Основы *UML моделирования*

- Метод объектно-ориентированного анализа позволяет описывать реальные сложные системы наиболее адекватным образом.
- Но с увеличением сложности систем возникает потребность в хорошей технологии моделирования.
- В качестве такой "стандартной" технологии используется унифицированный *язык моделирования UML* (*Unified Modeling Language*)

Назначение языка

- *UML* - унифицированный *язык объектно-ориентированного моделирования.*
- *UML* - язык формальный и искусственный.
- *UML* - язык графический.
- При описании формального искусственного языка описываются такие его элементы, как:
  - *Синтаксис* ;
  - *Семантика* ;
  - *Прагматика*.

Историческая справка

- В 80-е годы было множество различных методологий моделирования. Каждая из них имела свои достоинства и недостатки, а также свою нотацию. Разные люди использовали разные нотации, и для того чтобы понять, что описывает та или иная *диаграмма*, зачастую требовался "переводчик". Один и тот же символ мог означать в разных нотациях абсолютно разные вещи.
- К 1994-му существовало 72 метода, или частные методики. Многие из них "перекрывались", т. е. использовали похожие идеи, нотации и т. д.

Появление *ООП* требовало удобного инструмента для моделирования, единой нотации для описания сложных программных систем.

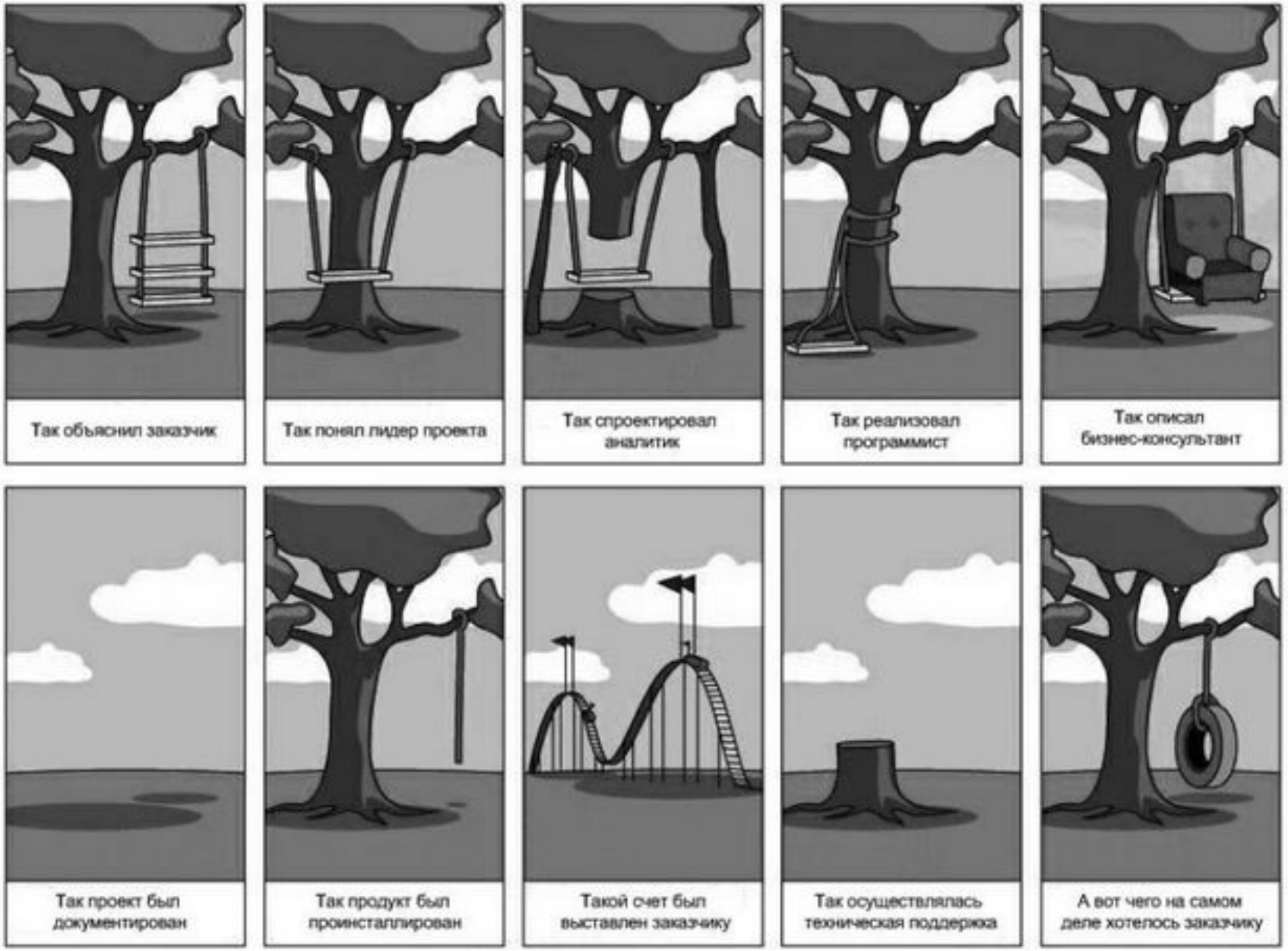
UML вобрал в себя черты нотаций Грейди Буча (Grady Booch), Джима Румбаха (Jim Rumbaugh), Айвара Якобсона (Ivar Jacobson) и многих других.

Эти три крупнейших специалиста, три автора наиболее популярных методов решили объединить свои разработки. В 1991-м каждый из них начал с написания книги, в которой изложил свой метод ООП. Каждая методология была по-своему хороша, но каждая имела и недостатки.

- В настоящее время унифицированный *язык моделирования* *UML* является самой используемой технологией в области программной инженерии.
- *UML* позволяет системным архитекторам представлять свое видение системы в виде набора стандартных диаграмм, которые, к тому же, служат отличным средством коммуникации в команде разработчиков и прекрасным помощником в общении с заказчиком.
- При этом, *UML* - достаточно логичная и простая для изучения *нотация*, навыками использования которой должен овладеть любой специалист, собирающийся работать в области программной инженерии.
- *Знание UML* нужно разработчикам, системным архитекторам, менеджерам...

Способы использования языка

Рассмотрим демонстрацию известной картинки, которая уже более двух десятилетий "живет" в Интернете, но источник ее никому не известен. Эта картинка прекрасно иллюстрирует типичный процесс создания продукта, или "решения" (поскольку продукт решает проблему заказчика)


Так объяснил заказчик
Так понял лидер проекта
Так спроектировал аналитик
Так реализовал программист
Так описал бизнес-консультант
Так проект был документирован
Так продукт был проинсталлирован
Такой счет был выставлен заказчику
Так осуществлялась техническая поддержка
А вот чего на самом деле хотелось заказчику

Авторы *UML* определяют его как графический *язык моделирования* общего назначения (т. е. его можно применять для проектирования чего угодно - от простой качели, до сложного аппаратно-программного комплекса или даже космического корабля), предназначенный для

- **спецификации**,
- **визуализации**,
- **проектирования**
- **документирования**

всех артефактов, создаваемых в ходе разработки.

Спецификация

- Спецификация - подробное описание системы, которое полностью определяет ее цель и функциональные возможности. Различают:
  - словесные спецификации на естественном языке;
  - модельные спецификации;
  - формальные спецификации.

UML позволяет создавать такие простые и понятные модели, описывающие систему с разных сторон, которые можно показать заказчику и обсудить с ним, т. е. служит средством коммуникации в команде.

Проектирование

- *UML* позволяет строить модели программных систем. *По* этим моделям потом может производиться генерация каркасного кода проектируемых приложений.
- Возможен процесс «реверс-инжиниринга» - т. е. создание *UML*-модели из существующего кода приложения.
- Пока качество получающегося кода или моделей при реверс-инжиниринге весьма далеко от идеала, но технологии и инструменты постоянно совершенствуются, так что можно надеяться, что когда-нибудь мы сможем создавать приложения визуально, не прибегая к языку программирования, а пользуясь лишь *UML*...

Документирование

- *UML*-модели сами по себе уже являются документами (весьма понятными, даже для неспециалиста).
- Любой элемент на любой диаграмме может быть снабжен ноутсом - текстовым комментарием.
- Построение набора диаграмм уже является процессом документирования будущей системы.
- Большинство инструментов *UML*-проектирования умеют извлекать текстовую информацию из моделей и генерировать относительно удобочитаемые тексты.

Для чего *UML* использовать нельзя

- *UML* не является языком программирования, хотя существуют средства выполнения *UML*-моделей как интерпретируемого кода (Unimod, FLORA и др.)
- *UML* - средство не программирования, а моделирования, т. е. создания не *программ*, а *моделей* любого уровня абстракции для систем из любой *предметной области*.
- *UML* не является спецификацией какого бы то ни было инструмента моделирования
- *UML* не является моделью какого-либо процесса разработки
- *UML* можно использовать независимо от того, какую методологию разработки *ПО* используют

Структура определения языка

- Авторы использовали так называемое четырехуровневое мета-*моделирование*:
  - Первый уровень - это сами данные.
  - Второй - это их модель, т. е., например, описание их в программе.
  - Третий - *метамодель*, т. е. описание языка построения модели.
  - Четвертый - мета-*метамодель*, т. е. описание языка, на котором описана *метамодель*.

