LBYCP12

Data Structures and Algorithm Analysis Laboratory

Laboratory Activity 6

NameSurfer

By

Chan, Patrick Matthew J, LBYCP12-EQ1

INTRODUCTION

In this activity, students are to integrate their knowledge in File I/O, using a database class to collect entries, adding a GCanvas to a program, using acm.graphics, using lists, and manipulating arrays to create a program that graphs the popularity of a given name based on a given data of rankings per decade. Besides that, this activity also introduces a number of new concepts. One of these concepts is the usage of interactors like JLabels, JFields, and JButtons, and the process of adding action listeners so that the program can properly respond to user interactions with these objects. Next, is the concept of resizable displays, in which, the display can respond accordingly whenever the user resizes the window itself. Lastly, a new data structure, Queue, was also introduced, and was to be used to correctly display the graphs of the inputted names in order.

OBJECTIVES

- To integrate past learned concepts into a new Java program
- To learn about the concept of interactors and action listeners
- To be able to create a program that also updates itself whenever the application window is resized
- To implement a Queue data structure
- To utilize and apply a Queue data structure for the sequencing of user inputs

MATERIALS

1. Java SE Development Kit (JDK) 8
2. NetBeans integrated development environment (IDE)
3. `acm.jar` by the ACM Java Task Force
4. Provided starter kit for the NameSurfer Assignment

PROCEDURE

1. Write a simple implementation for the Queue data structure so that it can be used in the code of the application.
2. Add the necessary interactors to the program (JLabel, JField, JButtons).

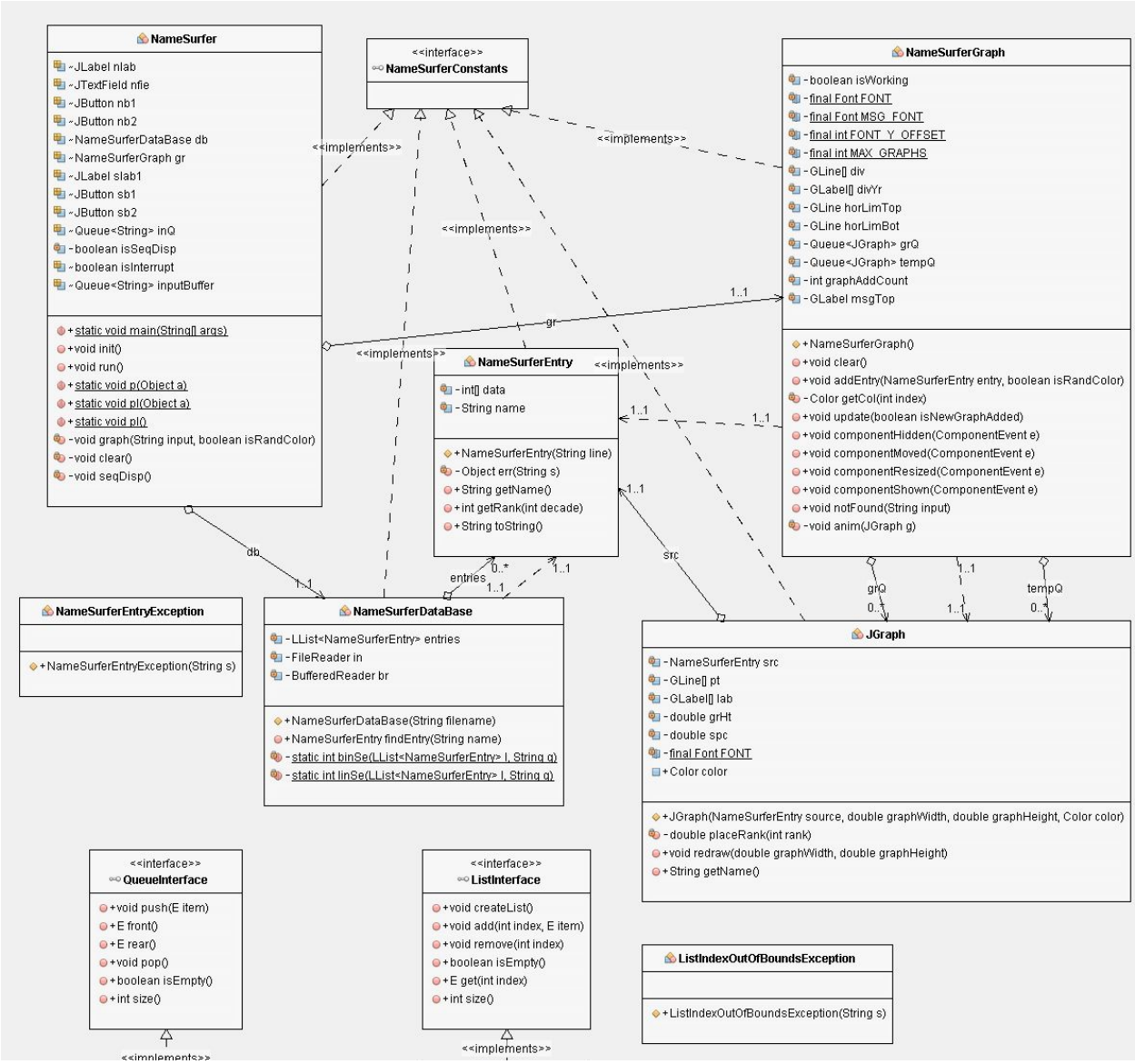
3. Implement the actionPerformed method for each of the interactors.
4. Temporarily set NameSurfer class to extend ConsoleProgram instead of Program in order to test Interactors' functionalities.
5. Test the interactors.
6. Get the starter file for the NameSurferEntry class.
7. Implement the given methods
8. Implement toString method to help with debugging and testing later
9. Use a test entry to test your implementation, and check if the initializer String line is properly parsed and split into the correct variables.
10. Test this out with the interactors in the main class.
11. Get the starter file for the NameSurferDatabase class.
12. Methods should be implemented with the following conditions:
    a. A constructor takes the name of a data file and uses that to read in the entire set of data from the file into internal data structures that allow the class to keep track of all the records as a database.
    b. A findEntry method that takes a name, looks it up in the stored database (note that your program should not be case sensitive regarding the name), and returns the NameSurferEntry for that name, or null if that name does not appear.
13. Add a line of code or two to the NameSurfer program so that it creates the NameSurferDataBase and then change the code for the button handlers so that clicking the "Graph" button looks up the current name in the database and then displays the corresponding entry (using its toString method).
14. Use this to test your implementation of the NameSurferEntry class.
15. Get the starter file for the NameSurferGraph class.
16. Note the following:
    a. This class extends GCanvas, which means that every NameSurferGraph object is not only a GCanvas but also an instance of all the superclasses of the GCanvas class. GCanvas is a subclass of Component in the standard java.awt package and therefore is part of the hierarchy of classes that can be added to the display area of a program. Moreover, it means we can call any of the GCanvas methods, such as adding or removing GObjects from the display, from within NameSurferGraph.

