Assignment 6

Intro to Programming: Assignment 6

- Due Nov 22 14:30

Goals

This assignment will give you experience building programs with simple event-driven input (mouse and buttons), and programs using an ArrayList of numbers.

Resources and links

- Zip file of necessary code and data.
- Submit your answers.
- Marks and Feedback
- Video of what your CircuitDrawer should do.
- Video of what your WaveformAnalyser should do.

Summary

- CircuitDrawer:
  ➡ Complete the `CircuitDrawer.java` program that allows the user to draw simple circuit diagrams.
- Waveform Analyser:
  ➡ Complete the `WaveformAnalyser.java` program that loads a waveform from a file, displays it, and analyses it in various ways.

Marking: For each program, up to 75 for the core, up to 90 for the completion, and up to 100 for the challenge.

To Hand in

You should submit your versions of `WaveformAnalyser.java`, and `CircuitDrawer.java` by the due date.

Preparation

Copy the files to your USB stick, or download and unzip the zip file from the web.

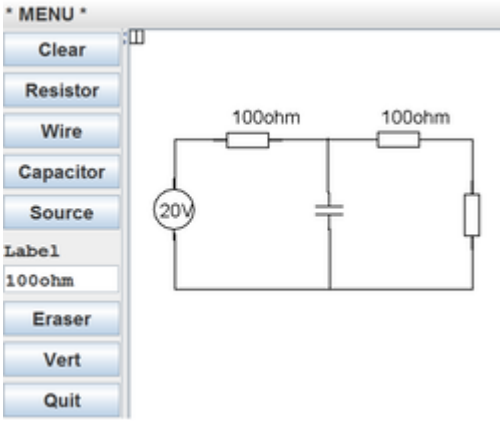
CircuitDrawer Program (Do in first week)

For this question, you must design and implement a simple application called `CircuitDrawer` that lets a user draw simple circuit diagrams on the screen.

`CircuitDrawer` lets the user select different "tools" to draw different components: resistors, capacitors, sources, wires, and labels. The user can use the mouse to add components or labels to the diagram. The user can also select an eraser to erase bits of the diagram. There are buttons to clear the screen, and to switch between horizontal and vertical mode.

`CircuitDrawer` has

- a field to store the name of the current tool and
- a field to store the current label.
- a constructor that will set up the GUI,
- methods to respond to the buttons,
- a method to respond to the mouse, which will call
- methods for drawing the components



`CircuitDrawer` should have the following buttons:

- **Clear**: Clears the graphics pane.
- **Resistor**: After the user has selected **Resistor**, the user can place a resistor (a small rectangle with a short wire on each end) at a point on the diagram using the mouse. The resistor will either be horizontal or vertical, depending on whether the program is in horizontal mode or vertical mode.
- **Capacitor**: After the user has selected **Capacitor**, the user can place a capacitor (two parallel lines with a short wire on each side) at a point on the diagram using the mouse. The capacitor will either be horizontal or vertical, depending on the mode.
- **Source**: After the user has selected **Source**, the user can place a source (a circle with a short wire on each side) at a point on the diagram using the mouse. The wires will either be horizontal or vertical, depending on the mode.
- **Wire**: After the user has selected **Wire**, the user can draw wires on the diagram by dragging the mouse from one point to another. Ideally, a wire is always made of a horizontal line and a vertical line. If the wire goes from (x,y) to (u,v), then the horizontal line goes from (x, y) to (u, y), and the vertical line goes from (u, y) to (u, v).
  Although an ideal program would show the wire being drawn while the user drags the mouse, `CircuitDrawer` is simpler, and only shows the wire once the user has released the mouse.
- **Label**: After the user enters some text in the **Label** box, the user can place the label at a points on the diagram using the mouse.
- **Eraser**: After the user has selected **Eraser**, the user can erase a small circular region by releasing the mouse button at some point on the screen.
- **Horiz/Vert**: Changes the mode between horizontal components and vertical components. Clicking the button should make the mode be the opposite of its current value.

