Проектирование ПО

Лекция 3: Объектно-ориентированное проектирование

Тимофей Брыксин
timofey.bryksin@gmail.com

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

Объекты revisited

`An object is the equivalent of the quanta from which the universe is constructed --` Object Thinking

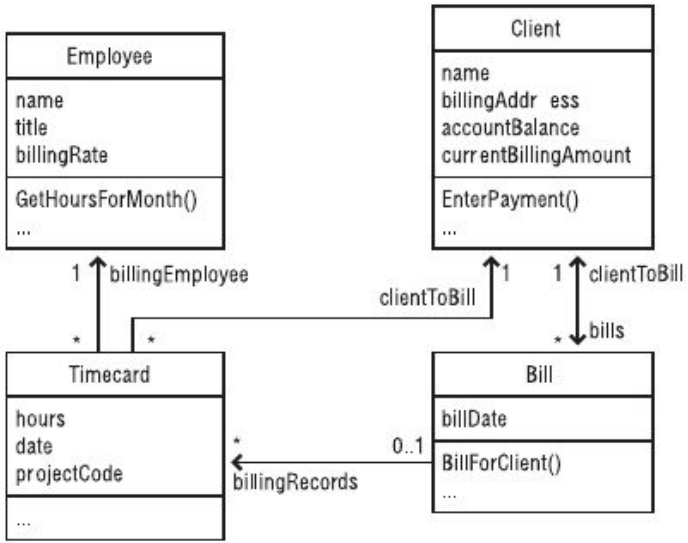
- “Живые организмы”
- Отражают сущности реального мира
- Взаимодействуют через посылку/приём сообщений
- Имеют
  - состояние
    - инвариант
  - поведение
  - идентичность

(Некоторые) принципы ОО-проектирования

- Определите объекты реального мира
- Определите согласованные абстракции
- Инкапсулируйте детали реализации
- Выбирайте между наследованием и композицией
- Скрывайте “лишнюю” информацию
- Определяйте области вероятных изменений
- Поддерживайте сопряжение слабым
- Стремитесь к максимальной связности
- Формализуйте контракты классов
- Проектируйте систему для тестирования
- Подумайте об использовании “грубой силы”
- Рисуйте диаграммы

Определите объекты реального мира

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

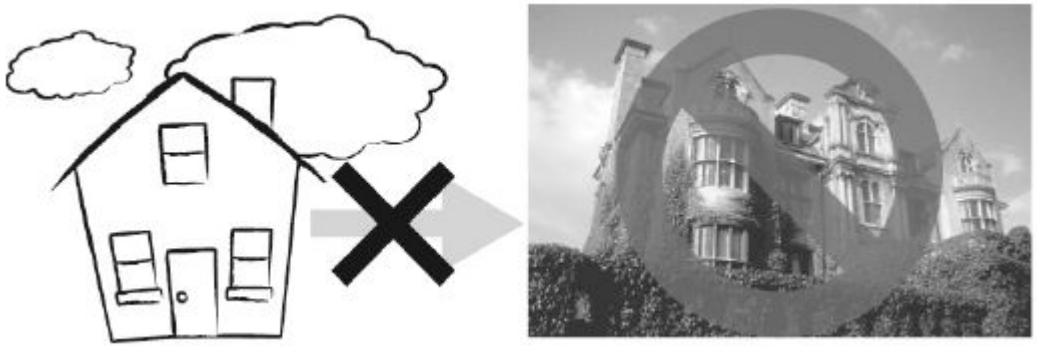


Определите согласованные абстракции

- 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

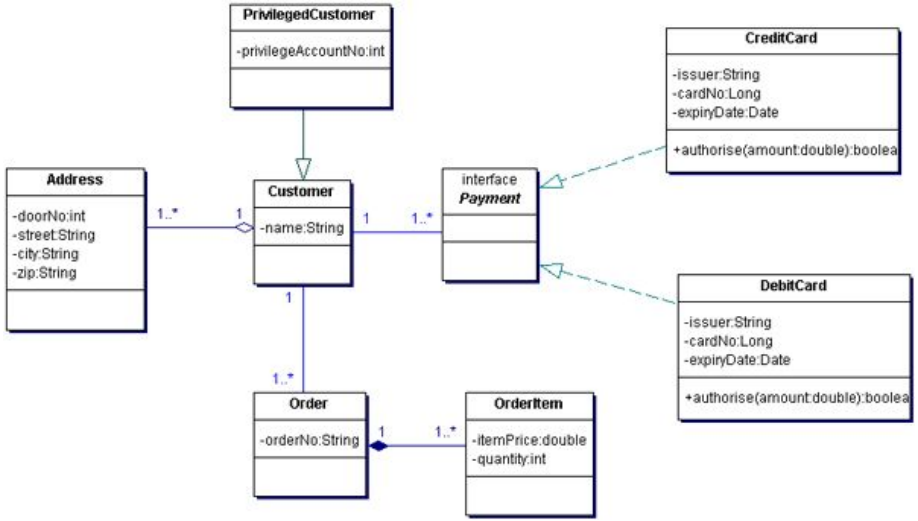
Инкапсулируйте детали реализации

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



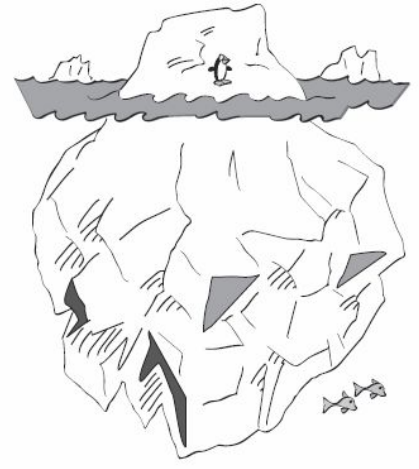
Выбирайте между наследованием и композицией

