Task I

Introducing the brand new GUI of Fotoshop.

Fotoshop icon

In the first task part I'll describe the GUI, the following points will be approached:

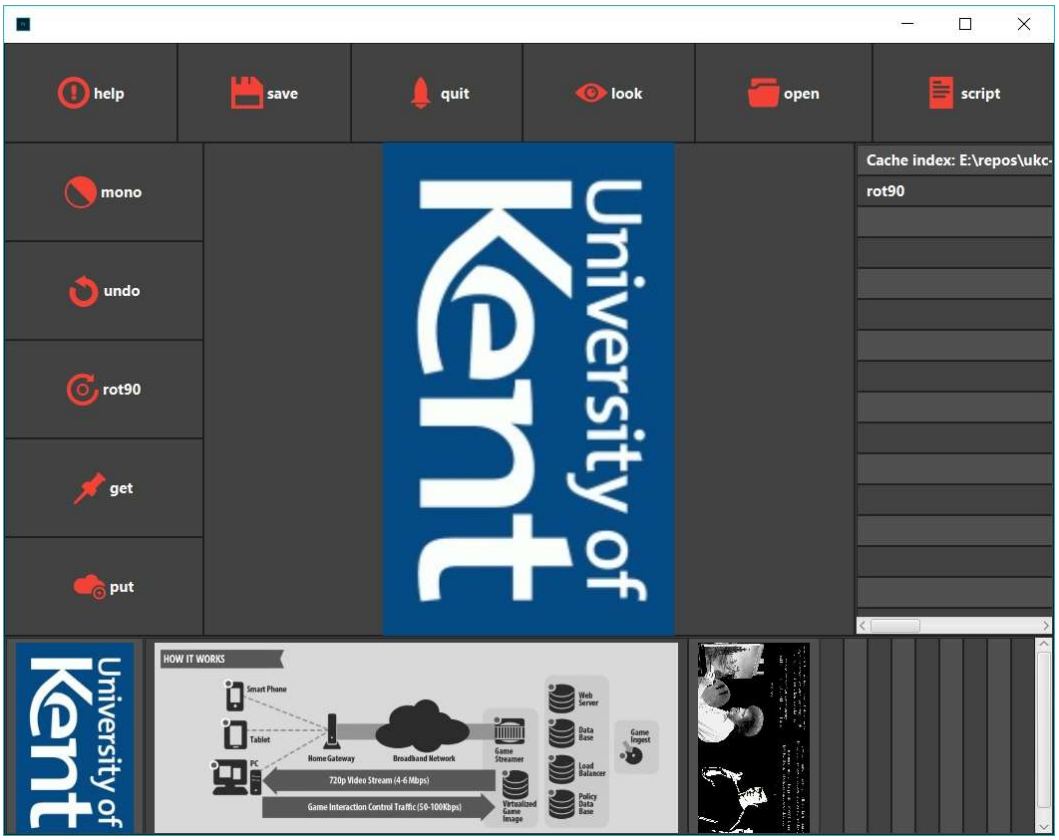
- GUI structure (panes, javafx components used)
- Features implemented (and how to interact with)
- Event handling which produced a lot of changes in the design of the code in order to have a clear design.

GUI structure

The main pane is a BorderPane which provides 5 layers (left, right, top, bottom and center). I found that this disposition of layers is perfectly suitable for the features of our GUI. I'll present the content of each layer in this BorderPane:

- TOP: That layer should implement the basic commands of the software which are not related to image editing (such as help, save, open, quit, look, script). This has been implemented as a bunch of Button objects which are stored in an HBox to have a horizontal layer.
- LEFT: That layer should implement the image editing commands of Fotoshop (such as mono, undo, rot90, get, put). They are implemented into Button objects which are stored in a VBox to have a vertical layer.
- CENTER: This is the main content it should display the current image in the cache. It has been implemented using an ImageView object which is stored in a VBox, here I used a VBox in order to scale the ImageView dimensions depending on the image size and window size.
- RIGHT: That layer should display the name of the current image and the list of filters applied (similar to look command), these are the image details. It has been implemented using a ListView object (this component is perfectly suitable to display a list of items) which is stored in a VBox in order to have a vertical layer.
- BOTTOM: That layer should display the state of the cache (preview of each image available in the cache). I implemented it with a ListView object (again I have to store a list of ImageView object so this component is a must use) which is stored in an HBox because we want to have a horizontal layer.

