

# Об архитектуре программного обеспечения

Юрий Литвинов  
y.litvinov@spbu.ru

21.05.2025

# Архитектура

- ▶ Совокупность важнейших решений об организации программной системы
  - ▶ Эволюционирующий свод знаний
  - ▶ Разные точки зрения
  - ▶ Разный уровень детализации
- ▶ Для чего
  - ▶ База для реализации, «фундамент» системы
  - ▶ Инструмент для оценки трудоёмкости и отслеживания прогресса
  - ▶ Средство обеспечения переиспользования
  - ▶ Средство анализа системы ещё до того, как она реализована

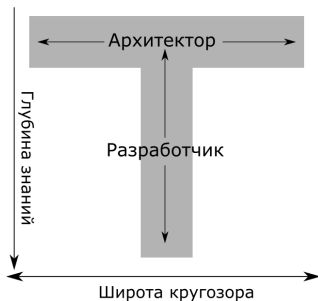


© Интернеты

# Профессия «Архитектор»

- ▶ Архитектор — человек (или группа людей), отвечающий за:
  - ▶ разработку и описание архитектуры системы
  - ▶ доведение её до всех заинтересованных лиц
  - ▶ контроль реализации архитектуры
  - ▶ поддержание её актуального состояния по ходу разработки и сопровождения

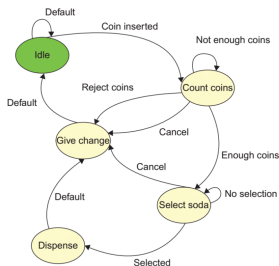
# Архитектор vs разработчик



- ▶ Широта знаний
- ▶ Коммуникационные навыки
- ▶ Часто архитектор играет роль разработчика и наоборот

# Моделирование ПО

- ▶ Основной продукт архитектора — архитектурная документация
- ▶ Модели — важная её часть
  - ▶ Предназначены прежде всего для управления сложностью
  - ▶ Могут моделировать как саму систему, так и окружение
  - ▶ Позволяют понять, проанализировать и протестировать систему до её реализации

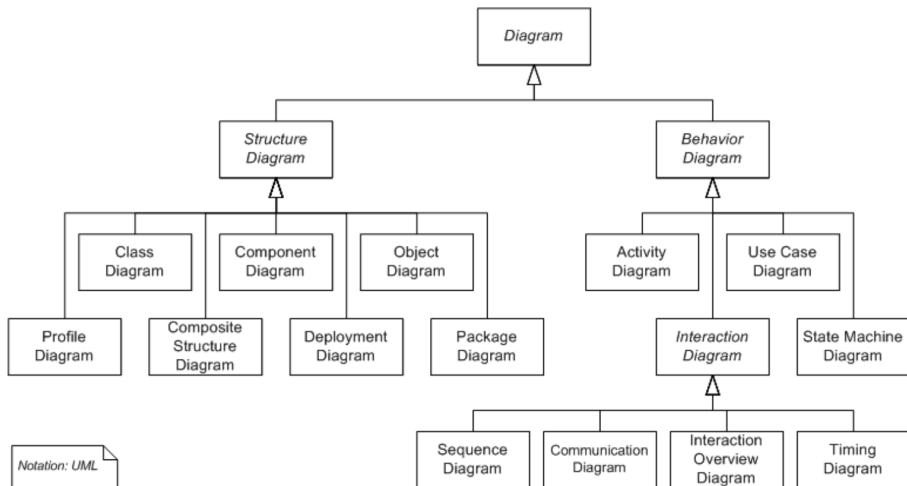


© N. Medvidovic

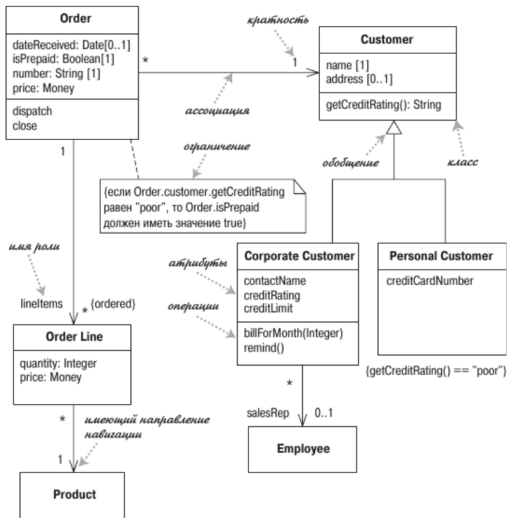
# Unified Modeling Language

- ▶ Семейство графических нотаций
  - ▶ 14 видов диаграмм
- ▶ Общая метамодель
- ▶ Стандарт под управлением Object Management Group
  - ▶ UML 1.1 — 1997 год
  - ▶ UML 2.0 — 2005 год
  - ▶ UML 2.5.1 — декабрь 2017 года
- ▶ Прежде всего, для проектирования ПО
  - ▶ После UML 2.0 стали появляться нотации и для инженеров
- ▶ Расширяем, но сложно

