

**CSE 142, Summer 2008**  
**Programming Assignment #6: Baby Names (20 points)**  
**Due: Friday, August 8, 2008, 11:00 PM**

*Special thanks to Stanford lecturer Nick Parlante for the original concept of this assignment!*  
also see: [http://www.peggyorenstein.com/articles/2003\\_baby\\_names.html](http://www.peggyorenstein.com/articles/2003_baby_names.html)

This assignment focuses on reading input data from files. Turn in a file named `BabyNames.java`. You will also need `DrawingPanel.java`, which you used on previous assignments.

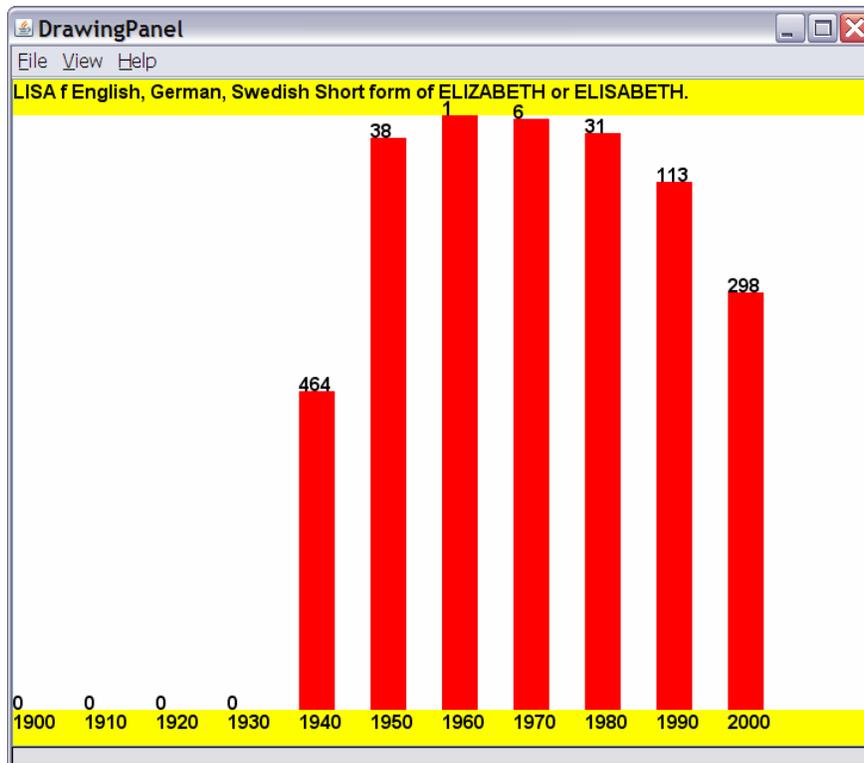
**Program Description:**

Every 10 years, the Social Security Administration gives data about the 1000 most popular boy and girl names for children born in the US. This data is provided on the web at <http://www.ssa.gov/OACT/babynames/>. Your task in this program is to prompt the user for a name, and then to display the name's meaning and popularity statistics about that name for each decade since 1900. You will display the data as console text and as a graphical bar chart on a `DrawingPanel`. The input data about names' popularity rankings and meanings comes from two input files provided on our web site.

Your program gives an introduction and then prompts the user for a first name. It reads a file searching for that name, case-insensitively (that is, you should find the name regardless of the capitalization the user uses when typing it). If the name is found in the file, you will print a line of statistics about that name's popularity in each decade. You will also look up that name's meaning in a second file and display information about its meaning.

This program graphs the popularity of a name in statistics recorded since the year 1900.

```
Type a name: lisa  
Lisa 0 0 0 0 464 38 1 6 31 113 298  
LISA f English, German, Swedish Short form of ELIZABETH or ELISABETH.
```



## Input Data and Files:

Your program will read data from two files:

- `names.txt`: popularity rankings for each name in each decade from 1900-2000
- `meanings.txt`: descriptions of the meanings of each name

Download these files from our web site, and save them to the same directory as your program.

Each line of `names.txt` contains a name, followed by the rank of that name in 1900, 1910, 1920, and so on. The default input file has 11 numbers per line, meaning that the last number represents the ranking in the year 2000. A rank of 1 was the most popular name that year, while a rank of 999 was not very popular. A rank of 0 means the name did not appear in the top 1000 that year at all. Here is a sample of the data:

```
Lionel 387 344 369 333 399 386 408 553 492 829 972
Lisa 0 0 0 0 464 38 1 6 31 113 298
Lise 0 0 0 0 0 997 0 0 0 0 0
Lisette 0 0 0 0 0 0 0 816 958 0 864
```

"Lionel" was #387 in 1900 and is slowly decreasing. "Lisa" first made the list in 1940 and peaked in 1960 at #1.

Once the user types a name, you should search each line of `names.txt` to see if it contains data about that name. If the name is found, you should output the line of data about the name to the console. You must then construct a `DrawingPanel` to graph the data. A description of the graphical output is on the next page. Your panel must exactly reproduce the window appearance of the examples for the same user input.

