**Задание №4: Разработка через тестирование.**

**Вариант 9**

**Сергиенко Лев, 12 группа, МСС**

**Вариант:**

Определить класс «Текст» - хранит последовательность предложений. Определить несколько конструкторов и методы: добавить предложение, удалить предложение, вставить предложение, количество (букв, слов, предложений), равны ли два текста.

## **Отчёт о разработке класса «Текст» по методологии TDD**

### **1. Планирование и постановка задачи**

**Задача:**Создать класс **Text**, который хранит последовательность предложений. Реализовать следующие возможности:

* **Конструкторы:** конструктор по умолчанию и конструктор, принимающий список предложений.
* **Методы:**
  + Добавить предложение.
  + Удалить предложение по индексу.
  + Вставить предложение по индексу.
  + Подсчитать количество букв, слов, предложений.
  + Переопределить метод **equals** для сравнения двух объектов.

**Особенности:**

* Обработка ошибочных ситуаций с помощью исключений.

Перед началом разработки было решено использовать подход TDD (Test Driven Development) для последовательного внедрения функциональности.

### 

### **2. Этапы разработки по TDD**

### Сначала определим наш тестовый класс:

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.*\**;

import lab5.*\**;

public class TextTest {

private Text text;

@Before

public void setUp() {

text = new Text();

}

@Test

public void testEmptyTextHasZeroSentences() {

assertEquals(0, text.getSentenceCount());

}

}

Этот простой тест не пройдет, потому что мы еще не создали класс Text. Следуя TDD, мы теперь реализуем ровно столько кода, чтобы этот тест прошел:

package lab5;

import java.util.ArrayList;

import java.util.List;

public class Text {

private List<String> sentences;

public Text() {

*this*.sentences = new ArrayList<>();

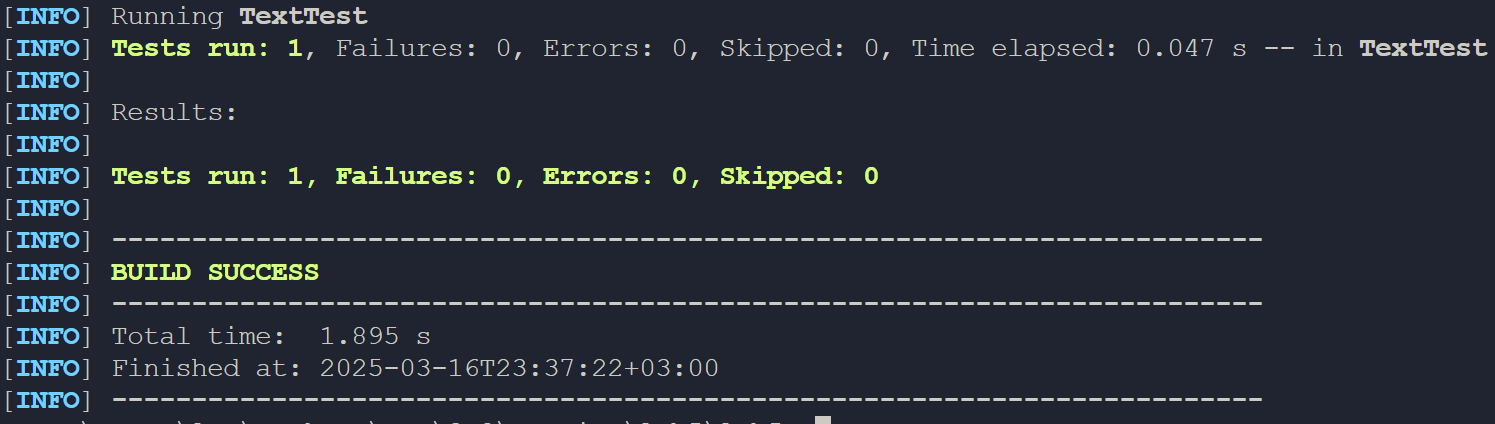
}

public int getSentenceCount() {

return sentences.size();

