпиской, и с одиночным заказом?

в. Пусть одиночная подписка наследует свойства одиночного заказа и подписки. С каким максимальным количеством билетов она может быть связана? Ответ поясните.

1.9. Умный дачный домик *SmartHouse* состоит из четырех стен *Wall* и крыши *Roof*. Домик реагирует на штормовые предупреждения *stormWarning* и укрепляет крышу *harden*, закрывает окна *closeWindows* в стенах. Используемые стройматериалы *Material* характеризуются ценой *price* и удельным весом *unitWeight*.

а. Добавьте стройматериалы для постройки домика: красный и белый кирпич *Brick*, доски *Plank* из сосны и дуба.

б. Укажите, что кирпич является материалом *material* стен. Используя ассоциации, покажите, что каркас крыши *Frame* сделан из не более чем сорока досок и может быть одного из видов *FrameKind*: мансарда, плоский или треугольный.

в. Каркас можно покрыть стройматериалом черепица *Tiling*, отразите это в модели.

г. Допустим, изобретен универсальный стройматериал, заменяющий доски, кирпичи и черепицу. Постройте из него дачный домик. Сколько экземпляров материала понадобится? Ответ поясните.

§2. СЦЕНАРИИ И ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ

ОСНОВНЫЕ ПОНЯТИЯ

Актором (actor) называется классификатор, который моделирует пользователя или систему, внешнего по отношению к моделируемой системе или компоненту. Акторов, которые используют систему для достижения собственных целей, называют основными. Акторов, которых система использует для достижения целей других акторов, называют второстепенными.

Вариантом использования (use case) называют классификатор, который описывает совокупность сценариев взаимодействия акторов с системой или компонентом для достижения какой-либо цели, значимой для акторов. Варианты использования могут различаться по уровню цели, достижение которой они обеспечивают: высокоуровневые цели, пользовательские цели и отдельные функции системы.

Субъектом (subject) варианта использования называют систему или компонент, взаимодействие акторов с которым он описывает.

Ассоциация (association) актора с вариантом использования указывает на взаимодействие актора с субъектом в одном из сценариев данного варианта использования.

Отношение **расширения (extension)** между вариантами использования указывает, что при выполнении заданного в **точке расширения (extension point)** условия сценарий расширяемого варианта использования будет приостановлен, и взаимодействие будет продолжено в рамках расширяющего варианта использования.

Отношение **включения (inclusion)** указывает, что в процессе выполнения сценарии базового варианта использования вызывают выполнение сценариев включаемого варианта использования.

Как и для других классификаторов, для акторов и вариантов использования определено отношение **обобщения (generalization)**.

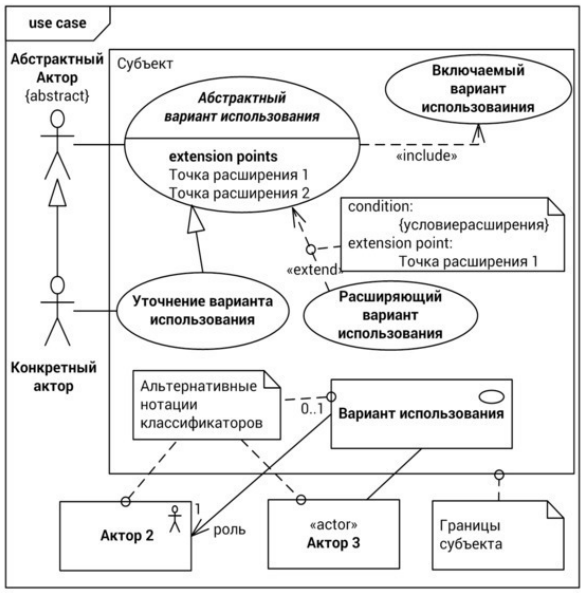


Рис. 3. Основная нотация диаграмм вариантов использования

ЗАДАЧИ

2.1. Актор *User* взаимодействует с системой *OnlineTranslator* в рамках абстрактного варианта использования *Translate*. Варианты использования *TranslateText* и *TranslateWebPage* уточняют *Translate*. Отобразите на диаграмме вариантов использования.

а. Вариант использования *TranslateWebPage* включает *«include»* вариант использования *SetURL*.

б. Вариант использования *SetLanguages* расширяет *«extend»* вариант использования *Translate* в точке расширения *specifyLaguages*. Условие расширения «язык не определен автоматически».

в. Добавьте в модель актора *ExperiencedUser*, уточняющего *User*. *ExperiencedUser* может взаимодействовать с системой в рамках варианта использования *ProposeTranslation*, который уточняет вариант использования *TranslateText*.

2.2. (см. решение в §11) Автор *Author* направляет статью *SendPaper* редактору журнала *Editor*. Редактор передает статью на рецензирование *Review* нескольким рецензентам *Reviewer*. Затем редактор возвращает отзывы рецензентов автору в том же варианте использования *SendPaper*.

а. Добавьте возможность автору вместе с корректором *ProofReader* подготовить статью к публикации *PrepareForPublishing*.

б. Доработайте модель, укажите, что подготовка статьи к публикации выполняется, только если она была одобрена редактором в варианте использования *SendPaper*.

2.3. Распознавателю текста *OCR* от модуля морфологии нужны возможность определить, принадлежит ли слово языку, и функция приведения слова к заданной форме, в частности, восстановления начальной формы. Также нужна функция получения грамматического значения конкретного слова.

а. Постройте модель модуля, выделите акторов, варианты использования и укажите отношения между ними.

б. Добавьте функцию вывода слов, похожих на введенное, если его нет в словаре языка. Каким образом данная возможность системы связана с другими функциями?

в. Укажите в модели, что все перечисленные задачи подразумевают выполнение поиска слова (или его основы) в словаре.

