Александр Леоненков

# Самоучитель UML 2

Санкт-Петербург

«БХВ-Петербург»

2007

УДК 681.3.06
ББК 32.973.26-018.1
Л47

**Леоненков А. В.**

Л47 Самоучитель UML 2. — СПб.: БХВ-Петербург, 2007. — 576 с.: ил.

ISBN 978-5-94157-878-8

Рассмотрена современная технология объектно-ориентированного анализа и проектирования программных систем и бизнес-процессов в контексте нотации унифицированного языка моделирования UML 2. Подробно изложены все понятия языка UML 2 в полном соответствии с оригинальной спецификацией последней версии этого языка. Приведены конкретные рекомендации по разработке канонических диаграмм языка и рассмотрены особенности разработки моделей с помощью CASE-средства Borland® Together® Designer. Описана нотация OCL — языка объектных ограничений, по которому практически отсутствует информация на русском.

*Для системных и бизнес-аналитиков, архитекторов программ, руководителей проектов и информационных служб, корпоративных программистов и студентов*

УДК 681.3.06
ББК 32.973.26-018.1

**Группа подготовки издания:**

| | |
|---|---|
| Главный редактор | *Екатерина Кондукова* |
| Зам. главного редактора | *Игорь Шишигин* |
| Зав. редакцией | *Григорий Добин* |
| Редактор | *Анна Кузьмина* |
| Компьютерная верстка | *Натальи Караваевой* |
| Корректор | *Зинаида Дмитриева* |
| Дизайн серии | *Инны Тачиной* |
| Оформление обложки | *Елены Беляевой* |
| Зав. производством | *Николай Тверских* |

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.12.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 46,44.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-94157-878-8

# Оглавление

# Предисловие

Книга посвящена последней версии унифицированного языка моделирования или, сокращенно, языка UML (Unified Modeling Language), который предназначен для описания, визуализации и документирования объектно-ориентированных систем и бизнес-процессов с ориентацией на их последующую реализацию в виде программного обеспечения.

Хотя в настоящее время отечественные разработчики применяют на практике несколько нотаций визуального моделирования, именно нотация UML все более активно используется системными аналитиками, архитекторами и руководителями проектов для разработки графических моделей при выполнении программных проектов. При этом последние версии интегрированных сред разработки приложений MS Visual® Studio .NET® и Borland® Delphi® Studio не только поддерживают нотацию языка UML в качестве базовой технологии моделирования программных систем, но и позволяют получить исполнимые модули программ на основе разработанной графической модели.

Возрастающая сложность прикладных программ и высокая стоимость их разработки и сопровождения практически не оставляют выбора в вопросе: стоит ли изучать язык UML? В разработке современных корпоративных информационных систем принимают участие десятки, а то и сотни различных специалистов, для которых построение предварительной модели системы до начала написания соответствующего программного кода становится настоятельной необходимостью. Разработка программных систем на заказ также приводит к необходимости поддержания единого стиля для различных версий программ при их постоянной доработке и модификации.

Основное требование к модели программной системы — модель должна быть понятна как заказчику, так и всем специалистам проектной группы, включая и программистов. Оказалось, что разработка соответствующего языка моделирования или нотации является непростым делом. Потребовалось

несколько лет, прежде чем усилия группы специалистов ведущих фирм-производителей программного и аппаратного обеспечения привели к появлению языка UML 2.0.

Какими бы свойствами ни обладал новый язык, важной особенностью для его жизнеспособности является реализация и поддержка соответствующей нотации в коммерческих программных продуктах. В последние годы практически наблюдается все более активный интерес к языку UML, о чем свидетельствуют десятки коммерческих программных инструментариев, предназначенных для автоматизации разработки программного обеспечения на основе построения предварительной объектно-ориентированной модели предметной области. Одним из первых инструментальных средств, в котором была реализована разработка моделей в нотации UML 2.0, является программа Borland® Together® Designer.

