Объектно-ориентированное проектирование

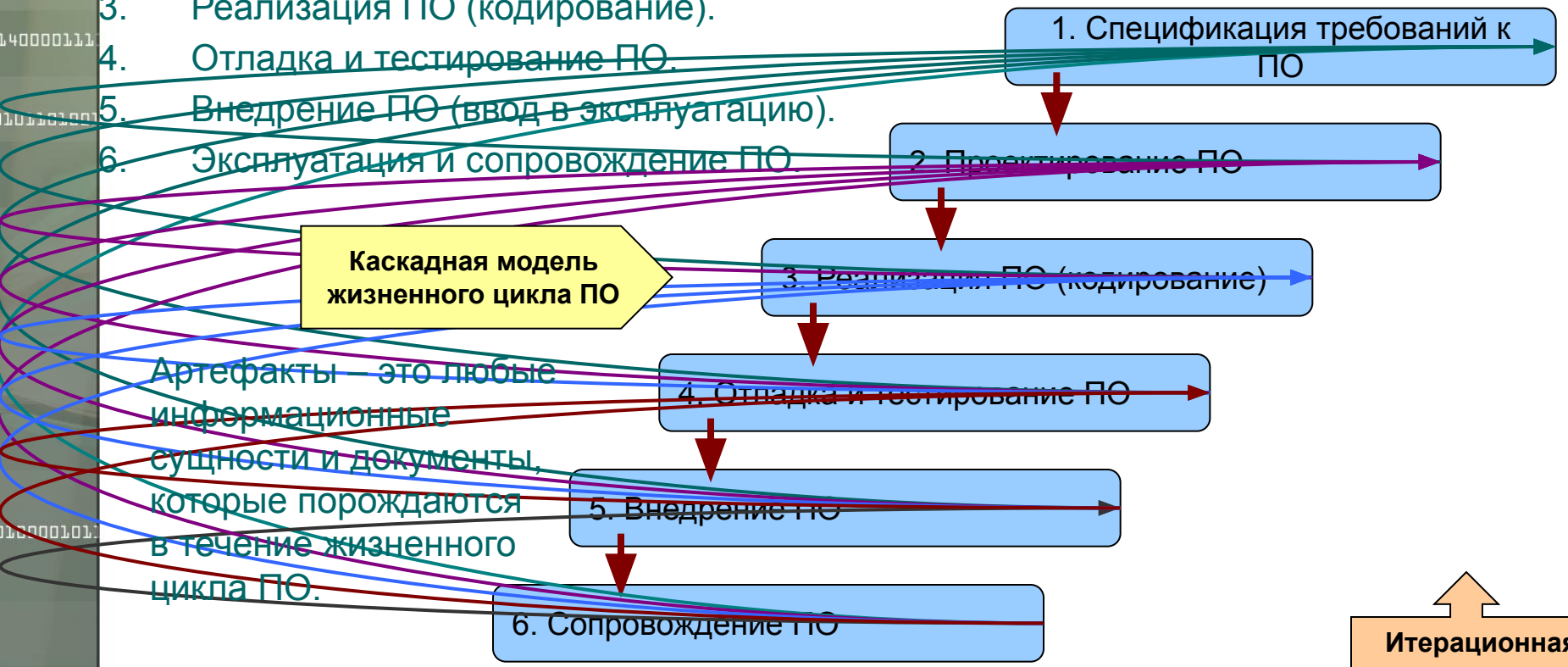
Вопросы проектирования ПО

Объектно-ориентированное проектирование

Модели жизненного цикла ПО

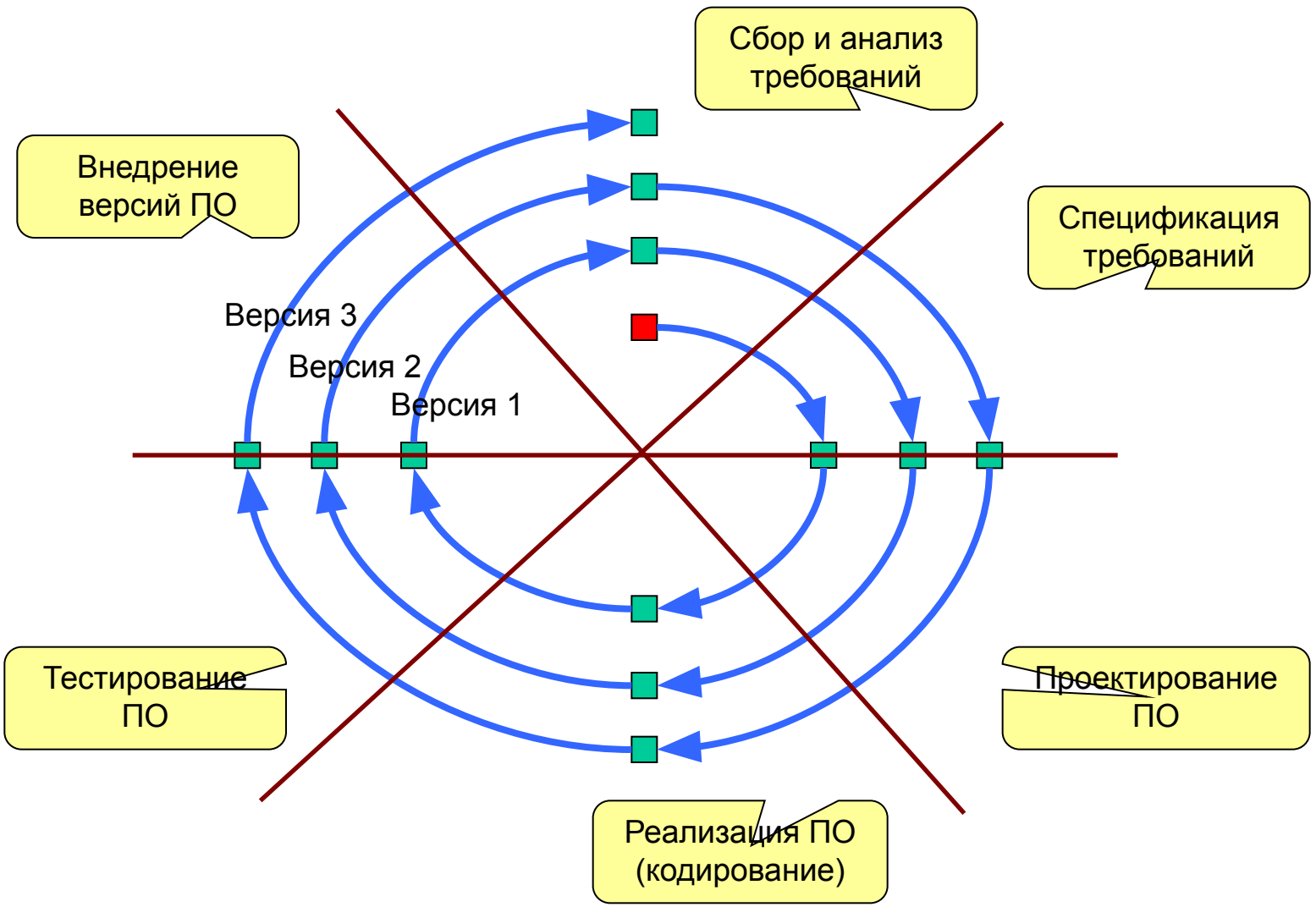
Под жизненным циклом программного обеспечения (ПО) понимается период разработки и эксплуатации ПО, в котором выделяют следующие этапы:

1. Спецификация требований к ПО.
2. Проектирование ПО.
3. Реализация ПО (кодирование).
4. Отладка и тестирование ПО.
5. Внедрение ПО (ввод в эксплуатацию).
6. Эксплуатация и сопровождение ПО.



Объектно-ориентированное проектирование

Спиральная модель жизненного цикла ПО



Объектно-ориентированное проектирование

Проектирование ПО (основные понятия)

Проектирование – это этап жизненного цикла ПО, во время которого разрабатывается структура и взаимосвязи элементов ПО. Результатом является проект, содержащий достаточное количество артефактов, необходимых для реализации ПО.

В российской практике результат проектирования ПО представляется в виде комплекса документов под названием «Технический проект» или «Эскизный проект», а в зарубежной практике - Software Architecture Document или Software Design Document.

Проектирование выполняется на основе: моделей предметной области, требований к ПО, паттернов (шаблонов) проектирования, опыта проектировщиков.

Предметная область - это часть реального мира, которая имеет существенное значение или непосредственное отношение к процессу функционирования ПО.

Требования к ПО – это документ, задающий внешние (видимые) свойства ПО, рассматриваемого как чёрный ящик.

Проектированию обычно подлежат: архитектура ПО, пользовательские интерфейсы, модули и компоненты ПО.

Архитектура ПО - это схемы, диаграммы и другие артефакты, которые дают представление о компонентах, составляющих систему, о взаимосвязях между этими компонентами и правилах, регламентирующих эти взаимосвязи.

Проектирование, кодирование, тестирования и внедрение ПО выполняют команды разработчиков. Для организации работ применяются методы проектного управления.

В зависимости от уровня сложности создаваемого ПО, процесс проектирования может осуществляться «вручную» или при помощи специальных CASE-средств.

CASE-средство – программный комплекс, который автоматизирует технологический процесс проектирования, реализации, тестирования и сопровождения сложного ПО.

Проектирование в лаб. работе №3 выполняется на языке UML в Visual Paradigm.

Объектно-ориентированное проектирование

UML (основные понятия)

Unified Modeling Language – унифицированный язык моделирования, который предназначен для описания, визуализации и документирования объектно-ориентированных систем в процессе их анализа и проектирования.

Язык UML обеспечивает стандартный способ написания проектной документации для ПО, включая концептуальные аспекты (бизнес процессы, функции системы, др.), и конкретные аспекты (классы, объекты, выражения языков программирования, схемы баз данных, повторно используемые компоненты, др.).

Язык UML не является методологией, процессом, языком программирования или формальным языком. **UML = нотация + семантика.**

Здесь под нотацией понимается система условных обозначений для графического представления визуальных моделей. Семантика – это система правил и соглашений, определяющая смысл и интерпретацию конструкций языка.

