Java Pro 3.4

TEL-RAN
by Starta Institute

Unit-тесты

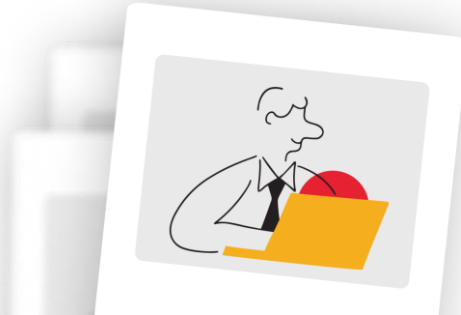
ПРЕПОДАВАТЕЛЬ

TEL-RAN
by Starta Institute

Юрий Костяной

Java/Kotlin backend-разработчик

- 3+ года опыта в коммерческой разработке
- 2+ года опыта в преподавании
- Проекты по интеграции сторонних платформ, CRM
- Проблемно-ориентированный подход в преподавании



ВАЖНО:

- Камера должна быть включена на протяжении всего занятия.
- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

TEL-RAN
by Starta Institute

ПЛАН ЗАНЯТИЯ

1. Повторение
2. Основной блок
3. Вопросы по основному блоку
4. Домашняя работа

TEL-RAN
by Starta Institute

1 ОСНОВНОЙ БЛОК

Введение

TEL-RAN
by Starta Institute

- Но мы же разработчики!
- Единица раз
- Пара метров
- Затыкать и дурачить

Проблема

TEL-RAN
by Starta Institute

Представьте, что Вам дали первую задачу на Вашей первой работе в качестве бэкенд-разработчика. Вы выполнили всё, что было указано в постановке с заданием.

Но как проверить, что задача решена правильно?

Можно отдать задачу на code-review, но чаще всего проверяющий смотрит на качество кода, а не на то, что функционал был реализован правильно. Кроме того, в большом приложении часто невозможно по коду определить, что доработки выполнены правильно.

Можно отдать доработку на тестирование QA-специалисту, но если все будут выдавать сырые решения на тестирование, то отдел QA будет перегружен.

Вам первому нужно убедиться, что код работает так, как должен.

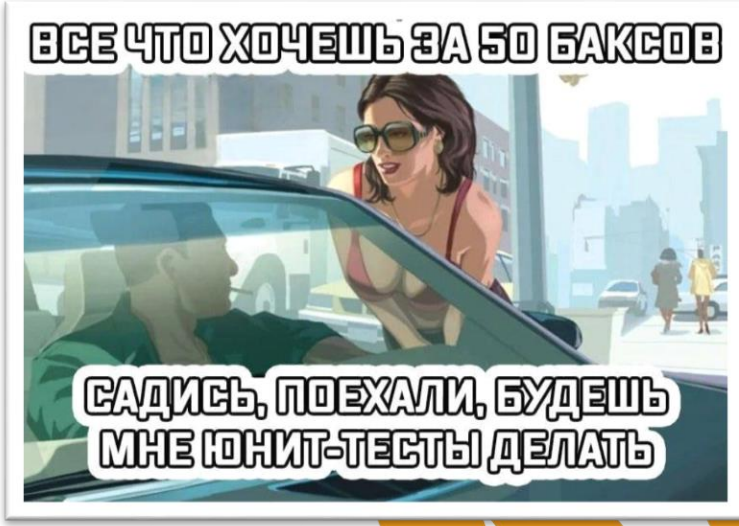
Но мы же разработчики!

Арсенал разработчика

TEL-RAN
by Starta Institute

Чаще всего разработчики используют два вида тестирования:

- **Модульные тесты (unit-тесты)** – тестирование функционала простейшей единицы кода (методов класса, структуры данных). Выполняются быстро, т.к. не требуют развёртывания среды (базы данных, сервисов обмена сообщениями, других приложений). Unit-тесты позволяют автоматизировать проверку приложения на этапе сборки проекта.



