Подготовительная программа по программированию на C/C++

Занятие №6

Валентина Глазкова

Объектно-ориентированное программирование

- Методология объектно-ориентированного подхода
- Объектно-ориентированное моделирование
- Унифицированный язык моделирования UML
- Разработка и анализ требований к программной системе

Регистрация на лекциях!

Объектно-ориентированный подход

Объектно-ориентированный подход использует объектную декомпозицию, при этом статическая структура программной системы описывается в терминах объектов реального мира и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами

Объектно-ориентированный подход

- При объектно-ориентированном подходе эти объекты (предметы и понятия) реального мира заменяются *моделями*, т.е. определенными формальными конструкциями.
- *Модель* содержит не все признаки и свойства представляемого ею предмета или понятия, а только те, которые существенны для разрабатываемой программной системы.

Объектно-ориентированный подход

- Простота модели по отношению к реальному предмету позволяет сделать ее формальной
- Благодаря такому характеру моделей при разработке можно четко выделить все зависимости и операции над ними в создаваемой программной системе
- Это упрощает как разработку и изучение (анализ) моделей, так и их реализацию на языке программирования

Объектно-ориентированный подход

- **OOA (object oriented analysis)**
  объектно-ориентированный анализ
- **OOD (object oriented design)**
  объектно-ориентированное проектирование
- **OOP (object oriented programming)**
  объектно-ориентированное программирование

Методология ООП

- Снижает сложности разработки программного обеспечения
- Повышает надежность программного обеспечения
- Обеспечивает возможность модификации отдельных компонентов программного обеспечения без изменения остальных его компонентов
- Обеспечивает возможность повторного использования отдельных компонентов программного обеспечения

Методология ООП

Объект **—** это мыслимая или реальная сущность, обладающая характерным поведением и отличительными характеристиками и являющаяся важной в предметной области (Гради Буч «Объектно-ориентированный анализ и проектирование»)

Методология ООП

- Каждый объект имеет определенное время жизни
- В процессе выполнения программы, или функционирования какой-либо реальной системы, могут создаваться новые объекты и уничтожаться уже существующие
- Каждый объект имеет состояние, обладает четко определенным поведением и уникальной идентичностью

Методология ООП

- ***Поведение (behavior)*** — действия и реакции объекта, выраженные в терминах передачи сообщений; видимая извне и воспроизводимая активность объекта
- ***Уникальность (identity)*** — свойство объекта; то, что отличает его от других объектов

Методология ООП

- ***Состояние (state)*** — совокупный результат поведения объекта: одно из стабильных условий, в которых объект может существовать и охарактеризованных количественно (Гради Буч)
- В любой момент времени состояние объекта включает в себя перечень (обычно статический) свойств объекта и текущие значения (обычно динамические) этих свойств

Методология ООП

- ***Класс*** **—** это шаблон поведения объектов определенного типа с заданными параметрами, определяющими состояние
- Все экземпляры одного класса (объекты, порожденные от одного класса) имеют один и тот же набор свойств и общее поведение, то есть одинаково реагируют на одинаковые сообщения

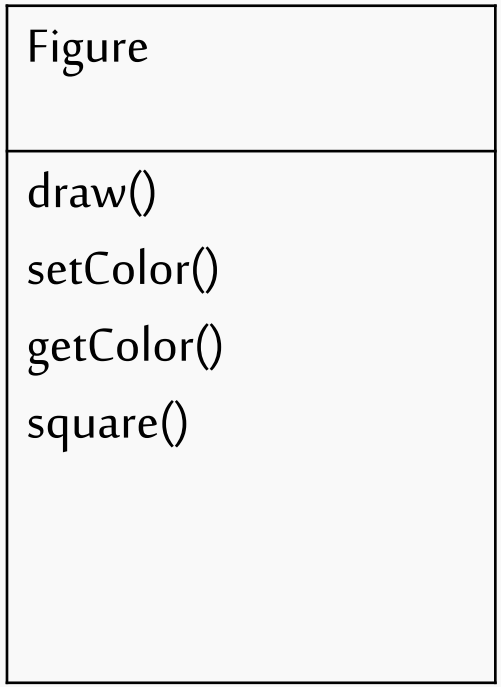
Класс и интерфейс

- Каждый объект может исполнить только определенный запрос
- Запросы, которые можно посылать объекту, определяются его *интерфейсом*, задаваемом в типе объекта (класса)

Класс и интерфейс

Имя типа –>

Интерфейс ->



Принципы объектного подхода

- ***Абстрагирование*** **—** выделяем главное, выявляем виды абстракций
- ***Инкапсуляция*** **—** скрываем детали реализации
- ***Иерархия*** **—** иерархия помогает разбить задачу на уровни и постепенно ее решать
- ***Агрегация и наследование*** **—** абстракции можно создавать на основе имеющихся
- ***Полиморфизм*** **—** полиморфизм позволяет действиям иметь естественные имена

Инкапсуляция

- ***Инкапсуляция*** ***(encapsulation)*** — это механизм, который связывает код вместе с обрабатываемыми им данными и сохраняет их в безопасности как от внешнего влияния, так и от ошибочного использования.
- Это защитная оболочка, которая предохраняет код и данные от произвольного доступа из других кодов, определенных вне этой оболочки

Инкапсуляция

- ***Инкапсуляция*** **—** это сокрытие реализации класса и отделение его внутреннего представления от внешнего (интерфейса), который обеспечивает и контролирует доступ к данным и коду внутри капсулы
- Доступен только интерфейс объекта, через который осуществляется все взаимодействие с ним
- Для доступа к свойствам класса принято задействовать специальные методы этого класса для получения и изменения его свойств

Наследование

- ***Наследование (inheritance)*** - это отношение между классами, при котором класс использует структуру или поведение другого класса (одиночное наследование), или других (множественное наследование) классов.