- Наследование
  - отношение “is-a”
  - создание новых сущностей
  - расширяет идею абстракции
- Композиция (агрегация)
  - отношение “has-a”
  - гибкое управление зависимостями в runtime



Скрывайте “лишнюю” информацию

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



Определяйте области вероятных 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

- Цель: слабое сопряжение и сильная связность
- Критерии сопряжения
  - объём
  - видимость
  - гибкость

Дополнительные принципы

- Формализуйте контракты классов
- Проектируйте систему для тестирования
- Подумайте об использовании “грубой силы”
- Рисуйте диаграммы

Принципы SOLID

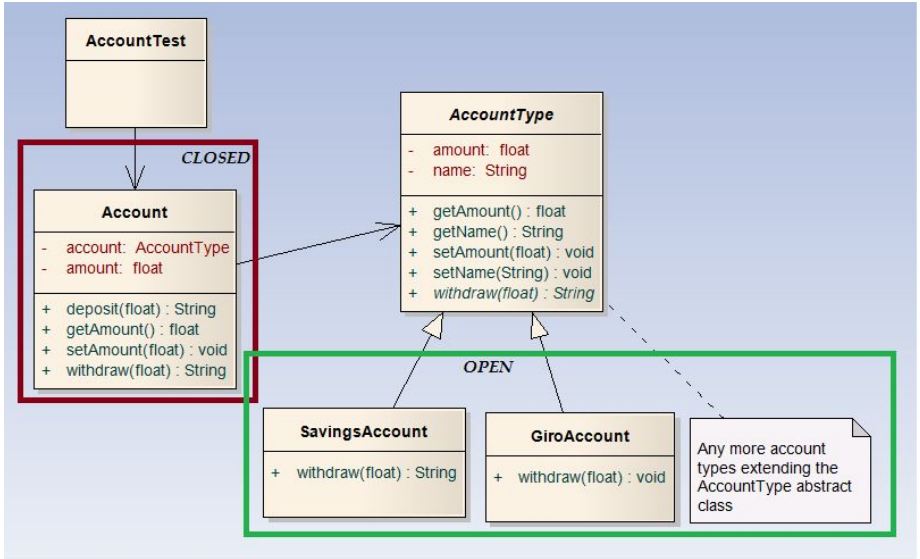
- Single responsibility principle
- Open/closed principle
- Liskov substitution principle
- Interface segregation principle
- Dependency inversion principle

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

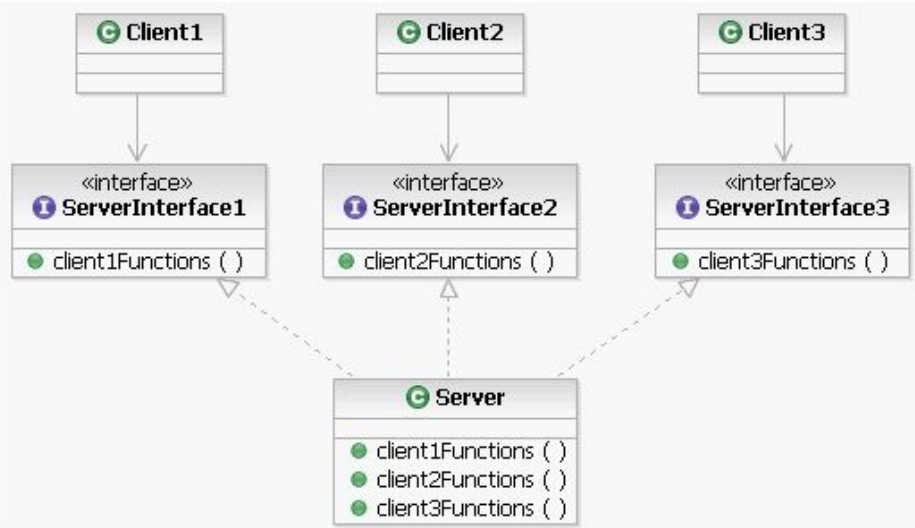


Liskov substitution principle

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

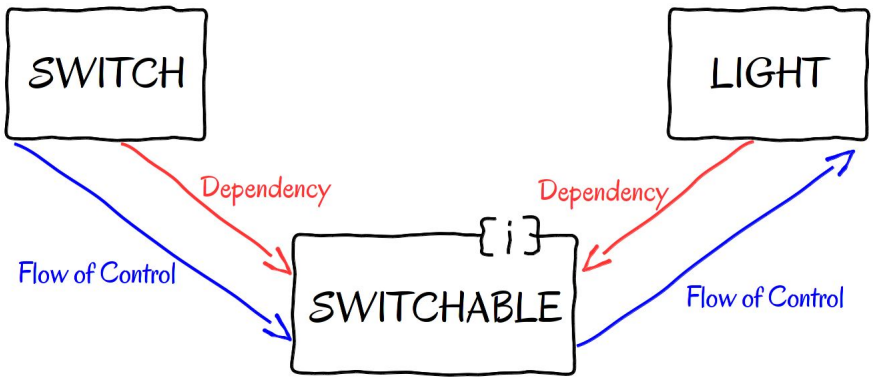
Interface segregation principle

- 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 A не должен иметь возможность получить непосредственный доступ к объекту C, если у объекта A есть доступ к объекту B, и у объекта B есть доступ к объекту C
  - book.pages.last.text
  - book.pages().last().text()
  - book.lastPageText()