В языке UML различают диаграммы (модели) следующих двух видов:

Структурные модели (structured models) – модели, предназначенные для описания статической структуры сущностей или элементов некоторой системы, включая классы, интерфейсы, атрибуты и отношения.

Модели поведения (behavioral models) – модели, предназначенные для описания процесса функционирования элементов системы, включая методы и правила взаимодействия между ними, а также процесс изменения состояний отдельных элементов и системы в целом.

Объектно-ориентированное проектирование

Канонические диаграммы языка UML 2.x

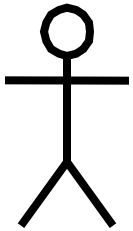
Объектно-ориентированное проектирование

Диаграммы прецедентов (нотация и семантика)

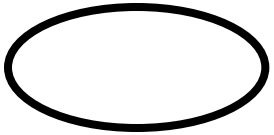
Назначение диаграмм прецедентов:

1. Определить границы функциональности ПО в контексте предметной области.
2. Специфицировать функциональные требования к ПО в форме прецедентов.
3. Получить концептуальную модель ПО для ее последующей детализации.
4. Подготовить документацию для взаимодействия разработчиков и заказчиков ПО.

Основные обозначения на диаграммах прецедентов:



актор (actor)

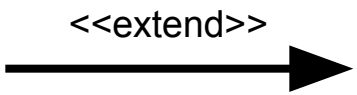


прецедент (use case)

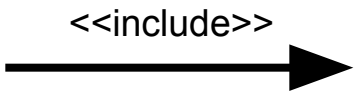
границы системы

ассоциация (коммуникация)

обобщение (наследование)



расширение



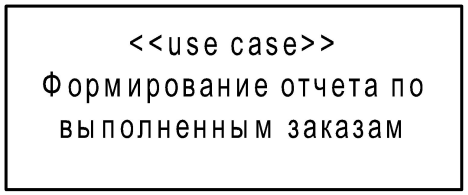
включение

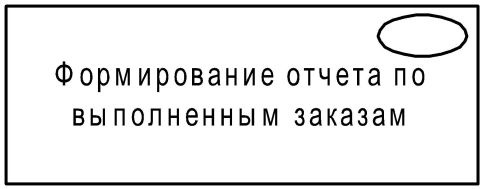
Объектно-ориентированное проектирование

Прецедент (use case) есть общая спецификация совокупности выполняемых системой действий с целью предоставления некоторого наблюдаемого результата, который имеет значение для одного или нескольких акторов.

Прецедент отвечает на вопрос «Что должна выполнять система?», не отвечая на вопрос «Как она должна выполнять это?».

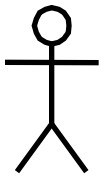
Прецедент именуется отглагольным существительным или глаголом в неопределенной форме.



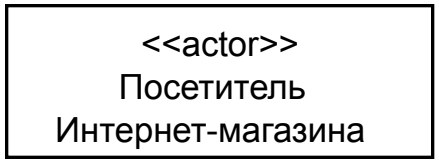


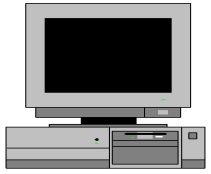
Актор есть любая внешняя по отношению к проектируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач.

Примеры акторов: кассир, клиент банка, банковский служащий, продавец магазина, пассажир авиарейса, администратор гостиницы, сотовый телефон.



Клиент банка





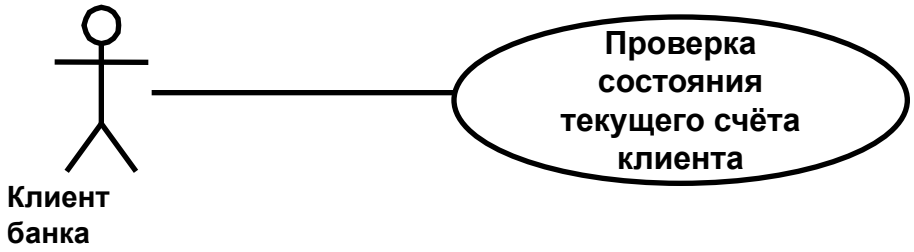
Удаленный пользователь

Объектно-ориентированное проектирование

Как выделить акторов? Типовые вопросы для идентификации акторов в системе:

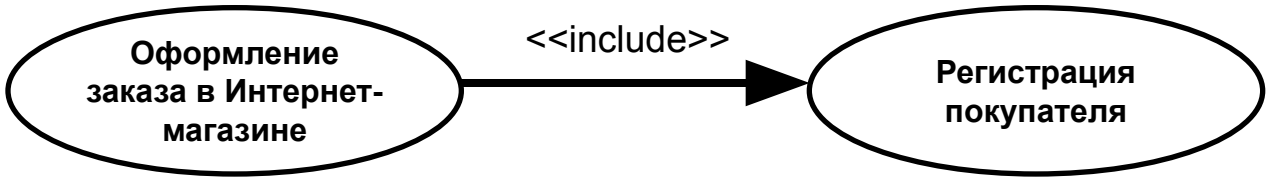
1. Какие организации или лица будут использовать систему?
2. Кто будет получать пользу от использования системы?
3. Кто будет использовать информацию от системы?
4. Будет ли система использовать внешние ресурсы?
5. Будет ли разрабатываемая система взаимодействовать с другими системами?
6. Может ли один пользователь играть несколько ролей при взаимодействии с системой?
7. Могут ли различные пользователи играть одну роль при взаимодействии с системой?
8. Будет ли система взаимодействовать с законодательными, исполнительными, налоговыми или другими органами?

На диаграммах прецедентов взаимодействие актора с прецедентом всегда обозначается отношением ассоциации, например так:



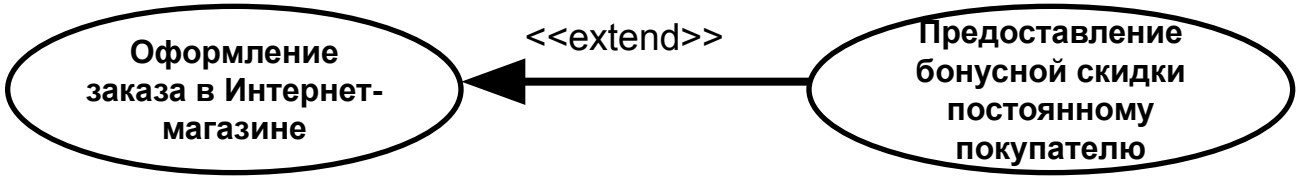
Объектно-ориентированное проектирование

Отношение включения (include) отражает тот факт, что некоторый прецедент содержит поведение, определенное в другом прецеденте, например:



прецедент А прецедент Б

Отношение расширения (extend) определяет взаимосвязь одного прецедента с некоторым другим прецедентом, функциональность или поведение которого задействуется первым прецедентом не всегда, а только при выполнении некоторых дополнительных условий.

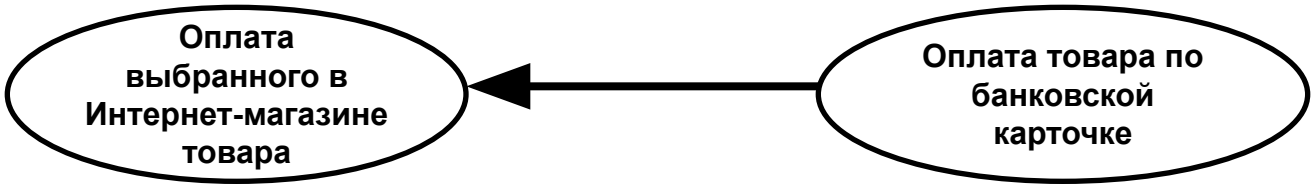


прецедент А прецедент Б

Объектно-ориентированное проектирование

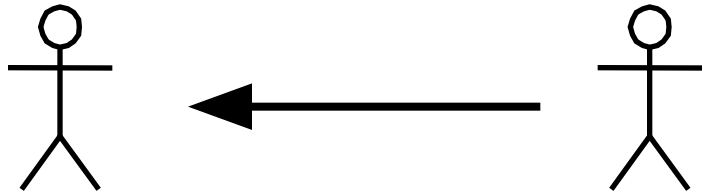
Отношение обобщения (generalization relationship) предназначено для отражения того факта, что один элемент модели является специальным или частным случаем другого элемента модели.

Например, так:



прецедент А прецедент Б

Или так:



Посетитель Интернет-магазина (актор А)

Покупатель (актор Б)

Легко видеть, что отношение обобщения действует аналогично отношению наследования, но направлено оно «в обратную сторону».

Объектно-ориентированное проектирование

Диаграммы прецедентов (примеры разработки)

Диаграммы прецедентов должны отражать функциональные требования к ПО

Требование к ПО – это некоторое свойство, которым должна обладать система или ее компонент, чтобы удовлетворять условиям контракта, положениям стандартов, формальной спецификации или технической документации.

