University of Washington
Computer Science & Engineering 142 (Computer Programming I), Summer 2005

Programming Assignment #6 (Baby Names)
Due: Tuesday, 8/2/2005, 6:00 PM

(The highlighted sections of this document represent extra credit functionality.)

Problem Description:

This assignment will give you practice with file processing in conjunction with all of the other constructs we have learned this quarter (loops, if/else, methods, and graphics). Turn in a class named `BabyNames` stored in a file named `BabyNames.java`. You will need to have both `Scanner.java` and `DrawingPanel.java` in the same directory as your program for it to compile properly. All of the files that you will need are available on the class web page.

Your task in this program is to prompt the user for a name, and then to display the meaning of that name and popularity statistics about that name for each decade since 1900. You will display both a text output of this data and a graphical line chart of this data on a `DrawingPanel`.

You will process a file with data obtained from the Social Security Administration. They provide a web site showing the distribution of names chosen for children over the last 100 years in the US. The site's URL is 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.

This data about name popularity by decade has been stored into a single text file named `names.txt`. On each line of tis file is a name, followed by the popularity rank of that name in 1900, 1910, 1920, ..., and 2000 (11 numbers). A rank of 1 was the most popular name that year, while a rank of 999 was not very popular. A 0 means the name did not appear in the top 1000 that year at all. The lines are in alphabetical order, although we will not depend on that.

Sample of data stored in `names.txt`

```
...
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
...
```

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 children.

A separate data file named `meanings.txt` has been created that contains a large list of names and their approximate historic meanings. Every name in `names.txt` has a corresponding entry in `meanings.txt`, but some of the entries contain only a "?" question mark to indicate that the name's meaning is not known. The lines appear in alphabetical order.

Sample of data stored in `meanings.txt`

```
...
Frederick Peace Ruler
Fresa Fortunate
Gabrielle Godly
Gail Joy
Garda Protector
...
```

Nick Parlante, the Stanford instructor who conceived this assignment, suggests the following questions to explore and recommends the original article that gave him the idea (http://www.farfilm.com/peggy/articles/wherehaveallthelisas.htm):

- Why is Rock popular in 1950 and Trinity in 2000?
- Type in your grandparents' names. Names like Ethel and Mildred and Clarence sound old fashioned – and they are! But wait long enough and the come back – Emma! Hannah!
- 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 1970s. A reaction to the 1960s, 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. I was thinking Adolf would vanish in the mid 30s but it seems to vanish 10 years before that. In any case, Adolf is tricky, since there are a bunch of variant names like Adolph.

Program Behavior:

Your program is to give an introduction and then prompt the user for a name to display. Then it will read through the data file searching for that name. If the name is not in the file `names.txt`, you should simply output that it was not found, and not print or draw any statistical data.

Example log of execution #1, for a name not contained in the file (user input underlined)

```
This program allows you to search through the
data from the Social Security Administration
to see how popular a particular name has been
since 1900.

Name? Zoidberg

The name "Zoidberg" was not found.
```

However, if the name is found in the file `names.txt`, you should print the name, its meaning (as found in the file `meanings.txt`), and the statistics about that name's popularity from 1900 to 2000:

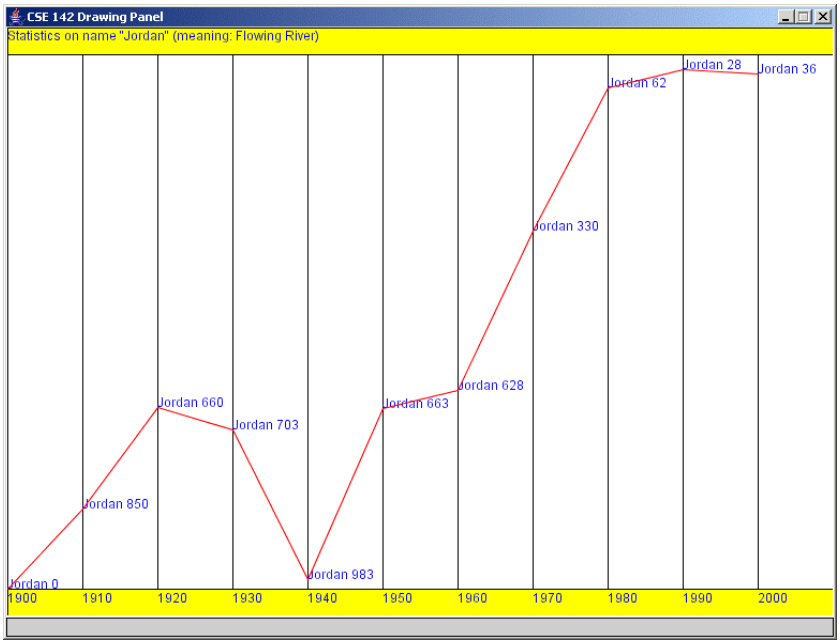
Example log of execution #2, for a name that is contained in the file (user input underlined)

```
This program allows you to search through the
data from the Social Security Administration
to see how popular a particular name has been
since 1900.

Name? Jordan

Statistics on name "Jordan" (meaning: Flowing River)
    1900: 0
    1910: 850
    1920: 660
    1930: 703
    1940: 983
    1950: 663
    1960: 628
    1970: 330
    1980: 62
    1990: 28
    2000: 36
```

