5 Лабораторная работа №5
Исследование способов модульного тестирования программного обеспечения в среде JUnit/NUnit

5.1. Цель работы

Исследовать эффективность использования методологии TDD при разработке программного обеспечения. Получить практические навыки использования фреймворка JUnit/NUnit для модульного тестирования программного обеспечения.

5.2. Постановка задачи

Был выдан вариант 14.

Реализовать на языке Java/C# один из классов, спроектированных в лабораторной работе № 1. Методы класса при этом не реализовывать. Разработать для созданного класса набор модульных тестов, включающий тесты для каждого метода. Запустить набор тестов, проанализировать и сохранить результаты. Поочередно реализовать методы класса, выполняя тестирование при каждом изменении программного кода. После того, как весь набор тестов будет выполняться успешно, реализацию классов можно считать завершенной.

5.3. Ход выполнения работы

5.3.1. Описание обязанностей тестового класса

Была разработана следующая спецификация:

- Название тестируемого класса: Matrix.

- Описание: проводится тестирование метода columnPositionWithLongestSeries — метода, возвращающего номер столбца, содержащего самую длинную серию одинаковых элементов.
- Название тестового случая: test_longestSeriesInMiddle_1column.
- Входные данные: матрица $\begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & 2 & -1 & -1 \\ 10 & 2 & 10 & 10 \\ 20 & 20 & 20 & 20 \end{pmatrix}$.
- Выходные данные: число 1.

5.3.2. Текст теста

На основе описания в предыдущем пункте был создан тестовый класс MatrixTest, вызывающий public-методы тестируемого класса. Текст класса представлен в листинге 1.

Листинг 1 — Тестовый драйвер MatrixTest

```
package org.cory7666.softwaretestingexample.task1;

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

public class MatrixTest
{
  @Test
  public void test_longestSeriesInMiddle_1column ()
  {
    Assertions
      .assertEquals(
        1,
        new Matrix(new int[][] { { 1, -1, 10, 20 }, { 1, 2, 2, 20 }, { 1, -1, 10, 20 }, { 1, -1, 10, 20 } })
          .columnPositionWithLongestSeries());
  }
}
```

Таким образом каждый из методов, имеющих аннотацию «@Test», в тестовом классе тестирует один метод Matrix#columnPositionWithLongestSeries.

5.3.3. Тестовый запуск и анализ полученных данных

После написания кода программы был запущен тест. Рисунок 5.1 содержит результат проведения теста.

