CS 4053/5053
Homework #5 – Blobster!
Due Thursday 2018.04.19 at the beginning of class.

All homework assignments are individual efforts, and must be completed entirely on your own.

In this assignment you will combine affine transformations, interactive editing, and convex hulls to create a simple app for visualizing social networks. You will apply translation to pan the entire view, rotation and scaling to draw individual nodes, and an incremental convex hull algorithm to draw a blob around the entire network of nodes.

The parts of the assignment are designed to be done in order, with new features being added in each part. Complete as many parts (and parts of parts) as you can. I will grade progressively and be generous with partial credit. It's all doable with what we've covered in class and in the textbook. Read everything first then carefully think about how to model it all as objects. *I've broken down the steps to read something like a mix of a user interface specification and a tutorial. Hopefully this will also make it much easier to work on the assignment piecemeal for lots of partial credit*.

The assignment has six main parts. The **first part** is to update your Gradle build. Do this just like you did for homework #4, except add a `hw05` build target for your application. Note that I added a class that you'll need (`Network`) to the `interaction` package, described below. I strongly encourage you to start with my code...even more strongly than I did for Homework #4.

The **second part** is to create a base view that supports panning interactions. The `View` class in the `interaction` package does a lot of this for you. Have a close look at how it works if you haven't already. The view applies a 2-D affine transform (in `updateProjection()`) to position the origin relative to the frame. The `KeyHandler` class already supports stepping in x and y. Your job is to add support for dragging in the `MouseHandler` class. When you detect a `mousePressed` event, record the starting location of the origin and the mouse cursor. When you detect a `mouseDragged` event, set the view origin based on that information along with the current location of the mouse cursor. Repeat until you detect a `mouseReleased` event.

The **third part** is to let the user create, select, and delete nodes that represent people in the social network. You can get a list of all names using the `Network.getAllNames()` method. Starting with zero nodes, support the following interactions and corresponding graphical effects:

- The `'<'` and `'>'` keys cycle through names *that don't yet appear as nodes*. Show the currently selected name as yellow bold text always in the lower left corner of the view. Don't forget to handle the case in which there are no unused names left!
- The `<return>` key adds a node for the currently selected name. Draw each node as a regular polygon, filled and edged. Call `Network.getSides()` to get the number of sides and `Network.getColor()` to get the fill color for a particular name. Randomly center the node near the origin and set its initial width and height to 0.1 (in view coordinates).
- The `','` and `'.'` keys cycle through the visible nodes. Edge the selected node in white, other nodes in dark gray. When a node is added, select it. Allow no node to be selected.
- The `<delete>` key removes the selected node (if any) and puts it back on the list of available names to add. Upon removal, automatically select the "next" node. Appropriately handle the case when the user deletes the last visible node.

The **fourth part** is to allow the user to change the appearance of individual nodes. First, allow the user to select a node with the mouse. When you detect a `mouseClicked` event, select the *frontmost* node (from the perspective of the user) that contains the click point. Second, if a node is selected, support the following interactions and corresponding graphical effects:

- With the `<shift>` key *up*, the four `<arrow>` keys translate the selected node by 0.1 times its current width or height.
- With the `<shift>` key *down*, the four `<arrow>` keys scale the selected node by 1.1 times its current width or height.
- With the `<shift>` key *up*, clicking inside a node then dragging causes the node to translate the corresponding amount, relative to the node's center.
- With the `<shift>` key *down*, clicking inside any node then dragging causes the node to rotate by the angle swept out during the drag, relative to the node's center.

To accommodate these last two interactions, only allow mouse drags to pan the entire view (see **part two**) when the `mousePressed` point is not in any of the nodes.