If the name is found, you must also construct a `DrawingPanel` to graph the data. No `DrawingPanel` should appear if the name is not found, so do not construct the `DrawingPanel` object unless you verify that the name is in the file. If the name is found, you are to exactly reproduce this window appearance, given the same user input.

Graphical Output:

Here is a screenshot of your expected graphical output and a detailed description of the window's appearance:



- The panel's overall height should always be 550 pixels.
- The panel's overall width should be 70 pixels times the number of decades being shown (the default number of decades is 11, so the default panel width is 770, but if the number of decades changes in the code, the panel's width should adjust itself accordingly.)
- The panel's background color is white.
- The panel has yellow rectangles with black borders along its top and bottom. Each rectangle is 25 pixels tall and spans across the entire `DrawingPanel`, leaving an area 500 pixels tall in the middle.
  The top rectangle contains a blue text String at position (0, 12) stating the name and its meaning, such as `Statistics on name "Jordan" (meaning: Flowing River)`

  The bottom yellow rectangle contains blue text labels for each decade. For example, the text "1900" in the picture has coordinates (0, 550) while the text "1910" has the coordinates (70, 550). You will want to write a loop to draw these various labels, especially since your program has to adapt properly if the constants are changed.
- The middle section of the panel is divided into sections of equal width to represent the different decades for which we have data. By default this makes 11 sections, 1 for each of our 11 decades from 1900 through 2000. These particular sections have a horizontal width of 70 pixels each. The 11 sections are separated by black vertical lines.
- A red line connects the data about the name's ranking over each decade. As noted earlier, the panel will always be 550 pixels high with the upper and lower 25 pixels not part of the plot area. That leaves exactly 500 pixels in the central area for plotting these values.

  The numbers range from 1 to 1000, so each pixel will represent two different rankings. Thus, a rank of 1 or 2 should be drawn at a y-coordinate of 25. A rank of 3 or 4 should be drawn at a y-coordinate of 26. A rank of 5 or 6 should be drawn at a y-coordinate of 27. And so on, up to ranks of 999 and 1000, which should be drawn at a y-coordinate of 524.

  A rank of 0 (which means the name didn't appear in the top 1000 at all) **is a special case and should be drawn at the very bottom of the plot range, at a y-coordinate of 525.**
- To the right of each decade's line, a blue String shows the name followed by a space followed by its rank for that decade. For example, the String "Sam 58" appears to the right of the line for 1900, because Sam had rank 58 in 1900. The text appears just to the right and just above the point you are plotting. In other words, you can use the same coordinates for drawing the String that you use for drawing the line for that decade.

Implementation Guidelines:

You will need to have the files `names.txt` and `meanings.txt` in the same directory as your program for Java to find them. If you are using DrJava, you will have to put these two files in the same folder as the DrJava program, or you will have to use a fully-qualified absolute path file name as described in section 6.2.2 of the book.

To draw the various text labels on the `DrawingPanel`, you will need to know a new drawing method of the Graphics object:

- `drawString(String s, int x, int y)`
  Draws the given text starting with its bottom-left corner at position (x, y).
  Example: `g.drawString("Test 1 2 3", 50, 80);`

Some of the text labels you'll want to write will be stored as ints, but you'll need to convert them into Strings. There are two ways to convert an int into the equivalent String (e.g., to turn an int value like 1900 into the String "1900").

