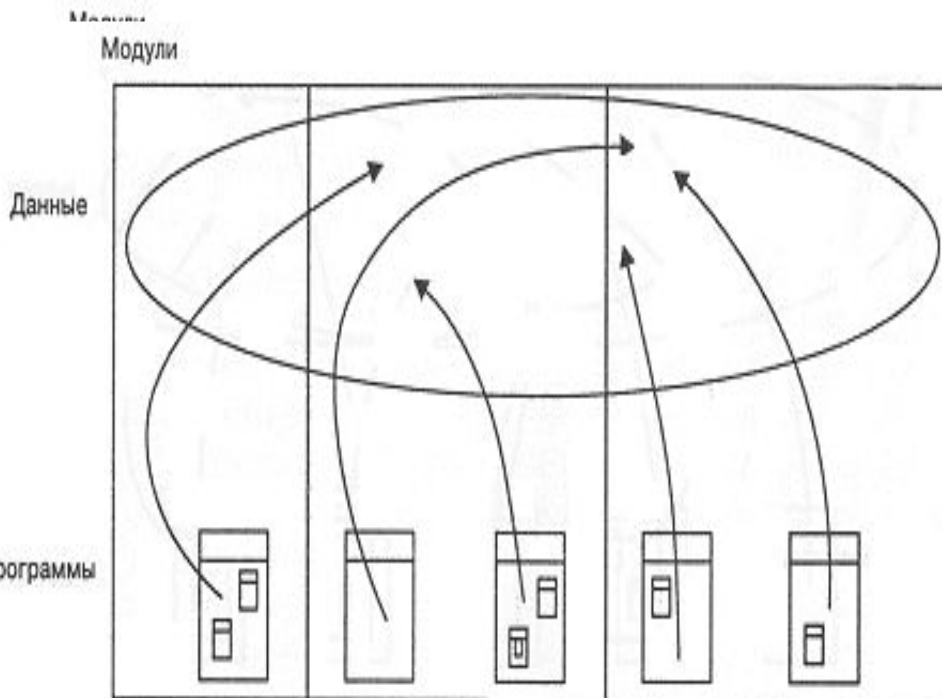
ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

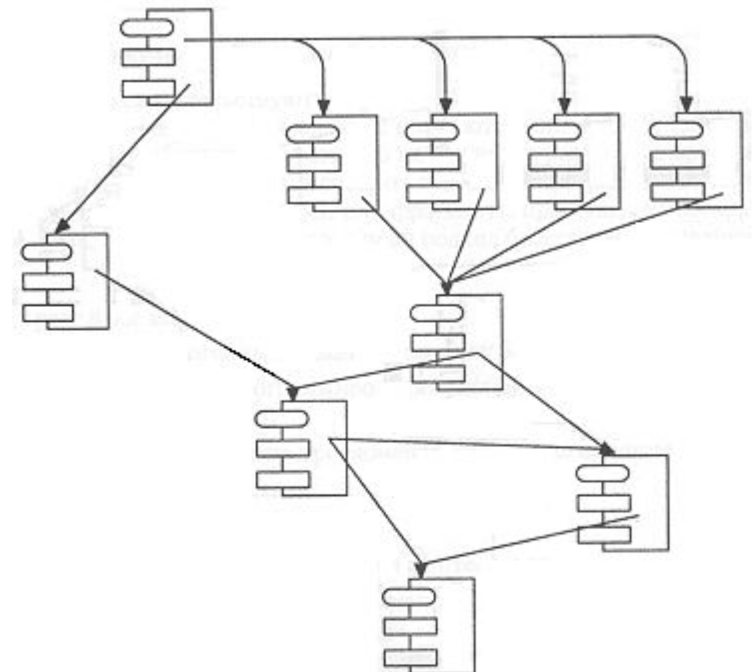
Объектно-ориентированное программирование – это мощная и естественная парадигма для создания программ, которые переживают неизбежные изменения, сопровождающие жизненный цикл любого большого программного проекта на всех этапах этого цикла.

ЭЛЕМЕНТЫ, СОСТАВЛЯЮЩИЕ ПРОГРАММЫ

Все компьютерные программы состоят из двух элементов ***кода*** и ***данных.***
В записи программы на исходном языке под ***кодом*** понимают набор исполняемых утверждений, определяющих алгоритм обработки данных, а под ***данными*** – описатели переменных, используемых в этом алгоритме.

Любая программа может быть концептуально организована либо ***вокруг ее кода,*** либо ***вокруг ее данных.***





ОСНОВНЫЕ ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ

Существуют две парадигмы (основополагающих подхода), которые управляют проектированием программ.

Первый подход называет программу моделью, которая ориентированна на процесс. Здесь программу определяют последовательности операторов ее кода. ***Модель, ориентированную на процесс, можно представить как кодовое воздействие на данные.*** Процедурные языки программирования такие как С++ или Паскаль успешно эксплуатируют такую модель. Недостатком этого подхода является размер и сложность программ.

Второй подход, названный **объектно – ориентированным программированием** был задуман для управления возрастающей сложностью программ. Объектно – ориентированное программирование организует программу вокруг своих данных, то есть вокруг объектов и набора хорошо определенных интерфейсов (взаимодействий) с данными. ***Объектно – ориентированную программу можно характеризовать как управляемый данными доступ к коду.*** Это дает некоторые организационные преимущества.

Три принципа объектно-ориентированного подхода

1. Инкапсуляция

Инкапсуляция (encapsulation) - это механизм, который объединяет данные и код, манипулирующий этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования.

Цель инкапсуляции предотвратить нежелательные побочные эффекты.

В объектно-ориентированном программировании код и данные могут быть объединены вместе; в этом случае говорят, что создаётся так называемый "чёрный ящик". Когда коды и данные объединяются таким способом, создаётся объект (object). Другими словами, объект - это то, что поддерживает инкапсуляцию.

2. Полиморфизм

Полиморфизм (polymorphism) (от греческого polymorphos) - это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач.

Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных.

Пример

В языке Си полиморфизм поддерживается недостаточно. Нахождение абсолютной величины числа требует трёх различных функций: abs(), labs() и fabs(). Эти функции подсчитывают и возвращают абсолютную величину целых, длинных целых и чисел с плавающей точкой соответственно.

В C++ каждая из этих функций может быть названа abs(). Тип данных, который используется при вызове функции, определяет, какая конкретная версия функции действительно выполняется. В C++ можно использовать одно имя функции для множества различных действий. Это называется перегрузкой функций (function overloading).