г. Некоторые языки могут не поддерживаться системой. Перед выполнением любой функции модуля морфологии нужно проверить, поддержан ли язык. Отобразите это в модели.

2.4. Ответственное лицо *ResponsiblePerson* может прикрепить документ *AttachToIssue* к обсуждаемому вопросу, выступая в роли автора *author*, и к постановлению *AttachToResolution*, выступая в роли председателя *chairman*.

а. Покажите в модели, что прикрепление документа выполняется согласно общему сценарию прикрепления, реализуемому в частном случае прикрепления к вопросу или прикрепления к постановлению.

б. Добавьте в модель оператора *Operator*, который является ответственным лицом с возможностью удаления документов *DeleteDocument*.

в. Доработайте модель, укажите, что при прикреплении документа рассылается оповещение *SendAnnouncement*. Несколько операторов могут выступать в роли контролеров *controller*.

г. Каким образом можно указать, что прикрепление документа возможно только к вопросу или к постановлению? Ответ поясните.

д. (*) Покажите в модели, что ответственное лицо участвует в сценарии прикрепления в роли пользователя *user*, объединяющей роли автора и председателя. *Указание*. Используйте производные свойства. См. §4.

2.5. Пользователь *User* настраивает подключаемые модули аудиоплеера *AudioPlayer* в рамках варианта использования *ConfigurePlugins*.

а. Добавьте к варианту использования *ConfigurePlugins* возможность выбора определенного модуля для настройки *SelectPlugin* и возможность настройки конкретного модуля *ChangeSettings*.

б. Добавьте в модель возможность обновить подключаемые модули *UpdatePlugins* с внешнего сервера *PluginsServer*.

в. Помимо обычного пользователя в системах обычно есть привилегированный пользователь *SuperUser*, который имеет права на изменение конфигурации системы. В системе аудиоплеера такой пользователь может обновить модули *UpdatePluginsList*. Обновление включает в себя удаление *DeletePlugin*, установку *InstallPlugins* и просмотр списка доступных на сервере *CheckPluginsList*.

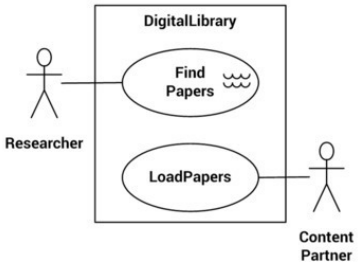


Рис. 4

2.6. Рассмотрим электронную библиотеку научных работ, представленную на рис. 4.

а. Поясните, каким образом используется электронная библиотека. Перечислите акторов и варианты использования.

б. Укажите, что аналитик *Analyst* принимает участие в индексировании статей, выполняемом в процессе их загрузки бизнес-партнером *ContentPartner*.

в. Предоставьте возможность исследователю *Researcher* использовать расширенный поиск *AdvancedSearch*, который позволяет указать другие параметры поиска в *FindPapers*.

г. Укажите, что все варианты использования преследуют цели уровня пользователя (user goal) системы. Указание. Уровни целей[1] не входят в стандарт UML2.

2.7. Клиент *Client* выполняет операции над своими счетами в банке *Bank*, используя банкомат *ATM* в рамках абстрактного варианта использования *PerformOperation*, который включает информирование об услугах в варианте использования *InformAboutServices*. Для выполнения операций *ATM* обращается к платежной системе *PaymentSystem*.

а. Перечислите основных и вспомогательных акторов системы *ATM*. Какие из них взаимодействуют с системой в варианте использования *PerformOperation*?

б. Отразите в модели вариантов использования, что клиенты могут только выполнять операции по получению наличных, в то время как клиенты *BankCustomers* банка, владеющего банкоматом, могут также оплачивать услуги из списка, предоставляемого банком *Bank*. При этом сценарии оплаты услуг и получения наличных отличаются между собой, но следуют общему сценарию выполнения операций.

[1] Коберн А. Современные методы описания функциональных требований к системам. – М.:Лори. – 2011. – 288 с.

в. Добавьте возможность получения наличных как в валюте счета, так и в другой валюте. При этом в обоих случаях банкомат запрашивает у клиента *Client* подтверждение на списание средств в валюте счета по курсу банка *Bank*.

2.8. Во время подготовки данных для морфологического модуля лингвист *Linguist* взаимодействует с системой подготовки данных *MorphoDPS* с целью изменения данных *ModifyData*. Кроме того, для проверки целостности модифицируемых данных лингвисты могут компилировать данные *Compile*. Компиляция также включает в себя экспорт данных *ExportData* в формат, понимаемый компилятором. Каждую ночь сервер сборки приложения *BuildServer* компилирует данные *Compile*.

а. Добавьте в систему программиста *Programmer*, которому доступны те же возможности, что и лингвисту. Кроме того, он может экспортировать данные *ExportData* для отладки подсистемы компиляции данных.

б. Укажите, что для повторного использования словаря, который хранится на сервере данных морфологии, модуль семантики *Semantics* может взаимодействовать с системой подготовки данных морфологии в варианте использования *ExportWordList*.

в. Добавьте функции изменения данных: добавление, изменение и удаление слова.

г. Добавьте в модель возможность при изменении данных в некоторых случаях проверять целостность данных перед сохранением в систему.

д. Будет ли проверяться целостность данных при удалении слова? Ответ поясните.

2.9. Инкассатор *Cashier* и заправщик *Loader* занимаются обслуживанием автомата с газировкой. В обязанности инкассатора входит сбор денег *CollectCash*, а заправщик загружает в автомат баллоны с водой *ChangeWater* и газом *ChangeGas*.

а. Выделите в модели общий сценарий обслуживания, который включает авторизацию в системе обслуживания автомата и завершение сессии обслуживания.