Однако наибольшее впечатление производит возможность синхронизации кода и моделей в нотации UML при интеграции этой программы с инструментальными средами программирования, такими как Borland® Together® Developer for JBuilder® и Borland® Together® Developer for MS .NET®. И, наконец, нельзя не упомянуть о среде Borland® Delphi® Studio 2006, в которой реализованы в полном объеме возможности разработки приложений, управляемых моделями в нотации UML. Изучив материал книги, читатели без особого труда смогут освоить редактор моделей в Borland® Delphi® Studio и самостоятельно разрабатывать управляемые моделями приложения в этой среде.

Язык UML 2.0 находит эксклюзивное применение во многих современных информационных технологиях, среди которых следует отметить концепцию управляемой моделями архитектуры (Model Driven Architecture, MDA), в рамках которой поддерживается управляемый моделями подход для разработки программного обеспечения. При разработке платформенно-независимых моделей (PIM), которые концентрируют внимание на общей архитектуре системы, используется только нотация языка UML 2.0.

Следует отметить еще одну область, в которой язык UML 2.0 является единственным средством визуального моделирования и документирования. Это так называемые паттерны (patterns) проектирования, которые концентрируют в себе положительный архитектурный опыт разработки программных приложений. Уже сейчас в некоторых интегрированных средах программирования можно разработать нетривиальное приложение, не написав ни одной строчки программного кода. И эта тенденция, как это не может показаться парадоксальным, со временем будет только усиливаться.

# Структура книги

Материал книги соответствует последней версии языка UML 2.0. В основу книги положены две основные идеи. С одной стороны, рассмотреть все базовые конструкции языка UML 2.0, без понимания которых вряд ли возможно адекватно и безошибочно использовать богатейший потенциал возможностей языка UML 2.0. С другой стороны, донести до читателя основы методологии объектно-ориентированного анализа и проектирования, которая необходима для самостоятельной разработки концептуальных, логических и физических моделей программных систем и бизнес-процессов.

Материал книги делится на три части. *Часть I* знакомит с основными понятиями объектно-ориентированного анализа и проектирования, а также с историей развития языка UML 2.0, что представляется совершенно необходимым для правильного понимания назначения и возможностей языка UML 2.0. Поскольку UML 2.0 не является формальным языком с фиксированным синтаксисом, описание языка рассматривается как некоторая открытая модель с определенными базовыми семантическими конструкциями и неформальными правилами их расширения.

*Часть II* является центральной в книге и содержит описание назначения элементов всех канонических диаграмм языка UML 2.0, которые являются основой построения концептуальных, логических и физических моделей. В отдельных главах этой части последовательно рассматриваются все 13 канонических диаграмм языка UML 2.0:

- диаграмма вариантов использования;
- диаграмма классов;
- диаграмма композитной структуры;
- диаграмма пакетов;
- диаграмма объектов;
- диаграмма последовательности;
- диаграмма деятельности;
- диаграмма коммуникации;
- диаграмма обзора взаимодействия;
- временная диаграмма;
- диаграмма конечного автомата;
- диаграмма компонентов;
- диаграмма развертывания.

Для каждой из диаграмм описываются все базовые элементы графической нотации, даются определения всех ключевых терминов, рассматривается их семантика и особенности графического изображения на различных диаграммах, а также приводятся примеры диаграмм, иллюстрирующие соответствующие понятия языка UML 2.0. Материал этих двух частей книги абсолютно не зависит от средств реализации языка UML 2.0 и может быть использован при разработке моделей с использованием любого CASE-средства, например, IBM Rational® Software Architect (IBM Corp) или Enterprise Architect (Sparx Systems).

*Часть III* содержит описание особенностей реализации языка UML 2.0 в уже упомянутом CASE-инструментарии — Borland® Together® Designer. В этой части также представлен сквозной пример — последовательная разработка всех диаграмм системы управления банкоматом. После его изучения читатели без особого труда смогут разработать соответствующие диаграммы и в любых других CASE-средствах, поддерживающих нотацию языка UML 2.0.