3. Наследование

Наследование (inheritance) - это процесс, посредством которого один объект может приобретать свойства другого. Точнее, объект может наследовать основные свойства другого объекта и добавлять к ним черты, характерные только для него.

Цель наследования сделать управляемыми большие потоки информации.

Наследование является важным, поскольку оно позволяет поддерживать концепцию иерархии классов (hierarchical classification).

Порождённый класс наследует все, связанные с родителем, качества и добавляет к ним свои собственные определяющие характеристики. Без использования иерархии классов, для каждого объекта пришлось бы задать все характеристики, которые бы исчерпывающи его определяли. Однако при использовании наследования можно описать объект путём определения того общего класса (или классов), к которому он относится, с теми специальными чертами, которые делают объект уникальным.

Наследование играет очень важную роль в объектно-ориентированном программировании.

Объектно-ориентированная терминология

Термин	*Определение*
Метод	Функция, входящая в состав объекта
Объект	Сущность, наделенная определенными обязанностями. Особая, самодостаточная структура, содержащая как данные, так и методы, манипулирующие этими данными. Данные объекта защищены от доступа со стороны внешних объектов
Свойство	Данные, принадлежащие объекту (данные-члены класса)
Класс	«Проект» объекта содержит описание методов и данных объектов соответствующего типа
Суперкласс	Класс, от которого происходят другие классы. Содержит базовые определения свойств и методов, которые будут использоваться всеми классоми-потомками (возможно с переопределением)
Абстрактный класс	Определяет методы и общие свойства некоторого множества классов, подобных друг другу на концептуальном уровне. Реализация абстрактных классов невозможна
Порожденный класс	Класс, который является специализацией своего суперкласса. Включает все свойства и методы суперкласса, но может также включать собственные свойства и иную реализацию методов суперкласса
Экземпляр	Конкретный объект, относящийся к некоторому классу
Конструктор	Специальный метод, который вызывается при создании экземпляра объекта
Деструктор	Специальный метод, который вызывается при удалении объекта
Реализация	Процесс создания экземпляра класса
Функциональная декомпозиция	Один из методов анализа, при котором поставленная задача разбивается на более мелкие подзадачи-функции

УРОВНИ ВИДИМОСТИ И ОБЛАСТИ ДЕЙСТВИЯ КЛАССА

Видимость свойства указывает на возможность его использования другими *классами*. Один *класс* может «видеть» другой, если тот находится в области действия первого и между ними существует явное или неявное отношение.

В языке *UML* определены три уровня видимости:

- **public** (общий) — любой внешний *класс*, который «видит» данный, может пользоваться его общими свойствами. Обозначаются знаком «+» перед именем атрибута или операции;
- **protected** (защищенный) — только любой потомок данного *класса* может пользоваться его защищнными свойствами. Обозначаются знаком «#»;
- **private** (закрытый) — только данный *класс* может пользоваться этими свойствами. Обозначаются символом «**-**» .

Еще одной важной характеристикой атрибутов и операций *классов* является область действия. Область действия свойства указывает, будет ли оно проявлять себя по-разному в каждом экземпляре *класса*, или одно и то же значение свойства будет совместно использоваться всеми экземплярами:

- **instance** (экземпляр) — у каждого экземпляра *класса* есть собственное значение данного свойства;
- **classifier** (классификатор) — все экземпляры совместно используют общее значение данного свойства (выделяется на диаграммах подчеркиванием).

УНИФИЦИРОВАННЫЙ ЯЗЫК ВИЗУАЛЬНОГО МОДЕЛИРОВАНИЯ
Unified Modeling Language (UML)

UML представляет собой набор соглашений, которые предназначены для облегчения процесса моделирования и обмена информацией в проектной группе и заказчиками. Наличие стандартизированной нотации позволяет сократить время на усвоение информации, упрощает общение и взаимодействие, облегчает документирование.

UML представляет собой графическую нотацию которая предназначена для моделирования и описания всех процессов протекающих в процессе разработки. Основу **UML** представляют диаграммы, которые различаются по типам и предназначены для моделирования различных аспектов разработки.

UML является средством достижения компромисса между различными подходами. Существует достаточное количество инструментальных средств, поддерживающих с помощью *UML* жизненный цикл информационных систем, и, одновременно, *UML* является достаточно гибким для настройки и поддержки специфики деятельности различных команд разработчиков.

СОЗДАНИЕ UML И КОНСОРЦИУМ ЕГО ПОЛЬЗОВАТЕЛЕЙ

Создание UML началось в октябре 1994 г., когда Джим Рамбо и Гради Буч из Rational Software Corporation стали работать над объединением своих методов OMT и Booch. Осенью 1995 г. увидела свет первая черновая версия объединенной методологии, которую они назвали Unified Method 0.8. После присоединения в конце 1995 г. к Rational Software Corporation Айвара Якобсона и его фирмы Objectory, усилия трех создателей наиболее распространенных объектно-ориентированных методологий были объединены и направлены на создание UML.

В настоящее время консорциум пользователей UML Partners включает в себя представителей таких грандов информационных технологий, как

- Rational Software,
- Microsoft,
- IBM,
- Hewlett-Packard,
- Oracle,
- DEC,
- Unisys,
- IntelliCorp,
- Platinum Technology.

ОСНОВНЫЕ ХАРАКТЕРИСТИКИ UML

UML представляет собой *объектно-ориентированный* язык моделирования, обладающий следующими основными характеристиками:

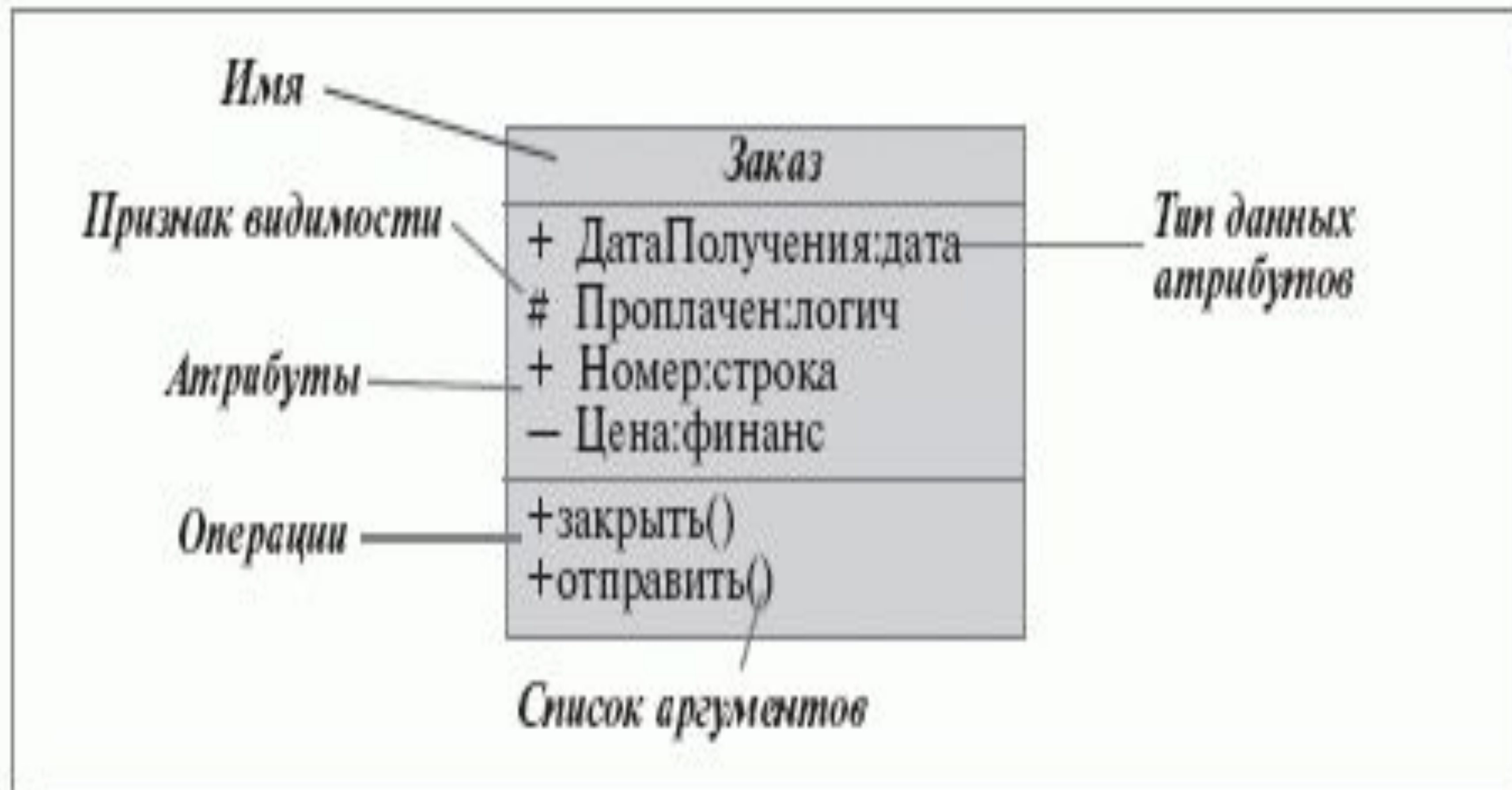
- является языком визуального моделирования, который обеспечивает разработку репрезентативных моделей для организации взаимодействия заказчика и разработчика ИС, различных групп разработчиков ИС;
- содержит механизмы расширения и специализации базовых концепций языка.

UML — это стандартная нотация визуального моделирования программных систем, принятая консорциумом Object Managing Group (OMG) осенью 1997 г., и на сегодняшний день она поддерживается многими объектно-ориентированными CASE-продуктами.

UML включает внутренний набор средств моделирования (модулей?) («ядро»), которые сейчас приняты во многих методах и средствах моделирования. Эти концепции необходимы в большинстве прикладных задач, хотя не каждая концепция необходима в каждой части каждого приложения. Пользователям языка предоставлены возможности:

- строить модели на основе средств ядра, без использования механизмов расширения для большинства типовых приложений;
- добавлять при необходимости новые элементы и условные обозначения, если они не входят в ядро, или специализировать компоненты, систему условных обозначений (нотацию) и ограничения для конкретных предметных областей.

ИЗОБРАЖЕНИЕ КЛАССА В UML



РАЗНОВИДНОСТИ КЛАССОВ

Возможное количество экземпляров *класса* называется его кратностью. В *UML* можно определять следующие разновидности *классов*:

- не содержащие ни одного экземпляра — тогда *класс* становится служебным (Abstract);
- содержащие ровно один экземпляр (Singleton);
- содержащие заданное число экземпляров;
- содержащие произвольное число экземпляров.

Принципиальное назначение *классов* характеризуют стереотипы. Это, фактически классификация объектов на высоком уровне, позволяющая определить некоторые основные свойства объекта (пример стереотипа — *класс* «действующее лицо»).

Механизм стереотипов является также средством расширения словаря *UML* за счет создания на основе существующих блоков языка новых, специфичных для решения конкретной проблемы.