Терминология и нотация

- UML - язык графический и модели (а точнее диаграммы) не "записывают", а рисуют. Одна из задач UML - служить средством коммуникации внутри команды и при общении с заказчиком. "В рабочем порядке" диаграммы часто рисуют на бумаге от руки, причем обычно - не слишком аккуратно. Поэтому при выборе элементов нотации основным принципом был отбор значков, которые хорошо смотрелись бы и были бы правильно интерпретированы в любом случае - будь они нарисованы карандашом на салфетке или созданы на компьютере и распечатаны на лазерном принтере.

Вообще же, в *UML* используется четыре вида элементов нотации: фигуры, линии, значки, надписи.

- Фигуры используются "плоские" - прямоугольники, эллипсы, ромбы и т. д. Но есть одно *исключение* - на диаграмме развертывания для обозначения узлов инфраструктуры применяется "трехмерное" изображение параллелепипеда. Это единственное *исключение* из правил. Внутри любой фигуры могут помещаться другие элементы нотации.

- Линии своими концами должны соединяться с фигурами.
  На *UML* диаграммах нет линий, нарисованных "сами по себе" и не соединяющих фигуры. Применяется два типа линий - сплошная и пунктирная. Линии могут пересекаться, и хотя таких случаев следует по возможности избегать, в этом нет ничего страшного.

ПО с соответствующими инструментами рисования существует.

Наиболее заметные программы этого класса:

IBM Rational Rose; Borland Together; Gentleware Poseidon; Microsoft Visio; Telelogic TAU G2.

Выводы

- UML - еще один формальный язык, который необходимо освоить каждому, кто собирается заниматься проектированием архитектур ИС.
- Знание UML не гарантирует построения разумных и понятных моделей, хотя и является для этого необходимым.
- UML предоставляет огромную свободу при рисовании диаграмм и выборе инструмента рисования. Производители инструментов также воспользовались этой свободой, чтобы по своему разумению "украсить" имеющуюся нотацию.
- В диаграммах выделяется лишь существенные для конкретной задачи свойства системы и строится ее модель, отображающая эти свойства.
- С помощью *UML* можно разработать подробную модель создаваемой системы, отображающую не только ее концепцию, но и конкретные особенности реализации. В рамках *UML*-модели все представления о системе фиксируются в виде специальных графических конструкций, получивших название диаграмм.

Виды диаграмм

UML 1.5 определял **двенадцать типов диаграмм**, разделенных на три группы:

- четыре типа диаграмм представляют статическую структуру приложения;
- пять представляют поведенческие аспекты системы;
- три представляют физические аспекты функционирования системы (диаграммы реализации).

Текущая версия *UML* 2.1 внесла не слишком много изменений. Диаграммы слегка изменились внешне (появились фреймы и другие визуальные улучшения), немного усовершенствовалась *нотация*, некоторые диаграммы получили новые наименования.

Для конкретной модели конкретного приложения количество типов диаграмм не является строго фиксированным.

Для простых приложений нет необходимости строить все без исключения диаграммы.

Важно понимать, что перечень диаграмм зависит от специфики разрабатываемого проекта и определяется самим разработчиком.

Рассмотрим такие *виды диаграмм*, как:

- диаграмма прецедентов;
- диаграмма классов;
- диаграмма объектов;
- диаграмма последовательностей;
- диаграмма взаимодействия;
- диаграмма состояний;
- диаграмма активности;
- диаграмма развертывания.

Диаграмма прецедентов (use case diagram)

- Любые (в том числе и программные) системы проектируются с учетом того, что в процессе своей работы они будут использоваться людьми и/или взаимодействовать с другими системами.
- Сущности, с которыми взаимодействует система в процессе своей работы, называются **эктор**ами, причем каждый эктор ожидает, что система будет вести себя строго определенным, предсказуемым образом.
- **Эктор (actor)** - это множество логически связанных ролей, исполняемых при взаимодействии с прецедентами или сущностями (система, подсистема или класс). Эктором может быть человек или другая система, подсистема или класс, которые представляют нечто вне сущности.

Графически эктор изображается либо " *человечком* ", подобным тем, которые мы рисовали в детстве, изображая членов своей семьи, либо *символом класса с соответствующим стереотипом*, как показано на рисунке.

Обе формы представления имеют один и тот же смысл и могут использоваться в диаграммах. "Стереотипированная" форма чаще применяется для представления системных экторов или в случаях, когда эктор имеет свойства и их нужно отобразить

- **Прецедент (use-case)** - описание отдельного аспекта поведения системы с точки зрения пользователя.
- **Прецедент (use case)** - описание множества последовательных событий (включая варианты), выполняемых системой, которые приводят к наблюдаемому эктором результату.
- Прецедент представляет поведение сущности, описывая взаимодействие между экторами и системой. Прецедент не показывает, "как" достигается некоторый результат, а только "что" именно выполняется.
- Прецеденты обозначаются в виде эллипса, внутри которого указано его название. *Прецеденты и экторы соединяются с помощью линий*. Часто на одном из концов линии изображают *стрелку*, причем *направлена она к тому, у кого запрашивают сервис*, другими словами, чьими услугами пользуются.


Поиск в каталоге
студент
Заказ книг
Работа в читальном зале
библиотекарь
Выдача заказов
Включая рекомендации книг по теме и обучение использованию поисковой системы и заполнению бланков заказа
Консультации


плата
за обучение
получение
стипендии
получение
кредита
переход на
бесплатное обучение
студент
финансовый
отдел
перевод с курса
на курс
администратор
составление
расписания курса
проведение
занятий курса
инструктор
записаться
на курс
отказаться
от курса
посетить занятия
курса
закончить
курс
доведение до
ведома студентов
секретарь
<<extend>>
<<extend>>
сообщение
графика оплаты
<<extend>>
учебная
часть
сообщение
расписания
сообщение списка
выбранных курсов
отказаться
от обучения
окончить
университет
написание
заявки на грант
исследователь

- Диаграммы прецедентов относятся к той группе диаграмм, которые представляют динамические или поведенческие аспекты системы. Это отличное средство для достижения взаимопонимания между разработчиками, экспертами и конечными пользователями продукта.
- Такие диаграммы очень просты для понимания и могут восприниматься и обсуждаться людьми, не являющимися специалистами в области разработки ПО.

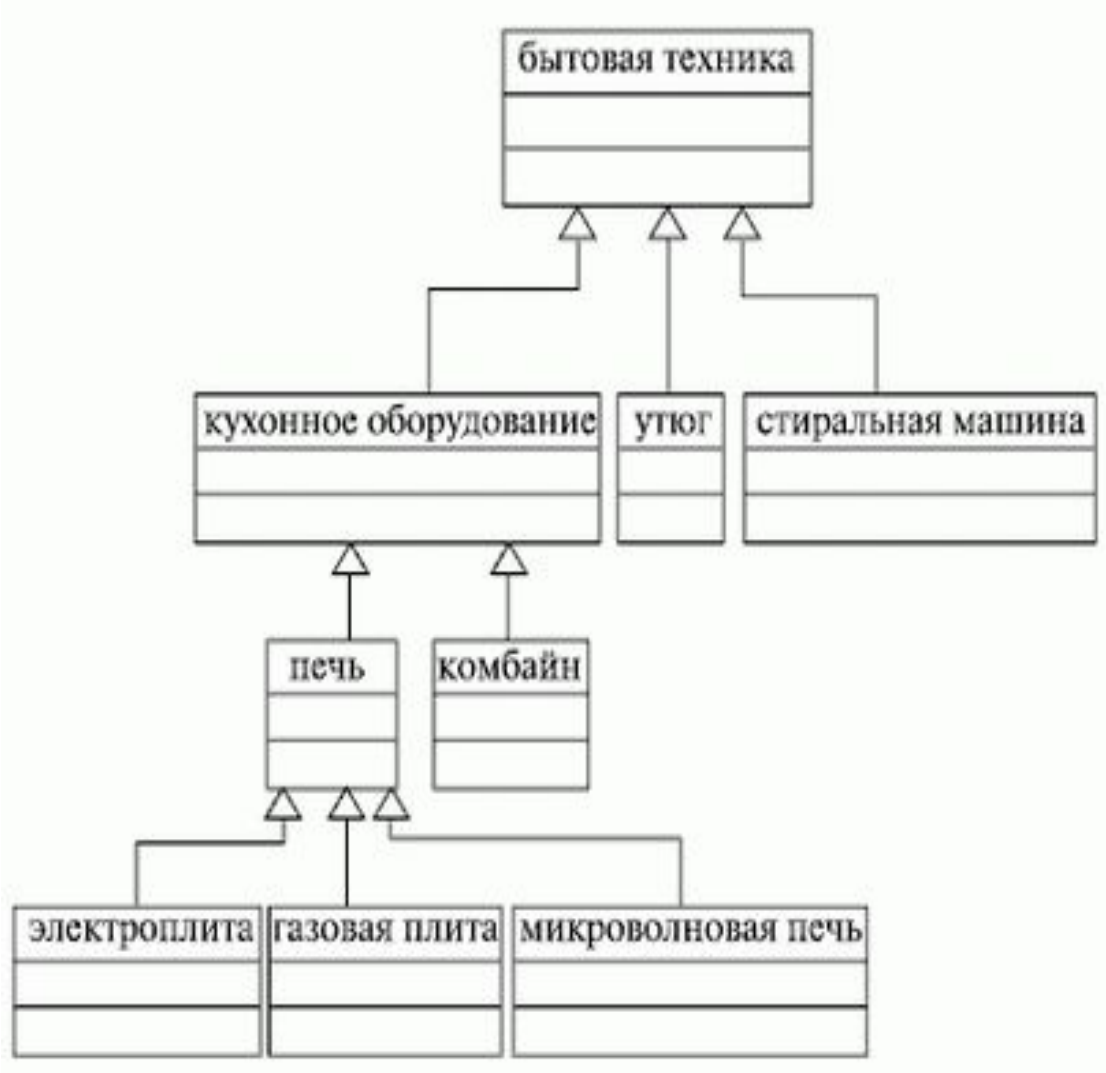
Цели создания диаграмм прецедентов:

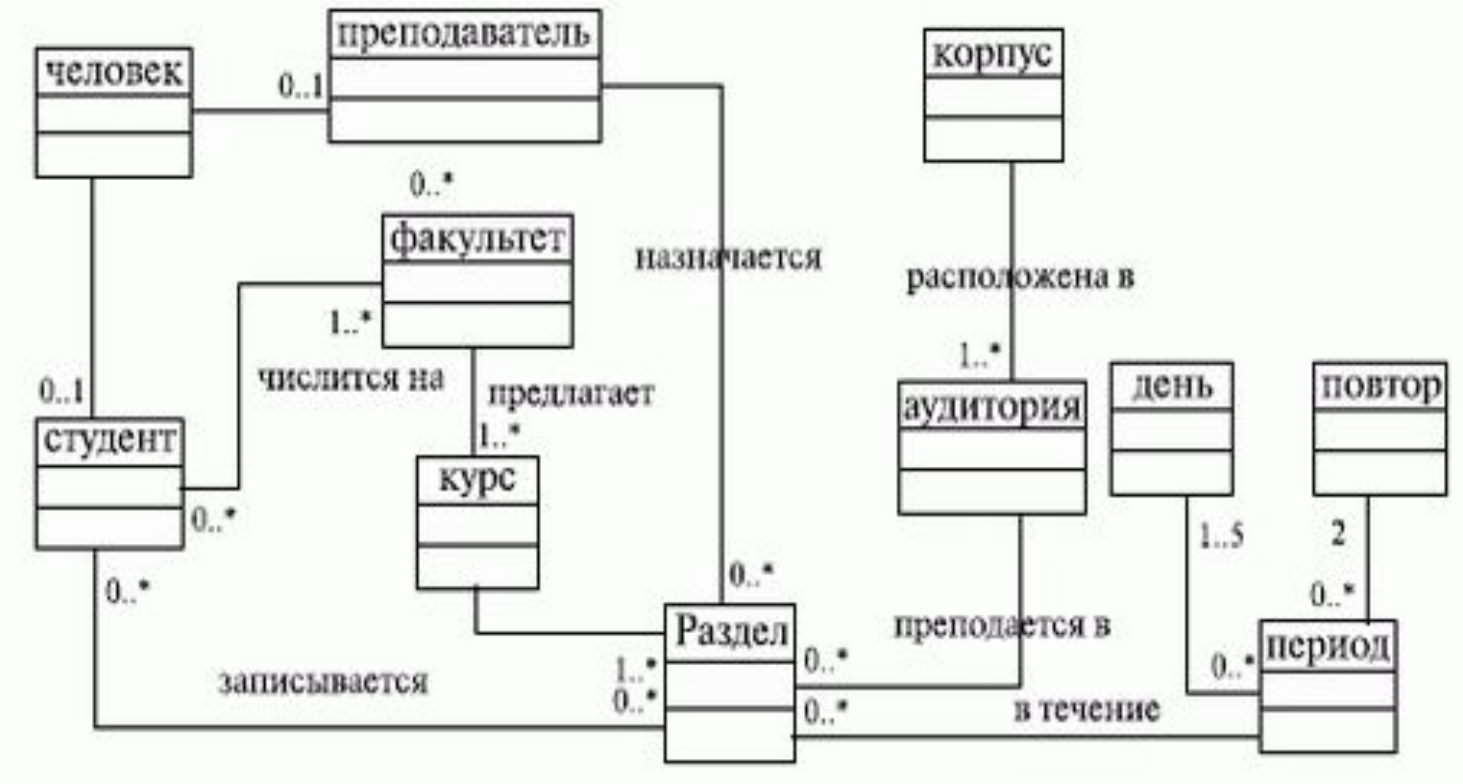
- определение границы и контекста моделируемой предметной области на ранних этапах проектирования;
- формирование общих требований к поведению проектируемой системы;
- разработка концептуальной модели системы для ее последующей детализации;
- подготовка документации для взаимодействия с заказчиками и пользователями системы.

Диаграмма классов (class diagram)

- **Класс (class)** - категория вещей, которые имеют общие атрибуты и операции. Классы - *это строительные блоки любой объектно-ориентированной системы*.
- Они представляют собой описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. При проектировании объектно-ориентированных систем диаграммы классов обязательны.
- Классы используются в процессе анализа предметной области для составления *словаря предметной области* разрабатываемой системы. Это могут быть как абстрактные понятия предметной области, так и классы, на которые опирается разработка и которые описывают программные или аппаратные сущности.

- Диаграмма классов - это *набор статических*, декларативных *элементов модели*.
- Диаграммы классов могут применяться и при прямом проектировании, то есть в процессе разработки новой системы, и при *обратном проектировании* - описании существующих и используемых систем.
- Информация с диаграммы классов напрямую отображается в исходный код приложения - в большинстве существующих инструментов UML-моделирования возможна *кодогенерация* для определенного языка программирования (обычно Java или C++). Таким образом, диаграмма классов - конечный результат проектирования и отправная точка процесса разработки.


бытовая техника
кухонное оборудование
утюг
стиральная машина
печь
комбайн
электроплита
газовая плита
микроволновая печь


человек
преподаватель
0..1
0..*
факультет
1..*
назначается
корпус
расположена в
1..*
0..1
числится на
предлагает
1..*
студент
0..*
курс
аудитория
день
повтор
0..*
0..*
Раздел
1..*
0..*
преподается в
1..5
2
0..*
период
0..*
записывается
0..*
в течение

