CSE 341, Winter 2008, Assignment 7
Due: 14 March, 8:00AM

Last updated: March 5, 2008

Overview: We provide a skeleton ∼270 lines of code for drawing Facebook-style "friend wheels." You will need to read, understand and extend this code (∼130 more lines) to complete its functionality. Some notes:

- A contact list is represented as an array of `Contact`s.
- Friendship is not symmetric—if `Contact` a is friends with b, that doesn't imply b is friends with a.
- Drawing friend wheels requires generating social networks, done by subclasses of `ContactsGenerator`, laying out each person on screen, done by subclasses of `ContactLayoutManager`, and coloring each person, done by subclasses of `ContactColorManager`. You will need to complete the skeletons we've provided, and implement additional subclasses of each of these three classes.
- We've provided two debugging routines to aid printing the social networks you create:
  - `debugPrintContactNames` takes a contact list and prints the name of each person followed by the names of their friends. If a person has secret admirers, those names are printed in parentheses.
  - `namesOnly` takes an array of (`Contact`s or arrays of (`Contact`s or arrays of ...)), and produces a new array of just the names of the contacts, with the same nesting structure as the original array.
- The `RGB` class provides three mechanisms to create colors: `from_str` takes a named color; `from_rgb` takes a $(red, green, blue)$ triple (each component is between 0 and 1); and `from_hue` takes a hue (from 0 to 1) and returns a color from purple to red.
- The skeleton code we provide you supports five key-bindings:
  - Press 'l' to change the layout manager of the current contacts, without changing colors.
  - Press 'c' to change the color manager of the current contacts, without changing layouts.
  - Press 'n' to change to another contact list, using the same color and layout managers.
  - Press 'p' to take a screenshot of your program. It will create a PostScript file of the name ⟨*layoutmanager*⟩_⟨*colormanager*⟩_⟨*generator*⟩.ps in your current directory.
  - Press 'q' to quit.

1. Naming contacts: Currently, all contacts display their names as "⟨*name*⟩". Redefine one method to change how *all* contacts display their names, to "⟨*name*⟩ (⟨*number of friends*⟩)". Sample solution is 5 lines (counting all boilerplate: `def`/`end`, `class` lines, etc.). Do not modify the top half of the file!

2. Coloring contacts: Contacts are colored by subclasses of `ContactColorManager`, which are responsible for setting the `color` field of contacts. We've provided two example color managers: one assigns colors randomly; the other assigns colors by contact type.

   (a) Complete the `color` method of `ContactColorByDegree` to assign a color to every contact by their out-degree (the number of people they have as contacts). Colors should span the rainbow with cooler colors (purple and blue) for lower degrees and warmer colors (red and orange) for higher degrees. Specifically, if your social network has a maximum degree of n and a minimum degree of m, and some person has degree d, then you should generate a color of hue $(d - m)/(n - m)$. See Fig. 1 for how the colors look for `ContactsGenCircle` and `ContactColorByDegree`, laid out by `ContactLayoutInCircle` (when complete). Sample solution is 8 lines counting all boilerplate.

   (b) Create a `ContactColorByDegreeSym` class that assigns colors as above, except it treats friendship as symmetric when computing degrees. See Fig. 2 for how the same contact list appears with this color manager and `ContactLayoutInCircle`. Sample solution is 9 lines counting all boilerplate.

3. Creating contact lists: We've provided a single contact list generator, `ContactsGenCircle`, which generates n people where person i is friends with person n and all people 0 through $i-1$. Create three more contact lists by subclassing `ContactsGenerator`, looking like:

   (a) A "fan": this creates n people, where person 0 is friends with everyone, and person i is friends with $i-1$, $i+1$ and 0. See Fig. 3. Sample solution is 20 lines counting boilerplate.

   (b) A "double-circle": this creates $2n$ people, which split into two circles of friends (as in the example), but neither circle is friends with the other. See Fig. 4. Sample solution is 11 lines with boilerplate.