ДИАГРАММЫ КЛАССОВ

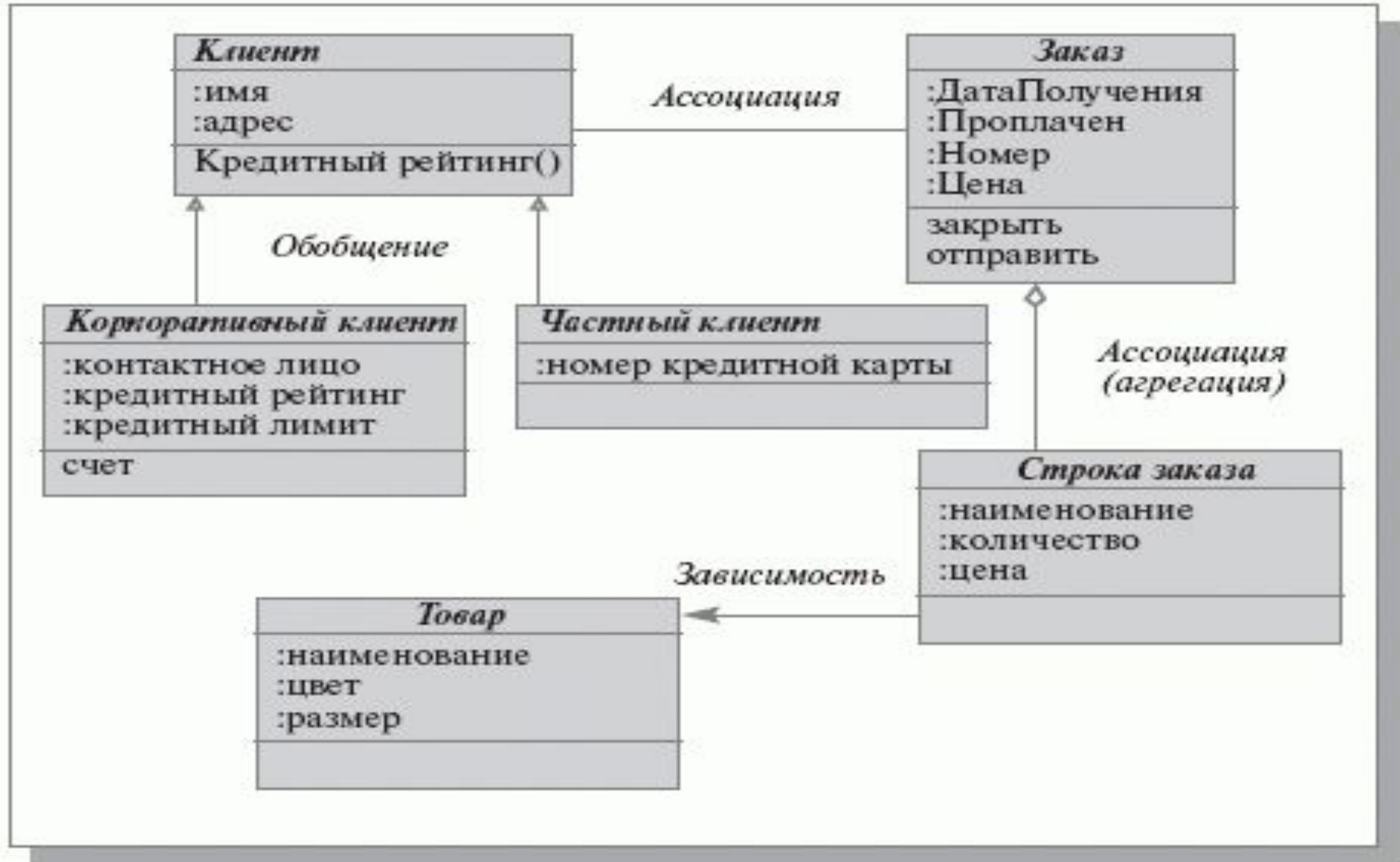
Классы в *UML* изображаются на ***диаграммах классов***, которые позволяют описать систему в статическом состоянии — определить типы объектов системы и различного рода статические связи между ними.

Классы отображают типы объектов системы.

Между *классами* возможны различные отношения

- зависимости, которые описывают существующие между *классами* отношения использования;
- обобщения, связывающие обобщенные *классы* со специализированными;
- ассоциации, отражающие структурные отношения между объектами *классов.*

ОТОБРАЖЕНИЕ СВЯЗЕЙ МЕЖДУ КЛАССАМИ НА ДИАГРАММЕ КЛАССОВ



Отношения между классами

***Обобщение* is-a –** один класс является подвидом другого класса. (На диаграмме классов в языке UML обозначается прозрачным треугольником).

При существующей взаимосвязи между классами:

***Ассоциация* has-a** – один класс «содержит» другой класс;
Один объект является частью другого объекта –***объединение*** (composition).
Например, двигатель в автомобиле.
(На диаграмме классов в языке UML обозначается закрашенным ромбом).
Имеется набор (коллекция) объектов, которые могут существовать сами по себе - ***агрегация***(aggregation).
Например, самолеты в аэропорту.
(На диаграмме классов в языке UML обозначается прозрачным ромбом).

***Зависимость* uses -** один класс «использует» другой класс. *Например*, автомобиль использует заправку.
(На диаграмме классов в языке UML обозначается пунктирной стрелкой).

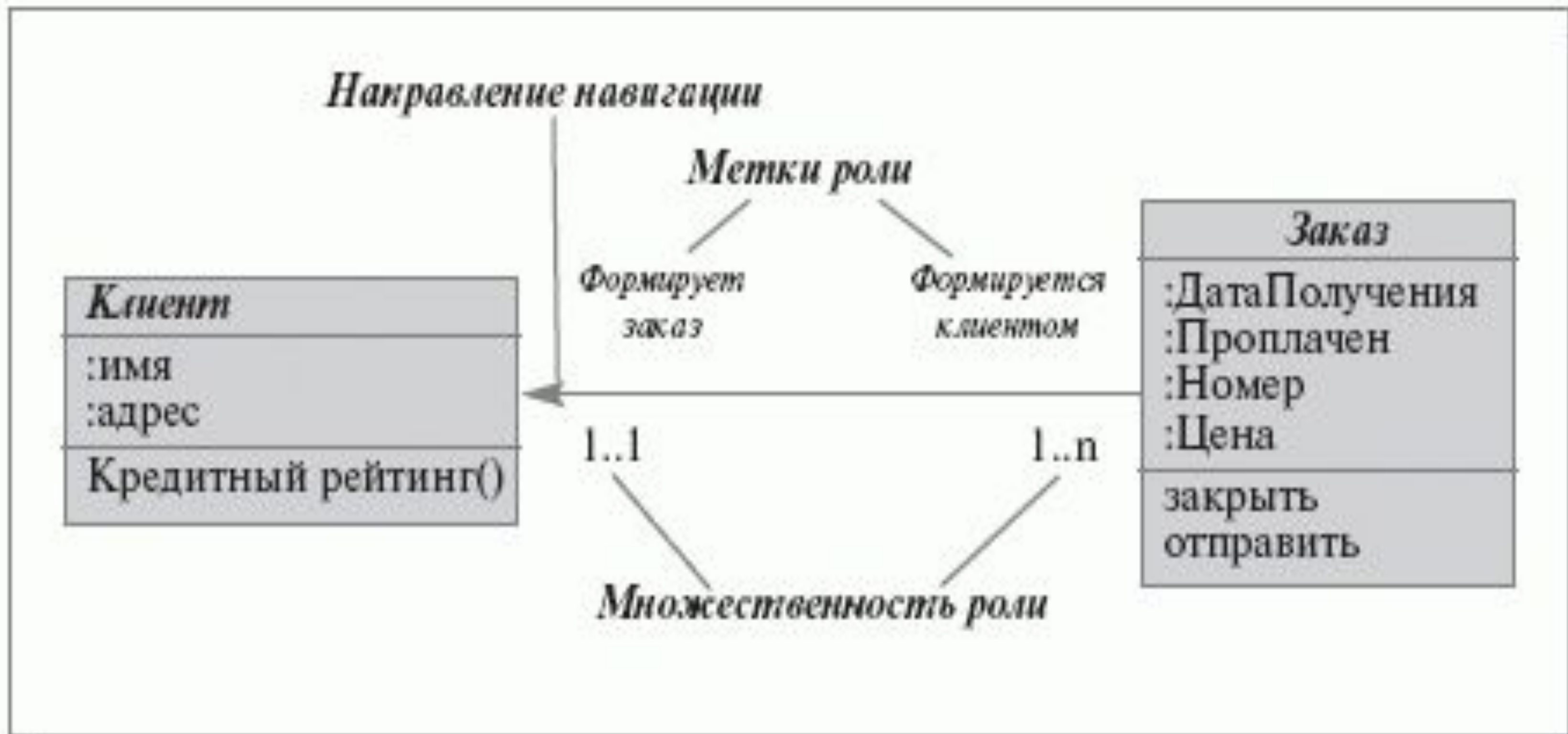
ОТНОШЕНИЯ МЕЖДУ КЛАССАМИ

Зависимостью называется отношение использования, согласно которому изменение в спецификации одного элемента (например, *класса* «товар») может повлиять на использующий его элемент (*класс* «строка заказа»). Часто зависимости показывают, что один *класс* использует другой в качестве аргумента.