Наследование

- Наследование вводит иерархию "*общее/частное*", в которой подкласс наследует от одного или нескольких более общих суперклассов их свойства и методы
- Подклассы обычно дополняют или *переопределяют* унаследованную структуру и поведение
- Использование наследования способствует уменьшению количества кода, созданного для описания схожих сущностей, а также способствует написанию более эффективного и гибкого кода.

Полиморфизм

- Свойство, позволяющее называть типизировано разные, но алгоритмически схожие алгоритмические действия одним именем, называется ***полиморфизмом***
- Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент *повторного использования кода*

Отношения между классами

- Между *классами* возможны различные отношения:
  - ***зависимости***, которые описывают существующие между *классами* отношения использования;
  - ***обобщения (наследование),*** связывающие обобщенные *классы* со специализированными;
  - ***ассоциации (агрегирование),*** отражающие структурные отношения между объектами *классов*.

Объектно-ориентированное моделирование

- Для визуального моделирования структуры классов и отношений между ними нужна специальная нотация или язык
- UML (unified modeling language) — это **язык** для
  - визуализации,
  - специфицирования,
  - конструирования,
  - документирования

  элементов программных систем
- UML — язык общего назначения, предназначенный для объектного моделирования.

Объектно-ориентированное моделирование

Преимущества *визуального моделирования*

- **Визуализация** упрощает понимание проекта в целом.
- **Визуализация** помогает согласовать терминологию и убедиться, что все одинаково понимают термины.
- **Визуализация** делает обсуждение конструктивным и понятным.

В результате OOA & OOD мы получаем «хороший» проект программной системы, прозрачный, удовлетворяющий требованиям, удобный для тестирования и отладки, коллективной разработки, развиваемый, допускающий повторное использование компонентов

Унифицированный язык моделирования
UML

- **Унифицированный язык моделирования UML (Unified Modeling Language)** предназначен для определения, представления, проектирования и документирования программных систем различных сфер назначения
- UML содержит стандартный набор диаграмм и нотаций

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

Диаграммы классов

- Классы представляют собой описание совокупностей однородных объектов с присущими им свойствами — атрибутами, операциями, отношениями и семантикой.
- В рамках модели каждому *классу* присваивается уникальное имя, отличающее его от других *классов*
- ***Атрибут*** — это свойство *класса*, которое может принимать множество значений.
- ***Операция*** — реализация функции, которую можно запросить у любого объекта *класса*

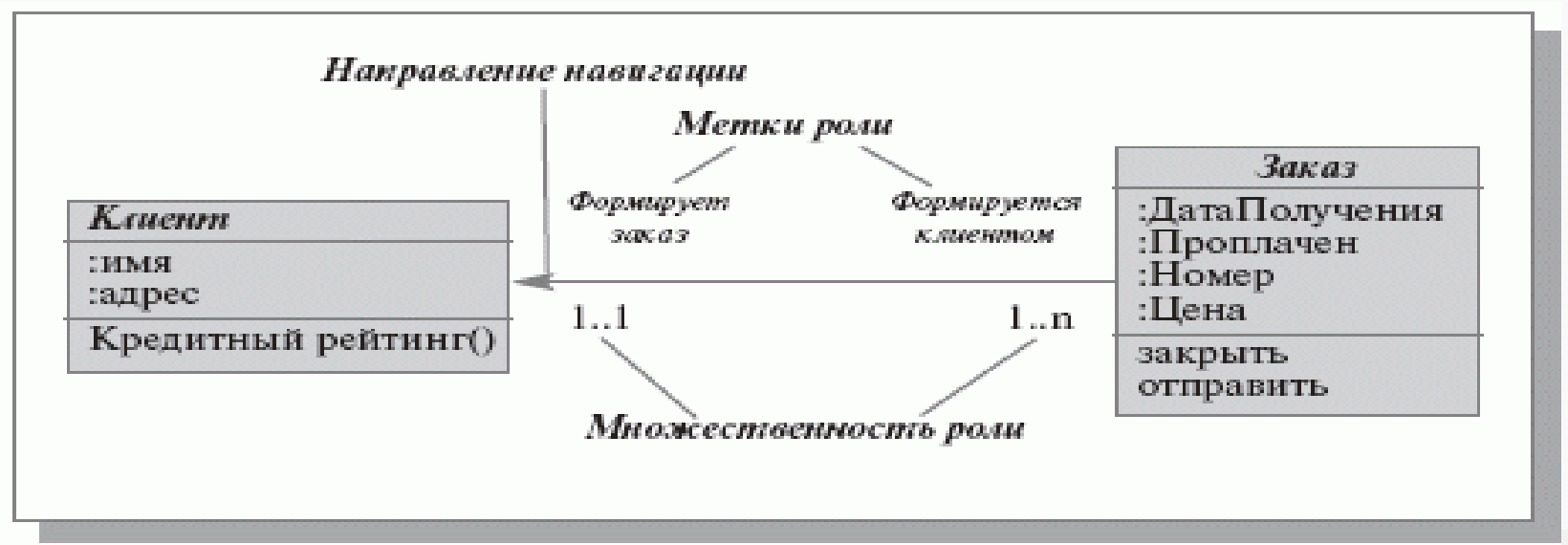
Диаграммы классов

Обобщение — это отношение между общей сущностью (родителем) и ее конкретным воплощением (потомком)