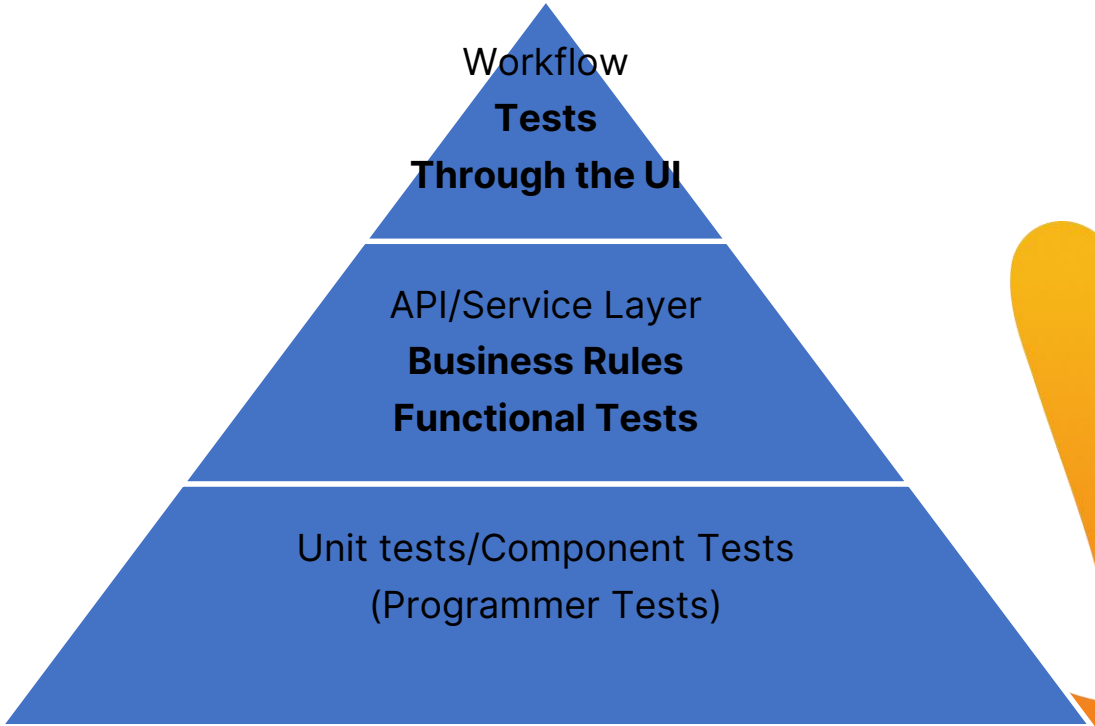
- **Интеграционные тесты** – тестирование изменений приложение в настроенной среде для проверки взаимодействия изменённого кода со связанными системами.

Но мы же разработчики!

Пирамида тестирования

TEL-RAN
by Starta Institute

Майк Кон предложил пирамиду автоматизации, чтобы помочь командам найти лучший подход к автоматизации тестирования:



Но мы же разработчики!

Для чего писать тесты

TEL-RAN
by Starta Institute

- Чтобы проверить новый функционал приложения или исправление бага.
- Чтобы проверить, что доработка/исправление не сломало ранее написанный код.
- Чтобы не сердить QA-специалистов, руководство и заказчика.

Но мы же разработчики!

Что нужно тестировать

Тестируют обычно интерфейс класса:

- все публичные методы;
- поведение класса при разных условиях (все ветки условных операторов);
- все исключения, выбрасываемые методами.

Задание

TEL-RAN
by Starta Institute

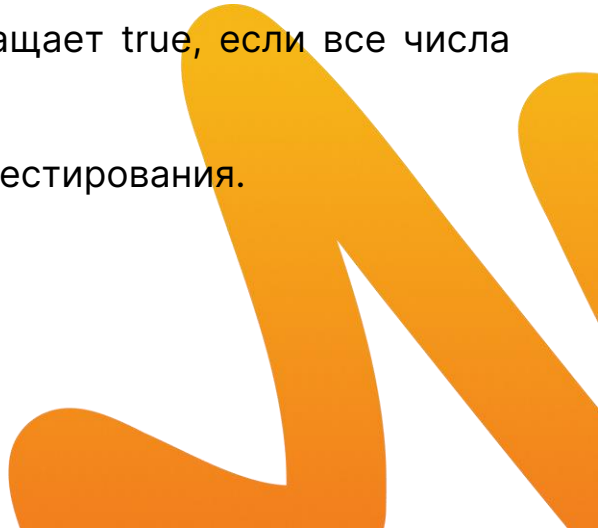
0 Создайте maven-проект. Измените JDK на 17. Добавьте зависимость https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter

1 Создайте класс MathUtils.