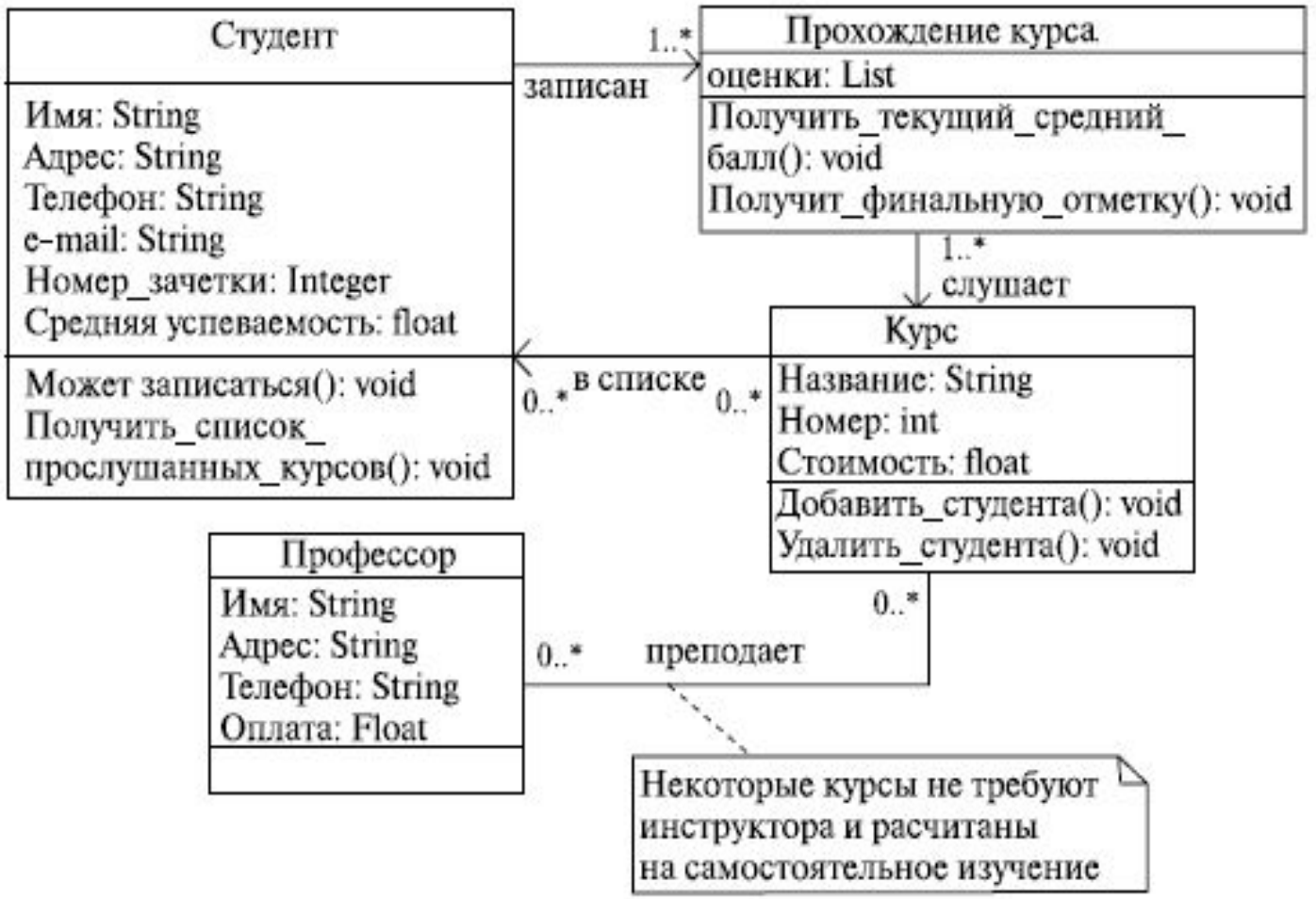

Студент
Имя: String
Адрес: String
Телефон: String
e-mail: String
Номер_зачетки: Integer
Средняя успеваемость: float
Может записаться(): void
Получить_список_
прослушанных_курсов(): void
1..*
записан
Прохождение курса
оценки: List
Получить_текущий_средний_
балл(): void
Получит_финальную_отметку(): void
1..*
слушает
Курс
Название: String
Номер: int
Стоимость: float
Добавить_студента(): void
Удалить_студента(): void
0..*
в списке
0..*
0..*
Профессор
Имя: String
Адрес: String
Телефон: String
Оплата: Float
0..*
преподает
0..*
Некоторые курсы не требуют
инструктора и расчитаны
на самостоятельное изучение

Диаграмма объектов (object diagram)

Объект (object) - экземпляр класса.

Объект (object) - конкретная материализация абстракции; сущность с хорошо определенными границами, в которой инкапсулированы состояние и поведение.

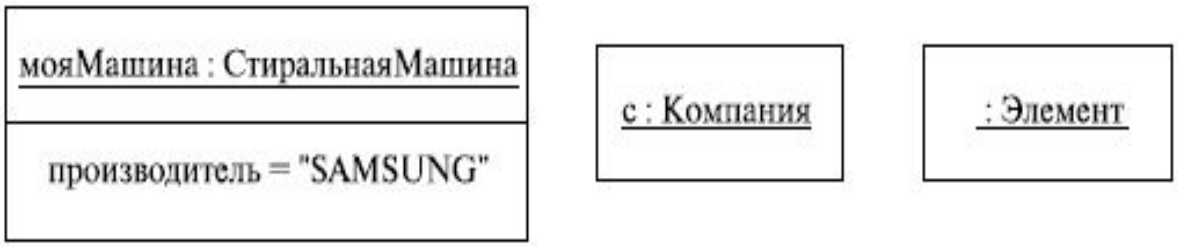
Объект уникально идентифицируется значениями атрибутов, определяющими его состояние в данный момент времени.

Например: объектом класса "Микроволновая печь" может быть и простейший прибор фирмы "*Saturn*" небольшой емкости и с механическим управлением, и навороченный агрегат с грилем, сенсорным управлением и системой трехмерного распределения энергии от Samsung или LG.

Пример:

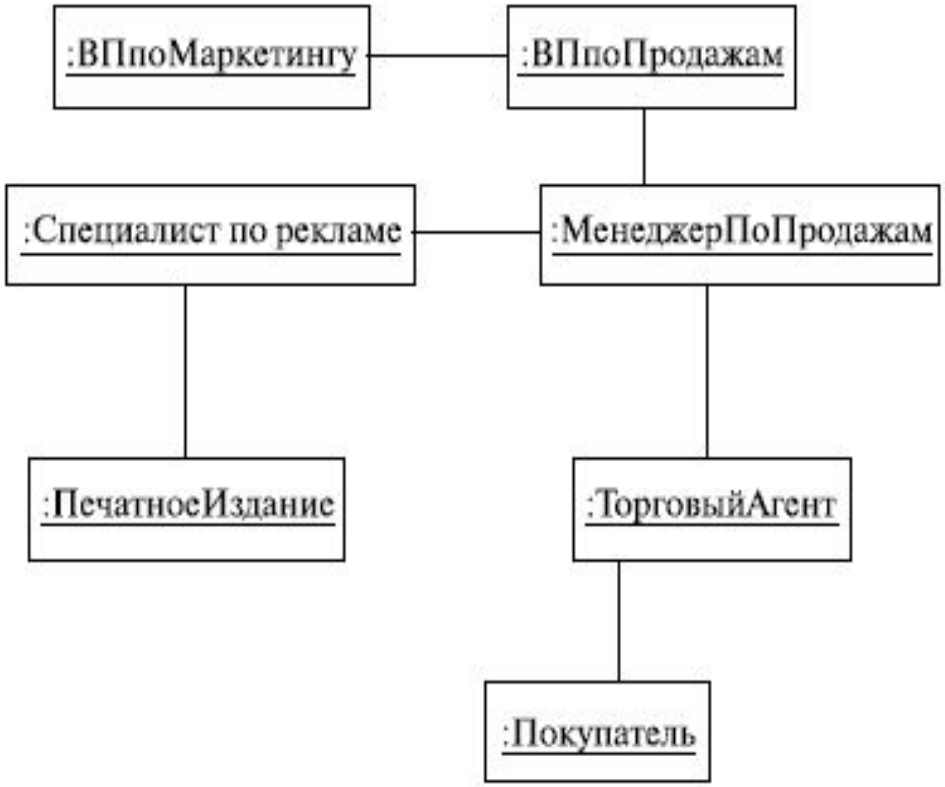
все мы являемся объектами класса "человек" и различимы между собой по таким значениям атрибутов, как имя, цвет волос, глаз, рост, вес, возраст и т. д. (в зависимости от того, какую задачу мы рассматриваем и какие свойства человека для нас в ней важны).

- Объект, *как и класс, обозначается прямоугольником, но его имя подчеркивается*. Под словом имя понимается название объекта и наименование его класса, разделенные двоеточием.
- Для указания значений атрибутов объекта в его обозначении может быть предусмотрена специальная секция. Объект может быть анонимным: это нужно в том случае, если в данный момент не важно, какой именно объект данного класса принимает участие во взаимодействии.



Диаграммы объектов показывают множество объектов - экземпляров классов (изображенных на диаграмме классов) и отношений между ними в некоторый момент времени. То есть *диаграмма объектов - это своего рода снимок состояния системы в определенный момент времени*, показывающий множество объектов, их состояния и отношения между ними в данный момент.

- *Диаграммы объектов* представляют *статический вид системы с точки зрения проектирования и процессов*, являясь основой для сценариев, описываемых диаграммами взаимодействия.
- *Диаграмма объектов используется для пояснения и детализации диаграмм взаимодействия*, например, диаграмм последовательностей.



Некоторая фирма продвигает новый товар или услугу. В этом процессе участвуют различные сотрудники, некое печатное издание и покупатель. На диаграмме отлично видно, кто с кем взаимодействует.
На этой диаграмме все объекты анонимные.

Показана взаимосвязь объектов - организационных единиц в некоторой компании

Диаграмма последовательностей (sequence diagram)

- **Диаграмма последовательностей** отображает взаимодействие объектов в динамике.
- В UML взаимодействие объектов понимается как обмен информацией между ними. При этом информация принимает вид сообщений.

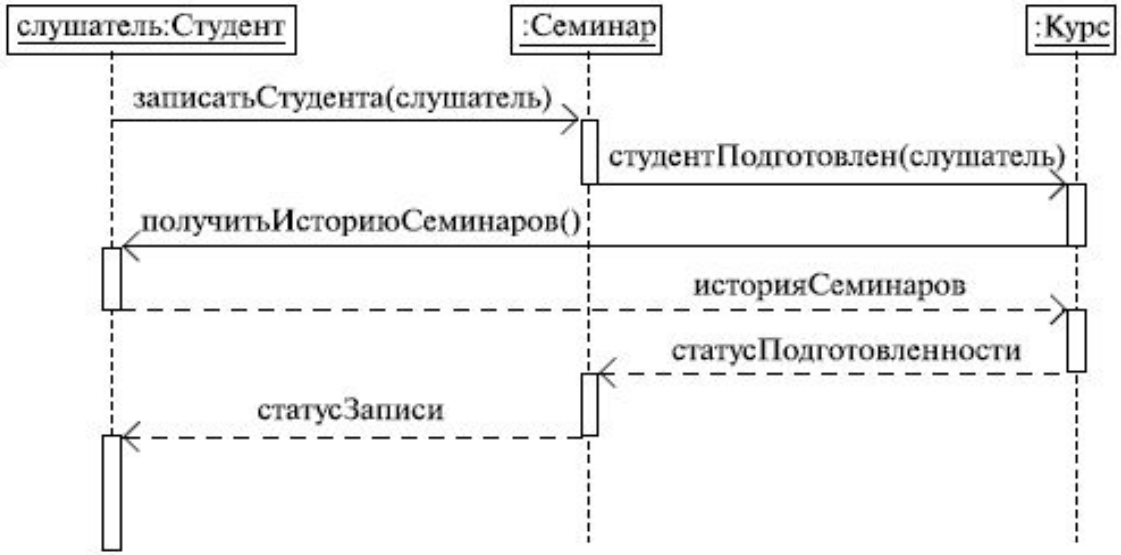
Сообщение несет какую-то информацию и некоторым образом также влияет на получателя.

