Хорошие практики тестирования и ООП

Юрий Литвинов
yurii.litvinov@gmail.com

03.04.2017г

Тестирование, зачем

- Любая программа содержит ошибки
- Если программа не содержит ошибок, их содержит алгоритм, который реализует эта программа
- Если ни программа, ни алгоритм ошибок не содержат, такая программа даром никому не нужна

Тестирование не позволяет доказать отсутствие ошибок, оно позволяет лишь найти ошибки, которые в программе присутствуют

Виды тестов

- Модульные
- Интеграционные
- Системные

- Регрессионные
- Приёмочные
- Дымовые (smoke-test)

- UI-тесты
- Нагрузочные тесты
- ...

Модульные тесты

- Тест на каждый отдельный метод, функцию, иногда класс
- Пишутся программистами
- Запускаются часто (как минимум, после каждого коммита)
- Должны работать быстро
- Должны всегда проходить
- Принято не продолжать разработку, если юнит-тест не проходит
- Помогают быстро искать ошибки (вы ещё помните, что исправляли), рефакторить код (“ремни безопасности”), продумывать архитектуру (мешанину невозможно оттестировать), документировать код (каждый тест — это рабочий пример вызова)

Почему модульные тесты полезны

- Помогают искать ошибки
  - Особо эффективны, если налажен процесс Continuous Integration
- Облегчают изменение программы
  - Помогают при рефакторинге
- Тесты — документация к коду
- Помогают улучшить архитектуру
- НЕ доказывают отсутствие ошибок в программе

Best practices

- Независимость тестов
  - Желательно, чтобы поломка одного куска функциональности ломала один тест
- Тесты должны работать быстро
  - И запускаться после каждой сборки
    - Continuous Integration!
- Тестов должно быть много
  - Следить за Code coverage
- Каждый тест должен проверять конкретный тестовый сценарий
  - Никаких try-catch внутри теста
    - @Test(expected = NullPointerException.class)
    - Любая нормальная библиотека юнит-тестирования умеет ожидать исключения
- Test-driven development

Hamcrest

```
assertThat(someString, is(not(equalTo(someOtherString))));
assertThat(list, everyItem(greaterThan(1)));
assertThat(cat.getKittens(), hasItem(someKitten));
assertThat("test",
   anyOf(is("testing"), containsString("est")));
assertThat(x,
   allOf(greaterThan(0), lessThanOrEqualTo(10)));
```

