Final Project: Baby Name Trends

Object Oriented Programming

Due Dates: There are four Milestones and a final delivery (see below)

This programming assignment is an application that slices, dices, and graphs data from a big database of names. This is a realistic and challenging project; it looks like a real program.

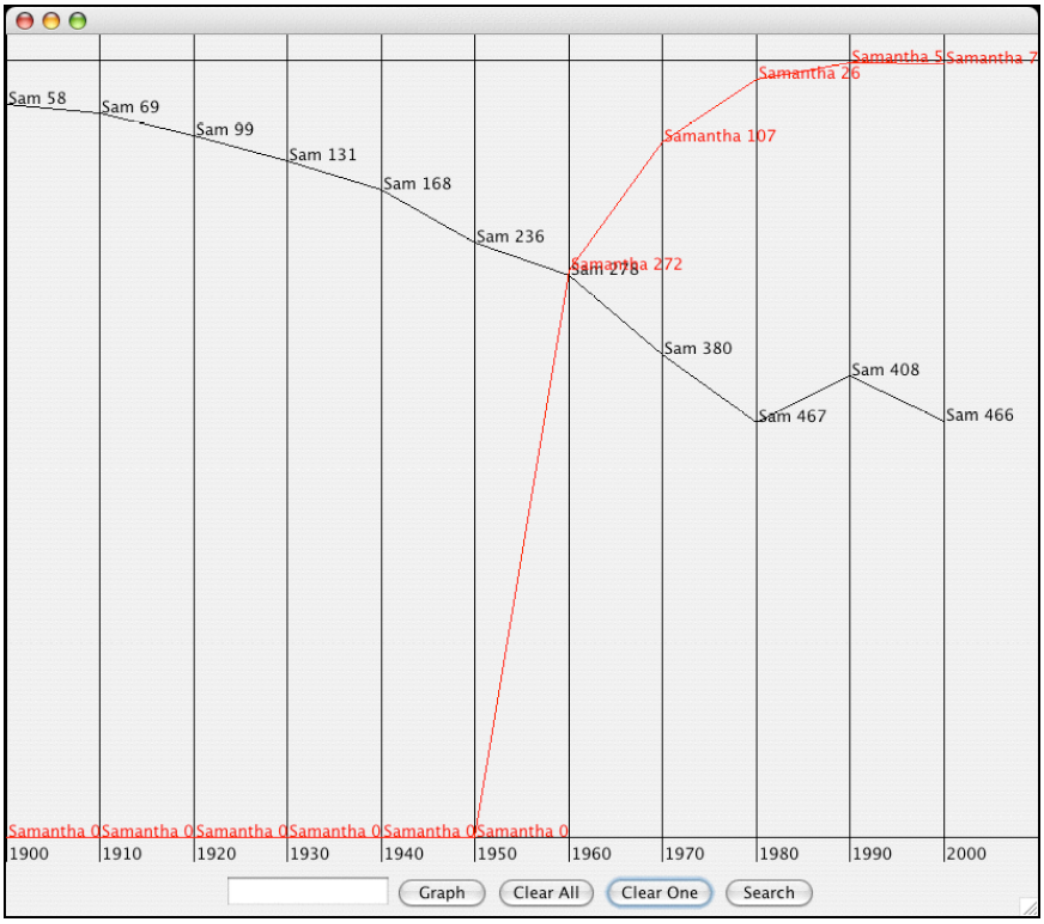
Against all bureaucratic stereotypes, the Social Security Administration, provides a neat web site showing the distribution of names chosen for kids over the last 100 years in the US: http://www.ssa.gov/OACT/babynames/

