Программная инженерия

# Анализ и Проектирование

# Проектирование ПО

# Что такое проектирование ПО?

- **Проектирование ПО** – это осознанный выбор решений о логической организации составных частей программного комплекса.
- Проект может быть представлен в виде модели UML, неформального документа или набросков представляющих определенные аспекты дизайна.
- Проектирование – это творческий процесс, который тяжело описать в виде формальных структурированных методов.

# Проектирование архитектуры

Разработка и документация архитектуры ПО обеспечивает следующие преимущества:

1. **Взаимодействие с заказчиком**
2. **Системный анализ**
3. **Высокоуровневое повторное использование ПО**

# Влияние нефункциональных требований на архитектуру ПО

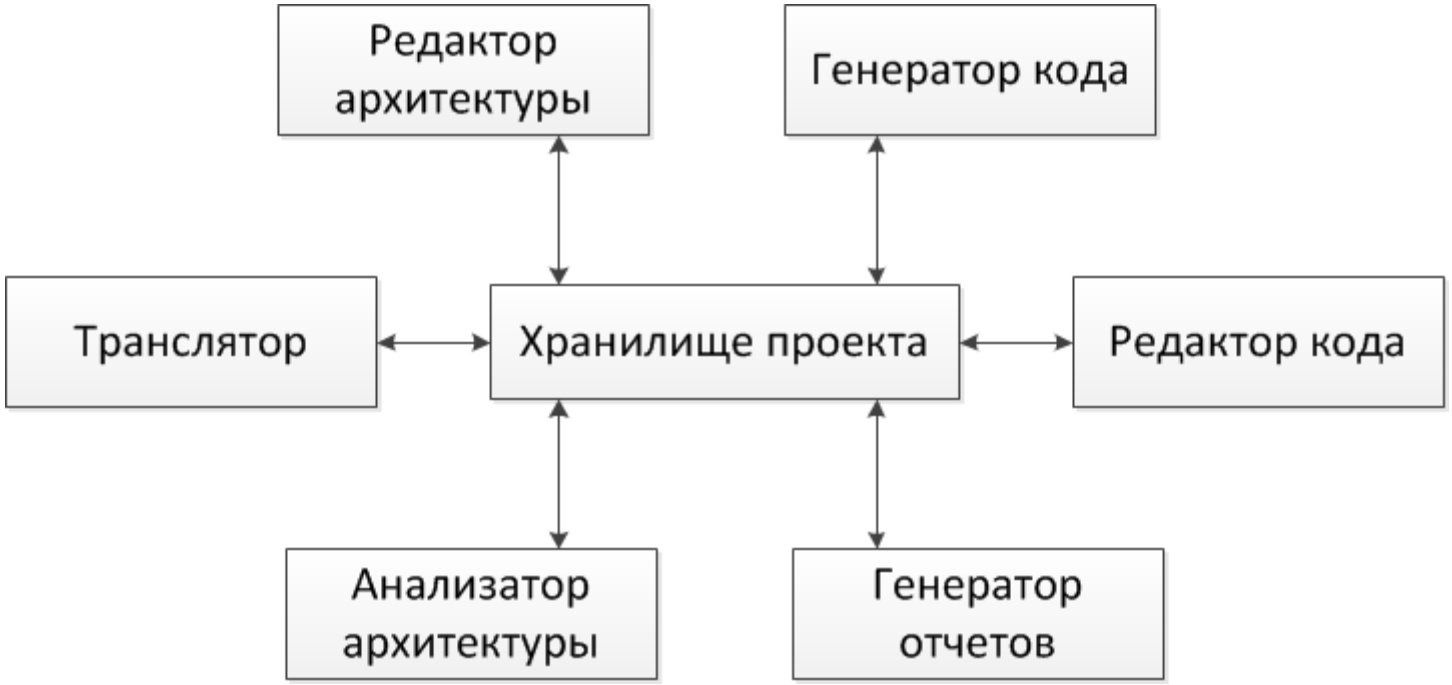
- **Производительность** – локализация критически-важных операций
- **Безопасность** – слоистая архитектура
- **Доступность** – избыточность используемых компонентов
- **Поддерживаемость** – мелкомодульная архитектура слабосвязанных компонентов.