б. Укажите, что автомат также может быть заправлен сиропом.

в. В каком случае инкассатор может загрузить в автомат баллон с водой? Ответ поясните.

г. Отразите в модели, что инкассатор может наблюдать за автоматом через Интернет с помощью встроенной видеокамеры с включением по сигналу датчика присутствия здания. Решение поясните.

2.10. Первоначальная модель загрузки *CreateDocument* и проверки документов *ReviewDocument* преподавателем *Professor* учебного заведения приведена на рис. 5.

а. Добавьте преподавателю возможность создавать курсы, как по шаблону, так и повторяя курс прошлого года.

б. Покажите на диаграмме, что у преподавателя есть три возможности проверки документа: с помощью мастера *GuidedReview*, совместно со студентом *JointReview*, и простое ревью *BasicReview*. При этом студенты сами могут загружать документы *Upload* и регистрироваться *Enroll* на курс.

в. Добавьте в модель ассистента преподавателя *TA* так, чтобы он обладал всеми обозначенными выше возможностями преподавателя, но не мог создавать курсы. При этом студент может быть ассистентом, но не преподавателем.

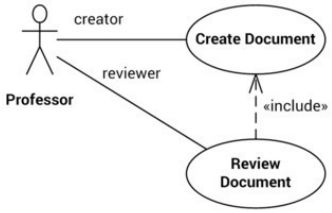

creator
Create Document
reviewer
Professor
«include»
Review Document

Рис. 5

§3. КООПЕРАЦИИ И ВЗАИМОДЕЙСТВИЯ КЛАССОВ

ОСНОВНЫЕ ПОНЯТИЯ

Структурированным классификатором (structured classifier) называется классификатор, который может включать соединители, связывающие содержащиеся в классификаторе свойства. Структурированные классификаторы определяют контекст для соединителей и позволяют описать связи между экземплярами, возникающие во время выполнения системы.

Соединители (connector) – это черты структурированного классификатора, имеющие тип и связывающие два или более свойств классификатора. В то время как связи (links) являются экземплярами ассоциаций, соединители ограничивают возможные связи между экземплярами в зависимости от контекстного классификатора, которому принадлежат эти экземпляры.

Частью (part) классификатора называется свойство, с которым классификатор связан отношением композиции.

Кооперация (collaboration) является структурированным классификатором, обладающим поведением, и определяет роли составляющих ее частей и взаимодействия между ними в контексте кооперации. Кооперации используются для определения взаимодействий, обеспечивающих достижение какой-либо цели или реализации функции системы.

Вхождение кооперации (collaboration use) связывает элементы модели с ролями, определенными в кооперации.

Ролями (role) в кооперации называют части кооперации, параметры поведения или локальные переменные в поведении кооперации.

Состоянием (state) экземпляра классификатора называют условие или ситуацию, во время которой его свойства удовлетворяют некоторому условию, он выполняет определенное собственное поведение или ожидает какого-либо события.

Поведение (behavior) классификатора описывает изменение состояния классификатора с течением времени в ответ на внешние события и внутренние вычисления. Поведение может быть исполняемым (executable) и производным (emergent). Исполняемое поведение является описанием процесса исполнения некоторого алгоритма экземпляром классификатора путем выполнения действий. Производное поведение возникает в результате взаимодействия нескольких экземпляров.

Суть **овеществления** (**reification)** заключается в представлении происходящего поведения в виде экземпляра класса поведения. Таким образом, выполнение поведения экземплярами классов отождествляется с созданием и уничтожением экземпляра класса этого поведения.

Событием (event) называется описание группы изменений, которые могут привести к модификации значений свойств экземпляров в модели или выполнению поведения.

Сигнал (signal) является специальным видом классификатора, который описывает асинхронные запросы, направляемые экземплярам классификаторов. Черта приема определенного типа сигнала (reception) указывает, что экземпляры принимающего активного класса обрабатывают направленные им сигналы данного типа.

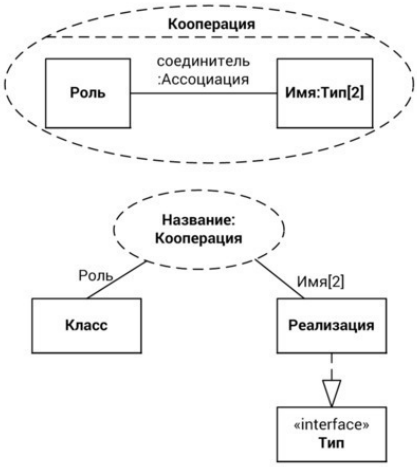


Рис. 6. Основная нотация кооперации

Траекторией (trace) называется частично упорядоченная последовательность возникновений событий (occurrence specification).

Взаимодействием (interaction) называется производное поведение участников, указывающее разрешенные и запрещенные траектории. Участникам сопоставлены **линии жизни (lifeline)**, на которых откладываются возникающие на траекториях события. Когда взаимодействие происходит в контексте динамического структурированного классификатора, линии жизни соответствуют ролям в этом классификаторе, локальным переменным данного взаимодействия или параметрам вызываемых операций и отправляемых сигналов. Если кратность участвующего во взаимодействии свойства, переменной или параметра больше единицы, то для соотнесения линии жизни с определенным значением из нескольких используются **селекторы** (**selector)**.

Сообщения (message), передаваемые в процессе взаимодействия, могут быть нескольких сортов: синхронный и асинхронный вызов операции, асинхронная отправка сигнала, создание и уничтожение экземпляра, и ответные (reply) сообщения. Передача сообщения между линиями жизни отмечается возникновением событий отправки и получения сообщения. Если отправитель или получатель находится вне взаимодействия, вместо него подставляется шлюз (gate).