Зависимостью называется отношение использования, согласно которому изменение в спецификации одного элемента может повлиять на использующий его элемент

Диаграммы классов

- ***Ассоциация*** — это отношение, показывающее, что объекты одного типа неким образом связаны с объектами другого типа



Диаграммы использования

- Диаграммы использования описывают функциональность ПС, которая будет видна внешним пользователям системы. «Каждая функциональность» изображается в виде «прецедентов использования» (use case)
- **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.

Диаграммы использования

Связь типа «расширение» применяется, когда один прецедент подобен другому, но несет несколько большую функциональную нагрузку.

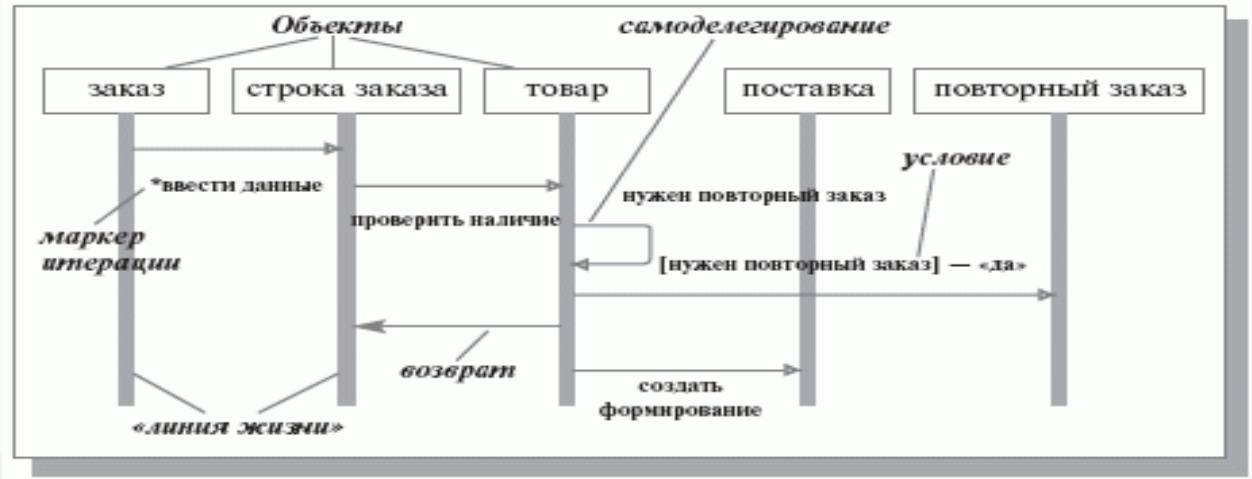
Связь типа «использование» позволяет выделить некий фрагмент поведения системы и включать его в различные прецеденты без повторного описания.

Диаграммы взаимодействия

- *UML* отделяет описание поведения в ***диаграммы взаимодействия***
- Поток сообщений между объектами выносится на ***диаграммы взаимодействия***
- ***Диаграмма взаимодействия*** охватывает поведение объектов в рамках одного варианта использования
- Существуют два вида ***диаграмм взаимодействия***:
  - диаграммы последовательностей
  - кооперативные диаграммы

Диаграммы последовательностей

- ***Диаграммы последовательностей*** используется для точного определения логики сценария выполнения прецедента. Они отображают типы объектов, взаимодействующих при исполнении прецедентов, сообщения, которые они посылают друг другу, и любые возвращаемые значения, ассоциированные с этими сообщениями



Кооперативные диаграммы

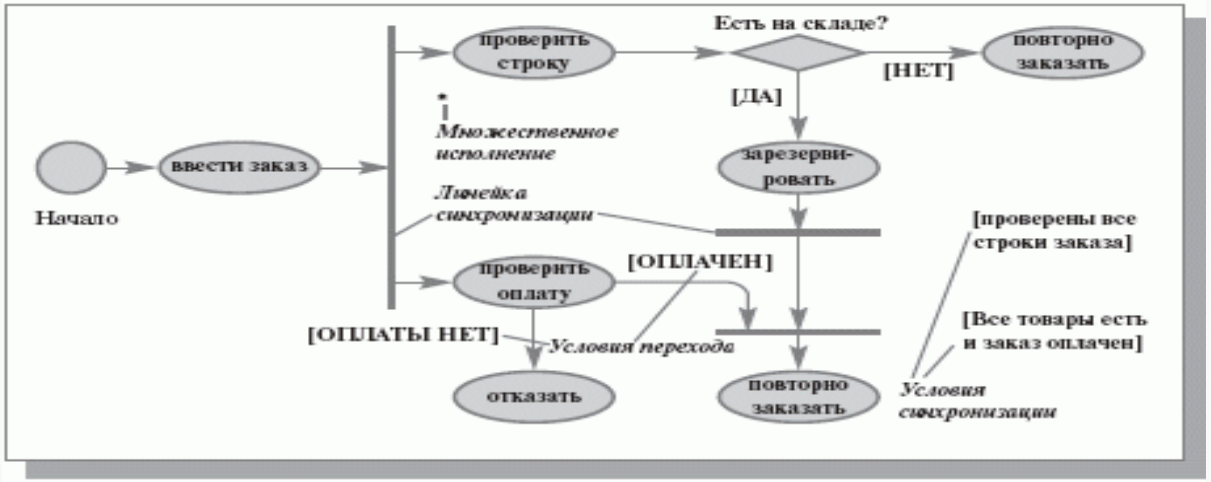
- На ***кооперативных диаграммах*** объекты (или ***классы***) показываются в виде прямоугольников, а стрелками обозначаются сообщения, которыми они обмениваются в рамках одного варианта использования. Временная последовательность сообщений отражается их нумерацией.

