Лабораторная работа № 6. Unit-тестирование. Фреймворк TestNG.

Цель работы

Получение навыков работы с Unit-тестами на примере языка Java и библиотеки TestNG.

Теоретическая часть

1.1. Основные понятия и определения

TestNG – это фреймворк для тестирования, написанный на языке Java. Он взял многое из таких фреймворков, как JUnit и NUnit, а также внедрил новые инновационные функции.

Gradle — система автоматической сборки, построенная на принципах Apache Ant и Apache Maven, но предоставляющая DSL (язык описания данных, domain-specific language) на языке Groovy вместо традиционной XML-образной формы представления конфигурации проекта.

TestNG является мощным и простым инструментом для тестирования.

Создаваемые файлы

При создании проекта Gradle создает файлы settings.gradle и build.gradle. Рассмотрим каждый из них.

Файл build.gradle мы можем рассматривать как «сердце» проекта. Для нашего примера он будет выглядеть следующим образом.

```
plugins {
id 'java'
}

group 'radik.labs'
version '1.0-SNAPSHOT' // версия проекта - начальная

repositories {
mavenCentral()    //    подключение    репозитория    с
зависимостями
}

buildDir = "target"

dependencies {
testCompile 'org.testng:testng:6.14.3' // зависимости
для тестов
}
```

Он состоит из кода, основанного на Groovy DSL для описания сборок. Здесь мы можем определить наши зависимости, а также добавить такие вещи, как репозитории Maven, используемые для разрешения зависимостей.

Gradle также генерирует файл settings.gradle:

rootProject.name = 'lab.devskills'

Settings.gradle предназначен для того, чтобы выяснить какие проекты должны участвовать в сборке.

1.2. Пример тестового класса

Содержимое класса Calc.java:

```
package com.devcolibri.logic;
public class Calc {

    public int sum(int a, int b){
        return a + b;
    }
}
```

Содержимое класса CalcTest.java:

```
package com.devcolibri.logic;
import org.testng.annotations.Test;

public class CalcTest {
    private Calc calc = new Calc(); // экземпляр
класса который мы будем тестировать
    private Calc calc = new Calc();

    @Test
    public void testSum() throws Exception {
Assert.assertEquals(5, calc.sum(2,3));
    }
}
```

Аннотация @Test говорит о том, что данный метод является тестовым и может запускаться в отдельном потоке.

1.3. Исключения

Отловить ожидаемую ошибку можно с помощью аннотации @Test через параметр expectedExceptions:

```
package com.devcolibri.logic;

import org.testng.Assert;
import org.testng.annotations.Test;

import java.util.List;

public class TestExpectedExceptionTest extends Assert
{

    @Test(expectedExceptions =
NullPointerException.class)
    public void testNullPointerException() {
        List list = null;
        int size = list.size();
    }
}
```

где expectedExceptions – ожидаемое исключение(ошибка).

В этом случае тест пройдет успешно, так как мы ожидаем, что данный тест выбросит нам NullPointerException (т.к. у ссылки null нельзя выполнить ни одного метода без получения NPE).

1.4. Группа команд Asserts and assertions

Для проверки участков кода тестовый метод должен проверять какие-то условия. В TestNG для проверки служат команды группы Assert. Рассмотрим самые распространённые из них

Assert.assertEqual(String actual,String expected) : Проверяет две строки на идентичность. Если они не идентичны, выбрасывает AssertionError.

Параметры:

actual – значение, полученное из проверяемого метода

expected – ожидаемое значение

Assert.assertEqual(String actual,String expected, String message) : Проверяет две строки на идентичность, но так же выводит сообщение message при неэквивалентных строках. Если они не идентичны, выбрасывает AssertionError с сообщением message.

Параметры:

message – сообщение с тектом ошибки

Assert.assertEquals(boolean actual,boolean expected) : Проверяет, чтобы два выражения логического типа были эквивалентны. Если это не так, выбрасывает AssertionError.

Assert.assertTrue(condition) : Проверяет истинность выражения. Если это не так, выбрасывает AssertionError.

Параметры:

condition – условие для выполнения

Assert.assertTrue(condition, message): Проверяет истинность значения. Если это не так, выбрасывает AssertionError с сообщением message.

