Benjamin Nicholls
bn65

CO871
29/11/2013

# Assessment 2: Graphical User Interface using Java Swing

## GUI Design

### Main GUI JFrame

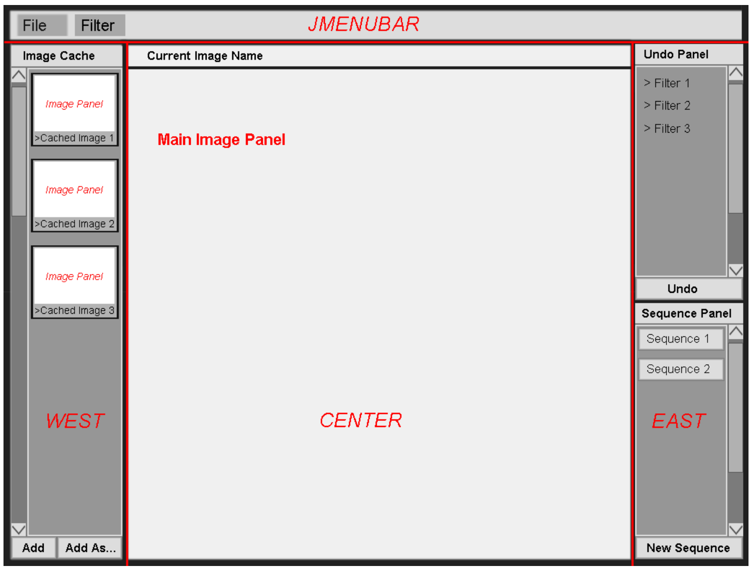

***Figure 1:** GUI JFrame layout design*

The *GUI* class extends *JFrame* and served as the main graphical user interface for the application. It allowed the various required custom components to be added and arranged as required. It also contained all methods for passing any changes, made in the main Editor class, on to the relevant components of the *GUI.*

Given a simple *BorderLayout*, which was sufficient for the basic arrangement of components, it was separated into east, west, and centre sections, along with a *JmenuBar.* (*Figure 1*) This allowed the various components to maintain a suitable size, and retain the remainder of the space for displaying the current image in the centre panel. Also implemented in this class was a *ResizeListener*, that would call the appropriate method of the centre *ImagePanel* in order to resize the image to fit, and re-paint it. This would allow the user to adjust the *GUI* and size of the main image to suit.

### JMenuBar

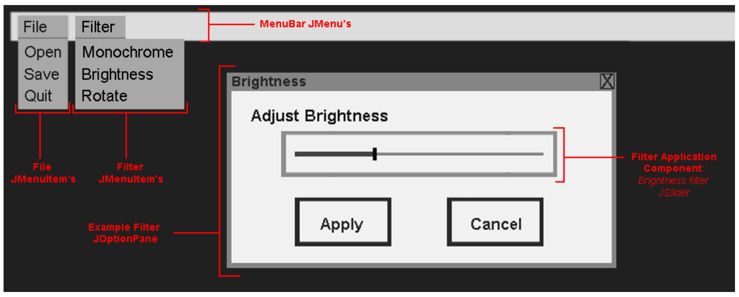

***Figure 2:** JMenuBar and Brightness Filter Pop-Up Dialog design*

The *GUI*'s JMenuBar handled two of the required sets of functionality of the specification by adding two dropdown *Jmenus* (*Figure 2*). The first of which, F*ileMenu,* consisted of *JMenuItem*'s controlling the Save and Load functionality required by the specification. This was simply implemented with the FileWindow class extending a standard *JFileChooser*, and making use of *showOpenDialog()* and *showSaveDialog()* methods to retrieve a File to send to the Editor to be opened or saved to.

It was decided that the filters that may be applied to the main image should be displayed via a JMenu, *FilterMenu*, as in the current application of fotoshop, because as the number of filters available could be expanded a *JMenu* could be easily populated depending on a list of filters passed in. In order to handle any unique filter controlling components, such as a slider for the *BrightnessFilter*, or a button for the *RotateFilter*, these items cause a subclass of *FilterPanel* to be displayed via a *JoptionPane*. *FilterPanel* simply constructing the *JPanel*s basic components, while each different subclass created specific components to be displayed, such as the *JSlider* component, for adjusting the brightness value, of the *FilterPanel_Brightness* class shown above (*Figure 2*).

**Centre Section**

A relatively simple section, this was comprised of a JPanel added to the *GUI* with *BorderLayout.CENTER*. This contained a *JLabel* displaying the name of the current image, and an *ImagePanel*.

***ImagePanel:***
Extending *JPanel* this panel simply used the Graphics package to display the image. Each time its *showImage()* method was called it also calculated a new size and position for the image to displayed at to ensure it was fully displayed and centred in the *ImagePanel* when adjusting the main *GUI JFrame* window.

**East Section**

This section was created to handle the remainder of the specifications requirements in two separate Jpanel's, added to the GUI, *BorderLayout.EAST.* These consisted of an *UndoPanel*, for listing the filters applied to the image and allowed them to be undone, and a *FilterSequencePanel*, which would allow the user to create and enact sequences of filters.

***UndoPanel:***
This *JPanel* (*Figure 3*) consisted of a *JScrollPane* displaying a list of buttons, each depicting a filter previously applied. A *JScrollPane* was used to handle an extensive number of filters applied. Each button firing an "undo" command to the Editor with a particular index.

