# Проектирование архитектуры информационных систем

Папшев С.В.

2023

# Наш план

- Информационная система – сложная система.
- **Объектно-ориентированный подход к проектированию ИС**
- UML
- Методологии  и моделирование ИС
- ОО-Шаблоны проектирования ИС

# Объектно-ориентированное проектирование

# Проектирование сложных систем

- Проектирование – инженерная задача
- Инженерия = наука + искусство
- Под проектированием обычно понимается некий унифицированный подход, с помощью которого мы ищем пути решения определенной проблемы, обеспечивая выполнение поставленной задачи
- Важность построения модели
- Элементы программного проектирования
  - условные обозначения — язык для описания каждой модели;
  - процесс — правила проектирования модели;
  - инструменты — средства, которые ускоряют процесс создания моделей, и в которых уже воплощены законы функционирования моделей. Инструменты помогают выявлять ошибки в процессе разработки.

# Методы проектирования программных систем

- Метод — это последовательный процесс создания моделей, которые описывают вполне определенными средствами различные стороны разрабатываемой программной системы.
- Методология — это совокупность методов, применяемых в жизненном цикле разработки программного обеспечения и объединенных одним общим философским подходом.
- Методы важны по нескольким причинам
  - упорядочивают процесс создания сложных программных систем, как общие средства доступные для всей группы разработчиков.
  - позволяют менеджерам в процессе разработки оценить степень продвижения и риск
- Основные методы проектирования:
  - метод структурного проектирования сверху вниз;
  - метод потоков данных;
  - объектно-ориентированное проектирование.

# Оптимальные методы работы над программными проектами трех популярных типов

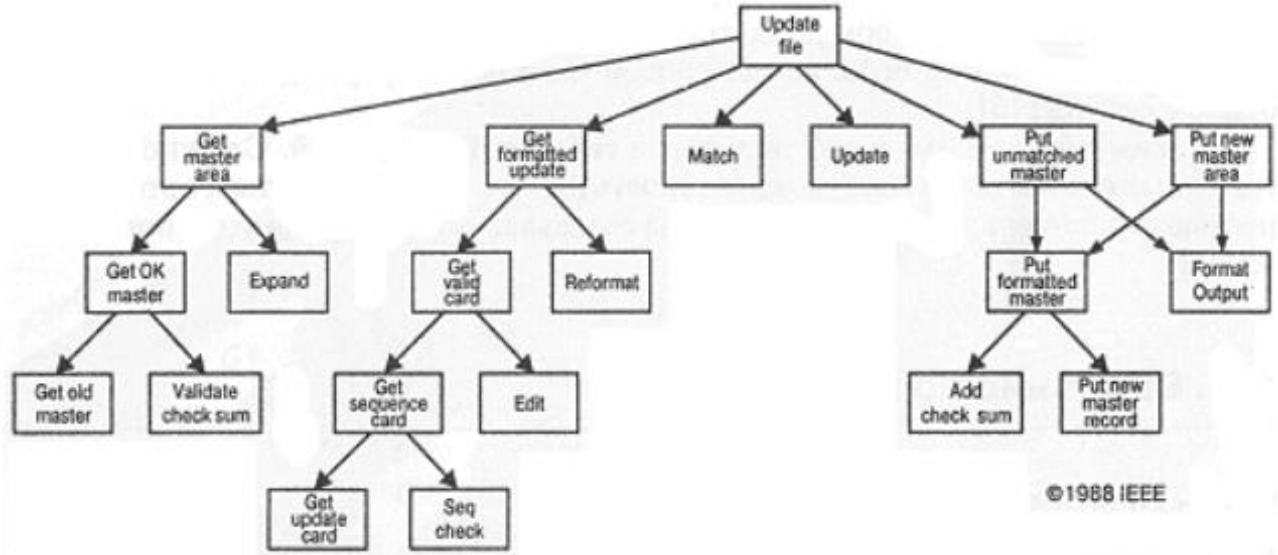
| | **Тип ПО** | | |
|---|---|---|---|
| | **Бизнес-системы** | **Системы целевого назначения** | **Встроенные системы, от которых зависит жизнь людей** |
| **Типичные приложения** | Интернет-сайты. Сайты в интрасетях. Системы управления материально-техническим снабжением. Игры. Системы управления информацией. Системы выплаты заработной платы. | Встроенное ПО. Игры. Интернет-сайты. Пакетное ПО. Программные инструменты. Web-сервисы. | Авиационное ПО. Встроенное ПО. ПО для медицинских устройств. Операционные системы. Пакетное ПО. |
| **Модели жизненного цикла** | Гибкая разработка (экстремальное программирование, методология Scrum, разработка на основе временных окон и т. д.). Эволюционное прототипирование. | Поэтапная поставка. Эволюционная поставка. Спиральная разработка | Поэтапная поставка. Спиральная разработка. Эволюционная поставка |
| **Планирование и управление** | Инкрементное планирование проекта. Планирование тестирования и гарантии качества по мере надобности. Неформальный контроль над изменениями | Базовое заблаговременное планирование. Базовое планирование тестирования. Планирование гарантии качества по мере надобности Формальный контроль над изменениями. | Исчерпывающее заблаговременное планирование. Исчерпывающее планирование тестирования. Исчерпывающее планирование гарантии качества. Тщательный контроль над изменениями. |
| **Выработка требований** | Неформальная спецификация требований по мере надобности | Полуформальная спецификация требований Обзоры требований | Формальная спецификация требований. Формальные инспекции требований |

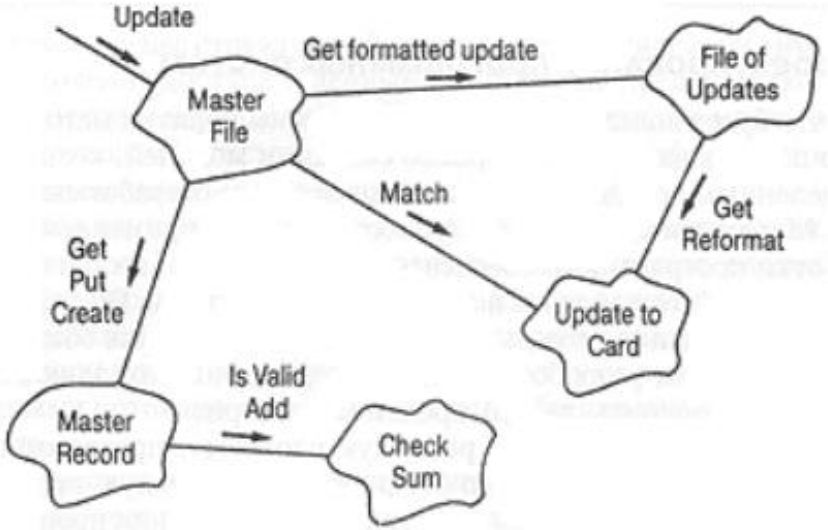
# Оптимальные методы работы над программными проектами трех популярных типов

| | Тип ПО | | |
|---|---|---|---|
| | **Бизнес-системы** | **Системы целевого назначения** | **Встроенные системы, от которых зависит жизнь людей** |
| **Проектирование** | Комбинация проектирования и кодирования. | Проектирование архитектуры Неформальное детальное проектирование Обзоры проекта по мере надобности. | Проектирование архитектуры Формальные инспекции архитектуры. Формальное детальное проектирование. Формальные инспекции детального проекта. |
| **Конструирование** | Парное или индивидуальное программирование. Неформальная процедура регистрации кода или ее отсутствие. | Парное или индивидуальное программирование. Неформальная процедура регистрации кода. Обзоры кода по мере надобности. | Парное или индивидуальное программирование. Формальная процедура регистрации кода. Формальные инспекции кода. |
| **Тестирование и гарантия качества** | Разработчики тестируют собственный код. Предварительная разработка тестов Тестирование отдельной группой проводится в малом объеме или не проводится вообще. | Разработчики тестируют собственный код Предварительная разработка тестов. Отдельная группа тестирования | Разработчики тестируют собственный код. Предварительная разработка тестов. Отдельная группа тестирования. Отдельная группа гарантии качества. |
| **Внедрение приложения** | Неформальная процедура внедрения. | Формальная процедура внедрения. | Формальная процедура внедрения. |

# Внесение порядка в хаос

- Декомпозиция
  - Алгоритмическая декомпозиция
  - Объектно-ориентированная декомпозиция
- Роль абстракции
- Роль иерархии



# Организованная и неорганизованная сложность

- Разного типа иерархии:
  - структурная иерархия типа «быть частью» - структура классов
  - иерархия типа «is-a» - структура объектов
- Структура классов и объектов называется архитектурой системы
- Ограниченность человеческих возможностей при проектировании программных систем


Каноническая форма сложной системы.

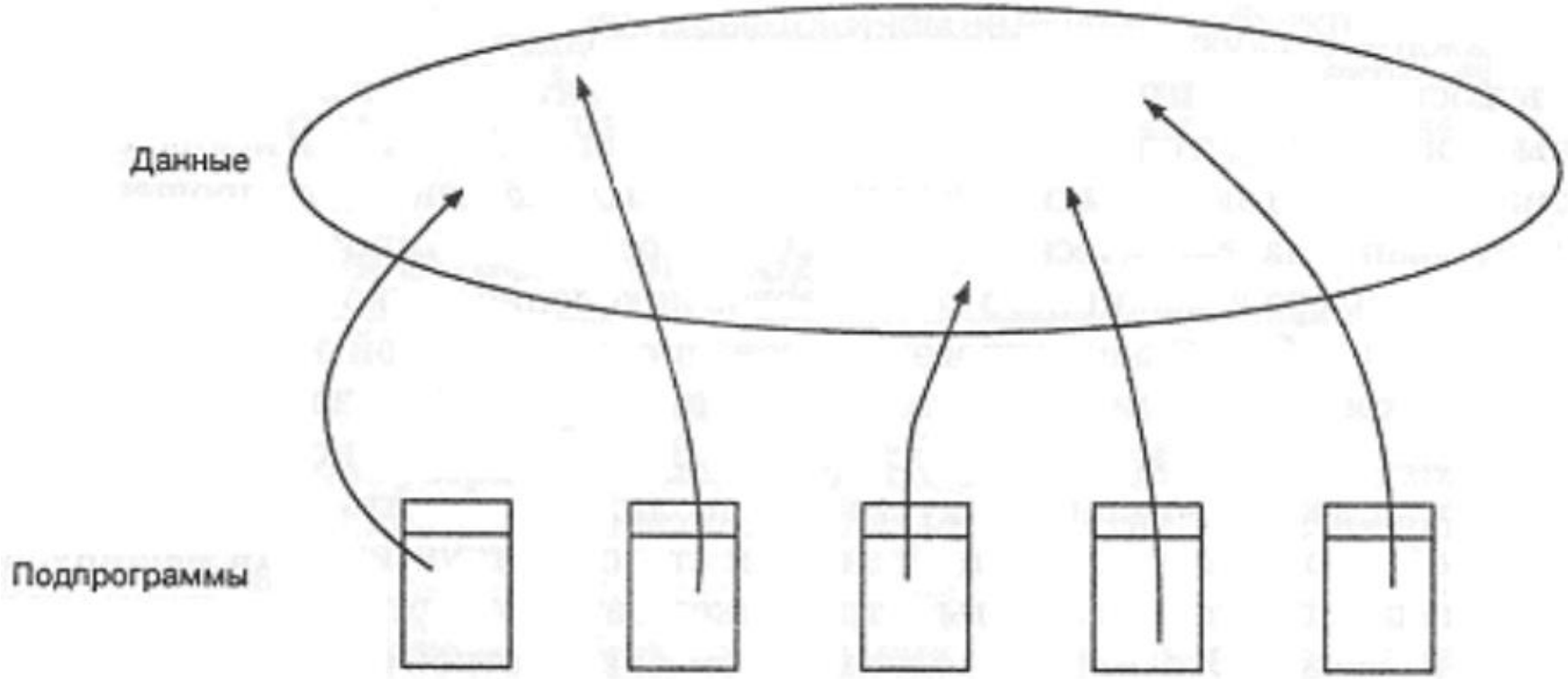
# Тенденции в проектировании

Wegner, P. [J 1981].

- Первое поколение(1954-1958)
  - FORTRAN I Математические формулы
  - ALGOL-58 Математические формулы
  - Flowmatic Математические формулы
  - IPL V Математические формулы
- Второе поколение(1959-1961)
  - FORTRAN II Подпрограммы, раздельная компиляция
  - ALGOL-60 Блочная структура, типы данных
  - COBOL Описание данных, работа с файлами
  - Lisp Обработка списков, указатели, сборка мусора
- Третье поколение(1962-1970)
  - PL/I FORTRAN+ALGOL+COBOL
  - ALGOL-68 Более строгий приемник ALGOL-60
  - Pascal Более простой приемник ALGOL-60
  - Simula Классы, абстрактные данные
- Потерянное поколение (1970-1980)
  - Много языков созданных, но мало выживших
  - Smalltalk (новаторски переработанное наследие Simula),
  - Ada (наследник ALGOL-68 и Pascal с элементами Simula, Alphard и CLU),
  - CLOS (объединивший Lisp, LOOPS и Flavors),
  - C++ (возникший от брака C и Simula) и
  - Eiffel (произошел от Simula и Ada)

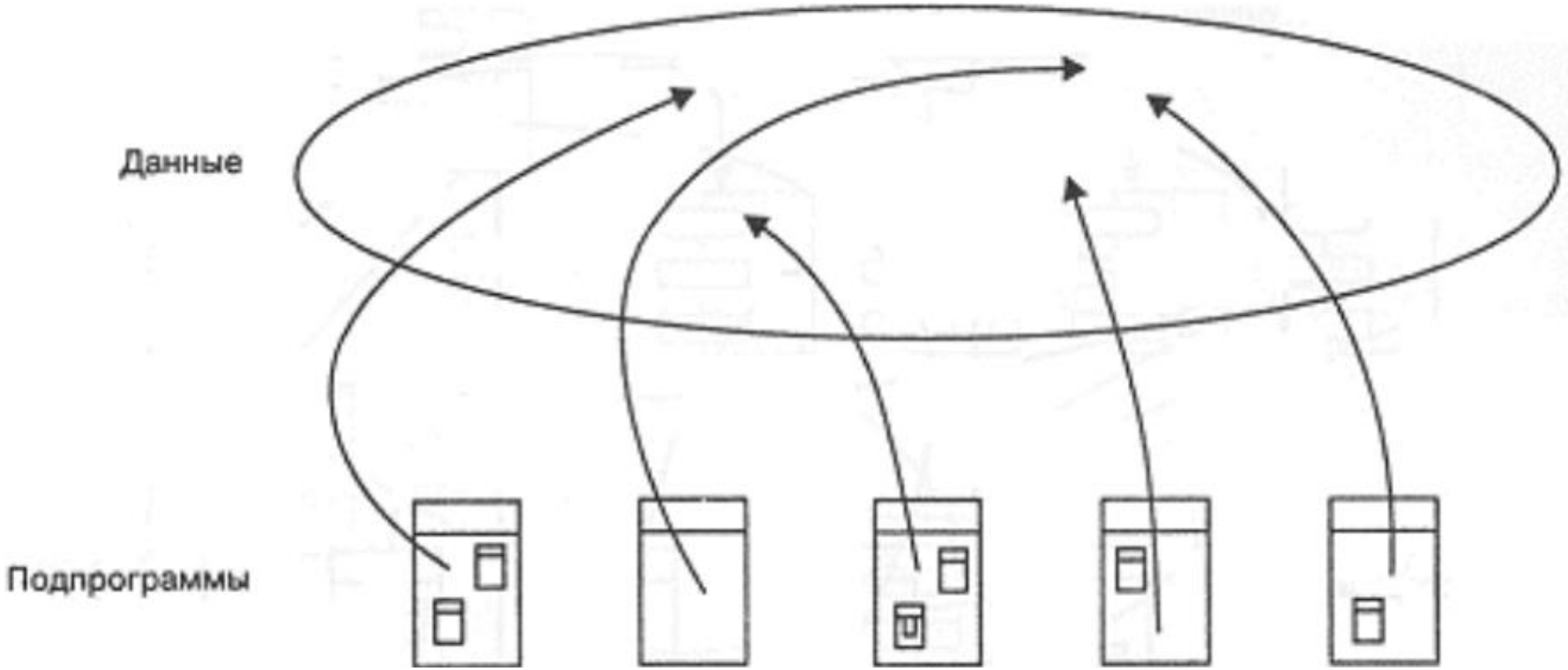
# Топология языков первого и начала второго поколения