I also used one CSS file to make up all the styles and I integrated icons for each command and for the program.



Fotoshop preview

Features implemented

We have the full functionalities of Fotoshop in command line as well as a visual representation of the cache, the current image and its details.

When clicked the help command shows a popup which indicates available commands. Save button pops up a file explorer which helps to choose a folder and enter a filename before saving the current image. Quit button has the same purpose as the red cross exit button which leaves the program. Look button has no effect because currently image details are shown in a ListView object but it is kept to be implemented in another way if needed. Open button pops up a file explorer which helps to choose a folder and enter a filename before opening an image and setting it as current image. Script button pops up a file explorer which helps to choose a folder and enter a filename before executing the script commands (transitions effects that are produced by some commands such as mono or rot90 are displayed when using the script command).

Now about the image editing commands, mono filter is implemented and it produces a fade effect transition in 2 steps, first the image fade and then start to be displayed with mono filter applied on. Undo command is implemented as in command line interface just click on it to undo. Rot90 filter is implemented and invokes a rotation transition on the current image. Get command is implemented but an improvement has been done, the user input is limited because I use a popup which displays a combo box so the user can only select existing indexes in the cache. Put command displays a popup which asks for an index name to enter in order to save the current image in the cache.

There is no additional user interaction with current image pane, current image details pane, and cache pane.

Event handling

In order to have a better event handling structure I decided to refactor some code from previous assessment. I abstracted the display interface into an interface UI which implements all standard output needs from previous assessment. Then I implemented the CLI class (used for command line interface) and the GUI class for this assessment. This way each command (e.g. TaskHelp class) calls the associated display function by accessing the UI object from Editor (e.g. "Editor.getInstance().getUI().help();" from TaskHelp.java). This is way more generic and enables to switch between interfaces easily.

I managed the event handling this way: in the code I have a HashMap<String, EventHandler<Action>> which stores every button click function indexed on the command name. Then for each command I implement its feature into a function which retrieves user input if needed (e.g. get, put, open, save, script commands) then calls the appropriate command function which itself calls a UI function which implements output to user (in case of the GUI, it is about handling transitions effects for rot90 and mono commands and updating data in multiple containers for get and put commands as example).

Task II

Use of streams improves readability, speed in execution and produces a more verbose code.

Examples:

From file Editor.java and function "mono":

```
for (int y=0; y<height; y++) {
   for (int x=0; x<width; x++) {
      Color pix = tmpImage.getPixel(x, y);
      int lum = (int) Math.round(0.299*pix.getRed()
                        + 0.587*pix.getGreen()
                        + 0.114*pix.getBlue());
      tmpImage.setPixel(x, y, new Color(lum, lum, lum));
   }
}
```

Which has been changed with streams to:

```
IntStream.range(0, tmpImage.getHeight())
            .forEach(y -> IntStream.range(0, tmpImage.getWidth())
                                    .forEach(x -> this.processPixel(tmpImage, x, y)));
```

The function processPixel does the inner instructions of the second "for" in initial code. This new code is more comprehensive and easier to maintain.

From file Editor.java and function “look”:

```
if (filter1 != null) {
    System.out.print(filter1 + " ");
}
if (filter2 != null) {
    System.out.print(filter2 + " ");
}
if (filter3 != null) {
    System.out.print(filter3 + " ");
}
if (filter4 != null) {
    System.out.print(filter4 + " ");
}
```

Which has been changed with streams to:

```
IntStream.range(0, ImageCache.getInstance().getCurrent().sizeFilter())
        .forEach(i -> System.out.println(ImageCache.getInstance().getCurrent().getFilter(i) + " "));
```

Code readability has been significantly improved.

Task III

Tests on the Image class have been implemented through all these methods.

Test name	Test overview	Should pass/fail
testLoadImage	Check if an image is loaded and its name well stored	
testGetName	Check if an image name is effectively stored	
testAddFilter	Check if a filter is added by checking the length of the filter list	
testGetFilter	Check if a filter added is equally the same as a filter retrieved	
testSizeFilter	Check if the filter list size increments well after adding filters	
testSetPixelSuccess	Check if setting a pixel with legit parameters works	
testSetPixelFail	Check if setting a pixel with wrong parameters works	
testGetPixelSuccess	Check if getting a pixel with legit parameters works	
testGetPixelFail	Check if getting a pixel with wrong parameters works	

There is a flaw in set and get pixel, x and y boundaries values are not checked so it can crash as described above if the value is under 0 or over image width/height.