## Рекомендации по изучению языка UML

Современные тенденции развития индустрии создания программного обеспечения складываются таким образом, что именно язык UML 2.0 де-факто оказывается общепризнанным стандартом в области разработки моделей систем и процессов с его последующей реализацией в соответствующих инструментальных средствах.

В то же время следует отметить одну важную особенность современной программной инженерии — это тенденция специализации в области разработки программ. Речь идет о появлении таких специалистов, как системный аналитик, менеджер проекта, бизнес-аналитик и архитектор системы, которые, наряду со знанием языков программирования, должны владеть методологией объектно-ориентированного анализа и моделирования предметной области. Для системного инженера и интегратора также важно разбираться в возможностях реализации конкретных проектов и использовать общепринятую систему обозначений для решения своих задач. Наконец, корпоративные программисты, занятые в масштабных проектах, должны четко понимать функциональные аспекты будущей программной системы. Для всех этих категорий специалистов и предназначена данная книга, поскольку язык UML 2.0 позволяет организовать эффективное и понятное для всех общение.

Уже сейчас элементы нотации языка UML 2.0 начинают использоваться в учебных программах для обучения студентов технических специальностей. Во всяком случае, многие преподаватели давно осознали ограниченность

существующих отечественных стандартов на разработку программной документации, которые не отражают современных тенденций развития программной инженерии. Как студенты, так и преподаватели найдут в книге интересный материал для размышления, который позволит понять целый ряд особенностей и перспектив профессионального образования в области современных информационных технологий.

Для понимания основных конструкций языка UML 2.0 достаточно общей эрудиции. Читатели, которые ставят перед собой такую цель, могут бегло просмотреть материал первой части и сразу приступить к изучению отдельных диаграмм. Однако для более углубленного его изучения полезно знакомство с одним из языков программирования. Для творческого овладения методологией объектно-ориентированного анализа и проектирования необходима, как представляется, определенная строгость мышления и знание некоторых общих понятий прикладного системного анализа. Соответствующий материал приводится в первой части книги и далее используется по мере изложения элементов конкретных диаграмм.

Для лучшего восприятия изложенного в книге материала используются следующие символы.


Этот символ указывает на определение отдельных терминов языка UML 2.0 или ООАП. При этом приводится оригинальное имя на английском и его перевод.

Этот символ обращает внимание на важную особенность или правило языка UML 2.0. Невыполнение или игнорирование этого правила или условия может послужить источником ошибок в представлении графических диаграмм языка UML 2.0.


Этот символ указывает на примечание, которое уточняет или дополняет основной материал. Приводимый здесь текст не оказывает влияния на нотацию и семантику языка UML 2.0.

# Благодарности

Автор искренне признателен всем сотрудникам УКЦ Interface Ltd (**www.interface.ru**) за совместную творческую работу. В понимании отдельных вопросов существенную помощь оказали неформальные консультации с сотрудниками компании Borland, которым автор выражает свою признательность. Написание современной книги немыслимо без использования ресурсов

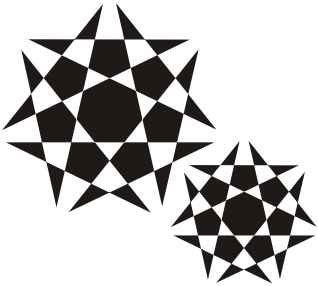
Интернета. В этой связи хотелось бы выразить неизменную признательность директору Междисциплинарного Центра СПбГУ (**www.icape.nw.ru**) профессору Н. В. Борисову за предоставленную возможность электронной коммуникации.

# Постскриптум

В основу написания книги положена оригинальная спецификация языка UML 2.0, которая за время работы над книгой дважды меняла свою редакцию. При этом изменился, а вернее, существенно увеличился состав ее авторов. Это не могло не отразиться на содержании спецификации. Именно по этой причине автор позволил себе в отдельных случаях сделать некоторые уточнения, способствующие, по его мнению, лучшему пониманию конструкций языка UML 2.0.