}

}



### Теперь продолжим цикл **"красный-зеленый-рефакторинг"**, добавляя больше тестов. Протестируем добавление предложения:

@Test

public void testAddSentence() {

text.addSentence("Это тестовое предложение.");

assertEquals(1, text.getSentenceCount());

assertEquals("Это тестовое предложение.", text.getSentence(0));

}

Чтобы этот тест прошел, нам нужно реализовать:

public void addSentence(String sentence) {

sentences.add(sentence);

}

public String getSentence(int index) {

return sentences.get(index);

}

Добавим тесты для остальной функциональности:

@Test

public void testDeleteSentence() {

text.addSentence("Первое предложение.");

text.addSentence("Второе предложение.");

text.deleteSentence(0);

assertEquals(1, text.getSentenceCount());

assertEquals("Второе предложение.", text.getSentence(0));

}

@Test

public void testInsertSentence() {

text.addSentence("Первое предложение.");

text.addSentence("Третье предложение.");

text.insertSentence(1, "Второе предложение.");

assertEquals(3, text.getSentenceCount());

assertEquals("Второе предложение.", text.getSentence(1));

}

@Test

public void testWordCount() {

text.addSentence("Это тестовое предложение.");

text.addSentence("Еще одно тестовое предложение.");

assertEquals(7, text.getWordCount());

}

@Test

public void testLetterCount() {

text.addSentence("АБВ.");

text.addSentence("ГДЕ.");

assertEquals(6, text.getLetterCount());

}

@Test(expected = IndexOutOfBoundsException.class)

public void testDeleteSentenceInvalidIndex() {

text.deleteSentence(0);

}

@Test

public void testTextEquality() {

Text text1 = new Text();

text1.addSentence("Тестовое предложение.");

Text text2 = new Text();

text2.addSentence("Тестовое предложение.");

Text text3 = new Text();

text3.addSentence("Другое предложение.");

assertTrue(text1.equals(text2));

assertFalse(text1.equals(text3));

}

@Test

public void testConstructorWithSentences() {

String[] sentences = { "Первое.", "Второе." };

Text textWithSentences = new Text(sentences);

assertEquals(2, textWithSentences.getSentenceCount());

assertEquals("Первое.", textWithSentences.getSentence(0));

assertEquals("Второе.", textWithSentences.getSentence(1));

}

@Test

public void testConstructorWithText() {

Text originalText = new Text();

originalText.addSentence("Оригинальное предложение.");

Text copiedText = new Text(originalText);

assertEquals(1, copiedText.getSentenceCount());

assertEquals("Оригинальное предложение.", copiedText.getSentence(0));

}

Теперь, следуя подходу TDD, давайте реализуем полный класс Text, чтобы все эти тесты прошли:

package lab5;

import java.util.ArrayList;

import java.util.List;

import java.util.Arrays;

import java.util.Objects;

public class Text {

private List<String> sentences;

*/\*\**

*\* Конструктор по умолчанию - создает пустой текст*

*\*/*

public Text() {

*this*.sentences = new ArrayList<>();

}

*/\*\**

*\* Конструктор с массивом предложений*

*\**

*\* @param sentences Массив предложений для инициализации текста*

*\*/*

public Text(String[] sentences) {

*this*.sentences = new ArrayList<>(Arrays.asList(sentences));

}

*/\*\**

*\* Конструктор копирования - создает новый текст как копию другого*

*\**

*\* @param other Текст для копирования*

*\*/*

public Text(Text other) {

*this*.sentences = new ArrayList<>(other.sentences);

}

*/\*\**

*\* Добавить предложение в конец текста*

*\**

*\* @param sentence Предложение для добавления*

*\*/*

public void addSentence(String sentence) {

sentences.add(sentence);

}

*/\*\**

*\* Удалить предложение по указанному индексу*

*\**

*\* @param index Индекс предложения для удаления*

*\* @throws IndexOutOfBoundsException если индекс вне диапазона*

*\*/*

public void deleteSentence(int index) {

if (index < 0 || index >= sentences.size()) {

throw new IndexOutOfBoundsException("Недопустимый индекс предложения: " + index);

