EGR 222 — Software Engineering
Homework #3 Baby Names (Updated!)
Due: Tuesday, 10/4/16, 10 PM

Updated content high-lighted in purple. This assignment focuses on reading input files. You will need to create a file named BabyNames.java. You will also need DrawingPanel.java, which you used on HW2. All supplementary files are provided HERE.

Program Description:

The Social Security Administration has published the 1000 most popular boy and girl names for children born in the US for all years after 1879 (see http://www.ssa.gov/OACT/babynames/). For this project, you will prompt the user for a name and gender, and then display the name's meaning and popularity 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. Your program should give an introduction and then prompt the user for a first name and gender. It should then read the name rank data file searching for that name/gender combination, case-insensitively (that is, you should find the name regardless of the capitalization the user uses when typing it). If the name/gender combination is found in the file, your program should print a line of statistics about that combination's popularity in each decade, the name's meaning and display information about the name graphically.
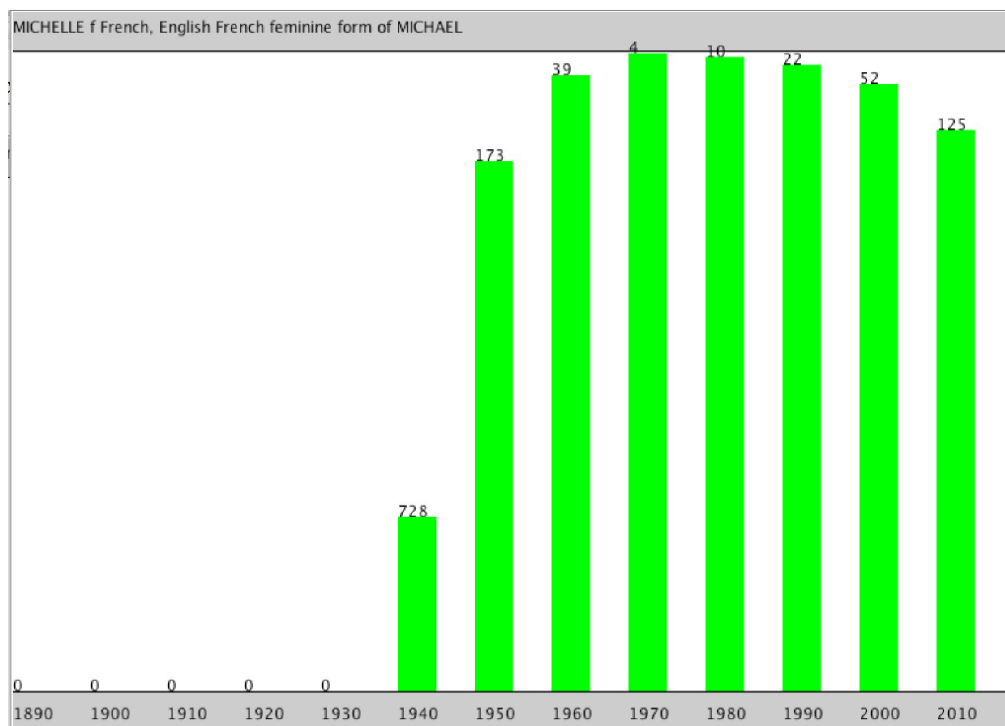
```
This program allows you to search through the
data from the Social Security Administration
to see how popular a particular name has been
since 1890.

Name: michelle
Gender (M or F): f
Michelle   F   0   0   0   0   0 728   173 39 4   10 22   52 125
MICHELLE f French, English French feminine form of MICHAEL
```

Input Data and Files:

Your program reads data from two files. Download them from our web site to the same folder as your program.

1. **names.txt**: popularity rankings for each name 1890-2010

Each line of names.txt contains a name followed by that name's rank in 1890, 1900, 1910, etc. The default file has 13 numbers/line, so the last represents the ranking in 2010. Rank #1 was the most popular that year, while rank #999 was not popular. Rank 0 means the name did not appear in the top 1000. For example:

```
Michelle    F   0   0   0   0   0 728   173 39 4   10 22   52 125
Michelle    M  0 0 0 0   0   0   0 0 736   897 0   0 0
Michial  M 0 0 0   0   0   0 987 0 0 0   0 0   0
```

"Michelle" as a female name first made the list in 1940 and peaked in 1970 at #4. It has been on a steady decline in popularity since. "Michial" only made the top-1000 in 1950. Notice that the spacing between numbers and other tokens varies. This won't be a problem if you use a Scanner to pull apart each line of input (versus, for example, searching for a string that has exactly one space between the name and the F/M gender notation).

Once the user types a name/gender combination, search each line of names.txt to see if it contains data for that combination. If it is found, output its data line to the console, then construct a DrawingPanel to graph the data (see Graphical Output requirement). Your code should not assume that the file is sorted alphabetically.

