

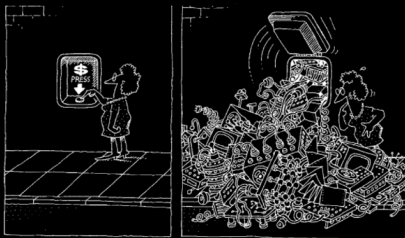
# Лекция 2: Декомпозиция, объектно-ориентированное проектирование

Юрий Литвинов  
yurii.litvinov@gmail.com

14.02.2022

# Сложность

- ▶ **Существенная сложность** (essential complexity) — сложность, присущая решаемой проблеме; ею можно управлять, но от неё нельзя избавиться
- ▶ **Случайная сложность** (accidental complexity) — сложность, привнесённая способом решения проблемы



# Свойства сложных систем

- ▶ Иерархичность — свойство системы состоять из иерархии подсистем или компонентов
  - ▶ Декомпозиция
- ▶ Наличие относительно небольшого количества видов компонентов, экземпляры которых сложно связаны друг с другом
  - ▶ Выделение общих свойств компонентов, абстрагирование
- ▶ Сложная система, как правило, является результатом эволюции простой системы
- ▶ Сложность вполне может превосходить человеческие интеллектуальные возможности

# Подходы к декомпозиции

- ▶ Восходящее проектирование
  - ▶ Сначала создаём “кирпичики”, потом собираем из них всё более сложные системы
- ▶ Нисходящее проектирование
  - ▶ Постепенная реализация модулей
  - ▶ Строгое задание интерфейсов
  - ▶ Активное использование “заглушек”
  - ▶ Модули
    - ▶ Четкая декомпозиция
    - ▶ Минимизация
    - ▶ Один модуль — одна функциональность
    - ▶ Отсутствие побочных эффектов
    - ▶ Независимость от других модулей
    - ▶ Принцип сокрытия данных

# Модульность

- ▶ Разделение системы на компоненты
- ▶ Потенциально позволяет создавать сколь угодно сложные системы
- ▶ Строгое определение контрактов позволяет разрабатывать независимо
- ▶ Необходим баланс между количеством и размером модулей



# Сопряжение и связность

- ▶ **Сопряжение (Coupling)** — мера того, насколько взаимозависимы разные модули в программе
- ▶ **Связность (Cohesion)** — степень, в которой задачи, выполняемые одним модулем, связаны друг с другом
- ▶ Цель: слабое сопряжение и сильная связность

# Объекты

- ▶ Objects may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods — **Wikipedia**
- ▶ An object stores its state in fields and exposes its behavior through methods — **Oracle**
- ▶ Each object looks quite a bit like a little computer — it has a state, and it has operations that you can ask it to perform — **Thinking in Java**
- ▶ An object is some memory that holds a value of some type — **The C++ Programming Language**
- ▶ An object is the equivalent of the quanta from which the universe is constructed — **Object Thinking**

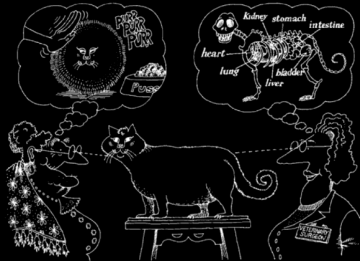
# Объекты

- ▶ Имеют
  - ▶ Состояние
    - ▶ Инвариант
  - ▶ Поведение
  - ▶ Идентичность
- ▶ Взаимодействуют через посылку и приём сообщений
  - ▶ Объект вправе сам решить, как обработать вызов метода (**полиморфизм**)
  - ▶ Могут существовать в разных потоках
- ▶ Как правило, являются экземплярами **классов**



# Абстракция

**Абстракция** выделяет существенные характеристики объекта, отличающие его от остальных объектов, с точки зрения наблюдателя



© G. Booch, "Object-oriented analysis and design"



# Наследование и композиция

## ▶ **Наследование**

- ▶ Отношение “Является” (is-a)
- ▶ Способ абстрагирования и классификации
- ▶ Средство обеспечения полиморфизма

## ▶ **Композиция**

- ▶ Отношение “Имеет” (has-a)
- ▶ Способ создания динамических связей
- ▶ Средство обеспечения делегирования

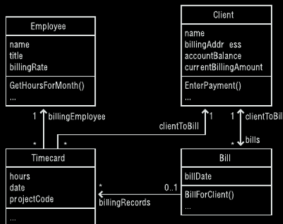
## ▶ **Более-менее взаимозаменяемы**

- ▶ Объект-потомок на самом деле включает в себя объект-предок
- ▶ Композиция обычно предпочтительнее

# Определение объектов реального мира

## Объектная модель предметной области

- ▶ Определение объектов и их атрибутов
- ▶ Определение действий, которые могут быть выполнены над каждым объектом (назначение ответственности)
- ▶ Определение связей между объектами
- ▶ Определение интерфейса каждого объекта



# Изоляция сложности

- ▶ Сложные алгоритмы могут быть инкапсулированы
- ▶ Сложные структуры данных — тоже
- ▶ И даже сложные подсистемы
- ▶ Надо внимательно следить за интерфейсами



# Изоляция возможных изменений

- ▶ Потенциальные изменения могут быть инкапсулированы
- ▶ Источники изменений
  - ▶ Бизнес-правила
  - ▶ Зависимости от оборудования и операционной системы
  - ▶ Ввод-вывод
  - ▶ Нестандартные возможности языка
  - ▶ Сложные аспекты проектирования и конструирования
  - ▶ Третьесторонние компоненты
  - ▶ ...

# Изоляция служебной функциональности

- ▶ Служебная функциональность может быть инкапсулирована
  - ▶ Репозитории
  - ▶ Фабрики
  - ▶ Диспетчеры, медиаторы
  - ▶ Статические классы (*Сервисы*)
  - ▶ ...

# Принципы SOLID

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



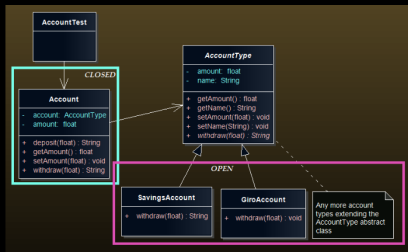
# Single responsibility principle

- ▶ 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



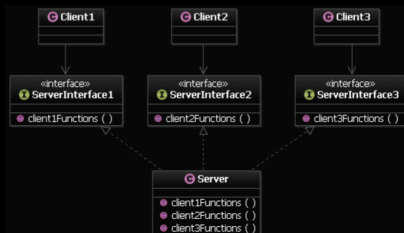
# Liskov substitution principle

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



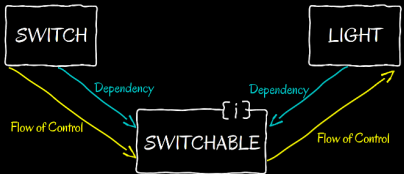
# Interface segregation principle

- ▶ 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`
  - ▶ `book.pages().last().text()`
  - ▶ `book.lastPageText()`
- ▶ Иногда называют “Крушение поезда”



© Р. Мартин, “Чистый код”