Последняя версия языка UML 2.0 во многих аспектах серьезно отличается от предыдущих версий. Не имея представления о многочисленных нюансах этих отличий, работа над книгой вначале представлялась автору несложной и связанной, главным образом, с дополнением и уточнением семантики отдельных понятий. На деле все оказалось иным, поскольку версия языка UML 2.0 в действительности оказалась и сложной, и масштабной. Такой же оказалась и работа над книгой.

Завершая работу, автор искренне надеется, что читатели оценят этот труд, поскольку книги вполне достаточно для самостоятельного овладения всеми конструкциями языка UML 2.0. Если же ее окажется недостаточно, то всех читателей автор будет рад видеть на своих курсах по соответствующей тематике в УКЦ Interface Ltd. В любом случае автор будет признателен за все отзывы и конструктивные предложения, связанные с содержанием книги и проблематикой моделирования в контексте языка UML 2.0, которые можно отправлять по адресу издательства "БХВ-Петербург" **mail@bhv.ru**.

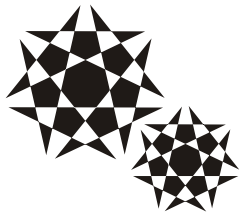


# Часть I

# Основы объектно-ориентированного анализа и проектирования информационных систем

Данная часть книги посвящена изложению концептуальных основ новой методологии разработки программных приложений — объектно-ориентированному анализу и проектированию. Хотя первые графические системы обозначений для описания структуры программ возникли еще в процедурном программировании, именно в рамках этой новой методологии получил развитие унифицированный язык моделирования — язык UML. Для понимания основ этого языка необходимо не только владеть терминологией объектно-ориентированного программирования, но и иметь общее представление о возможностях визуального моделирования в контексте концепции разработки архитектур, управляемых моделями. В этой части книги излагаются базовые понятия моделирования систем и программных приложений, а также приводятся сведения по наиболее распространенным графическим нотациям. Описываются основные элементы языка UML 2.0 с их краткой характеристикой. В последующем эти элементы рассматриваются более подробно при изучении отдельных видов канонических диаграмм в *части II*.

# Глава 1



# Базовые принципы и понятия технологии разработки объектно-ориентированных информационных систем

Интенсивное развитие компьютерных и информационных технологий в последние годы тесно связано с использованием все более сложных концепций, которые аккумулируют в себе самые последние достижения в сфере науки и техники. Внедрение в практику современного бизнеса корпоративных информационных систем, адаптация их функционала к нуждам конкретных компаний, выполнение масштабных проектов по разработке программных систем и аппаратно-программных комплексов происходит на фоне постоянного усложнения как физических и программных компонентов, так и лежащих в их основе концепций и идей. Целый ряд конкурентных преимуществ в современном бизнесе тесно связан с использованием последних достижений в области информационных технологий.

Одновременно с увеличением стоимости информационных проектов заметно возросли риски успешного их выполнения. Анализ отрицательных результатов свидетельствует о вполне сложившихся тенденциях, среди которых, наряду с низкой квалификацией персонала рабочих групп и менеджеров проектов, одно из главных мест занимает отсутствие адекватных моделей и архитектур разрабатываемых программных приложений. Все эти особенности приводят к настоятельной необходимости моделирования структуры и процесса функционирования программных систем до начала написания соответствующего кода. При этом непременным условием успешного завершения проекта становится построение предварительной визуальной модели программной системы.

# 1.1. Основные понятия моделирования систем и программных приложений


*Модель (model)* — абстракция произвольной системы или объекта, рассматриваемая с определенной точки зрения и представленная на некотором языке или в графической форме.

Модель является абстракцией физической или ментальной системы в контексте некоторой цели. Эта цель определяет, что должно быть включено в модель, а что является неинтересным и может быть проигнорировано без ущерба для ее понимания. Другими словами, модель должна полностью описывать только те аспекты системы, которые являются релевантными конкретной цели моделирования на подходящем уровне детализации.

Общим свойством всех моделей является их подобие оригинальной системе или системе-оригиналу. Важность построения моделей заключается в возможности их использования для получения информации о свойствах или поведении системы-оригинала. При этом процесс построения и последующего применения моделей для получения информации о системе-оригинале получил название *моделирование*.

