Username: zaki1

Code Design Document

Code

I decide to split the code up in to several packages so that it would be based on an MVC architecture that would make it much easier to distinguish the GUI design code, which was mainly written in fxml, from the rest of the base functionality of the application and the base data structure classes used to create manage the computation and the business logic.

This approach made it much more readable and allows others, should they need to be able to further work on development.

Along with this, I included many Javadoc comments which would help a developer to understand my code, which would also greatly improve the ease in maintainability.

In refactoring I decided to have the all the calculation be subclasses of an overarching Calculations class. This decision was made in order to take full advantage of the OOP style programming that java provides to allow me to have the general “submit” function in the FxController class. This meant that I could have less large repetitive sections of code for the individual submit function needed for each calculation. This made the code much more readable which in turn also helps sustain the maintainability of the code in the long term. Should it need to one day be extended, the code allows that to be quite easily done. One would just have to add a new subclass to the code which would extend calculation and then to write quite a short submit method of about 3 line that would make use the generalised submit (rather than having to write something the site of the generalised submit each and every time).

Also because of the small scale of the project and the very few calculations, the class calculations were nested within a parent class the extends Calculation, in order to keep everything concise, rather than having many classes to refer to as also to allow for the relationship between the ratio in the one class and the future and present values in their value class.

Look and Feel

I tried to make sure that the financial calculator didn’t appear to old in it look and feel. With much of the modern business world, the main distinguishing feature between applications is not simply the functionality of a program, but its ease of use and its visual appeal. Someone should want to use the product. The use of the product itself should be a positive experience for the user, in order for the product to have any real customer satisfaction.

Not only does this help boost the revenue of a company with, as people are more likely to continue to use the product, but there is also a much higher likely hood of thee being referrals or recommendations to use “the best” product in a wide range of product categories, the GUI being and the forefront for users, who are not going to be all that

interested in the source code and it's beauty, beyond the scope of it being fast and functional.

Colour

Colour definitely a main point of interest. I tried to use colour that were not to distracting from the functionality of the GUI, but that were also pleasant. Being for adults, it did try to stray from using primary colours which are often associated with and aimed at children's products. However, it did not want a very bland display either that would be overly corporate, thus I when for lighter more neutral colours for the backdrop.

Buttons

As for the buttons, I ensured that they would be distinct enough from the background so as to ensure that they are noticed. Also, it wanted it to be clear which button they had pressed, by having a deep purple coloration for the button when pressed (almost creating something similar to an inverted colour effect.)

Scenes

I ensured that each scene was labelled appropriately in so as to not have the user confused as to where they are in the application. Also each scene of the application can links back to the main front scene of the application, which means that the user does not get trapped in any one scene.