Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по дисципение Тестирование Программного Обеспечения

Вариант 239

Студент: Саржевский Иван

Группа: P3402

г. Санкт-Петербург

2020 г.

Задание

1. Для указанной функции провести модульное тестирование разложения функции в степенной ряд. Выбрать достаточное тестовое покрытие.
2. Провести модульное тестирование указанного алгоритма. Для этого выбрать характерные точки внутри алгоритма, и для предложенных самостоятельно наборов исходных данных записать последовательность попадания в характерные точки. Сравнить последовательность попадания с эталонной.
3. Сформировать доменную модель для заданного текста. Разработать тестовое покрытие для данной доменной модели

Вариант

1. Функция sin(x)
2. Программный модуль для работы с красно-черным деревом
3. Описание предметной области:

   Например, в тот самый момент, когда Артур произнес "А у меня, кажется, большие проблемы с образом жизни в ткани пространства-времени открылась случайная дыра и перенесла его слова далеко-далеко во времени через почти бескрайние просторы космоса в далекую галактику, где странные воинственные существа балансировали на грани ужасной межзвездной войны.

Используемый JUnit

Для выполнения лабораторной работы использовался `JUnit 5` из `org.junit.jupiter.api.*`

Артефакты JUnit jupiter

- `junit-jupiter-api`: API для написания тестов
- `junit-jupiter-engine`: тестовая среда `JUnit`, необходима в рантайме
- `junit-jupiter-params`: поддержка параметризованных тестов
- `junit-jupiter-migrationsupport`: поддержка миграции с `JUnit 4`

Основные аннотации

- `@Test`: аннотация для тестовых методов
- `@Disabled`: отключение тестового метода
- `@BeforeEach`: метод для выполнения перед каждым тестом
- `@AfterEach`: метод для выполнения после каждого теста
- `@BeforeAll`: метод для выполнения перед всеми тестами

- `@AfterAll`: метод для выполнения после всех тестов
- `@Tag`: пометка тестового случая, для последующей фильтрации
- `@RepeatedTest`: метод является тестовым шаблоном для повторяющихся тестов
- `@Nested`: вложенный нестатический тестовый класс
- `@Timeout`: задание таймаута для теста или всех тестов класса
- `TestTemplate`: метод является шаблоном для будущих тестов
- `@DisplayName`: настраивоемое отображаемое название тестового метода или класса

К другим особенностям библиотки `JUnit 5` можно отнести:

- Требует `Java` минимум восьмой версии
- Создание пользовательских аннотаций
- Выбор версии `JRE` (`@EnabledOnJre`, `@EnabledForJreRange`)
- Поддержка лямбда-выражений
- Параллельное выполнение тестов

Исходный код

`https://github.com/johnny-keker/itmo-4th-year/tree/master/testing/Lab1`

Вывод

В ходе выполнения лабораторной работы я получил навыки модульного тестирования программного обеспечения при помощи библиотеки `JUnit`.

- Было проведено модульное тестирование функции, вычисляющей синус при помощи разложения в степенной ряд. При тестировании использовался подход анализа эквивалентности - функция была разбита на классы эквивалентности, участки на которых она ведет себя схожим образом, были протестированы значения функции на границах этих интервалов и внутри каждого из них. Помимо этого было проведено негативное тестирование - мы убедились, что функция корректно обрабатывает невалидные входные данные (`NaN, +inf, -inf`).
- Затем был протестирован модуль для работы с красно-черным деревом. Тестирование проводилось методом серого ящика - мы тестировали систему относительно предоставленного интерфейса взаимодействия (`insert, deleteNode, keyExists, printTree`), однако после этих манипуляций проверялась вся внутренняя структура дерева с помощью механизма рефлексии. Это позволило судить о том, что этот алгоритм не просто выдает ожидаемые значения при данных входных параметрах, а действительно содержит внутри себя корректную реализацию красно-черного дерева без необходимости нарушать правила области видимости компонентов и без добавления специальных интерфейсов для тестов в исследуемом модуле. Сценарии тестирования были определены

на основании характерных точек алгоритма и сожержали разные последовательности добавления, удаления и проверок на анличие элементов, при составлении сценариев акцент был сделан на том, чтобы входные данные обеспечивали как можно большее покрытие внутренней логики модуля.

- Затем было осуществлено модульное тестирование разработанной доменной модели. Было разработано множество тестовых сценариев, как позитивных, так и негативных, которые были призваны протестировать все логические части исследуемой модели, что было подтверждено анализатором тестового покрытия.