**Name: Emon Monsur** **Student Number: 121367643**

**Class: CS3318 – Advanced Java Programming**

**Assignment: Assignment 3 Test Driven Development**

**Content: Assignment 3 Report**

## Introduction:

**Test Driven Development (TDD)** is a software development methodology that emphasises writing tests before implementation. How it works is by first writing a failing unit test, then writing production code to make the previously failing test pass, refactoring the code if needed, then repeating this cycle with another new test. This cycle ensures that code is well-tested and that meets all requirements. In this project, TDD was used to develop the ColourTable class, which represents a palette of RDB colours. In this report we will discuss the TDD process used, its benefits, its detriments, and any issues identified with the specification.

## TDD Process:

The following cycle was used to create the code during this project:

1. **Writing a Failing Test:** Unit tests were written to specify the expected behaviour of each feature in the ColourTable class. Initially, the test will fail as the feature has not been implemented yet.
2. **Implementing the Code:** Minimal code is written to make this test pass.
3. **Refactoring:** After this test has past, the code is then refactored, if needed, to improve readability, maintainability, improve code quality – all done without changing the codes functionality.

## Implementation Process:

This project was developed iteratively, starting with basic functionality, moving towards more complex functions:

1. **Constructor Tests:**
   - The first test checked if the constructor managed valid sizes – sizes needed to be a power of two that is greater than one.
   - Constructors of invalid sizes were also tested to ensure that the constructor threw exceptions when appropriate.
   - Corresponding code was implemented to validate the palette size.
2. **Adding Colours:**
   - The next tests were created to check that valid RGB values could be added to the palette.
   - Minimum and maximum RGB values were also tested.
   - Code for the add method was implemented afterwards, which included validation for RGB values.

3. **Capacity Handling:**
    - Tests to check that the size limit of the palette was enforced, throwing an exception when exceeded.
    - This functionality was added to the add method.

Each of these test cases focused on a single aspect of ColourTable's behaviour, adhering to TDD principles.

## Benefits of TDD in This Project:

1. **Early Debugging:** When a test fails, its much easier to pinpoint the cause of the issues, as the scope of the changes is limited to the most recent iteration.
2. Improved Code Reliability: TDD helped identify edge cases - e.g. Invalid RGB values  - (-1, 100, 100). Exceeding capacity now triggers an exception.
3. **Easier Refactoring:** Once all tests were passing, the ColourTable class was refactored to improve readability (e.g. making side a final field and improving exception messages).

## Challenges and Drawbacks of TDD:

1. **Initial Time Investment:** Writing tests (such as the testAddValidRGBValues) before implementation required additional upfront time. However, this upfront investment saved time later by preventing bugs which would time to reduce.
2. **Ambiguity in specification:** Certain design decisions were left to the developer, e.g. how to represent the colour internally (a choice of List<list> was made).

## Issues Identified in the Specification:

- Guidance on error messages was absent in the specification. Standardising error outputs leads to uniformity across the project.
- No specification on how to manage duplicate colours was included. Including this could have been a useful addition.

## Conclusion:

Overall, the TDD method provided a structured approach to the implementing the ColourTable class and ensured that the code met all functional requirements and had high reliability The cycle of writing tests, implementing production code, and refactoring proved invaluable during this project. While TDD may require more time and effort initially, TDD facilitated confident development, catching bugs early and enforcing a clean design.

The project demonstrated how TDD, when coupled with clear specifications and testing, leads to maintainable and reliable code.