Project 3: The Name Search Class

Purpose:

According to the 1990 census, these are some of the most popular female first names in the United States. Write them—or create your own list—to a file called names.txt.

```
MARY
PATRICIA
LINDA
BARBARA
ELIZABETH
JENNIFER
MARIA
SUSAN
MARGARET
DOROTHY
LISA
NANCY
KAREN
BETTY
HELEN
SANDRA
DONNA
CAROL
RUTH
SHARON
MICHELLE
LAURA
SARAH
KIMBERLY
DEBORAH
JESSICA
SHIRLEY
CYNTHIA
ANGELA
MELISSA
BRENDA
AMY
ANNA
REBECCA
VIRGINIA
KATHLEEN
PAMELA
MARTHA
DEBRA
```

AMANDA
STEPHANIE
CAROLYN
CHRISTINE
MARIE
JANET
CATHERINE
FRANCES
ANN
JOYCE
DIANE
ALICE
JULIE
HEATHER
TERESA
DORIS
GLORIA
EVELYN
JEAN
CHERYL
MILDRED
KATHERINE
JOAN
ASHLEY
JUDITH
ROSE

Deliverables, Requirements and Timeline

Create a PopularNames Class that can:

- Read a list of names from a test file into a Name array.
- Perform a binary search from the Name array to pull out a particular name.
- Sort the Name array in ascending order.

Write a PopularNames class that reads these names into a String array. Use the Selection Sort—see Listing 6.8, pages 188-191 of your book. Or, use the Quicksort algorithm—see Listing 23.7, pages 754 and 755—to sort the names in ascending order.

Create a NameSearch Class that can:

- Ask the user to enter a name for search.
- Call the PopularNames Class to perform a search.
- Return the result.
- Ask the user for another name.
- Exit the program.

Here is a program fragment of PopularNames.java:

```
import java.io.*;

/**
   The PopularNames class reads a list of names from a file into an array.
   It provides a search method that can be used to search for a particular name
   in the array.
*/

public class PopularNames
{
   private String[] names;

   /**
      Constructor
      @param filename Name of the file to read
                      census names from.
      @throws IOException When a file I/O error occurs.
   */
   public PopularNames(String filename) throws IOException
   {
      // Create the file stream objects.
      FileReader freader = new FileReader(filename);
      BufferedReader inFile = new BufferedReader(freader);

      // Get the number of lines in the file.

      // Create an array large enough to hold all the
      // Strings in the file.

      // Read the names from the file into the array.

      // Close the file.
      inFile.close();

      // Sort the names.
      quickSort();
   }

   /**
      The search method performs a binary search on the names array. If the search
      value is found, its array subscript is returned. Otherwise -1 is returned
      indicating the search value was not found in the array.
      @param value The string to search for.
      @return The subscript of the name if it was found;
              -1 otherwise.
   */
```

```
public int search(String value)
{
  int first;       // First names element
  int last;        // Last names element
  int middle;      // Mid point of search
  int position;    // Position of search value
  boolean found;   // Flag

  // Set the inital values.

  // Search for the value.

  // Return the position of the item, or -1
  // if it was not found.
}

/**
  The numLines method counts the number of lines
  in a text file.
  @param filename Name of the file to read.
  @return The number of lines in the file.
  @throws IOException When a file I/O error occurs.
*/
private int numLines(String filename) throws IOException
{
  // Create the file stream objects.

  // Initialize a counter.

  // Read the first line from the file.

  // Read the remaining contents and count
  // the lines that are read.

  // Close the file.

  // Return the number of lines.
}

/**
  The quickSort method calls the doQuickSort method
  to sort the names array.
*/
  private void quickSort()
{
```

```
   doQuickSort(names, 0, names.length - 1);
}

/**
   The doQuickSort method uses the QuickSort algorithm
   to sort an array of strings.
   @param array The array to sort.
   @param start The starting subscript of the list to sort
   @param end The ending subscript of the list to sort
*/

private void doQuickSort(String array[], int start, int end)
{
   int pivotPoint;

   if (start < end)
   {
      // Get the pivot point.
      pivotPoint = partition(array, start, end);

      // Sort the first sub list.
      doQuickSort(array, start, pivotPoint - 1);

      // Sort the second sub list.
      doQuickSort(array, pivotPoint + 1, end);
   }
}

/**
   The partiton method selects a pivot value in an array and arranges the array
   into two sub lists. All the values less than the pivot will be stored in the left
   sub list and all the values greater than or equal to the pivot will be stored in
   the right sub list.
   @param array The array to partition.
   @param start The starting subscript of the area to partition.
   @param end The ending subscript of the area to partition.
   @return The subscript of the pivot value.
*/

private int partition(String array[],
                      int start, int end)
{
   String pivotValue; // To hold the pivot value
   int endOfLeftList; // Last element in the left sub list.
   int mid;           // To hold the mid-point subscript
```

```
      // Find the subscript of the middle element.
      // This will be our pivot value.

      // Swap the middle element with the first element.
      // This moves the pivot value to the start of the list.

      // Save the pivot value for comparisons.

      // For now, the end of the left sub list is
      // the first element.

      // Scan the entire list and move any values that
      // are less than the pivot value to the left
      // sub list.

      // Move the pivot value to end of the
      // left sub list.

      // Return the subscript of the pivot value.
   }

   /**
      The swap method swaps the contents of two elements
      in an array of String objects.
      @param The array containing the two elements.
      @param a The subscript of the first element.
      @param b The subscript of the second element.
   */

   private void swap(String[] array, int a, int b) {
   }
}
```