- Топология - основные элементы языка программирования и их взаимодействие



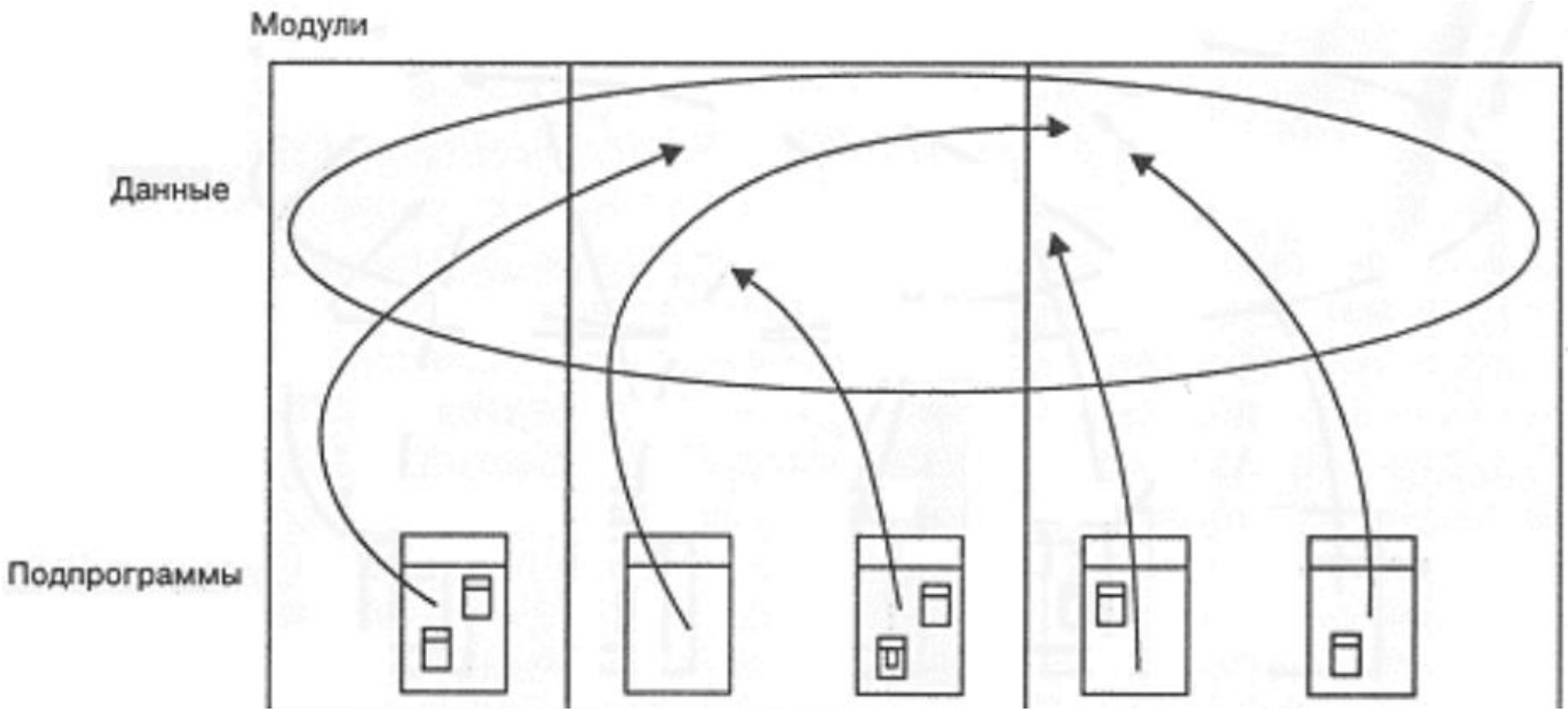
# Топология языков позднего второго и раннего третьего поколения

- Первая программная абстракция, названная процедурной абстракцией
- Заложены основы структурного проектирования и программирования



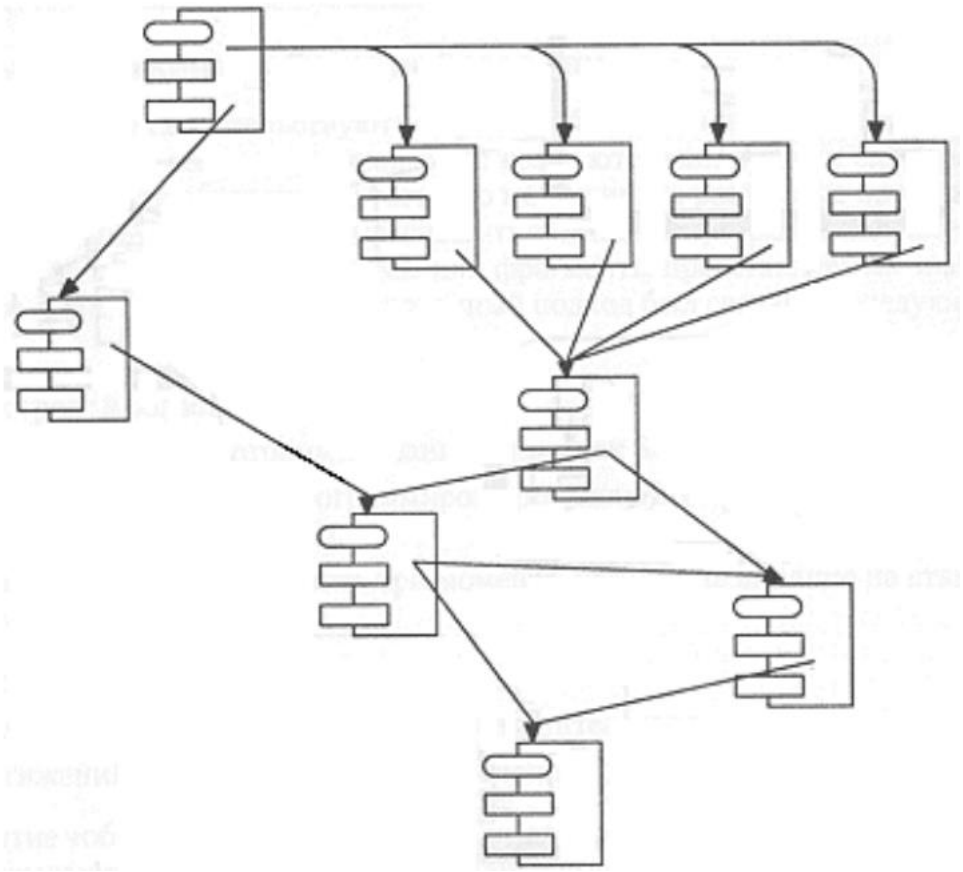
# Топология языков конца третьего поколения

- Модульное программирование



# Топология объектных и объектно-ориентированных языков

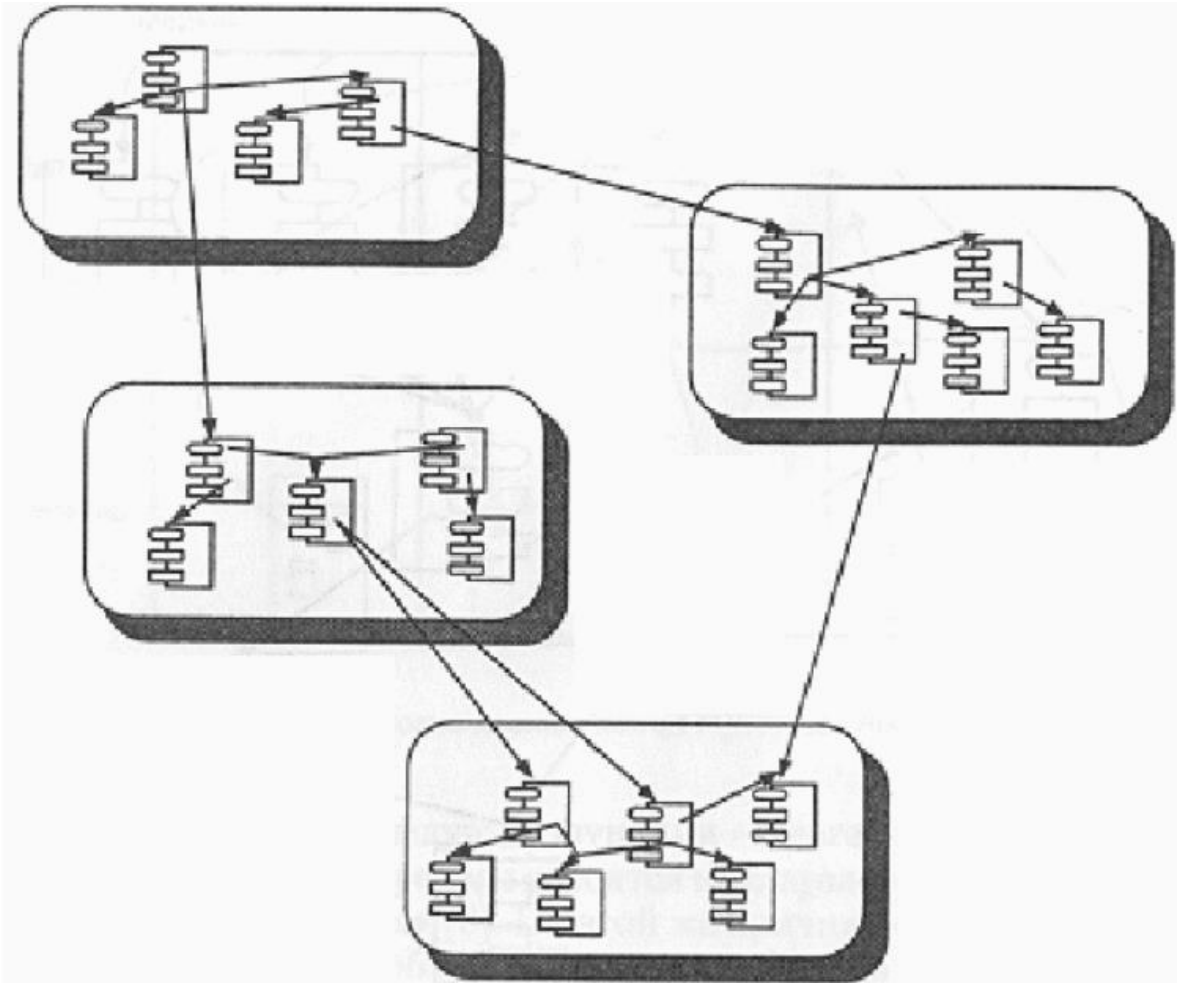
- Абстрагирование, достигаемое посредством использования процедур, хорошо подходит для описания абстрактных действий, но не годится для описания абстрактных объектов
- Во многих практических ситуациях сложность объектов, с которыми нужно работать, составляет основную часть сложности всей задачи
- Возникают методы проектирования на основе потоков данных, которые вносят упорядоченность в абстракцию данных в языках, ориентированных на алгоритмы
- Появляется теория типов, которая воплощается в таких языках, как Pascal



Основным элементом конструкции служит модуль, составленный из логически связанных классов и объектов

# Топология больших приложений в объектных и объектно-ориентированных языках

- Объектный подход масштабируется и может быть применен на все более высоких уровнях.
- Кластеры абстракций в больших системах могут представляться в виде многослойной структуры.
- На каждом уровне можно выделить группы объектов, тесно взаимодействующих для решения задачи более высокого уровня абстракции.
- Внутри каждого кластера находится такое же множество взаимодействующих абстракций



# Истоки становления ОО-подхода

- Объектно-ориентированный подход был связан со следующими событиями:
  - «прогресс в области архитектуры ЭВМ
  - развитие языков программирования, таких как Simula, Smalltalk, CLU, Ada
  - развитие методологии программирования, включая принципы модульности и скрытия данных»
- Оказали влияние на становление объектного подхода:
  - развитие теории баз данных
  - исследования в области искусственного интеллекта
  - достижения философии и теории познания.

# ООП, ООП, ООА

- Объектно-ориентированное программирование — это методология программирования, основанная на представлении программы в виде совокупности **объектов**, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования
- Объектно-ориентированное проектирование — это методология проектирования, соединяющая в себе процесс **объектной декомпозиции** и **приемы представления** логической (классы, объектов) и физической (модули, процессы), а также статической и динамической моделей проектируемой системы
- Объектно-ориентированный анализ — это методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области

# ОО-анализ, ОО-программирование и ОО-проектирование

- **Объектно-ориентированное программирование (**object-oriented programming, OOP)**)**— это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.
- Страуструп: «если термин объектно-ориентированный язык вообще что-либо означает, то он должен означать язык, имеющий средства хорошей поддержки объектно-ориентированного стиля программирования...»
- **Язык программирования является объектно-ориентированным** тогда и только тогда, когда выполняются следующие условия:
  - Поддерживаются объекты, то есть абстракции данных, имеющие интерфейс в виде именованных операций и собственные данные, с ограничением доступа к ним.
  - Объекты относятся к соответствующим типам (классам).
  - Типы (классы) могут наследовать атрибуты супертипов (суперклассов)»

# ОО-анализ, ОО-программирование и ОО-проектирование (2)

- **Объектно-ориентированное проектирование (**object-oriented design, OOD**)** — это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.
- Программирование прежде всего подразумевает правильное и эффективное использование механизмов конкретных языков программирования. Проектирование, напротив, основное внимание уделяет правильному и эффективному структурированию сложных систем
- **Объектно-ориентированный анализ (**object-oriented analysis, OOA**)**— это методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области
- На результатах OOA формируются модели, на которых основывается OOD; OOD в свою очередь создает фундамент для окончательной реализации системы с использованием методологии OOP

# Объектно-ориентированные модели

- Объектно-ориентированный анализ и проектирование — это метод, логически приводящий нас к объектно-ориентированной декомпозиции
- объектно-ориентированное проектирование предлагает богатый выбор моделей

# Основы объектно-ориентированного проектирования

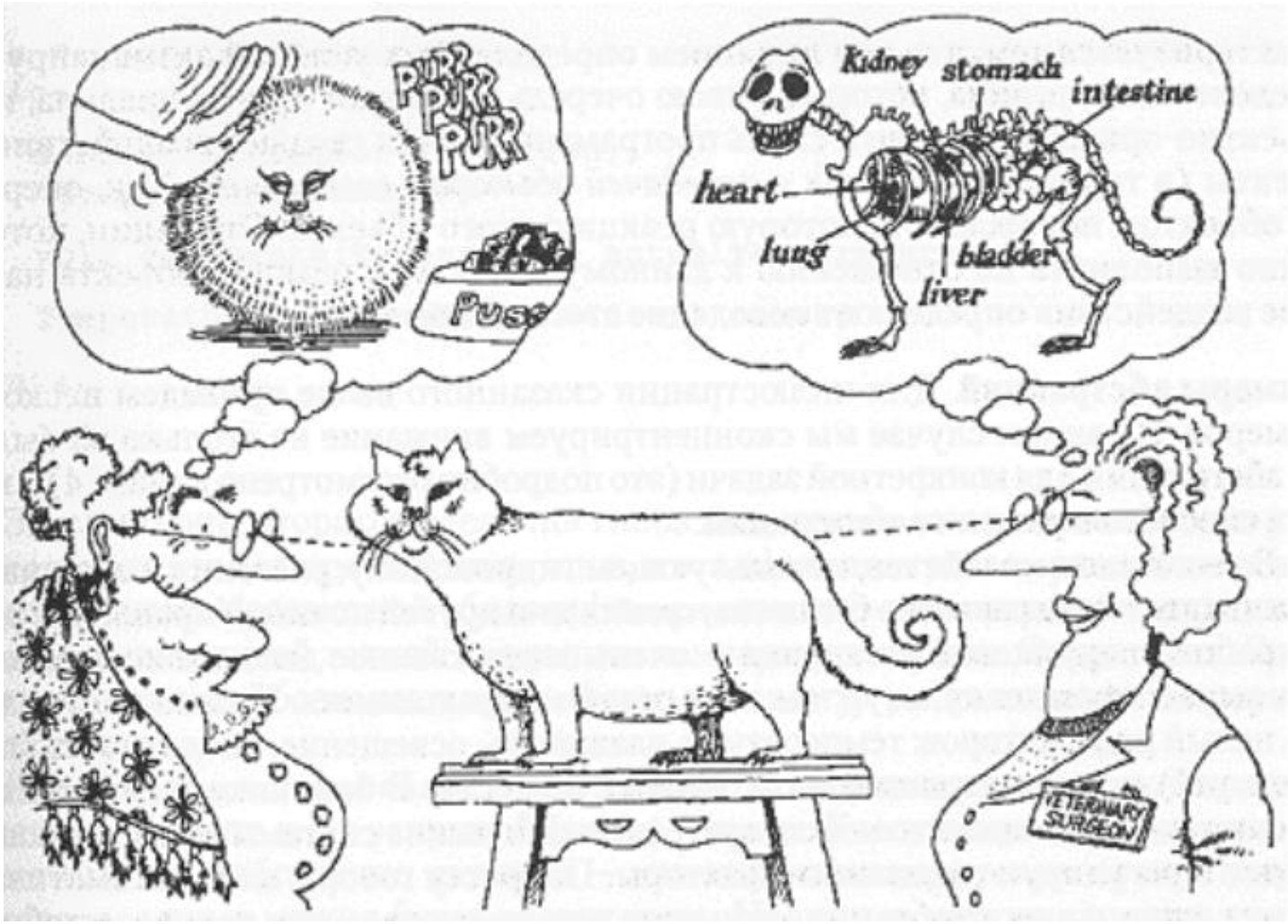
Базовые принципы ООП

- **Абстракция** — отделение концепции от ее экземпляра;
- **Полиморфизм** — реализация задач одной и той же идеи разными способами;
- Наследование — способность объекта или класса базироваться на другом объекте или классе. Это главный механизм для повторного использования кода. Наследственное отношение классов четко определяет их иерархию;
- **Инкапсуляция** — размещение одного объекта или класса внутри другого для разграничения доступа к ним.
- **Модульность** - это свойство системы, которая была разложена на внутренне связные, но слабо связанные между собой модули.
- **Иерархия** — это упорядочение абстракций, расположение их по уровням.
- **Типизация** — это способ защититься от использования объектов одного класса вместо другого, или по крайней мере управлять таким использованием
- **Параллелизм** — это свойство, отличающее активные объекты от пассивных
- **Сохраняемость** — способность объекта существовать во времени, переживая породивший его процесс, и (или) в пространстве, перемещаясь из своего первоначального адресного пространства.