If the combination is not found, output a "not found" message. No DrawingPanel should appear.

```
This program allows you to search through the
data from the Social Security Administration
to see how popular a particular name has been
since 1890.

Name: zOIDberG
Gender (M or F): m
"zOIDberG" not found.
```

Though the data shown above has 13 decades' worth of rankings, 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 13 decades, but your code should process as many decades of data as it finds in the line. Do NOT assume that there will be exactly 13 decades when writing this program. HERE is a file named names2.txt with 8 decades. The graphical output for names2.txt has the start year as 1863 (THIS IS DIFFERENT START YEAR THAN names.txt) and 50 pixels width and 20 pixels legend height. See Stylistic Guideline on page 4 for more detail on start year, width, and height.

2. **meanings.txt**: descriptions of the meanings of each name

If the name/gender combination is found in names.txt, you should also read meanings.txt to find its meaning. The line containing the combination's meaning should be printed to the console and also drawn on the DrawingPanel. Every combination in names.txt is also in meanings.txt, so you

do not need to worry about a combination having rankings but no meaning data. Some combinations have long meanings that may stretch past the right edge of the DrawingPanel. Each line of meanings.txt contains a name in upper case, followed by a gender and meaning, as in:

```
MICHELLE f French, English French feminine form of MICHAEL
MICHELYNE f English (Modern) Pet form of MICHELLE
MICHI f Japanese Means "pathway" in Japanese.
MICHIAL m (no meaning found)
```

Though the two input files contain different data, the task of searching for a name/gender combination in names.txt is very similar to the task of searching for a combination in meanings.txt. Your code should take advantage of this fact and should avoid redundancy. You will be using several different Scanner objects for this program. You will have one Scanner that you use to read information from the console. You will use a different Scanner to read from each file. And because the input file is line-based, you should construct a different Scanner object for each line of the input file, as in section 6.3 of the book. You should write your code in such a way that you stop consuming lines of input once you find one that has the name you're searching for.

Graphical Output:

The panel's overall size is 780x560 pixels. Its background is white. It has light gray (Color.LIGHT_GRAY) filled rectangles along its top and bottom with a black line at their bottom and top, respectively. The two rectangles are each 30 pixels tall and span across the entire panel, leaving an open area of 780x500 pixels in the middle. The line of data about the name's meaning appears in the top gray rectangle at (0, 16).

Each decade is represented by a width of 60 pixels. The bottom light gray rectangle contains black labels for each decade, 8px from the bottom of the DrawingPanel. For example, with default constant values (see style guidelines), the text "1890" is at (0, 552) and "1910" is at (120, 552).

Starting at the same x-coordinate, a bar shows the name ranking data for each year. The bars are green. Bars are half as wide as each decade (30px). The table at right shows the mapping between rankings and y-values of the tops of bars. Y-values start at 30 (below the top gray rectangle), and there is a vertical scaling factor of 2 between pixels and rankings; divide a ranking by 2 to get its y-coordinate.

At the same coordinate as the top-left of each bar, black text shows the name's rank for that decade. For example, Michelle was #4 in 1970, so "4" appears at (480, 32). A 0 rank means the name was not in the top 1000. No bar should appear in such a case, and "0" should be drawn right above the bottom gray bar. For example, in the screenshot above, Michelle's 0 in 1920 is drawn at (180, 530).

Rank	Top y
1	30
2, 3	31
4, 5	32
6, 7	33
...	...
996, 997	528
998, 999	529
0	530

Implementation Guidelines:

We suggest you begin with the text output and file processing, then any "fixed" graphical output, and then the 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. To draw an int as text, you can convert it into a String using the + operator with an empty string. For example, for an int variable named n with value 100, the expression "" + n yields the String "100". To draw this at (50, 120), you could write: g.drawString("" + n, 50, 120);

Stylistic Guidelines:

You should have at least these **three class constants**. If the constant values are changed, your output should adapt.

- The **starting year** of the input data, as an integer (default of 1890) 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 green decade bar is 50px apart and 25px thick.
- The **height** of the legend rectangles, as an integer (default of 30) e.g. If you change the legend height to 20, the gray rectangles at top and bottom are 20px tall. The panel is 540px tall so that the open area in the middle is 500px tall. The decade labels are at y=532.

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

- The `main` method should <u>NOT</u> draw on a DrawingPanel, <u>NOR</u> read lines of input from a file (nextLine).
- The method that asks the user for a name must <u>NOT</u> also read lines of input from a file.
- <u>SPLIT</u> the displaying of graphical data into at least two methods. For example, you could have one method to draw "fixed" graphics (gray legend rectangles, etc.) and another for graphics that come from the file (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 <u>NOT</u> 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.