Every 10 years, the data gives the 1000 most popular boy and girl names for kids born in the US. The data can be boiled down to a single text file as shown below. On each line we have the name, followed by the rank of that name in 1900, 1910, 1920, ... 2000 (11 numbers). A rank of 1 was the most popular name that year, while a rank of 997 was not very popular. A 0 means the name did not appear in the top 1000 that year at all. The elements on each line are separated from each other by a single space. The lines are in alphabetical order, although we will not depend on that.

```
...
Sam 58 69 99 131 168 236 278 380 467 408 466
Samantha 0 0 0 0 0 0 272 107 26 5 7
Samara 0 0 0 0 0 0 0 0 0 0 886
Samir 0 0 0 0 0 0 0 0 920 0 798
Sammie 537 545 351 325 333 396 565 772 930 0 0
Sammy 0 887 544 299 202 262 321 395 575 639 755
Samson 0 0 0 0 0 0 0 0 0 0 915
Samuel 31 41 46 60 61 71 83 61 52 35 28
Sandi 0 0 0 0 704 864 621 695 0 0 0
Sandra 0 942 606 50 6 12 11 39 94 168 257
...
```

We see that "Sam" was #58 in 1900 and is slowly moving down. "Samantha" popped on the scene in 1960 and is moving up strong to #7. "Samir" barely appears in 1980, but by 2000 is up to #798. The database is for children born in the US, so ethnic trends show up when immigrants have kids.

Ultimately, we want to organize that data to graph it as shown below with the names Sam and Samantha:



One of the neat things about this project is that it uses real data (and lots of it!). There are over 4500 names in the database. The data just records what people put on the forms, so there are things like "A" and "Baby" recorded as names (the data is more cleaned up in the later years). We will not worry about that, and we will not combine names that are similar in some sense -- "Cathy" and "Catherine" and "Kathryn" and "Katie" and "Kati" will all count as different names. Looking at the data, long names ("Michael") are becoming more popular compared to their short versions ("Mike").

To the surprise of no one, the best approach to this somewhat large problem is to decompose the problem into separate classes that can be built up gradually:

- NameRecord – encapsulates the data for a single name.
- NameComponent – draws line graphs showing the history of a few names.
- NameFrame – the main class for the whole thing. Stores all the data and directs the other two classes.

Part 1 – Name Loading

The first step is creating the NameRecord class. Each NameRecord encapsulates the data for one name — the name and its rank over the years. This is essentially the data of one line from the file shown above. Use an int array to store the int rank numbers.

- Constructor – takes a String line as in the file above and sets up the NameRecord object.
- String getName() – returns the name
- int getRank(int decade) – returns the rank of the name in the given decade. Use the convention that decade=0 is 1900, decade=1 is 1910, and so on. The NameRecord constants START=1900 and DECADES=11 define the start year and the number of decades in the data.
- int bestYear() – returns the year where the name was most popular, using the earliest year in the event of a tie. Looking at the data above Samir's best year is 2000, while Sandra's best year is 1940. Returns the actual year, for example 1920, so the caller does not need to adjust for START. It is safe to assume that no rank in the data is ever larger than 1500, and every name has at least one year with a non-zero rank.

NameFrame read()

Create the NameFrame class as a subclass of JFrame. At this step, the frame is visually empty. It will get components in later steps – for now we just deal with getting the data loaded. Implement a read(String filename) method in NameFrame that reads the given file, creates a NameRecord for each line in the file, and stores them all in an ArrayList ivar *(instance variable)*. In main() create a new NameFrame and send it read("name-data.txt") – this will, in essence, read the entire file of name data into your ArrayList ivar, ready for use.

Milestone 1 *due date: Saturday, December 4, 2021 (7 points)*

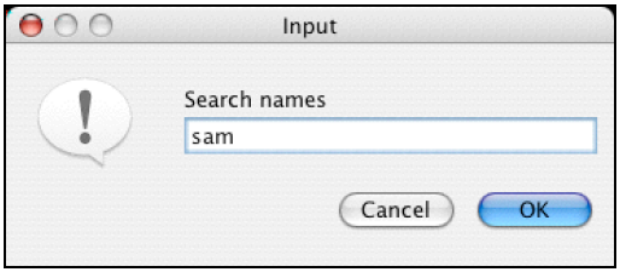
Create the NameFrame and read the data. Use the debugger to look in the ArrayList at a few of the NameRecord objects. They should have the correct name rank data in them from the file.