# Абстракция

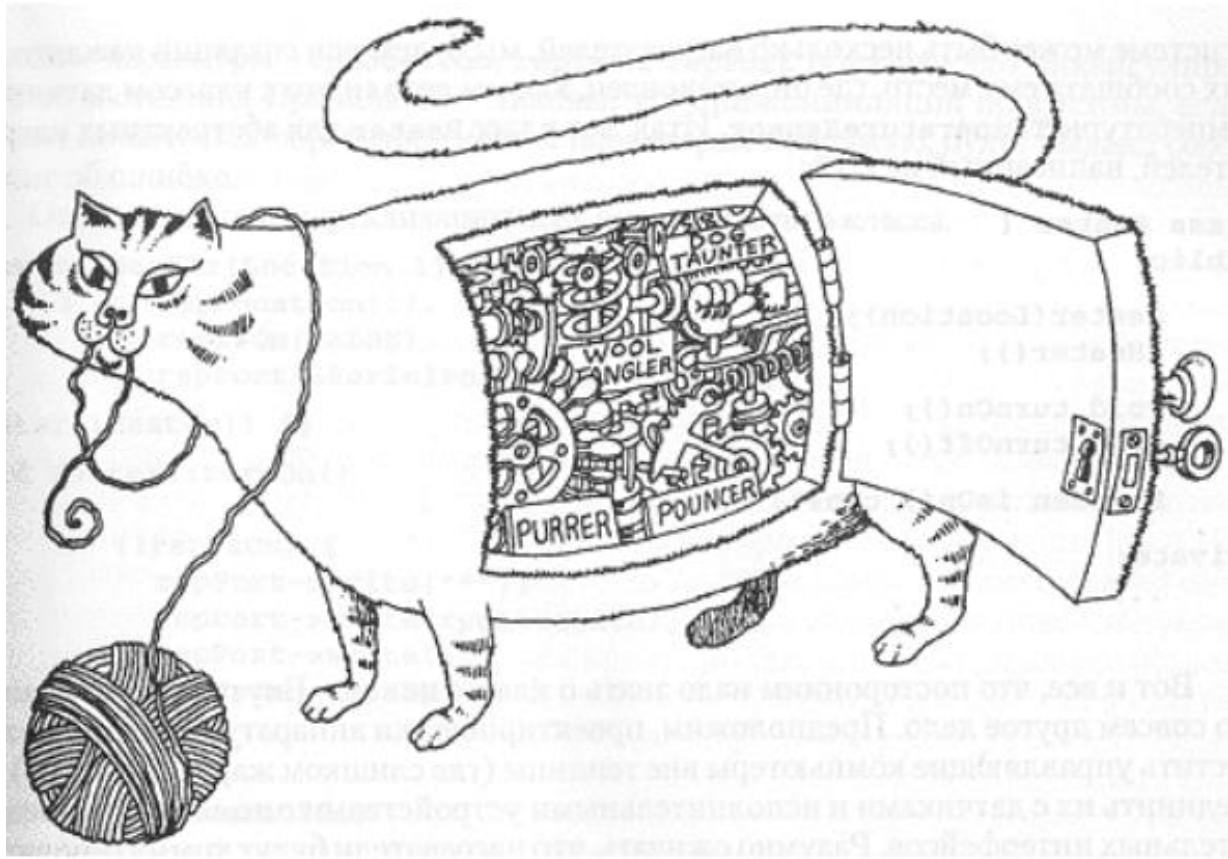
- *Абстракция выделяет существенные характеристики некоторого объекта, отличающие его от всех других видов объектов и, таким образом, четко определяет его концептуальные границы с точки зрения наблюдателя*
- Барьер абстракции - разделение смысла и реализации
- Принцип наименьшего удивления  - абстракция должна охватывать все поведение объекта, но не больше и не меньше, и не привносить сюрпризов или побочных эффектов, лежащих вне ее сферы применимости
- Выбор правильного набора абстракций для заданной предметной области представляет собой главную задачу объектно-ориентированного проектирования

# Абстракция фокусируется на существенных с точки зрения наблюдателя характеристиках объекта



# Инкапсуляция

- Абстракция и инкапсуляция дополняют друг друга: абстрагирование направлено на наблюдаемое поведение объекта, а инкапсуляция занимается внутренним устройством
- *Инкапсуляция — это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение*
- Инкапсуляция служит для того, чтобы изолировать контрактные обязательства абстракции от их реализации
- Никакая часть сложной системы не должна зависеть от внутреннего устройства какой-либо другой части системы




*Инкапсуляция скрывает детали реализации объекта*

# Модульность

- Модули образуют физическую структуру системы
- *Модульность — это свойство системы, которая была разложена на внутренне связные, но слабо связанные между собой модули*
- Барбара Лисков: «Модульность — это разделение программы на фрагменты, которые компилируются по отдельности, но могут устанавливать связи с другими модулями»
- Связи между модулями — это их представления друг о друге
- В большинстве языков, поддерживающих принцип модульности как самостоятельную концепцию, интерфейс модуля отделен от его реализации
- Бритон и Парнас: «Конечной целью декомпозиции программы на модули является снижение затрат на программирование за счет независимой разработки и тестирования. Структура модуля должна быть достаточно простой для восприятия; реализация каждого модуля не должна зависеть от реализации других модулей; должны быть приняты меры для облегчения процесса внесения изменений там, где они наиболее вероятны
- Следует стремиться к тому, чтобы интерфейсная часть модулей была возможно более узкой (в пределах обеспечения необходимых связей)

# Иерархия

- ***Иерархия — это упорядочение абстракций, расположение их по уровням***
- Основными видами иерархических структур применительно к сложным системам являются структура классов (иерархия «is-a») и структура объектов (иерархия «part of»).
- Наследование создает такую иерархию абстракций, в которой подклассы наследуют строение от одного или нескольких суперклассов. Часто подкласс достраивает или переписывает компоненты вышестоящего класса.
- Принцип наследования позволяет упростить выражение абстракций, делает проект менее громоздким и более выразительным
- **Принципы абстрагирования, инкапсуляции и иерархии находятся между собой в некоем здоровом конфликте**
- Существуют три способа нарушения инкапсуляции через наследование: “подкласс может получить доступ к переменным экземпляра своего суперкласса, вызвать закрытую функцию и, наконец, обратиться напрямую к суперклассу своего суперкласса”
- Агрегация позволяет физически сгруппировать логически связанные структуры, а наследование с легкостью копирует эти общие группы в различные абстракции

# Типизация

- Дойч: «тип - точная характеристика свойств, включая структуру и поведение, относящуюся к некоторой совокупности объектов»
- ***Типизация — это способ защититься от использования объектов одного класса вместо другого, или по крайней мере управлять таким использованием.***
- Конкретный язык программирования может иметь сильный или слабый механизм типизации
  - В сильно типизированных языках нарушение согласования типов может быть обнаружено во время трансляции программы
  - Нарушение согласования типов может не обнаружиться во время трансляции и обычно проявляется как ошибка исполнения. C++ тяготеет к сильной типизации, но в этом языке правила типизации можно игнорировать или подавить полностью
- Преобразований типа надо избегать, поскольку они часто представляют собой нарушение принятой системы абстракций
- Статическое и динамическое связывание. **Полиморфизм возникает там, где взаимодействуют наследование и динамическое связывание**

# Параллелизм

- Процесс (поток управления) — это фундаментальная единица действия в системе.
- Каждая программа имеет по крайней мере один поток управления, параллельная система имеет много таких потоков
- Реальная параллельность достигается только на многопроцессорных системах, а системы с одним процессором имитируют параллельность за счет алгоритмов разделения времени
- Многие современные операционные системы предусматривают прямую поддержку параллелизма, и это обстоятельство очень благоприятно сказывается на возможности обеспечения параллелизма в объектно-ориентированных системах
- В то время, как объектно-ориентированное программирование основано на абстракции, инкапсуляции и наследовании, параллелизм главное внимание уделяет абстрагированию и синхронизации процессов

# Параллелизм

- Каждый объект может представлять собой отдельный поток управления (абстракцию процесса). Такой объект называется активным.
- На основе  ООП мир может быть представлен, как совокупность взаимодействующих объектов, часть из которых является активной и выступает в роли независимых вычислительных центров
- *Параллелизм — это свойство, отличающее активные объекты от пассивных.*
- Проблема синхронизации отношений активных объектов друг с другом, а также с остальными объектами, действующими последовательно.
- Например, если два объекта посылают сообщения третьему, должен быть какой-то механизм, гарантирующий, что объект, на который направлено действие, не разрушится при одновременной попытке двух активных объектов изменить его состояние.
- В этом вопросе соединяются абстракция, инкапсуляция и параллелизм.
- В параллельных системах недостаточно определить поведение объекта, **надо еще принять меры, гарантирующие, что он не будет разрушен под воздействием нескольких независимых процессов.**

# Сохраняемость

- Любой программный объект существует в памяти и живет во времени
- Программисты разрабатывают специальные схемы для сохранения объектов в период между запусками программы, а конструкторы баз данных переиначивают свою технологию под короткоживущие объекты
- Введение сохраняемости, как нормальной составной части объектного подхода привело к объектно-ориентированным базам данных (OODB, object-oriented databases).
- На практике подобные базы данных строятся на основе проверенных временем моделей — последовательных, индексированных, иерархических, сетевых или реляционных, но программист может ввести абстракцию объектно-ориентированного интерфейса, через который запросы к базе данных и другие операции выполняются в терминах объектов, время жизни которых превосходит время жизни отдельной программы
- *Сохраняемость — способность объекта существовать во времени, переживая породивший его процесс, и (или) в пространстве, перемещаясь из своего первоначального адресного пространства.*

# Использование объектного подхода

- В настоящее время объектно-ориентированное проектирование — единственная методология, позволяющая справиться со сложностью, присущей очень большим системам

| | |
|---|---|
| Авиационное оборудование | Обработка коммерческой информации |
| Автоматизация учреждении | Операционные системы |
| Автоматизированное обучение | Планирование инвестиций |
| Автоматизированное проектирование | Повторно используемые компоненты |
| Автоматизированное производство программного обеспечения | Средства разработки программ и космической техники |
| Анимация | Программные средства космических станций |
| Базы данных | Проектирование интерфейса пользователя |
| Банковское дело | Проектирование СБИС |
| Гипермедиа | Распознавание образов |
| Кинопроизводство | Робототехника |
| Контроль программного обеспечения | Системы телеметрии |
| Математический анализ | Системы управления и регулирования |
| Медицинская электроника | Подготовка документов |
| Моделирование авиационной | Телекоммуникации |
| Музыкальная композиция | Управление воздушным движением |
| Написание сценариев | Управление химическими процессами |
| Нефтяная промышленность | Экспертные системы |

# Выводы

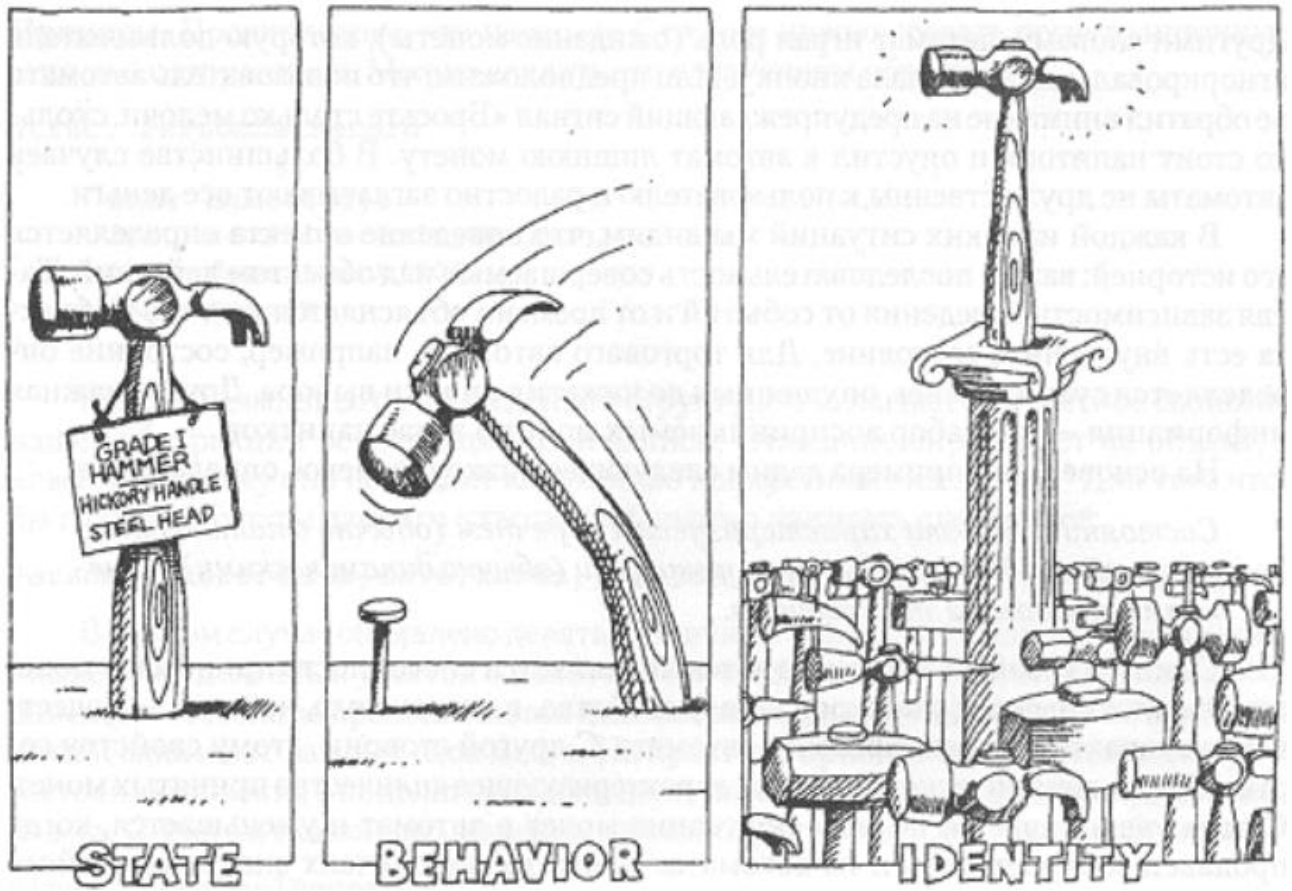
- Развитие программной индустрии привело к созданию методов объектноориентированного анализа, проектирования и программирования, которые служат для программирования «в большом».
- В программировании существует несколько парадигм, ориентированных на процедуры, объекты, логику, правила и ограничения.
- Абстракция определяет существенные характеристики некоторого объекта, которые отличают его от всех других видов объектов и, таким образом, абстракция четко очерчивает концептуальную границу объекта с точки зрения наблюдателя.
- Инкапсуляция — это процесс разделения устройства и поведения объекта; инкапсуляция служит для того, чтобы изолировать контрактные обязательства абстракции от их реализации.

# Классы и объекты

“Подобно тому, как взявший в руки молоток начинает видеть во всем окружающем только гвозди, проектировщик с объектно-ориентированным мышлением начинает воспринимать весь мир в виде объектов”

# Объекты

- *Объект обладает состоянием, поведением и идентичностью*
- *Структура и поведение схожих объектов определяет общий для них класс*
- Термины "экземпляр класса" и "объект" взаимозаменяемы.
- Объект имеет состояние, обладает некоторым хорошо определенным поведением и уникальной идентичностью



# Состояние

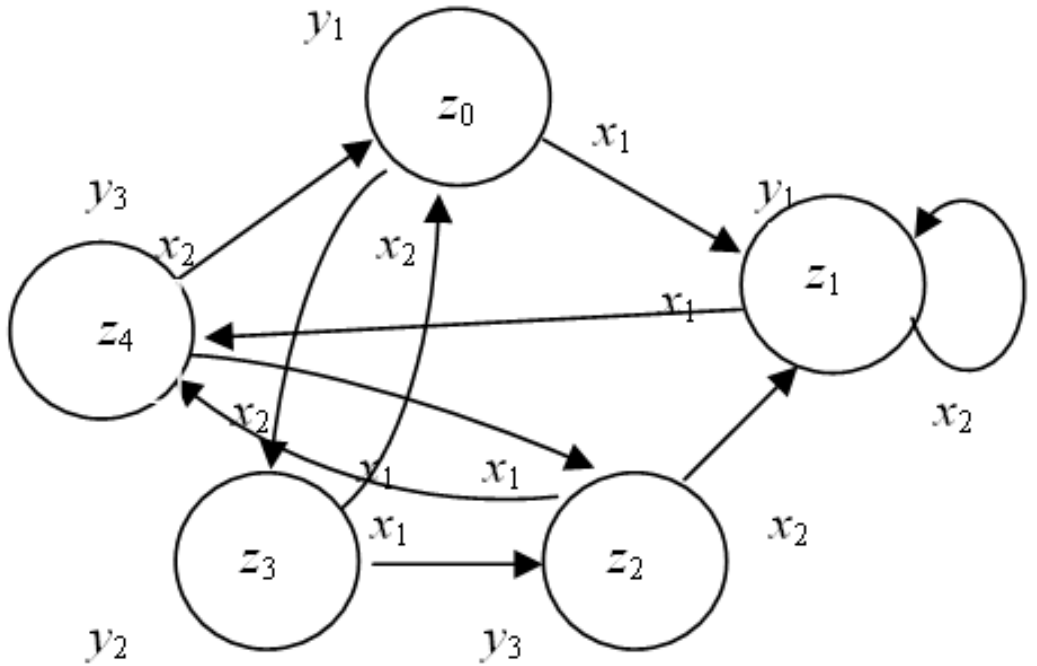
- Поведение объекта определяется его историей: важна последовательность совершаемых над объектом действий.
- Зависимость поведения от событий и от времени обуславливается наличием *внутренних состояний*
- Пример: торговый автомат
- *Состояние объекта характеризуется перечнем (обычно статическим) всех свойств данного объекта и текущими (обычно динамическими) значениями каждого из этих свойств*
- Перечень свойств объекта является, как правило, статическим, поскольку эти свойства составляют неизменяемую основу объекта. Все свойства имеют некоторые значения

# Поведение

- *Поведение - это то, как объект действует и реагирует;*
- *Поведение выражается в терминах состояния объекта и передачи сообщений*
- Поведение объекта - это его наблюдаемая и проверяемая извне деятельность
- В объектно-ориентированных языках операции, выполняемые над данным объектом, называются методами и входят в определение класса объекта
- Состояние объекта также влияет на его поведение
- Состояние объекта представляет суммарный результат его поведения.

