

# Юнит-тестирование и системы сборки

Юрий Литвинов  
yurii.litvinov@gmail.com

16.01.2019г

# Юнит-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

# Информация к размышлению

- ▶ Программа из сотни строк может иметь  $10^{18}$  путей исполнения
  - ▶ Времени жизни вселенной не хватило бы, чтобы их покрыть
- ▶ После передачи на тестирование в программах в среднем от 1 до 3 ошибок на 100 строк кода
- ▶ В процессе разработки — 1.5 ошибок на 1 строку кода (!)
- ▶ Если для исправления ошибки надо изменить не более 10 операторов, с первого раза это делают правильно в 50% случаев
- ▶ Если для исправления ошибки надо изменить не более 50 операторов, с первого раза это делают правильно в 20% случаев

# Модульное тестирование

- ▶ Тест на каждый отдельный метод, функцию, иногда класс
- ▶ Пишутся программистами
- ▶ Запускаются часто (как минимум, после каждого коммита)
- ▶ 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 протестировать), документировать код (каждый тест – это рабочий пример вызова)

# JUnit 5

- ▶ JUnit — самая известная библиотека юнит-тестирования для Java
- ▶ JUnit 5 — его последняя версия (часто до сих пор используется JUnit 4)
- ▶ Отлично интегрируется с IDEA
  - ▶ Генерация заглушек тестов, запуск тестов прямо из редактора и т.д.
- ▶ Тест — это отдельный класс, в котором есть методы, помеченные аннотацией `@Test`
- ▶ Внутри теста обычно три фазы
  - ▶ Настройка тестового окружения
  - ▶ Выполнение действия, которое хотим тестировать
  - ▶ Проверка результатов
    - ▶ Вызовы `assertEquals`, `assertNull`, `assertTrue` и т.д.

# Демонстрация

# Демонстрация

# Ещё аннотации

- ▶ **@BeforeEach** — метод, запускающийся перед каждым тестом
- ▶ **@AfterEach** — метод, запускающийся после каждого теста
- ▶ **@BeforeAll** — метод, запускающийся перед всеми тестами в классе (должен быть static)
- ▶ **@AfterAll** — метод, запускающийся после всех тестов в классе (должен быть static)
- ▶ **@Disabled** — тест отключён и не запускается

# Assertions

## Наиболее интересные

- ▶ `assertEquals`
  - ▶ Обратите внимание на порядок параметров, сначала ожидаемое, затем то, что получилось
- ▶ `assertArrayEquals`
- ▶ `assertTrue/assertFalse`
- ▶ `assertNull/assertNotNull`
- ▶ `assertThrows`
- ▶ `fail`



# 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
  - ▶ Следите за Code coverage, должно быть близко к 100%
- ▶ Каждый тест должен проверять конкретный тестовый сценарий
  - ▶ Нельзя ловить исключения в тесте без крайней нужды
  - ▶ С большой осторожностью пользоваться Random-ом
- ▶ Именование тестов: <http://stackoverflow.com/questions/155436/unit-test-naming-best-practices>
- ▶ Test-driven development

# Системы сборки

- ▶ Нужны, чтобы сборка не зависела от среды разработки
  - ▶ Воспроизводимость сборки
  - ▶ “Пакетный режим”, CI
- ▶ Умеют вычислять зависимости между компонентами, управлять внешними зависимостями
  - ▶ И автоматически скачивать нужные пакеты при сборке!
- ▶ Не только, собственно, сборка: тестирование, создание документации, инсталляторов, отчётов и т.д.
- ▶ Maven
- ▶ Gradle

# Maven, основные принципы

- ▶ Декларативность
- ▶ Convention Over Configuration
  - ▶ Стандартная структура папок
- ▶ Плагины
- ▶ Project Object Model
- ▶ Жизненный цикл
- ▶ Координаты
  - ▶ groupId
  - ▶ artifactId
  - ▶ version
- ▶ Репозиторий и on-demand-скачивание пакетов
  - ▶ <https://mvnrepository.com/>
  - ▶ Локальный кеш: `/.m2`

# Maven, стандартная структура папок

- ▶ pom.xml
- ▶ src
  - ▶ main
    - ▶ java
    - ▶ resources
  - ▶ test
    - ▶ java
    - ▶ resources
- ▶ target

# Maven, демонстрация

## Демонстрация

# Gradle

- ▶ Более легковесный, чем Maven
- ▶ Декларативный предметно-ориентированный язык на основе Groovy
- ▶ Использует Convention Over Configuration, но легко переопределить
  - ▶ Та же структура папок по умолчанию
- ▶ Использует репозиторий Maven для динамической загрузки зависимостей
- ▶ build.gradle, settings.gradle
- ▶ gradle-wrapper
  - ▶ gradlew, gradlew.bat
  - ▶ gradle-wrapper.jar, gradle-wrapper.properties
- ▶ Для Java 11 нужна Gradle 5, с IDEA (2018.3) поставляется Gradle 4

# Пример build.gradle

```
plugins {  
    id 'java'  
}  
  
group 'com.example'  
version '1.0-SNAPSHOT'  
  
sourceCompatibility = 11  
  
repositories {  
    mavenCentral()  
}  
  
dependencies {  
    testCompile('org.junit.jupiter:junit-jupiter-api:5.3.2')  
    testRuntime('org.junit.jupiter:junit-jupiter-engine:5.3.2')  
}  
  
test {  
    useJUnitPlatform()  
}
```

# Gradle, демонстрация

## Демонстрация

В домашке требуется сборка либо `mvn install`, либо `gradlew build` из  
КОНСОЛИ