Обобщение — это отношение между общей сущностью (родителем — *класс* «клиент») и ее конкретным воплощением (потомком — *классы* «корпоративный клиент» или «частный клиент»). Объекты *класса*-потомка могут использоваться всюду, где встречаются объекты *класса*-родителя, но не наоборот. При этом он наследует свойства родителя (его атрибуты и операции). Операция потомка с той же сигнатурой, что и у родителя, замещает операцию родителя; это свойство называют полиморфизмом. *Класс*, у которого нет родителей, но есть потомки, называется корневым. *Класс*, у которого нет потомков, называется листовым.

Ассоциация — это отношение, показывающее, что объекты одного типа неким образом связаны с объектами другого типа («клиент» может сделать «заказ»). Если между двумя *классами* определена ассоциация, то можно перемещаться от объектов одного *класса* к объектам другого. При необходимости направление навигации может задаваться стрелкой. Допускается задание ассоциаций на одном *классе*. В этом случае оба конца ассоциации относятся к одному и тому же *классу*. Это означает, что с объектом некоторого *класса* можно связать другие объекты из того же *класса*. Ассоциации может быть присвоено имя, описывающее семантику отношений. Каждая ассоциация имеет две роли, которые могут быть отражены на диаграмме. Роль ассоциации обладает свойством множественности, которое показывает, сколько соответствующих объектов может участвовать в данной связи.

СВОЙСТВА АССОЦИАЦИИ



Каждый заказ может быть создан единственным клиентом (множественность роли 1.1). Каждый клиент может создать один и более заказов (множественность роли 1..n). Направление навигации показывает, что каждый заказ должен быть «привязан» к определенному клиенту.

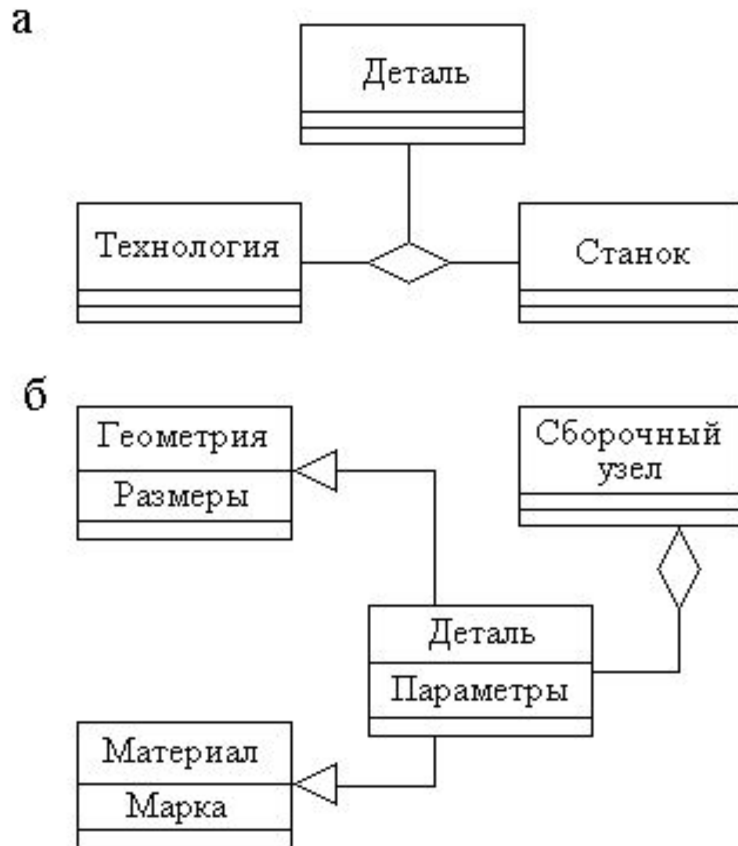
ПРОСТЫЕ И АГРЕГИРОВАННЫЕ АССОЦИАЦИИ

Ассоциация может быть простой и отражать отношение между равноправными сущностями, когда оба *класса* находятся на одном концептуальном уровне и ни один не является более важным, чем другой.

Если приходится моделировать отношение типа «часть-целое», то используется специальный тип ассоциации — агрегирование. В такой ассоциации один из *классов* имеет более высокий ранг (целое — *класс* «заказ») и состоит из нескольких меньших по рангу *классов* (частей — *класс* «строка заказа»).

В *UML* используется и более сильная разновидность агрегации — композиция, в которой объект-часть может принадлежать только единственному целому. В композиции жизненный цикл частей и целого совпадают, любое удаление целого обязательно захватывает и его части.

Отношения тернарной ассоциации (а), агрегирования и наследования (б) в диаграммах классов



Частные случаи ассоциаций — обобщение и агрегирование. Отношение обобщения (наследования) изображают сплошной линией, заканчивающейся незакрашенной стрелкой около родительского элемента. Отношение *агрегирования* (отношение "часть — целое") показывают такой же линией, но с ромбовидной стрелкой, заканчивающейся у элемента "целое". Ромбовидная стрелка закрашивается, если части не могут существовать без целого, т.е. если при ликвидации класса "целое" ликвидируются и все его "части". Отметим, что в этом случае отношение агрегирования иногда называют отношением композиции.

ДИАГРАММЫ ИСПОЛЬЗОВАНИЯ

Диаграммы использования описывают функциональность ИС, которая будет видна пользователям системы. «Каждая функциональность» изображается в виде «прецедентов использования» (use case) или просто прецедентов.

Прецедент — это типичное взаимодействие пользователя с системой, которое при этом:

- описывает видимую пользователем функцию,
- может представлять различные уровни детализации,
- обеспечивает достижение конкретной цели, важной для пользователя.