Фрагмент взаимодействия (interaction fragment) является частью взаимодействия и включает множества разрешенных и запрещенных подпоследовательностей возникновений событий для всех или некоторых линий жизни.

Операторы взаимодействия (interaction operator) используются для изменения траекторий комбинированного фрагмента взаимодействия, состоящего из нескольких фрагментов. Определены операторы альтернативного выбора (alt), цикла (loop), параллельного возникновения событий фрагментов (par), условного выполнения (opt) и другие.

Спецификация исполнения (execution specification), отложенная на линии жизни, указывает на выполнение экземпляром классификатора соответствующего данной линии некоторого исполняемого поведения.

Вхождение взаимодействия (interaction use) служит для повторного использования взаимодействий, вместо фрагмента подставляется содержимое указанного взаимодействия.

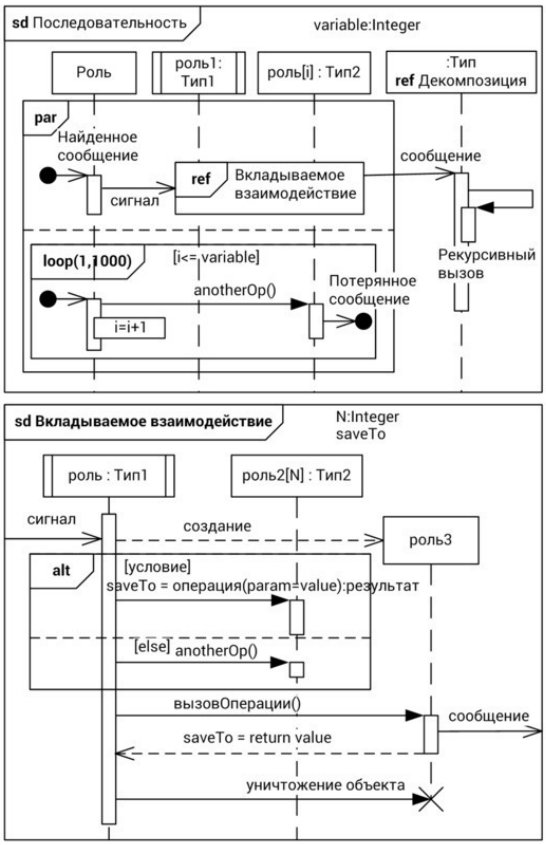


Рис. 7. Основная нотация диаграмм последовательности

ЗАДАЧИ

3.1. (см. решение в §11) Кооперация продажа *Sale* включает роли продавец *Salesman* и покупатель *Customer*.

а. Покажите, что продавец и покупатель могут взаимодействовать друг с другом.

б. Используя вхождения кооперации *Sale*, создайте модель кооперации продажи с посредником *BrokeredSale*, в которой покупатель взаимодействует с посредником *Broker*, а посредник как покупатель взаимодействует с продавцом.

3.2. Моделируется серверная часть веб-приложения интернет-магазина, построенного на основе паттерна Model-View-Controller (MVC). Взаимодействие между ролями *Model*, *View* и *Controller* отобразим на диаграмме последовательности.

а. Разместите на диаграмме роли *Controller*, *Model*, а также роль типа *ORM* с именем *db*. Синхронное найденное сообщение *postBuy (purchase)* приходит на линию жизни *Controller*. После этого *Controller* посылает *Model* синхронное сообщение *addPurchase (purchase)*. В ответном сообщении *Model* возвращает объект *purchaseDetails*.

б. Реализуем на диаграмме поведение *Model* в ответ на сообщение *addPurchase*. *Model* посылает синхронное сообщение *addPurchase (purchase)* линии жизни *db*. Затем открывается фрагмент *alt*. При условии *purchase.needDelivery Model* посылает сообщение *addDelivery(purchase.address)* линии жизни *db*. Фрагмент *alt* окончен. *Model* посылает *db* синхронное сообщение *saveChanges ()*.

в. В ответ на запрос покупки *Controller* должен сообщить пользователю, что заказ совершен успешно. После получения от *Model* ответного сообщения *Controller* создает новую линию жизни с ролью *View* сообщением *createConfirmationView*. Затем *Controller* посылает линии жизни *View* сообщение *setPurchaseDetails (purchaseDetails)*.

Затем *Controller* отправляет в ответ на входящий запрос ответное сообщение, содержащее *View*.

г. Отдел доставки нужно уведомить о том, что требуется доставить новый заказ. Добавим в *Model* (в опциональный фрагмент *alt*) посылку асинхронного сообщения *notifyNewDelivery*. Сообщение является потерянным.

д. Код интернет-магазина достаточно универсальный. Можно сделать на базе этого кода несколько сайтов для разных магазинов. Для этого нужно заполнить *View* при создании информацией о конкретном магазине. Добавим в контекст взаимодействия переменную *shopInfo* типа *ShopInformation*. После сообщения *setPurchaseDetails* начинается фрагмент использования взаимодействия *ref* с именем *FillShopInformation*. В этот фрагмент входят линии жизни *Controller* и *View*. В виде аргумента во взаимодействие передается переменная *shopInfo*.

3.3. (см. решение в §1) Автор *Author* направляет статью сообщением *manuscript* редактору *Editor* и ожидает от него подтверждения получения. Редактор отправляет сообщением *evaluate* статью рецензенту *Peer*. Рецензент отправляет сообщение редактору с оценкой статьи *review*. Редактор направляет сообщение автору с результатами *resolution* и рецензенту с благодарностью *thanks*.

а. Восстановите структурную модель взаимодействия в виде кооперации *ReviewManuscript*, укажите кратность роли рецензента так, чтобы статья направлялась на рецензию одному из пяти рецензентов.

б. Укажите, используя фрагменты, что статья направляется на рецензирование каким-либо трем из пяти рецензентов.

