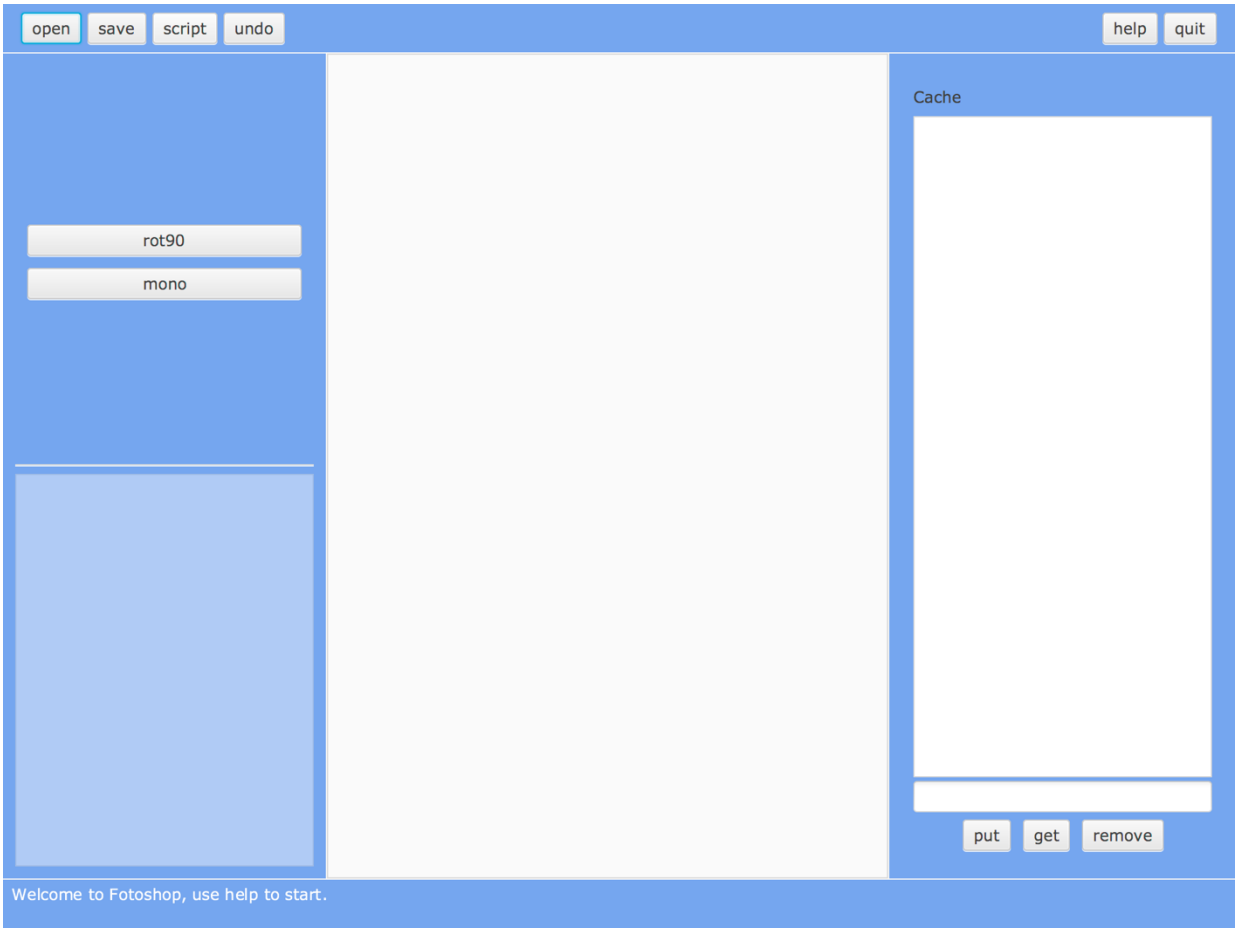
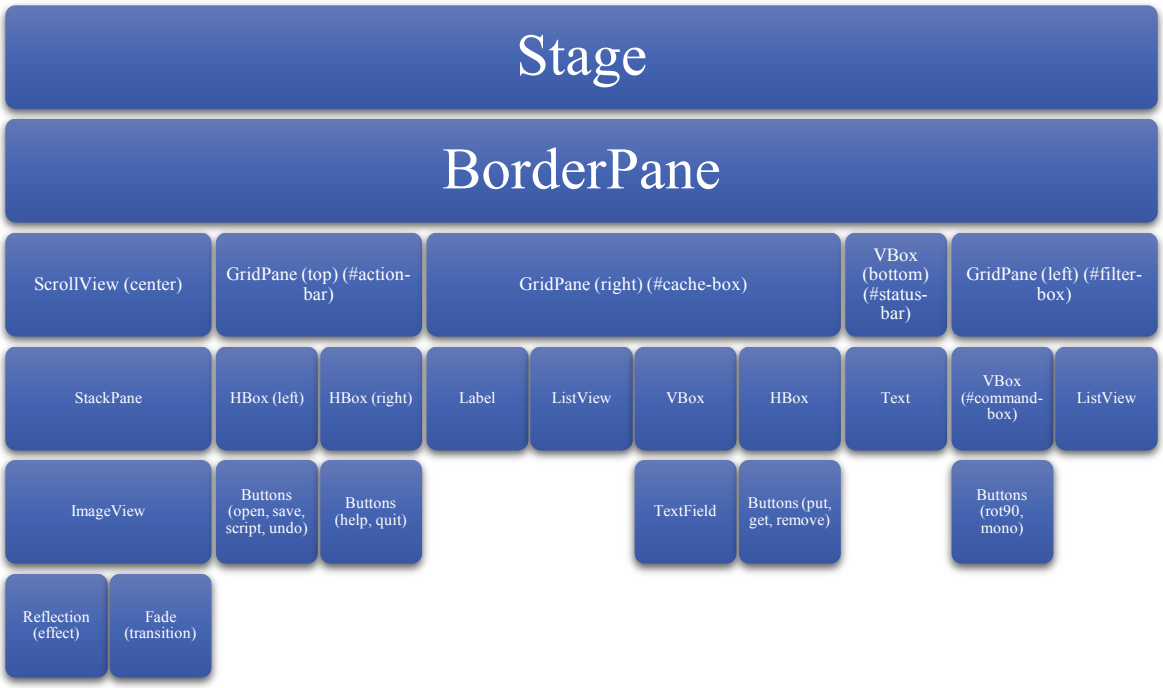