Submit in one zip file, your NameRecord.java and NameFrame.java source code files, plus 3 sample screenshots that display different elements of your ArrayList as viewed in the debugger.

Part 2 — Search

Implement a search(String target) method in NameFrame that takes a string, and searches through all the names for any that contain that substring (not case-sensitive). Print out a line for each matching name to System.out, printing the name followed by its best year. It's fine to search the obvious way – just loop through them all and check with an if statement. Soon, we will learn another way to find a String in a collection, but in this case the simplest strategy still does the job in a blink of an eye.

Implement a doSearch() method that loops, calling showInputDialog() to get a string, and then calls search() with that string to print all the matching names. Continue looping until the user hits cancel. From main(), if the SEARCH constant is true, send doSearch(). Here we do a search with the string "sam" and the names and best years print on the console:




```
Isamar 1990
Rosamond 1910
Sam 1900
Samantha 1990
Samara 2000
Samir 2000
Sammie 1930
Sammy 1940
Samson 2000
Samuel 2000
```

Milestone 2 *due date: Tuesday, December 7, 2021 (7 points)*

NameFrame main() should create the frame, read the data, and then doSearch() should drive the search() method. Compare the results for a few names to the real data to check your basic NameRecord constructor and bestYear() code. An "end-to-end" check means we spot-check some of the raw data in the file against the output data to see that the intervening code is not mangling the data in the process. As part of divide-and-conquer, we want to work out the basic data bugs now, not later in part-4 when we're concentrating on the graphing.

The empty string counts as a substring of anything, so searching for the empty string will print the stats for all 4500 names. The computation is quite fast, although Eclipse can be a bit slow to display so much console output.

Submit your updated source code (*.java files) plus a typed report (2-3 pages) explaining your Milestone 2 progress. Make sure to include relevant screenshots.

Part 3 — Graph

Create a NameComponent subclass of JComponent, with a preferred size of SIZE=600 by 600.

paintComponent()

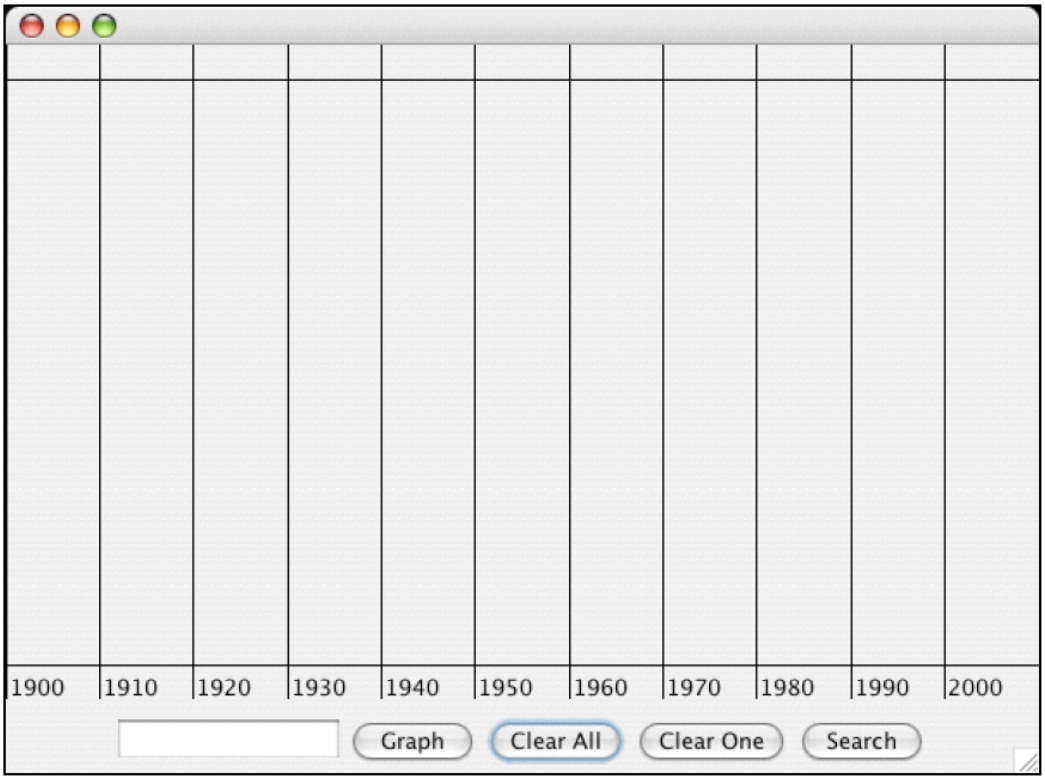
The paintComponent() method should draw a line graph as follows:

- Draw 11, evenly spaced lines to make each decade, with the 1900 line at x=0. drawString() a "1900", "1910", ... just to the right of each line.
- Reserve SPACE=20 pixels of space at the very top and bottom of the graph. Draw black horizontal lines to mark the top and bottom of the graph.

Milestone 3 *due date: Friday, December 10, 2021 (7 points)*

In the NameFrame constructor, create a BorderLayout, and install a single NameComponent in the center. Its paintComponent() code should draw the basic graph structure (we'll put in data in the next step). Resizing the window should resize the graph. (Never mind the controls at the bottom of the frame, those are part 5.)

Submit your updated source code (*.java files) plus a typed report (2-3 pages) explaining your Milestone 3 progress. Make sure to include relevant screenshots.



Part 4 — Graphing

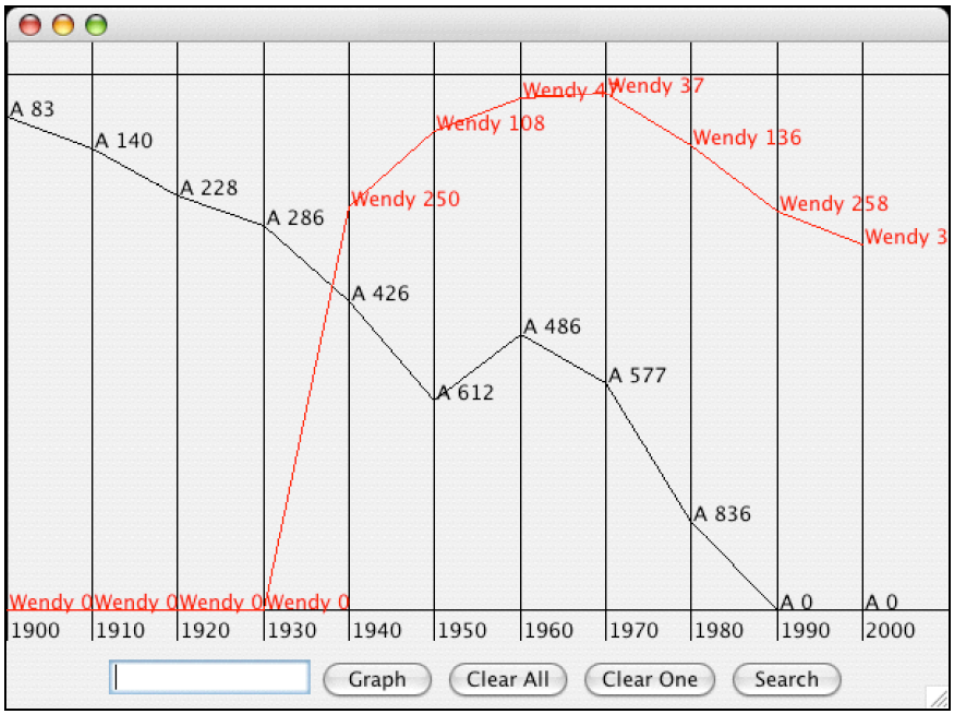
The NameComponent should maintain an ArrayList of NameRecords, and draw a line graph for each.