# Объекты как автоматы

- Объект можно представить в качестве маленькой независимой машины
- Активный объект имеет свой поток управления, а пассивный - нет.
- Активный объект в общем случае автономен, то есть он может проявлять свое поведение без воздействия со стороны других объектов.
- Пассивный объект, напротив, может изменять свое состояние только под воздействием других объектов
- Активные объекты системы - источники управляющих воздействий.



# Идентичность

- *Идентичность - это такое свойство объекта, которое отличает его от всех других объектов*
- В большинстве языков программирования и управления базами данных для различения временных объектов их именуют, тем самым путая адресуемость и идентичность.
- Большинство баз данных различают постоянные объекты по ключевому атрибуту, тем самым смешивая идентичность и значение данных".
- Источником множества ошибок в объектно-ориентированном программировании является неумение отличать имя объекта от самого объекта.
- Проблемы: синонимы, использование указателей на объекты.
- Все это источники утечки памяти.

# Класс

# Принципы объектно-ориентированного проектирования

- Инкапсулируйте все, что может изменяться;
- Уделяйте больше внимания интерфейсам, а не их реализациям;
- Каждый класс в вашем приложении должен иметь только одно назначение;
- Классы — это их поведение и функциональность.

Используйте следующее вместе с наследованием

- Делегация — перепоручение задачи от внешнего объекта внутреннему;
- Композиция — включение объектом-контейнером объекта-содержимого и управление его поведением; последний не может существовать вне первого;
- Агрегация — включение объектом-контейнером ссылки на объект-содержимое; при уничтожении первого последний продолжает существование.

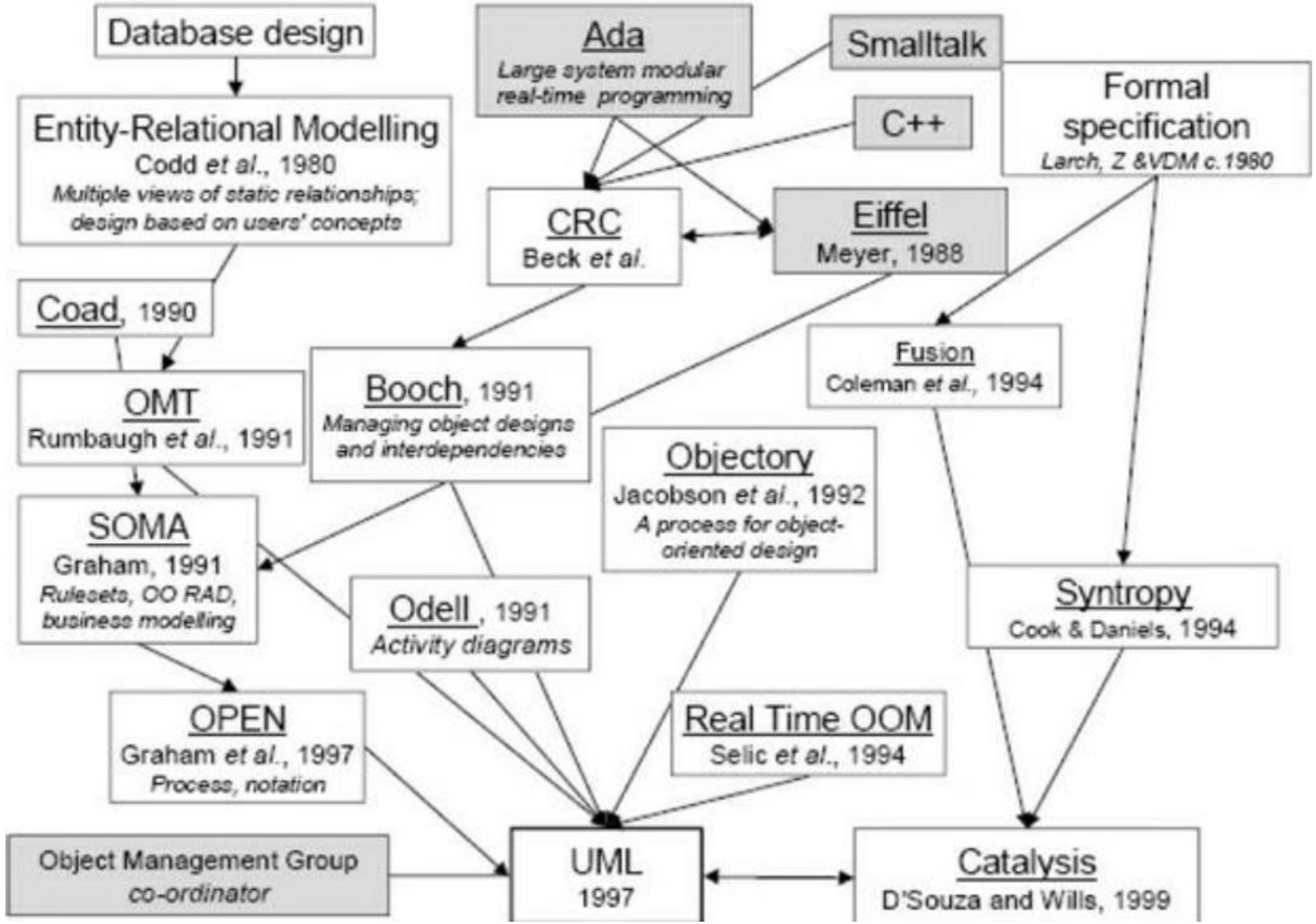
- Не повторяйся (Don’t repeat yourself — DRY)
  - Избегайте повторного написания кода, вынося в абстракции часто используемые задачи и данные. Каждая часть вашего кода или информации должна находиться в единственном числе в единственном доступном месте. Это один из принципов читаемого кода.
- Принцип единственной обязанности
  - Для каждого класса должно быть определено единственное назначение. Все ресурсы, необходимые для его осуществления, должны быть инкапсулированы в этот класс и подчинены только этой задаче.
- Принцип открытости/закрытости
  - Программные сущности должны быть открыты для расширения, но закрыты для изменений.

- Принцип подстановки Барбары Лисков
  - Методы, использующие некий тип, должны иметь возможность использовать его подтипы, не зная об этом.
- Принцип разделения интерфейсов
  - Предпочтительнее разделять интерфейсы на более мелкие тематические, чтобы реализующие их классы не были вынуждены определять методы, которые непосредственно в них не используются.
- Принцип инверсии зависимостей
  - Система должна конструироваться на основе абстракций «сверху вниз»: не абстракции должны формироваться на основе деталей, а детали должны формироваться на основе абстракций.

# Выводы

- Программам присуща сложность, которая нередко превосходит возможности человеческого разума.
- Задача разработчиков программных систем — создать у пользователя разрабатываемой системы иллюзию простоты.
- Познавательные способности человека ограничены; мы можем раздвинуть их рамки, используя декомпозицию, выделение абстракций и создание иерархий.
- Сложные структуры часто принимают форму иерархий; полезны обе иерархии: и классов, и объектов.
- Сложные системы обычно создаются на основе устойчивых промежуточных форм.
- Сложные системы можно исследовать, концентрируя основное внимание либо на объектах, либо на процессах; имеются веские основания использовать объектно-ориентированную декомпозицию, при которой мир рассматривается как упорядоченная совокупность объектов, которые в процессе взаимодействия друг с другом определяют поведение системы.
- Объектно-ориентированный анализ и проектирование — метод, использующий объектную декомпозицию; объектно-ориентированный подход имеет свою систему условных обозначений и предлагает богатый набор логических и физических моделей, с помощью которых мы можем получить представление о различных аспектах рассматриваемой системы.

# Основы UML


Database design
Entity-Relational Modelling
Codd et al., 1980
Multiple views of static relationships; design based on users' concepts
Coad, 1990
OMT
Rumbaugh et al., 1991
SOMA
Graham, 1991
Rulesets, OO RAD, business modelling
OPEN
Graham et al., 1997
Process, notation
Object Management Group
co-ordinator
Ada
Large system modular real-time programming
Smalltalk
C++
Formal specification
Larch, Z &VDM c.1980
CRC
Beck et al.
Eiffel
Meyer, 1988
Booch, 1991
Managing object designs and interdependencies
Fusion
Coleman et al., 1994
Objectory
Jacobson et al., 1992
A process for object-oriented design
Odell, 1991
Activity diagrams
Syntropy
Cook & Daniels, 1994
Real Time OOM
Selic et al., 1994
UML
1997
Catalysis
D'Souza and Wills, 1999

# Основы UML- история

- Цель UML — предоставить стандартную нотацию, которая может использоваться всеми объектно-ориентированными методами
- Техника объектного моделирования OMT [James Rumbaugh 1991], которая была лучшей для анализа информационных систем с большим объемом данных.
- Booch [Grady Booch 1994] — Грэди Буч много работал с языком Ада и был крупным игроком в разработке объектно-ориентированных методов для языка. Метод Буча был сильным, нотация была воспринята менее хорошо, например, в его моделях преобладали формы облаков, что выглядело не очень аккуратно.
- OOSE (объектно-ориентированная программная инженерия [Ivar Jacobson 1992]) — модель, известная как модель прецедентов — это мощная методология для понимания поведения всей системы, область, где ООП традиционно была слабой.
- 1995 – объединение усилий: новый унифицированный метод, который теперь называется Unified Modeling Language.

# Цели UML

- Предоставить пользователям готовый, выразительный язык визуального моделирования, чтобы они могли разрабатывать и обмениваться осмысленными моделями.
- Обеспечить механизмы расширяемости и специализации для расширения основных понятий.
- Быть независимым от конкретных языков программирования и процессов разработки.
- Обеспечить формальную основу для понимания языка моделирования.
- Поощрять рост рынка объектно-ориентированных инструментов.
- Поддержка высокоуровневых концепций разработки, таких как совместная работа, структуры, шаблоны и компоненты.
- Интегрировать лучшие практики.

# ЧТО ВЫ МОЖЕТЕ МОДЕЛИРОВАТЬ С ПОМОЩЬЮ UML?

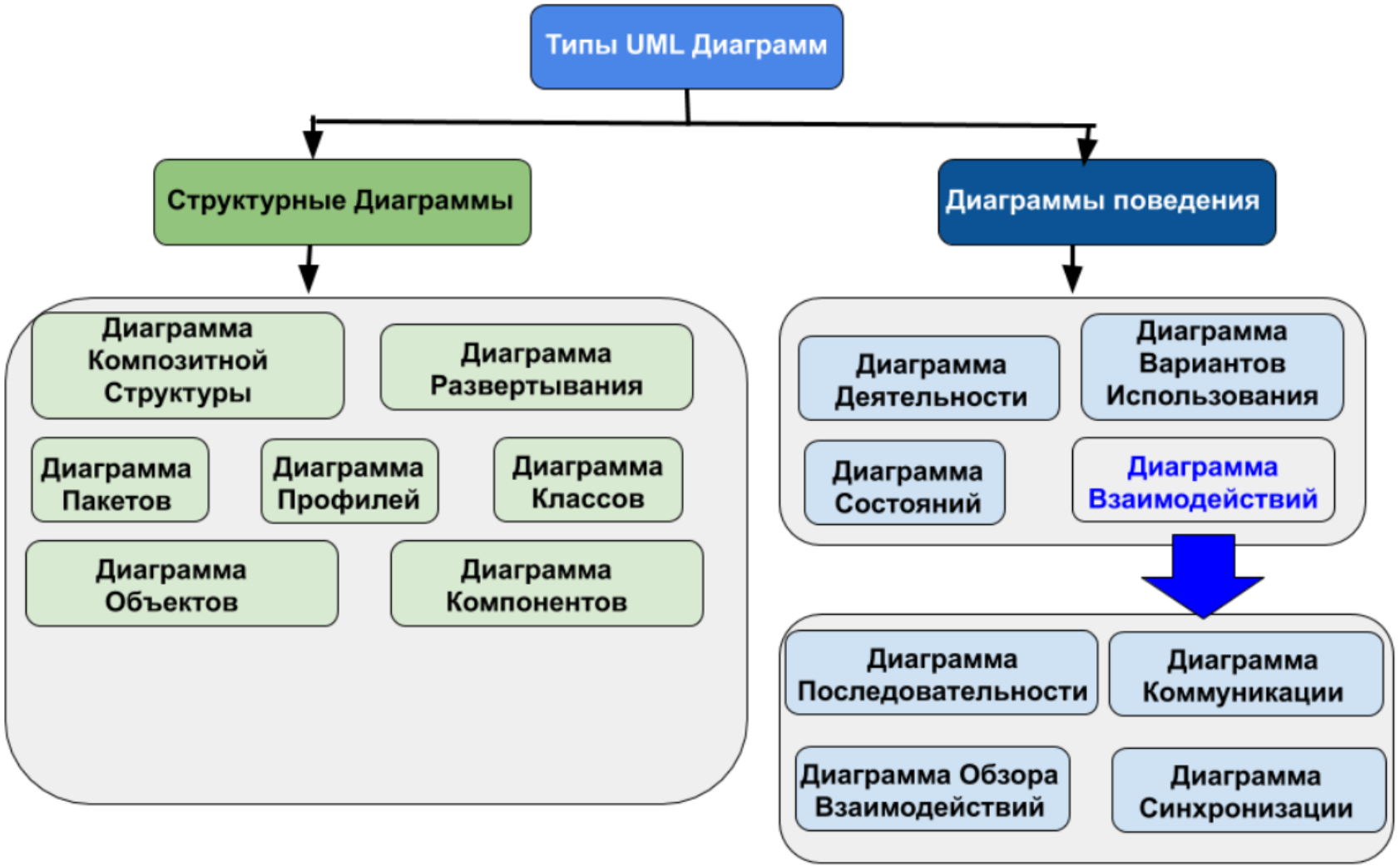
UML 2.0 определяет тринадцать типов диаграмм, разделённых на три категории:

- шесть типов диаграмм представляют статическую структуру приложения;
- три представляют общие типы поведения; и
- четыре представляют различные аспекты взаимодействия:

- Структурная схема включает диаграмму классов, диаграмму объектов, диаграмму компонентов, составную структурную схему, диаграмму пакетов и диаграмму развертывания.
- Диаграммы поведения включают диаграмму вариантов использования (используется некоторыми методологиями при сборе требований); диаграмму операций и диаграмму конечного автомата.
- Диаграммы взаимодействия, полученные из более общей схемы поведения, включают диаграмму последовательности, диаграмму связи, временную диаграмму и обзорную диаграмму взаимодействия.

# Основы UML

- Диаграммы UML подразделяют на два типа — это структурные диаграммы и диаграммы поведения.

