Java Assignment 3

Implementation Details

Palette Representation

Initially, I considered representing the palette using a 3-dimensional boolean array of size $256 \times 256 \times 256 = 16,777,216$ possible color combinations. However, given the assignment's palette size constraint (a maximum of 1024 colors), this approach seemed excessive. Instead, I opted to use a HashSet to store hexadecimal representations of the colors. This decision offers efficiency for lookups and prevents duplicate entries while remaining scalable for the palette's size limitations.

Method Implementation and Input Validation

To provide flexibility when interacting with the palette, I implemented overloading in the add and remove methods:

- The add method supports adding colors via both RGB integer values and hexadecimal strings.
- Similarly, the remove method allows colors to be removed using either representation.

To ensure the validity of inputs:

- RGB values are constrained between 0 and 255.
- Hexadecimal strings are validated for length and formatting. Helper functions were included to standardize input (e.g., converting lowercase or shorthand formats into proper six-character uppercase hexadecimal strings).

Preventing Duplicate Additions

Adding the same color multiple times has no effect on the palette, aligning with real-world use cases where duplicate entries are unnecessary. This behavior mirrors how tools like Photoshop and Paint handle color management.

Additional Functionality

The implementation includes the following:

1. **Helper Methods**:
   To convert RGB values to hexadecimal.
   To standardize and validate hexadecimal strings.

2. **Count Functionality**:
   A countColors() method to return the current number of colors in the palette. This feature is useful for testing and monitoring the palette's state.

The combination of these methods and validations ensures robust and intuitive management of the ColourTable.

Test-Driven Development Process

The Test-Driven Development (TDD) approach was a novel experience for me in this assignment. Writing tests before implementing code encouraged me to think through the required specifications and edge cases early in the development process.

Benefits of TDD

1. **Incremental Development**:
   By focusing on individual components, I could build the ColourTable incrementally and confidently, knowing that previously implemented features were validated by tests.

2. **Early Bug Detection**:
   TDD helped me catch minor issues, such as an incorrect RGB-to-hex conversion and a typo in an exception message, during the implementation phase.

3. **Enhanced Code Reliability**:
   Having a comprehensive test suite allowed me to refactor code confidently without fear of introducing new defects.

Challenges with TDD

1. **Time-Consuming**:
   Writing and running tests repeatedly became tedious and occasionally detracted from the core implementation.

2. **Suitability for Small Projects**:
   For a concise class like ColourTable, the benefits of TDD felt marginal compared to its cost. However, I recognize its value in larger, collaborative projects where the test suite serves as a shared specification.

Issues Encountered

The only notable issue arose at the project setup stage. I encountered differences in IntelliJ's folder structure depending on the JDK version used to initialize the project. This discrepancy caused confusion when switching between devices. Resolving the issue required standardizing the JDK version across both machines.

Reflection and Final Thoughts

This assignment has highlighted the strengths and limitations of TDD. While it proved time-consuming for a small-scale project like this, it also demonstrated its potential to improve code reliability and design. Through this experience, I have gained valuable insights into effective testing strategies and incremental development, both of which I expect will be instrumental in larger, team-based projects.