2 Напишите метод, который принимает список целых чисел и возвращает произведение этих чисел. Если в метод передан пустой список, методы выбрасывает исключение.

3 Напишите метод, который принимает список чисел и возвращает true, если все числа положительные.

4 Протестируйте работу методов помощью обычного ручного тестирования.

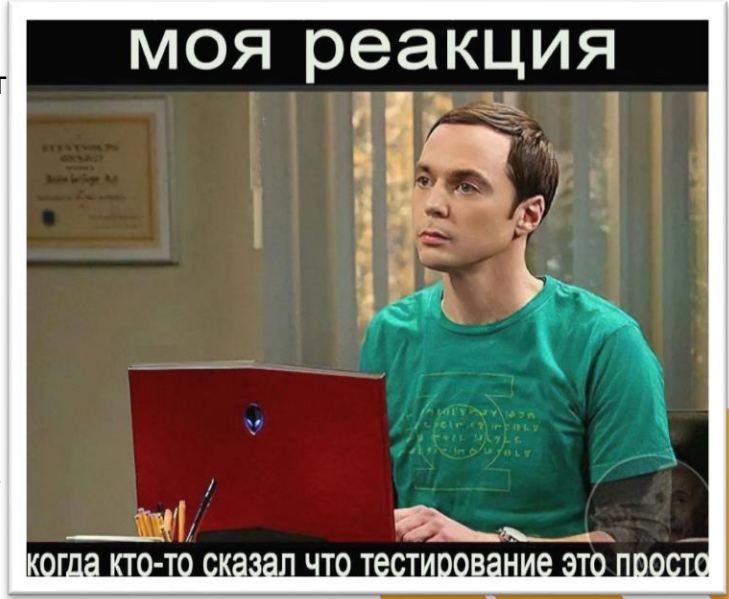


Единица раз

Шаги тестирования

TEL-RAN
by Starta Institute

1. Разработчик рассматривает компонент и принцип его работы. После этого он продумывает логику того, как будет работать тест
2. Пишется тест. Тест необходимо запустить до написания, чтобы увидеть, что он не проходит.
3. В тесте необходимо определить, что является ожидаемым результатом.
4. При необходимости можно организовать входные данные для теста.
5. В тесте разработчик вызывает тестируемый метод и сохраняет результат его выполнения.



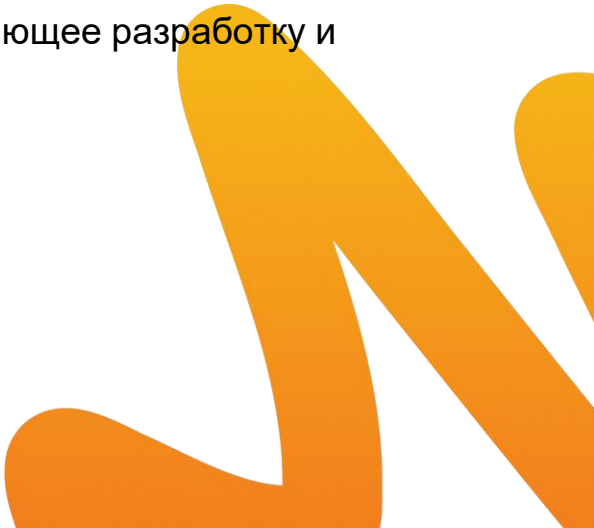
6. В коде происходит сравнение ожидаемого результата и полученного. Если результаты не совпали, то тест выбрасывает специальное исключение.
7. Для каждого случая пишется свой тест. Тесты должны быть максимально простыми и, желательно, не содержать условных конструкций или сложного кода.