- Maintain an ArrayList ivar that stores some NameRecords.
- public void addName(NameRecord record) should add a name to the component's ArrayList and repaint().
- public void paintComponent() should draw a line graph for each of the NameRecord as follows:

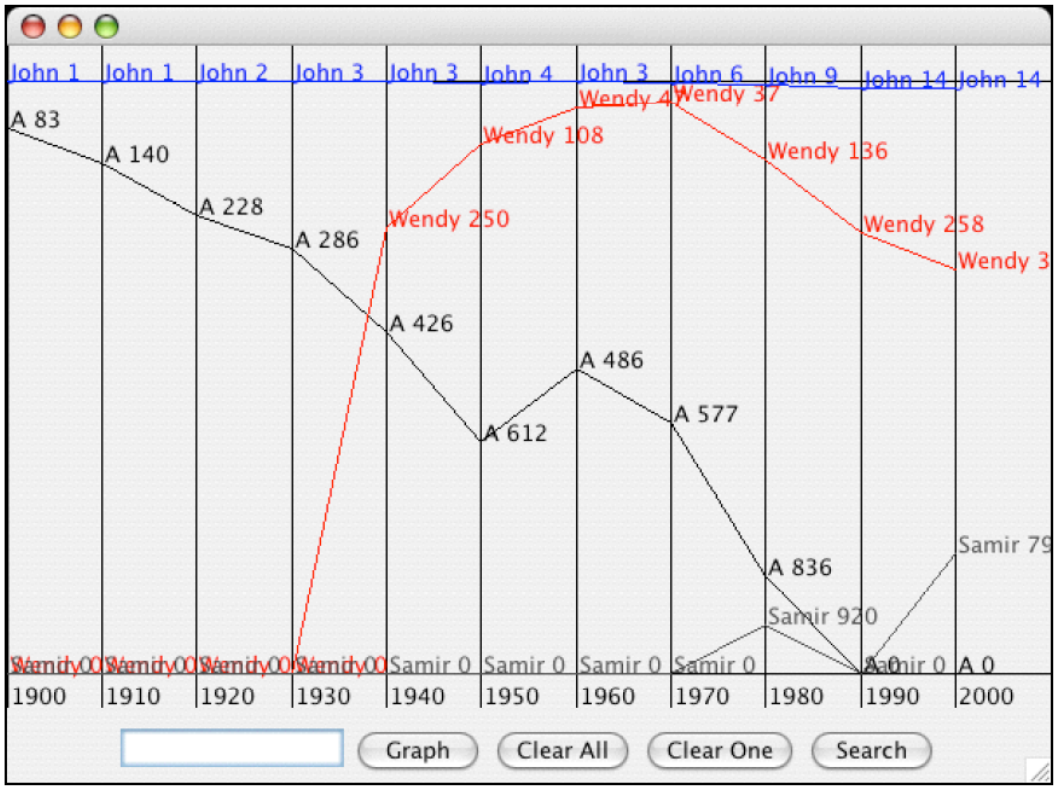
  For each decade vertical line (1900, 1910), figure out the y value for the name's rank that year. If the rank is 1, the y should be at the very top (covering the top horizontal line). If rank is 1000 or more, or 0, the y should be at the very bottom (covering the bottom horizontal line). Connect the points with lines. 2 pixels to the right of each point, drawString() the name and its rank number.

  The constant COLORS is an array of 4 Color objects. Pick a color for each name depending on its order in the ArrayList. Draw name 0 with color 0. Draw name 1 with color 1, and so on. When the number of names is greater than the number of colors, wrap around to use the first color again. (The mod operator % is perfect for this wrap-around notion.)

As we add names, they will tend to draw on top of each other, especially at the very top and very bottom. Since the name string is repeated each decade, it's still possible to figure out which line is which. Here we see the names "A" and "Wendy". "A" starts strong in 1900 and trails off to 0 in 1990. Wendy is at 0 until 1940.