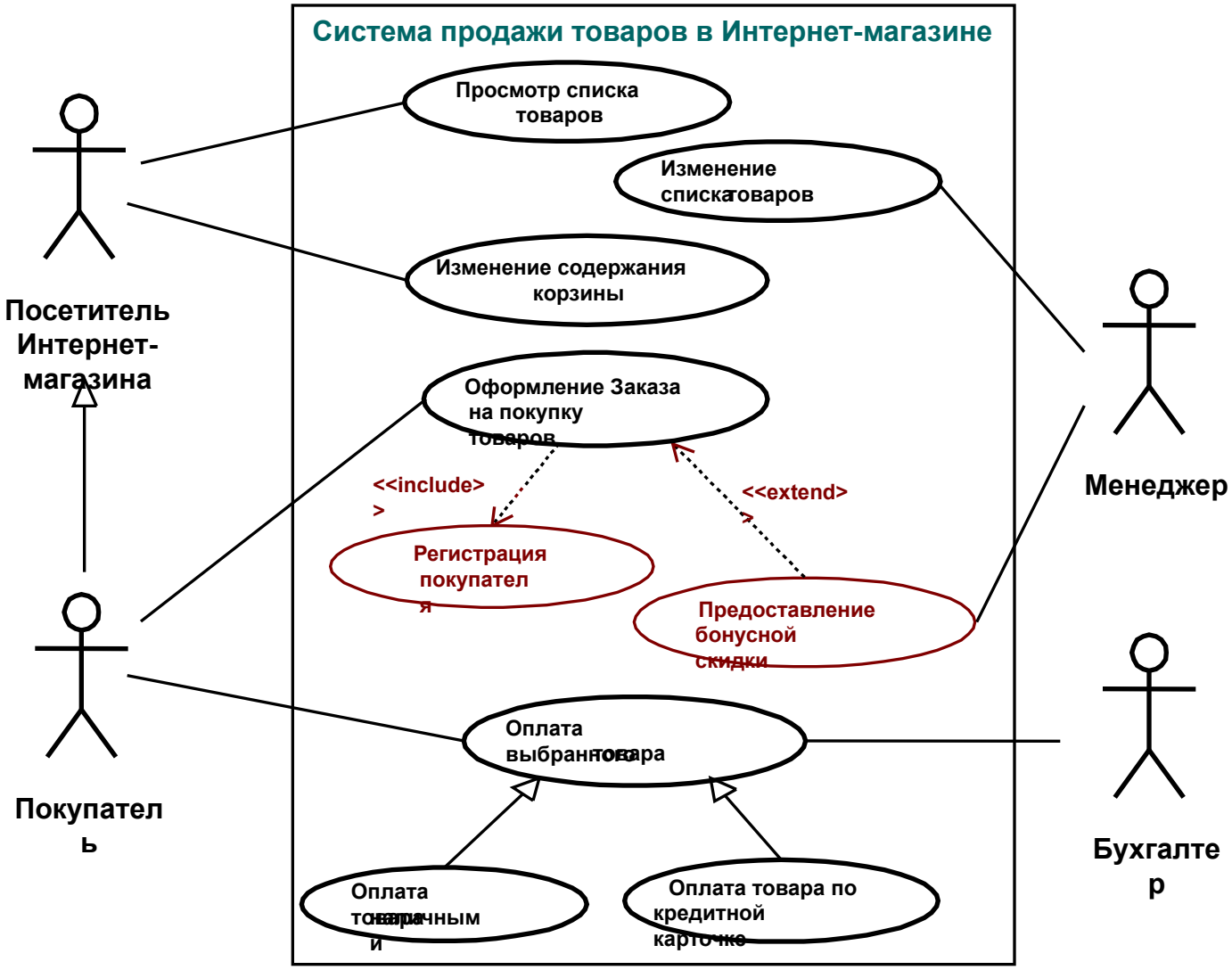
Функциональные требования определяют действия, которые должна быть способна выполнить система, без рассмотрения физических особенностей их реализации. Функциональные требования определяют внешнее поведение системы. Лучше всего они описываются в форме прецедентов. Каждому функциональному требованию должен соответствовать отдельный прецедент

Последовательность разработки диаграмм прецедентов следующая:

1. Выявить основных акторов и определить их цели по отношению к системе.
2. Выявить и специфицировать все базовые (основные) прецеденты.
3. Выделить цели базовых прецедентов, интересы акторов в контексте этих прецедентов, предусловия и постусловия прецедентов.
4. Написать типовой успешный сценарий выполнения базовых прецедентов.
5. Определить исключения (нештатные ситуации) в сценариях прецедентов и написать сценарии для всех выявленных исключений.
6. Выделить прецеденты исключений и изобразить их со стереотипом «extend».
7. Выделить общие фрагменты функциональности прецедентов и отобразить их как отдельные прецеденты со стереотипом «include».
8. Проверить диаграмму на отсутствие дублирования прецедентов и акторов.

Объектно-ориентированное проектирование

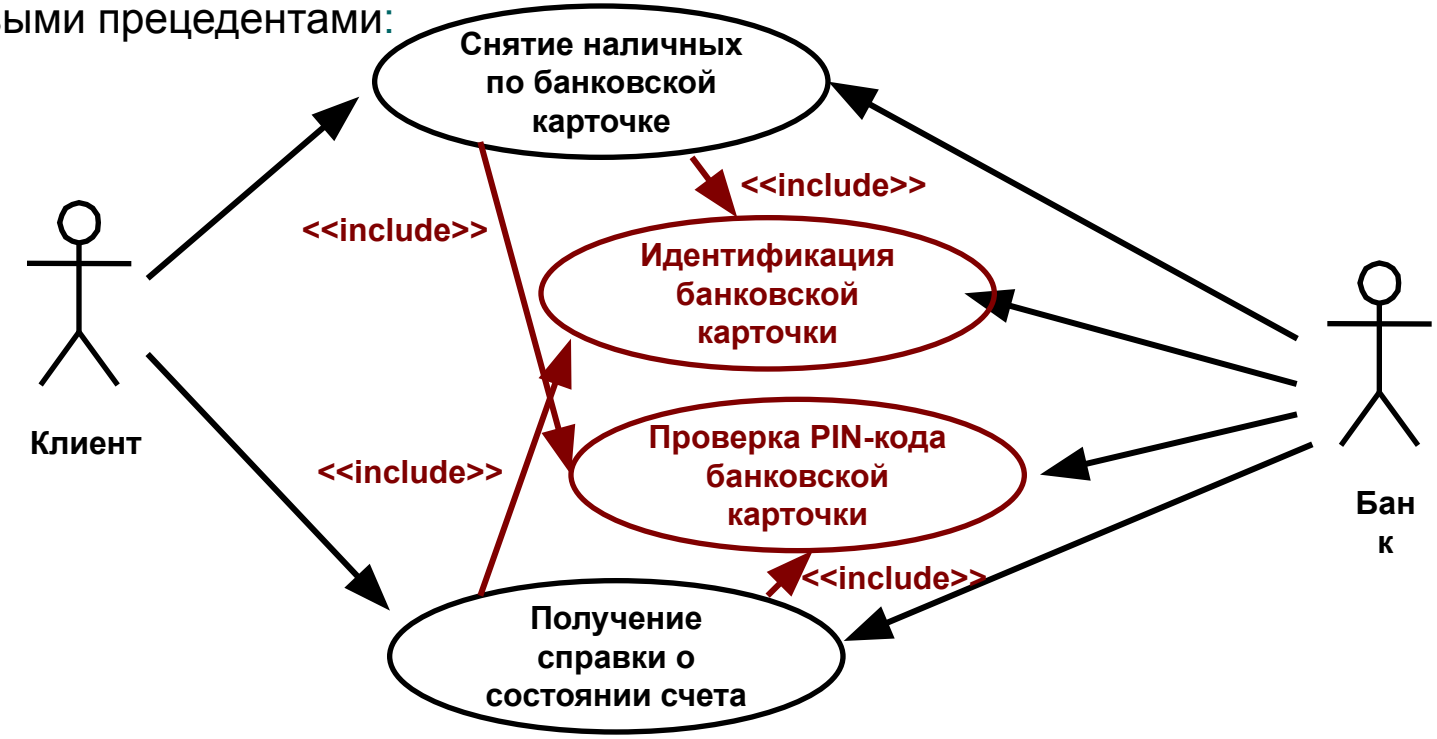
Пример 1. Диаграмма прецедентов для продажи товаров через Интернет-магазин



Объектно-ориентированное проектирование

Пример 2. Диаграмма прецедентов для модели банкомата

Система имеет двух акторов - Клиент банкомата и Банк. Каждый из акторов взаимодействует с банкоматом. Функциональные требования к банкомату следующие: 1) снятие наличных по банковской карточке; 2) получении справки о состоянии счета. Эти функциональные требования специфицируются двумя базовыми прецедентами:



Для выполнения базовых прецедентов необходимо идентифицировать банковскую карточку и проверить её PIN-код, поэтому оба базовые прецедента обращаются к дополнительным сервисам (другим прецедентам).

Объектно-ориентированное проектирование

Показатели качества диаграмм прецедентов:

1. Все ли функциональные требования к ПО отражены в виде прецедентов?
2. Не содержат ли диаграммы прецедентов лишнее поведение, которого нет в требованиях?
3. Действительно ли необходимы все отношения включения, расширения и обобщения?
4. Правильно ли произведено деление модели на пакеты прецедентов?
5. Стала ли диаграмма (модель) в результате деления её на пакеты проще и/или удобнее для восприятия?
6. Можно ли на основе диаграмм прецедентов составить четкое представление о функционировании системы в контексте ее пользователей?

Типичные ошибки при разработке диаграмм прецедентов:

1. Превращение диаграммы прецедентов в диаграмму деятельности за счет желания детально отразить все функциональные действия (излишняя детализация).
2. Инициатором действий является разрабатываемая система, а не акторы.
3. Преждевременная спецификация свойств и методов классов до того, как сформулированы все прецеденты.
4. Использование кратких, невыразительных имен для прецедентов.
5. Описание прецедентов в терминах, непонятных пользователям и заказчикам.
6. Отсутствие описаний альтернативных сценариев (исключений, нештатных ситуаций).
7. Тратится слишком много времени на решение вопросов о том, какие стереотипы и ассоциации использовать в диаграмме.

Объектно-ориентированное проектирование

Сценарии выполнения прецедентов (пример)

Для каждого базового прецедента диаграммы прецедентов необходимо продумать, разработать и описать текстовый сценарий его выполнения.

Сценарий выполнения каждого базового прецедента имеет, в общем случае, три раздела, которые оформляются как таблицы.

В качестве примера рассмотрим сценарий выполнения базового прецедента «Снятие наличных по банковской карточке» для модели банкомата (разделы сценария: «Общее описание сценария», «Типичный ход событий», «Исключения»).