Единица раз
JUnit

TEL-RAN
by Starta Institute

JUnit - *фреймворк* для Java, полностью совместимый с самим языком и его инструментами, поэтому им удобнее всего пользоваться для тестирования Java-проектов. Актуальная версия JUnit 5, но в проектах ещё встречается JUnit 4.

Фреймворк (*англ.* «каркас, рама; структура») – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.



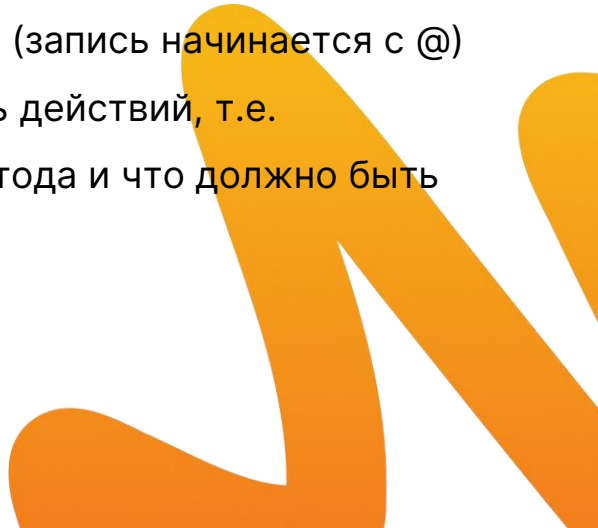
Единица раз

Аннотации JUnit

TEL-RAN
by Starta Institute

JUnit использует **аспектно-ориентированное программирование (АОП)** – это парадигма программирования, являющейся дальнейшим развитием процедурного и объектно-ориентированного программирования (ООП). Идея АОП заключается в выделении так называемой сквозной функциональности.

Для реализации АОП в JUnit используются аннотации. Аннотации (запись начинается с @) «заворачивают» методы класса в некоторую последовательность действий, т.е. показывает, что должно быть выполнено до аннотированного метода и что должно быть выполнено после него.



Единица раз

Аннотации JUnit

TEL-RAN
by Starta Institute

@Test – определяет что метод является тестовым.

@BeforeEach – указывает на то, что метод будет выполнятся перед каждым тестом.

@AfterEach – указывает на то что метод будет выполнятся после каждого теста.

@BeforeAll – указывает на то, что метод будет выполняться до запуска тестов в данном классе.

@AfterAll – указывает на то, что метод будет выполнятся после всех тестов класса.

@Disable – говорит, что метод будет проигнорирован в момент проведения тестирования.

@DisplayName – указывает, как отображать имя метода, если имя должно отличаться от названия метода

@Tag – дополнительные теги для сортировки тестов.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class JUnit5Test {

    @Test
    void helloJUnit5() {
        assertEquals(10, 5 + 5);
    }
}
```

Единица раз

Аннотации JUnit

TEL-RAN
by Starta Institute

Один из основных классов для сравнения ожидаемого результата и полученного результата *org.junit.jupiter.api.Assertions.* Основные методы класса:

assertEquals(expected, actual) – проверяет, что ожидаемый и полученный результат эквивалентны.

assertNotNull(actual) – проверяет, что значение не null

assertTimeout(time, runnable) – проверяет, что действие runnable выполняется дольше времени time.

assertTrue(), assertFalse() – проверяет, что утверждение истинно или ложно.

assertThrows(exceptionType, runnable) – проверка, что действие runnable бросает исключение.

assertAll – проверка набора утверждений