Модель состоит из множества элементов, которые совместно описывают моделируемую систему. При этом модель сложной системы может быть представлена в некоторой иерархической форме и в виде различных представлений. Каждое такое представление специфицирует рассматриваемую систему с определенной точки зрения в интересах определенной категории заинтересованных лиц, например разработчиков, пользователей или клиентов этой системы, и на определенном уровне абстракции.


*Представление (view)* — проекция модели, которая рассматривается с определенной точки зрения и учитывает только существенные аспекты модели.

Модель может считаться законченной, если она представляет физическую систему в целом, хотя в ней могут быть представлены только те аспекты, которые являются релевантными ее цели, т. е. на заданном уровне абстракции и с заданной точки зрения. Элементы модели могут быть организованы в некоторую иерархию, в которой наиболее верхнее представление или подсистема выступает в качестве границы рассматриваемой системы.

Важнейшими характеристиками любой системы являются ее структура и процесс функционирования. При этом под *структурой системы* понимают

устойчивую во времени совокупность взаимосвязей между ее элементами или компонентами. Именно структура связывает воедино все элементы и препятствует распаду системы на отдельные составляющие ее элементы. Структура системы может отражать самые различные взаимосвязи, в том числе и вложенность элементов одной системы в другую. В этом случае принято называть более мелкую или вложенную систему *подсистемой*, а более крупную — *метасистемой*.

Структуру программных систем на верхнем уровне представления часто называют архитектурой, понимая под этим термином наиболее общие структурные компоненты приложений и взаимосвязи между ними.

Процесс функционирования системы отражает поведение системы во времени и может быть представлен как последовательное изменение ее состояний. Если система изменяет одно свое состояние на другое состояние, то принято говорить, что система переходит из одного состояния в другое. Совокупность признаков или условий изменения состояний системы в этом случае называется переходом. Для системы с дискретными состояниями процесс функционирования может быть представлен в виде последовательности состояний с соответствующими переходами.

Для одной и той же физической системы могут быть разработаны различные модели, которые могут отличаться формой и уровнем представления информации об этой системе. При этом различные модели должны дополнять друг друга и специфицировать разные аспекты системы с различных перспектив (точек зрения). Модели могут уточнять друг друга и декомпозироваться на более детальные представления. Обязательное наличие нескольких моделей для адекватного представления особенно характерно для моделирования сложных систем.

Сложность системы и, соответственно, ее модели может быть рассмотрена с различных точек зрения. Прежде всего, можно выделить сложность структуры системы, которая характеризуется количеством элементов системы и различными типами взаимосвязей между этими элементами. Если количество элементов превышает некоторое пороговое значение, которое, вообще говоря, не является фиксированным, то такая система может быть названа сложной. Например, если программное приложение насчитывает более 100 отдельных форм ввода и вывода информации, то многие программисты сочтут ее сложной. Транспортная система современных мегаполисов также может служить примером сложной системы.

Вторым аспектом сложности является сложность процесса функционирования системы. Это может быть связано как с непредсказуемым характером поведения системы, так и невозможностью формального представления правил преобразования входных воздействий в выходные. В качестве примеров

сложных программных систем можно привести современные операционные системы, которым присущи черты сложности как структуры, так и поведения.

Процесс разработки моделей можно представить как поуровневый спуск от наиболее общих моделей и представлений концептуального уровня к более частным и детальным представлениям логического и физического уровня. При этом на каждом из этапов процесса разрабатываемые модели последовательно дополняются все большим количеством деталей, что позволяет им более адекватно отражать различные аспекты реализации сложной системы. Общая архитектура моделей и представлений сложной системы схематично изображена на рис. 1.1.


**Рис. 1.1.** Общая архитектура моделей и представлений сложной системы

Форма представления моделей тесно связана с исходными целями моделирования. В практике разработки программных систем и приложений наибольшее распространение получили визуальные модели, которые используют для представления своих элементов специальную графическую нотацию.