UML полностью соответствует основным принципам ООП, в соответствии с которыми информационное взаимодействие между объектами сводится к отправке и приему сообщений.

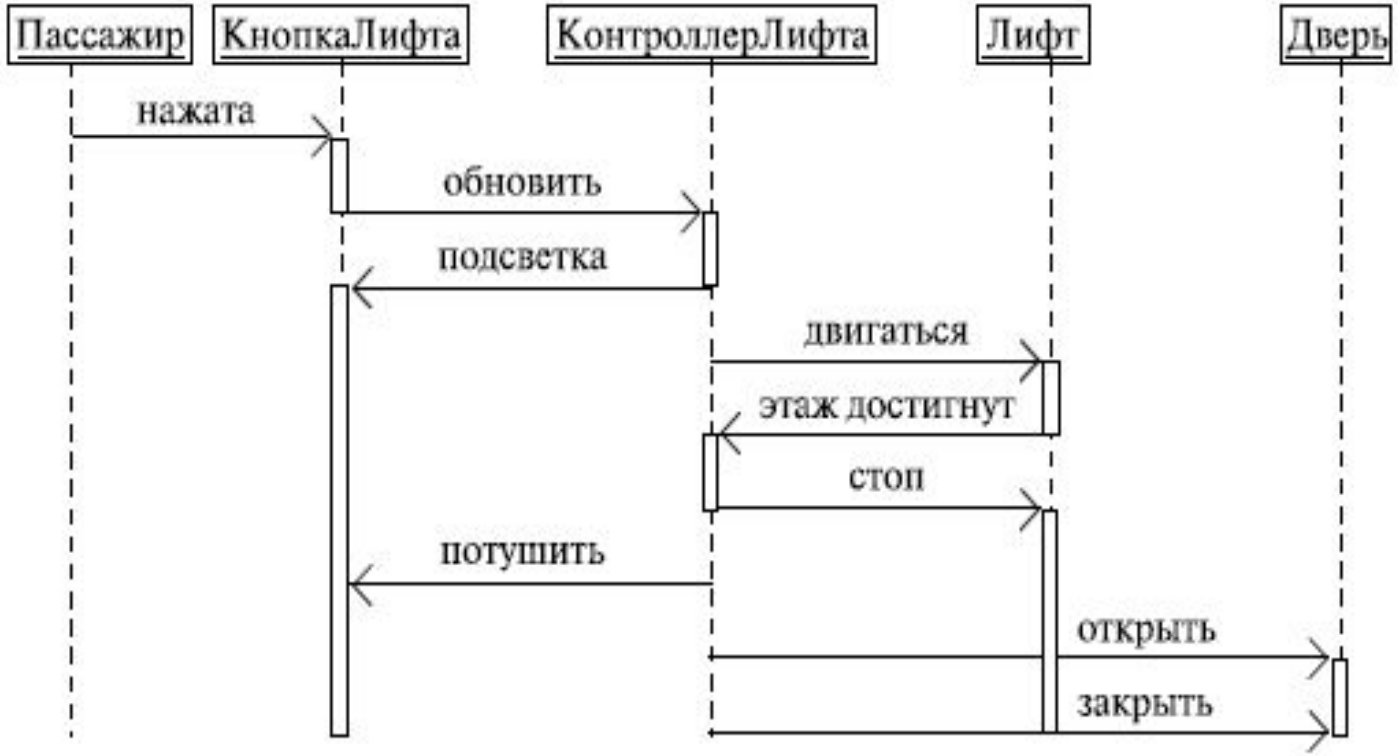
- Диаграмма последовательностей относится к диаграммам взаимодействия UML, описывающим поведенческие аспекты системы, но рассматривает взаимодействие объектов во времени. Она отображает временные особенности передачи и приема сообщений объектами.
- Нечто подобное делает и диаграмма прецедентов.

И диаграммы последовательностей можно использовать для уточнения диаграмм прецедентов, более детального описания логики сценариев использования. Это отличное средство документирования проекта с точки зрения сценариев использования! Диаграммы последовательностей обычно содержат объекты, которые взаимодействуют в рамках сценария, сообщения, которыми они обмениваются, и возвращаемые результаты, связанные с сообщениями.

- Объекты обозначаются прямоугольниками с подчеркнутыми именами (чтобы отличить их от классов), сообщения (вызовы методов) - линиями со стрелками, возвращаемые результаты - пунктирными линиями со стрелками.
- Прямоугольники на вертикальных линиях под каждым из объектов показывают "время жизни" (фокус) объектов.



Студент хочет записаться на некий семинар, предлагаемый в рамках некоторого учебного курса. С этой целью проводится проверка подготовленности студента, для чего запрашивается список (история) семинаров курса, уже пройденных студентом (перейти к следующему семинару можно, лишь проработав материал предыдущих). После получения истории семинаров объект класса "Слушатель" получает статус подготовленности, на основе которой студенту сообщается результат (статус) его попытки записи на семинар.

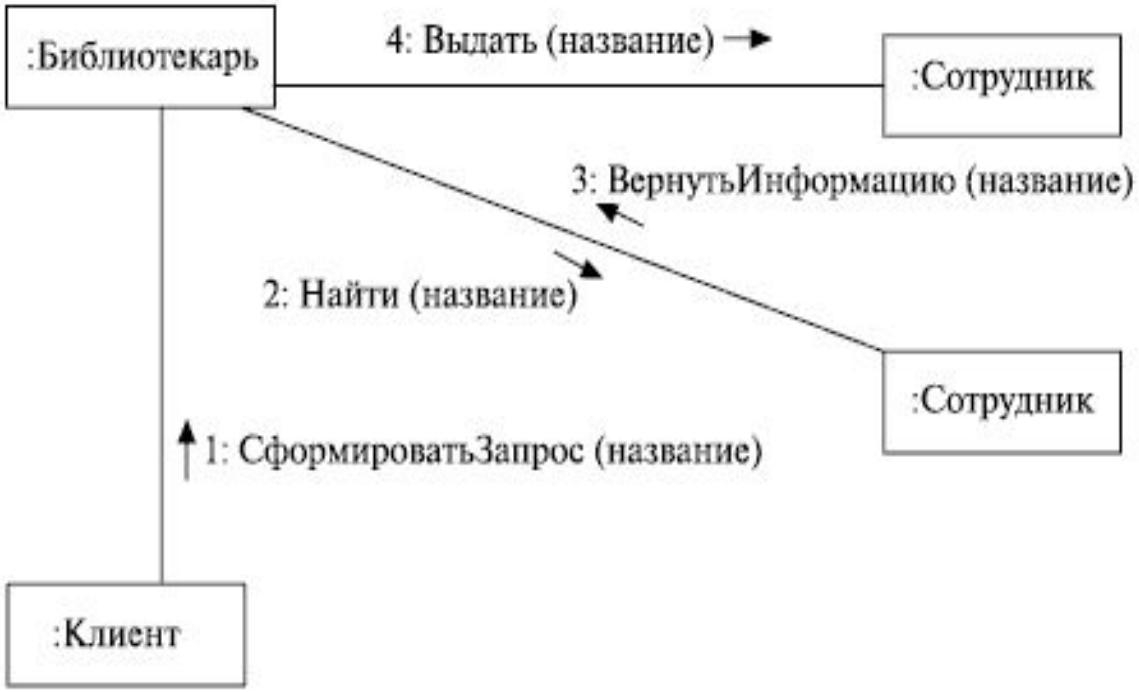


Это работа обычного домового лифта

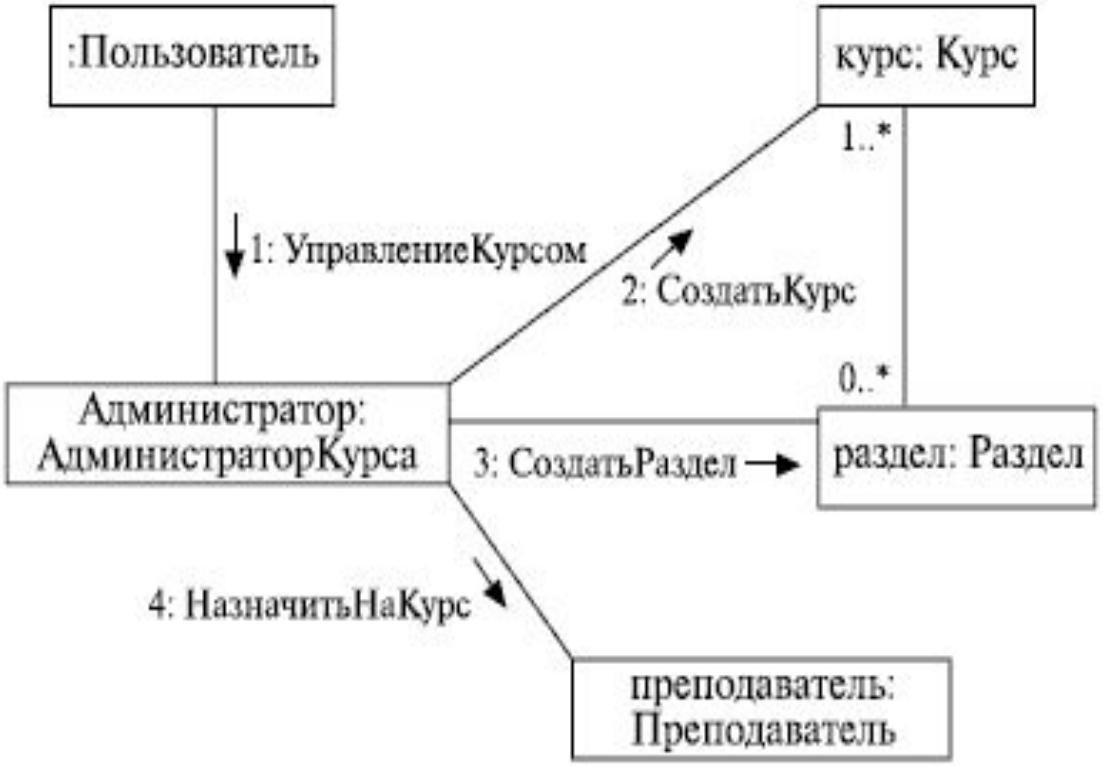
Диаграмма взаимодействия - коммуникации (collaboration diagram)