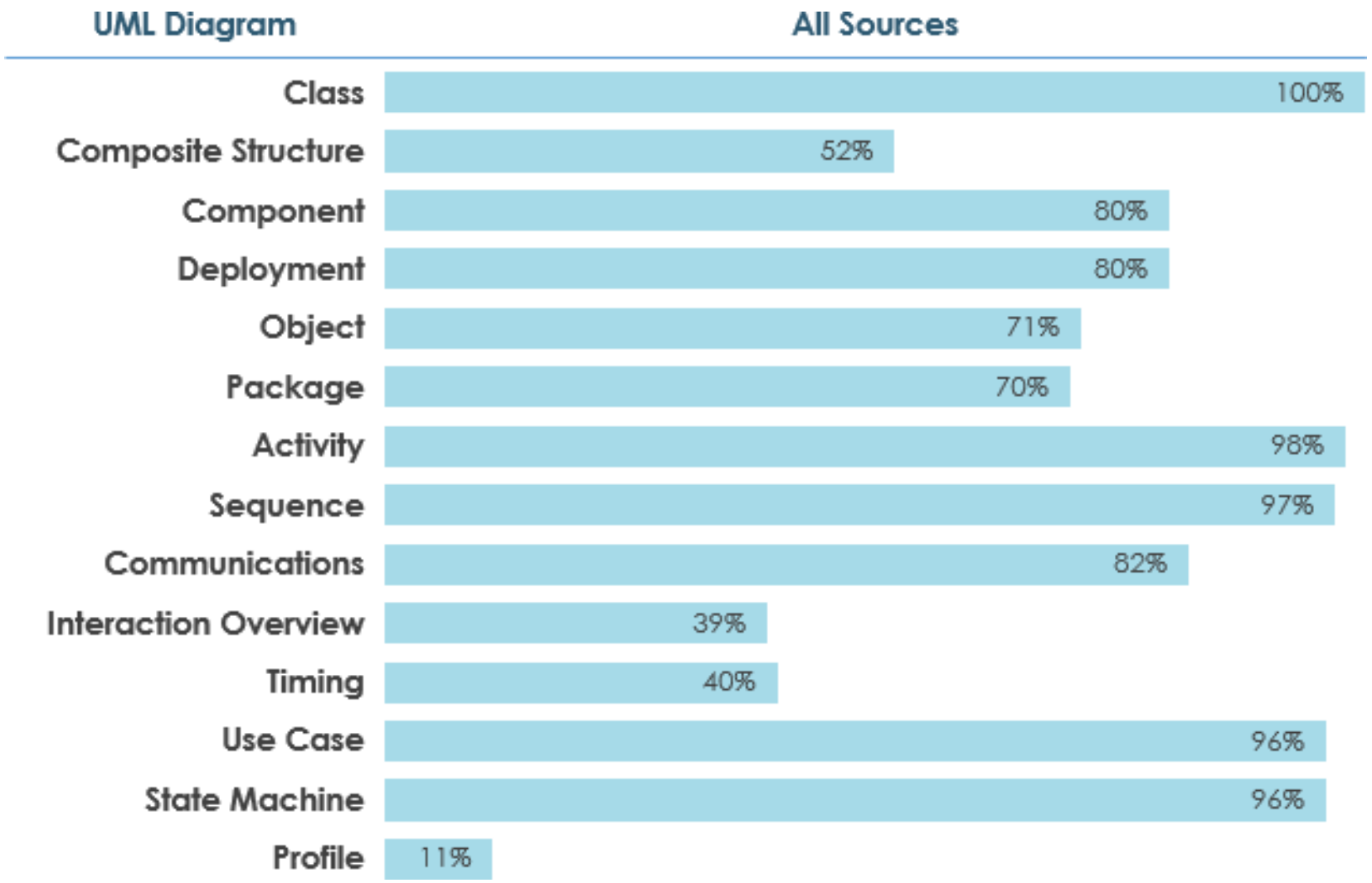


# Типы структурных диаграмм

- Структурные диаграммы показывают статическую структуру системы и ее частей на разных уровнях абстракции и реализации, а также их взаимосвязь. Элементы в структурной диаграмме представляют значимые понятия системы и могут включать в себя абстрактные, реальные концепции и концепции реализации
  - Диаграмма составной структуры
  - Диаграмма развертывания
  - Диаграмма пакетов
  - Диаграмма профилей
  - Диаграмма классов
  - Диаграмма объектов
  - Диаграмма компонентов

# Наиболее важные диаграммы и нотации UML

- Грэди Буч: «для 80% всего программного обеспечения требуется только 20% UML».



# Диаграммы поведения

- Диаграммы поведения показывают динамическое поведение объектов в системе, которое можно описать, как серию изменений в системе с течением времени.
  - Диаграмма деятельности
  - Диаграмма прецедентов
  - Диаграмма состояний
  - Диаграмма последовательности
  - Диаграмма коммуникаций
  - Диаграмма обзора взаимодействия
  - Временная диаграмма

# Диаграмма классов

- Диаграмма классов — это центральная методика моделирования, которая используется практически во всех объектно-ориентированных методах. Эта диаграмма описывает типы объектов в системе и различные виды статических отношений, которые существуют между ними.

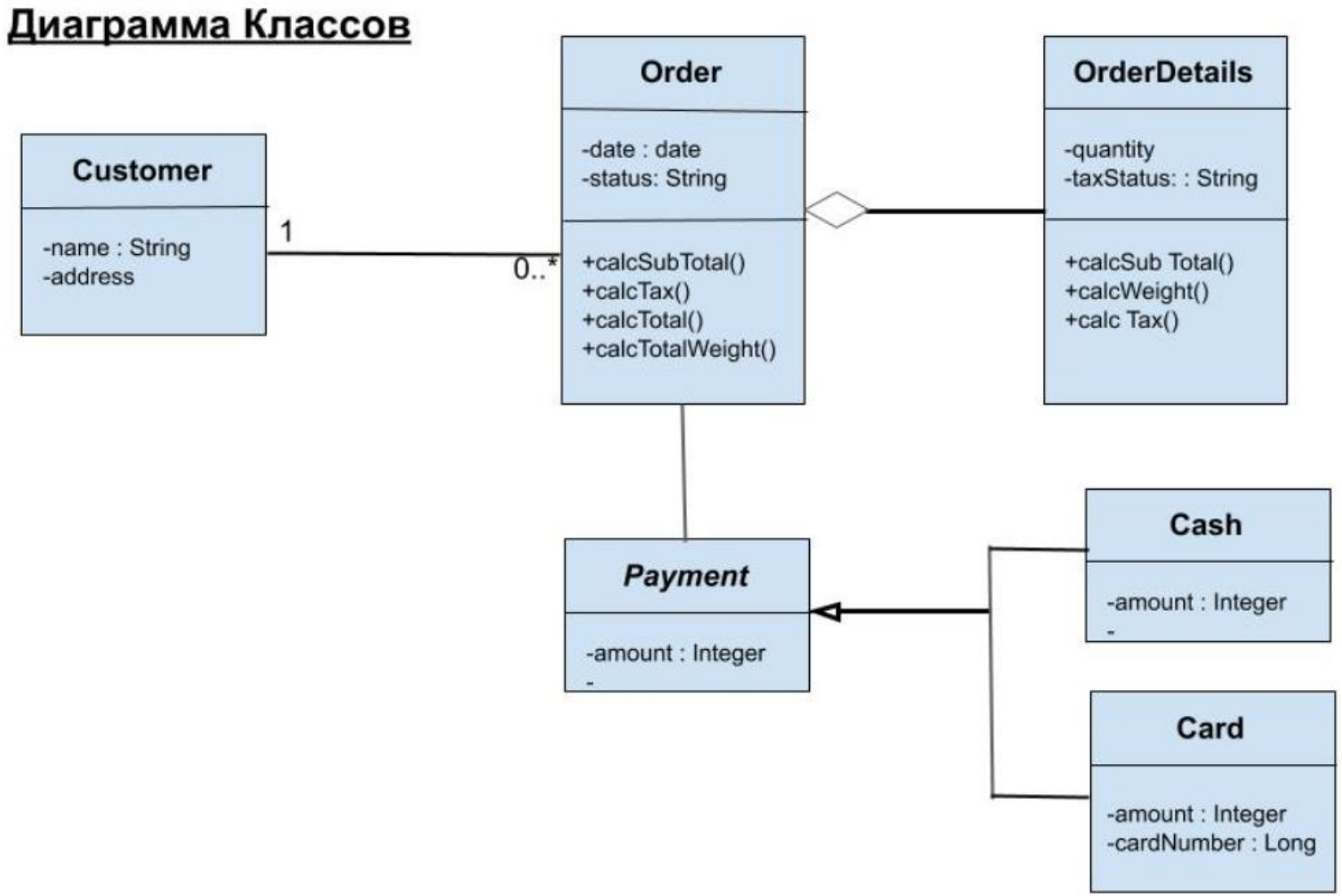
  Три наиболее важных типа отношений в диаграммах классов (на самом деле их больше), это:

  Ассоциация, которая представляет отношения между экземплярами типов, к примеру, человек работает на компанию, у компании есть несколько офисов.

  Наследование, которое имеет непосредственное соответствие наследованию в Объектно-Ориентированном дизайне.

  Агрегация, которая представляет из себя форму композиции объектов в объектно-ориентированном дизайне.

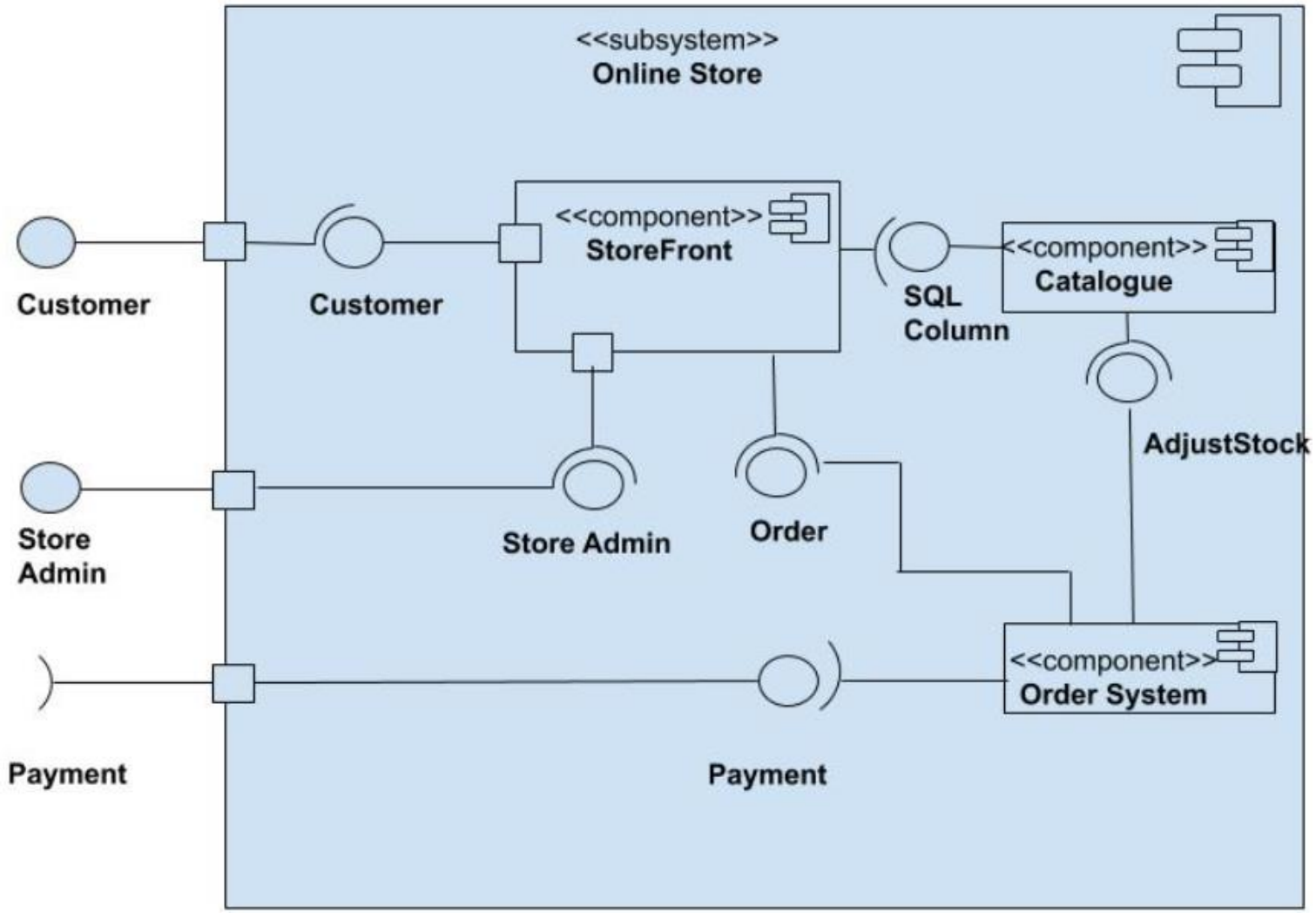
# Диаграмма классов



# Диаграмма компонентов

- На языке унифицированного моделирования диаграмма компонентов показывает, как компоненты соединяются вместе для формирования более крупных компонентов или программных систем.
- Она иллюстрирует архитектуры компонентов программного обеспечения и зависимости между ними.
- Эти программные компоненты включают в себя компоненты времени выполнения, исполняемые компоненты, а также компоненты исходного кода.

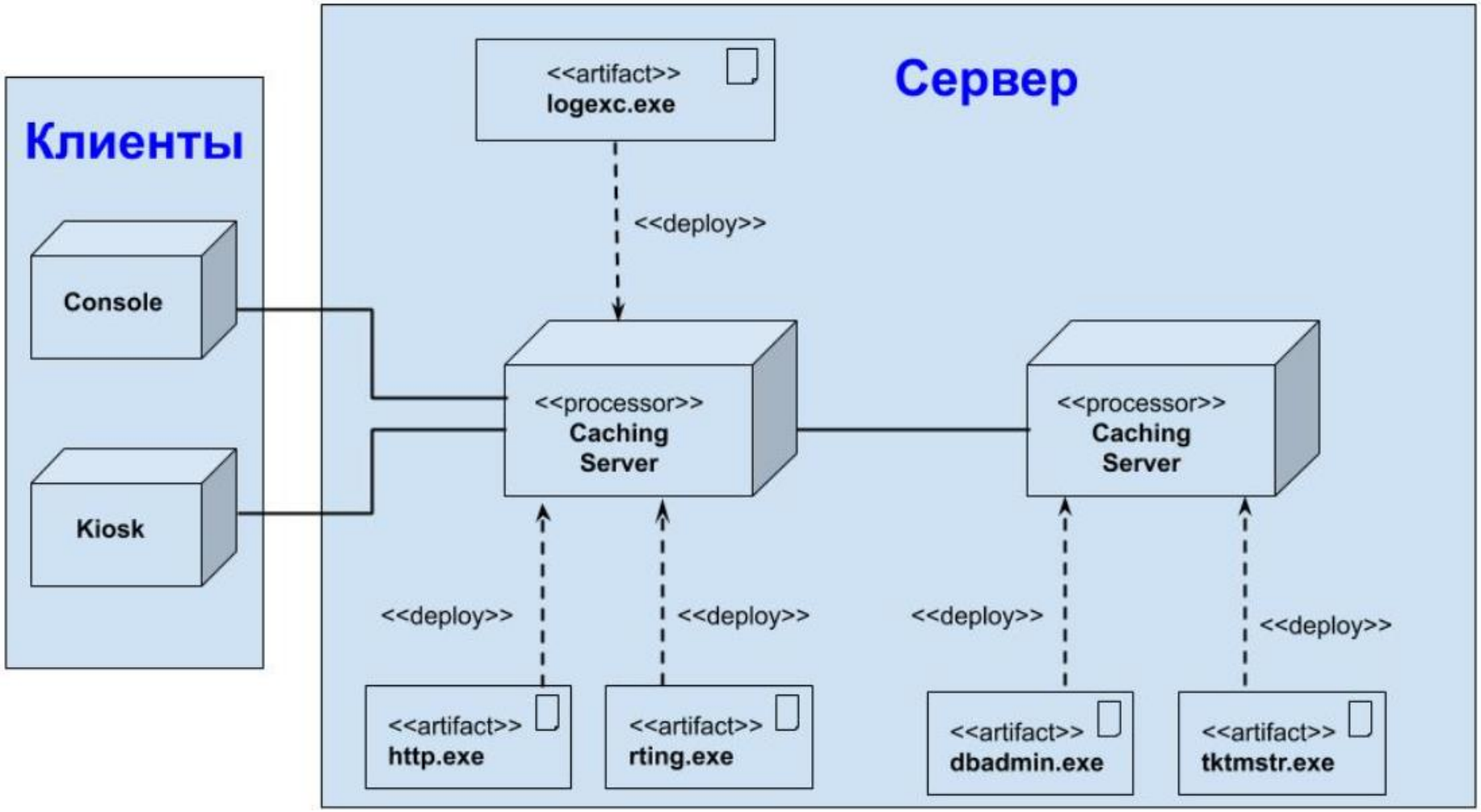
# Диаграмма компонентов



# Диаграмма развертывания

- Диаграмма развертывания помогает моделировать физический аспект объектно-ориентированной программной системы. Это структурная схема, которая показывает архитектуру системы, как развертывание (дистрибуции) программных артефактов.
- Артефакты представляют собой конкретные элементы в физическом мире, которые являются результатом процесса разработки.
- Диаграмма моделирует конфигурацию времени выполнения в статическом представлении и визуализирует распределение артефактов в приложении.
- В большинстве случаев это включает в себя моделирование конфигураций оборудования вместе с компонентами программного обеспечения, на которых они размещены.