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. Они обычно используются для описания поведения одного объекта в нескольких прецедентах

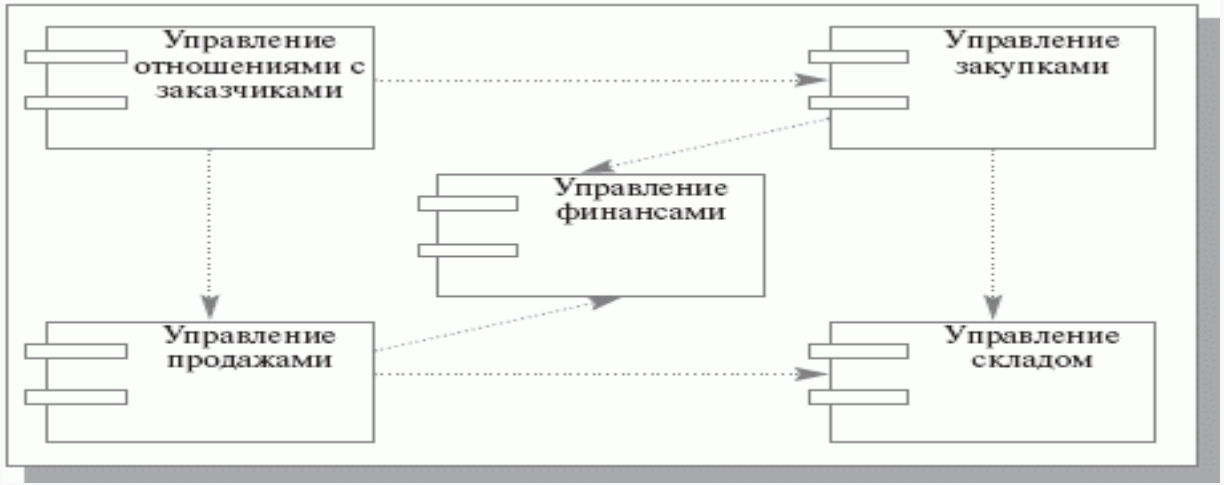
Диаграммы деятельности

- 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.



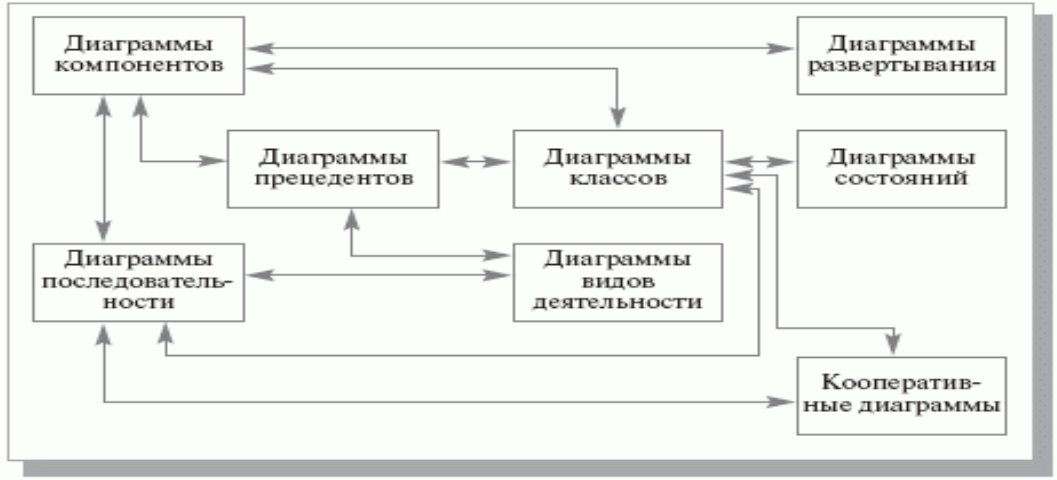
Диаграммы компонентов

- ***Диаграммы компонентов*** позволяют изобразить модель системы на физическом уровне
- Элементами диаграммы являются компоненты — физические замещаемые модули системы.