# Виды диаграмм

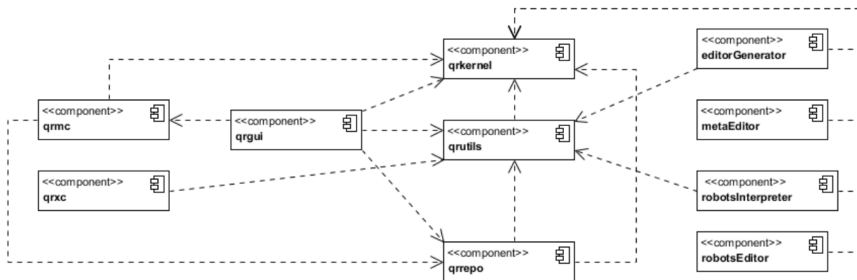


# Диаграмма классов





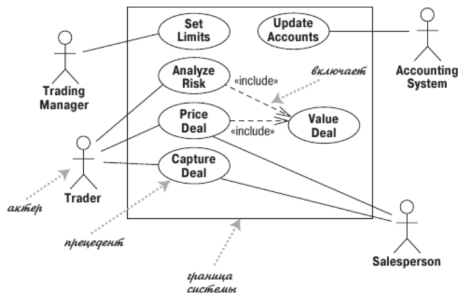
# Диаграмма компонентов



# Диаграмма случаев использования UML

## Диаграмма прецедентов

- ▶ Ивар Якобсон, 1992 год
- ▶ Актеры (или актёры, роли) — внешние сущности, использующие систему
  - ▶ Люди или другие программные системы
- ▶ Случаи использования (прецеденты) — цель использования системы актором
  - ▶ Раскрываются в набор сценариев, описываемых чаще текстом

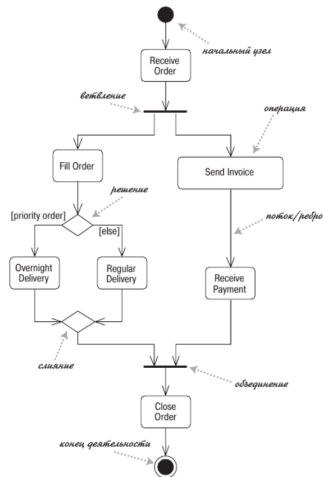


© М. Фаулер, UML. Основы

# Диаграмма активностей UML

## Диаграммы деятельности

- ▶ Используются для моделирования бизнес-процессов, тоже на первых этапах
  - ▶ Может быть визуализацией сценария использования
- ▶ Иногда — для моделирования алгоритма
- ▶ Расширенные блок-схемы



# Инструменты для рисования диаграмм

- ▶ «Рисовалки»
  - ▶ <https://diagrams.net/>
  - ▶ Visio
  - ▶ Dia
  - ▶ SmartDraw
  - ▶ LucidChart
  - ▶ <http://plantuml.com/>
- ▶ Полноценные CASE-системы
  - ▶ Visual Paradigm
  - ▶ Enterprise Architect
  - ▶ Rational Software Architect
  - ▶ MagicDraw
- ▶ Браузерные инструменты
  - ▶ <https://www.websequencediagrams.com/>
  - ▶ <http://yuml.me/>

# Принципы SOLID

- ▶ Single responsibility principle
- ▶ Open/closed principle
- ▶ Liskov substitution principle
- ▶ Interface segregation principle
- ▶ Dependency inversion principle

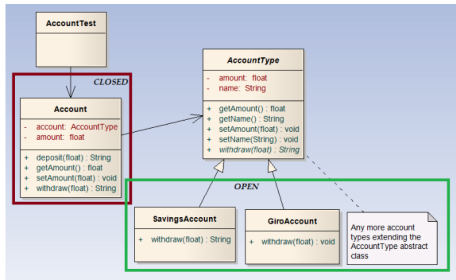
# Single responsibility principle

- ▶ Каждый объект должен иметь одну обязанность
- ▶ Эта обязанность должна быть полностью инкапсулирована в объект



# Open/closed principle

- ▶ Программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для изменения
  - ▶ Переиспользование через наследование
  - ▶ Неизменные интерфейсы



# Liskov substitution principle

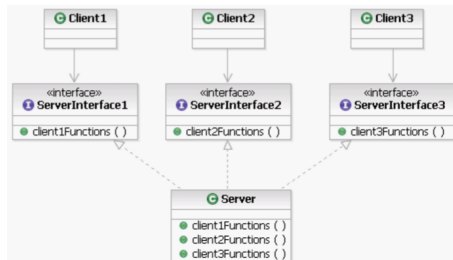
- ▶ Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом





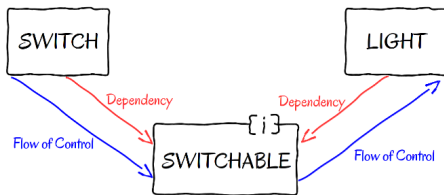
# Interface segregation principle

- ▶ Клиенты не должны зависеть от методов, которые они не используют
  - ▶ Слишком «толстые» интерфейсы необходимо разделять на более мелкие и специфические



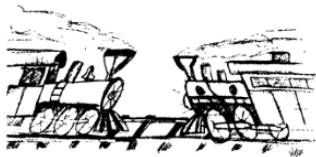
# Dependency inversion principle

- ▶ Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций
- ▶ Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций



# Закон Деметры

- ▶ «Не разговаривай с незнакомцами!»
- ▶ Объект А не должен иметь возможность получить непосредственный доступ к объекту С, если у объекта А есть доступ к объекту В, и у объекта В есть доступ к объекту С
- ▶ `book.pages().last().text()` vs `book.lastPageText()`
- ▶ Иногда называют «Крушение поезда»



© Р. Мартин, «Чистый код»

## Другие принципы

- ▶ Don't Repeat Yourself (DRY), он же «Копипаст суть ересь»
- ▶ Keep It Simple, Stupid (KISS)
- ▶ You Aren't Gonna Need It (YAGNI), он же «нет Big Design Up Front»

# Паттерны проектирования

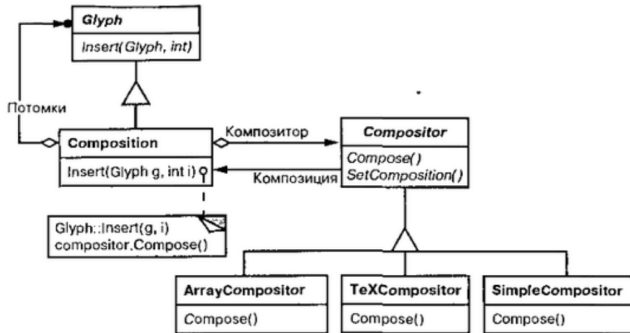
**Шаблон проектирования** — это повторяемая архитектурная конструкция, являющаяся решением некоторой типичной технической проблемы

- ▶ Подходит для класса проблем
- ▶ Обеспечивает переиспользуемость знаний
- ▶ Позволяет унифицировать терминологию
- ▶ В удобной для изучения форме
- ▶ НЕ конкретный рецепт или указания к действию

# Форматирование текста

- ▶ Задача — разбиение текста на строки, колонки и т.д.
- ▶ Высокоуровневые параметры форматирования
  - ▶ Ширина полей, размер отступа, межстрочный интервал и т.д.
- ▶ Компромисс между качеством и скоростью работы
- ▶ Инкапсуляция алгоритма

# Compositor и Composition

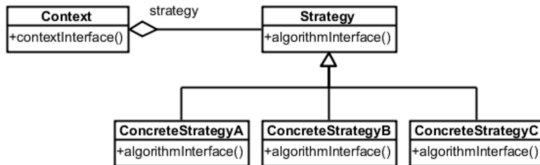


© Э. Гамма и др., Приемы объектно-ориентированного проектирования

# Паттерн «Стратегия»

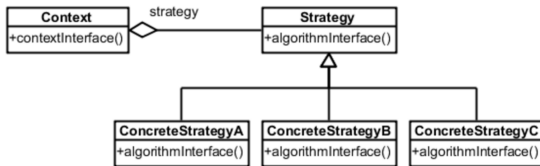
## Strategy

- ▶ Назначение — инкапсуляция алгоритма в объект
- ▶ Самое важное — спроектировать интерфейсы стратегии и контекста
  - ▶ Так, чтобы не менять их для каждой стратегии
- ▶ Применяется, если
  - ▶ Имеется много родственных классов с разным поведением
  - ▶ Нужно иметь несколько вариантов алгоритма
  - ▶ В алгоритме есть данные, про которые клиенту знать не надо
  - ▶ В коде много условных операторов





## «Стратегия» (Strategy), детали реализации



- ▶ Передача контекста вычислений в стратегию
  - ▶ Как параметры метода — уменьшает связность, но некоторые параметры могут быть стратегии не нужны
  - ▶ Передавать сам контекст в качестве аргумента — в Context интерфейс для доступа к данным
- ▶ Стратегия может быть параметром шаблона
  - ▶ Если не надо её менять на лету
  - ▶ Не надо абстрактного класса и нет оверхеда на вызов виртуальных методов
- ▶ Стратегия по умолчанию

# Что почитать

- ▶ Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес, Приемы объектно-ориентированного проектирования. Паттерны проектирования.
- ▶ Мартин Фаулер, UML: основы.
- ▶ Роберт Мартин, Чистый код: создание, анализ и рефакторинг