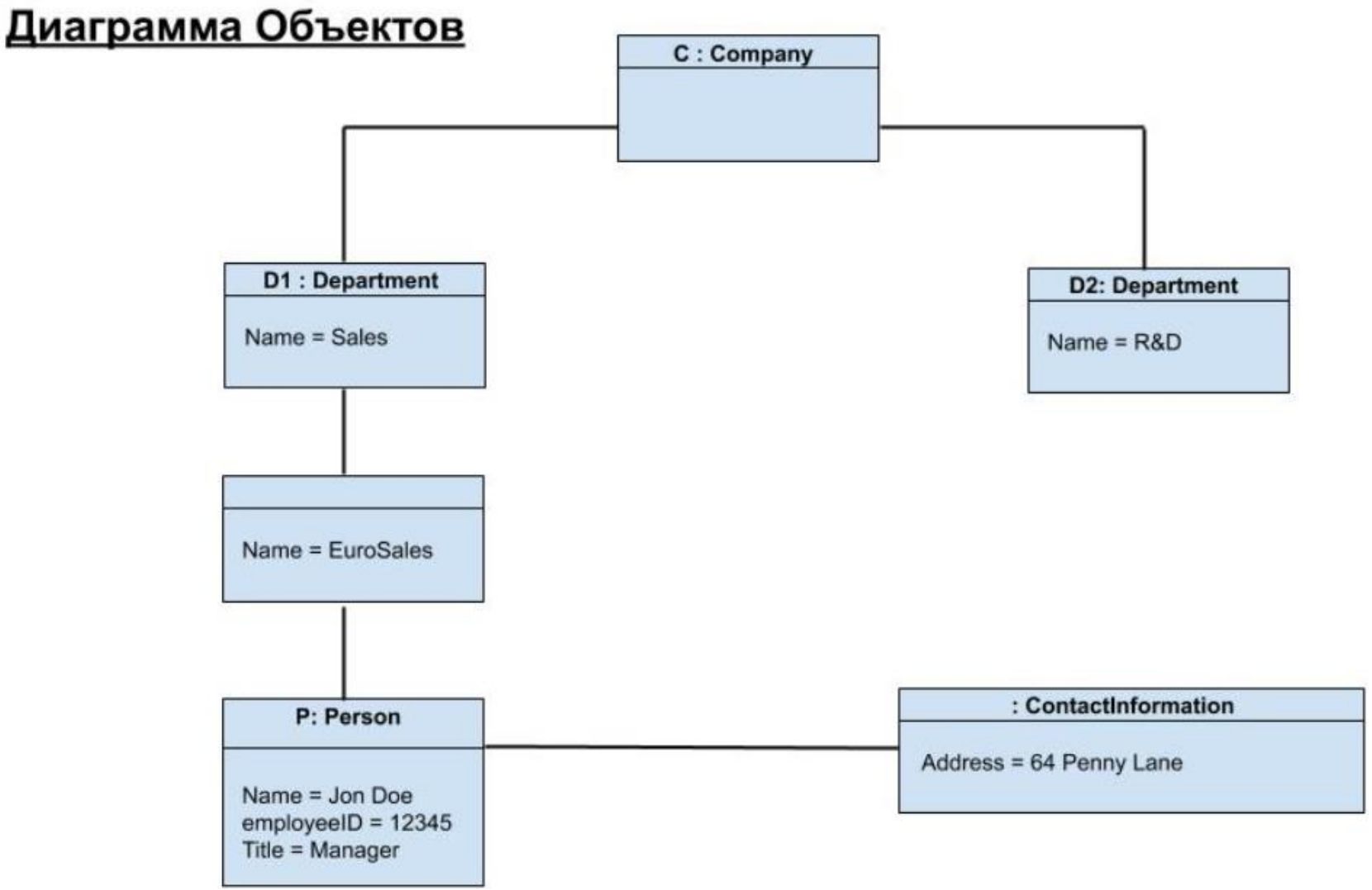
# Диаграмма развертывания



# Диаграмма объектов

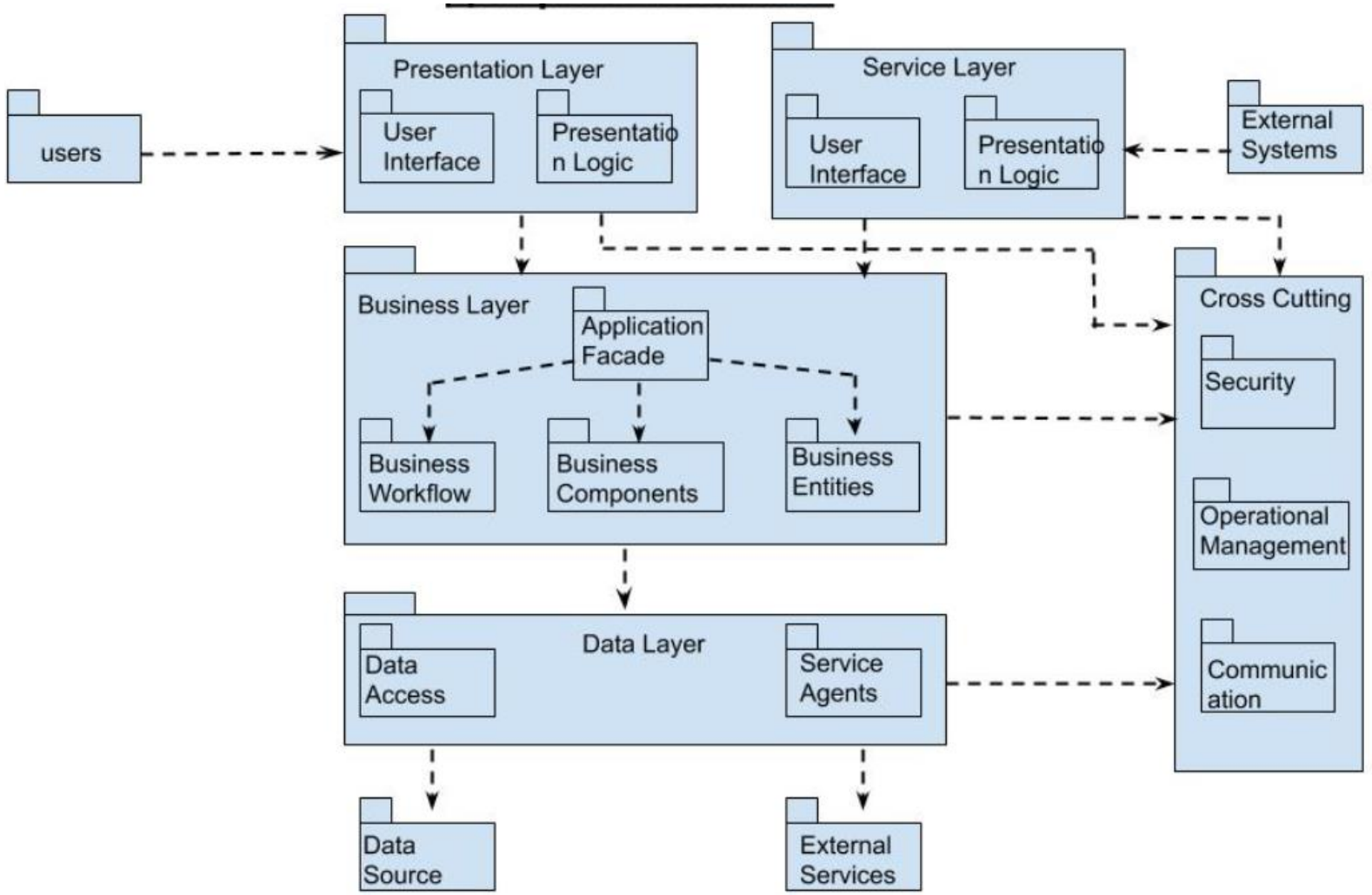
- Статическая диаграмма объектов является экземпляром диаграммы класса; она показывает снимок подробного состояния системы в определенный момент времени. Разница в том, что диаграмма классов представляет собой абстрактную модель, состоящую из классов и их отношений.
- Тем не менее, диаграмма объекта представляет собой экземпляр в конкретный момент, который имеет конкретный характер.
- Использование диаграмм объектов довольно ограничено, а именно — чтобы показать примеры структуры данных.

# Диаграмма объектов



# Диаграмма пакетов

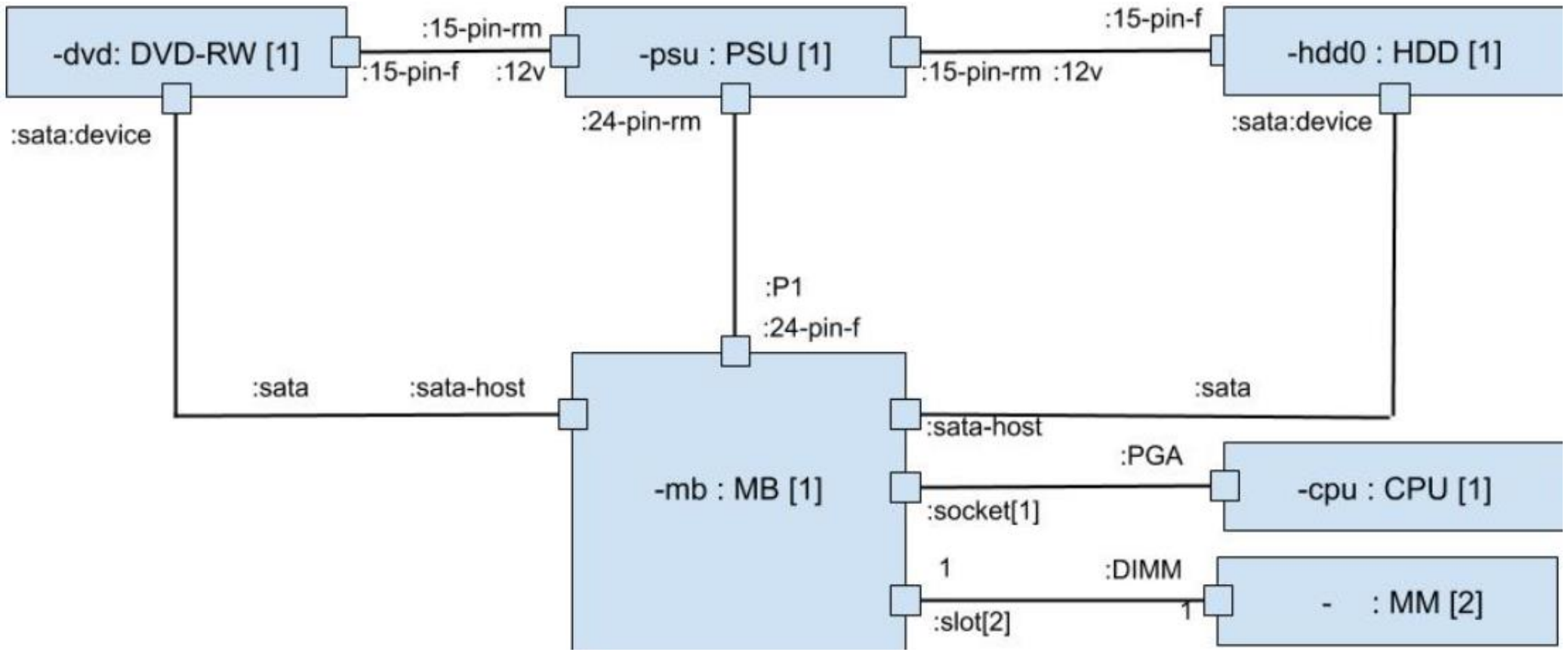
- Диаграмма пакетов — это структурная схема UML, которая показывает пакеты и зависимости между ними.
- Она позволяет отображать различные виды системы, например, легко смоделировать многоуровневое приложение.



# Диаграмма составной структуры

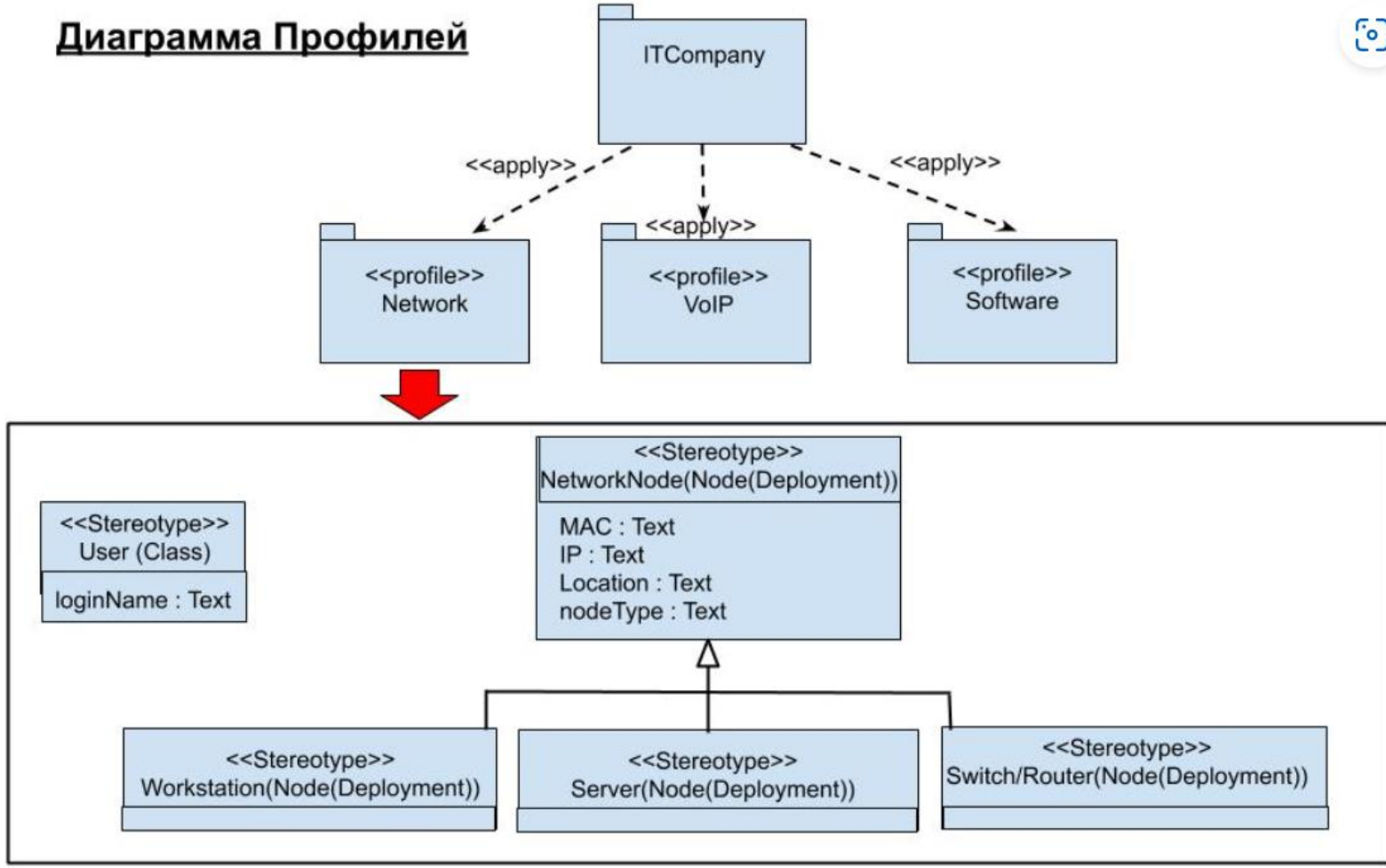
- Диаграмма составной структуры аналогична диаграмме классов и является своего рода диаграммой компонентов, используемой в основном при моделировании системы на микроуровне, но она изображает отдельные части вместо целых классов.
- Это тип статической структурной диаграммы, которая показывает внутреннюю структуру класса и взаимодействия, которые эта структура делает возможными.
- Эта диаграмма может включать внутренние части, порты, через которые части взаимодействуют друг с другом или через которые экземпляры класса взаимодействуют с частями и с внешним миром, и соединители между частями или портами.
- Составная структура — это набор взаимосвязанных элементов, которые взаимодействуют во время выполнения для достижения какой-либо цели. Каждый элемент имеет определенную роль в сотрудничестве.

# Диаграмма составной структуры



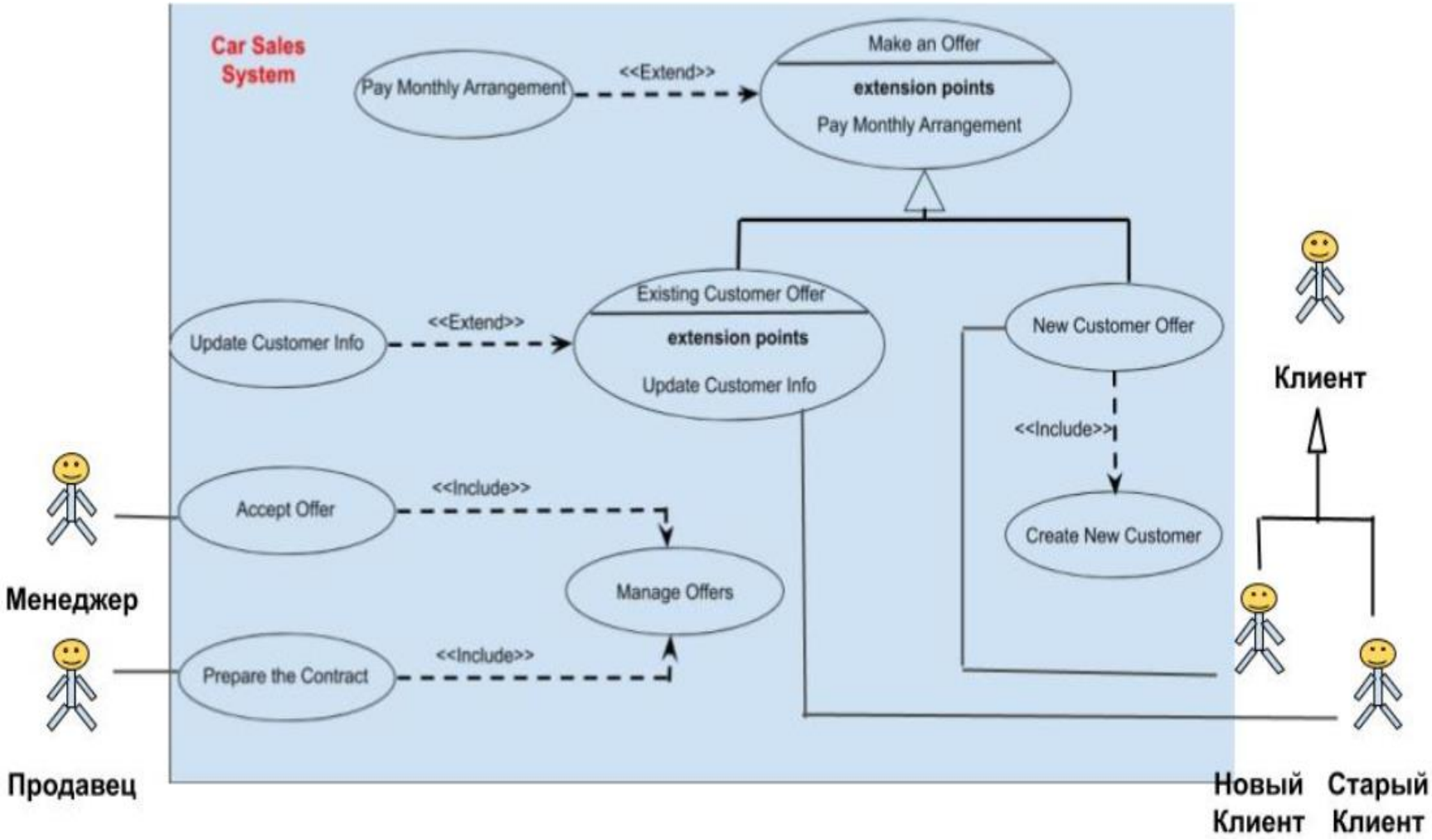
# Диаграмма профилей

- Диаграмма профилей позволяет нам создавать специфичные для домена и платформы стереотипы и определять отношения между ними.
- Мы можем создавать стереотипы, рисуя формы стереотипов и связывая их с композицией или обобщением через интерфейс, ориентированный на ресурсы.
- Мы также можем определять и визуализировать значения стереотипов.