- Диаграммы последовательностей - это средство документирования поведения системы, детализации логики сценариев использования.
- Но есть еще один способ - использовать диаграммы взаимодействия.
- Диаграмма взаимодействия *показывает поток сообщений между объектами системы и основные ассоциации между ними* и является альтернативой диаграммы последовательностей.
- Можно сказать, что *диаграмма объектов* делает то же самое. *Диаграмма объектов показывает статику*, некий снимок системы, связи между объектами в данный момент времени, диаграмма же взаимодействия, как и диаграмма последовательностей, показывает взаимодействие объектов во времени, т. е. в динамике.

- Использование диаграммы последовательностей или диаграммы взаимодействия - личный выбор каждого проектировщика и зависит от индивидуального стиля проектирования.
- Объекты обозначаются прямоугольниками с подчеркнутыми именами, ассоциации между объектами указываются в виде соединяющих их линий, над ними может быть изображена стрелка с указанием названия сообщения и его порядкового номера.
- Необходимость номера сообщения объясняется очень просто - в отличие от диаграммы последовательностей, *время на диаграмме взаимодействия не показывается в виде отдельного измерения*. Поэтому последовательность передачи сообщений можно указать только с помощью их нумерации.



Эта диаграмма описывает работу персонала библиотеки по обслуживанию клиентов: библиотекарь получает заказ от клиента, поручает сотруднику найти информацию по нужной клиенту книге, а после получения данных поручает еще одному сотруднику выдать книгу клиенту.



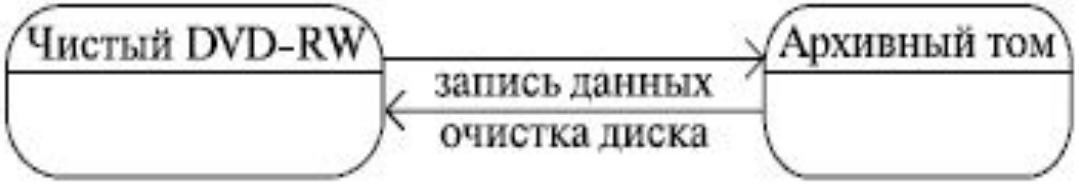
Эта диаграмма описывает процесс управления учебными курсами (очевидно, путем создания их из готовых модулей) для некоего учебного центра.

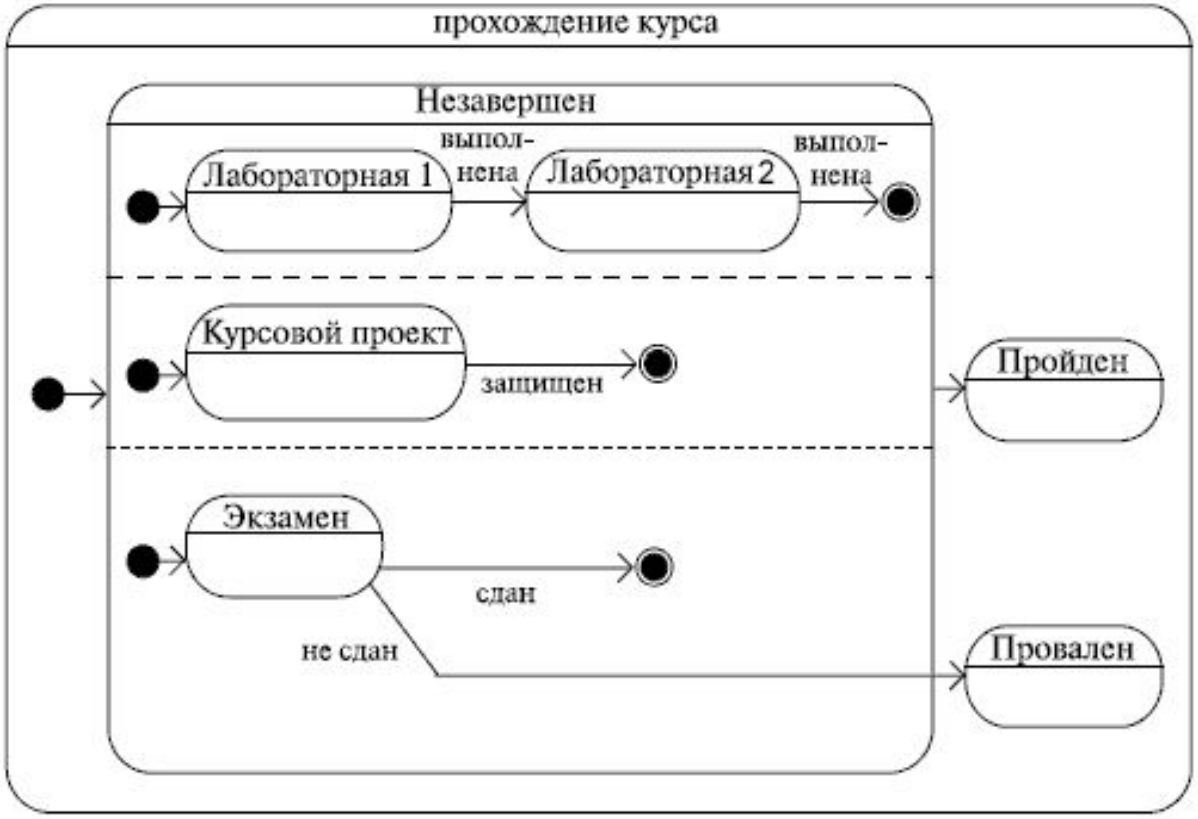
Диаграмма состояний (statechart diagram)

- Объекты характеризуются поведением и состоянием, в котором находятся. Например, человек может быть новорожденным, младенцем, ребенком, подростком или взрослым.
- **Диаграммы состояний** применяются для того, чтобы объяснить, каким образом работают сложные объекты.
- **Состояние (state)** - ситуация в жизненном цикле объекта, во время которой он удовлетворяет некоторому условию, выполняет определенную деятельность или ожидает какого-то события. Состояние объекта определяется значениями некоторых его атрибутов и присутствием или отсутствием связей с другими объектами.

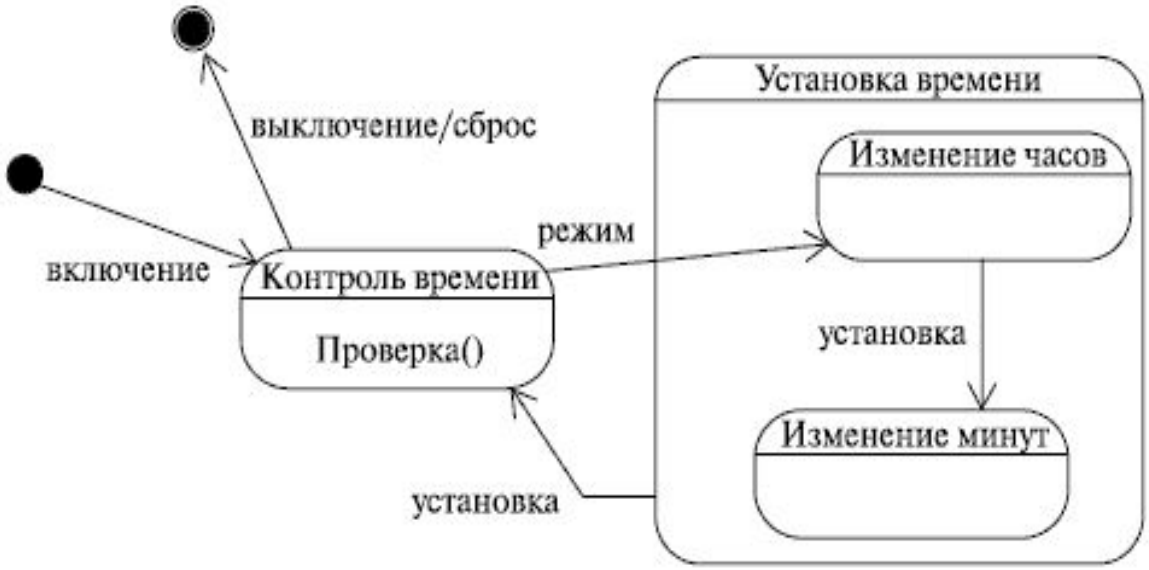
- Диаграмма состояний показывает, как объект переходит из одного состояния в другое.
- Диаграммы состояний служат для моделирования динамических аспектов системы (как и диаграммы последовательностей, кооперации, прецедентов и, как мы увидим далее, диаграммы деятельности).
- Диаграмма состояний полезна при *моделировании жизненного цикла* объекта.
- От других диаграмм диаграмма состояний отличается тем, что описывает процесс изменения состояний только одного экземпляра определенного класса - одного объекта, причем объекта *реактивного*, то есть объекта, поведение которого характеризуется его реакцией на внешние события. Понятие жизненного цикла применимо как раз к реактивным объектам, настоящее состояние (и поведение) которых обусловлено их прошлым состоянием. Но диаграммы состояний важны не только для описания динамики отдельного объекта. Они могут использоваться для *конструирования исполняемых систем* путем прямого и *обратного проектирования*.