```
@Test
void groupAssertions() {
    int[] numbers = {0, 1, 2, 3, 4};
    assertAll("numbers",
            () -> assertEquals(numbers[0], 1),
            () -> assertEquals(numbers[3], 3),
            () -> assertEquals(numbers[4], 1)
    );
}
```

Задание

TEL-RAN
by Starta Institute

Напишите тесты для методов класса MathUtils с помощью JUnit.

Проблема

TEL-RAN
by Starta Institute

В тестировании принято выделять положительные случаи (positive cases), когда код отрабатывает корректно, и отрицательные случаи (negative cases), когда код бросает исключения.

Часто бывает, что все позитивные случаи имеют одну логику проверки, а отличаются только входными параметрами и ожидаемым результатом. Дублировать один и тот же код тестов неправильно (DRY).

Для этих целей в JUnit предусмотрены параметризированные тесты.

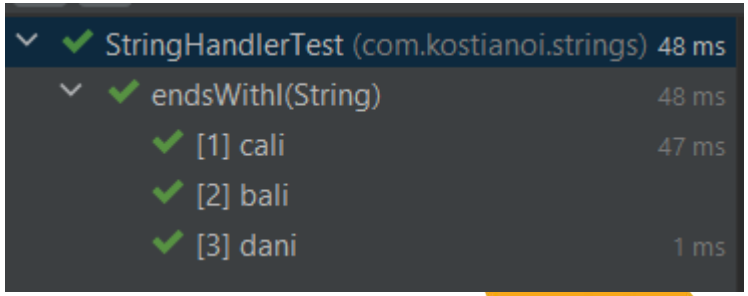
Пара метров

Параметризированные тесты

TEL-RAN
by Starta Institute

Параметризированные тесты – это тесты, которые запускаются множество раз, обычно с разными данными на входе (*@ParameterizedTest*).

```
@ParameterizedTest
@ValueSource(strings = { "cali", "bali", "dani" })
void endsWithI(String str) {
    assertTrue(str.endsWith("i"));
}
```



Пара метров

Источник тестовых данных

TEL-RAN
by Starta Institute

Перечень значений (*@ValueSource*);

```
@ParameterizedTest
@ValueSource(strings = { "cali", "bali", "dani" })
void endsWithI(String str) {
    assertTrue(str.endsWith("i"));
}
```

```
@ParameterizedTest
@ValueSource(ints = { 1, 2, 3 })
void testWithValueSource(int argument) {
    assertTrue(argument > 0 && argument < 4);
}
```

Пара метров

Источник тестовых данных

TEL-RAN
by Starta Institute

null и/или пустое значение(*@NullAndEmptySource, @NullSource, @EmptySource*)

```
@ParameterizedTest
@EmptySource
void isBlank_ShouldReturnTrueForEmptyStrings(String input) {
    assertTrue(input.isBlank());
}
```

Пара метров

Источник тестовых данных

TEL-RAN
by Starta Institute

Удобно указывать перечисление в качестве входных аргументов тестов (*@EnumSource*).

```
@ParameterizedTest
@EnumSource(Month.class) // passing all 12 months
void getValueForAMonth_IsAlwaysBetweenOneAndTwelve(Month month) {
    int monthNumber = month.getValue();
    assertTrue(monthNumber >= 1 && monthNumber <= 12);
}
```

Пара метров

Источник тестовых данных

TEL-RAN
by Starta Institute

Можно ограничить набор перечисления, либо исключить отдельные элементы

```
@ParameterizedTest
@EnumSource(value = Month.class, names = {"APRIL", "JUNE", "SEPTEMBER", "NOVEMBER"})
void someMonths_Are30DaysLong(Month month) {
    final boolean isALeapYear = false;
    assertEquals(30, month.length(isALeapYear));
}
```

```
@ParameterizedTest
@EnumSource(
        value = Month.class,
        names = {"APRIL", "JUNE", "SEPTEMBER", "NOVEMBER", "FEBRUARY"},
        mode = EnumSource.Mode.EXCLUDE)
void exceptFourMonths_OthersAre31DaysLong(Month month) {
    final boolean isALeapYear = false;
    assertEquals(31, month.length(isALeapYear));
}
```

Пара метров

Источник тестовых данных

TEL-RAN
by Starta Institute

Источником тестовых данных могут быть csv-литералы или csv-файл

```
@ParameterizedTest
@CsvSource(value = {"test:test", "tEst:test", "Java:java"}, delimiter = ':')
void toLowerCase_ShouldGenerateTheExpectedLowercaseValue(String input, String expected) {
    String actualValue = input.toLowerCase();
    assertEquals(expected, actualValue);
}
```

```
/*input,expected
test,TEST
tEst,TEST
Java,JAVA*/
@ParameterizedTest
@CsvFileSource(resources = "/data.csv", numLinesToSkip = 1)
void toUpperCase_ShouldGenerateTheExpectedUppercaseValueCSVFile(String input, String expected) {
    String actualValue = input.toUpperCase();
    assertEquals(expected, actualValue);
}
```

Пара метров

Источник тестовых данных

TEL-RAN
by Starta Institute

Наконец, источником данных может быть метод, подающий аргументы в виде потока (Stream).