Now we add "John" who is very near 1 the whole time, and "Samir" who comes on the scene only starting in 1980. Both Wendy and Samir are 0's in 1900, 1910, ... so they draw on top of each other there. That's fine – we draw what we can and if they draw on top of each other, so be it. The user can make the window bigger to try to make more space for things.



Milestone 4 *due date: Tuesday, December 13, 2021 (7 points)*

From the NameFrame side, just for testing, send a single addName() message to the NameComponent to add the "A" NameRecord (the very first NameRecord in the ArrayList). The A graph should appear as above. Resizing the window should resize the graph. Getting this one bit of drawing to work is a good part of the challenge. Adding the other features is relatively easy once the graphing works.

Submit your updated source code (*.java files) plus a typed report (2-3 pages) explaining your Milestone 4 progress. Make sure to include relevant screenshots.

Part 5 — User Interface

In the NameFrame constructor we will set up the rest of the controls. Add a JPanel to the SOUTH. In that JPanel, add a JTextField that holds 15 characters, and the Graph, Clear All, Clear One, and Search JButtons.

Use the standard pattern to connect the controls to the frame: implement the ActionListener interface, addActionListener() for each control, implement the actionPerformed() method with logic to detect which button was clicked.

Write a findName(String name) method that takes a string and returns the matching NameRecord with that name, or null if none is found. The name should match exactly (not case-sensitive), as opposed to the earlier search() code where any substring match worked. Clicking the Graph button should findName() the text from the text field. If it finds a matching NameRecord, add it to the NameComponent to graph. If the name has no match. beep with the following code phrase: "Toolkit.getDefaultToolkit().beep();". Typing return in the text field should work the same as clicking the Graph button.

Implement a clearAll() method on the NameComponent that removes all the NameRecords and repaints. Implement a clearOne() method that removes the earliest added name and repaints (so if I add names Nick, Wendy, and Samir, clearOne() removes Nick). (The remove(0) message sent to an ArrayList removes the 0th element.) When the buttons are clicked, the NameFrame should send those corresponding messages to the NameComponent. It's fine that which color goes with which name changes as names are added and removed.

Finally, add a Search button that gets the text out of the text field and calls the search() from part-2 with that string, printing to the console. This works without the showInputDialog() stuff, since we can now use our own JTextField to get the text from the user.

Neat Searches

Now you can spend many hours playing around with different searches to see things in the data. Think of a pattern you might expect and see if it's there.

- Why is Rock popular in 1950 and Trinity in 2000?
- Michael is very popular. To see the growing Spanish speaking influence in the US, look at Miguel. For a more recent immigration, look at Muhammad and Samir.
- Apparently Biblical old-testament names came back in the 1970's. A reaction to the 1960's maybe? Try Rachel and Rebecca. The pattern seems to generalize – Sarah, Abraham, Adam; Eve and Moses are out of luck for some reason though.
- Try historical names like Sigmund or Adolf. Adolf is tricky, since there are a bunch of variant names like Adolph. Use the Search feature to find them.

Final Deliverables *due date: Friday, December 17, 2021 (9 points)*

Congratulations on finishing a very realistic and functional project: OOP, data files, searching, and complex drawing. You should be able to use the text field and Graph buttons to put in multiple names. They should be graphed correctly. You should be able to resize the window, and everything should scale. You should be able to use Clear and Search buttons.

The basic assignment provides plenty of challenge, and no further work is required. However, for people interested in playing with it more, here are some possible extra features: One feature to add is a JTextArea to show the output of search, rather than using println(). JTextArea is like JTextField, but bigger. Its constructor takes the int number of rows and columns, and it responds to setText() to install text. Another interesting feature would be to type in a decade, like 1970, and the program figures out the top 5 or whatever names for that decade and graphs them.

Submit your final source code (all your *.java files) plus a typed report presenting your concluding test results. Make sure to include relevant screenshots.