# Общие методы системной организации

- **Системная организация** описывает общую стратегию, используемую для структурирования системы.
- Выделяют следующие модели системной организации:
  - Модель репозитория
  - Клиент-серверная модель
  - Многослойная модель

# Модель репозитория (общего хранилища)

- Весь объем данных программной системы хранится в едином **общем хранилище (базе данных)**, доступ к которому может быть получен из любого программного компонента



# Достоинства модели репозитория

- Эффективный метод обработки больших объемов данных
- Нет проблемы с обменом данными между компонентами
- Резервное копирование, безопасность, ограничение доступа централизованы
- Модель использования общих данных ясно видна из схемы репозитория

# Недостатки модели репозитория

- Все компоненты должны подчиняться модели данных репозитория
- Проблемы развития системы, масштабируемости, изменения модели
- Невозможно установить политику безопасности или резервного копирования для каждого отдельного компонента
- Затруднительно распределение репозитория по нескольким машинам

# Клиент-серверная модель

- Система организуется в виде набора серверов, предоставляющих определенные сервисы клиентам. Основные компоненты модели:
  - Набор серверов, предоставляющих сервисы другим подсистемам
  - Набор клиентов, использующих сервисы
  - Промежуточная среда, обеспечивающая взаимодействие между клиентом и сервером

# Пример клиент-серверной модели

# Достоинства и недостатки клиент-серверной модели

- Клиент-серверная модель опирается на распределенную архитектуру и эффективно реализуется в РВС.
- Легко можно добавлять новые сервера или обновлять старые без заметных изменений в инфраструктуре.
- Но невозможно так же легко разделять общие данные между клиентами и серверами. В связи с этим необходимо применять специальные форматы данных для обмена.

# Многослойная модель (модель абстрактной машины)

- Система распределяется на слои, каждый из которых представляет набор определенных сервисов.
- Каждый слой можно представить в виде «абстрактной машины» язык которой определяется набором сервисов, предоставляемых на соответствующем слое.

# Пример многослойной модели

# Достоинства и недостатки многослойной модели

- Поддержка инкрементальной разработки
- Возможность замены слоев
- При изменении затрагиваются только смежные слои

НО

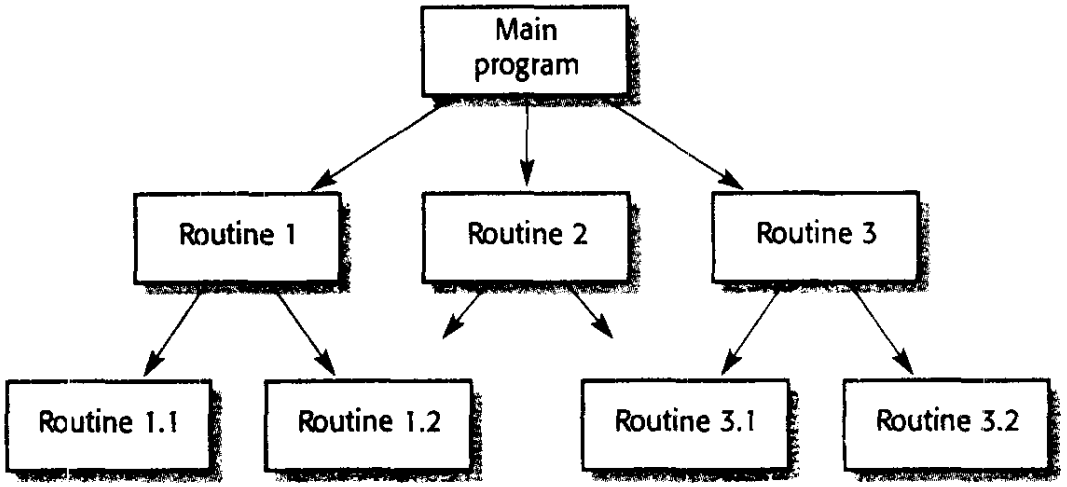
- Сложность организации архитектуры
- Низкая скорость работы

