Group Members:

- Christian Wohlwend
- Umang Agarwal
- Brian Rolf

Github Group Repository:

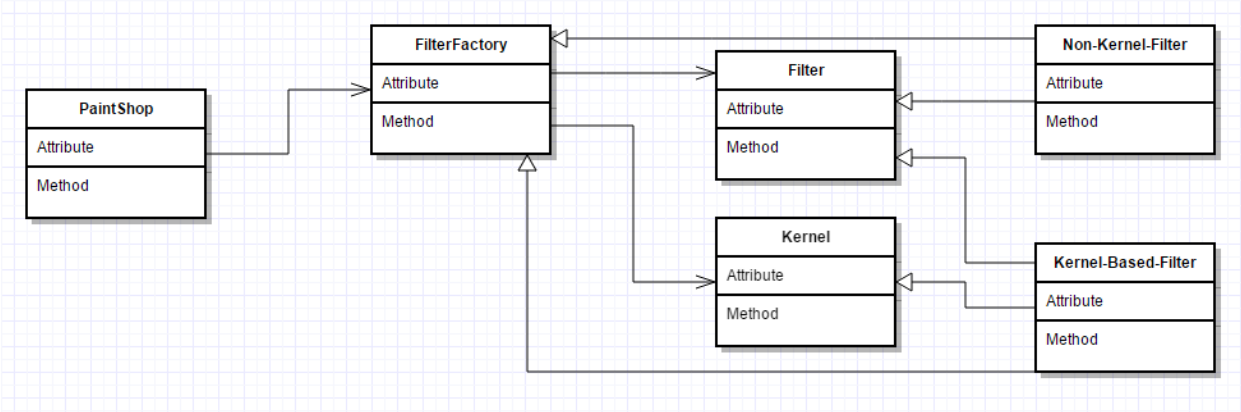
- https://github.umn.edu/CSCI3081S15-010-012-3/PaintShop

Statement of Completion:

- We believe that our implementation of paint shop works to the specification of the design document provided, albeit with a few tweak of preference to one of the brush sizes and the gaps in drawing when quickly moving the cursor.

Self Assessment:

Christian Wohlwend	***
Umang Agarwal	***
Brian Rolf	***



Most Important Design Decision:

We abstracted filter creation to another class to add clarity to our design during this iteration. This design groups functions that are common between the filters. It also cleans up the include section of other files. Paintshop only has to include FilterFactory to build an instance of the filter. FilterFactory includes Filter and each of the filter files. Filter also includes each filter file. These nested dependencies hide some of the complexity. It also reduces the number of files you need to touch when adding a new filter.

Alternative Design:

We considered creating filters using the simple, two level design we used for tools in the last iteration. We were familiar with that method and knew that it worked. One benefit of not planning out a complicated multi-level design would save us some time in the short-term. A major downside would be the added clutter in the directories and include sections. The biggest downside is how static our program would become, mandating edits in lots of places just to add one new filter.

Why We Chose Our Design:

We eventually decided to go with the three level filter factory design for a number of reasons: organization, consistency, and dynamicness. The foremost being organization, the structure itself documents what is happening and that makes it easier for someone not familiar with the project to quickly understand the code. We started this iteration from the code provided by the TA's which used a tool factory. Using factories to handle all the tools and filters cuts down on the clutter in the source directory. It also adds to the overall consistency of the entire project. The time that it took us to design and implement this slightly more complicated structure was shorter than the time we would have ended up spending on fixing bugs and rewriting repeated code in the simpler design.

Second Most Important Design Decision:

Along with filters come kernels. We identified the commonalities between the desired kernels and combined those into a set of functions for kernel creation. For example: filling in the center of a kernel, filling in the border, or filling in a cross.

Why This Was a Difficult Decision:

The Kernels had to be dynamic because the strength of the filter could be changed with a slider, requiring a different kernel. Changes such as size, direction of value input, and other patterns on value setting were required, which mandated the dynamic filters.

The idea of hardcoding kernels was suggested in hopes of saving time. We quickly realized that hardcoding variable strength kernels would take much more time than just writing kernel building functions. We considered hijacking the mask building functions to create kernels because how similar they are, they both use a small array to affect the canvas.

Later we dismissed the idea because of the differences between the two. Masks affect the area surrounding any pixel the mouse clicks or drags over. Masks have to be fast to keep up with multiple mouse drags in quick succession. Mask values can be precomputed because they don't change. Kernels affect every pixel on the canvas based on the area surrounding it. Kernels can't be precomputed because they are subject to change by the sliders. Masks and kernels are different enough to necessitate handling their creation differently. It was so fundamentally different that we used a completely different data structure to implement them. Instead of 2d arrays we used 2d vectors to allow dynamic sizes.

Final Thoughts:

It is important to have organized design strategies like the ones we choose here because the negative effects of not doing so would be multiplied in larger programs. While this program has many functionalities, it is basic compared to some photoshop programs on the market.

Special Filter: Michael Bay Filter

Before Image:

After Image:

Intentions and approach:

When working on the the filters that had kernels similar to edge detect, we experimented with other values of the surrounding values and center values. This caused us to stumble upon a filter that over blew the image white balance and sharpened the image to cringe worthy artifact levels. This effect just happens to be the proper values for a michael bay post effect process. The implementation of this filter is exactly the same as a filter like edge detect, however this filter results in a an opening box office earning of $100 million dollars.