Раздел «Общее описание сценария»

Прецедент	Снятие наличных по банковской карточке
Акторы	Клиент, Банк
Цель	Получение Клиентом требуемой суммы наличными
Краткое описание	Клиент запрашивает требуемую сумму. Банкомат обеспечивает доступ к счету Клиента. Банкомат выдает Клиенту наличные.
Тип	Базовый прецедент
Ссылки на другие прецеденты	• Идентификация банковской карточки • Проверка PIN-кода банковской карточки

Объектно-ориентированное проектирование

Раздел «Типичный ход событий»

Действия акторов	Отклик системы
1. Клиент вставляет кредитную карточку в устройство чтения банкомата Исключение №1: Банковская карточка недействительна или неверно вставлена	2. Банкомат выполняет идентификацию банковской карточки 3. Банкомат предлагает ввести PIN-код
4. Клиент вводит PIN-код Исключение №2: Клиент вводит неверный PIN-код	5. Банкомат проверяет PIN-код 6. Банкомат отображает опции меню
7. Клиент выбирает в меню опцию «Снятие наличных со своего счета»	8. Система делает запрос в Банк и выясняет текущее состояние счета Клиента 9. Банкомат предлагает ввести требуемую сумму
10. Клиент вводит требуемую сумму 11. Банк проверяет введенную сумму Исключение №3: Требуемая сумма превышает лимит средств на счете Клиента, доступных для снятия	12. Банкомат изменяет состояние счета Клиента, выдает наличные и чек
13. Клиент получает наличные и чек	14. Банкомат предлагает Клиенту забрать банковскую карточку
15. Клиент получает свою банковскую карточку	16. Банкомат отображает сообщение о готовности к работе

Объектно-ориентированное проектирование

Раздел «Исключения»

Действия акторов	Отклик системы
Исключение №1: Банковская карточка недействительна или неверно вставлена	3. Банкомат отображает информацию о недействительной или неверно вставленной банковской карточке 14. Банкомат предлагает Клиенту забрать банковскую карточку
15. Клиент получает свою банковскую карточку	
Исключение №2: Клиент вводит неверный PIN-код	6. Банкомат отображает информацию о неверном PIN-коде
4. Клиент вводит новый PIN-код	
Исключение №3: Требуемая сумма превышает лимит средств на счете Клиента, доступных для снятия	12. Банкомат отображает информацию о превышении лимита средств на счете Клиента, доступных для снятия
10. Клиент вводит новую требуемую сумму	

Объектно-ориентированное проектирование

Диаграммы классов (нотация и семантика)

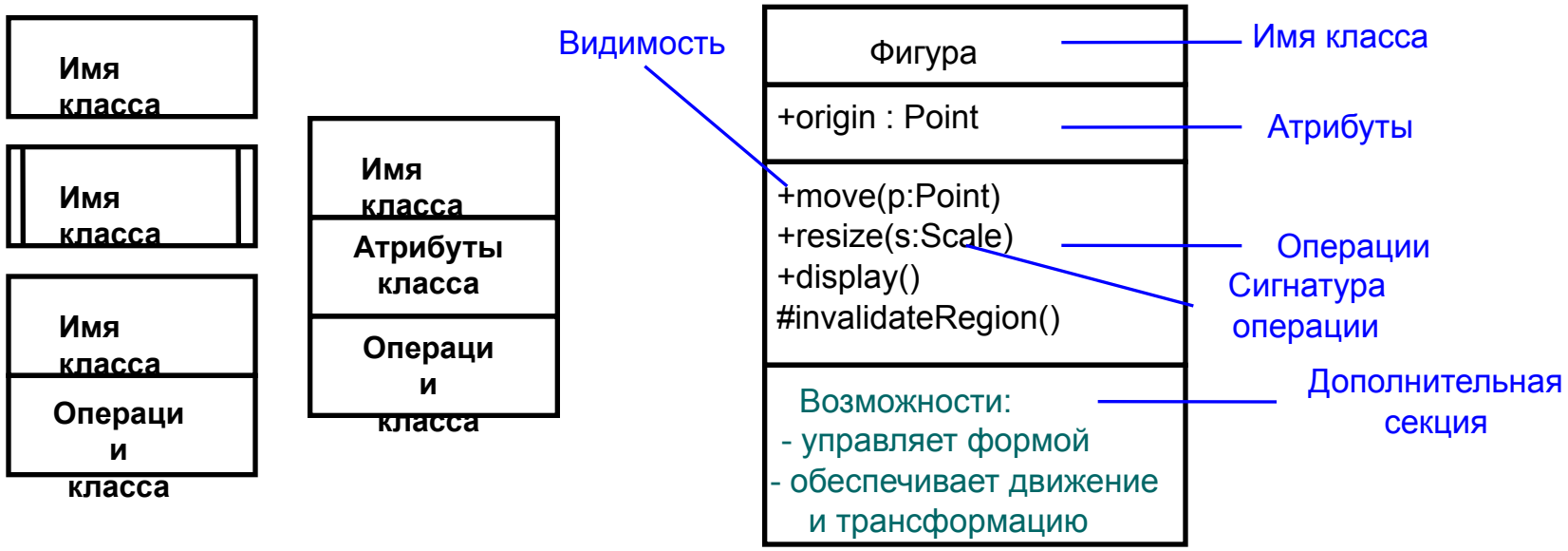
Диаграмма классов есть основная логическая модель проектируемой системы.

Диаграмма классов (class diagram) - предназначена для представления статической структуры программной системы в терминах классов ООП.

Диаграмма классов представляет собой граф, узлами которого являются элементы типа “классификатор”, которые связаны различными типами структурных отношений.

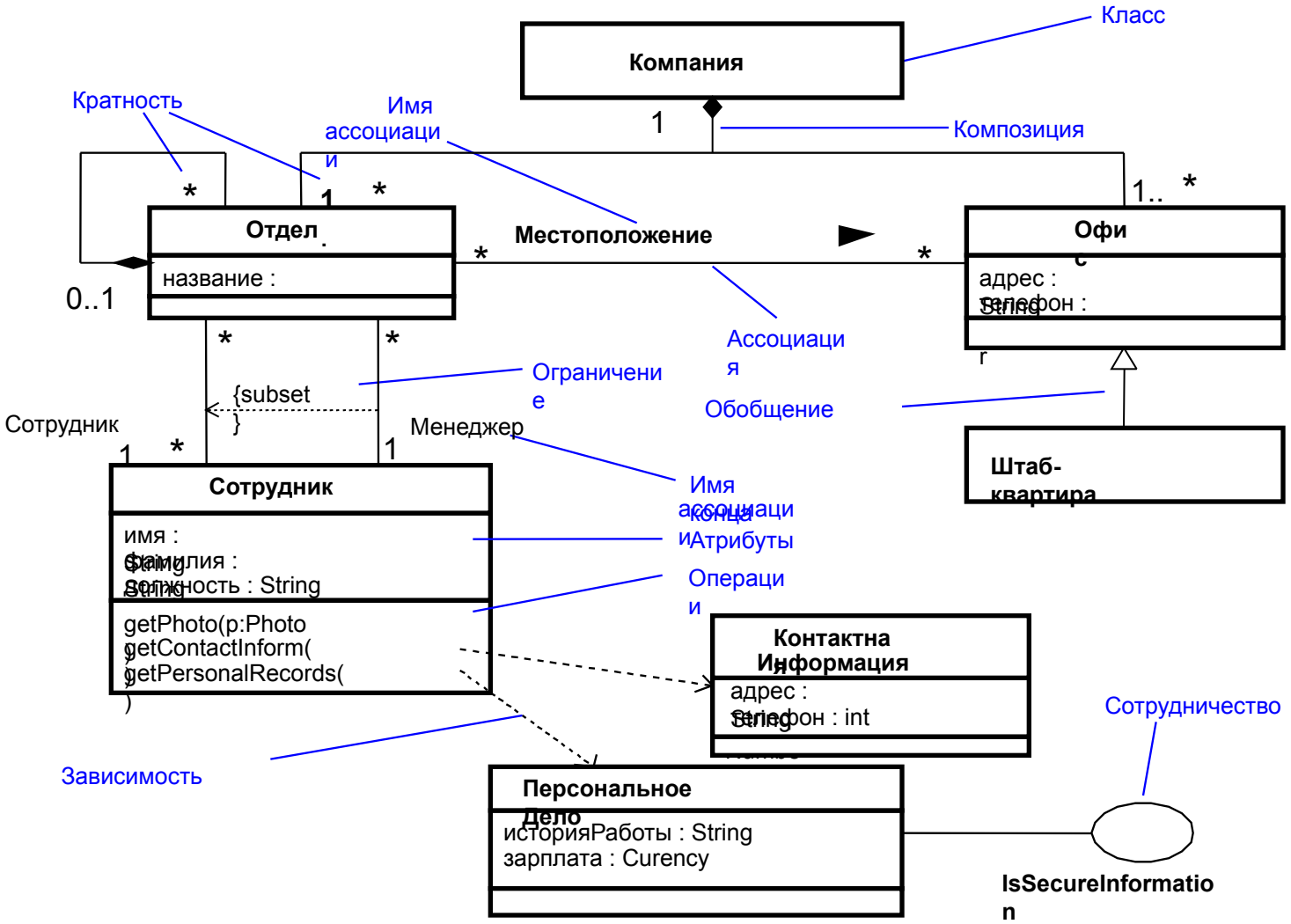
Классификатор (classifier) - специальное понятие, предназначенное для классификации экземпляров (объектов), которые имеют общие характеристики.

Варианты графического изображения классов на Диаграмме классов:



Объектно-ориентированное проектирование

Основные обозначения на Диаграмме классов



Объектно-ориентированное проектирование

Пример Диаграммы классов.

Имеется два пакета классов: «Холдинг» и «Персонал».