Разработка ПО

- UML обеспечивает поддержку всех этапов жизненного цикла ПС и предоставляет для этих целей ряд графических средств – диаграмм



Разработка модели бизнес-прецедентов

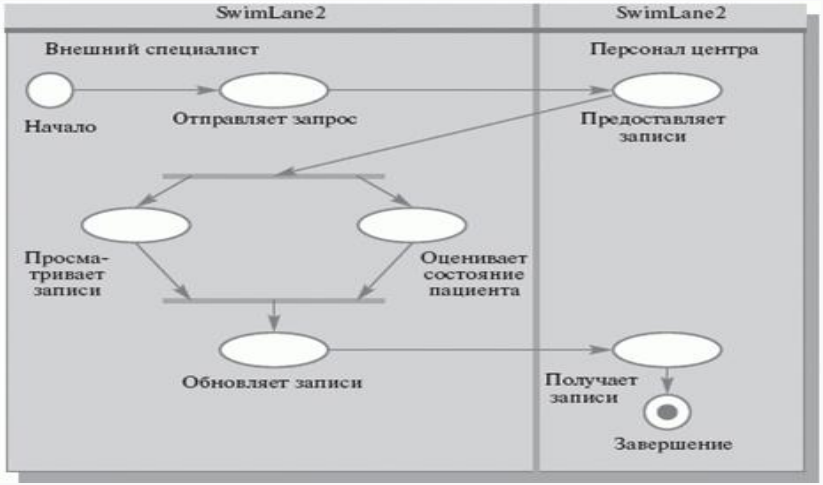
- ***Модель бизнес-прецедентов*** описывает бизнес-процессы с точки зрения внешнего пользователя, т.е. отражает взгляд на деятельность организации извне.

Разработка модели бизнес-прецедентов

- Для включения в диаграмму выбранные прецеденты должны удовлетворять следующим критериям:
  - прецедент должен описывать, **ЧТО** нужно делать, а не **КАК**;
  - прецедент должен описывать действия с точки зрения **ИСПОЛНИТЕЛЯ**;
  - прецедент должен возвращать исполнителю некоторое **СООБЩЕНИЕ**;
  - последовательность действий внутри прецедента должна представлять собой одну **НЕДЕЛИМУЮ** цепочку.

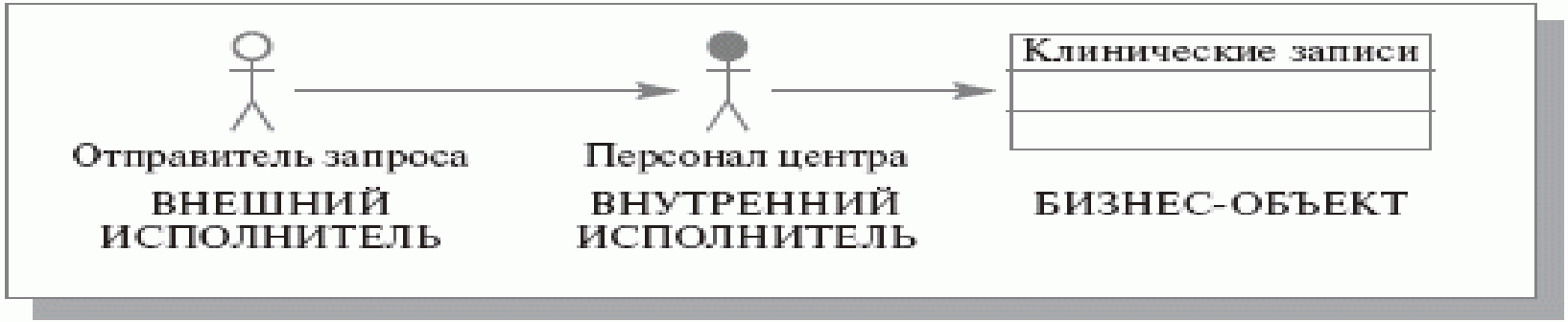
Разработка модели бизнес-прецедентов

- Выполнение прецедента описывается с помощью диаграмм видов деятельности, которые отображают исполнителей и последовательность выполнения соответствующих бизнес-процессов



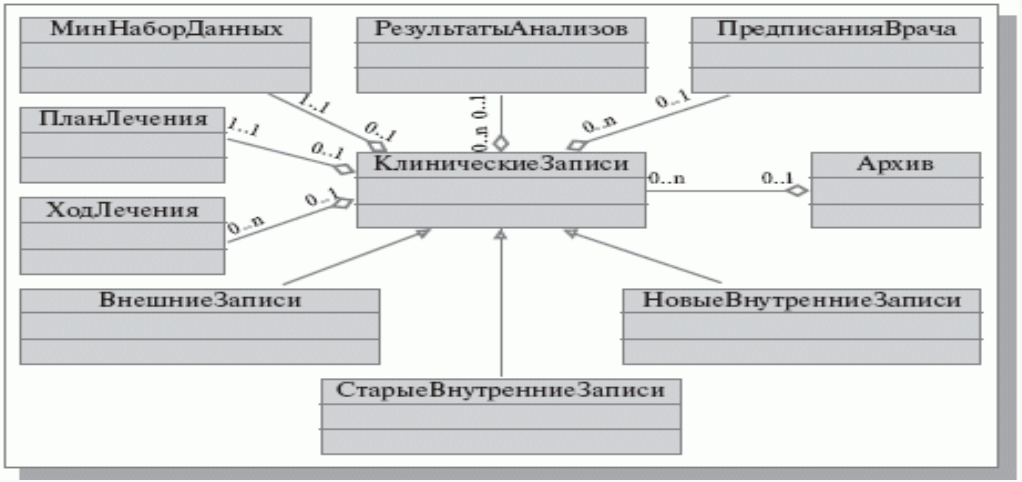
Разработка модели бизнес-объектов

- Следующим этапом проектирования ИС является разработка *модели бизнес-объектов*, которая показывает выполнение бизнес-процессов организации ее внутренними исполнителями. Основными компонентами ***моделей бизнес-объектов*** являются внешние и внутренние исполнители, а также бизнес-сущности, отображающие все, что используют внутренние исполнители для реализации бизнес-процессов.