# Методы модульной декомпозиции

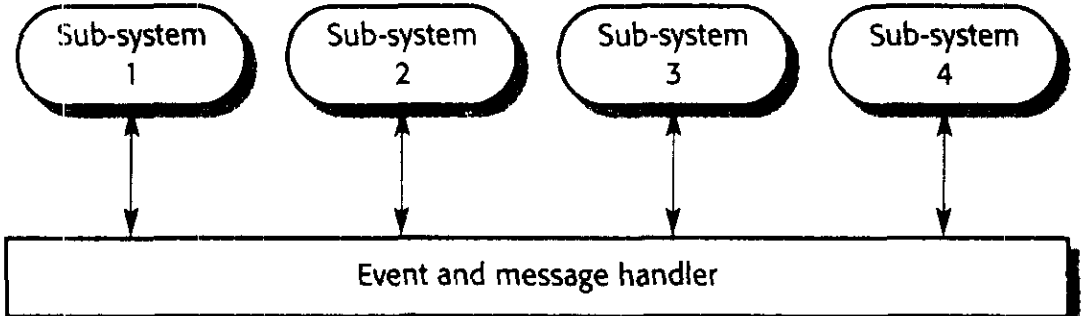
- Объектно-ориентированная декомпозиция: система представляется в виде набора взаимодействующих объектов.
- Функционально-ориентированная декомпозиция: система представляется в виде функциональных модулей, на вход которых поступают одни входные данные, а выходят выходные.

# Методы организации контроля

- **Системы с централизованным контролем:** одна подсистема отвечает за контроль, запускает и останавливает другие подсистемы.
- **Событийно-ориентированные системы**: каждая подсистема может отвечать на внешние события (из других подсистем или из окружающей среды)




Централизованная система: «Запрос - ответ»

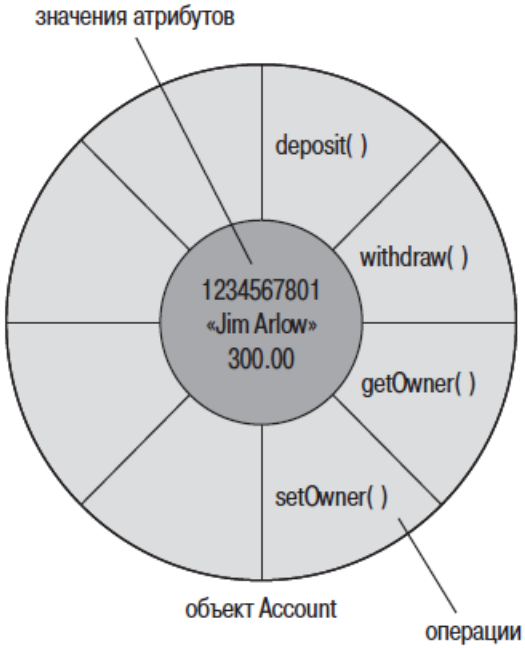



Событийно-ориентированные система

# Объектно-ориентированное проектирование

# ОО-ПРОЕКТИРВОАНИЕ

- ОО-проектирование основывается на концепции объекта: сущности, сохраняющей свое внутреннее состояние и поддерживающей операции для управления внутренним состоянием.