- You can call the method `String.valueOf` passing it the int and it will return a corresponding String.
  Example:
```
int x = 1900;
String s = String.valueOf(x);    // now s stores "1900"
```

- Or you can concatenate the int with an empty String using the + operator.
  Example:
```
int x = 1900;
String s = "" + x;               // now s stores "1900"
```

You should ignore case when comparing the name typed by the user with the names in the input file. For example, if the user asks you to search for "SAM" or "sam", you should find it even though the input file has it as "Sam". The name that is displayed on the console and on the `DrawingPanel` should appear exactly as the user typed it, including the casing.

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 the input file. And because the input file is line-based, you should construct a different Scanner object for each line of the input file, as explained in section 6.3 of the book. You should write your code in such a way that you stop reading lines of input once you find one that has the name you're searching for.

Extra Credit:

All features mentioned previously that are related to name meanings (all highlighted portions of this document) are extra credit. You can earn a full 20 / 20 score even if you completely disregard the name meanings. You may omit the (meaning: ...) text from your text and graphical output and not even open the `meanings.txt` file.

If you implement the name meanings data correctly, including printing the name's meaning as text and displaying it atop the `DrawingPanel`, you will receive **+2 points extra credit**. No homework assignment's score may go above 100%, so you cannot exceed an overall score of 20 / 20 for this program. But doing the extra credit may give you a buffer against other potential deductions. (Plus, it might be fun!)

Stylistic Guidelines:

In terms of style points, we will be grading on your appropriate use of control structures like loops and if/else statements, your ability to avoid redundancy and your ability to break down the overall problem into methods that each solve part of the overall problem. No one method should be overly long. The `main` method in particular should not perform drawing operations on a `DrawingPanel`, nor should it directly read lines from files; perform these tasks in other methods. You should be able to come up with at least two different methods other than `main` that each perform some nontrivial part of the problem. Some example methods that would be considered satisfactory are:

- a method to print a welcome message, ask the user to input a name, and return that name
- a method to scan the `names.txt` file and return the line of popularity data about a particular name
- a method to display an appropriate `DrawingPanel`, given a name and the String of data about that name
- a method to scan the `meanings.txt` file and retrieve a particular name's meaning (if you do the extra credit)

You should use at least the following 3 global constants in your program. It should be possible to change these values and have your program adapt appropriately. You can introduce other class constants if you want to, in addition to the three required constants.

- the number of decades (11)
- the starting year (1900)
- the horizontal width per decade (70)

You can make sure your program works properly by changing the number of decades to 9, the start year to 1920 and the horizontal width to 90 and changing the file name to names2.txt (which has just 9 decades worth of data). On the course web site is a sample output for the name “Ethel” with only 9 decades' worth of visible data. Note that after changing the starting year and number of decades, the ending year will not necessarily be 2000, so your code should not rely on this.

You are required to properly indent your code and will lose points if you make significant indentation mistakes. See section 2.5.3 of the book for an explanation and examples of proper indentation. You should also use white space to make your program more readable, such as between operators and their operands, between parameters, and blank lines between groups of statements or methods.

Give meaningful names to methods and variables in your code. Follow Java's naming standards about the format of ClassNames, methodAndVariableNames, and CONSTANT_NAMES. Localize variables whenever possible -- that is, declare them in the smallest scope in which they are needed.

Include a comment at the beginning of your program with basic information and a description of the program and include a comment at the start of each method. Since this program has longer methods than past programs, put brief comments inside the methods explaining the more complex sections of code.

Submission and Grading:

Name your file `BabyNames.java` and it them in electronically from the "Assignments" link on the course web page. You do not have to turn in `Scanner.java` or `DrawingPanel.java`. This assignment is worth 20 points total.

Part of your program's score will come from its "external correctness." External correctness measures whether your log of execution matches exactly what is expected, including identical prompting for user input, identical console output, and identical `DrawingPanel` graphical output.

The rest of your program's score will come from its "internal correctness." Internal correctness measures whether your source code follows the stylistic guidelines specified in this document. This includes using `while` or `do/while` loops to capture indefinite repetition, using `for` loops to capture definite repetition, representing the structure and redundancy of the program using appropriate parameterized static methods with return values as needed, using global constants to replace otherwise "magic number" values, commenting, naming identifiers, and indentation of your source code.