If the name is not found, output a "not found" message and not draw any data. No `DrawingPanel` should appear.

```
This program graphs the popularity of a name
in statistics recorded since the year 1900.
```

```
Type a name: zOIDberG
"zOIDberG" not found.
```

Though the input data shown above has 11 decades' worth of data, your program may not assume this. Your program should work properly with any number of decades of data (at least 1). Since there is a limit to the size of the `DrawingPanel`, you'd only be able to see data from roughly 12 decades, but your code should process as many decades' worth of data as it finds in the appropriate line of the file. Again, **do not assume that there will be exactly 11 decades when writing this program**. On the course website is a file named `names2.txt` with 8 decades of data to help you test this behavior.

If the name typed by the user is found in `names.txt`, you should also read `meanings.txt` to find the meaning of that name. The line containing the name's meaning should be printed to the console and also drawn on the `DrawingPanel`. Every name found in `names.txt` is also found in `meanings.txt`, so you do not need to worry about a name having rankings but no meaning data.

Each line of `meanings.txt` contains a name in upper case, followed by the name's meaning. For example:

```
LIONEL m French Pet form of LON
LISA f English, German, Swedish Short form of ELIZABETH or ELISABETH.
LISE f French French short form of ELISABETH
LISETTE f French French pet form of ELIZABETH
```

Note that even though the two input files contain different data, the task of searching for a name in `names.txt` is very similar to the task of searching for a name in `meanings.txt`; both contain the name as the first token of each line. Your code should take advantage of this fact and should avoid redundancy.

## Graphical Output:

The panel's overall size is 720x560 pixels. Its background is white. It has yellow filled rectangles along its top and bottom, each being 30 pixels tall and spanning across the entire panel, leaving an open area of 720x500 pixels in the middle. The line of data about the name's meaning appears on the panel at (0, 16).

Each decade is represented by a width of 60 pixels. The bottom yellow rectangle contains black labels for each decade, at  $y=546$ . For example, the text "1900" is at (0, 546) and "1910" is at (60, 546).

Starting at the same x-coordinate, a red bar shows the name ranking data over each decade. The bar is 30px thick (half as wide as each decade). The table at right shows the mapping between rankings and y-values of the tops of these bars. The y-values start at 30, and there is a vertical scaling factor of 2 between pixels and rankings, so you should divide a ranking by 2 when calculating its y-coordinate. For example, a ranking of 38 in 1950 results in a 30x481 bar occupying the pixels from (300, 49) through (329, 529).

Rank	Top y
1	30
2, 3	31
4, 5	32
...	...
998, 999	529
0	530

At the same coordinate as the top-left of each red bar, black text shows the name's rank for that decade. For example, Lisa was #38 in 1950, so "38" appears at (300, 49). A 0 rank means the name wasn't in the top 1000. No red bar should appear in such a case, and "0" should be drawn at the bottom at  $y=530$ . For example, Lisa's 0 in 1910 is drawn at (60, 530).

## Implementation Guidelines:

We suggest you begin with the text output and file processing, then any "fixed" graphical output, and then the red bars. The 0-ranking case is particularly tricky to draw, so you may want to do this last. (Hint: Treat rank 0 as a rank of 1000.)

Your program should work correctly regardless of the capitalization the user uses to type the name. If the user types "LiSa" or "lisa", you should find it even though the input files have it as "Lisa". and "LISA".

Draw text labels on the `DrawingPanel` using the `drawString` method of the `Graphics` object. Some text you'll want to write will be `ints`, but you can convert them into `Strings` using the `+` operator with an empty string. For an `int` named `n` with value 100, the expression `" " + n` yields the string "100". To draw this at (50, 120):

```
g.drawString(" " + n, 50, 120);
```

All text is drawn using bold "SansSerif" font, size 16. Set this using the `Graphics` object's `setFont` method:

```
g.setFont(new Font("SansSerif", Font.BOLD, 16));
```

## Stylistic Guidelines:

You should have the following **two class constants**. If the constant values are changed, your output should adapt.

- The **starting year** of the input data, as an integer (default of 1900)  
e.g. If you change the start year to 1825, the program should assume the data comes from 1825, 1835, etc.
- The **width** of each decade on the `DrawingPanel`, as an integer (default of 60)  
e.g. If you change the width to 50, each red decade bar is 50px apart and 25px thick. The panel's width should also adjust. The original width of 720 mentioned previously really comes from 60px per decade times 12 decades. But if you change the constant to 50, the panel's width becomes 600px.

We will be especially picky about redundancy. For full credit, your methods should obey these constraints:

- The `main` method should not draw on a `DrawingPanel`, nor read lines of input from a file (`nextLine`).
- The method that asks the user for a name must not also read lines of input from a file.
- Split the displaying of graphical data into at least two methods. For example, you could have one method to draw "fixed" graphics (yellow bars, decade labels) and another for graphics that come from the file (red bars, ranks).

Your methods should be well-structured and avoid redundancy, and your `main` method should be a concise summary of the overall program. Avoid "chaining," which is when many methods call each other without ever returning to `main`.

For this assignment you are limited to the language features in Chapters 1 through 6 of the textbook. In particular, **you are not allowed to use arrays on this assignment**. Follow past stylistic guidelines about indentation, line lengths, identifier names, and localizing variables, and commenting at the beginning of your program, at the start of each method, and on complex sections of code. For reference, our solution occupies ~100 lines and has 4 methods other than `main`.