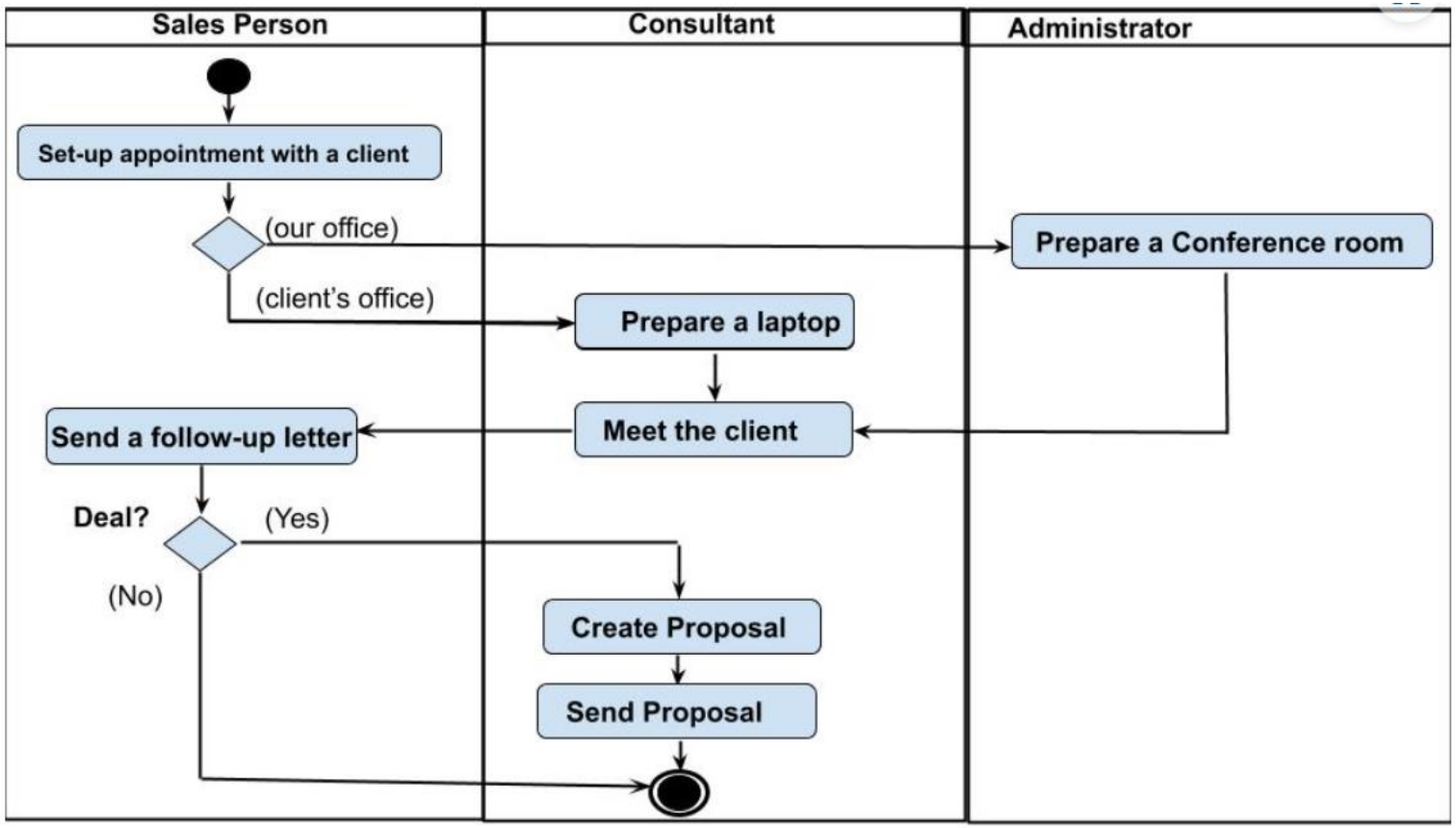
# Диаграмма прецедентов

- Диаграмма прецедентов описывает функциональные требования системы с точки зрения прецедентов.
- По сути дела, это модель предполагаемой функциональности системы (прецедентов) и ее среды (акторов).
- Прецеденты позволяют связать то, что нам нужно от системы с тем, как система удовлетворяет эти потребности.



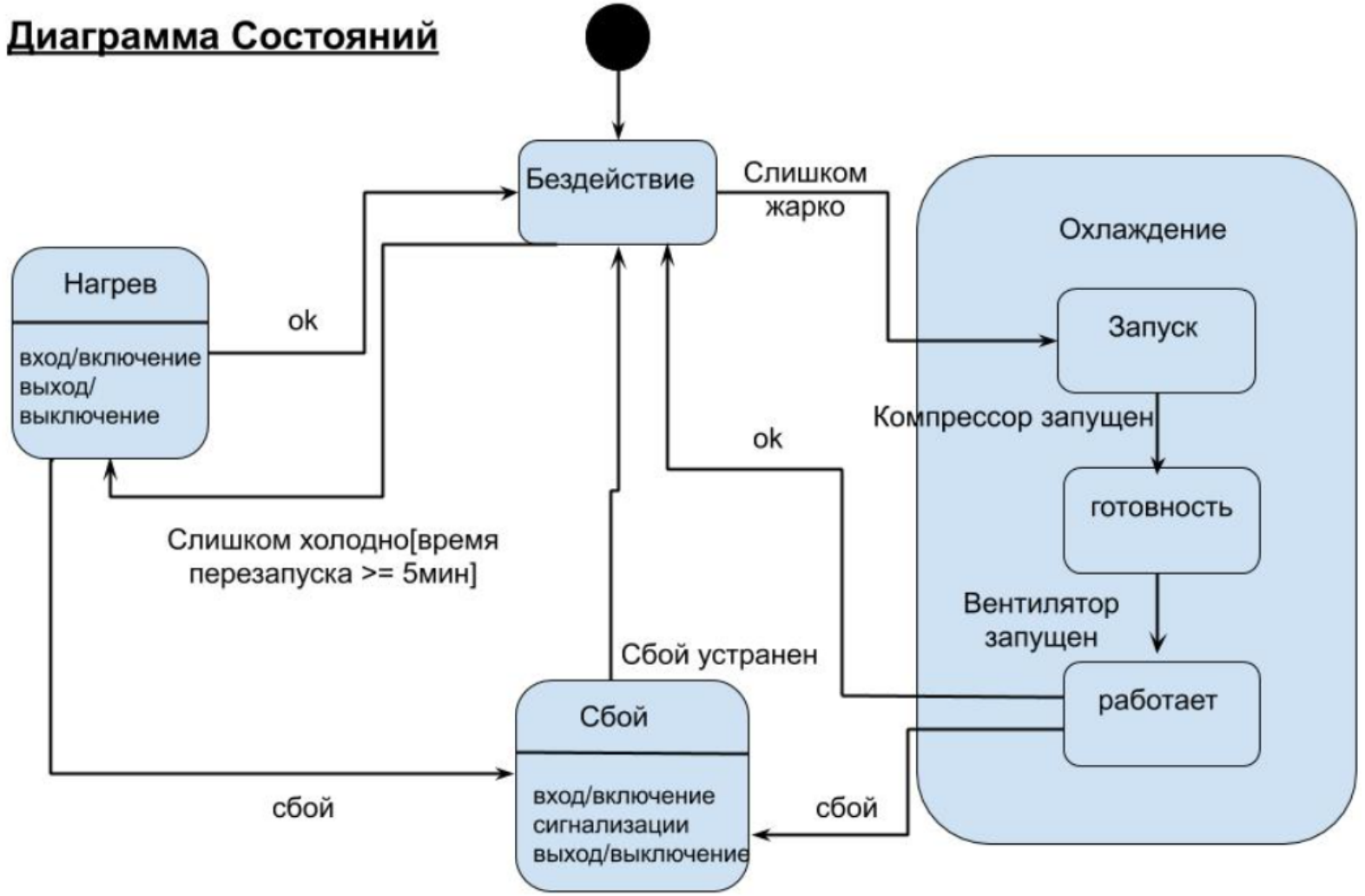
# Диаграмма деятельности

- Диаграммы деятельности представляют собой графическое представление рабочих процессов поэтапных действий и действий с поддержкой выбора, итерации и параллелизма.
- Они описывают поток управления целевой системой, такой как исследование сложных бизнес-правил и операций, а также описание прецедентов и бизнес-процессов.
- В UML диаграммы деятельности предназначены для моделирования как вычислительных, так и организационных процессов.



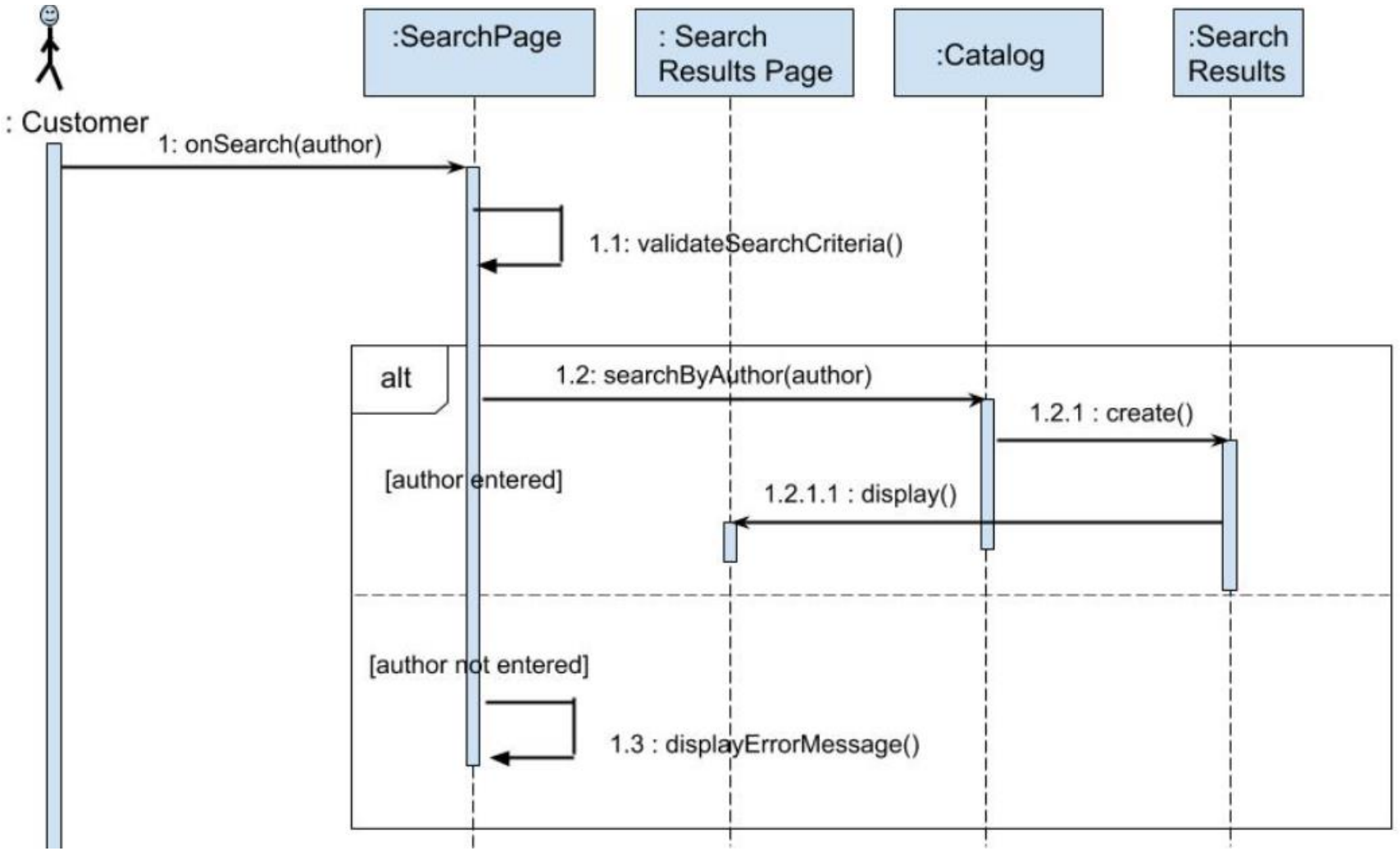
# Диаграмма состояний

- Диаграмма состояний — это тип диаграммы, используемый в UML для описания поведения систем, который основан на концепции диаграмм состояний Дэвида Харела.
- Диаграммы состояний отображают разрешенные состояния и переходы, а также события, которые влияют на эти переходы.
- Она помогает визуализировать весь жизненный цикл объектов и, таким образом, помогает лучше понять системы, основанные на состоянии.



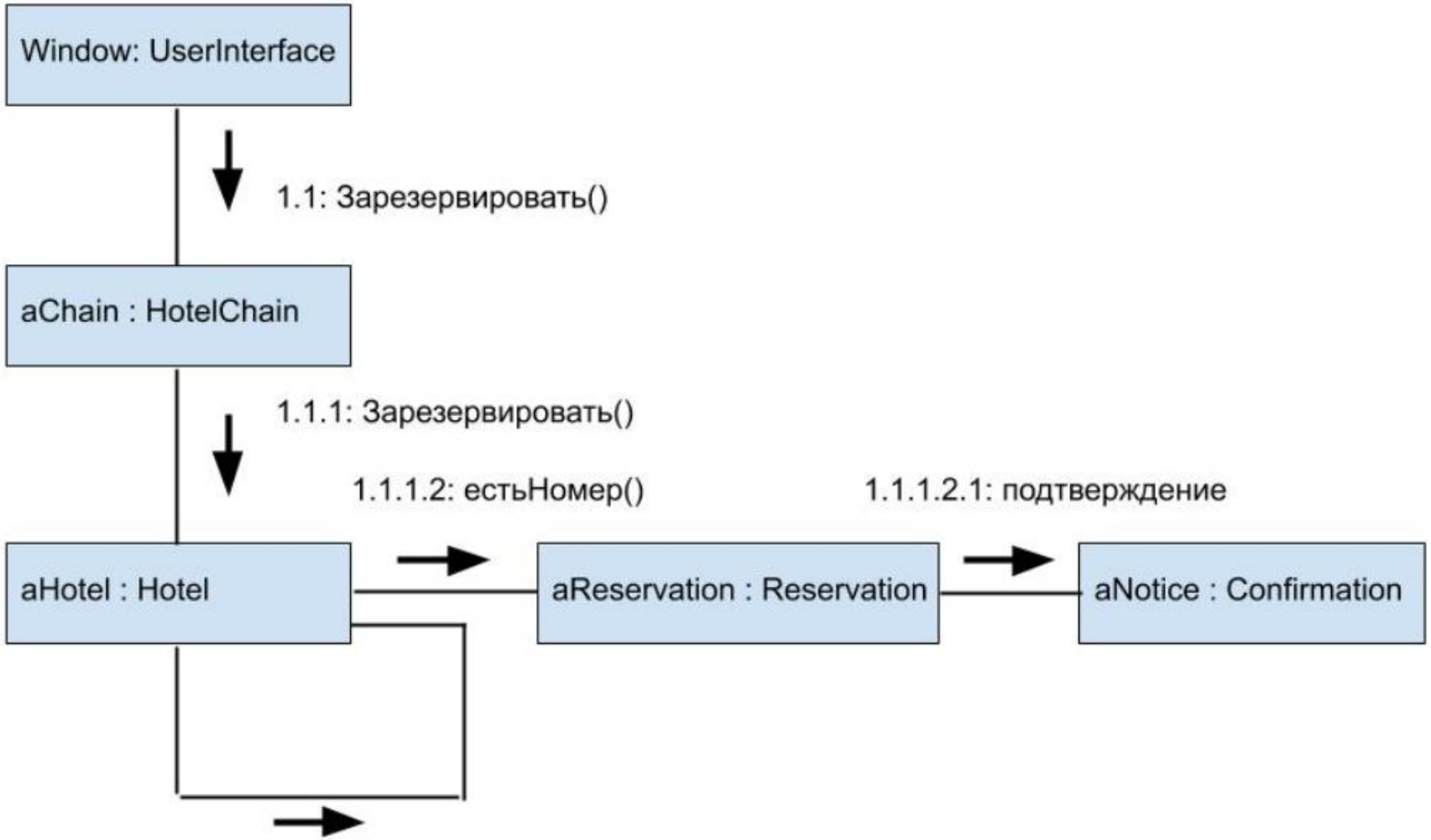
# Диаграмма последовательности

- Диаграмма последовательности моделирует взаимодействие объектов на основе временной последовательности.
- Она показывает, как одни объекты взаимодействуют с другими в конкретном прецеденте.