- Скругленные прямоугольники представляют состояния, через которые проходит объект в течение своего жизненного цикла. Стрелками показываются переходы между состояниями, которые вызваны выполнением методов описываемого диаграммой объекта.
- Существует также *два вида псевдосостояний: начальное*, в котором находится объект сразу после его создания (обозначается сплошным кружком), и *конечное*, которое объект не может покинуть, если перешел в него (обозначается кружком, обведенным окружностью).





Представлено *составное состояние*, включающее другие состояния, одно из которых содержит также параллельные *подсостояния*. Это диаграмма прохождения академического курса студентом. Для того чтобы пройти курс, студент должен выполнить лабораторные работы, защитить курсовой проект и сдать экзамен.



Это работа устройства таймера. Такой прибор может применяться в составе различных реле, например, для отключения телевизора по истечении указанного промежутка времени. Основное его назначение - контроль времени (проверка, не истек ли указанный промежуток), но у него есть еще один режим работы - установка. По истечении указанного промежутка времени или при "сбросе" устройство отключается.

Диаграмма активности (деятельности, activity diagram)

- Блок-схемы помогают наглядно изобразить алгоритм решения некоторой задачи.
- Моделируя поведение проектируемой системы, часто недостаточно изобразить процесс смены ее состояний, а нужно также раскрыть детали алгоритмической реализации операций, выполняемых системой.
- Для этой цели традиционно использовались блок-схемы или структурные схемы алгоритмов. В UML для этого существуют **диаграммы деятельности**, являющиеся частным случаем диаграмм состояний. Диаграммы деятельности удобно применять для визуализации алгоритмов, по которым работают операции классов.

Это оформление заказа в интернет-магазине

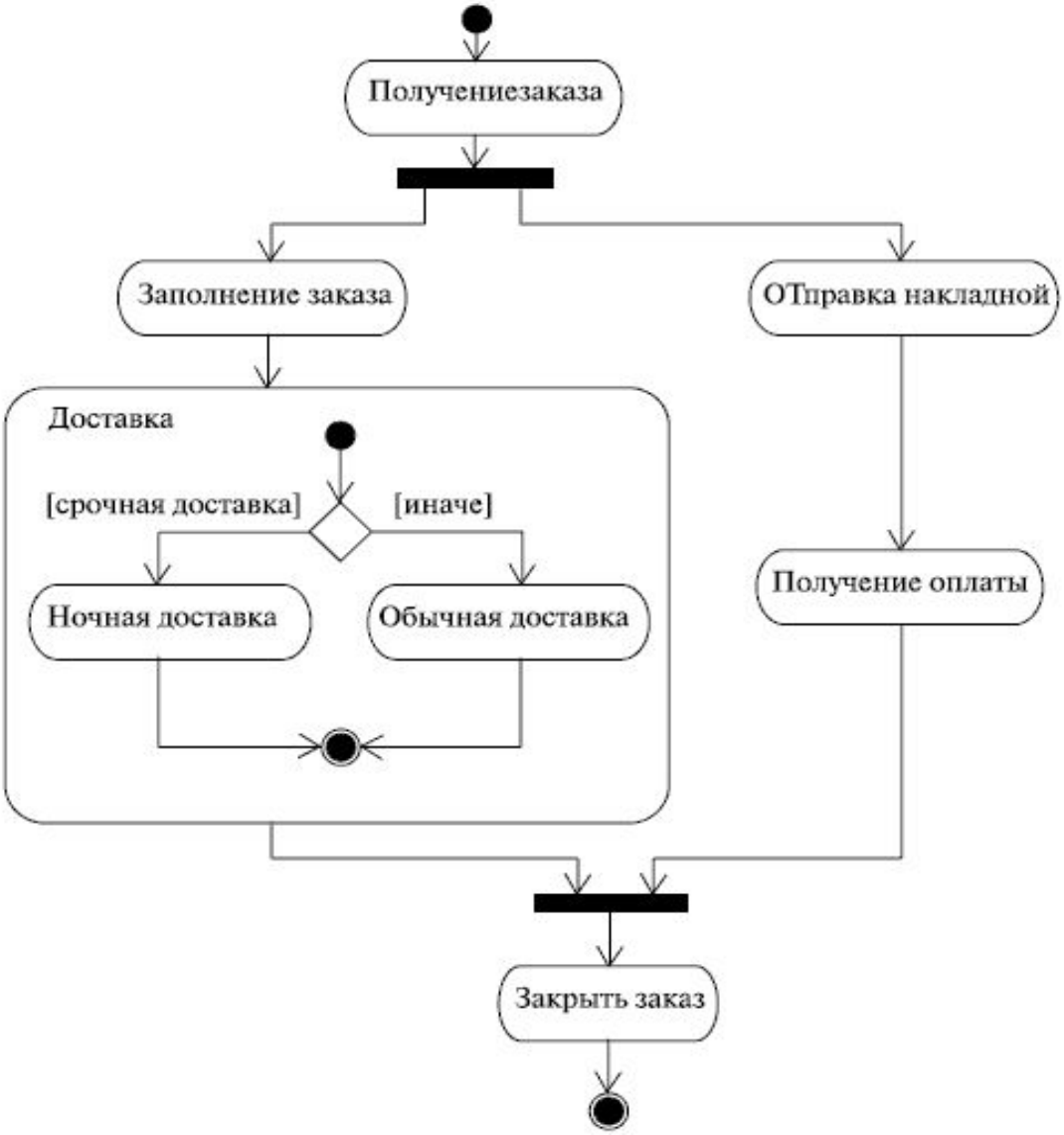

Получениезаказа
Заполнение заказа
ОТправка накладной
Доставка
[срочная доставка]
[иначе]
Ночная доставка
Обычная доставка
Получение оплаты
Закрыть заказ

Диаграмма развертывания (deployment diagram)

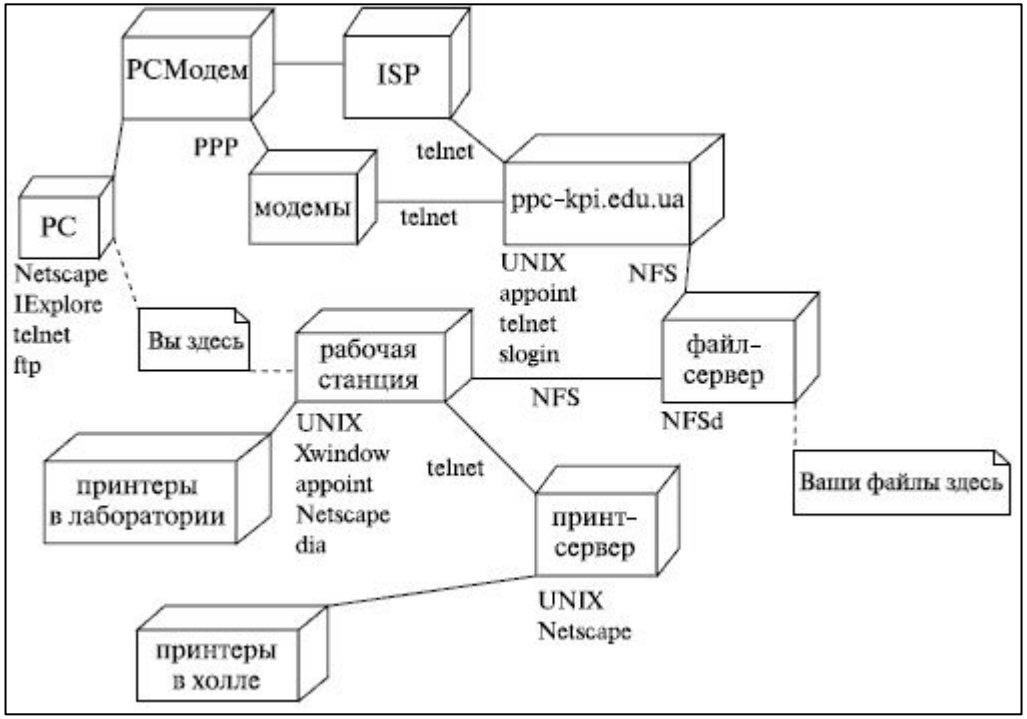
Когда мы пишем программу, мы пишем ее для того, чтобы запускать на компьютере, который имеет некоторую аппаратную конфигурацию и работает под управлением некоторой операционной системы.