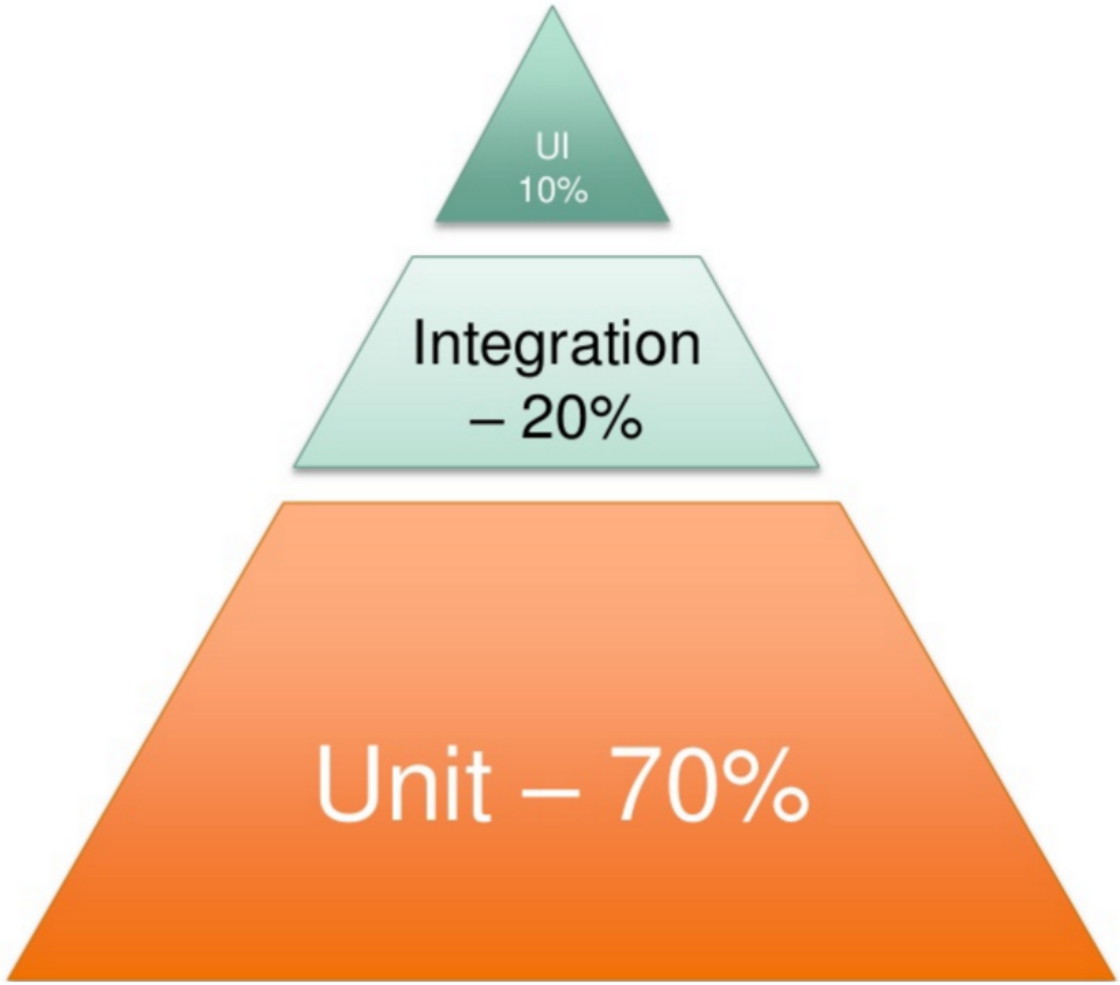
Mock-объекты

- Объекты-заглушки, симулирующие поведение реальных объектов и контролирующие обращения к своим методам
  - Как правило, такие объекты создаются с помощью библиотек
- Используются, когда реальные объекты использовать
  - Слишком долго
  - Слишком опасно
  - Слишком трудно
  - Для добавления детерминизма в тестовый сценарий
  - Пока реального объекта ещё нет
  - Для изоляции тестируемого объекта
- Для mock-объекта требуется, чтобы был интерфейс, который он мог бы реализовать, и какой-то механизм внедрения объекта

Пример: Mockito

```java
@Test
public void test() throws Exception {
    // Arrange, prepare behaviour
    Helper aMock = mock(Helper.class);
    when(aMock.isCalled()).thenReturn(true);
    // Act
    testee.doSomething(aMock);
    // Assert - verify interactions (optional)
    verify(aMock).isCalled();
}
```

Соотношение тестов



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

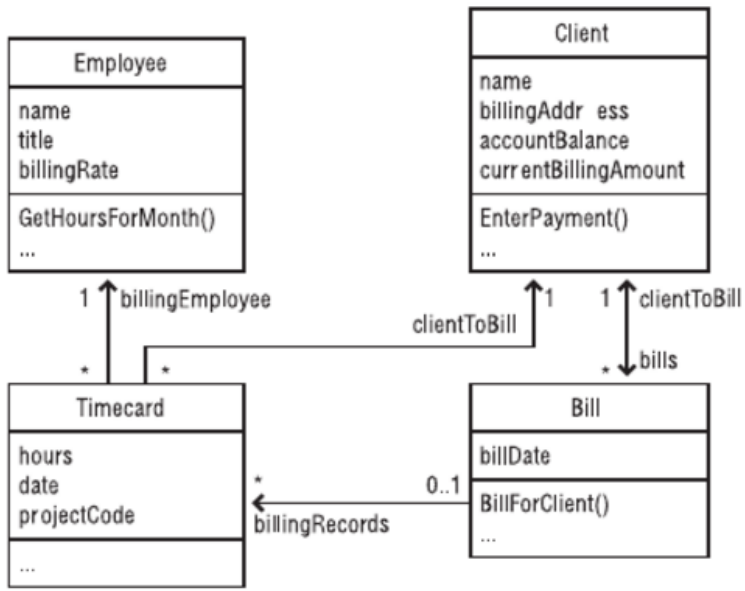
- Восходящее проектирование
- Нисходящее проектирование
  - Постепенная реализация модулей
  - Строгое задание интерфейсов
  - Активное использование “заглушек”
  - Модули
    - Четкая декомпозиция
    - Минимизация
    - Один модуль — одна функциональность
    - Отсутствие побочных эффектов
    - Независимость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

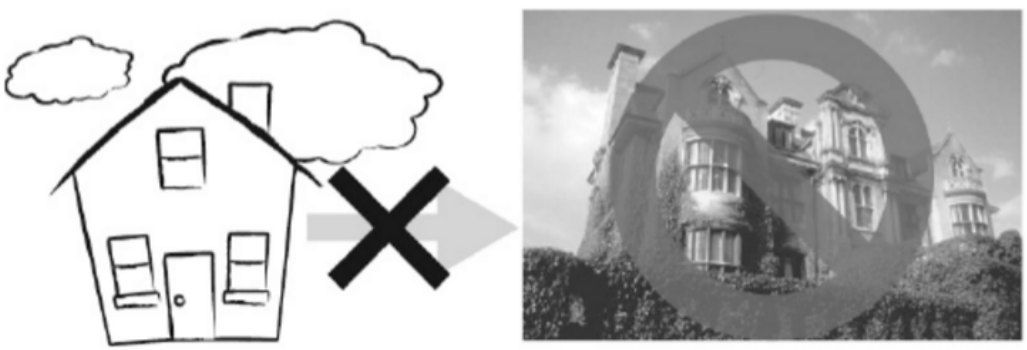


Согласованные абстракции

- 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



Сокрытие “лишней” информации

- 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

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

Дополнительные принципы

- Формализуйте контракты классов
- Проектируйте систему для тестирования
- Рисуйте диаграммы