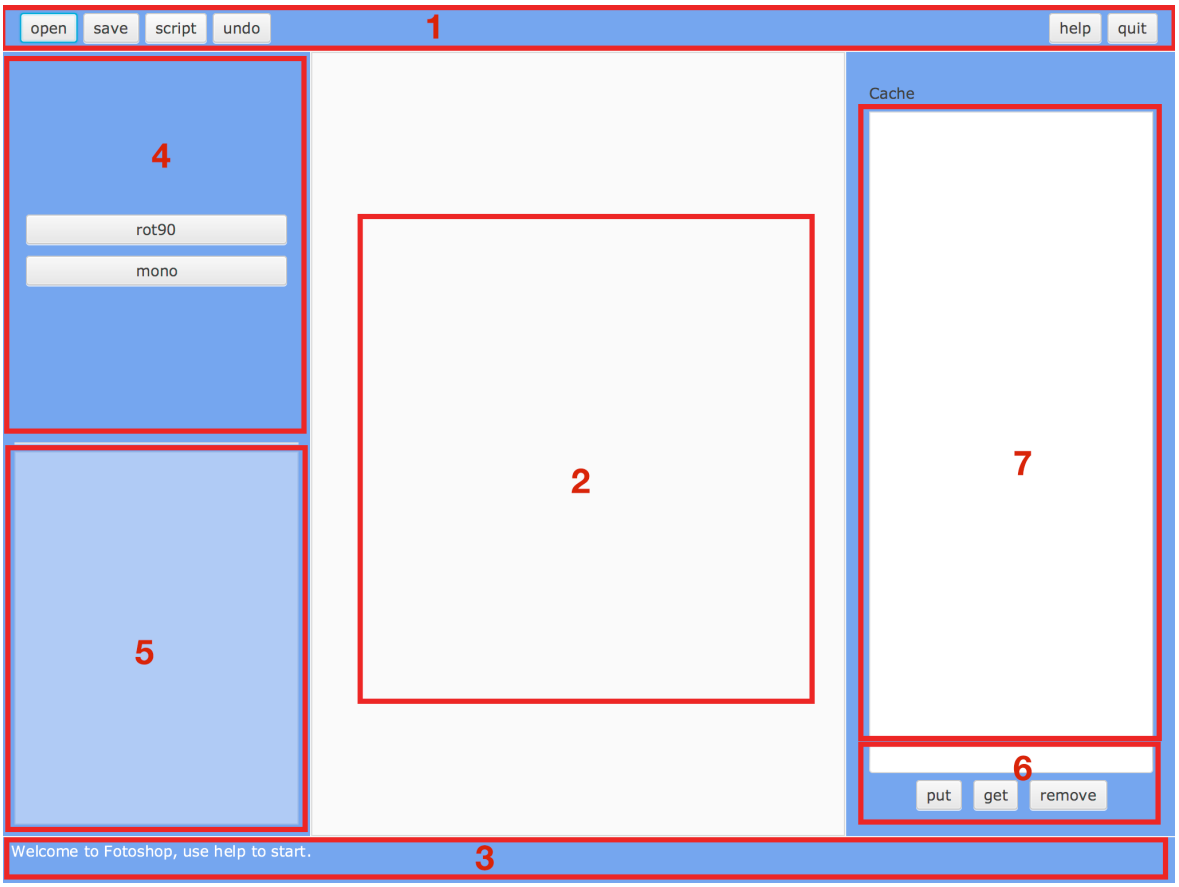
# CO871 – Assignment 2
# Fotoshop GUI