*Нотация (notation)* — система условных обозначений, специально разработанная для представления элементов модели в графической форме.

Важным достоинством той или иной графической нотации является возможность образного закрепления содержательного смысла отдельных понятий за тем или иным геометрическим символом, что существенно упрощает процесс моделирования и интерпретации соответствующих моделей среди разработчиков. Основное требование к модели программной системы — модель должна быть понятна заказчику и всем специалистам проектной группы, включая бизнес-аналитиков и программистов. Именно для разработки такой нотации потребовались усилия группы специалистов ведущих фирм производителей программного и аппаратного обеспечения, которые привели к появлению языка UML.


*Унифицированный язык моделирования (Unified Modeling Language, UML)* — язык визуального моделирования, предназначенный для спецификации, визуализации и документирования объектно-ориентированных систем и бизнес-процессов во время их проектирования и разработки.

Следует отметить, что довольно часто встречаются неточные варианты перевода этого термина и неверная интерпретация сокращения UML. Так, например, даже в англо-язычных источниках UML иногда называют универсальным языком, забывая его первоначальное определение.

Использование той или иной нотации, обладающей широким набором графических символов для представления визуальных моделей, вообще говоря, еще не отвечает на вопрос о том, каким образом и в какой последовательности необходимо разрабатывать сами модели. Если ранние нотации изображения схем программных приложений, как правило, содержали рекомендации по методике их построения, то с появлением более сложных нотаций ситуация кардинально изменилась. Главной особенностью развития современных технологий разработки программных систем стало появление различных методологий, которые детально описывают сам процесс разработки независимо от используемых при этом нотаций.


*Методология (methodology)* — совокупность принципов и методик разработки программных систем и приложений, определяющих процесс построения моделей при выполнении соответствующих проектов.

Современные методологии разработки программных приложений тесно связаны с концепцией автоматизированной разработки программного обеспечения

(Computer Aided Software Engineering, CASE) и соответствующими программными средствами, реализующими эту концепцию.

Хотя появление первых CASE-средств было встречено с некоторой настороженностью, со временем появились как восторженные отзывы об их применении, так и критические оценки их возможностей. Причин для столь противоречивых мнений было несколько. Первая из них заключается в том, что ранние CASE-средства были простой надстройкой над некоторой системой управления базами данных (СУБД). Хотя визуализация процесса разработки концептуальной схемы БД имеет немаловажное значение, она не решает проблем разработки приложений других типов.


*CASE-средства (CASE-tools)* — программное обеспечение, которое предназначено для разработки визуальных моделей программных приложений и генерации исходного кода на некотором языке программирования или схемы базы данных.

Вторая причина имеет более сложную природу, поскольку связана с графической нотацией, реализованной в том или ином CASE-средстве. При этом попытки предложить универсальную систему обозначений, как для визуального представления концептуальных схем БД, так и для архитектуры программных приложений, закончились безрезультатно. На этом фоне разработка и стандартизация унифицированного языка моделирования UML были восприняты с большим оптимизмом всем сообществом корпоративных программистов.

Современные CASE-средства не только поддерживают одну или несколько графических нотаций, но и могут быть встроены в интегрированные среды разработки приложений, такие как Borland Studio или MS Visual Studio. Дополнительная поддержка технологий синхронизации программного кода и визуальных моделей позволяет не только реализовать двунаправленную разработку приложений, но и существенно сократить сроки выполнения проектов. Взаимосвязь нотации, методологии и CASE-средств при выполнении проектов можно условно изобразить в форме треугольника, при этом выбор конкретных элементов представляет лишь один из возможных вариантов, который и рассматривается в данной книге (рис. 1.2).

Разработка и использование моделей в нотации UML осуществляется в рамках общей методологии объектно-ориентированного анализа и проектирования, которая, в свою очередь, является обобщением и дальнейшим развитием методологии объектно-ориентированного программирования. Далее рассматриваются основные принципы и понятия этой методологии, которые служат концептуальным базисом для понимания всех основных понятий языка UML 2.0.


