Maxim Chopivskyy, 118364841

Report on the Colour project

Firstly, I decided to create a test that tests whether a colour with the default RGB colour model will return the correct component values when prompted to return the component values. After I created this test, I created the constructor that stores the component values in an array and the model in a String. I decided to store the values in an array instead of individual integers, as I theorised, this would be better coding practise. For example, if we decided later to allow the Colour model to take 4 values, it would be easier to modify our code to suit this. I found using the TDD process to be useful for this step. It was simple and made me feel confident with my implementation of the code. However, one small detriment with this process is that it takes a bit of time to set up, for when you want to test a very simple functionality that sometimes doesn't feel necessary to test.

Next, I tested and implemented the functionality of creating a colour with the option of specifying a model. It was similar as with the default model, so it was easy to implement. After implementing this, I refactored the code so that the default colour constructor calls the "specified model" constructor with the model "RGB". This shortened and made the code more efficient.

Next, I wanted to add more functionality inside the constructor that would make sure an exception is thrown when an illegal parameter is supplied when creating a Colour. I made a test for if exception is thrown when model is under or above length 3. Implemented this as a private method called in the constructor. I repeated this same process for checking the component values satisfy the allowable range, and checking if all the characters in the model are unique(this was so I could store a dictionary of the model characters and values). These are all implemented as private methods called in the constructor. I wasn't able to test these methods directly as they are private methods, so instead, I test the effect they have on the constructor. In my test for whether the constructor throws an error when values out of range are passed, I decided to use a parameterised test for it. This was just to make sure that the same result is observed if only the first value is out of range or the third value is out of range.

Next, I had to test and implement functionality of comparing two colours. I decided to make a non-static equals method to compare two colours. A static method would have worked also. I tested it by using assertEquals. This checks if two objects equal each other. I compared two colours that had the same parameters. I implemented the method and the test passed. I made tests also to check that colours do NOT equal each other when their models are different and when their values are different, respectively. These tests passed, and I was confident my function worked correctly. I refactored the equals method.

The functionality of adding two colours was implemented as a non-static method (a static method would work also). I tested the correctness by creating two colours, adding them and checking if the result was equal to the expected colour object result, using assertEquals. I implemented this and created another test where a value from the sum went over 255. The test failed, so I had to add some modification the add method I already created, in order to pass the test. After, it was correctly implemented, I refactored my code.

In conclusion, I found the TDD process to be a very useful tool in making the development process much more tidy, secure and organised. You feel safer from creating tricky bugs. Creating the test first before the actual implementation, makes the goal much more clear and helps you not to add unnecessary features.

However, there are also small disadvantages, I feel, from the TDD process. Sometimes it can be impossible to have one test fail without making another test fail as a result of the same mistake in the implementation. For example, if my implementation of the compare functionality in the Colour project were to fail, it would also cause the addition test to fail, as that test needs to compare the sum with the expected result. This could happen if you were to modify your code later on and make a mistake. You would be mislead by the failure of the addition test, even if there's problem in that implementation.