# CSE 190 D, Winter 2013
# Programming Assignment #5: Baby Names (18 points)
## Due: Monday, February 11, 2013, 11:30 PM

*Thanks to Nick Parlante from Stanford for the assignment concept.*

This assignment focuses on reading input files. Turn in a file named `BabyNames.java`.

## 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 then display the name's meaning and popularity as console output. The input data about names' popularity rankings and meanings comes from two input files provided on the course web site.

Your program should give an introduction and then prompt the user for a first name. It should then read the name rank data 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, your program should print a line of statistics about that name's popularity in each decade, the name's meaning and the year it was most popular and what that popularity was.

```
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
Michelle f 0 0 0 0 0 728 174 39 4 10 22 52 125
MICHELLE f French, English French feminine form of MICHAEL
Michelle was most popular in 1970 with a popularity of 4
```

## 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 0 728 174 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.

Once the user types a name, search each line of `names.txt` to see if it contains data for that name. If the name is found, output its data line to the console. If the file includes two lines with the same name, your program should use the first. For example, the data above shows that "Michelle" can be both a male and female name but your program should only display information about the female name since it happens to come first in the file. **Your code should not assume that the file is sorted alphabetically**.

If the name is not found, output a "not found" message and don't show any data.

```
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
"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). 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**. There is a file on the course website named `names2.txt` with 8 decades of data to help you test this behavior.

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

If the name is found in `names.txt`, you should also read `meanings.txt` to find its meaning. The line containing the name's meaning should be printed to the console. Every name in `names.txt` is also 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:

```
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 in `names.txt` is very similar to the task of searching for a name 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.

## Implementation Guidelines:

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"`.

## Stylistic Guidelines:

You should have at least one **class constant**. If the constant value is 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.

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

- The `main` method should not 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.

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 ~80 lines and has 4 methods other than `main`.