**Рис. 1.2.** Взаимосвязь нотации, методологии и CASE-средств

## 1.2. Методология объектно-ориентированного анализа и проектирования

Методология объектно-ориентированного анализа и проектирования, которая пришла на смену методологии объектно-ориентированного программирования, существенно расширила сферу своих интересов, включив в нее не только процесс написания программного кода, но и разработку моделей, а также анализ архитектур приложений. Наиболее существенным в новой методологии явилось осознание того факта, что процесс написания программного кода может быть отделен от процесса проектирования структуры программы.

Действительно, до того, как начать программирование классов, их свойств и методов, необходимо дать ответы на такие вопросы, как: сколько и какие классы нужно определить для решения поставленной задачи, какие свойства и методы необходимы для придания классам требуемого поведения, а также установить взаимосвязи между классами. Эта совокупность задач не столько связана с написанием кода, сколько с общим анализом требований к будущей программе, а также с анализом конкретной предметной области, для которой разрабатывается программа.


Объектно-ориентированный анализ и проектирование (Object-Oriented Analysis/Design) — методология разработки программных систем, в основу которой положена объектно-ориентированная концепция представления моделей предметной области в форме классов, обладающих структурными свойствами и поведением.

Фундаментальными понятиями методологии ООАП являются собственно понятия класса и объекта, а также ее основные принципы — абстракция, наследование, инкапсуляция и полиморфизм.


*Абстракция (abstraction)* — характеристика сущности, которая отличает ее от других сущностей.

Абстракция определяет границу представления соответствующего элемента модели и используется для определения фундаментальных понятий ООАП, которыми являются понятия класса и объекта.


*Класс (class)* — абстракция совокупности реальных объектов, которые имеют общий набор свойств и обладают одинаковым поведением.

Каждый объект в этом случае рассматривается как экземпляр соответствующего класса. Объекты, которые не имеют полностью одинаковых свойств или не обладают одинаковым поведением, по определению не могут быть отнесены к одному классу.

Важной особенностью классов является возможность их организации в виде некоторой иерархической структуры, которая по внешнему виду напоминает схему классификации понятий формальной логики. Иерархия понятий строится следующим образом. В качестве наиболее общего понятия или категории выбирается понятие, имеющее наибольший объем и, соответственно, наименьшее содержание. Это самый высший уровень абстракции для рассматриваемой иерархии. Затем наиболее общее понятие некоторым образом конкретизируется, тем самым уменьшается его объем, но увеличивается содержание. Появляется менее общее понятие, которое на схеме иерархии располагается на уровень ниже исходного понятия.

Этот процесс конкретизации понятий может быть продолжен до тех пор, пока на самом нижнем уровне не будут получены понятия, дальнейшая конкретизация которых в данном контексте либо невозможна, либо нецелесообразна. Если в качестве понятий использовать классы, то такая иерархия может служить иллюстрацией принципа наследования ООАП.


*Наследование (inheritance)* — принцип, в соответствии с которым знание о более общей категории разрешается применять для более частной категории.

Наследование тесно связано с иерархией классов, которая определяет, какие классы следует считать наиболее абстрактными и общими по отношению к другим классам. При этом если некоторый общий или родительский класс (предок) обладает фиксированным набором свойств и поведением, то любой наследуемый от него класс (потомок) должен содержать этот же набор свойств и поведение, а также, возможно, иметь дополнительные, которые будут характеризовать уникальность данного класса-потомка. В этом случае просто говорят, что класс-потомок наследует свойства и поведение класса-предка.

Для иллюстрации принципа наследования можно привести следующий пример. Рассмотрим в качестве общего класс `Автомобиль`. Этот класс определяется как некоторая абстракция свойств и поведения всех реально существующих автомобилей. При этом свойствами класса `Автомобиль` могут быть такие общие свойства, как наличие двигателя, трансмиссии, колес и рулевого управления. Если в качестве класса-потомка рассмотреть класс `Легковой автомобиль`, то все выделенные выше свойства будут присущи и этому классу. Можно сказать, что класс `Легковой автомобиль` наследует свойства родительского класса `Автомобиль`. Однако кроме перечисленных свойств этот класс-потомок будет содержать дополнительные свойства, например, такое как наличие салона с количеством посадочных мест 2—5.