*Guillaume Lefrant*
*gtvl2@kent.ac.uk*
*ID 15906901*



## The GUI


Stage
BorderPane
ScrollView (center)
GridPane (top) (#action-bar)
GridPane (right) (#cache-box)
VBox (bottom) (#status-bar)
GridPane (left) (#filter-box)
StackPane
HBox (left)
HBox (right)
Label
ListView
VBox
HBox
Text
VBox (#command-box)
ListView
ImageView
Buttons (open, save, script, undo)
Buttons (help, quit)
TextField
Buttons (put, get, remove)
Buttons (rot90, mono)
Reflection (effect)
Fade (transition)


open
save
script
undo
1
help
quit
4
rot90
mono
2
Cache
7
5
6
put
get
remove
Welcome to Fotoshop, use help to start.
3

## Layout and Interface

As shown above, Fotoshop GUI uses a BorderPane to structure the layout. I will describe the different parts of the layout I chose to use:

1. It is a menu bar called “action-bar” in the code. The pane used here is a GridPane. I didn’t want to use the MenuBar component because the ergonomic would have changed too much and I have not enough components to display in menus. However, I did another version with it and I could show you in case you want to. This area is used for basic action such as open, save or quit.
2. This is the main part of Fotoshop v2.0. This is where the picture is displayed in an ImageView. This view is inside a StackPane, nested inside a ScrollView. The ScrollView allow me to catch scroll events and to navigate inside the picture (if it is too large for the window).
3. The status bar (Text component) is shown on the bottom of the window. It displays basic information such as welcome message or help message. Errors are managed differently, using Alert class.
4. This area is used for the commands and filters to apply on the picture loaded. It is basically a VBox with Buttons.
5. Here is a disabled ListView that lists every filters applied to the picture. I disabled it because I don’t want the user to be able to click, select or even edit one of the filters. It is linked to the data (List<Filter>) contained in the model Editor.
6. It is the part I chose to implement the cache. You can type the name in a Textfield and then, click on the put Button to put the current picture (including filters) in cache, you can also click on an item in the ListView above and use get or remove (names or explicit).
7. The ListView that lists the different version cached of the picture (can be different pictures too!).

## Event Handling

To handle the different events, I extended my interface ICommand (from Fotoshop v1) to add a method called “handler” that return a new EventHandler<ActionEvent> associated with the command. To generate my buttons, I created a method “createButton” which creates a new button with the correct text and attach the handler, based on the EnumCommand in parameter, via Reflection.

## Bonus point

I didn’t give up the console view with the GUI project. You can still launch Fotoshop in your terminal by providing an environment variable when launching the program:

*$> env CONSOLE=1 java -jar fotoshop.jar*

# Streams

I rewrote almost all the program so I’m not sure of where I “replaced” loops to streams but I used them when possible.

Here are two examples:

1. As a joining collector in multiple places. For example in uk.ac.gtvl2.commands.LookCommand to display the filters currently applied or in uk.ac.gtvl2.models.EnumCommand for the method getList (that return a String of all the available commands, delimited by commas).
2. As a filter like in uk.ac.gtvl2.models.EnumCommand.getCmd() to get the right Command according to the String in input.

I could have write these blocks of code with for loop but Streams are definitely nicer to use. Code is easier to read and lazy evaluation gives good performances to the methods. I also use them to cast Stack

to List easily. I could have use them for the Parser but this class is working great and it wasn't the objective to improve the console side.

## Testing

For testing, I used JUnit 4 to test the class ColorImage.

I wrote tests for the two methods originally present in the class, setPixel() and getPixel(). These methods aren't doing a lot of things so I mainly tested boundaries.

Exceptions for accessing out of array bounds are generated and not caught (and should be) by the class. There is also an error when the Color is null, it should be checked by the class.

## Conclusion

I really liked to work on this project. It was interesting to think of the GUI, how to setup the layout for a good user experience, how to link buttons with action and how to keep the ConsoleView working!

Was a nice challenge.