в. Используя фрагменты, покажите, что порядок отправки результатов рецензирования автору и благодарностей рецензентам не имеет значения.

3.4. Терапевт *Therapist* ведет прием посетителей *Person*, в ходе которого выписывает лекарства *Medicine*; посетители принимают лекарства.

а. Постройте логическую модель, включающую классы *Therapist*, *Medicine* и *Person*, и отношения между ними.

б. Используя кооперации, покажите, что на приеме терапевт выполняет обязанности врача *Doctor*, посетитель является пациентом *Patient*, лекарства выписываются в виде рецептов *Subscription*.

3.5. Автомобиль *Car* состоит из двигателя класса *Engine*, пары передних *front* и задних *rear* колес класса *Wheel*.

а. Добавьте привод *drivetrain* так, чтобы автомобиль был переднеприводным.

б. Расширьте модель так, чтобы наряду с переднеприводными автомобилями, она описывала полноприводные автомобили как частный случай переднеприводных. Добавьте необходимые элементы, используйте двигатель *DoubleEngine* с двумя приводами типа *drivetrain*.

3.6. Пассажир *Person* заходит в лифт и нажимает кнопку *pressButton* лифта *Lift* с указанием целочисленного номера этажа *floor*. Лифт закрывает двери и начинает движение синхронным вызовом операции *startMoving*. После этого сообщает пассажиру номера проезжаемых лифтом этажей сообщением *floorReached* с указанием номера этажа. Затем лифт вызывает операцию *stopMoving* и останавливается. Пассажир нажимает кнопку *pressDoors* лифта для открытия дверей.

а. Как можно уточнить модель взаимодействия, если известно, что лифт обслуживает с первого по пятый этажи?

б. Уточните взаимодействие пассажира с лифтом. Укажите, что до нажатия кнопки этажа, пассажир обязан закрыть двери кнопкой *pressDoors*.

в. Используя фрагменты, покажите, что пассажир не может нажать кнопку открытия и закрытия дверей в процессе движения лифта.

г. (*) Укажите, что лифт проезжает один этаж за три секунды.

3.7. Менеджер подключаемых модулей *pluginsManager* класса *PluginsManager* получает сообщение *loadPlugins* – указание на необходимость загрузки доступных модулей. Он синхронно запрашивает у объекта *settings* класса *PluginManagerSettings* пути к директориям с модулями и получает от *settings* значение свойства *pluginsDirs*. После чего в цикле для каждой директории и каждой библиотеки **.dll* загружает модули вызовом собственной операции *loadPlugins*, передавая путь к библиотеке в параметрах.

а. Реализуйте операцию *loadPlugins* класса *PluginsManager*. Взаимодействие начинается с создания нового экземпляра *PluginsDll*, затем идет получение количества подключаемых модулей в библиотеке *getPluginsCount* и получение всех модулей через вызовы *getPlugin* с параметром – номером модуля. После этого происходит инициализация каждого полученного модуля *IPlugin* вызовом метода *initPlugin* класса *PluginsManager*.

б. Добавьте в модель описание действий по инициализации модуля. Метод *initPlugin* проверяет, обрабатывает ли модуль события графического интерфейса вызовом *isUIHandled*. Если обрабатывает, то регистрирует модуль в качестве слушателя событий *addListener* в классе *PlayerUIPresenter*.

3.8. Взаимодействие выбора этажа *SelectFloor* содержит линию жизни кабины, представленной экземпляром активного класса *Cabin*, линию жизни *floor* экземпляра класса кнопки этажа *FloorButton* с селектором *«1»*, и линию жизни класса *Algorithm*. Взаимодействие начинается с синхронного вызова кабиной операции нажатия кнопки *isPressed* на линии *floor*. Операция возвращает логическое значение «истина», если кнопка нажата. Затем экземпляр класса *Cabin* вызывает операцию *selectFloor* у алгоритма на линии в данном взаимодействии.

а. Используя оператор цикла, покажите, что проверяется нажатие кнопок всех этажей. Переменную цикла, содержащую номер кнопки этажа, объявите как атрибут взаимодействия.

б. Уточните взаимодействие, добавив вызовы операции указания алгоритму этажей, кнопки которых нажаты.

в. Пусть взаимодействие описывает поведение кооперации, линии жизни соответствуют ролям в этой кооперации. Каким образом необходимо изменить взаимодействие, если кооперация владеет ролью с линией жизни алгоритма?

г. (*) Приведите по одному примеру разрешенной и неразрешенной траекторий в данном взаимодействии.

3.9. Пользовательский интерфейс мультимедиа проигрывателя *PlayerView* сообщает *ev* контроллеру *UIPresenter* о начале проигрывания элемента списка воспроизведения *playEntry*. Контроллер: 1) рассылает это сообщение всем своим слушателям собственной операцией *raiseEvent*; 2) получает от управляющего компонента *Engine* тип контента по имени *name* выбранного элемента *item* события *ev* вызовом *getContentType*; 3) по типу контента получает медиапоток вызовами *getAudioStream* или *getVideoStream*; 4) переводит интерфейс в состояние проигрывания либо видео, либо аудио вызовом *setPlaybackMode* и указывает медиапоток операцией *setMediaStream*; 5) запускает воспроизведение операцией *play* контроллера.

а. Добавьте в модель возможность параллельной обработки сообщений в *raiseEvent* так, что операция обработки *handleEvent* вызывается асинхронно у каждого зарегистрированного слушателя типа *IUIListener*. Для отображения в модели используйте два анонимных слушателя.