Assert.assertFalse(condition) : Проверяет ложность выражения. Если это не так, выбрасывает AssertionError.

Assert.assertFalse(condition, message) : Проверяет ложность выражения. Если это не так, выбрасывает AssertionError с сообщением message.

2. Работа в интегрированной среде разработки.

Запуск тестов (около имени класса нажимаем зелёную стрелку) проиллюстрирован на рисунке 13:

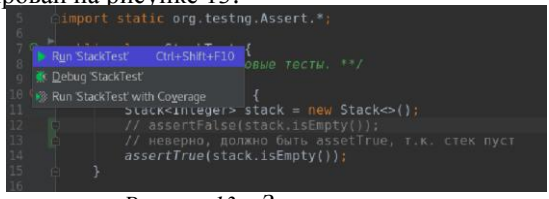

Рисунок 13 – Запуск тестов

Результат запуска тестов – на рисунке 14.

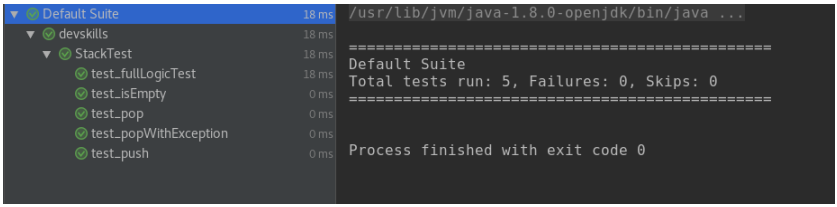

Рисунок 14 – Результат запуска тестов

Результаты проверки покрытия: (запускается так же как и тесты, но используется команда «Run with coverage») (рисунок 15).

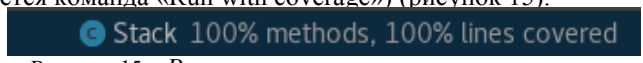

Рисунок 15 – Результат проверки покрытия в классе

Практическая часть

Вам дан код, частично покрытый тестами. Вам нужно написать тесты на остальную часть кода, а также исправить ошибки в уже существующих коде и тестах (1 ошибка в коде и 1 в одном из тестов).

Замечание: замените * на номер вашего варианта.

1. Клонируйте код из репозитория https://lexie62rus@bitbucket.org/lexie62rus/iunit-testirovanie.git.

2. Вы должны работать с программным кодом, находящимся в папке variant_*.

3. Найдите одну ошибку в логике работы программы и одну в уже существующих тестах. Прокомментируйте, как вы их исправили (в отчёте и комментариями в коде). Неправильные строки оставьте закомментированными, не удаляйте их.

4. Допишите тесты на непокрытую тестами часть кода. Тесты должны покрывать основной функционал и ветви с выбросом исключений. Также добавьте тест, полностью проверяющий всю логику работы класса (все методы класса), назовите его test_fullLogicTest.

5. Доведите покрытие тестами до 100%.

6. Опишите в отчёте, какими методами вы пользовались для тестирования.

Содержание отчёта

По результатам выполнения работы оформляется отчет в соответствии с требованиями ГОСТ 7.32-2017 «Отчет о научно-исследовательской работе. Структура и правила оформления», включающий:

- титульный лист;
- цель работы;
- задание согласно вашему варианту;

- исходный код программы для тестирования (можно сразу с исправленной ошибкой и комментариями к ней);
- исходные коды тестов;
- тестовый класс после того, как будут разработаны тесты, комментарии к методам — какие методы из библиотеки были использованы (различные assert'ы). По названию теста должно быть понятно, какой метод он тестирует;
- какая ошибка была исправлена в уже существующем тесте;
- результат запуска тестов;
- результаты проверки покрытия;
- выводы.

Контрольные вопросы:

1. Что представляет из себя автоматизация сборки? Какие действия включает?
2. Перечислите преимущества автоматической сборки.
3. Какие файлы создаются системой сборки Gradle, за что отвечает каждый из них?
4. Назовите несколько самых распространённых методов типа Assert.
5. Назовите несколько самых распространённых аннотаций TestNG.
6. Как проверить в тесте, что метод выбрасывает исключение?
7. Как игнорировать тест?
8. Как ограничить время проведения теста для определённого тестового метода?
9. Отличие зависимых тестов от независимых. Каких из них следует избегать и почему?