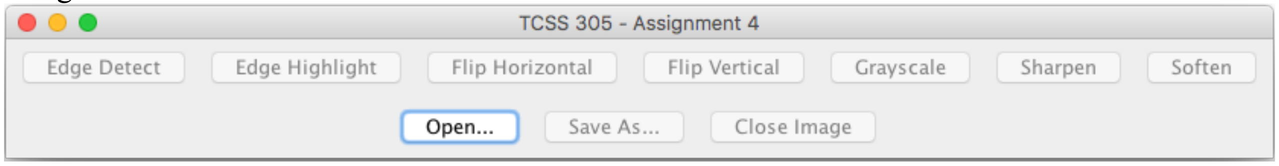
Institute of Technology, University of Washington Tacoma
TCSS 305 Programming Practicum, Autumn 2018
Assignment 4 – SnapShop
Value: 4% of the course grade
Due: Friday, 2 November 2018, 23:59:59

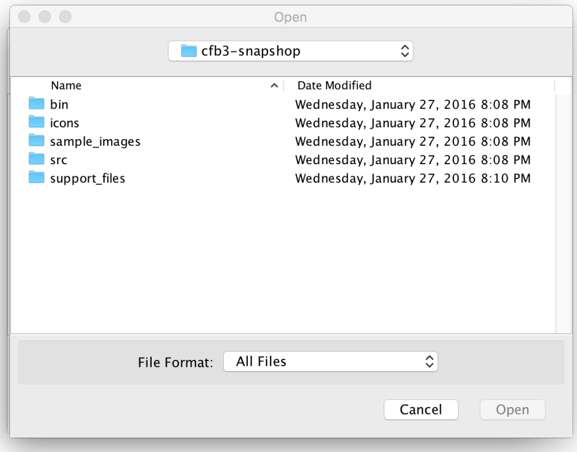
Program Description:

This assignment is designed to test your understanding of graphical user interface components in Java using Swing. You will write the graphical user interface (GUI) for an application that displays and manipulates images; the image processing classes have been provided for you.

When the program initially loads, it should have the following appearance. The window title should be "TCSS 305 – Assignment 4".



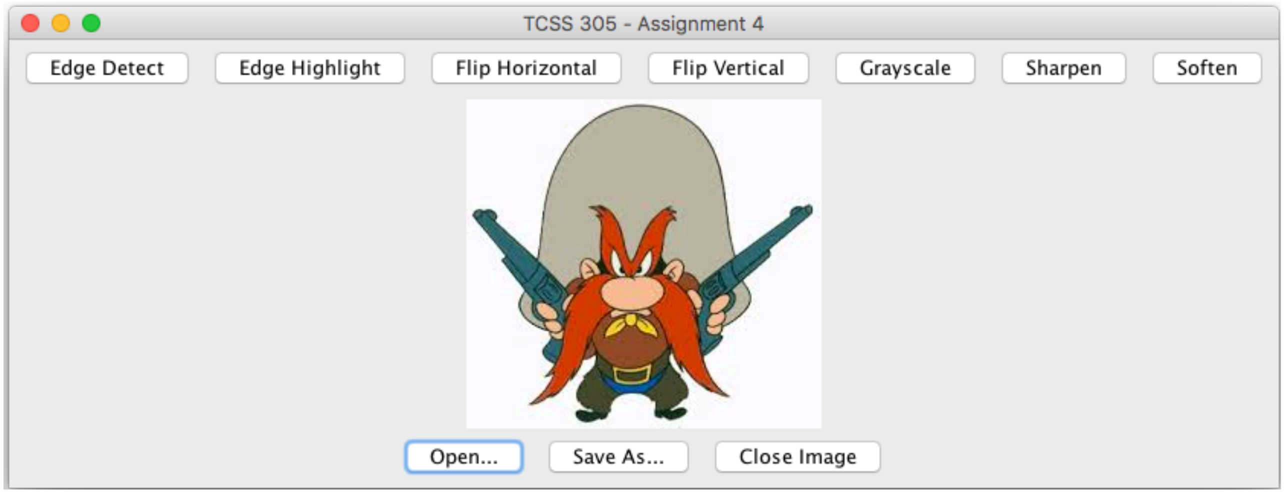
There are buttons centered along the top of the window labeled `"Edge Detect"`, `"Edge Highlight"`, `"Flip Horizontal"`, `"Flip Vertical"`, `"Grayscale"`, `"Sharpen"`, `"Soften"`, `"Open..."`, and `"Save As..."` (each with 3 dots called an ellipses) and a button labeled `"Close Image"`. Note that the top buttons are initially disabled and so are the `"Save As..."` and `"Close Image"` buttons. (The only button initially enabled is the `"Open..."` button.) When the user clicks `"Open..."`, a file selection dialog appears to let the user load an image. Have the file selection dialog open in the current directory as shown below (use relative addressing).