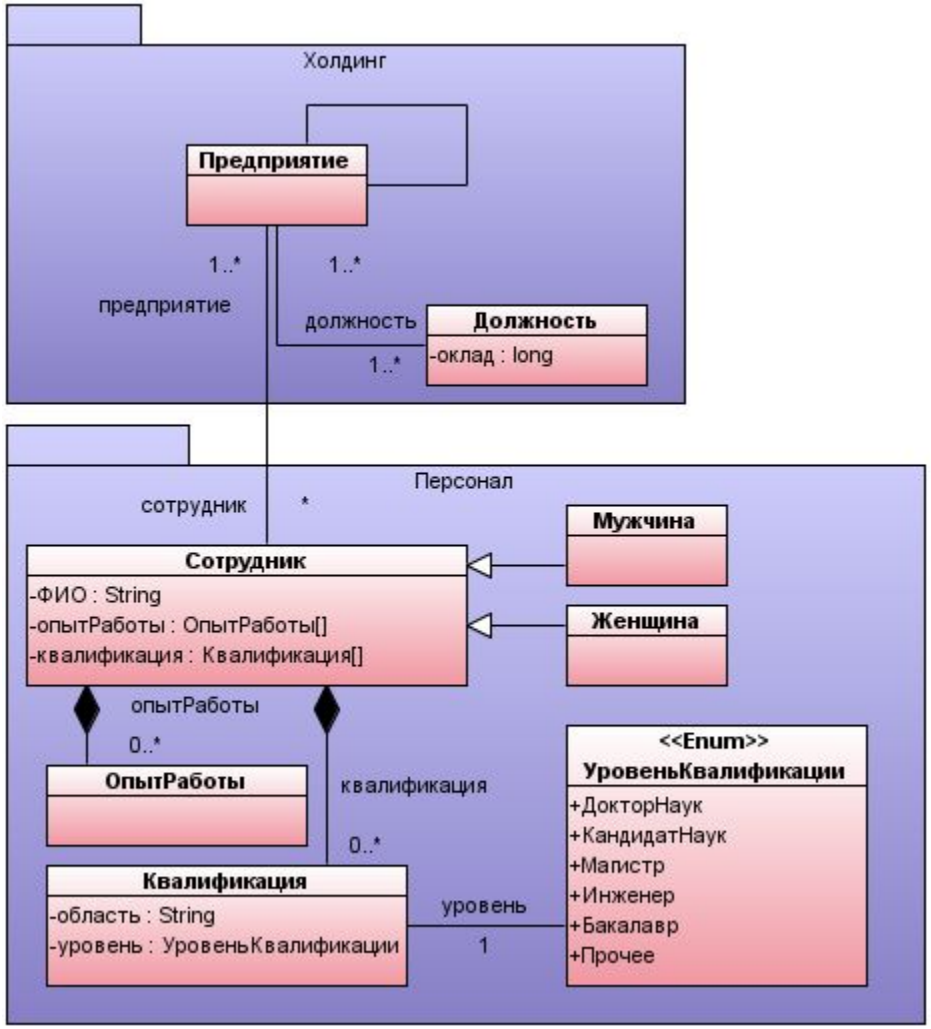
Класс «Предприятие» многократно ассоциирован с классами «Должность» и «Сотрудник».

Для *C++* это значит, что в классе «Предприятие» есть два контейнера указателей на объекты этих классов.

Класс «Сотрудник» содержит объекты классов «ОпытРаботы» и «Квалификация». Также содержит контейнер указателей на объекты класса «Предприятие» (для *C++*).

Классы «Мужчина» и «Женщина» есть потомки класса «Сотрудник».

Класс «УровеньКвалификации» по сути представляет собой перечислимый тип. Он однократно ассоциирован с классом «Квалификация».



Объектно-ориентированное проектирование

Диаграммы классов (атрибуты)

Атрибуты класса задают свойства, которые являются общими для всех объектов данного класса. В C++ элементы данных класса по сути и есть атрибуты класса.

Формат задания атрибутов класса (БНФ):

<атрибут>::= [<видимость>] [‘/’] <имя атрибута> [‘:’<тип атрибута>] [‘[‘<кратность>‘]’] [‘=’ <значение по умолчанию>] [‘{‘<модификаторы атрибута> ’}’]

<видимость> атрибутов может принимать следующие значения:

+ public (общедоступный).

- private (скрытый, видим только внутри класса).

protected (защищенный, видим внутри класса и для потомков).

~ package (видим только в ближайшем охватывающем пакете)

Символ“/” означает, что атрибут является производным. Значение производного атрибута может быть вычислено на основе значений других атрибутов этого или других классов. Поэтому данный атрибут называют иногда вычислимым.

<имя атрибута> - строка текста, которая используется в качестве идентификатора атрибута. Имя должно быть уникальным в пределах данного класса.

<тип атрибута> - имя классификатора, который является типом данного атрибута.

<кратность> - характеризует общее количество конкретных значений для атрибута, которые могут быть заданы для объектов данного класса. По сути, задает мощность множества значений атрибута. Используется специальный синтаксис.

Объектно-ориентированное проектирование

<значение по умолчанию> - выражение, которое служит для задания начального значения или значений данного атрибута в момент создания объекта класса.

<модификатор атрибута> - текстовое выражение, которое придает атрибуту дополнительную семантику. Формат следующий (БНФ):

<модификатор атрибута> ::= ‘readOnly’ | ‘union’ | ‘subsets‘ <имя атрибута> | ‘redefines’ <имя атрибута> | ‘ordered’ | ‘unique’ | <ограничение атрибута>

Примеры записи атрибутов	Комментарий
+ имяСотрудника : String {readOnly}	Общедоступный атрибут типа String. Разрешено только чтение значений.
~ датаРождения : Data {readOnly}	Доступный в пределах пакета атрибут типа Data. Разрешено только чтение значений.
# /возрастСотрудника : Integer	Защищенный производный атрибут типа Integer. Значение атрибута вычислимо.
+ номерТелефона : Integer [1..*] {unique}	Общедоступный атрибут типа Integer. Может принимать значения >= 1. Значения не могут повторяться.
– заработнаяПлата : Currency = 500.00	Скрытый атрибут типа Currency. Значение по умолчанию есть 500.00.

Объектно-ориентированное проектирование

Диаграммы классов (операции)

Операции класса задают поведение, которое является общим для всех объектов данного класса. В *C++* функции-члены класса по сути и есть операции класса.

Формат задания операций класса (БНФ):

<операция> ::= [<видимость>] <имя операции> ‘(‘ [<список параметров>] ‘)’

[‘:’ [<тип результата>] ‘{‘ <свойства операции> ‘}’]

<список параметров> - список формальных аргументов, формат их следующий:

<параметр> ::= [<направление>] <имя параметра> ‘:’ <тип параметра> [‘[‘<кратность>’]’] [‘=’ <значение по умолчанию>] [‘{‘ <свойства параметра > ‘}’]

<направление> ::= ‘in’ | ‘out’ | ‘inout’| ‘return‘. Если не указано, по умолчанию “in”.

in – значения параметра передаются в операцию вызывающим объектом.

inout – значения параметра передаются в операцию вызывающим объектом и затем передаются вызывающему объекту после окончания выполнения операции.

out – значения параметра передаются вызывающему объекту после окончания выполнения операции.

return – значения параметра передаются в качестве возвращаемых значений вызывающему объекту после окончания выполнения операции.

<видимость>, <имя операции>, <имя параметра>, <тип параметра>, <кратность> и <значение по умолчанию> имеют тот же смысл, что и для атрибутов.

Объектно-ориентированное проектирование

<свойства параметра > - указывает дополнительные свойства значений данного формального параметра. Имеет тот же смысл что и <модификатор атрибута>.

<свойства операции> - может принимать следующие значения:

redefines <имя операции> – данная операция переопределяет некоторую наследуемую операцию с именем <имя операции>.

query – операция не изменяет состояния системы и не имеет побочных эффектов.

ordered – значения возвращаемого параметра являются упорядоченными.

unique – значения возвращаемого параметра не могут повторяться.

Примеры записи атрибутов	Комментарий
+добавить(in номерТелефона : Integer [*] {unique})	Общедоступная операция. Значение параметра типа Integer передается в операцию вызывающим объектом. Может принимать значения >= 1, не может повторяться.
–изменить(in заработнаяПлата : Currency)	Скрытая операция. Значение параметра типа Currency передается вызывающим объектом.
+создать() : Boolean	Общедоступная операция без параметров. Возвращает значение типа Boolean.
toString(return : String)	Операция с одним параметром типа String.
toString( ) : String	Операция без параметров. Возвращает значение типа String.

Объектно-ориентированное проектирование

Диаграммы классов (отношения)

В диаграммах классов используются отношения следующих шести видов:

Отношение	Нотация	Семантика
Ассоциация (association)		Произвольное отношение между объектами классов
Обобщение (generalization)		Отношение типа «общее – частное». По сути, это есть «наследование наоборот».
Агрегация (aggregation)		Отношение типа «часть-целое»
Композиция (composition)		Сильная форма отношения типа «часть – целое»
Реализация (realization)		Отношение между значком «интерфейс» и его реализацией в виде класса
Зависимость (dependency)		Направленное отношение между двумя элементами диаграммы с открытой семантикой

Ассоциация является наиболее универсальным видом отношений. Для корректного построения диаграмм классов необходимо крайне аккуратно относиться к выбору и спецификации отношений между классами, подробности см далее.

Объектно-ориентированное проектирование

Отношения ассоциации

Имя конца ассоциации специфицирует роль, которую играет класс, расположенный на соответствующем конце рассматриваемой ассоциации.

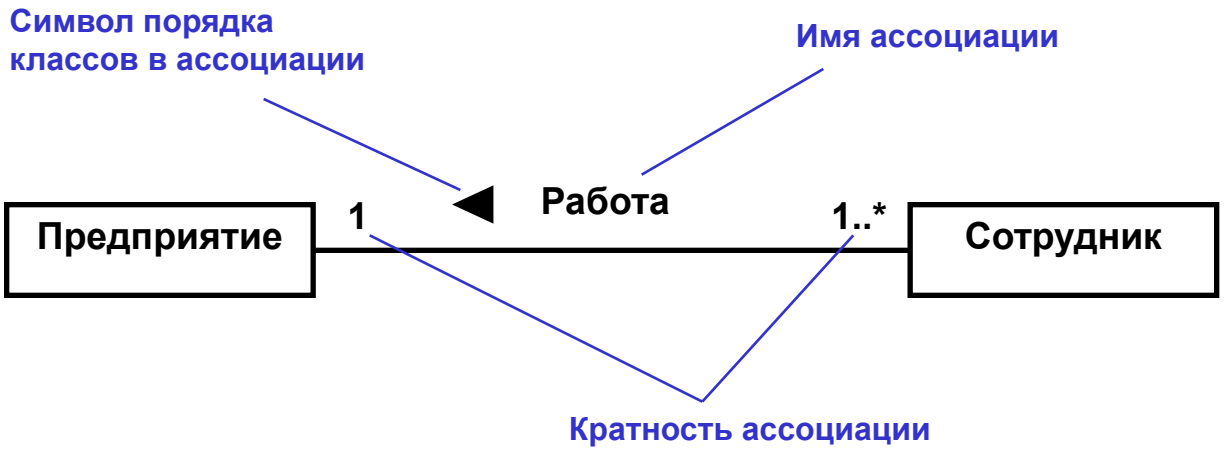
Видимость конца ассоциации специфицирует возможность доступа к данному концу ассоциации с других ее концов.