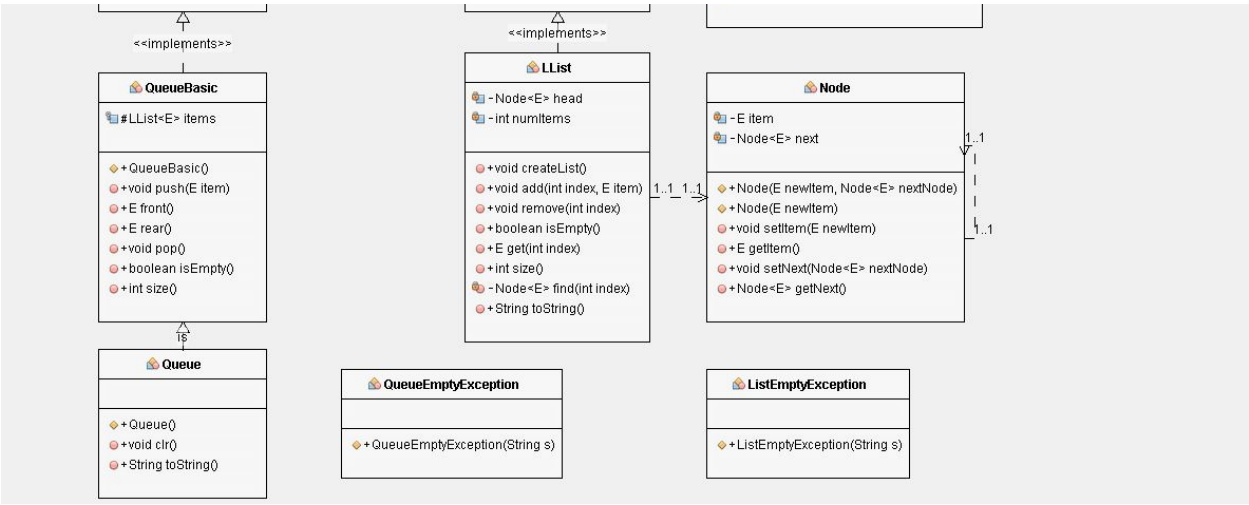
    b. The starter file includes a tiny bit of code that monitors the size of the window and calls update() whenever the size changes. This code requires only a couple of lines to implement.
17. Change the main class to extend the Program class instead.
18. Create an instance of NameSurferGraph in the main class, and add this to the display.
19. Write the code to create the background grid for the graph, using GLine and GLabel to the appropriate positions, which consists of:
    a. The vertical line separating each decade,
    b. The horizontal lines that provide space for the top
    c. Bottom borders (which are there to ensure that the text labels stay within the window bounds),
    d. The labels for the decades.
20. Implement addEntry and clear methods, such that neither of these actually changes the display. Make it so, that to make changes in the display, you need to call update, which deletes any existing GObjects from the canvas and then assembles everything back up again.
21. Take note of the following as you implement addEntry and clear:
    a. To make the data easier to read, the lines on the graph are shown in different colors. In the sample applet on the web site, the first data entry is plotted in blue, the second in red, the third in purple, and the fourth in black. After that, the colors cycle around again through the same sequence. You can choose the colors as you see fit.
    b. The fact that rank 1 is at the top of the window and rank 1000 is at the bottom means that it can sometimes seem confusing that rank 0—which means that the name does not appear in the top 1000 values for that year—appears at the bottom. To reduce the apparent discontinuity between rank 1 and rank 0, the entries for names that are absent from the data for a decade are listed with an asterisk instead of a numeric rank.
22. Find way(s) to utilize the Queue data structure, such as in:
    a. Removing the oldest entry when the display is cluttered with too much names