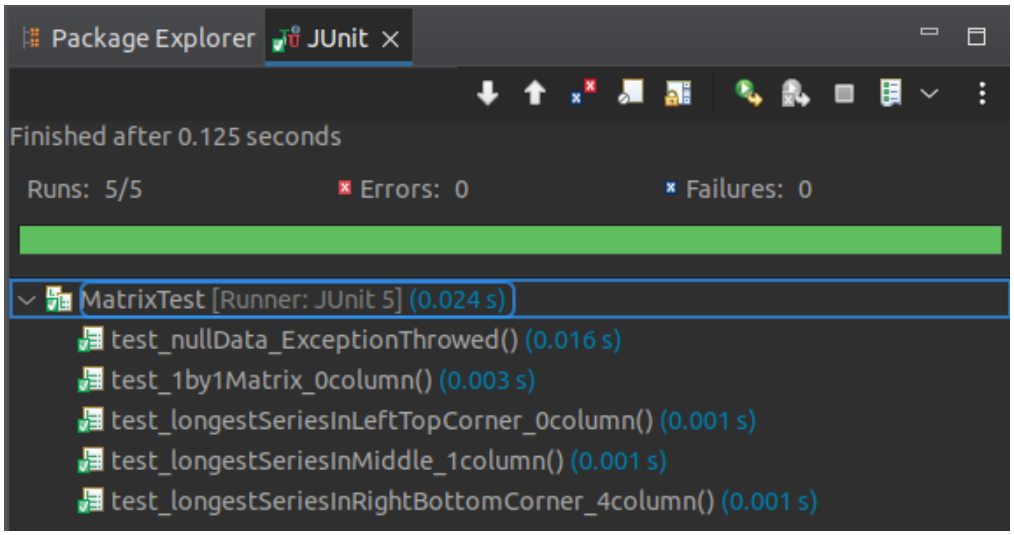

Рисунок 5.1 — Результат работы JUnit теста

Выводы

При выполнении данной лабораторной работы были получены навыки составления модульных тестов в среде JUnit. Также были получены навыки программирования при работе по методологии TDD. Выявлено, что подобного типа тесты следует применять в случаях разработки программы через тестирование.

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем заключаются основные принципы методологии TDD (Разработка через тестирование)?

Разработка через тестирование (Test-driven development, TDD) представляет собой одну из современных «гибких» (agile) методологий разработки программного обеспечения. Эта методология предполагает использование модульного тестирования для контроля разрабатываемого программного кода. Основная идея состоит в том, что модульные тесты разрабатываются до разработки программного кода, который они будут тестировать. Разработка таких тестов заставляет программиста подробно разобраться в требованиях к разрабатываемому модулю до начала разработки, четко определиться с конечными целями разработки; успешное выполнение тестов является критерием прекращения разработки. Разработку через тестирование можно представить в виде последовательности следующих основных этапов:

1) получение программы в непротиворечивом состоянии и набора успешно выполняемых модульных тестов;

2) разработка нового модульного теста;

3) максимально быстрая разработка минимального программного кода, позволяющего успешно выполнить весь набор тестов;

4) рефакторинг разработанного программного кода с целью улучшения его структуры и устранения избыточности;

5) контроль переработанного программного кода с помощью полного набора тестов.

В результате выполнения этих шагов в программу будет добавлена новая функциональность, а работоспособность модифицированной программы будет гарантироваться выполнением полного набора модульных тестов. Такой подход позволяет избежать ситуации, когда добавление новой функции нарушает работоспособность ранее разработанного кода.

2. Какие можно выделить достоинства и недостатки подхода TDD?

Такой подход дает следующие преимущества:

- предотвращается возникновение ошибок во вновь разработанном коде;
- появляется возможность вносить изменения в существующий программный код без риска нарушить его работоспособность, т. к. возникающие ошибки будут сразу же обнаружены модульными тестами;
- модульные тесты могут быть использованы в качестве документации к программному коду, показывать способы обращения к соответствующим программным модулям (объектам, методам и т.п.);
- улучшается дизайн кода: тесты заставляют создавать более «легкие» и независимые компоненты, которые проще поддерживать и модифицировать;
- повышается квалификация разработчиков, т. к. разработка качественных модульных тестов требует глубоких знаний ООП и паттернов проектирования.

Таким образом, применение методологии TDD позволяет создавать более качественный программный код, снижает вероятность возникновения ошибок и ускоряет процесс разработки.

К недостаткам разработки через тестирование можно отнести, во-первых, высокие требования к квалификации разработчиков. Еще одной проблемой является тестовое покрытие программного кода. В идеале набор модульных тестов должен обеспечивать покрытие 100% программного кода. Однако на практике этого достичь не удается, особенно в тех случаях, когда TDD применяется для модификации уже существующего программного обеспечения. Поэтому необходимо обеспечить покрытие тестами наиболее критических модулей, а также компонентов, которые будут подвергаться рефакторингу.

3. В чем состоит общий порядок модульного тестирования с помощью NUnit.

Общий порядок модульного тестирования с помощью NUnit включает следующие шаги:

Установка фреймворка NUnit - необходимо скачать и установить библиотеку NUnit.

Создание проекта для модульного тестирования - в Visual Studio необходимо создать проект типа "Unit Test Project", который будет содержать тестовые методы.

