Plan

2023-09-21

1. Homework Review

---

1. Разбор домашнего задания

Theory

▶ **English**

▼ **На русском**

Intro

- Ранее мы проверяли правильность работы созданных нами методов, выполняя несколько раз аппликацию, с различными наборами исходных данных, и сравнивая результат полученный в консоли с ожидаемым. Это очень не удобно. Например любое изменение кода, требует повторения всей вышеописанной процедуры заново. Существуют специальные библиотеки, которые позволяют облегчить и автоматизировать этот процесс. Одна из самых популярных для Java, это JUnit

- Работа с JUnit заключается в создании класса с тестами и настройке его. В классе в качестве полей мы можем указать объекты для тестирования и необходимые данные. В методе помеченном аннотацией @BeforeEach мы даем начальные настройки тестируемому объекту. А в методах помеченных аннотацией @Test мы пишем код проверяющий тестируемый объект. Проверки осуществляются при помощи методов начинающихся со слова assert... (которых существует большое колличество на все случаи), и принимающих для сравнения полученный результат и ожидаемый. Все указанные методы и аннотации предоставляются библиотекой JUnit. Каждый метод помеченный аннотацией @Test, по сути является отдельным "мейном".

Зачем нужны юнит-тесты:

1. **Обеспечение качества кода:** Проверяют, работает ли код правильно.
2. **Улучшение структуры кода:** Заставляют делать код более модульным и чистым. 3 **Предотвращение регрессий:** Помогают избегать ошибок после изменений. 4 **Экономия времени и ресурсов:** Позволяют выявлять проблемы раньше. 5 **Повышение уверенности:** Делают код надежным.

Пример:

Представьте, что вы разрабатываете программное обеспечение для автомобильной системы управления двигателем. Эта система контролирует множество параметров, включая топливо, температуру и выхлопные газы, чтобы обеспечить безопасную и эффективную работу двигателя. Важно, чтобы эта система работала без сбоев, чтобы предотвратить потенциально опасные ситуации.

Без юнит-тестов:

- Если вы не пишете юнит-тесты, вы можете внести изменения в код системы управления двигателем, не замечая, что они повредили какой-то аспект работы системы.
- Это может привести к тому, что двигатель начнет работать некорректно, но проблема не будет обнаружена до тех пор, пока автомобиль не будет на дороге, что может стать серьезной угрозой безопасности.

С юнит-тестами:

- Если вы напишете юнит-тесты для каждой функции в системе управления двигателем, вы сможете обнаруживать ошибки на ранних стадиях разработки.
- Например, если после изменения кода один из тестов начнет завершаться неудачей, это будет сигналом, что что-то не так с работой системы.
- Вы сможете легко найти и исправить ошибку до того, как она повлечет за собой серьезные проблемы.

Таким образом, юнит-тесты помогают обеспечивать безопасность и надежность автомобильных систем, а также предотвращать потенциально опасные ситуации на дороге.

Что такое JUnit?

JUnit - это фреймворк для юнит-тестирования (*англ. unit testing*) в Java, который помогает разработчикам проверять правильность функционирования отдельных частей кода (юнитов). Он спроектирован так, чтобы облегчить процесс создания и запуска тестовых сценариев для вашего кода.

Основные концепции JUnit:

1. **Аннотации JUnit:** JUnit использует аннотации для определения тестовых методов. Две основные аннотации - `@Test`, которая помечает метод как тестовый, и `@Before` / `@After`, которые выполняются перед и после каждого теста.
2. **Утверждения (Assertions):** Для проверки ожидаемых результатов тестов используйте методы из класса `org.junit.Assert` или его статические импорты, например, `assertEquals`, `assertTrue`, `assertFalse` и другие.

3. **Тестовые наборы:** Вы можете группировать тесты в классы и пакеты с использованием аннотации `@RunWith` и `@Suite`.

Сценарии тестирования

Позитивное тестирование (Positive Testing):**

Позитивное тестирование - это методика проверки функциональности программы на правильность выполнения в рамках ожидаемых и допустимых сценариев. В этом случае тесты выполняются с использованием входных данных, которые соответствуют ожидаемым условиям использования программы.