Разработка концептуальной модели данных

- На основе информации, выявленной на этапах бизнес-моделирования, выполняется разработка ***концептуальной модели данных***, которые будут использоваться в разрабатываемой системе.



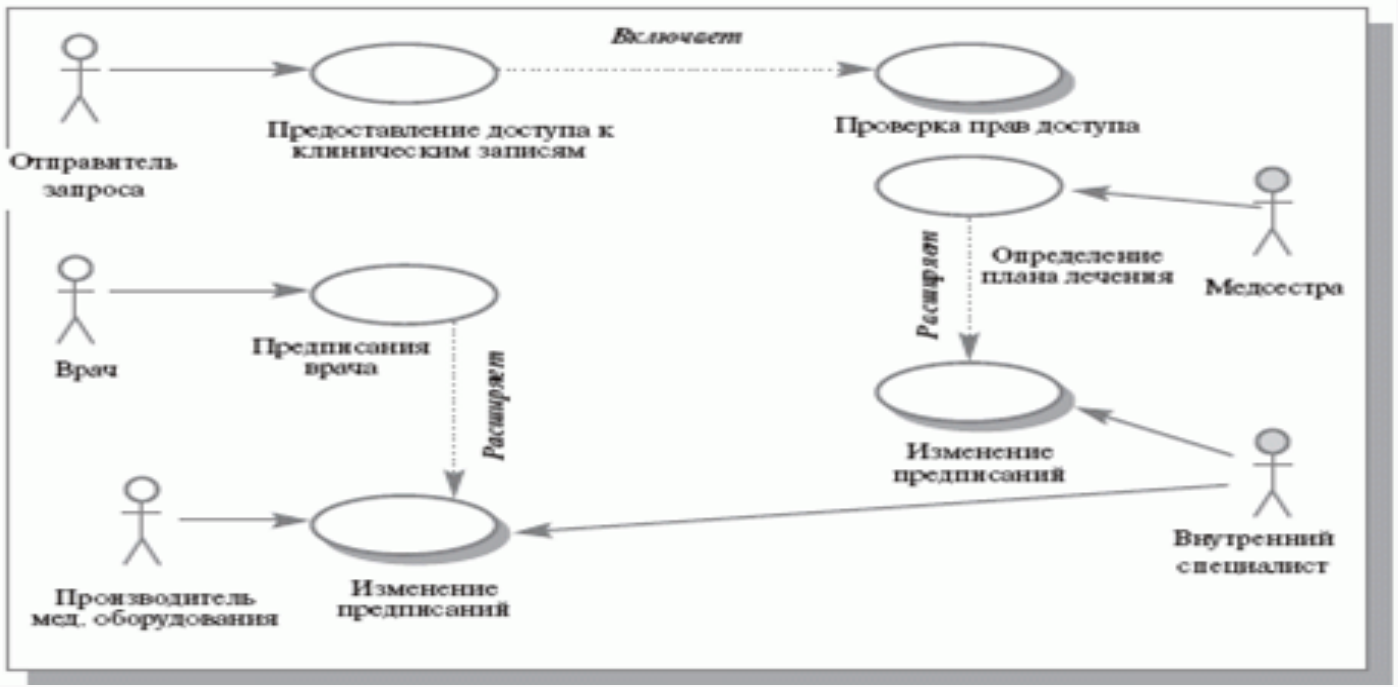
Разработка требований к системе

- На этапе формирования требований, прежде всего, необходимо определить область действия разрабатываемой системы и получить точное представление о желаемых возможностях системы.
- Основой разработки требований является ***модель системных прецедентов (вариантов использования)***, отражающая выполнение конкретных обязанностей внутренними и внешними исполнителями с использованием ПС

Разработка требований к системе

- Источником данных для создания *модели системных прецедентов* являются разработанные на предыдущем этапе бизнес-модели.
- При создании модели полезно предварительно составить детальные описания прецедентов, содержащие определения используемых данных и точную последовательность их выполнения. Описание обычно включает следующие разделы:
  - заголовок (название прецедента, ответственный за исполнение, дата создания шаблона/внесения изменений);
  - краткое описание прецедента;
  - ограничения;
  - предусловия (необходимое состояние системы или условия, при которых должен выполняться прецедент);
  - постусловия (возможные состояния системы после выполнения прецедента);
  - предположения;
  - основная последовательность действий;
  - альтернативные последовательности действий и условия, их инициирующие;
  - точки расширения и включения прецедентов.