б. Уточните взаимодействие при обработке события запуска проигрывания песни модулем *lyricsPlugin*, зарегистрированным в качестве слушателя событий *UIPresenter*. Модуль определяет тип контента элемента, передаваемый в сообщении о запуске. Для аудиоконтента создает запрос о тексте песни, вызывая собственный метод *makeRequest*. Затем модуль асинхронно вызывает операцию *dispatchRequest* класса *NetController*, передавая в параметрах запрос и себя как обработчика ответа. Получив ответ от сервера, *NetController* передает *processResponse* его обработчику. Модуль вызывает *displayPage* у *UIPresenter* и передает полученную HTML-страницу с текстом песни

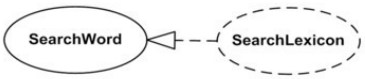


Рис. 8

3.10. (*) При решении основных задач, морфологический модуль ищет запрашиваемое слово в словаре языка. Таким образом, многие задачи в модуле зависят от реализации функции поиска слова в словаре. См. диаграмму на рис. 8. Для уменьшения объема, который занимает словарь, используется тип данных префиксное дерево[2]. Каждый узел дерева содержит ассоциативный массив буква – следующий узел. Таким образом, если слово есть в словаре, то в дереве есть путь, начинающийся в корне и проходящий по вершинам, соответствующим буквам слова.

а. Постройте модель кооперации поиска слова в словаре, используя роли класса *Dictionary* и узла дерева *Node*. Для моделирования префиксного дерева используйте квалифицированные соединители и булевый атрибут *leaf*, указывающий на конечный узел слова.

б. Добавьте в модель поведение кооперации. Актор парсер строки *StringParser* вызывает операцию *hasWord* класса *Dictionary* с параметром *word* типа *String*. Метод *hasWord*, реализующий операцию *hasWord*, получает корневой узел с помощью операции *getRoot* класса *Dictionary*. Получив экземпляр класса *Node*, метод в цикле для каждой следующей буквы слова *word* вызывает у этого экземпляра операцию *getNextLetter* с параметром *c* типа *char* – текущей буквы слова. Данная операция возвращает дочерний узел дерева. Когда буквы слова *word* закончились, нужно вернуть актору значение операции *isLeaf* последнего полученного узла *Node*.

в. Модифицируйте поведение. Если в какой-то момент вызов *getNextLetter* прерван по исключению *NoSuchLetter*, операция *hasWord* должна вернуть *false.*

3.11. Вариант использования просмотр каталога *SearchCatalog* реализован кооперацией *GetAllRecords*. Основной сценарий варианта использования начинается с получения контроллером приложения *AC* команды *showRecords*. *AC* отображает *show* в пользовательском интерфейсе *UI* сообщение «Идет запрос». *AC* параллельно отправляет источнику данных *DataSource* запрос *readRecords*. *DataSource* передает *AC* одну запись в параметре действия *acceptRecord*. Затем *AC* показывает запись в *UI*.

а. Укажите, что перед запросом записи, *AC* запрашивает *getListSize* количество записей у источника данных. Результат присваивается переменной *listSize*.

б. Измените модель так, чтобы источник данных передавал контроллеру *listSize* записей по одной, а контроллер отображал *show* в *UI* все полученные записи вместе.

в. Реализуйте альтернативный сценарий, когда источник данных не содержит записей.

г. Перечислите все необходимые соединители в кооперации *GetAllRecords*, укажите, какие сообщения по ним передаются. Ответ поясните.

[2] Префиксное дерево, «Trie», http://en.wikipedia.org/wiki/Trie.

д. (*) Какое минимальное количество экземпляров классов необходимо, чтобы выполнить описанное поведение? Ответ поясните.

ГЛАВА 2. МЕТОДЫ И ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ

Парадигмы проектирования. Возникновение проектирования программного обеспечения можно связать с появлением языков высокого уровня в 60-х и 70-х годах. Проектирование возникло как дисциплина, целью которой было управление сложностью программных систем и расширение возможностей разработчиков по созданию систем большего размера предсказуемого качества.

В развитии проектирования как дисциплины выделяют несколько этапов, в каждом из которых доминировала одна из парадигм проектирования. В 70-е и 80-е такой была структурная парадигма проектирования. Ее основу составляют нисходящие декомпозиционные методы, итеративно разделяющие систему на функциональные блоки, все более понятные и простые в реализации. Примерами таких методов могут служить метод постепенного уточнения (stepwise refinement) [6], метод структурного анализа и структурного проектирования SSA/SD [6], техника структурного анализа и проектирования SADT[3]. Среди восходящих методов структурного проектирования следует отметить метод структурного проектирования Джексона [6].

Преимущественные нотации моделей в этой парадигме – это структурная схема, схема потоков данных, диаграмма сущность-связь, диаграммы IDEF.

В 80-е и 90-е преобладающими стали методы объектно-ориентированного проектирования. Основой методов является выделение общего описания групп объектов и использование этого описания в качестве абстрактного типа данных. Различные эвристики выделения общего описания и процедуры создания модели нашли выражение в различных методах анализа и проектирования. В результате объединения Objectory, OMT и OOAD был создан унифицированный процесс разработки RUP и язык моделирования UML. Во многом благодаря обсуждению принципов проектирования на страницах таких журналов как C++ Report были сформулированы эвристики повышения изменяемости и сопровождаемости систем SOLID [7]. Следует также отметить методы проектирования, основанные на выделении и группировке обязанностей RDD и связанные с ними эвристики назначения обязанностей GRASP [8]. А также методы, построенные на прямом использовании модели предметной области для построения программной системы. Позднее они нашли отражение в книге Эванса по предметно-ориентированному проектированию [5].

Сейчас объектно-ориентированные методы применяются для разработки отдельных компонент сложных систем. Методы структурного проектирования нашли применение при разработке систем обработки данных, в проектировании архитектуры систем масштаба предприятия.