   (c) A "tree": this creates 2^n people, where person i is friends with person $i/2$ (his "parent") and $2i$ and $2i+1$ (his "children"). See Fig. 5. Sample solution is 11 lines counting boilerplate.

   NOTE: until you complete problem 5 (laying out contacts), your displayed result will not look like the diagrams. Until then, you should test your routines with either the debugging routines or with very small group sizes, so as to make sense of the random layout...

4. New contact types: Define a `SecretAdmirer` class that *does not* draw lines connecting it to anyone else, but that does affect the number of friends of all of its contacts: if a secret admirer s likes a person p, and p isn't a friend of s, then p's friend count is increased but no line is drawn. If p is a friend of s also, then a line is drawn and the friend count isn't affected. Note that p could be a `SecretAdmirer`, too. To do this, create a new subclass of `Contact`, then figure out which other classes may need to be modified or extended. (Hints: how can contacts keep count of their secret admirers? Look in `debugPrintContactNames` for some ideas. Also, you may want to define a new contact list generator that creates multiple types of people, including secret admirers, so you can test your approach.) Sample solution is 30 lines counting all boilerplate.

5. Laying out contacts: Layout managers are responsible for setting the `x`, `y`, `tx` and `ty` fields to lay out the contact and its name. We've provided one layout manager that lays out contacts randomly.

   (a) Complete `ContactLayoutInCircle` to lay out contacts as seen in Figre 1. Using the provided `polarToRect` method will make points on the unit circle fill the available canvas space, where θ starts at the right and turns clockwise. Sample solution is 6 lines counting all boilerplate.

   (b) Complete `ContactLayoutInTree` to lay out contacts as seen in Fig.5. Make sure your solution fills the available space (see `polarToRect` for how to get the canvas' size). Sample solution is 15 lines counting all boilerplate.

   (c) **Challenge Problem:** Implement `ContactLayoutInTreeCentered`, another layout manager that is similar to `ContactLayuoutInTree`, but where each level of the tree is horizontally centered in the screen. See Fig. 6. Sample solution is 17 lines counting all boilerplate.

Turn-in Instructions

- Put all your solutions in one file, `lastname_hw7.rb`, where `lastname` is replaced with your last name.
- The first line of your `.rb` file should be a Ruby comment with your name and the phrase `homework 7`.
- Take screenshots of the following combinations:
  1. "Double circle" laid out in a Tree colored by Degree
  2. "Tree" laid out in a Tree colored by Type
  3. "Circle" laid out in a Circle colored by Symmetric Degree
  4. "Fan" laid out in a Circle colored by Degree.
  5. A social group using `SecretAdmirer`s laid out in a Circle colored by Degree.
  6. **Challenge Problem** "Double circle" laid out in a Centered Tree colored by Degree.
- Go to `https://catalysttools.washington.edu/collectit/dropbox/djg7/1359` (link available from the course website), follow the "Homework 7" link, and upload your files (Ruby code and screenshots).