Модель системных прецедентов



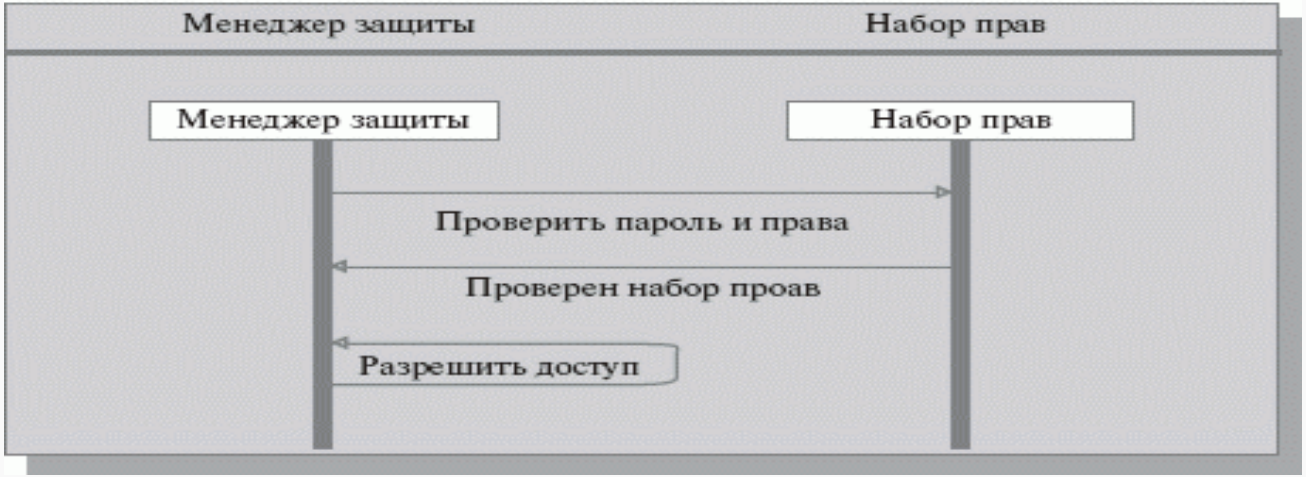
Модель системных прецедентов

- В процессе создания ***модели системных прецедентов*** осуществляется преобразование и перенос компонентов бизнес-моделей на новые диаграммы

Элементы бизнес-модели	Элементы модели системных прецедентов
Бизнес-прецеденты	Подсистемы
Внешние исполнители	Исполнители
Внутренние исполнители	Исполнители или прецеденты
Процессы, выполняемые внутренними исполнителями	Прецеденты

Анализ требований и предварительное проектирование системы

- Для каждого варианта использования разрабатывается диаграмма последовательности, описывающая его исполнение

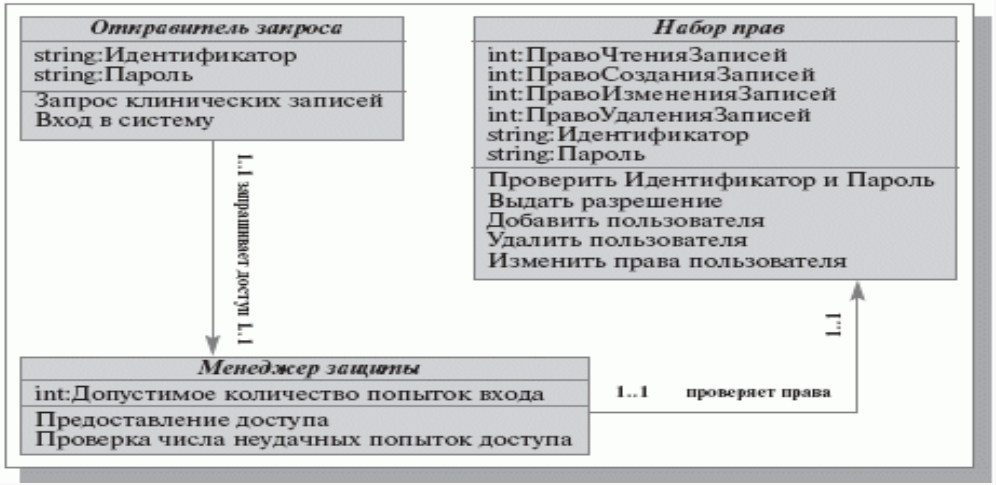


Анализ требований и предварительное проектирование системы

- Основным инструментом на данном этапе являются диаграммы классов системы, которые строятся на основе разработанной ***модели системных прецедентов***.
- Одновременно на этом этапе уточняются диаграммы последовательностей выполнения отдельных прецедентов, что приводит к изменениям в составе объектов и диаграммах классов.

Анализ требований и предварительное проектирование системы

- Диаграммы классов системы заполняются объектами из ***модели системных прецедентов***. Они содержат описание этих объектов в виде классов и описание взаимодействия между классами.