Систематизация и количественный подход. Существование разных подходов к проектированию в одних и тех же отраслях ведет к необходимости их сравнения и определения предпочтительного и указания границ применимости. Инициатива по выработке базовых методов и теории программной инженерии SEMAT ставит своей целью каталогизацию методов, выработке их описания на основе нескольких базовых понятий и, таким образом, создания основ накопления данных об использовании методов для их последующего сравнения.

[3] Дэвид А. Марка, Клемент Л. МакГоуэн. Методология структурного анализа и проектирования: [Пер. с англ.] / Предисл. Д. Т. Росса. – М.: Фирма «Мета Технология». – 1993. – 240 с.; ил.

Распространенный подход к накоплению знаний в области проектирования – составление каталога повторно применимых приемов решения часто встречающихся задач проектирования, которое могут быть адаптированы для разных случаев – паттернов проектирования. В конце прошлого века были составлены первые каталоги паттернов. Наиболее известным является набор из двадцати двух паттернов «банды четырех» GoF [3]. В сфере высокоуровневого (архитектурного) проектирования аналогом паттернов выступают архитектурные стили [9], комбинации которых, исходя из требований к системе, составляют первоначальное архитектурное описание.

Результатом процесса проектирования являются отраженные в модели решения по реализации программной системы. Для прогнозирования нефункциональных характеристик системы, в том числе сопровождаемости, переносимости и других, вводят показатели проектировочного решения (метрики дизайна). Показатели используют для количественного описания размера системы, сложности решения, выявления недостатков и потенциально проблемных мест в системе. В данной книге рассматриваются показатели сложности проектировочного решения Чидамбера-Кемерера [10].

Применение методов. До автоматического проектирования программных систем пока еще далеко, в проектировании остается существенная доля искусства. В том числе вследствие самой сути задач проектирования, относящихся к так называемым плохо поставленным задачам (wicked problems), условия которых неполны, противоречивы, меняются со временем и в процессе решения. Тем не менее, владение базовыми методами позволяет не отвлекаться на обдумывание задач, решение которых уже известно, и, таким образом, повысить качества результата и сократить время его создания.

Рассматриваемые в сборнике методы проектирования отражают современное состояние практики разработки и проектирования на уровне компонентов и приложений. Их изучение позволит как вступить в специальность проектирования, восполнить отдельные пробелы, так и по-новому взглянуть на уже известную область.

Принципы проектирования классов и интерфейсов SOLID. Аббревиатура SOLID расшифровывается по первым буквам сокращений названий принципов проектирования классов.

Single responsibility principle (SRP), принцип ограничения обязанностей, говорит, что модуль или класс должен иметь только один набор функционально сходных обязанностей.

Open-closed principle (OCP), принцип открытости-закрытости, указывает, что класс или модуль должен быть расширяем (открыт для расширения) без внесения в него изменений (закрыт для изменения).

Liskov substitution principle (LSP), принцип подстановки (описан в статье Барбары Лисков, отсюда название), указывает правило построения иерархии типов так, что любой подтип или дочерний класс подставим вместо базового типа или класса соответственно.

Interface segregation principle (ISP), принцип разделения интерфейса, говорит, что для обозначения разных ролей, которые играет класс в разных взаимодействиях, следует использовать разные интерфейсы.

Dependency inversion principle (DIP), принцип обращения зависимостей, указывает на корректное применение принципа сокрытия информации в объектно-ориентированном подходе к проектированию, когда зависимости направлены от реализации класса к выделенным абстракциям: описаниям типов данных или интерфейса

§4. РАСШИРЕННЫЕ КЛАССЫ И ОБЪЕКТЫ

ОСНОВНЫЕ ПОНЯТИЯ

Квалификатор (qualifier) используется для разделения всех связей ассоциации на подмножества согласно уникальным ключам. Обычно в бинарной ассоциации «один-ко-многим» по ключу выделяют связи «один-к-одному».

Класс ассоциации (association class) используют, когда логическое отношение между объектами обладает сложной структурой или поведением. Класс ассоциации является одновременно и ассоциацией, и классом, поэтому может иметь свойства и операции.

Напомним, что операция – это поведенческая черта класса, которая определяется именем, набором параметров, их типами и кратностями, типом и кратностью возвращаемого значения. В дополнение к этому, можно задать **ограничения** на реализацию операции: **предусловие (precondition)**, **постусловие (postcondition)**, **ограничение возвращаемого значение (body condition)**.

Если операция не изменяет значения свойств класса, то к операции добавляется украшение **запрос** (*query*).

Каждый параметр операции может иметь имя, тип, множественность. Параметру можно задать **направление параметра** (*in*, *out*, *inout*, *return*), значение по-умолчанию, ограничение на множественный параметр: **упорядоченность** значений *(ordered*), **уникальность** значений (*unique*).

Производные свойства класса (derived property) вычисляются на основе других свойств или результатов вызова операций класса или других классов модели. Способ вычисления значений свойства указывается вместе с определением самого свойства, при этом вычисление значений не должно иметь сторонних эффектов и должно быть идемпотентным (повторные вычисления дают тот же результат при неизменности остальной модели). Часто используются производные свойства, значения которых составлены из объединения **union** множеств значений свойств, выделяющих подмножества **subsets** в базовых свойствах.

Шаблонные классы (template class) по сути не являются полноценными классами, а только их заготовками, определенными с точностью до значений параметров шаблона. Шаблонным может быть не только класс, но и другие элементы модели. Операция присвоения значения параметру называется **связыванием (bind)**. По смыслу, связывание класса с шаблонным классом с приданием значения параметру аналогично созданию класса из шаблона с указанным значением параметра и наследованием от этого класса.

Переопределение (derived) позволяет заменить определение черты классификатора в рамках его **контекста переопределения (redefinition context)**, составляющего совокупность всех классов, обобщающих данный. При переопределении можно заменить, например, сигнатуру операции. После переопределения при обращении к новой операции в классе следует использовать новую сигнатуру. При этом наследованная переопределенная операция также доступна для использования.

