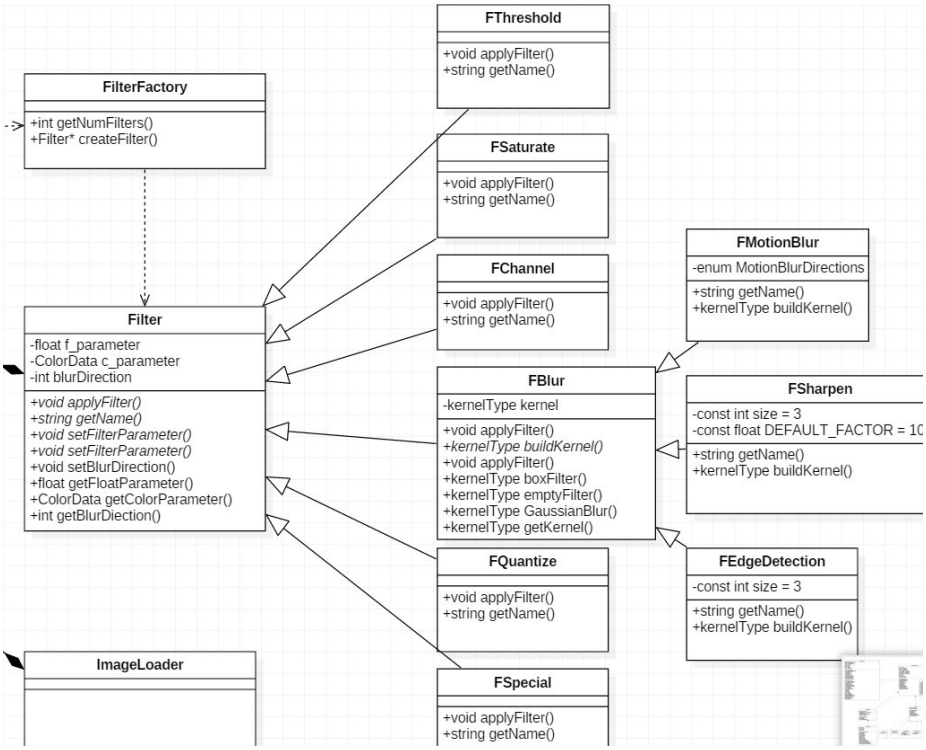
Jacob Reynolds
Juhwan Park
Zhexuan Zachary Yang

git: TwelfthNight

We believe we have fully adhered to the standards given in the handout document and our application should be fully functioning



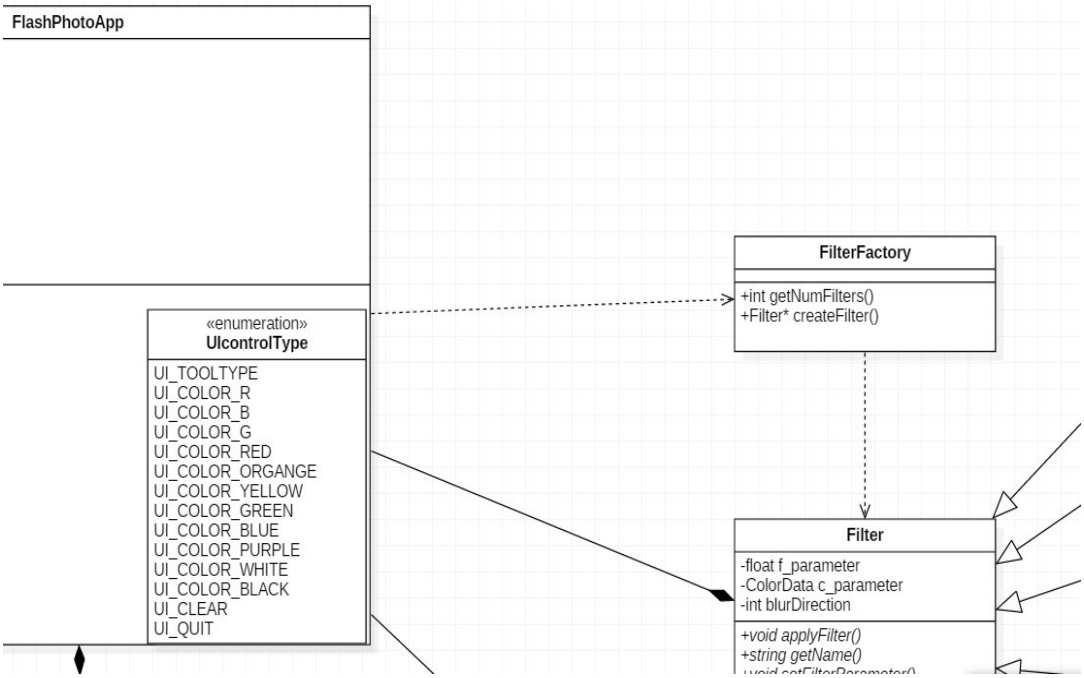
The one main focus of our designs was how we should implement the Filter classes. There was a large amount of discussion between us about whether 1) we should implement all filter effects in a single class, 2) every filter should inherit from one base filter class directly, or 3) using one base filter class along with the convolution based filters sub-inheriting from a convolution filter class. We eventually followed the same approach as we implemented tools in the first iteration. This group of filter classes are similar to the group of tool classes in the first iteration. Inside the filter classes, we used inheritance in order to avoid writing duplicated code for each filter class. We let filter be a class with some pure virtual functions which means all function need to be implemented in its derived class.