Note that none of the buttons (except **Clear**) actually change anything on the graphics pane when they are selected. All actual drawing happens in response to dragging and releasing the mouse on the graphics pane. The effect of a button is to change the internal state of the program, so that the next mouse action will draw a different shape.

Core

Implement `CircuitDrawer` so that it will

- Set up the user interface correctly (the constructor and the fields).
- Respond to the **Clear** button correctly,
- Select the **Resistor**, **Capacitor**, **Source**, **Wire** and **Erase** tools correctly.
  (The `doSetResistor` etc, methods)
- Draw a horizontal resistor centered where the mouse is released if the **Resistor** tool is selected.
  (The `doMouse` and `drawResistor` methods).
- Draw a horizontal capacitor centered where the mouse is released if the **Capacitor** tool is selected.
  (The `doMouse` and `drawCapacitor` methods)
- Draw a circle with short wires on each side centered where the mouse is released if the **Source** tool is selected.
  (The `doMouse` and `drawSource` methods)

- Draw a wire from the pressed point to the released point if the **Wire** tool is selected.
  For the core, the wire can be a straight line between the two points.
  (The `doMouse` and `drawWire` methods)
- Erase a circular region where the mouse is released point if the **Eraser** tool is selected.
  (The `doMouse` and `doErase` methods)

Hint: The most common difficulty that students have is working out exactly when and where everything should happen. Make sure that you watch the video carefully to see what happens when the user clicks buttons or drags on the diagram. Notice especially that when you click on the buttons (except the **Clear** button), the program *does not draw anything* - all it does is to remember which button was chosen. It is only when you release the mouse that it actually draws a wire or a component. What component it draws depends on what it remembered from an earlier button press. Note also that it doesn't draw anything when the mouse is pressed; only when the mouse is released. Therefore, to draw wires it must remember where the mouse was pressed.

Completion

Add the following features to your program:

- Make the wires draw correctly: The wire should consist of a horizontal line and a vertical line. But if the line is very short it only draws a small dot.
- Make the **Label** tool work so that the user can place text labels on the diagram.
- Make the **Horizontal/Vertical** button work so that it changes the mode.
- Make the `drawResistor`, `drawCapacitor` and `drawSource` methods draw a horizontal or vertical component, depending on the mode.

Challenge

There are lots of ways in which this program could be made nicer to use: (Note, some will require the UI to respond to MouseMotion.)

- Make the **Horiz/Vert** button show the current mode in some way (either by modifying the button, or drawing an icon in the corner of the diagram).
- Make the eraser tool erase the region everywhere the mouse is dragged
- Make the wire tool "rubber-band" - as the user drags the mouse, the program should show the wire being dragged out. However, it should not erase the bits of the diagram that it is dragged over. Hint: use the `invertLine` method.
- Make the tool buttons show which tool is active, (either by modifying the button, or drawing an icon in the corner of the diagram).

Waveform Analyser (Do in second week)

A very important part of electronics and communications is dealing with signals that are waveforms. A waveform can be stored as a sequence of numbers. This program will display a waveform and show interesting properties of it.

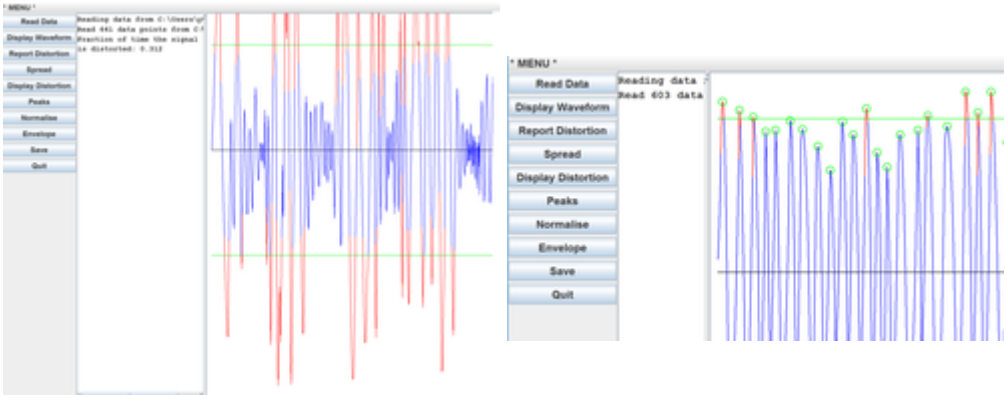
There are several aspects of a waveform that are interesting:

- The highest and lowest (most negative) points on the waveform are significant.
- The distortion level: if the signal is too large (positive or negative), the signal might get distorted. Identifying the points over the distortion level is useful.
- The peaks of the waveform: all the points that are greater than both their neighbouring points.
- The upper envelope which is the line linking all the peaks of the signal.
- The lower envelope which is the line linking all the "negative" peaks (the peaks below the zero line).
- Editing the waveform, eg by scaling it down so it is all smaller than the distortion level.

`WaveformAnalyser` is an extension of the **`NumberAnalyser`** program from assignment 4. They both read a sequence of number, display them, and perform some analysis. However, instead of reading data directly from the user, **`WaveformAnalyser`** reads the data from the file (eg **`waveform1.txt`** and **`waveform2.txt`**) into an ArrayList stored in the `waveform` field, which means that it can analyse the data more easily and in more different ways. It can also cope with much larger sets of numbers.

Another difference from the **`NumberAnalyser`** program is that the values can be **negative** and the signal swings above and below zero.

Here are the waveforms from the two files above. The one on the left shows the distortion. The one on the right shows the peaks as well as the distortion.



The code that sets up and responds to the buttons is provided for you - the parts of the program you have to complete are the ones involving the ArrayList.

The constructor sets up an interface with ten buttons:

- **Read Data**: [core] reads data from a waveform file into an ArrayList field.
- **Display Waveform**: [core] displays the waveform as a line graph (with straight lines between each pair of values).
- **Report Distortion**: [core] prints out the fraction of time the signal is distorted.
- **Spread**: [core] displays the maximum and minimum values with two horizontal lines (on top of the waveform).

- **Display Distortion**: [completion] displays a waveform that highlights in red the distorted values.
- **Peaks**: [completion] plots the peaks with small green circles.

- **Normalise**: [challenge] normalises all the values by scaling them down (multiplying each value by the same scale factor) so there is no distortion, and redisplays the waveform.
- **Envelope**: [challenge] displays the upper and lower "envelope" of the displayed waveform by connecting all the peaks.
- **Save**: [challenge] saves the current waveform values into a file.

Core

For the core, you should complete the following methods:

- `doRead()`: creates an ArrayList stored in the `waveform` field, asks user for a waveform file and reads data from the file into the ArrayList.
- `doDisplay()`: Displays the waveform as a line graph. The horizontal line representing the value zero should also been drawn. If the lines go off the window, that is OK.
- `doReportDistortion()`: Computes and prints the faction of time the signal is distorted. This fraction of time is defined as the number of values that are larger than the threshold (in the constant `THRESHOLD`) divided by the number of values. Note, a distorted value is one that is greater than the positive value of the threshold or less than the negative value of the threshold.
- `doSpread()`: Finds the maximum positive value and the minimum negative value, and displays two horizontal lines on the waveform, a green line for the maximum value and a red line for the minimum value.

Completion

For the completion, you should complete the following methods:

- `doDisplayDistortion()`: Displays the waveform as a line graph, showing the distorted part of the signal in red. This methods draws in red every line that has either end point beyond the distortion threshold.
- `doHighlightPeaks()`: Displays the waveform and the places small green circles on all the peaks. A peak is defined as a point that is greater or equal to **both** neighbouring points. Note the size of the circle is stored in the constant `SIZE_CIRCLE`

Challenge:

For the challenge, you should complete the following methods:

- `doNormalise()`: Finds the largest value (positive or negative) in the waveform, and scale all the values so that the largest value is now equal to the distortion threshold. Then redraws the waveform.
- `upperEnvelope()`: displays the upper envelope with green lines connecting all the peaks, as in the demo.
- `lowerEnvelope()`: displays the lower envelope with red lines connecting all the "negative" peaks, as in the demo.
- `doSave()`: asks the user for a filename and saves the current waveform to that file.