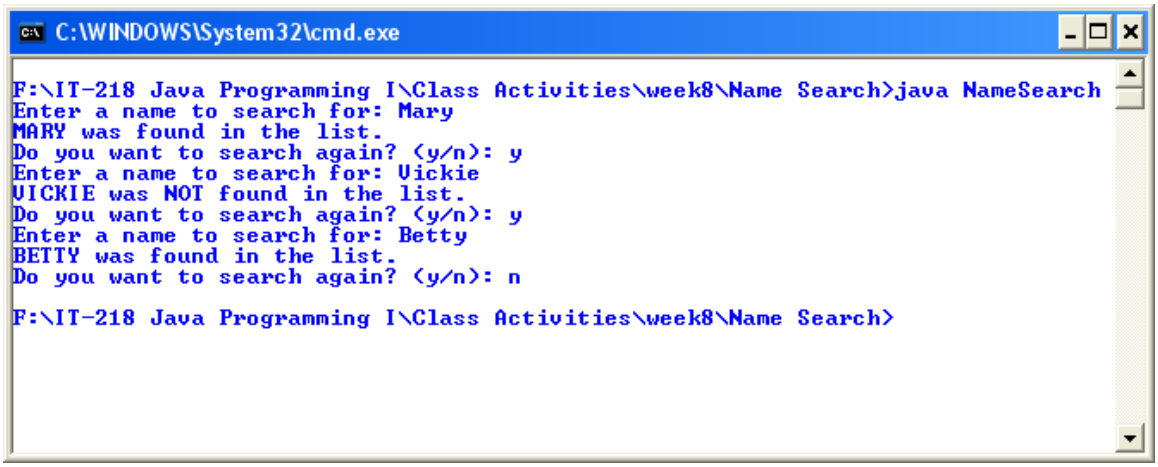
Then, write another NameSerach class to allow the user to search for a name in the array. Use the binary search algorithm—see Listing 6.6, pages 186-188—to perform the search.

Here is a program fragment of NameSearch.java:

```
import java.util.Scanner;
import java.io.IOException;

/**
   This program lets the user search for names among the 65 most popular females
   names in the 1990 US census.
*/
```

```
public class NameSearch
{
  public static void main(String[] args) throws IOException
  {
    String input;  // To hold keyboard input

    // Create the Scanner object for keyboard input.
    Scanner keyboard = new Scanner(System.in);

    // Create a PopularNames object with the census names.
    PopularNames names = new PopularNames("names.txt");

    // start a do-while loop
    {
      // Get a name to search for.

      // Search for the name.

      // Display the search results.
    }
  }
}
```

Here is a sample output:



Submit your project on a floppy disk or CD, or print out the source code with screen shots.

Assigned and Due Date:
Assigned: Unit 8
Due: Unit 9