Прецедент обозначается на диаграмме овалом, связанным с пользователями, которых принято называть действующими лицами (актерами, actors). Действующие лица используют систему (или используются системой) в данном прецеденте. Действующее лицо выполняет некоторую роль в данном прецеденте. На диаграмме изображается только одно действующее лицо, однако реальных пользователей, выступающих в данной роли по отношению к ИС, может быть много. Список всех прецедентов фактически определяет функциональные требования к ИС, которые лежат в основе разработки технического задания на создание системы.

ДИАГРАММА ПРЕЦЕДЕНТОВ (диаграммы использования)

На ***диаграммах прецедентов***, кроме связей между действующими лицами и прецедентами, возможно использование еще двух видов связей между прецедентами: «использование» и «расширение» . Связь типа «расширение» применяется, когда один прецедент подобен другому, но несет несколько большую функциональную нагрузку. Ее следует применять при описании изменений в нормальном поведении системы. Связь типа «использование» позволяет выделить некий фрагмент поведения системы и включать его в различные прецеденты без повторного описания.

Диаграммы взаимодействий, последовательностей и кооперации

Диаграмма взаимодействий (Interaction diagram) описывает взаимодействия, состоящие из множества объектов и отношений между ними, включая сообщения, которыми они обмениваются.

Диаграммой последовательностей (Sequence diagram) называется диаграмма взаимодействий, акцентирующая внимание на временной упорядоченности сообщений. Графически такая диаграмма представляет собой таблицу, объекты в которой располагаются вдоль оси X, а сообщения в порядке возрастания времени - вдоль оси Y.

Диаграммой кооперации (Collaboration diagram) называется диаграмма взаимодействий, основное внимание в которой уделяется структурной организации объектов, принимающих и отправляющих сообщения. Графически такая диаграмма представляет собой граф из вершин и ребер. Общие свойства Поскольку диаграмма взаимодействий - это частный случай диаграммы, ей присущи общие для всех диаграмм свойства: имя и графическое содержание, являющееся одной из проекций модели. От других диаграмм ее отличает содержание.

ДИАГРАММЫ ВЗАИМОДЕЙСТВИЯ

Динамические аспекты поведения системы отражаются диаграммами взаимодействия.

В отличие от некоторых подходов объектного моделирования, когда и состояние, и поведение системы отображаются на *диаграммах классов*, *UML* отделяет описание поведения в ***диаграммы взаимодействия***.

В *UML диаграммы классов* не содержат сообщений, которые усложняют их чтение. Поток сообщений между объектами выносится на *диаграммы взаимодействия*. Как правило, *диаграмма взаимодействия* охватывает поведение объектов в рамках одного варианта использования.

Прямоугольники на диаграмме представляют различные объекты и роли, которые они имеют в системе, а линии между *классами* отображают отношения (или ассоциации) между ними. Сообщения обозначаются ярлыками возле стрелок, они могут иметь нумерацию и показывать возвращаемые значения.

Существуют два вида *диаграмм взаимодействия*: диаграммы последовательностей и кооперативные диаграммы.

ДИАГРАММЫ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

Этот вид диаграмм используется для точного определения логики сценария выполнения прецедента.

Диаграммы последовательностей отображают типы объектов, взаимодействующих при исполнении прецедентов, сообщения, которые они посылают друг другу, и любые возвращаемые значения, ассоциированные с этими сообщениями.

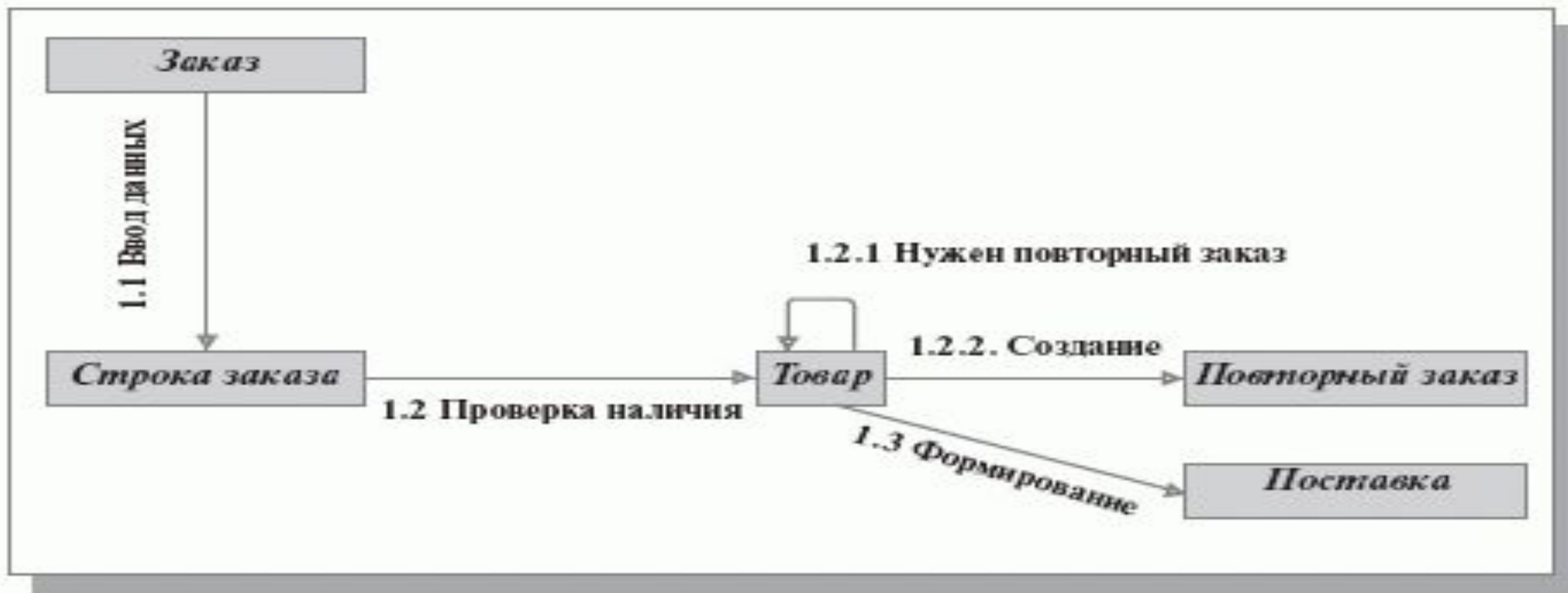
Прямоугольники на вертикальных линиях показывают «время жизни» объекта.