Примеры позитивных тестов:

- Проверка сложения двух положительных чисел.
- Проверка входа в систему с корректными учетными данными.
- Проверка, что веб-страница открывается без ошибок при корректном URL.

Негативное тестирование (Negative Testing):

Негативное тестирование - это направлено на проверку того, как программа обрабатывает недопустимые или некорректные данные и сценарии. Целью негативного тестирования является обнаружение и проверка на корректность обработки ошибок и исключительных ситуаций.

Примеры негативных тестов:

- Попытка деления на ноль.
- Попытка входа в систему с неправильными учетными данными.
- Попытка открыть несуществующую веб-страницу.

Позитивное и негативное тестирование вместе обеспечивают полное покрытие тестами и помогают убедиться, что ваше программное обеспечение работает как в ожидаемых, так и в непредвиденных сценариях.

Основные методы класса Assertions в JUnit:

Конечно, вот основные методы класса Assertions в виде таблицы:

Метод	Описание
`assertEquals(expected, actual)`	Проверяет, что `expected` равно `actual`.
`assertNotEquals(expected, actual)`	Проверяет, что `expected` не равно `actual`.
`assertTrue(condition)`	Проверяет, что `condition` истинно.
`assertFalse(condition)`	Проверяет, что `condition` ложно.
`assertNull(object)`	Проверяет, что `object` является `null`.
`assertNotNull(object)`	Проверяет, что `object` не является `null`.

Метод	Описание
`assertArrayEquals(expectedArray, actualArray)`	Проверяет, что два массива expected и actual равны по содержимому.

Именование тестовых методов (шестерни) в JUnit:

Хорошее именование тестовых методов важно для понимания их назначения без необходимости анализа кода. Вот рекомендации:

- Используйте осмысленные имена, отражающие суть теста.
- Начинайте имена тестов с глаголов, таких как "test" или "verify".
- Используйте подчеркивания или camelCase.
- Будьте специфичны и избегайте общих имен, таких как "test1" или "test2".

Примеры правильного именования тестовых методов:

- `testAdditionWithPositiveNumbers`
- `testDivisionByZero`
- `testInvalidInputHandling`

Практические задания для обучения

Задание 1: Создайте класс `Calculator`, который содержит методы для выполнения математических операций (сложение, вычитание, умножение, деление и т. д.). Напишите юнит-тесты для этих методов, проверяя правильность результатов с использованием утверждений JUnit.

Задание 2: Создайте класс `StringUtils`, который содержит методы для работы со строками (например, конкатенация строк, поиск подстроки и т. д.). Напишите тесты для этих методов с разными входными данными и ожидаемыми результатами.

Помните, что хорошее тестирование включает в себя как позитивные, так и негативные тесты, чтобы убедиться, что ваш код работает правильно во всех сценариях.

Полезные ссылки:

- User Guide - https://junit.org/junit5/docs/current/user-guide/index.html
- How to set up JUnit for your project - https://www.jetbrains.com/help/idea/junit.html

Homework

▶ **English**

▼ **На русском**

Задача 1

Написать/дописать тесты для ...

Code

code/Lesson_27/src/CalculatorTest.java

```java
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

/**
 * @author Andrej Reutow
 * created on 12.10.2023
 */
public class CalculatorTest {

//    @Test
//    public void tes_add_sum4() {
//        int expectedResult = 4;
//        int result = Calculator.add(2, 2);
//
//        Assertions.assertEquals(expectedResult, result);
//    }
}
```

code/Lesson_27/src/PasswordEncoderDecoder.java

```java
import java.util.Random;
import java.util.Scanner;

/**
 * @author Andrej Reutow
 * created on 12.10.2023
 */

/*
### Задание 2* Программа кодирования и декодирования пароля

### Техническое задание

#### Цель:

Разработать программу на Java для кодирования и декодирования пароля с испо
```

- Каждый четный индекс символа пароля сдвигается на 1000 символов вправо в U
- Каждый нечетный индекс символа пароля сдвигается на 10 символов влево в U
- В закодированный пароль добавляются 6 случайных символов из таблицы Unico

Функциональные требования:

1. Программа должна иметь два основных метода: `encodePassword` для кодирова декодирования пароля.
2. Метод `encodePassword` должен принимать строку с паролем в качестве вход в виде строки.
3. Метод `decodePassword` должен принимать закодированный пароль в виде стр строки.
4. Программа должна добавлять 6 случайных символов из Unicode в закодирован

Нефункциональные требования:

1. Программа должна быть простой в использовании через текстовый интерфейс
2. Программа должна быть легко читаемой и поддерживаемой.
3. Программа должна быть протестирована (smoke test - https://ru.wikipedia.

```
 */
public class PasswordEncoderDecoder {

    public static final int N = 6;

    /**
     * для кодирования пароля
     */
    public static String encodePassword(String password) {
        if (password == null) {
            return null;
        }

        String encodedPassword = ""; // переменная в которую запишем зашифр

        for (int i = 0; i < password.length(); i++) {
            char originalChar = password.charAt(i);

            if (i % 2 == 0) {
                encodedPassword += (char) (originalChar + 1000);
            } else {
                encodedPassword += (char) (originalChar - 10);
            }
        }
```

```
        Random random = new Random();
        for (int i = 0; i < N; i++) {
            char randomChar = (char) random.nextInt(Character.MAX_CODE_POIN
            encodedPassword += randomChar;
        }

        return encodedPassword;
    }

    public static String decodePassword(String encodedPassword) {
        String result = "";

        for (int i = 0; i < encodedPassword.length() - N; i++) {
            char encodedChar = encodedPassword.charAt(i);

            if (i % 2 == 0) {
                result += (char) (encodedChar - 1000);
            } else {
                result += (char) (encodedChar + 10);
            }
        }

        return result;
    }

    public static void main(String[] args) {
        Scanner consoleInput = new Scanner(System.in);

        System.out.println("Menu:");
        System.out.println("1 - encode password");
        System.out.println("2 - decode password");

        String action = consoleInput.nextLine();
        int selectedAction = Integer.valueOf(action);

        if (selectedAction == 1) {
            System.out.println("Please enter your password to encode:");
            String userPassword = consoleInput.nextLine();
            String encodedUserPassword = encodePassword(userPassword);
            System.out.println("Your encoded password:");
            System.out.println(encodedUserPassword);
        } else if (selectedAction == 2) {
```

```
                System.out.println("Please enter your password to decode:");
                String decodedUserPassword = consoleInput.nextLine();
                String encodedUserPassword = decodePassword(decodedUserPassword
                System.out.println("Your decoded password:");
                System.out.println(encodedUserPassword);
            } else {
                System.out.println("Error, this action not present");
            }
        }

    }
```

code/Lesson_27/src/PasswordEncoderDecoderTest.java

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

    /**
     * @author Andrej Reutow
     * created on 12.10.2023
     */
    public class PasswordEncoderDecoderTest {


        public void testDecodePasswordPasswordDecoded() {

        }

        @Test
        public void test_decodePassword_passwordDecoded() {
            // Дано:
            String password = "QWERTZ123";

            // Когда
            String encodedPassword = PasswordEncoderDecoder.encodePassword(pass
            String decodedPassword = PasswordEncoderDecoder.decodePassword(enco

            // Тогда

            Assertions.assertEquals(password, decodedPassword);
        }

        @Test
        public void test_encodePassword_encodedPassword() {
```

```
        String password = "qWerty123456";

        String encodedPassword = PasswordEncoderDecoder.encodePassword(pass

        Assertions.assertNotNull(encodedPassword);
        Assertions.assertNotEquals(password, encodedPassword);
        Assertions.assertEquals(password.length() + 6, encodedPassword.leng
    }

    @Test
    public void test_encodePassword_emptyString_encodedPassword() {
        String password = "";

        String encodedPassword = PasswordEncoderDecoder.encodePassword(pass

        Assertions.assertNotNull(encodedPassword);
        Assertions.assertNotEquals(password, encodedPassword);
        Assertions.assertEquals(6, encodedPassword.length());
    }


    @Test
    public void test_encodePassword_passIsNull_resultNull() {
        String password = null;

        String encodedPassword = PasswordEncoderDecoder.encodePassword(pass

        Assertions.assertNull(encodedPassword);
    }
}
```