Another *JButton* was added to the base of the panel that would simply undo the previous filter applied, stepping back through the list.

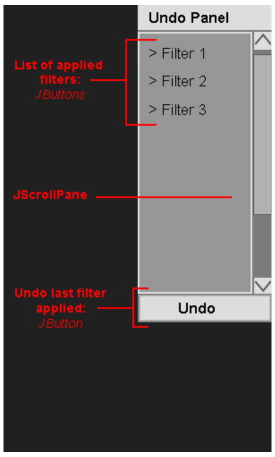

***Figure 3:*** *Undo Panel design*

***FilterSequencePanel:***
Consisting of a similar design as the *UndoPanel* this panel, as seen in *Figure 4* below, displayed an indeterminate number of *SequenceButton*'s in a *JScrollPane*. Each button containing a list of commands to be fired to the Editor upon selection.

To allow the user to create a new sequence of filters and add an associated button to the panel, a "New Sequence" button was added to the bottom of the panel. Selecting this displayed a *Dialog* containing a *SequenceConstructorPanel*, also seen below (*Figure 4*). This panel consisted of a *JTextField*, allowing the user to enter a name for this sequence of filters, a number of *JButton*s corresponding to the available filters, and a *JScrollPanel* of current filters in the sequence. Selecting on of the filter buttons would add a label and a *JFormattedTextField* to this *JScrollPane*. The *textfield* for the purpose of setting the value to be passed to the filter along with the image, for instance the number of rotations to apply with the *RotateFilter*, or the percentage to adjust the brightness by with the *BrightnessFilter.*

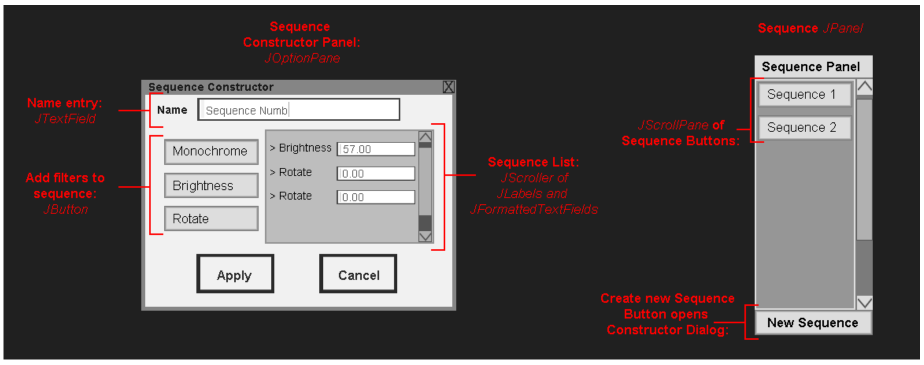
Sequence Constructor Panel: JOptionPane
Sequence JPanel
Sequence Panel
Sequence 1
Sequence 2
Name entry: JTextField
Sequence Constructor
Name
Sequence Numb
Monochrome
Brightness
Rotate
> Brightness 57.00
> Rotate 0.00
> Rotate 0.00
Add filters to sequence: JButton
JScrollPane of Sequence Buttons:
Sequence List: JScroller of JLabels and JFormattedTextFields
Apply
Cancel
Create new Sequence Button opens Constructor Dialog:
New Sequence

***Figure 4:*** *Sequence Panel and Sequence Constructor Panel design*

**West Section: Interface Extension – Image Cache display**

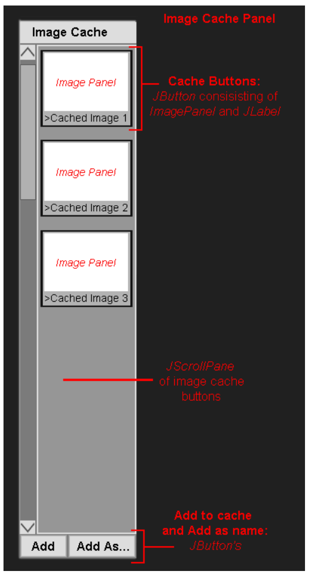
Image Cache
Image Cache Panel
Image Panel
>Cached Image 1
Cache Buttons: JButton consisting of ImagePanel and JLabel
Image Panel
>Cached Image 2
Image Panel
>Cached Image 3
JScrollPane of image cache buttons
Add to cache and Add as name: JButton's
Add
Add As...

*Figure 5: Image Cache Panel design*

***CachePanel:***
As the first assignment required the addition of a cache of images currently being worked on it seemed like a logical step to implement a graphical display for this cache as the final extension in this assignment. This would ensure that this implementation would not be rendered redundant, and if implemented as a sequence of thumbnail images (in a similar fashion as the display for 'layers' in a popular piece of image editing software) would allow users to keep track of their work on the various images stored in the cache.

This panel, shown to the left (*Figure 5)*, was designed in a similar manner as the *UndoPanel* and *FilterSequencePanel*. In consisted of a *JScrollPane* containing the variable number of cached image *JButtons*, and two *JButtons* at the base of the panel. These two buttons, “Add” and “Add As...”, allowed the current image to be added to the *ImageCache* either under its current name, or using the “Add As...”, under a specific name acquired via a standard *JOptionPane* displayed with the *showInputDialog()* method.

The image cache buttons, within the *JScrollPane*, each consisted of a *JButton* containing an *ImagePanel* object and a *JLable* with the name given to the stored image.