Создание тестовых методов - в созданном проекте необходимо создать тестовые методы, которые будут проверять функциональность отдельных модулей программы.

Написание утверждений (assertions) - в каждом тестовом методе необходимо написать утверждения, которые проверят правильность работы тестируемого модуля программы.

Запуск тестов - после написания тестовых методов необходимо запустить тесты и проверить их результаты.

Анализ результатов тестирования - после завершения тестирования необходимо проанализировать результаты и определить, были ли найдены ошибки или дефекты в тестируемых модулях программы.

Исправление ошибок и повторное тестирование - в случае обнаружения ошибок или дефектов необходимо исправить их и провести повторное тестирование для проверки, что исправления работают правильно.

Использование фреймворка NUnit помогает автоматизировать процесс тестирования и упрощает написание и выполнение тестовых методов. Он предоставляет множество методов для написания утверждений и проведения различных проверок, что позволяет более эффективно тестировать программное обеспечение.

4. Каково назначение основных атрибутов NUnit?

Основные атрибуты NUnit включают:

TestFixture - это атрибут, который помечает класс как тестовый набор (fixture), содержащий набор тестовых методов. Тестовый набор - это группа тестовых методов, которые тестируют один и тот же функциональный блок или компонент приложения.

Test - это атрибут, который помечает метод как тестовый. Тестовый метод - это метод, который содержит код для проверки ожидаемого поведения функционального блока или компонента приложения.

TestCase - это атрибут, который позволяет определить несколько наборов входных данных для одного тестового метода. Каждый набор входных данных представлен в виде аргументов метода.

SetUp - это атрибут, который помечает метод инициализации тестового набора, который вызывается перед каждым тестовым методом в тестовом наборе. Этот метод может использоваться для создания тестовых данных и настройки окружения для каждого тестового метода.

TearDown - это атрибут, который помечает метод очистки тестового набора, который вызывается после каждого тестового метода в тестовом наборе. Этот метод может использоваться для освобождения ресурсов, которые были выделены в методе SetUp.

Category - это атрибут, который позволяет классифицировать тесты по категориям, таким как функциональность, производительность, безопасность и т.д. Категории могут использоваться для запуска только тех тестов, которые относятся к определенной категории.

Ignore - это атрибут, который помечает тестовый метод или тестовый набор, который должен быть проигнорирован во время запуска тестов. Этот атрибут может быть использован для временного отключения тестов в случае, если они не проходят или не актуальны.

Repeat - это атрибут, который позволяет повторить выполнение теста определенное количество раз. Этот атрибут может быть использован для проверки стабильности тестов…

5. Какие типы проверок реализованы в классе Assert фреймворка NUnit?

В нашем случае в методе AddTest() для проверки правильности работы метода Add() класса Calc создается экземпляр калькулятора, выполняется операция

сложения и полученный результат сравнивается с ожидаемым (результатом выполнения одноименной операции в языке C#). Класс Assert, использованный в этом примере, содержит набор методов, позволяющих выполнять различные проверки данных. Если данные неверны, метод оповещает среду выполнения NUnit, что тест не пройден. Ниже приведены некоторые проверки, доступные в классе Assert:

1. Assert.AreEqual – проверяет равенство входных параметров;

2. Assert.AreNotEqual – проверяет то, что входные параметры неравны;

3. Assert.AreSame – проверка на то, что входные параметры ссылаются на один и тот же объект;

4. Assert.AreNotSame – входные параметры не ссылаются на один и тот же объект;

5. Assert.Contains – метод получает на входе объект и коллекцию и проверяет, что данный объект содержится в это коллекции;

6. Assert.IsNull – входной параметр – null;

7. Assert.IsEmpty – входной параметр – пустая коллекция.

8. Assert.Fail – прерывает выполнение теста и среде NUnit, что тест не пройден. Эта функция может быть использована, когда нужно организовать более сложные типы проверок.