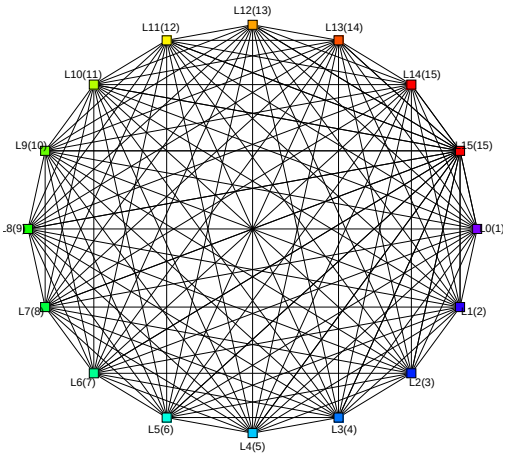

Figure 1: "Circle" laid out in a Circle colored by Degree

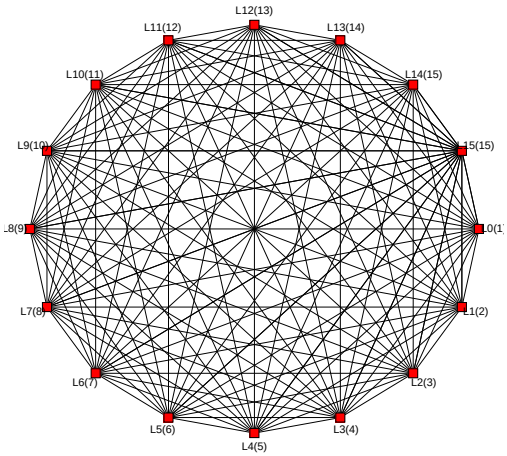

Figure 2: "Circle" laid out in a Circle colored by Symmetric Degree

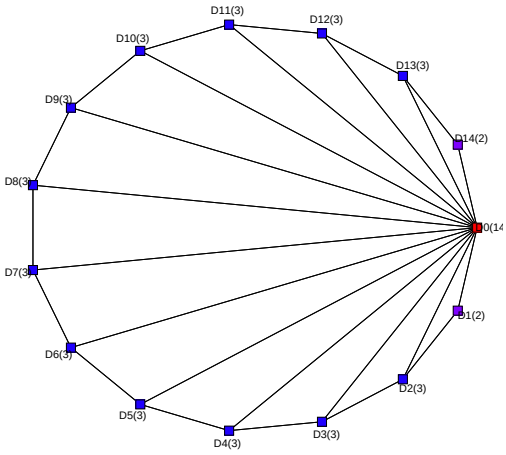

Figure 3: "Fan" laid out in a Circle colored by Degree

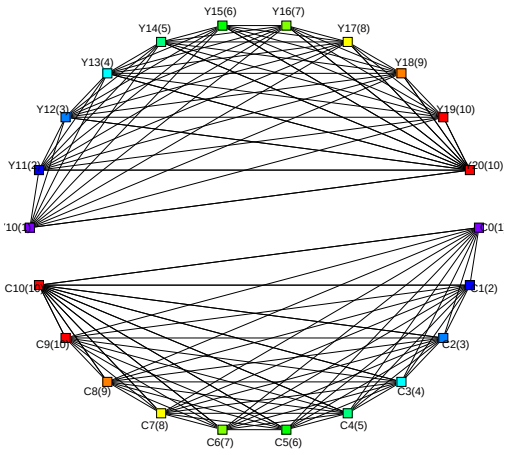

Figure 4: "Double circle" laid out in a Circle colored by Degree

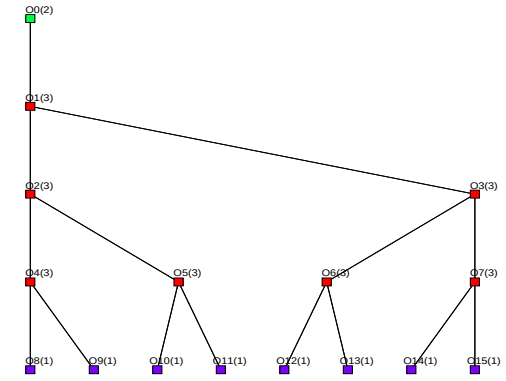

Figure 5: "Tree" laid out in a Tree colored by Degree

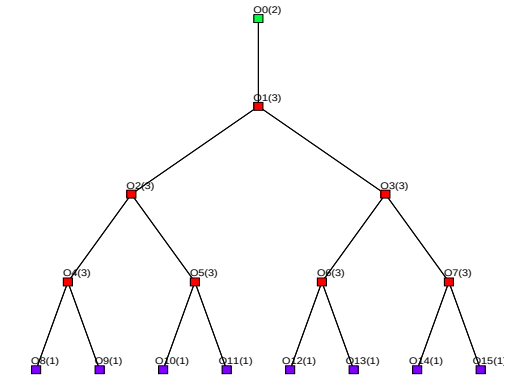

Figure 6: **Challenge:** "Tree" laid out in a Centered Tree colored by Degree