Кратность конца ассоциации специфицирует количество объектов класса, которое может соотноситься с одним объектом класса на другом конце этой ассоциации.

Символ наличия навигации изображается с помощью простой стрелки в форме буквы «V» на конце ассоциации (например, контейнер будет снабжен итератором).

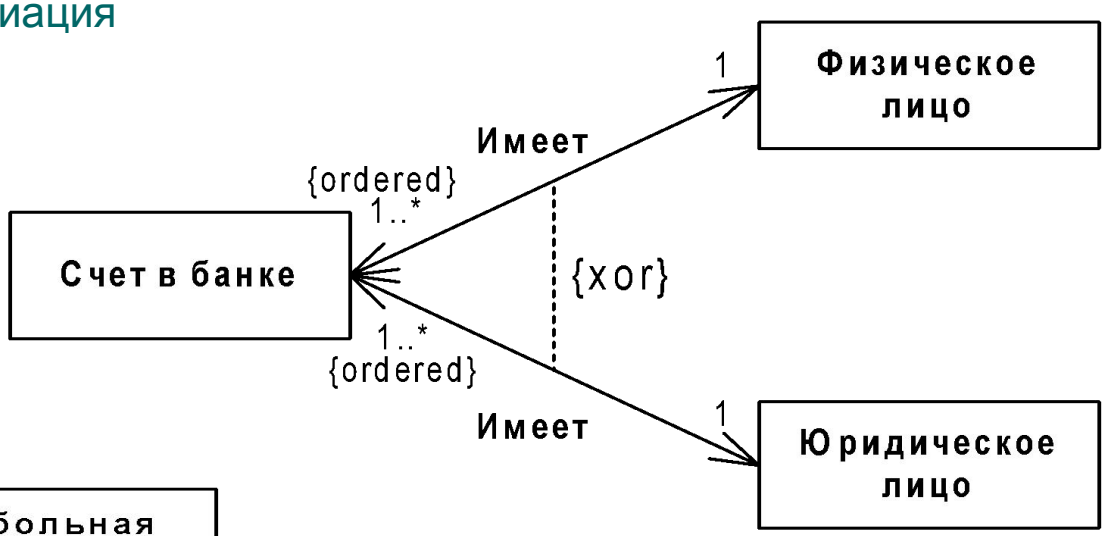
Символ отсутствия навигации изображается с помощью буквы «X» на линии у конца ассоциации (например, контейнер не будет снабжен итератором).

Пример 1. Бинарная ассоциация

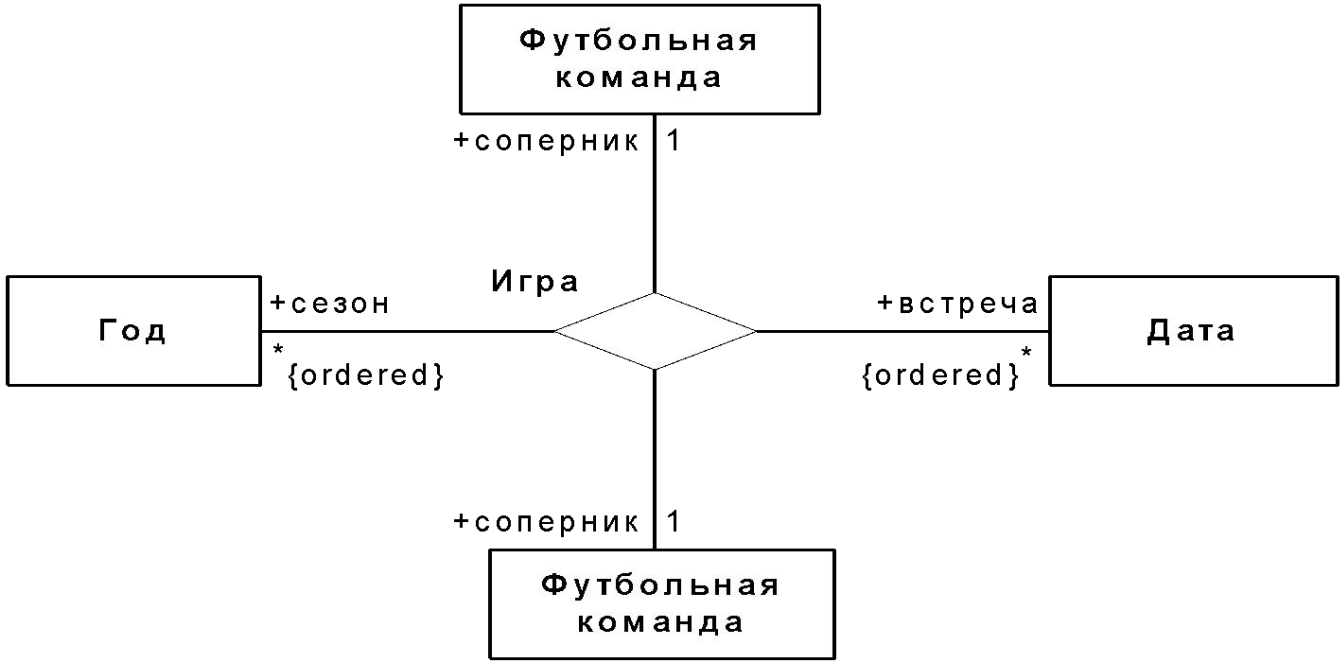


Объектно-ориентированное проектирование

Пример 2. Исключающая ассоциация



Пример 3. 4-арная ассоциация



Объектно-ориентированное проектирование

Пример 4. Ассоциация класс

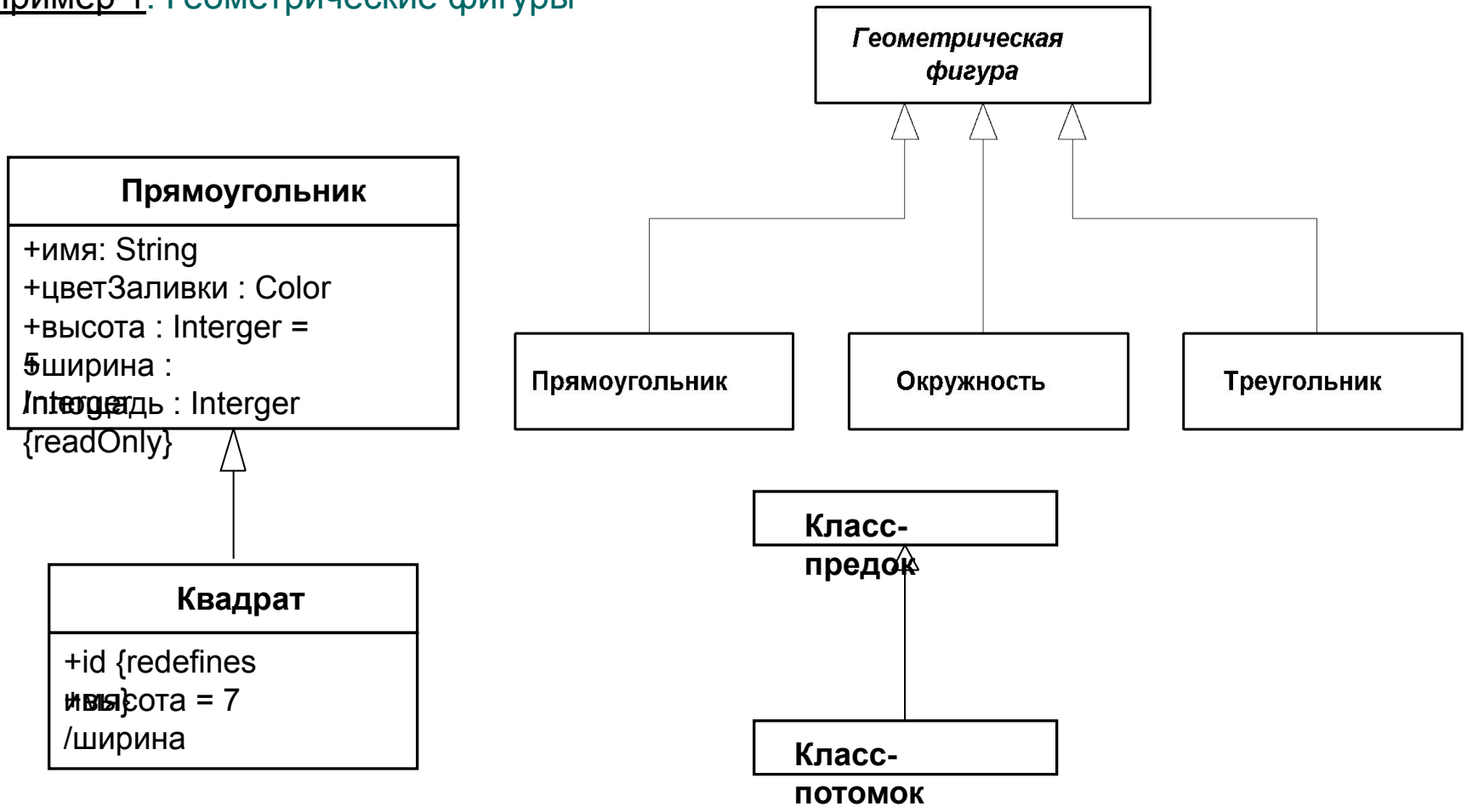
Ассоциация класс есть элемент диаграммы классов, который имеет свойства как ассоциации, так и класса, и предназначен для спецификации дополнительных свойств ассоциации в форме атрибутов и, возможно, операций класса.