```
private static Stream<Arguments> provideStringsForIsBlank() {
    return Stream.of(
            Arguments.of(null, true),
            Arguments.of("", true),
            Arguments.of("  ", true),
            Arguments.of("not blank", false)
    );
}

@ParameterizedTest
@MethodSource("provideStringsForIsBlank")
void isBlank_ShouldReturnTrueForNullOrBlankStrings(String input, boolean expected) {
    assertEquals(expected, input.isBlank());
}
```

Пара метров

Повторяемые тесты

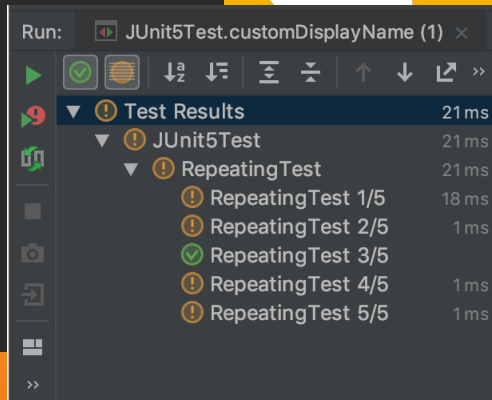
TEL-RAN
by Starta Institute

Повторяемые тесты – тесты, которые запускаются много раз (*@RepeatedTest*). Их применяют, когда число вызовов метода влияет на результат выполнения.

В таком тесте указывают количество повторений.

Этот вид тестов особенно полезен при тестировании пользовательского интерфейса, например, с фреймворком Selenium.