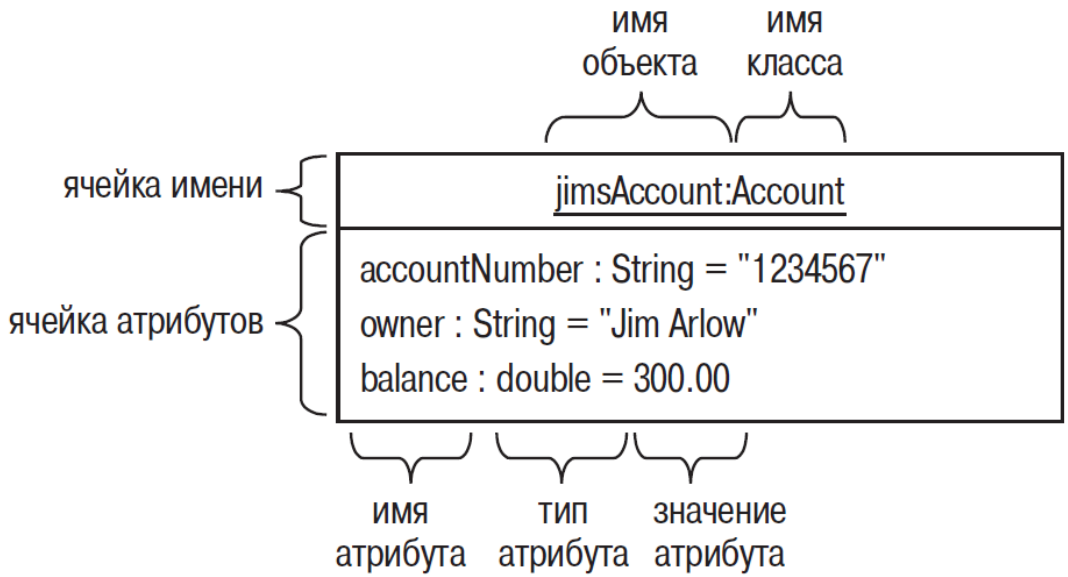


# Процесс объектно-ориентированного решения задачи

- ОО анализ – формирование объектно-ориентированной модели предметной области.
- ОО проектирование – развитие ОО модели программной системы, для достижения определенных требований.
- ОО реализация – реализация разработанной модели посредством ОО языка программирования.

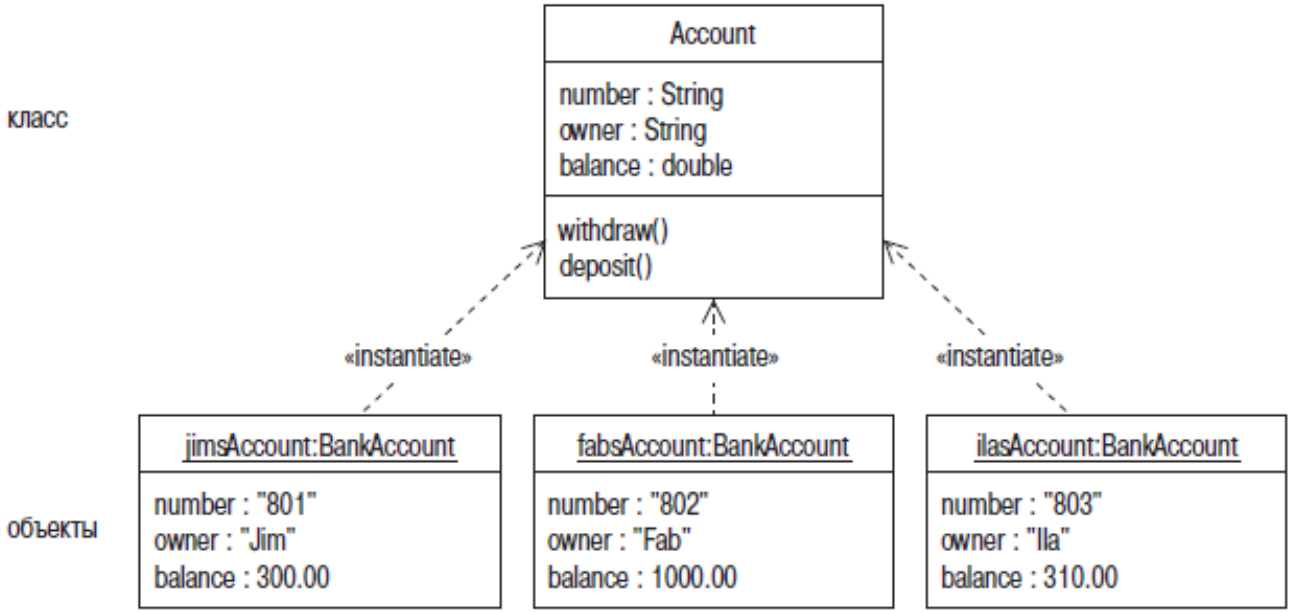
# Нотация объектов UML

- Прямоугольник с двумя ячейками:
  - Идентификатор объекта (подчеркнутый) и/или имя класса через двоеточие
  - Блок атрибутов (по выбору, т.к. набор атрибутов определяет класс)