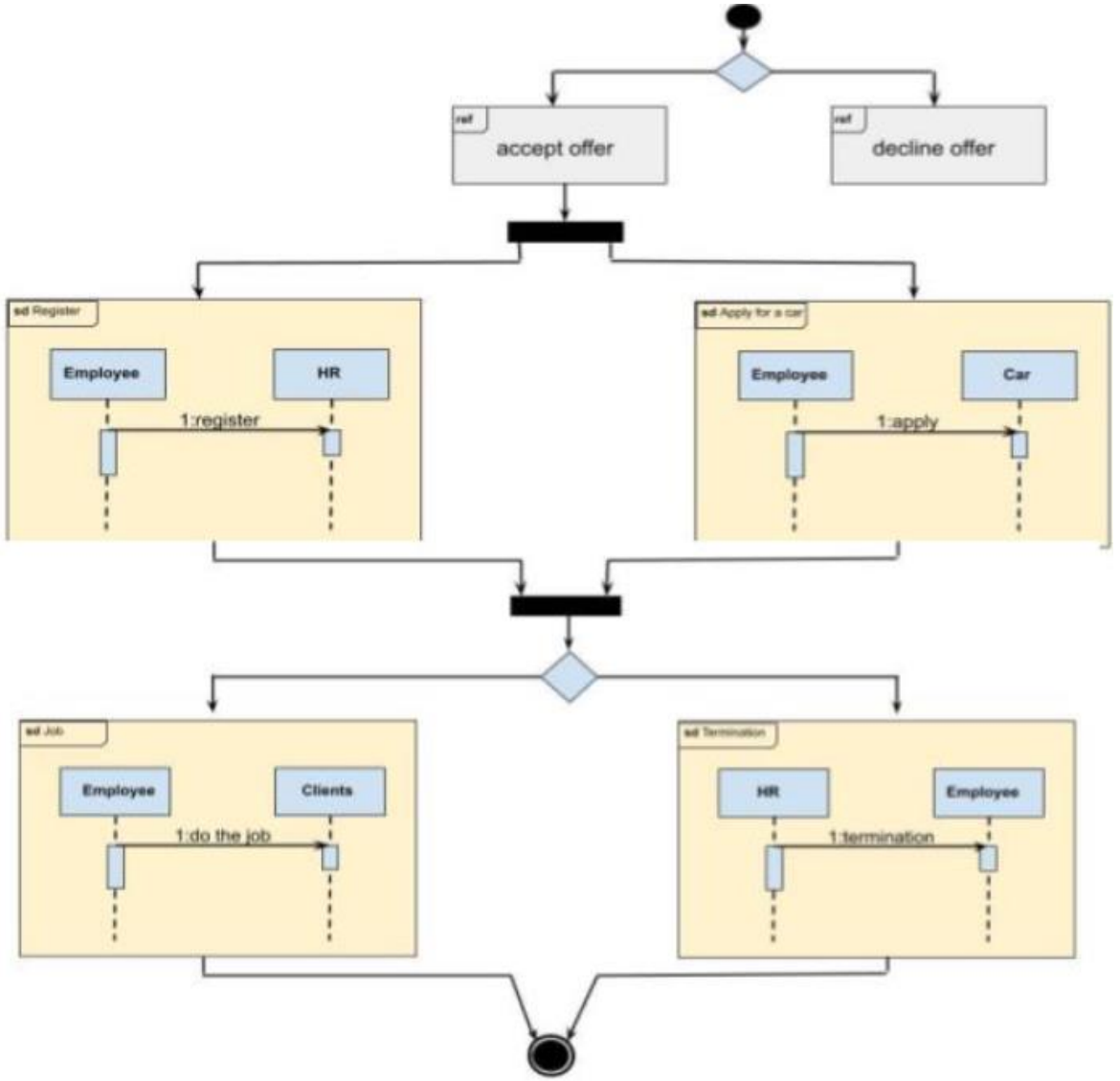
# Диаграмма Коммуникации

- Как и диаграмма последовательности, диаграмма коммуникации также используется для моделирования динамического поведения прецедента.
- Если сравнивать с Диаграммой последовательности, Диаграмма коммуникации больше сфокусирована на показе взаимодействия объектов, а не временной последовательности.
- На самом деле, диаграмма коммуникации и диаграмма последовательности семантически эквивалентны и могут перетекать одна в другую.



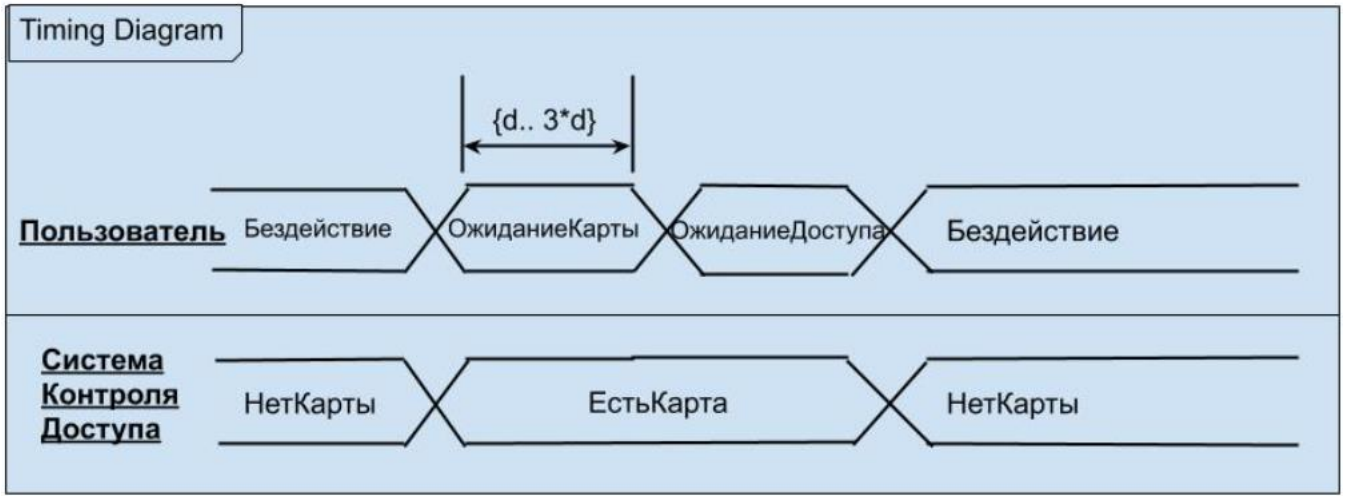
# Диаграмма обзора взаимодействия

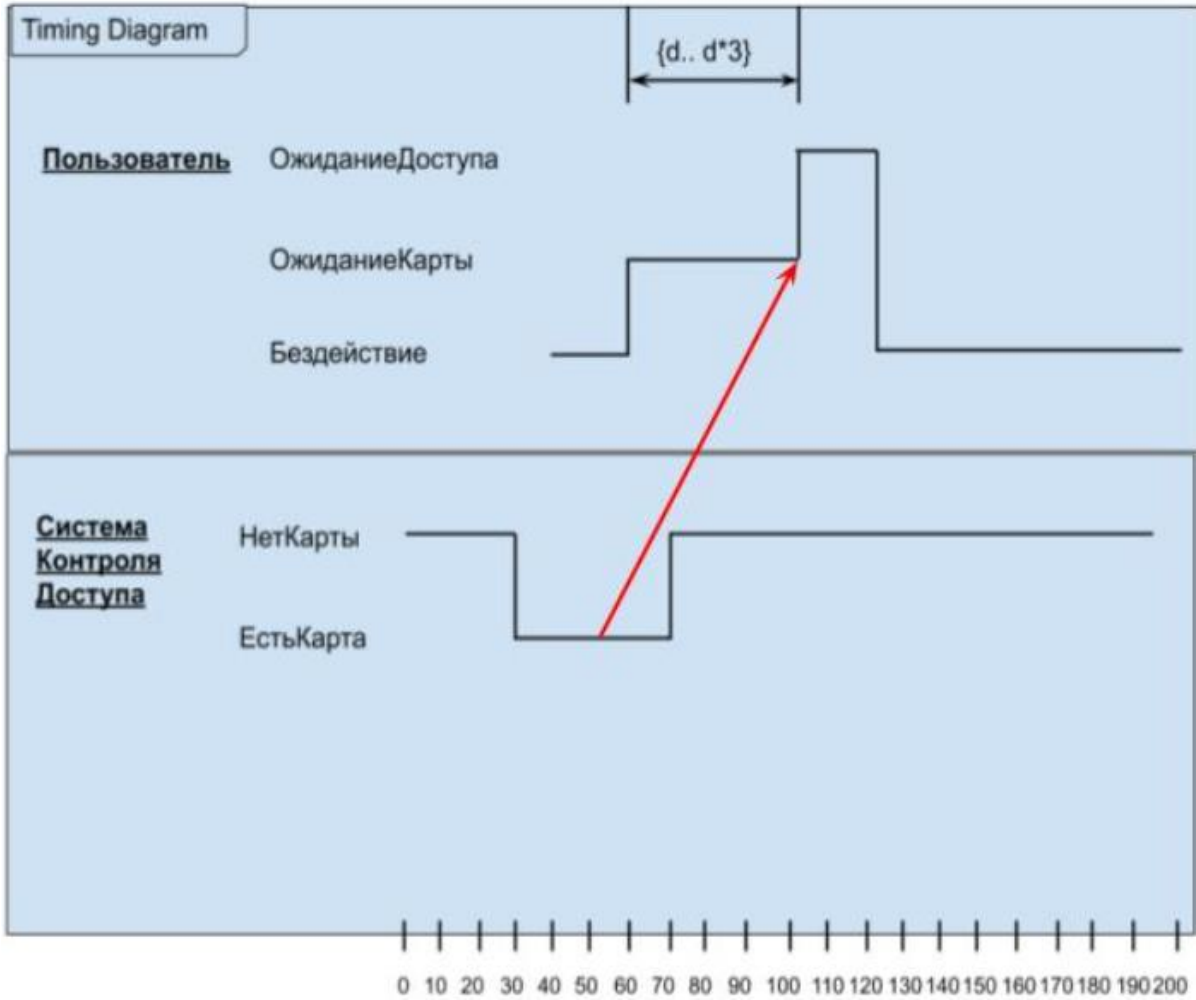
- Диаграмма обзора взаимодействий фокусируется на обзоре потока управления взаимодействиями.
- Это вариант Диаграммы деятельности, где узлами являются взаимодействия или события взаимодействия.
- Диаграмма обзора взаимодействий описывает взаимодействия, в которых сообщения и линии жизни скрыты.
- Мы можем связать «реальные» диаграммы и добиться высокой степени навигации между диаграммами внутри диаграммы обзора взаимодействия.



# Временная диаграмма

- Временная диаграмма показывает поведение объекта (ов) в данный период времени.
- По сути — это особая форма диаграммы последовательности и различия между ними состоят в том, что оси меняются местами так, что время увеличивается слева направо, а линии жизни отображаются в отдельных отсеках, расположенных вертикально.



1. 14 UML диаграмм за 10 минут – YouTube
2. UML Диаграмма Классов (UML Class Diagram) – YouTube
3. UML Диаграмма Компонентов (UML Component Diagram) – YouTube
4. UML Диаграмма Объектов (UML Object Diagram) – YouTube
5. UML Диаграмма Развертывания (UML Deployment Diagram) – YouTube
6. UML Диаграмма Пакетов (UML Package Diagram) - YouTube
7. Диаграмма Составных Структур UML (UML Composite Structure Diagram) – YouTube
8. UML Диаграмма Профилей (UML Profile Diagram) - YouTube
9. UML Диаграмма Прецедентов (UML Use Case Diagrams) – YouTube
10. UML Диаграмма Обзора Взаимодействий (UML Interaction Overview Diagram) – YouTube
11. UML Диаграмма Деятельности (UML Activity Diagram) – YouTube
12. UML Временная Диаграмма (UML Timing Diagram) – YouTube
13. UML Диаграмма Состояний (UML State Diagram) – YouTube
14. UML Диаграмма Коммуникации (UML Communication Diagram) – YouTube
15. UMl Диаграмма Последовательности (Sequence Diagram) – YouTube

## General Areas Tested in the UML 2 Foundation Exam

| | |
|---|---|
| Class Diagram | 25% |
| Activity Diagram | 20% |
| Sequence Diagram | 15% |
| Why We Model | 15% |
| State Machine Diagram | 10% |
| Object Diagram | 5% |
| Package Diagram | 5% |
| Use Case Diagram | 5% |
| Total | **100%** |


Требования к системе
Анализ
Проектирование
Код
Диаграмма прецедентов
Диаграмма компонентов
Диаграмма классов
Диаграмма состояний
Диаграмма деятельностей
Диаграмма последовательностей
Исходные коды
Диаграмма кооперации
Диаграмма развертывания

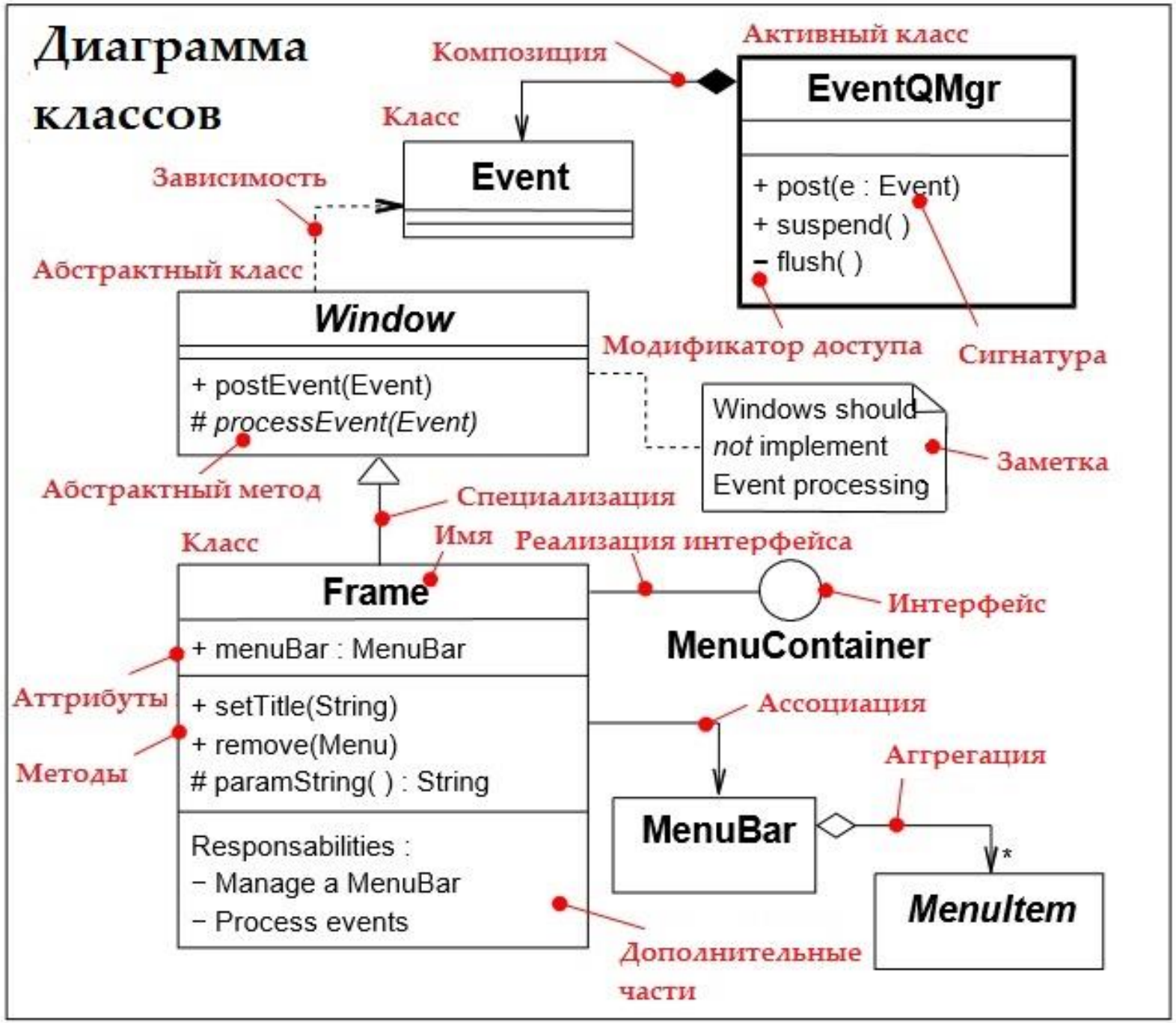


Диаграмма классов
Композиция
Активный класс
EventQMgr
+ post(e : Event)
+ suspend( )
– flush( )
Класс
Event
Зависимость
Абстрактный класс
Window
+ postEvent(Event)
# processEvent(Event)
Модификатор доступа
Сигнатура
Windows should
not implement
Event processing
Заметка
Абстрактный метод
Специализация
Класс
Имя
Реализация интерфейса
Frame
+ menuBar : MenuBar
+ setTitle(String)
+ remove(Menu)
# paramString( ) : String
Responsabilities :
– Manage a MenuBar
– Process events
Интерфейс
MenuContainer
Аттрибуты
Методы
Ассоциация
Аггрегация
MenuBar
*
MenuItem
Дополнительные части