Линии со стрелками и надписями названий методов означают вызов метода у объекта.

ДИАГРАММЫ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

- вводятся строки заказа;
- по каждой строке проверяется наличие товара;
- если запас достаточен — инициируется поставка;
- если запас недостаточен — инициируется дозаказ (повторный заказ).

ПРИМЕР КООПЕРАТИВНОЙ ДИАГРАММЫ



На кооперативных диаграммах объекты (или *классы*) показываются в виде прямоугольников, а стрелками обозначаются сообщения, которыми они обмениваются в рамках одного варианта использования. Временная последовательность сообщений отражается их нумерацией.

Сообщения появляются в той последовательности, как они показаны на диаграмме — сверху вниз. Если предусматривается отправка сообщения объектом самому себе (самоделегирование), то стрелка начинается и заканчивается на одной «линии жизни».

ДИАГРАММЫ СОСТОЯНИЙ

Диаграммы состояний используются для описания поведения сложных систем. Они определяют все возможные состояния, в которых может находиться объект, а также процесс смены состояний объекта в результате некоторых событий. Эти диаграммы обычно используются для описания поведения одного объекта в нескольких прецедентах.

Прямоугольниками представляются состояния, через которые проходит объект во время своего поведения. Состояниям соответствуют определенные значения атрибутов объектов.

Стрелки представляют переходы от одного состояния к другому, которые вызываются выполнением некоторых функций объекта.

Имеется также два вида псевдо-состояний: начальное состояние, в котором находится только что созданный объект, и конечное состояние, которое объект не покидает, как только туда перешел.

Переходы имеют метки, которые синтаксически состоят из трех необязательных частей

<Событие> <[Условие]> < / Действие>.

На диаграммах также отображаются функции, которые выполняются объектом в определенном состоянии. Синтаксис метки деятельности:

выполнить/< деятельность >.

ПРИМЕР ДИАГРАММЫ СОСТОЯНИЙ

ДИАГРАММЫ ДЕЯТЕЛЬНОСТИ

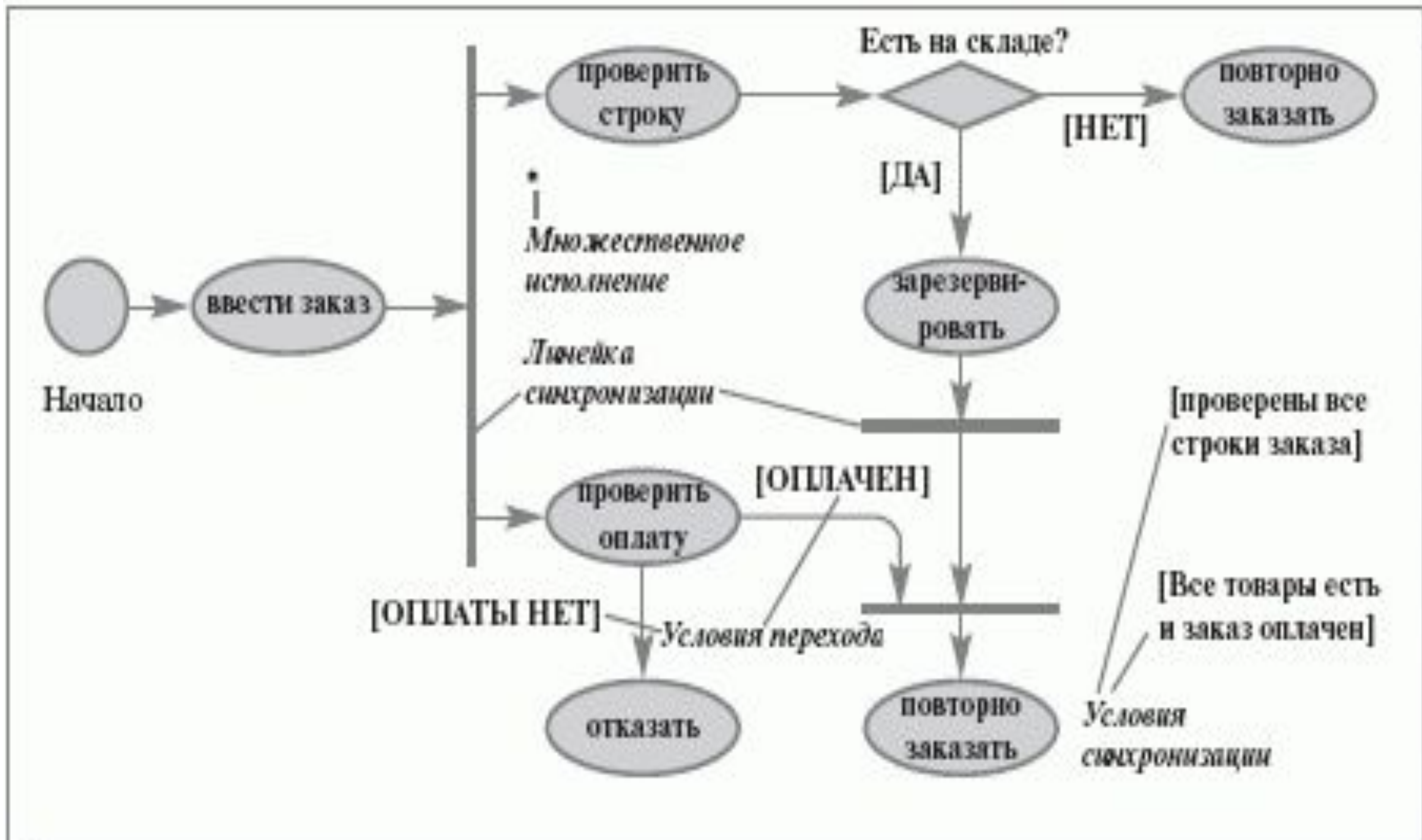
Диаграмма деятельности — это частный случай *диаграммы состояний*. На диаграмме деятельности представлены переходы потока управления от одной деятельности к другой внутри системы. Этот вид диаграмм обычно используется для описания поведения, включающего в себя множество параллельных процессов.

Основными элементами диаграмм деятельности являются:

- овалы, изображающие действия объекта;
- линейки синхронизации, указывающие на необходимость завершить или начать несколько действий (модель логического условия «И»);
- ромбы, отражающие принятие решений по выбору одного из маршрутов выполнения процесса (модель логического условия «ИЛИ»);
- стрелки — отражают последовательность действий, могут иметь метки условий.