# Классы и объекты UML

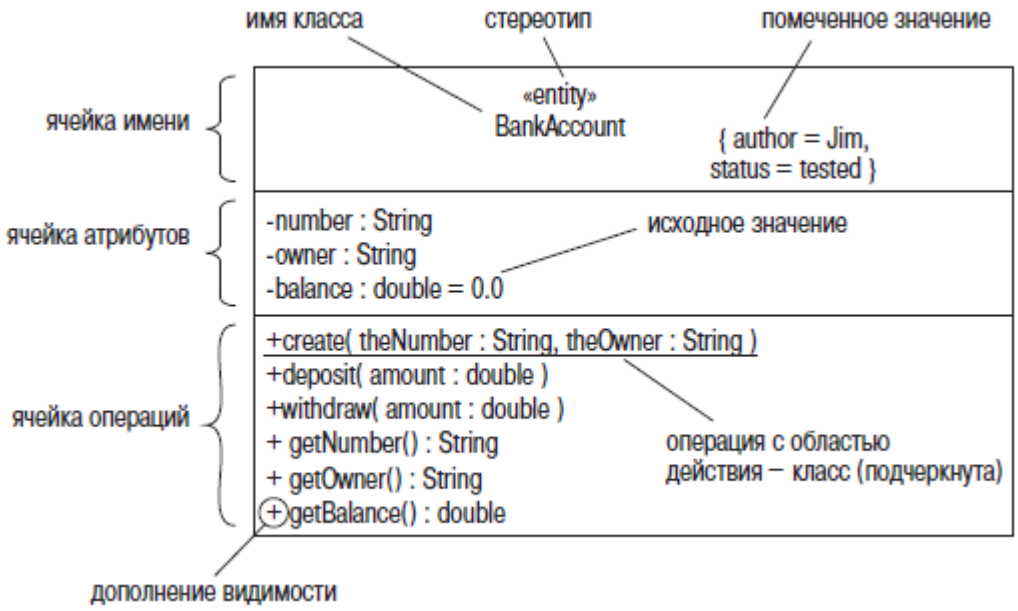
- Между классом и объектами этого класса устанавливается отношение «instantiate» (создать экземпляр)




- Отношение зависимости означает, что изменение сущности поставщика оказывает влияние на сущность клиент

# Нотация классов UML

- Обязательной частью в визуальном синтаксисе является только ячейка с именем класса. Все остальные ячейки и дополнения необязательны.



# Нотация атрибутов класса

- Единственная обязательная часть UML-синтаксиса для атрибута является его имя.

видимость имя : тип [множественность] = начальноеЗначение

обязателен

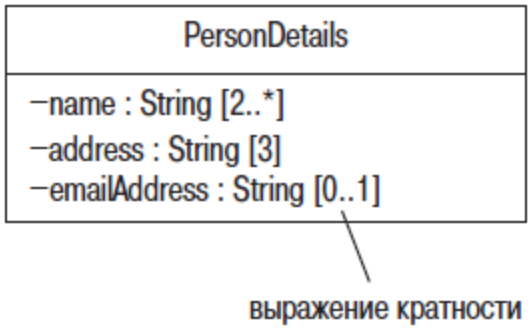
## Видимость:

| | |
|---|---|
| + | Public |
| - | Private |
| # | Protected |
| ~ | Package |

## Базовые типы:

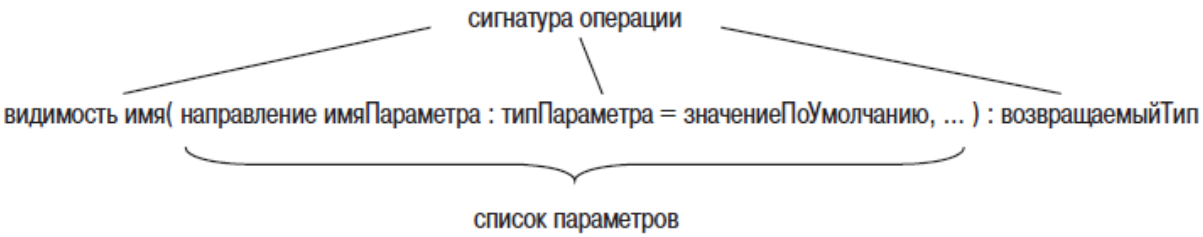
| |
|---|
| Integer |
| UnlimitedNatural |
| Boolean |
| String |
| Real |

## Множественность:



# Нотация операций

- Сигнатура операции включает имя, тип всех ее параметров и возвращаемый тип




- Направление параметра:
  - in (по умолчанию) – входной параметр
  - inout – параметр ввода/вывода
  - out – выходной параметр
  - return – возвращаемый параметр (UML 2.1 – только 1)

# Нотация области действия

- Иногда нужно определить атрибуты, которые имеют единственное, общее для *всех объектов класса значение.* И нужны операции (как операции создания объектов), не относящиеся ни к одному конкретному экземпляру класса. Говорят, что такие атрибуты и операции имеют *область действия – класс.*