The **fifth part** is to draw a convex hull around the centers of all visible nodes. Recall that for a set of 2-D points, the convex hull is the minimal polygon that contains all of them. Use the basic incremental algorithm described in class. It's not optimal, but it is pretty easy to implement and works well when points can move, such as in response to interaction. To insert a point:

- If the hull is empty, it becomes just the insert point.
- If the hull is one point, it becomes the line segment connecting it to the insert point. (Unless the points are the same, in which case the new point is "on" the hull and you're done.)
- If the hull is a line segment, it becomes the triangle connecting the points counterclockwise. (Unless the three points are co-linear, in which case the hull becomes the minimal line segment containing all three.)
- If the hull is a polygon (three or more sides): (1) Starting with any point on the polygon, loop around the edges of the existing hull clockwise until you find one that the insert point is strictly "to the left" of. (2) From there loop around the edges counterclockwise until you find one that the insert point is strictly "to the right" of. If there isn't such an edge, the point is on or in the hull and you're done. (3) Loop counterclockwise starting from the *next* edge, removing the start point of each edge from the hull until the insert point is strictly "to the left" of an edge. Add the insert point to the polygon just before the start point of that edge.

To remove a point (brute force, but easy and effective):

- If it's not one of the hull points, you're done.
- If it is one of the hull points: (1) Remove it from the hull. (2) Loop through *all the other non-hull points*, trying to insert each one following the point insertion steps above.

Hint: The `java.util.LinkedList` class is a convenient way to store the hull points (and also the non-hull points) for use with the above algorithm. Push and pop to loop the polygon...

Maintain the convex hull from the start, using node centers as points. Update the hull whenever adding, deleting, and moving nodes. *To move, remove then (re)insert.* To draw the hull, fill and edge it with colors of your choice. Use the edge color when the hull isn't a polygon.

The **sixth part** is to add a balloon effect to the convex hull calculation, like we covered in class. *(Download slides-20180405.pdf from Files/Slides in Canvas. The relevant whiteboard photos are on the last slide in the "Parametric Curves" section.)* To balloon a hull, move each edge of the hull "outward" then connect each pair of edges with an arc. Handle the special cases of non-polygon hulls appropriately. Let the user control the amount of ballooning interactively, with zero as the minimum balloon amount (equivalent to a regular convex hull).

There are a variety of bonuses available, **and this time the maximum score can be above 20/20**. The bonuses are designed to be independent and to vary a lot in difficulty.

BONUS (second part): Support zooming of the entire view with the keyboard and/or mouse.

BONUS (second part): Support rotation of the entire view with the keyboard and/or mouse.

BONUS (third part): Support keyboard interactions to lay out the currently visible nodes in a line or circle (with reasonable spacing).

BONUS (fourth part): Store and recall the location, size, and rotation of each node even when they are deleted then re-added later.

BONUS (fifth part): Support dragging the entire convex hull (including interior points) when the user clicks inside the hull (but the click isn't inside one of the nodes).

BONUS (fifth part): For each visible node, base the convex hull calculation on its polygon points instead of its center.

BONUS (fifth part): Clearly improve upon the brute force approach used to remove a point in the convex hull calculation. (Document your approach clearly and thoroughly in both your code and your `description.txt` file if you choose to attempt this bonus.)

BONUS (sixth part): Give the convex hull a shaded 3-D appearance like a raised rounded button, with its apparent height proportional to its balloon amount.

You will be graded on: (1) how completely and correctly you realize the interactive appearance and behavior in the above parts, and (2) the clarity and appropriateness of your code. It's a complex set of features, though, so **don't** worry about getting all the parts and subparts working perfectly. **Do** use affine transformations—of the entire view and when doing calculations with individual polygons—wherever you can to make your job much easier.

To **turn in** your homework, first append your 4x4 to the `homework05` directory; mine would be `homework05-weav8417`. Second, perform several interactions to build an interesting looking social network view, take a screenshot of your creation, trim it, and put it in the `Results` subdirectory as `snapshot.jpg` or `snapshot.png`. Third, write brief descriptions of (1) anything special you'd like me to be aware of in grading each part, and (2) which bonuses you attempted and how. Put these in the `Results` subdirectory in `description.txt`. Where appropriate, include longer blocks of documentation in your code and label them to make it clear that that's what they are. Fourth, run `gradle clean` to reduce the size of your build before turning in your homework. Leave your java files where they are in the build. Zip your entire renamed `homework05` directory and submit it to the "Homework 05" assignment in Canvas.