Объектно-ориентированное проектирование

Отношения обобщения

Обобщение – это таксономическое отношение между некоторым общим классификатором (предком) и более специальным классификатором (потомком).

Пример 1. Геометрические фигуры



Объектно-ориентированное проектирование

Пример 2. Спецификация ограничений при обобщении.

Рядом со стрелкой отношения обобщения может размещаться текст в фигурных скобках, который задает некоторые ограничения для данного отношения.

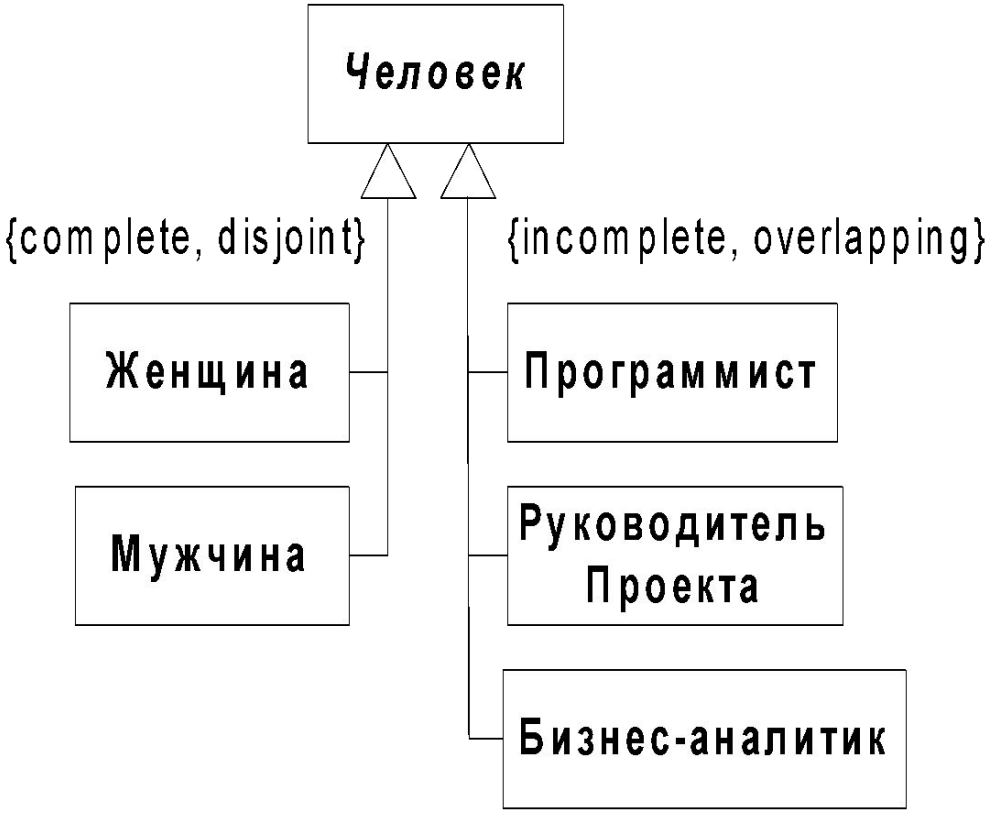
Возможны следующие виды ограничений при обобщении:

{complete} - все потомки рассматриваемого класса определены (возможно, не показаны на данной диаграмме), и список потомков не может быть расширен.

{disjoint} – запрещено множественное наследование.

{incomplete} - возможно появление новых классов-потомков.

{overlapping} – разрешено множественное наследование.



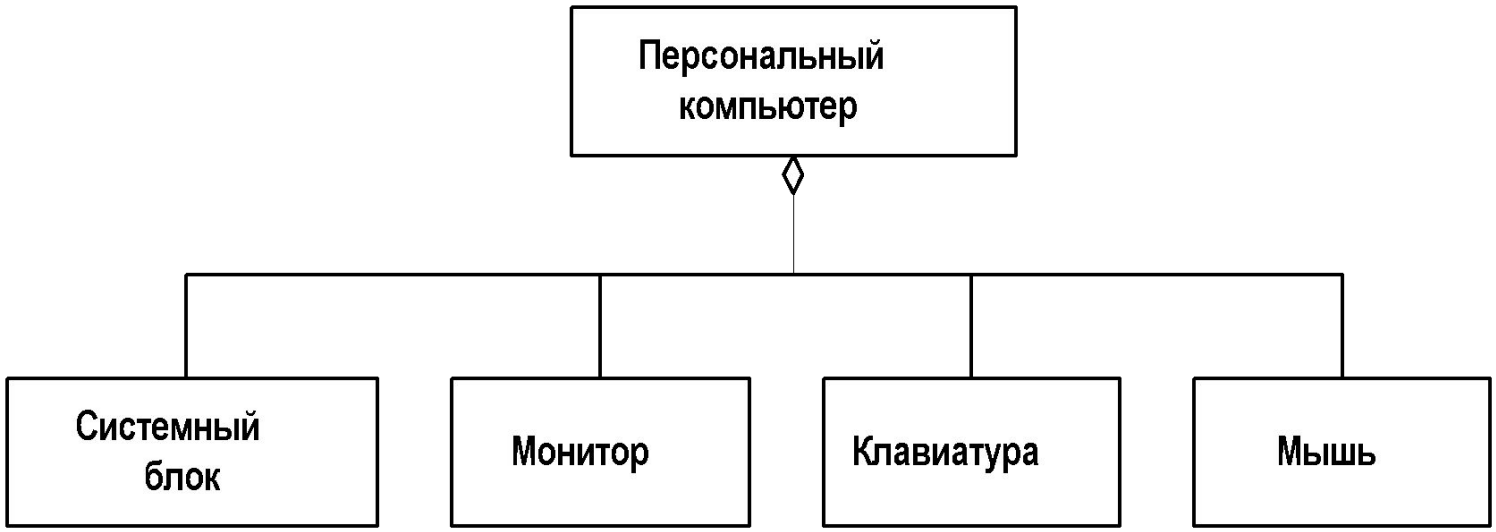
Объектно-ориентированное проектирование

Отношения агрегации

Отношение агрегации – это направленное отношение между двумя классами, когда один из классов представляет собой некоторую сущность, которая включает в себя в качестве составных частей объекты других классов.

Пример 1. Контейнер

Пример 2. Персональный компьютер

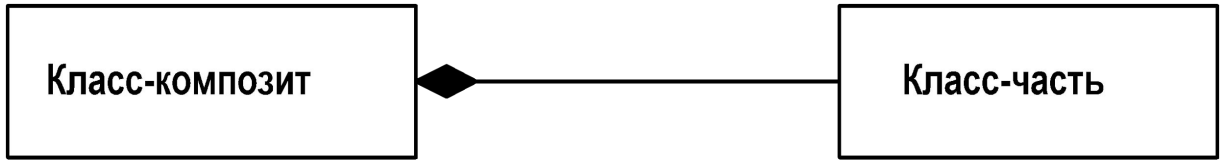


Объектно-ориентированное проектирование

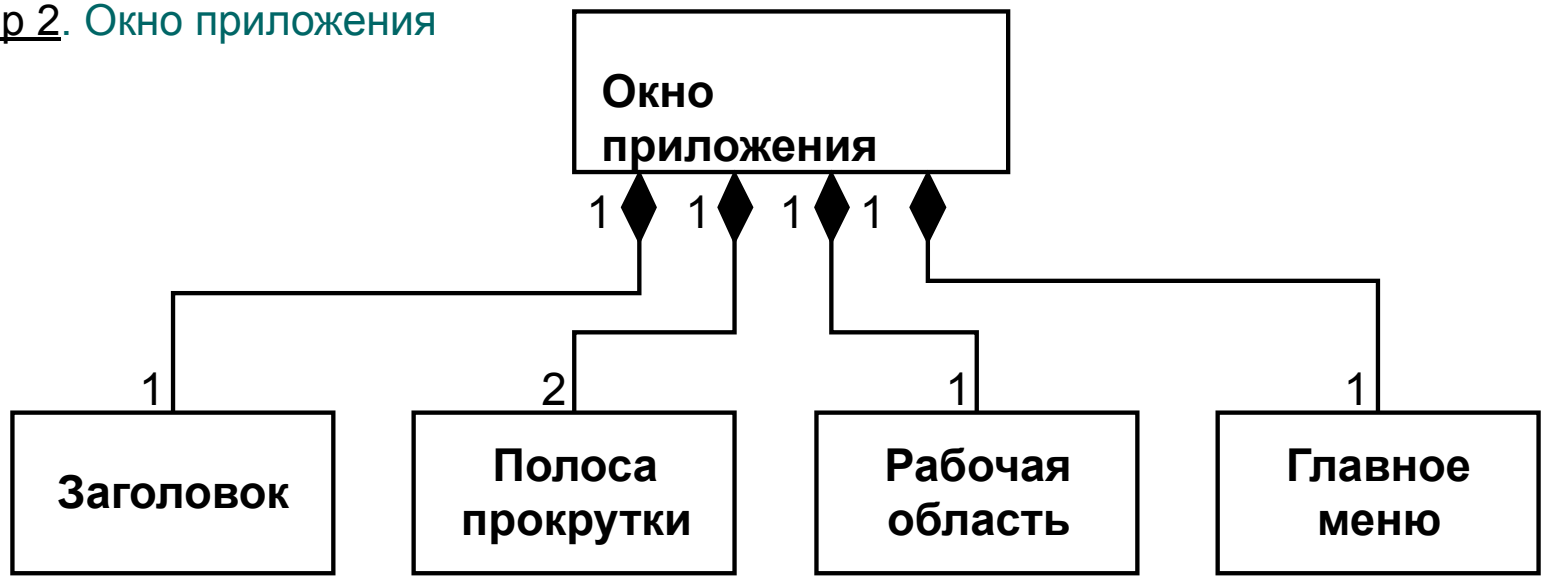
Отношения композиции

Отношение композиции есть сильная форма отношения "часть-целое", при которой с уничтожением объекта класса-контейнера (класса-композита) уничтожаются и все объекты, являющимися его составными частями.