In the first approach, we discarded this option because we thought it would be a bit messy. It is not a good abstraction, since we are pulling all of the code in to a single class and we have to implement all of the filter effects in a single class. This would end up being one huge file with terrible documentation, which goes against all of the principles of this class.

In the second approach, we initially thought could work, but we discarded this option since simple filers and convolution-based filters are different. This is because convolution-based filters need to interact with a kernel matrix to apply each effect efficiently. If every convolution-based filter inherited from one base filter class like other simple filters, then we will have to write a lot of duplicated code in these classes, and base filter class would be messy.

The main advantage of our design compared to alternative #1 is clarity. Our design has a simple to follow inheritance structure. This allows us to remove duplicate code and have expected member variables of every filter.

The main advantage of our design compared to alternative #2 is classifying filters concisely. With alternative #2 we would have had to rewrite all of the convolution based filter functions in each convolution based filters. It would be redundant since we could eliminate rewriting by inheritance. With our base filter class and blur filter class, we can simply create one function and every subclass will have that function. It allows us to remove duplicate code and have expected, consistent functionality.



Our second most important design decision was the FilterFactory class to directly associate with FlashPhotoApp, and base filter class. We decided to make our design as the factory pattern to isolate further changes in iteration 3 and encapsulate object creation. We used enums rather than a string as the parameter in FilterFactory class to be flexible with further changes.

Whenever FlashPhotoApp requests a filter, it will send requests to the FilterFactory class and FilterFactory will construct and return appropriate filters by associating with the base filter class. This was a much more favorable result for us. However, we went through a couple ideas while designing this.

Our first implementation was questioning the necessity of a FilterFactory class. We thought it might be unnecessary to create one extra class for the fixed amount of filter classes. But after we tried to implement the special filter class at the end of our iteration, we felt that the FilterFactory class would be necessary to isolate further changes and to deal with the possible addition of filters in iteration 3. This lead us to implement FilterFactory class.

The second implementation after implementing FilterFactory class was questioning the necessity of a base filter class. After we implemented FilterFactory class, we thought we could directly connect every filter class to FilterFactory. However, it leads us again to rewrite every function for each filter redundantly.

The main advantage of our design over the first is abstraction, it allows us an access point to the filters that only exposes necessary information and allows easy additions in the future. The main advantage of our design over the second is inheritance and code usage. The FilterFactory needs a Filter class so that it does not need to deal with non-factory logic. Using the FilterFactory we were able to deal with just getting the filters, and not having to worry about the individual filters' implementation details.

We implemented our final design by using the FilterFactory class associated with the base Filter class for construction. We think our main advantage of the final design is having more control and flexibility in adding, deleting, and modifying our filters easily by implementing the FilterFactory class to isolate the further change.