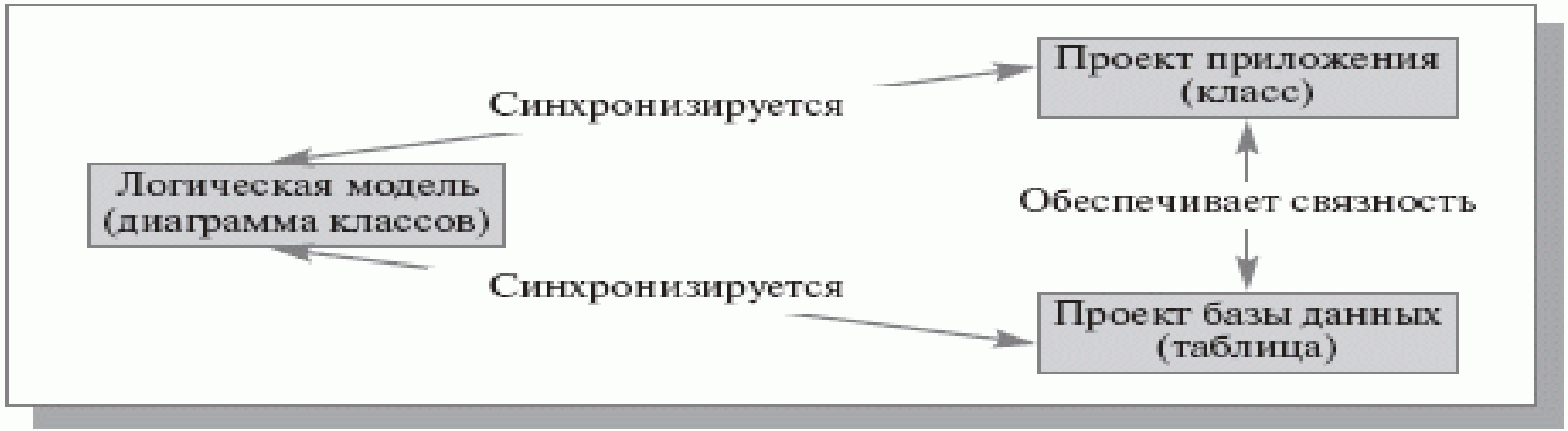


Разработка модели базы данных

- На этом этапе осуществляется отображение элементов полученных ранее моделей классов в элементы моделей базы данных:
  - классы отображаются в таблицы;
  - атрибуты – в столбцы;
  - типы – в типы данных используемой СУБД;
  - ассоциации – в связи между таблицами (ассоциации «многие-ко-многим» преобразуются в ассоциации «один-ко-многим» посредством создания дополнительных таблиц связей)

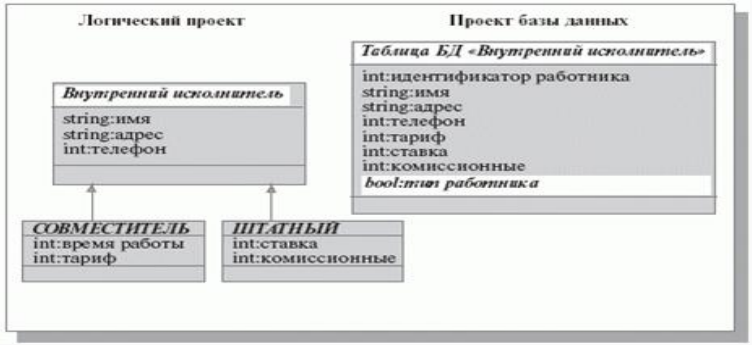
Разработка моделей базы данных

- Поскольку модель базы данных строится на основе единой логической модели, автоматически обеспечивается связность этих проектов



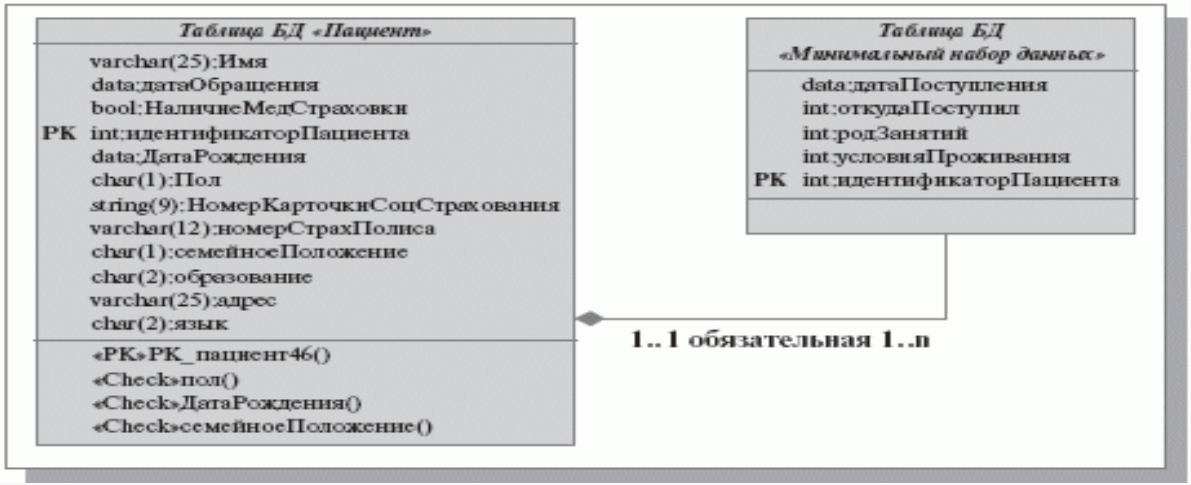
Разработка модели базы данных

- Отображение классов подтипов в таблицы осуществляется одним из стандартных способов:
  - одна таблица на класс;
  - одна таблица на иерархию



Разработка модели базы данных

- Результатом этапа является детальное описание проекта базы данных системы



Сопровождение ПО

70% стоимости ПО приходится на его сопровождение

Никакое изучение
качества ПО
не может быть удовлетворительным,
если оно игнорирует
этот аспект

Валентина Глазкова

Спасибо за внимание!