}

sentences.remove(index);

}

*/\*\**

*\* Вставить предложение по указанному индексу*

*\**

*\* @param index Позиция для вставки предложения*

*\* @param sentence Предложение для вставки*

*\* @throws IndexOutOfBoundsException если индекс вне диапазона*

*\*/*

public void insertSentence(int index, String sentence) {

if (index < 0 || index > sentences.size()) {

throw new IndexOutOfBoundsException("Недопустимый индекс предложения: " + index);

}

sentences.add(index, sentence);

}

*/\*\**

*\* Получить количество предложений в тексте*

*\**

*\* @return Количество предложений*

*\*/*

public int getSentenceCount() {

return sentences.size();

}

*/\*\**

*\* Получить предложение по указанному индексу*

*\**

*\* @param index Индекс предложения*

*\* @return Предложение*

*\* @throws IndexOutOfBoundsException если индекс вне диапазона*

*\*/*

public String getSentence(int index) {

if (index < 0 || index >= sentences.size()) {

throw new IndexOutOfBoundsException("Недопустимый индекс предложения: " + index);

}

return sentences.get(index);

}

*/\*\**

*\* Подсчитать общее количество слов в тексте*

*\**

*\* @return Количество слов*

*\*/*

public int getWordCount() {

int count = 0;

for (String sentence : sentences) {

*// Разделяем по пробелам и считаем непустые части*

String[] words = sentence.split("\\s+");

for (String word : words) {

if (!word.isEmpty()) {

count++;

}

}

}

return count;

}

*/\*\**

*\* Подсчитать общее количество букв в тексте*

*\**

*\* @return Количество букв (алфавитных символов)*

*\*/*

public int getLetterCount() {

int count = 0;

for (String sentence : sentences) {

for (char c : sentence.toCharArray()) {

if (Character.isLetter(c)) {

count++;

}

}

}

return count;

}

*/\*\**

*\* Сравнить два текста на равенство*

*\**

*\* @param obj Объект для сравнения*

*\* @return true, если тексты содержат одинаковые предложения, иначе false*

*\*/*

@Override

public boolean equals(Object obj) {

if (*this* == obj)

return true;

if (obj == null || getClass() != obj.getClass())

return false;

Text other = (Text) obj;

return Objects.equals(sentences, other.sentences);

}

*/\*\**

*\* Получить строковое представление текста*

*\**

*\* @return Все предложения, объединенные вместе*

*\*/*

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

for (String sentence : sentences) {

sb.append(sentence);

if (!sentence.endsWith(" ")) {

sb.append(" ");

}

}

return sb.toString().trim();

}

}

### **3. Итоговый результат**

**Реализованный класс «Text»** включает:

* Конструкторы для создания пустого текста и текста с набором предложений.
* Методы для добавления, удаления и вставки предложений с проверкой корректности входных данных.
* Методы для подсчёта букв, слов и предложений.
* Переопределённый метод equals для сравнения двух текстов.

**JUnit-тесты** обеспечивают полное покрытие функциональности:

* Тестирование корректного поведения для валидных и невалидных входных данных.
* Проверка работы методов подсчёта и корректности обработки исключительных ситуаций.

**Демонстрационное приложение** (метод main) показывает пример использования класса, где результаты операций выводятся в консоль.

### **4. Заключение**

Методология TDD позволила разработать класс «Text» шаг за шагом:

1. **Красный этап:** написали тесты, описывающие требуемое поведение, и убедились, что без реализации тесты падают.
2. **Зелёный этап:** реализовали минимальный код, чтобы тесты прошли.
3. **Рефакторинг:** оптимизировали и улучшили качество кода без изменения функциональности.

Такой подход гарантирует, что каждая новая функция покрывается тестами и при внесении изменений разработчик сразу получает обратную связь о том, что ничего не сломалось. Методология TDD способствует созданию чистого, поддерживаемого кода, который легко расширять в будущем.