COSC 1P02 Assignment 5

"Don't cut yourself on all that edge."

Refer to Sakai for due date/time

The emphasis for this assignment is processing the contents of a collection through indexing, and considering the contents of multiple elements simultaneously. Since our in-class demonstrations focused on sounds, this assignment will focus on pictures.

In preparation for this assignment, create a folder called `Assign_5` for submission of the assignment and two subfolders: `Assign_5_A` and `Assign_5_B` for the two DrJava projects for the assignment.

Part A – Edge Detection

Write a Java program to do simple edge detection on a picture. The resulting image will have black pixels wherever there is an edge in the original picture, and white pixels everywhere else. For example, the picture:

After edge detection looks like:

Your program should receive a picture, display it (waiting for the user), and then create a *new* picture that reflects the detected edges.

Detecting edges relies on identifying portions of the image where there appears to be a *boundary* between two shapes. This boundary is identified by a sudden change in brightness between two adjacent pixels. There are different algorithms for deciding how many (and which) pixels to compare, how to weight them, etc. But you'll be going with the simple: A pixel is an edge if is far brighter — or far dimmer — than the pixel immediately above it.

In other words, simple edge detection only requires inspecting two pixels at a time:

- For any pixel, determine the *intensity* (brightness) of that pixel's colour
- Additionally retrieve the intensity of the pixel immediately above it
- If the *difference* in intensities (positive or negative) has a magnitude of 10 or more, you have found a boundary, so that pixel is an edge pixel (otherwise it isn't)

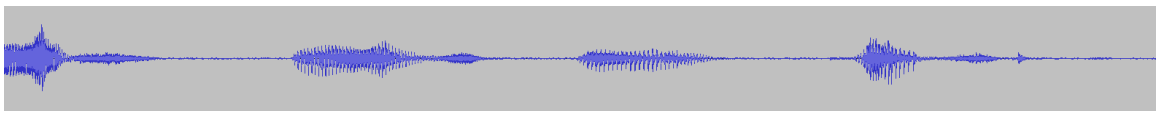
Tips:

- Since you're providing one version of the Picture, to generate a new one, that means a *function*
- Since each pixel's value is defined by both the pixel in the corresponding position in the original, and that of the pixel above it, that doesn't explain how to calculate the pixel values for the very first row (that has *nothing* above it)
  - You're welcome to just not assign anything to that row
- The final step should be to let the user save the new image (just use `.save()`)

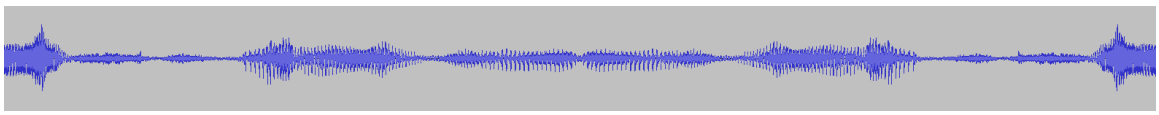
Part B – Forwards and Backwards:

Write a program to load a user-selected sound file, and then create a *new* sound that's the result of superimposing the normal version of the sound, and a reversed copy of the same sound.

e.g. the “This is a test” sound clip:



ends up like:



Tips:

- Again, you'll want a function to return the newly-created sound
- You don't need to amplify, 'normalize', or worry about clipping for sounds. For each index you're simply summing the amplitudes from two source samples, and storing the result into a sample in the target sound
- You may wish to start with just reversing, before moving on to combining data
- Don't forget you have a SoundInspector available to you

Submission:

For submission, you **must** submit a **.zip** file containing the following:

- A folder called `Assign_5_A`, containing your Part A submission
  - Your source file(s) that includes your solution
  - A `.pdf` copy of your final pattern (i.e. output)
  - Your `.drjava` file (or other project file; see below)
- A folder called `Assign_5_B`, containing your Part B submission
  - And the equivalent files as above

On Sakai, submit your .zip file as an attachment, and click to Submit.

Note: Do not submit a `.rar`, `.tar.gz`, `.7z`, etc. *Every* major operating system supports .zip, so it is mandatory if you wish to receive a grade for your assignment.

Standards:

Ostensibly, you'll be graded for following basic coding standards and documentation requirements. However, as we haven't really learned any yet, all you really need is:

1. **A comment at the top of all source (`.java`) files including your *name*, *username*, and *student number***
2. Variable names that are either standard or descriptive
   - Using things like `i` and `j` for loop counters? Standard
   - Using a variable like `count`? We can guess what you meant
3. For any *blocks* of code (e.g. loops and methods) insert a brief comment so the reader knows what's inside (e.g. *what*'s being repeated)
4. Anything you think the marker might not understand? Add a quick comment

Additionally, try to remember to fix the indentation of your source files. Jagged margins can be harder to follow along with.

DrJava (and other platforms)

DrJava is recommended, to make grading as easy as possible. However, anything that will run on the COSC lab computers also works.

Make sure to include *everything* (i.e. the whole folder), to avoid forgetting an important file. (e.g. even if you include a .class file, and a .pdf copy of the source file, the marker can't confirm that compiles)