Лабораторная работа 4

по курсу Технология программирования

на тему «Основы тестирования JUnit»

Составил: Питикин А.Р.

Кафедра ИУ-3

Москва, 2024

Введение

Тестирование является важнейшей частью разработки. Не даром, великий говорил – «Тестирование – важнейшее из искусств»☺ В настоящее время умение писать юнит-тесты является одним из ключевых навыков любого программиста.

Теоретическая часть

JUnit - это популярный фреймворк для тестирования Java-приложений. Он предоставляет программистам инструменты и структуры для написания, организации и выполнения автоматизированных тестов, которые помогают проверить корректность работы кода в различных сценариях.

Вот некоторые ключевые аспекты JUnit:

1. **Тестовые методы**: В JUnit тесты представлены в виде методов в классах, аннотированных аннотацией @Test. Эти методы могут проверять ожидаемые результаты выполнения определенных частей вашего кода.

2. **Утверждения (Assertions)**: Для проверки ожидаемых результатов JUnit предоставляет набор методов утверждений (assertions), таких как assertEquals, assertTrue, assertNotNull и другие. С помощью утверждений можно сравнивать значения переменных, проверять условия и т.д.

3. **Жизненный цикл тестов**: JUnit обеспечивает структурированный жизненный цикл для тестовых методов с аннотациями, такими как @Before, @After, @BeforeClass, @AfterClass. Например, аннотация @Before позволяет выполнять определенный код перед запуском каждого тестового метода.

4. **Другие возможности**: JUnit также поддерживает параметризованные тесты, тестирование исключений, исполнение тестов в различных порядках и т.д.

Использование JUnit позволяет повысить качество программного кода и ускорить процесс разработки за счет автоматизации тестирования. Он широко применяется в индустрии разработки программного обеспечения для тестирования Java-приложений всех уровней сложности.

Практическая часть

Лучше приведем сразу пример. Проект очень простой – есть два класса, Animal и Cat:

```java
public class Animal {
    private String name;
    private int age;

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public void makeSound() {
        System.out.println("Animal makes a sound");
    }
}
```

```java
public class Cat extends Animal {
    private String color;

    public Cat(String name, int age, String color) {
        super(name, age);
        this.color = color;
    }

    public String getColor() {
        return color;
    }

    @Override
    public void makeSound() {
        System.out.println("Meow!");
    }
}
```

Разумеется, логика ваших методов значительно сложнее, пример носит исключительно иллюстративный характер. Предположим, нам нужно протестировать данные методы, вдруг, в процессе нашего программирования, кот начал говорить другим голосом? Всякое бывает в Bauman`s Gate…

Создадим отдельный класс:

```java
import org.testng.annotations.Test;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

import static org.testng.AssertJUnit.assertEquals;

public class AnimalTest {

    @Test
    public void testAnimalProperties() {
```

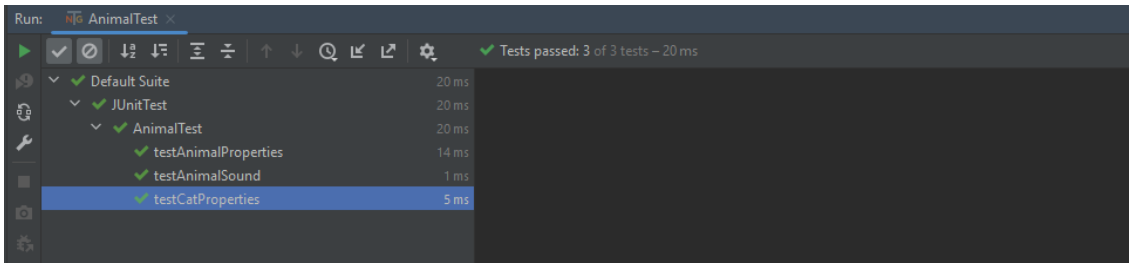
```
        Animal animal = new Animal("Lion", 5);
        assertEquals("Lion", animal.getName());
        assertEquals(5, animal.getAge());
    }

    @Test
    public void testAnimalSound() {
        // Перенаправляем стандартный поток вывода в ByteArrayOutputStream
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        System.setOut(new PrintStream(outputStream));

        Animal animal = new Animal("Lion", 5);
        // Вызываем метод, который выводит текст на консоль
        animal.makeSound();
        // Получаем вывод из ByteArrayOutputStream в форме строки
        String consoleOutput = outputStream.toString().trim();
        assertEquals("Animal makes a sound", consoleOutput);
        // Важно вернуть стандартный поток вывода в исходное состояние
        System.setOut(System.out);
    }

    @Test
    public void testCatProperties() {
        Cat cat = new Cat("Whiskers", 3, "Grey");
        assertEquals("Whiskers", cat.getName());
        assertEquals(3, cat.getAge());
        assertEquals("Grey", cat.getColor());
    }
}
```

После запуска тестового класса получим нечто подобное:



Другие аннотации, Before, After и тд полезны тогда, когда нужно некоторый код выполнить до тестов. Например, создать необходимые объекты, добавить юнитов и тд.

Задание

Плох тот программист, который плох. У нас таких нет, поэтому мы пишем тесты к Bauman`s Gate! Итак, в рамках данной ЛР будем покрывать наш код…тестами.

Чтобы было несколько проще, предлагается в рамках этой ЛР написать тесты к ЛР2 или к ЛР3, по вашему выбору. Условимся, что при выборе

выполнения данной ЛР к ЛР2 максимальное количество баллов будет составлять 60% от максимума.

В таблице ниже описан функционал, который необходимо покрыть тестами и для тех, кто выберет ЛР2 и для тех, кто выберет ЛР3:

1. Корректность завершения игры победой игрока
2. Корректность завершения игры победой бота
3. Корректность штрафов перемещения
4. Корректность дальности атаки
5. Корректность совершения атаки
6. Корректность возможности перемещения(нельзя быть на клетке с другим существом, нельзя выйти за пределы поля и тд)
7. Корректность смерти юнита
8. Корректность того, как работает защита юнита
9. Корректность начальной покупки юнитов в магазине
10. Корректность действий бота
11. Корректность отображения поля сражения
12. Корректность вашего индивидуального задания к ЛР2(тут зависит от задания, тестов может быть несколько)

Для тех, кто выберет чуть более сложный путь, предлагаются дополнительные тесты к ЛР3:

1. Корректность создания новой карты
2. Корректность возможности добавления новых препятствий и обозначений
3. Корректность возможности редактирования созданных карт
4. Корректность загрузки ранее созданной карты
5. Корректность увеличения уровня зданий
6. Корректность спец зданий – Академии, Рынка, Ремесленной мастерской
7. Корректность сохранения прогресса в прокачке зданий