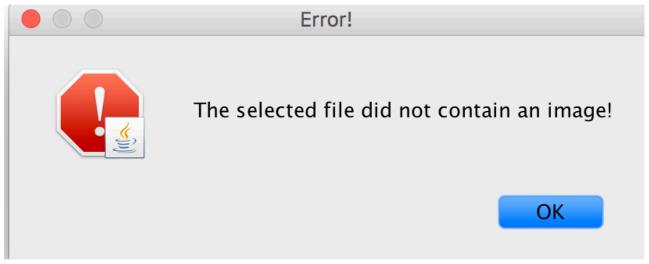


Name	Date Modified
bin	Wednesday, January 27, 2016 8:08 PM
icons	Wednesday, January 27, 2016 8:08 PM
sample_images	Wednesday, January 27, 2016 8:08 PM
src	Wednesday, January 27, 2016 8:08 PM
support_files	Wednesday, January 27, 2016 8:10 PM

When the user selects a file, the system loads image data from that file and displays that image in the center of the window (as shown on the next page). The window resizes to exactly fit the image and buttons; if the image is too large, this may make some of the buttons appear outside the screen area (this is expected behavior, and you do not need to handle it in any special way). If the user later pushes the `"Open..."` or `"Save As..."` button again, it should open the same folder where the file chooser was left previously. If the user cancels the file chooser, the contents of the window are left unchanged.

When an image is loaded, all buttons become enabled (as shown on the next page). Clicking each filter button causes a modification of the image: flipping, sharpening, softening, *etc*. Applying more than one filter to an image has a cumulative effect. These operations do not modify the original image file, they just change the onscreen appearance. If the user chooses a file that does not contain valid image data, the program displays an error message dialog (as shown on the next page). If the user chooses `"Close Image"`, the image should close, all buttons should disable except the `"Open..."` button, and the GUI should resize to its initial size and appearance. If the `"Open..."` button is used while an image is already open, a new image may be selected to replace the current image. Only one image can be open and displayed for editing at a time. When the GUI window is closed, the program should exit.





The selected file did not contain an image!

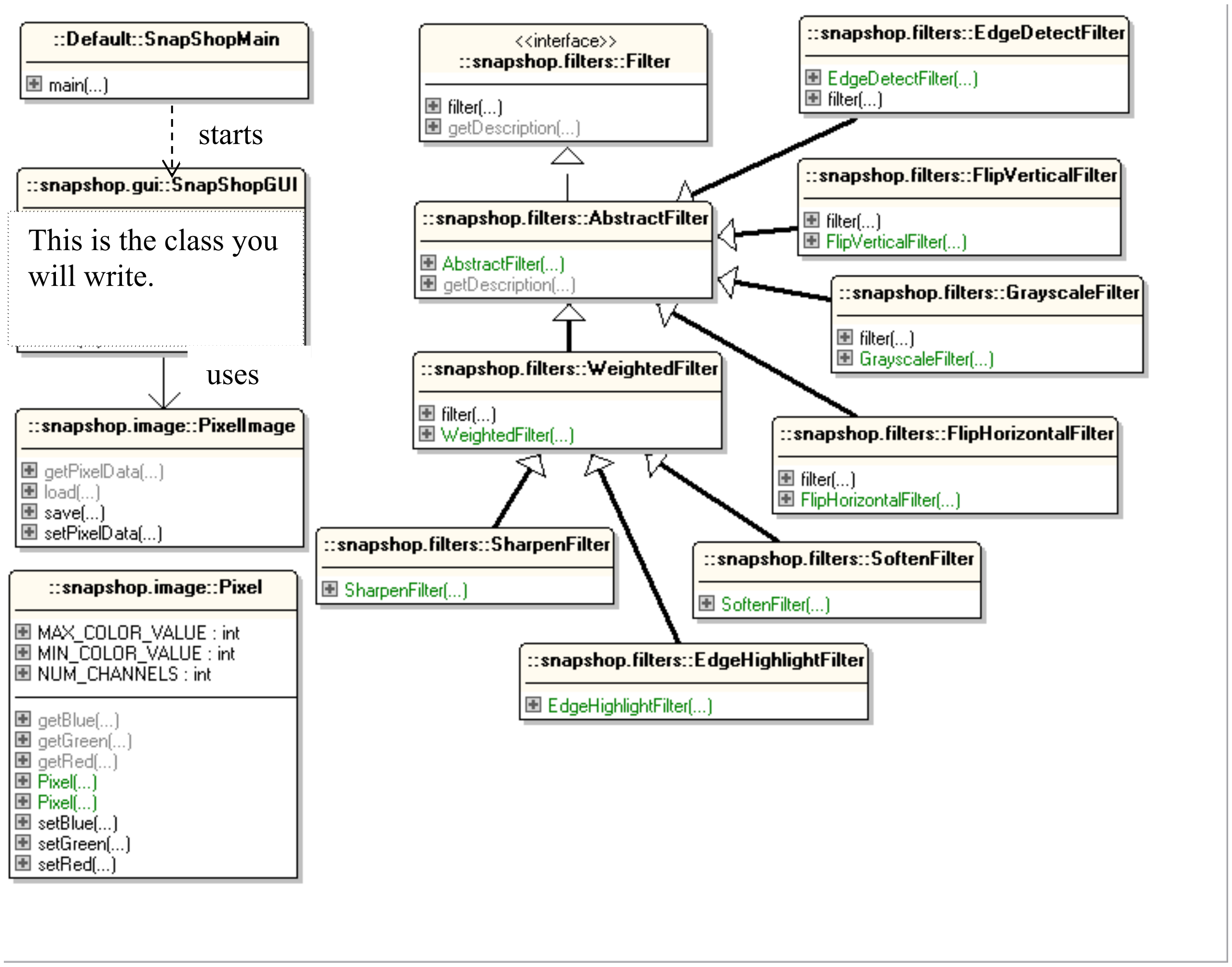
Implementation Guidelines:

You will write a class named `SnapShopGUI` in the `gui` package that contains the implementation of your graphical user interface. You may also write additional helper classes, if you find it necessary. Your class may extend `JFrame`, but this is not required (if you do not extend `JFrame`, you will have to store a `JFrame` as an instance field). Do one or the other, but not both. Your JFrame must be resizable. As the JFrame is resized, any loaded image should remain centered both horizontally and vertically.

The constructor to your class must require no parameters, and must not display the GUI on the screen; instead, your class must contain a method named start() that performs all necessary work to create and show the GUI on the screen. Note that this does not mean that all the work should be performed in the start() method itself; the start() method should call other methods to create various parts of the user interface (buttons, panels), set up event handlers, etc. Note that the main method in the SnapShopMain class calls the GUI constructor and then calls start().

You do not need to write your own code to apply the different transformation filters to the pixels in the image. Do this by using the instructor-provided filter classes. Note that a single filter object can (and should) be used multiple times; you *must not* create more than one of each type of filter object.

The following class diagram gives a summary of the classes in the project and their relationships. The `SnapShopGUI` is constructed by the `SnapShopMain` class. The GUI uses `Filters` to filter `PixelImages` that are composed of `Pixels`.



Provided Files:

The following classes and interfaces are provided for your use. You ***must not*** modify these files when writing your program. The classes have other methods beyond what is listed, but the listed methods are all you need.

Interface `Filter`:

```
public interface Filter {
  // Applies this filter to the given image.
  public void filter(PixelImage theImage);

  // Returns a text description of this filter, such as "Edge Highlight".
  public String getDescription();
}
```

The following classes implement the `Filter` interface. Each has a no-argument constructor.

```
public class EdgeDetectFilter implements Filter
public class EdgeHighlightFilter implements Filter
public class FlipHorizontalFilter implements Filter
public class FlipVerticalFilter implements Filter
public class GrayscaleFilter implements Filter
public class SharpenFilter implements Filter
public class SoftenFilter implements Filter
```

Class `Pixel`:

```
public class Pixel {
  public Pixel()  // constructs a black pixel
  public Pixel(int theRed, int theGreen, int theBlue) // RGB values, 0-255
  public int getRed(), getGreen(), getBlue()
  public void setRed(int theRed), setGreen(int theGreen), setBlue(int theBlue)
}
```

Class `PixelImage`:

```
public class PixelImage {
  // Loads an image from the given file and returns it.
  // Throws an exception if the file cannot be read.
  public static PixelImage load(File theFile) throws IOException

  // Saves this image to the given file.
  // Throws an exception if the file cannot be written.
  public void save(File theFile) throws IOException

  // Methods to get and set the 2D grid of pixels that comprise this image.
  // The first dimension is the rows (y), the second is the columns (x).
  public Pixel[][] getPixelData()
  public void setPixelData(Pixel[][] theData)
}
```

Class `SnapShopMain`:

```
public class SnapShopMain {
  // Runs your program by constructing a SnapShopGUI object.
  // (You must write the SnapShopGUI class.)
  public static void main(String... theArgs)
}
```

Hints:

Display an image on the GUI by setting it to be the icon of an onscreen `JLabel`. A `JLabel`'s icon is set by calling its `setIcon` method. The `setIcon` method accepts an `ImageIcon` object as a parameter. An `ImageIcon` object can be constructed by passing a `PixelImage` as the parameter. Here is an example that creates a new label and sets its icon (note that you should not create a new label every time you change the image, but instead should change the icon of the label you already have).

```
PixelImage image = PixelImage.load(new File("apple.jpg"));
JLabel label = new JLabel();
label.setIcon(new ImageIcon(image));
```

Use a `JFileChooser` for your file selection dialog. A save dialog can be shown by calling `showSaveDialog` on a `JFileChooser` object, and an open dialog can be shown by calling `showOpenDialog`. You can ask for the file the user selected by calling the `getSelectedFile` method on the `JFileChooser`. If image loading fails, catch the `IOException` and use a JOptionPane to display an error message dialog like the one shown on the page 2 of these instructions. Your program should create only one JFileChooser object and reuse it repeatedly.

When an image is selected and loaded, call the pack method on your JFrame to make it resize itself to fit the image and buttons. If the image is too large, pack may make your window larger than the screen; that’s OK (I won’t test the program with images that are too large). After packing the JFrame, set a minimum size on the JFrame equal to the current size of the JFrame (so that it cannot be resized too small to display the image or the buttons).

Try to design and implement an efficient way of creating filter objects and their corresponding buttons. Ideally, it should take you *exactly* one line of code to create each filter and its corresponding button and the button's ActionListener (7 such lines of code for the 7 filter/button/listener combinations). The goal is to make it possible to easily add (or remove) a filter and its corresponding button without making changes to other parts of the project code.

Included with the project are some image files you can use for testing and some icon files which you could add to the buttons at the bottom of the GUI if you choose.

Submission and Grading:

Create your Eclipse project by downloading the `hw4-project.zip` file from the Assignment 4 page on Canvas, importing it into your workspace (as described for `hw0-project.zip` in Assignment 0), and using "Refactor" to change "username" in the project name to your UWNetID.

You must check your Eclipse project into Subversion (following the instructions from Lecture 1 and Assignment 0), including all configuration files that were supplied with it (even if you have not changed them from the ones we distributed). When you have checked in the revision of your code you wish to submit, make a note of its Subversion revision number. To get the revision number, *perform an update* on the top level of your project; the revision number will then be displayed next to the project name.

After checking your project into Subversion, you must submit (on Canvas) an *executive summary*, containing the Subversion revision number of your submission, an "assignment overview" (1 paragraph, up to about 250 words) explaining what you understand to be the purpose and scope of the assignment, and a "technical impression" section (1-2 paragraphs, about 200-500 words) describing your experiences while carrying out the assignment. The assignment overview shows how well you understand the assignment; the technical impression section helps to determine what parts of the assignment need clarification, improvement, *etc.* for the future.

The filename for your executive summary must be "`username-snapshop.txt`", where `username` is your UWNetID. An executive summary template is available on the Canvas. Executive summaries will *only* be accepted in plain text format – other file formats (RTF, Microsoft Word, Acrobat PDF, Apple Pages) are *not* acceptable. You will, in general, not lose any points on your executive summary itself unless you fail to turn it in or it is short of the length requirement and / or trivial.

Part of your program's score will come from its "external correctness." For this assignment, external correctness is graded by running the GUI and examining the result. Exceptions should not occur under normal usage. The program should not produce any console output.

Another part of your program's score will come from its "internal correctness." Internal correctness includes meaningful and systematically assigned identifier names, proper encapsulation, avoidance of redundancy, good choices of data representation, the use of comments on particularly complex code sections, and the inclusion of headers (as described above) on your classes. On this assignment, internal correctness also includes avoidance of redundancy, because it is easy for GUI code to become redundant especially when there are many similar elements (such as the filter buttons). In particular, redundancy in file choosers (using more than one file chooser object) and filters and their buttons (using more than one of each filter, writing the same code multiple times with only minor textual differences to create each filter button) will hurt your internal correctness grade. Try to find the most concise and elegant way possible to create this GUI without repeating nearly identical code many times. Internal correctness also includes whether your source code follows the stylistic guidelines discussed in class. This includes criteria such as the presence of Javadoc comments on *every* method and field (even private ones!), the use of variable names, spacing, indentation, and bracket placement specified in the class coding standard, and the absence of certain common coding that can be detected by the tools. It is therefore to your advantage to be sure the plugin tools like your code before you submit it.

For this assignment, the percentage breakdown is 10% executive summary, 55% external correctness, 35% internal correctness.