```
@RepeatedTest(value = 5, name = "{displayName} {currentRepetition}/{totalRepetitions}")
@DisplayName("RepeatingTest")
void customDisplayName(RepetitionInfo repInfo, TestInfo testInfo) {
    int i = 3;
    System.out.println(testInfo.getDisplayName() +
            "-->" + repInfo.getCurrentRepetition()
    );
    assertEquals(repInfo.getCurrentRepetition(), i);
}
```



Задание

TEL-RAN
by Starta Institute

1 Напишите метод линейного или бинарного поиска.

2 Напишите тесты для метода.

Проблема

Когда работа тестируемого метода зависит от другого метода, который ещё не написан, или вообще от другого сервиса, то вместо реального сервиса можно использовать заглушки.

Такой подход позволяет протестировать только тот функционал, который был добавлен/изменён в рамках текущей доработки.

Затыкать и дурачить

Mockito

Mockito - открытая и довольно популярная библиотека. Обширно используется при тестировании, например, Spring Boot приложений. Mockito также относится к модульному тестированию и позволяет создавать и настраивать макеты объектов для тестов.

Затыкать и дурачить

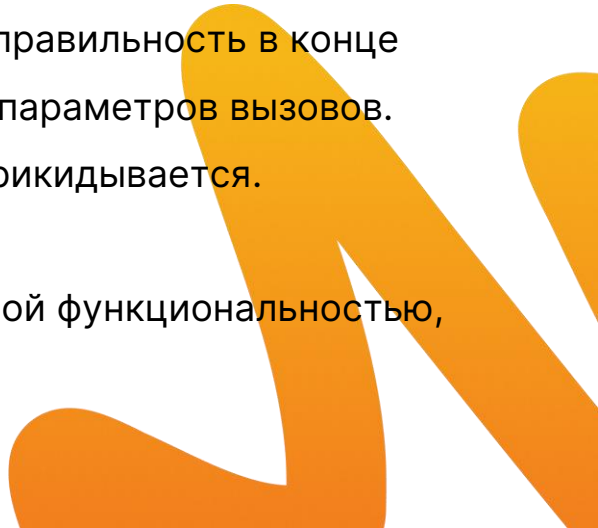
Mockito

TEL-RAN
by Starta Institute

Заглушки (test stub) - используются для получения данных из внешней зависимости, подменяя её. При этом заглушка игнорирует все данные поступающие из тестируемого объекта, возвращая заранее определённый результат.

Шпионы (test spy) - разновидность заглушки, которая умеет протоколировать сделанные к ней обращения из тестируемой системы, чтобы проверить их правильность в конце теста. При этом фиксируется количество, состав и содержание параметров вызовов. Шпион может вызывать настоящие методы объекта, которым прикидывается.

