

# Хорошие практики ООП

Юрий Литвинов  
yurii.litvinov@gmail.com

12.07.2017г

# Модульность

- ▶ Разделение системы на компоненты
- ▶ Потенциально позволяет создавать сколь угодно сложные системы



# Информационная закрытость

- ▶ Содержание модулей должно быть скрыто друг от друга
  - ▶ Все модули независимы
  - ▶ Обмениваются только информацией, необходимой для работы
  - ▶ Доступ к операциям и структурам данных модуля ограничен
- ▶ Обеспечивается возможность разработки модулей различными независимыми коллективами
- ▶ Обеспечивается лёгкая модификация системы

# Подходы к декомпозиции

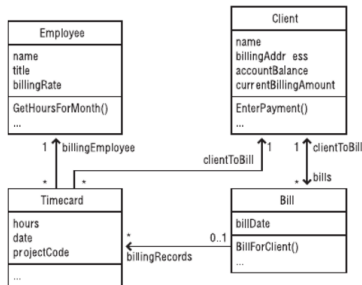
- ▶ Восходящее проектирование
- ▶ Нисходящее проектирование
  - ▶ Постепенная реализация модулей
  - ▶ Строгое задание интерфейсов
  - ▶ Активное использование “заглушек”
  - ▶ Модули
    - ▶ Четкая декомпозиция
    - ▶ Минимизация
    - ▶ Один модуль — одна функциональность
    - ▶ Отсутствие побочных эффектов
    - ▶ Независимость от других модулей
    - ▶ Принцип сокрытия данных

# 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**

# Определение объектов реального мира

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



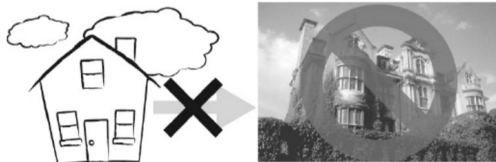
## Согласованные абстракции

- ▶ Выделение существенных характеристик объекта и игнорирование несущественных
- ▶ Определение его концептуальных границы с точки зрения наблюдателя
  - ▶ Определение интерфейсов
- ▶ Управление сложностью через фиксацию внешнего поведения
- ▶ Необходимы разные уровни абстракции



## Инкапсуляция деталей реализации

- ▶ Отделение друг от друга внутреннего устройства и внешнего поведения
- ▶ Изолирование контрактов интерфейса от реализации
- ▶ Управление сложностью через сокрытие деталей реализации





# Соккрытие “лишней” информации

- ▶ Изоляция “личной” информации
  - ▶ секреты, которые скрывают сложность
  - ▶ секреты, которые скрывают источники изменений
- ▶ Барьеры, препятствующие соккрытию
  - ▶ избыточное распространение информации
  - ▶ поля класса как глобальные данные
  - ▶ снижение производительности



## Изоляция возможных изменений

- ▶ Определите элементы, изменение которых кажется вероятным
- ▶ Отделите элементы, изменение которых кажется вероятным
- ▶ Изолируйте элементы, изменение которых кажется вероятным
- ▶ Источники изменений
  - ▶ Бизнес-правила
  - ▶ Зависимости от оборудования
  - ▶ Ввод-вывод
  - ▶ Нестандартные возможности языка
  - ▶ Сложные аспекты проектирования и конструирования
  - ▶ Переменные статуса
  - ▶ Размеры структур данных
  - ▶ ...

# 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

# Принципы SOLID

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

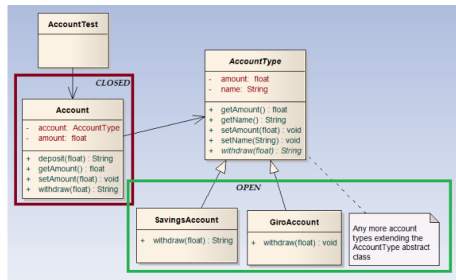
# Single responsibility principle

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



# 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



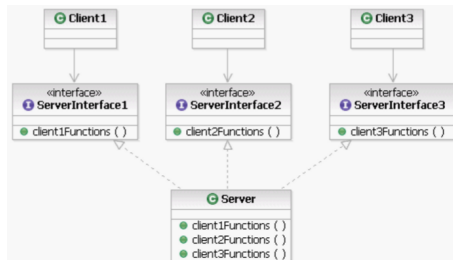
# Liskov substitution principle

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



# Interface segregation principle

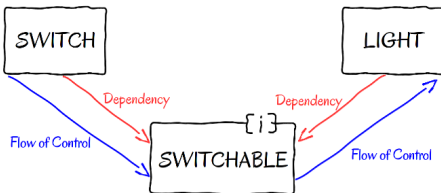
- ▶ Клиенты не должны зависеть от методов, которые они не используют
  - ▶ слишком “толстые” интерфейсы необходимо разделять на более мелкие и специфические





# 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()`