На диаграмме деятельности могут быть представлены действия, соответствующие нескольким вариантам использования. На таких диаграммах появляется множество начальных точек, поскольку они отражают теперь реакцию системы на множество внешних событий. Таким образом, диаграммы деятельности позволяют получить полную картину поведения системы и легко оценивать влияние изменений в отдельных вариантах использования на конечное поведение системы. Любая деятельность может быть подвергнута дальнейшей декомпозиции и представлена в виде отдельной диаграммы деятельности или спецификации (словесного описания). ^

ПРИМЕР ДИАГРАММЫ ДЕЯТЕЛЬНОСТИ



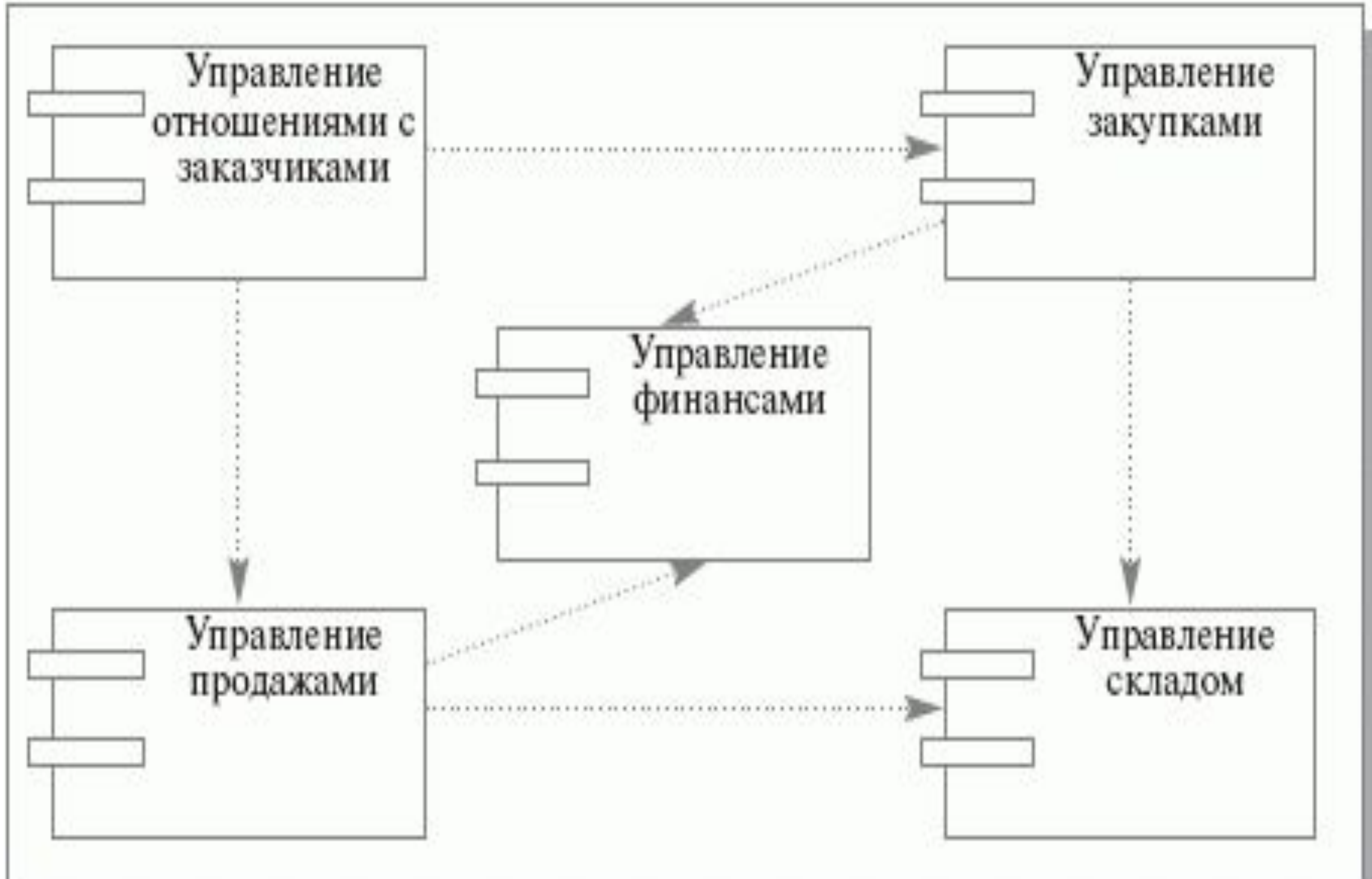
ДИАГРАММЫ КОМПОНЕНТОВ

Диаграммы компонентов позволяют изобразить модель системы на физическом уровне.
Элементами диаграммы являются компоненты — физические замещаемые модули системы. Каждый компонент является полностью независимым элементом системы. Разновидностью компонентов являются узлы. Узел — это элемент реальной (физической) системы, который существует во время функционирования программного комплекса и представляет собой вычислительный ресурс, обычно обладающий как минимум некоторым объемом памяти, а часто еще и способностью обработки. Узлы делятся на два типа:

- устройства — узлы системы, в которых данные не обрабатываются.
- процессоры — узлы системы, осуществляющие обработку данных.

Для различных типов компонентов предусмотрены соответствующие стереотипы в языке *UML*.
Компонентом может быть любой достаточно крупный модульный объект, такой как таблица или экстент базы данных, подсистема, бинарный исполняемый файл, готовая к использованию система или приложение. Таким образом, *диаграмму компонентов* можно рассматривать как *диаграмму классов* в более крупном (менее детальном) масштабе. Компонент, как правило, представляет собой физическую упаковку логических элементов, таких как *классы*, интерфейсы и кооперации.
Основное назначение *диаграмм компонентов* — разделение системы на элементы, которые имеют стабильный интерфейс и образуют единое целое. Это позволяет создать ядро системы, которое не будет меняться в ответ на

ПРИМЕР ДИАГРАММЫ КОМПОНЕНТОВ



ПАКЕТЫ UML

Пакеты представляют собой универсальный механизм организации элементов в группы.

В пакет можно поместить диаграммы различного типа и назначения.

В отличие от компонентов, существующих во время работы программы, пакеты носят чисто концептуальный характер, то есть существуют только во время разработки.

Изображается пакет в виде папки с закладкой, содержащей, как правило, только имя и иногда — описание содержимого.

Диаграмма пакетов содержит пакеты *классов* и зависимости между ними. Зависимость между двумя пакетами имеет место в том случае, если изменения в определении одного элемента влекут за собой изменения в другом. По отношению к пакетам можно использовать механизм обобщения (см. выше раздел «*Диаграммы классов*»).