Корпоративные приложения часто требуют для своей работы некоторой ИТ-инфраструктуры, хранят информацию в базах данных, расположенных где-то на серверах компании, вызывают веб-сервисы, используют общие ресурсы и т. д.

В таких случаях неплохо бы иметь графическое представление инфраструктуры, на которую будет развернуто приложение. Для этого-то и нужны **диаграммы развертывания**, которые иногда называют диаграммами размещения.

Такие диаграммы есть смысл строить только для аппаратно-программных систем.

- Диаграмма развертывания – это графическое представление ИТ-инфраструктуры, поэтому ее использование помогает *более рационально распределить компоненты системы по узлам сети*, от чего, как известно, зависит в том числе и производительность системы.
- Такая диаграмма может помочь *решить множество вспомогательных задач*, связанных, например, с обеспечением безопасности.
- *Диаграмма развертывания* показывает топологию системы и распределение компонентов системы по ее узлам, а также соединения - маршруты передачи информации между аппаратными узлами. Это единственная диаграмма, на которой применяются "трехмерные" обозначения: узлы системы обозначаются кубиками. Все остальные обозначения в UML - плоские фигуры.



Это инфраструктура некоего учебного заведения, включающая шлюз, файл-сервер, принт-сервер, принтеры в лабораториях и холле и т. д.

Пользователь (вероятно, студент или преподаватель) может получить доступ к этим ресурсам либо со своей домашней машины, либо с рабочих станций, находящихся в лабораториях вуза. Подписи под линиями, показывают линии передачи информации. Например, видно, что рабочая станция получает доступ к файлам, хранящимся на файл-сервере, посредством NFS. Рядом с обозначением узла перечисляется программное обеспечение, установленное на данном узле, как это сделано, например, для рабочей станции.

Последовательность построения *UML* диаграмм

В ходе проектирования архитектуры ИС необходимо ответить для себя на такие вопросы:

- ✔ Какие именно виды диаграмм лучше всего отражают архитектуру системы и возможные технические риски, связанные с проектом?
- ✔ Какие из диаграмм удобнее всего превратить в инструмент контроля над процессом разработки системы?

Диаграммы можно и нужно строить в некоторой логической последовательности.

В *UML*-проектировании, как и при создании любых других моделей, важно уметь **абстрагироваться** от несущественных свойств системы.

Начинать необходимо с *выявления и анализа прецедентов*. Они помогут отработать навыки выявления четких абстракций.

Построение *модели предметной области лучше делать в виде диаграммы классов*! Это хороший способ понять, как визуализировать *множества* взаимосвязанных абстракций. Таким же образом стройте модели статической части задач.

Динамическую часть задачи нужно начинать моделировать с помощью простых *диаграмм последовательностей и кооперации*.

Хорошо начать с *модели взаимодействия* пользователя с системой - так можно легко выделить наиболее важные *прецеденты*.

- *UML* применяется для того, чтобы прояснить неявные детали реализации существующей системы или использованные в ней "хитрые *механизмы* программирования".
- Построение *UML*-модели необходимо осуществлять прежде чем начинать новый проект.
- Средства *UML* подходят для моделирования компонентов, параллельности, распределенности, паттернов проектирования.

Можно выделить такую последовательность построения базовых UML диаграмм:

- диаграмма прецедентов,
- диаграмма классов,
- диаграмма объектов,
- диаграмма последовательностей,
- диаграмма взаимодействия,
- диаграмма состояний,
- диаграмма активности,
- диаграмма развертывания.

Представления архитектуры ИС с точки зрения моделирования

Ситуация в разработке

Визуализация, специфицирование, конструирование и документирование программных систем требуют их представления с *различных точек зрения*.

Разные заинтересованные лица – конечные пользователи, аналитики, разработчики, системные интеграторы, тестировщики, технические писатели и менеджеры проекта – имеют различное представление о проекте, и каждый из них видит информационную систему по-разному в разные моменты жизненного цикла проекта.

Системная архитектура является продуктом, который может быть использован для управления всеми этими разнообразными точками зрения и тем самым управляет итеративным и пошаговым процессом разработки информационной системы на протяжении всего ее жизненного цикла.

Архитектура ИС – это набор существенных решений относительно:

- организации программной системы

- выбора структурных элементов, составляющих систему, и их интерфейсов

- поведения этих элементов, определенного в их кооперациях

- объединения этих структурных и поведенческих элементов в более крупные подсистемы

- архитектурного стиля, определяющего организацию системы: статические и динамические элементы и их интерфейсы, кооперацию и композицию

Архитектура информационной системы с точки зрения UML

Архитектура ИС может быть наилучшим описана с помощью пяти взаимосвязанных **представлений**.

Каждое **представление** – проекция организации и структуры системы, сосредоточенная на определенном ее аспекте.

Представление вариантов использования (Use Case View) системы охватывает варианты использования, описывающие поведение системы с точки зрения конечных пользователей, аналитиков и тестировщиков. Представление специфицирует не истинную организацию программной системы, а лишь некие движущие силы, формирующие системную архитектуру.

- **статические аспекты представления: диаграммы вариантов использования - прецедентов,**
- **динамические аспекты: диаграммы взаимодействий, состояний и деятельности**

Представление системы с точки зрения проектирования (дизайна, design view) **охватывает классы, интерфейсы и кооперации, формирующие словарь проблемы и ее решение.**

Предназначено для описания словаря предметной области, то есть, в парадигме объектно-ориентированного программирования, классов, а также таких вспомогательных сущностей как, например, интерфейсы или кооперации

Поддерживает функциональные требования к системе, то есть сервис, который она должна

Представление системы с точки зрения проектирования (дизайна, design view) охватывает классы, интерфейсы и кооперации, формирующие словарь проблемы и ее решение.

- Статические аспекты: диаграммы классов и объектов
- Динамические аспекты: диаграммы взаимодействий, состояний и деятельности

Представление взаимодействия (вид с точки зрения процессов, Process View) системы показывает поток управления, проходящий через разные ее части, включая возможные механизмы параллелизма и синхронизации. Это представление касается производительности, масштабируемости и пропускной способности системы

- **Структурные аспекты: диаграммы классов и объектов (для классов/объектов, которые передают сообщения)**
- **Поведенческие аспекты: диаграммы взаимодействия, состояний и деятельности**

Представление реализации (компонентов) (Component view) - описание системы на уровне артефактов (компонентов, файлов и т.д.), используемых для сборки, выпуска, конфигурации программного продукта. Представление относится к управлению конфигурацией версий системы, состоящей из файлов и компонентов, которые могут быть собраны различными способами для формирования работающей системы.

- **Структурные аспекты: диаграммы компонентов, артефактов, объектов**
- **Поведенческие аспекты: диаграммы взаимодействия, состояний и деятельности**

Представление развертывания (deployment view) отражает топологию связей аппаратных средств и размещения на них компонентов. Это представление в основном связано с поставкой, распределением и установкой частей, составляющих физическую систему.

- Структурные аспекты: диаграммы размещения
- Поведенческие аспекты: диаграммы взаимодействия, состояний и деятельности

Процесс моделирования с учетом рассмотренной архитектуры

Процесс моделирования (независимо от назначения модели) является не линейным последовательным, а итеративным и параллельным.

Представление использования - что делает система?

Указать:

- **Действующие лица**
- **Границы системы**
- **Функциональные требования**
- **Ответственность компонентов**

Представление структуры - из чего состоит система?

Выделение структурных элементов – составных частей системы – и описания взаимосвязей между ними. Принципиальным является чисто *статический характер описания*, то есть отсутствие понятия времени в любой форме, в частности, в форме последовательности событий и/или действий. В данном представлении отражается:

- **Предметная область**
- **Архитектура системы**
- **Структура хранения**
- **Детали реализации**

Представление структуры ИС:

Основные диаграммы:

- **диаграммы классов (class diagram)**

Дополнительно:

- **диаграммы компонентов (component diagram)**
- **диаграммы размещения deployment diagram**
- **диаграммы внутренней структуры (composite structure diagram)**
- **диаграммами объектов (object diagram)**

Представление поведения - как работает система ?

Определяющим признаком для отнесения элементов модели к представлению поведения является явное использование понятия времени, в частности, в форме описания последовательности событий / действий, то есть в форме алгоритма.

- **Бизнес-процессы**
- **Пользовательский интерфейс**
- **Алгоритмы обработки**
- **Жизненные циклы**

Представление поведения ИС:

Основные диаграммы:

- **диаграмма автомата (state machine diagram)**
- **диаграмма деятельности (activity diagram)**
- **обзорная диаграммой взаимодействия (interaction overview diagram)**
- **диаграмма коммуникации (communication diagram)**
- **диаграмма последовательности (sequence diagram)**
- **диаграмма синхронизации (timing diagram)**

Контрольные вопросы:

1. Почему нужно строить разные диаграммы при моделировании системы?
2. Какие диаграммы соответствуют статическому представлению о системе?
3. Назовите виды представлений архитектуры ИС. Какие аспекты ИС отражает каждое из них?
4. Представьте, что вы разрабатываете компьютерную программу для игры в шахматы. Какие UML диаграммы были бы полезны в этом случае и почему?
5. Составьте список вопросов потенциальному пользователю (заказчику) такой программы при беседе с ним на стадии проектирования ИС. Объясните, почему вы хотели бы задать именно такие вопросы.