Фикция (mock object) похож на шпиона, но обладает расширенной функциональностью, заранее заданными поведением и реакцией на вызовы.



Задание

TEL-RAN
by Starta Institute

0 Добавьте зависимость на Junit 5 и Mockito https://mvnrepository.com/artifact/org.mockito

1 Создайте метод, который принимает Scanner и с помощью него принимает от пользователя числа, пока не будет введено stop. Метод возвращает список полученных чисел.

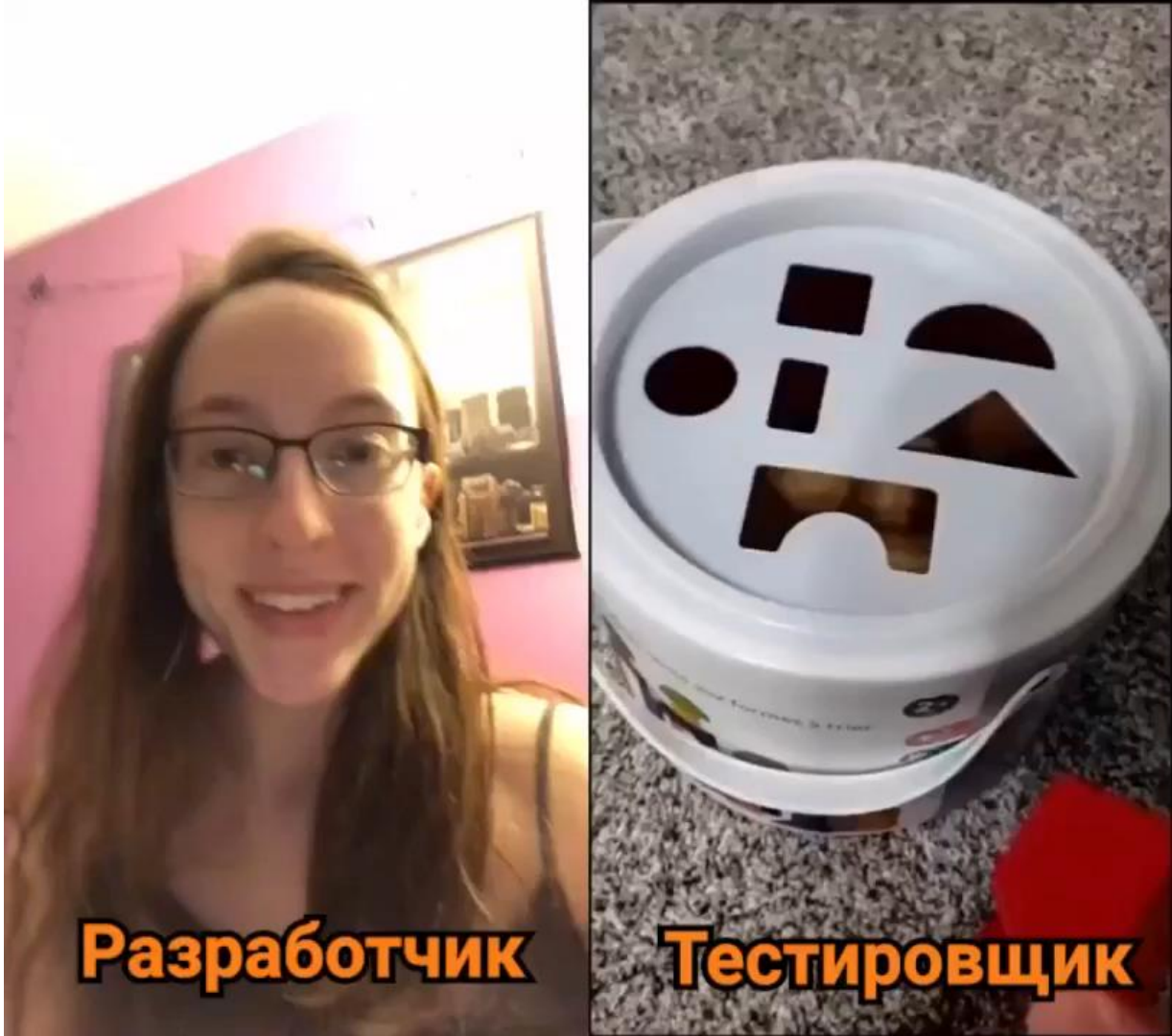
2 Напишите тест для проверки работы метода. Создайте mock для класса Scanner, чтобы эмулировать ввод пользователя.

Проблема

TEL-RAN
by Starta Institute

Можно ли вообще не писать тесты? Это же дополнительная работа.

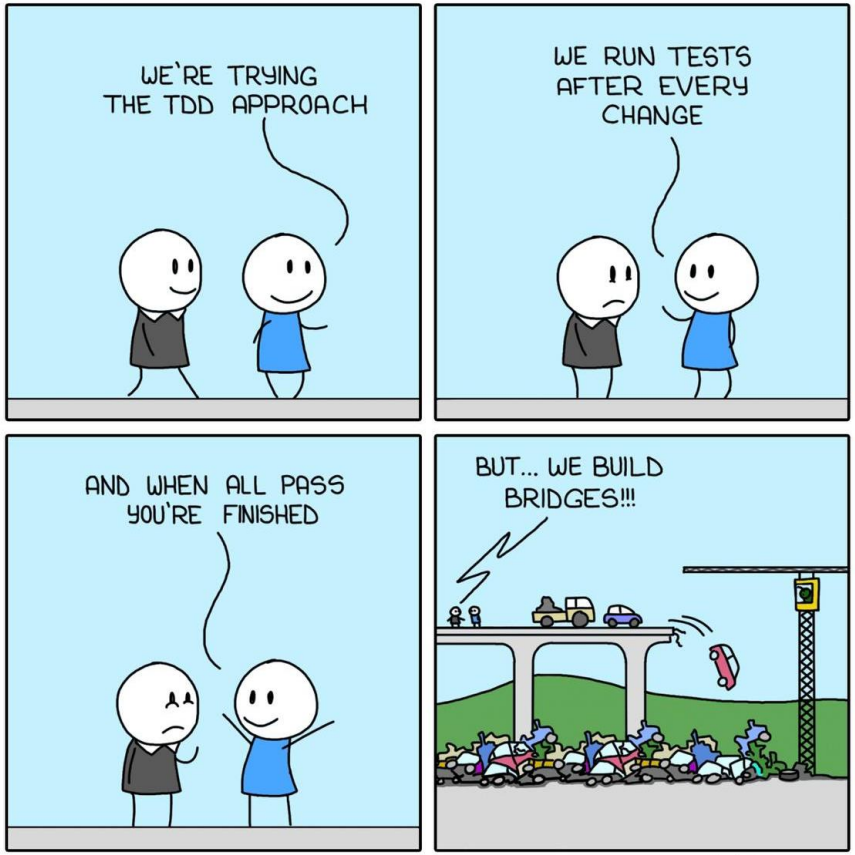
Ответ



TDD

Пишите тесты для самопроверки. Тогда меньше придётся отвечать за свои же ошибки.

Есть подход **Test driven development (TDD)** – подход к разработке ПО, при котором тест пишется раньше, чем сам тестируемый метод. И каждое изменение в коде сопровождается запуском теста.



TEL-RAN
by Starta Institute

2 Домашнее задание

Домашнее задание

TEL-RAN
by Starta Institute

Напишите тесты к классу CommandLineParser (см. домашнее задание по теме «Исключения»).

Тесты должны проверять работу публичных методов и выбрасываемые исключения. При наличии условных конструкций в методе должны проверяться все ветки.

ЗАКЛЮЧЕНИЕ

TEL-RAN
by Starta Institute