Множество обобщения (generalization set) позволяет логически группировать отношения обобщения. Можно указать, что в данном множестве обобщений приведены все возможные уточнения базового класса, или что совмещение непосредственно уточняющих классов в одном экземпляре или подклассе не допускается.

Супертип (powertype) используется совместно с множеством обобщений. Супертип это классификатор, экземплярами которого являются классы-элементы множества обобщений.

Пакеты (package) позволяют группировать элементы модели под общим именем. Для обращения к элементу пакета необходимо использовать квалифицированное имя, состоящее из имени пакета и имени элемента.

Между пакетами определены отношения **доступа** *«access»*, которое для элементов пакета делает доступным элементы указанного пакета без необходимости указания квалифицированного имени, и отношение **импорта** *«import»*, которое аналогично доступу, но делает элементы также доступными при последующем импорте или доступе из другого пакета.

Отношение **объединения пакетов (merge)** *«merge»* позволяет объединить в одном элементе определения этого элемента в других пакетах.

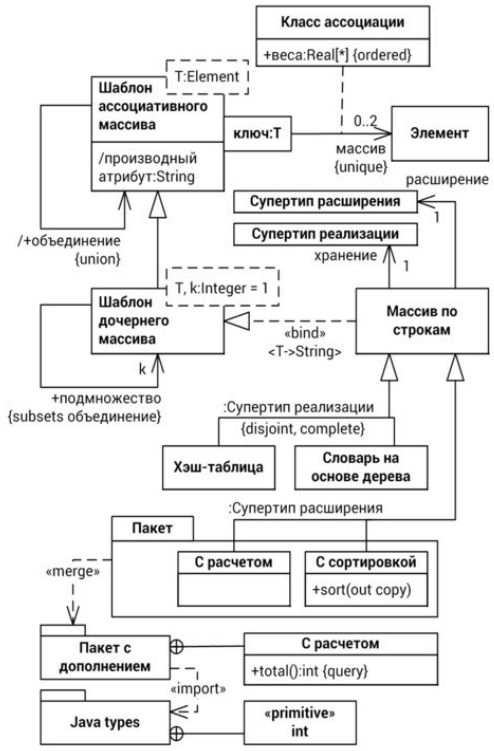


Рис. 9. Расширенная нотация диаграмм классов

Сигналом (signal) называют особый вид классификатора, экземпляром которого является сообщение, передаваемое асинхронно отправителем получателю или группе получателей. Для того, чтобы обрабатывать сигналы, получатель должен быть **активным классом (active class)**, объявлять черту поведения – **получение сигнала (reception)**, с которой может быть связан метод, либо определять собственное поведение, которое обрабатывает поступающие сигналы.

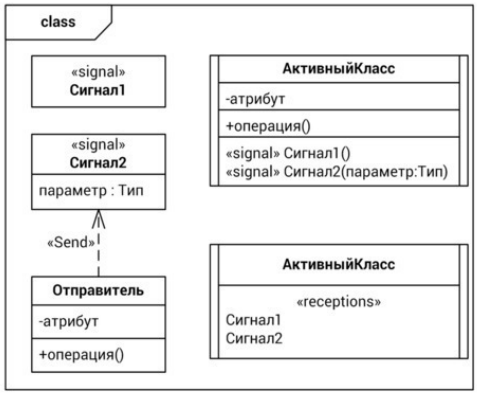


Рис. 10. Нотация сигналов

ЗАДАЧИ

4.1. На рис. 11 представлены шаблонные интерфейсы *Map* и *Entry*. Интерфейс *Map* позволяет по ключу типа *K* получить значение типа *V*. Интерфейс *Entry* представляет собой пару значений.

а. Измените модель так, чтобы шаблон *Entry* использовал параметры шаблона *Map*.

б. Определите интерфейс *Map_StringInteger*, который указывает *String* типом ключа и *Integer* типом значения в шаблоне *Map*.

в. Сколько операций содержит интерфейс *Map_StringInteger*? Ответ поясните.

4.2. Диск *Disk* содержит несколько папок *Folder*, которые могут содержать файлы *File* и папки. Произведения *Composition* хранятся на дисках в виде файлов.

а. Используя классы ассоциаций, постройте модель хранения произведений на дисках.

б. Дополните модель, укажите, что произведение может быть картинкой *Picture*, либо музыкой *Music*, либо фильмом *Movie*.

в. Может ли произведение храниться на одном диске в разных файлах? Ответ поясните.

г. (*) Сравните способы реализации в модели хранения произведения в нескольких файлах на одном диске. Приведите примеры на диаграмме экземпляров.

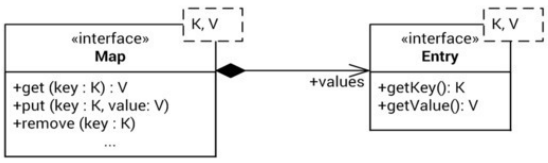


Рис. 11

4.3. На заседании *Meeting* обсуждается *discuss* не менее одного вопроса *Issue*. Вопрос может быть посвящен обсуждению артефакта *Artifact*. В каждом вопросе должно быть указано текстовое название, числовой код и имя автора.

а. Добавьте в модель вопрос по постановлению, отдельный вопрос и сложный вопрос.

б. Укажите, что постановление *Resolution* связано с вопросом по постановлению, как тема *topic*, и с несколькими артефактами *documents*.

в. К сложному вопросу примените паттерн Composite так, чтобы сложный вопрос включал несколько других вопросов.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «ЛитРес».

Прочитайте эту книгу целиком, купив полную легальную версию на ЛитРес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.