Пример 1. Композитный класс

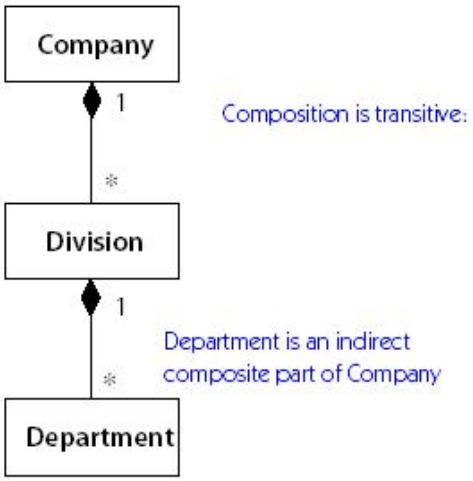


Пример 2. Окно приложения



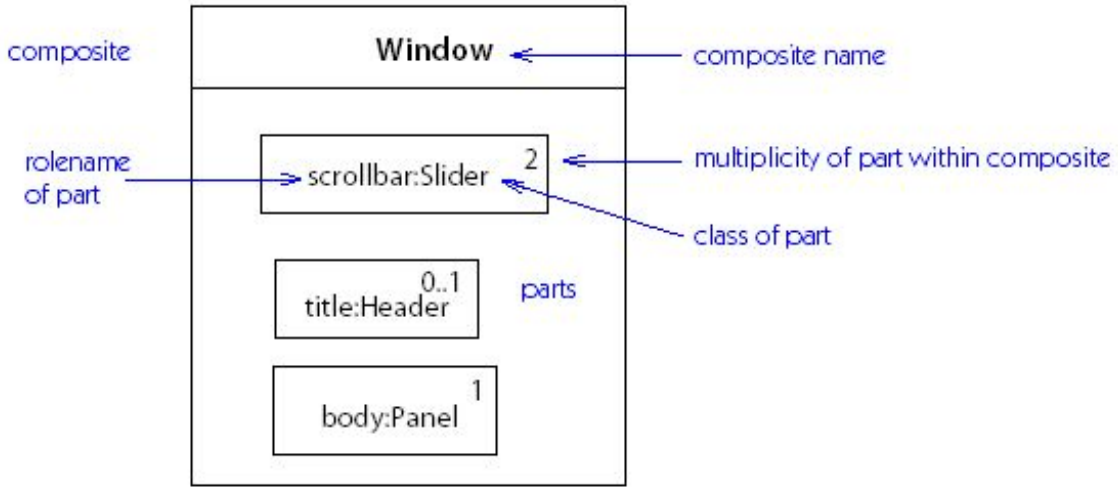
Объектно-ориентированное проектирование

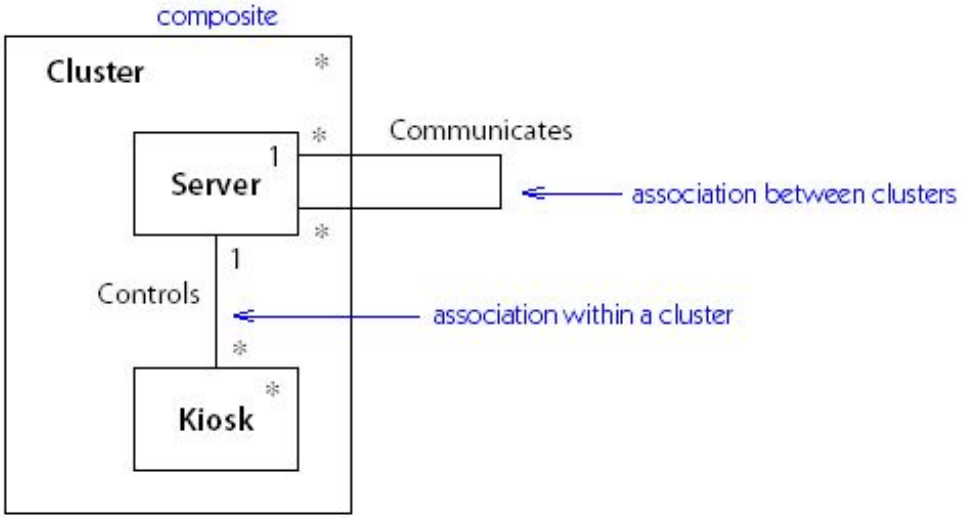
Пример 3. Применение различных нотаций при композиции



Composition is transitive:

Department is an indirect composite part of Company

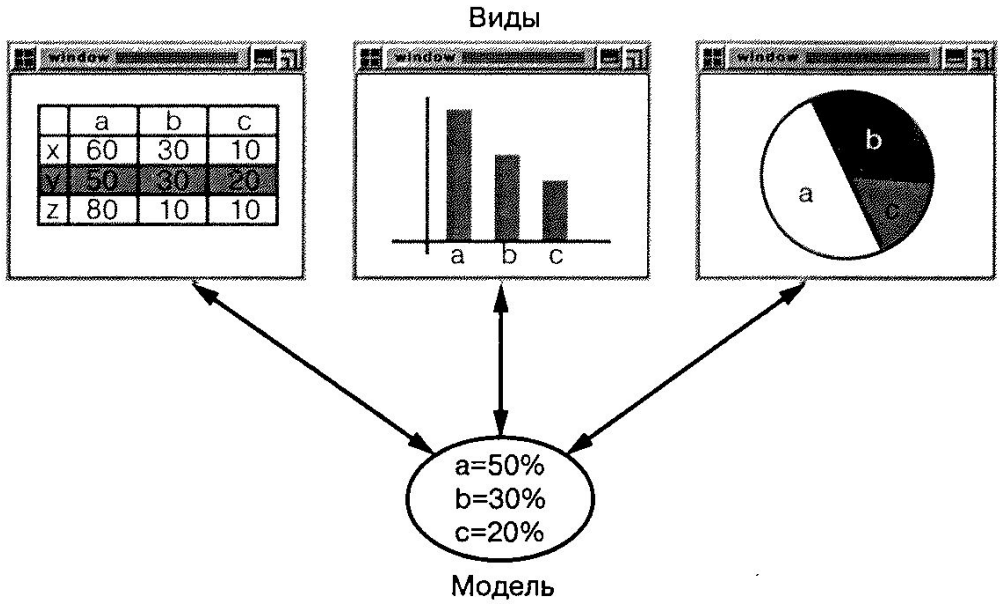




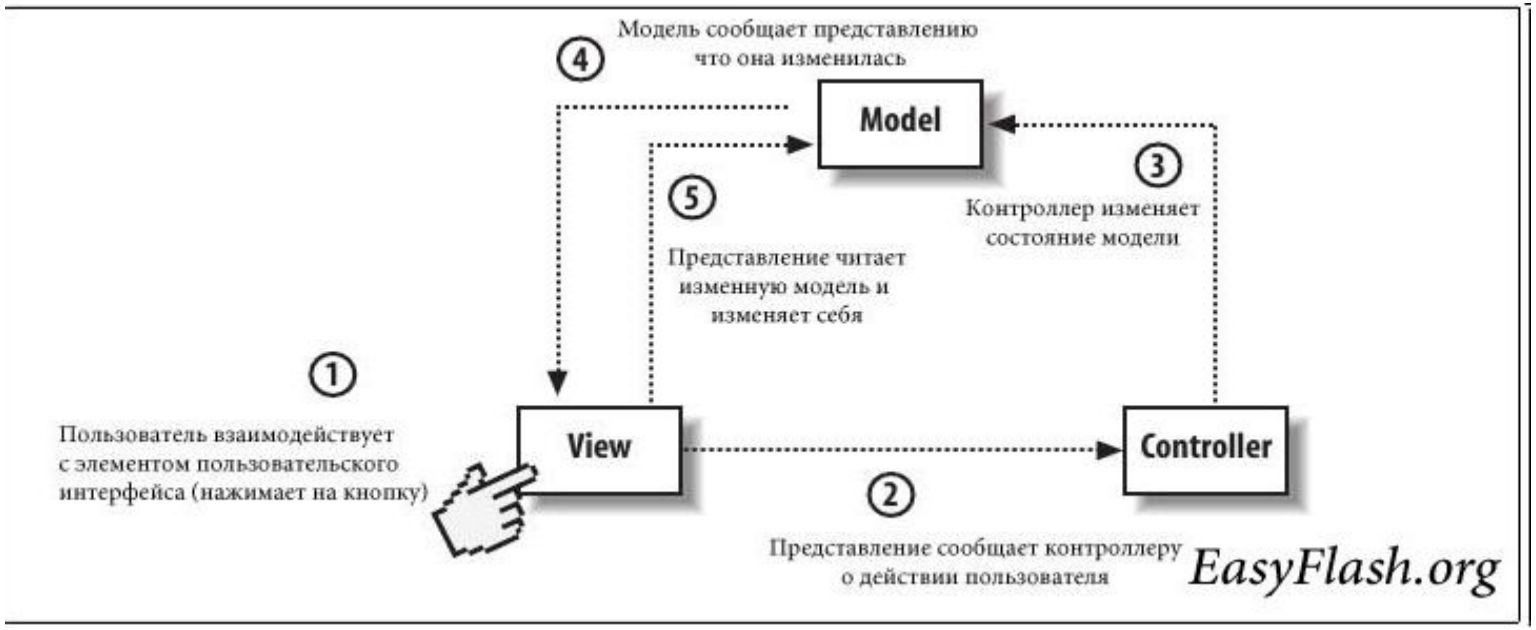
Паттерн проектирования MVC

Паттерн MVC (Model-View-Controller) применяется при разработке интерактивных приложений (в том числе web-приложений) с гибким пользовательским интерфейсом.

Иногда используется русскоязычное название «Данные-Представление-Обработка».



Объектно-ориентированное проектирование



Взаимодействие между элементами паттерна MVC

1. Пользователь взаимодействует с элементом пользовательского интерфейса (например, нажимает на кнопку в представлении);
2. Представление отправляет событие о нажатии на кнопке в контроллер для решения, как отреагировать на это событие;
3. Контроллер изменяет модель.
4. Модель информирует представление о том, что модель была изменена.
5. Представление читает информацию изменённой модели и изменяет себя.