В свою очередь, класс `Легковой автомобиль` может быть классом-предком для других классов, которые вполне могут соответствовать, например, моделям конкретных фирм-производителей. С этой точки зрения можно рассматривать класс `Легковой автомобиль производства ВАЗ`. Поскольку Волжский автомобильный завод выпускает несколько моделей автомобилей, одним из классов-потомков для предыдущего класса может быть конкретная модель автомобиля, например, ВАЗ-2110. Рассмотренный фрагмент классификации автомобилей может быть представлен графически в форме диаграммы классов в нотации UML 2.0 (рис. 1.3).


Следует заметить, что диаграмма классов на рис. 1.3 является неполной в том смысле, что на ней не представлены другие модели автомобилей. В дополнение к этому все классы, кроме классов самого нижнего уровня, являются абстрактными. При этом никакие свойства классов здесь не указаны. Все это должно навести на серьезные размышления по поводу требований к графической нотации, которая должна позволять представлять дополнительную информацию о классах.

В языке UML 2.0 для представления классификаций и наследования классов используется специальное отношение обобщения *(см. главу 3)*.

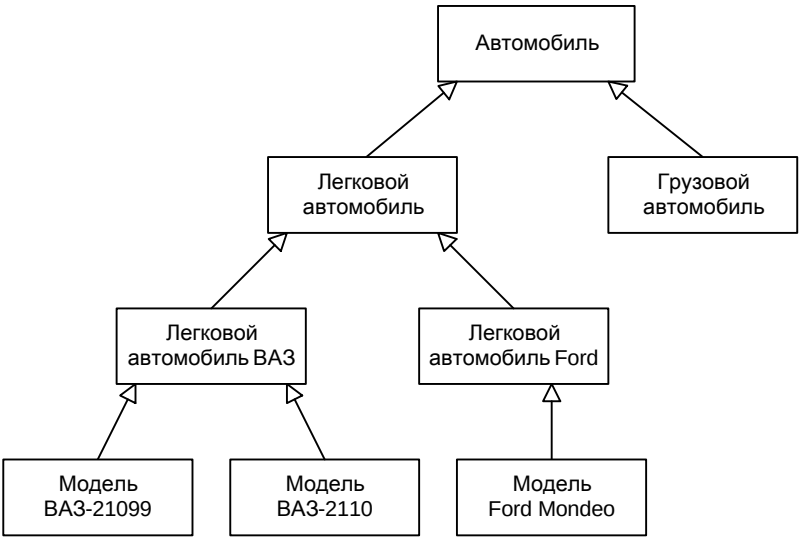



**Рис. 1.3.** Фрагмент схемы классификации автомобилей в нотации UML 2.0

Следующий принцип ООАП — инкапсуляция.


*Инкапсуляция (encapsulation)* характеризует сокрытие отдельных деталей внутреннего устройства классов или компонентов от внешних по отношению к нему объектов или пользователей.

Действительно, взаимодействующему с классом клиенту нет необходимости знать, каким образом реализован тот или иной метод класса, услугами которого он решил воспользоваться. Конкретная реализация присущих классу свойств и методов, которые определяют поведение этого класса, является собственным делом данного класса. Более того, отдельные свойства и методы класса вообще могут быть невидимы за пределами этого класса, что является основной идеей введения различных форм видимости для элементов класса.

Если продолжить рассмотрение примера с классом `Легковой автомобиль`, то нетрудно проиллюстрировать инкапсуляцию следующим образом. Основным пользователем, который взаимодействует с объектами этого класса, является водитель. Очевидно, что не каждый водитель в совершенстве знает внутреннее устройство легкового автомобиля. Более того, отдельные детали этого устройства сознательно скрыты в корпусе двигателя или коробке передач. А в случае нарушения работы автомобиля или при возникновении неисправностей в работе его подсистем необходимый ремонт выполняет профессиональный механик.