Suneet Mahajan
121513663

Assignment 3

CS3318
December 6, 2023

This is my work. I have read the UCC Plagiarism Policy and understand that plagiarism is an offence that results in disciplinary procedures.

Implementation Details

- Palette Representation

Upon reading the assignment brief, I initially thought to represent the palette using a 3-dimensional boolean array, of dimensions $256 \times 256 \times 256 = 2^{24} = 16,777,216$ total fields. However, as the limit on the number of colours on the palette was relatively small (2^{10} vs 2^{24} possible colour combinations), I decided to use an ArrayList of Strings (containing the hexadecimal values for the colours added to the palette). Near the end of my implementation, I changed the ArrayList representation to a HashSet one instead due to greater efficiency once a large number of colours have been added, and to provide greater scalability.

- add() Method Signature & Input Validation

I used overloading to implement two different add() methods to provide more flexibility for inserting colours into the palette. The first method takes three integers between 0 and 255 as its parameters, the red, green, and blue values of the colour, while the second one takes a string of six hexadecimal digits, namely the hexadecimal value of the colour being added to the palette.

- Multiple Additions of Same Colour

If the same colour is added multiple times to the ColourTable, on subsequent calls to add() for the same colour, no changes will occur to the state of the palette. This was keeping in mind real-world applications of such a program (and trying to recreate such a scenario in both Paint and Adobe Photoshop), where an exception would not be necessary.

- Additional Functionality

Aside from the above-mentioned methods of adding colours to the palette using either RGB values or hexadecimal strings, I have also added other methods not explicitly mentioned in the assignment brief. Helper methods were written to convert RGB values to hexadecimal, to format hexadecimal strings correctly for parsing (so that the user can pass "fff123", "fFf123" or "#FFF123" as a parameter without causing errors), and to validate hexadecimal values.

Furthermore, I wrote two methods to allow the user to also remove colours from the palette, once again, accepting both RGB values and hexadecimal strings. This was a contributor to

changing the representation of the palette from an ArrayList to a HashSet.

Finally, I added a countColours() method, which allows the user to query the number of colours present in the palette at any given time, as well as adding more flexibility in creating tests.

The TDD Process & Thoughts

I found the Test-Driven Development process a unique and interesting way to approach writing code.

Writing tests before the actual code helped me identify potential issues early in the development process, saving time that could have been spent debugging a larger and more troublesome codebase later. By running tests frequently, I could catch bugs and errors as soon as they were introduced.

In designing tests for future code, TDD forced me to think about the code's specifications and expected behaviour prior to the implementation stage. This gave me a clearer understanding of the ColourTable, and a mental map or template that I planned to follow, making the coding stage much quicker.

One particular aspect of TDD that I enjoyed was how it allowed me to isolate each part of the code and encouraged incremental development, rather than my usual, more chaotic method of overloading myself and having to focus on the entire codebase as a whole. The assurance that the previously written code was correct gave me the freedom to fixate fully on the active code, increasing my productivity as a result and giving me the confidence to refactor code for better performance without the fear of introducing defects, like a safety net of sorts.

On the other hand, I found having to write tests repeatedly to be quite time-consuming and detracting from the end goal. At times, making tests overshadowed the ColourTable implementation entirely, and it was easy to lose sight of the main target of this assignment.

I also felt that TDD, while novel, wasn't all that beneficial to me in this instance, as my code was generally quite concise. The two bugs I did catch through this process were quite minor; I made a spelling mistake while typing in one of the exception messages, and the code snippet I found and modified from StackExchange to convert three separate RGB values to a hexadecimal String was wrong and had to be removed.

I think TDD is more valuable in large-scale scenarios in which multiple developers are collaborating. The test suite serves as a clear specification of the expected behaviour, facilitating team members to work on different parts of the codebase without interfering with each other. More developers also increase the quality of the tests against which the code is run, as a single programmer can get tunnel-visioned and one-dimensional whilst making tests.

Issues

The only issue I encountered during my undertaking of this assignment was at the beginning; IntelliJ creates a different folder structure when Maven is enabled depending on the JDK version provided. This led to confusion as I tried to change devices from my laptop to my home PC, and this was resolved after I downloaded the same JDK version and used that to create a project instead.

Reflection & Final Thoughts

In conclusion, adopting Test-Driven Development has been a valuable learning opportunity. Overcoming the initial learning curve, I've gained insights into effective testing strategies and the benefits of incremental development. Experiencing the positive impact on code reliability and design has not only emphasised the importance of TDD but has also contributed to my personal growth as a developer.