b. Adding an option for the user to have a sequential display of the names inputted, one by one, and in order
c. Having a buffer when the user inputs too quickly, so that these inputs are not lost when the program is still in the middle of finishing a process

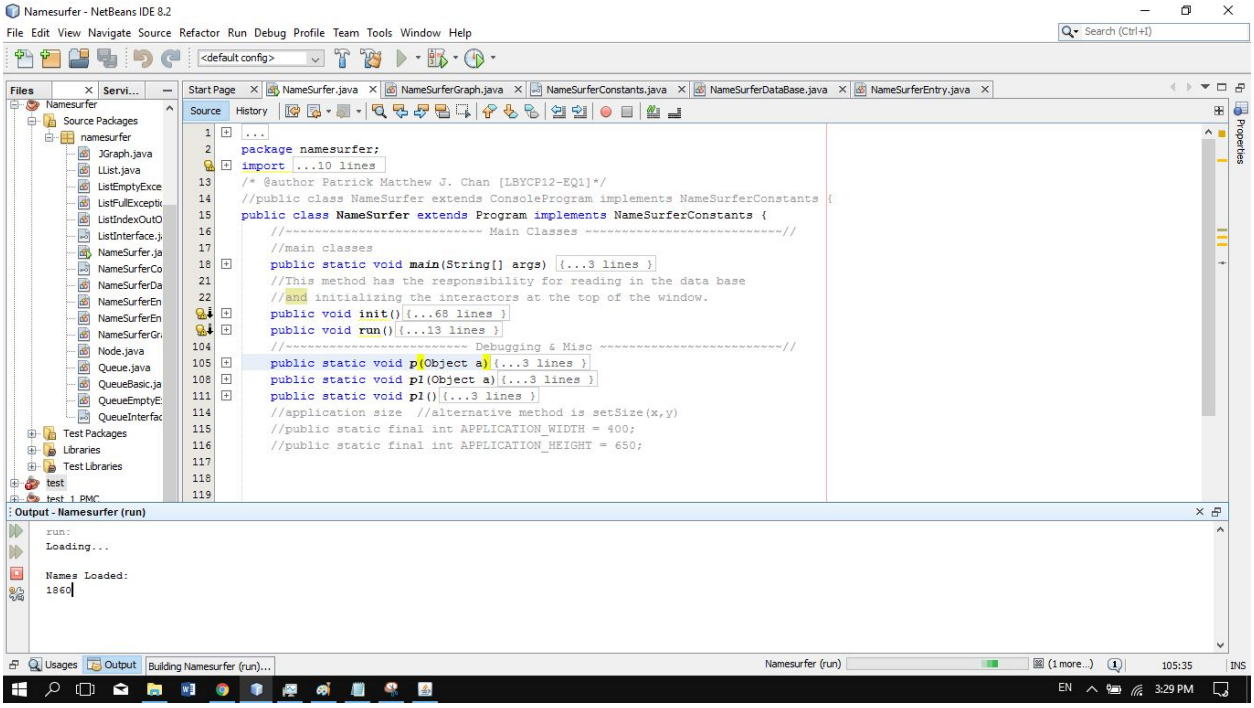
RESULTS AND DISCUSSION

The UML:

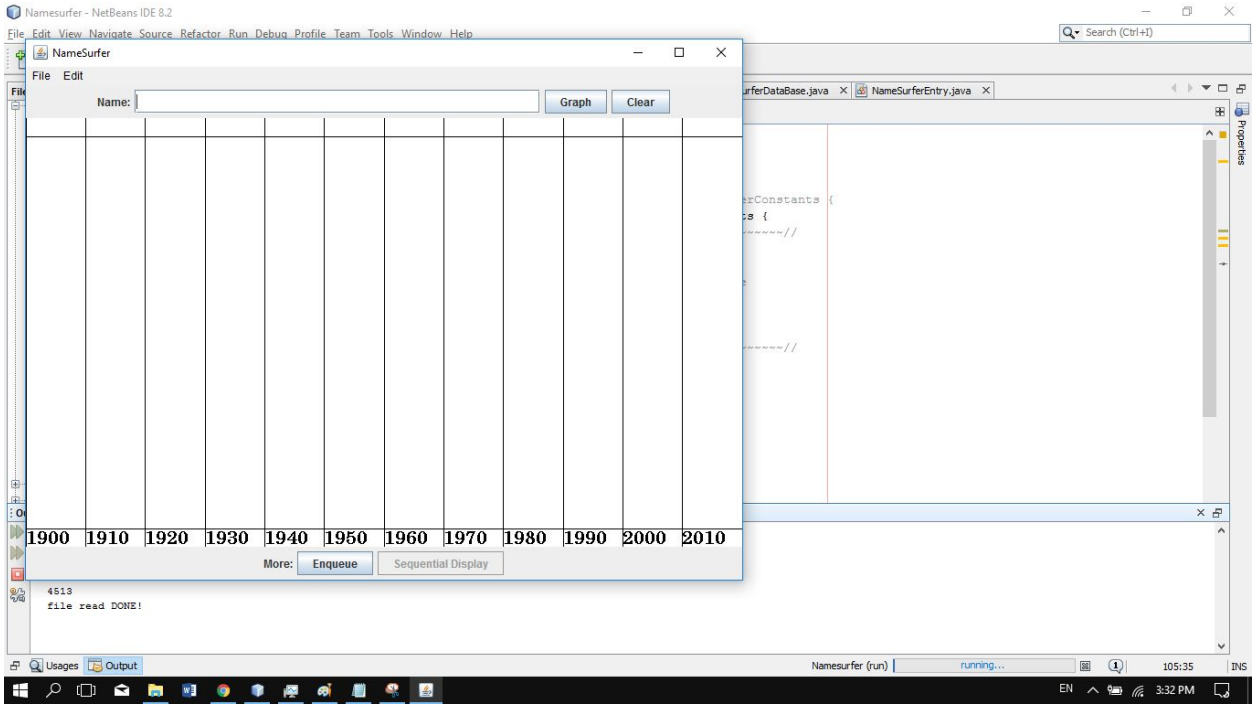




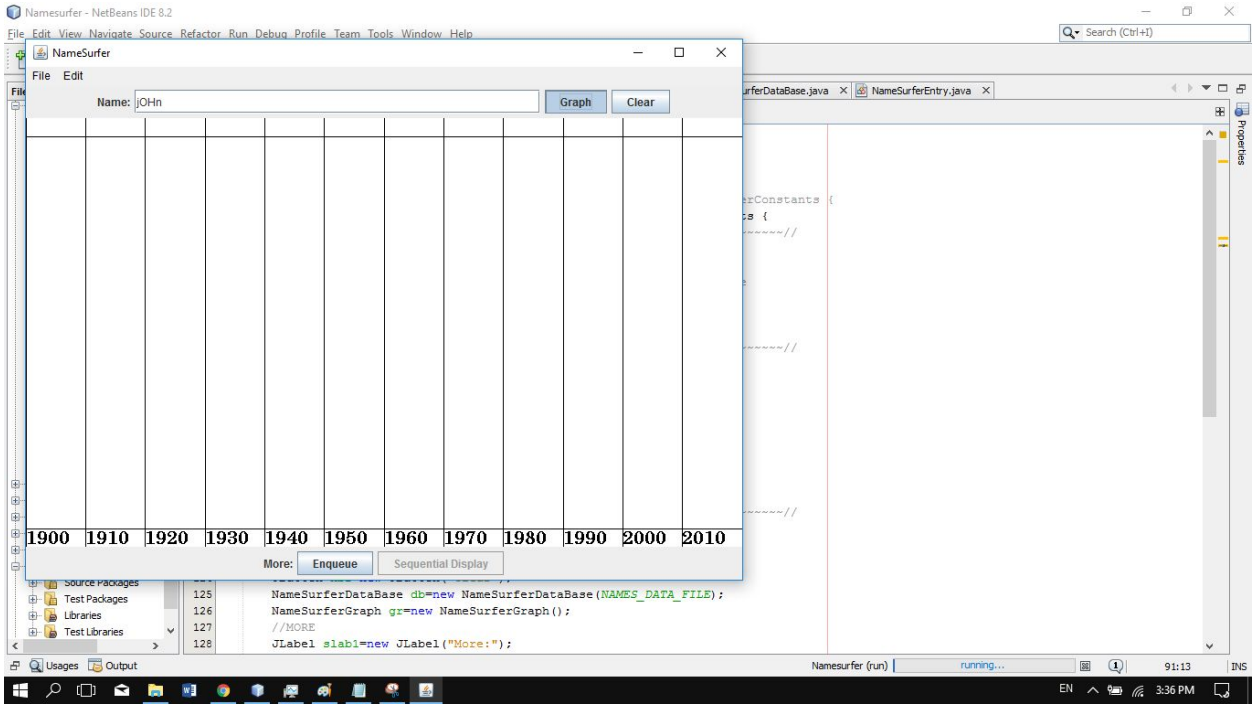
Screenshots:



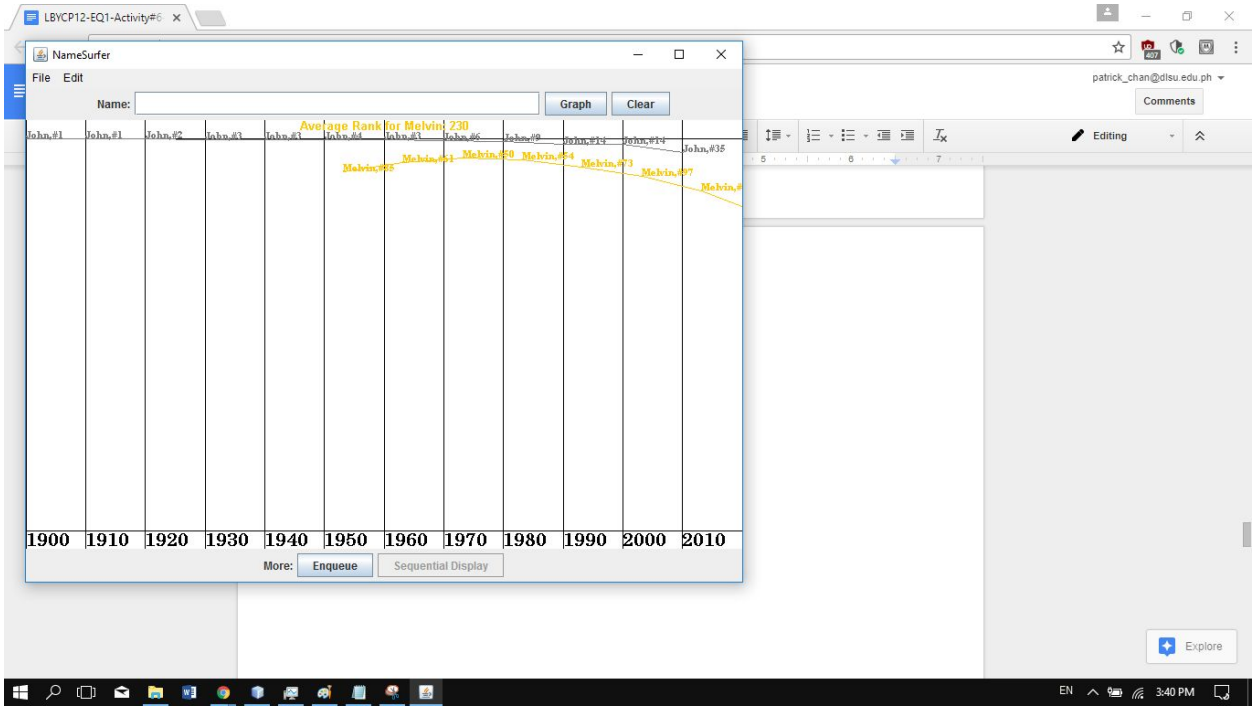
Program starts by showing the amount of names loaded, to show that the program is running, in case of long wait times for slower computers.



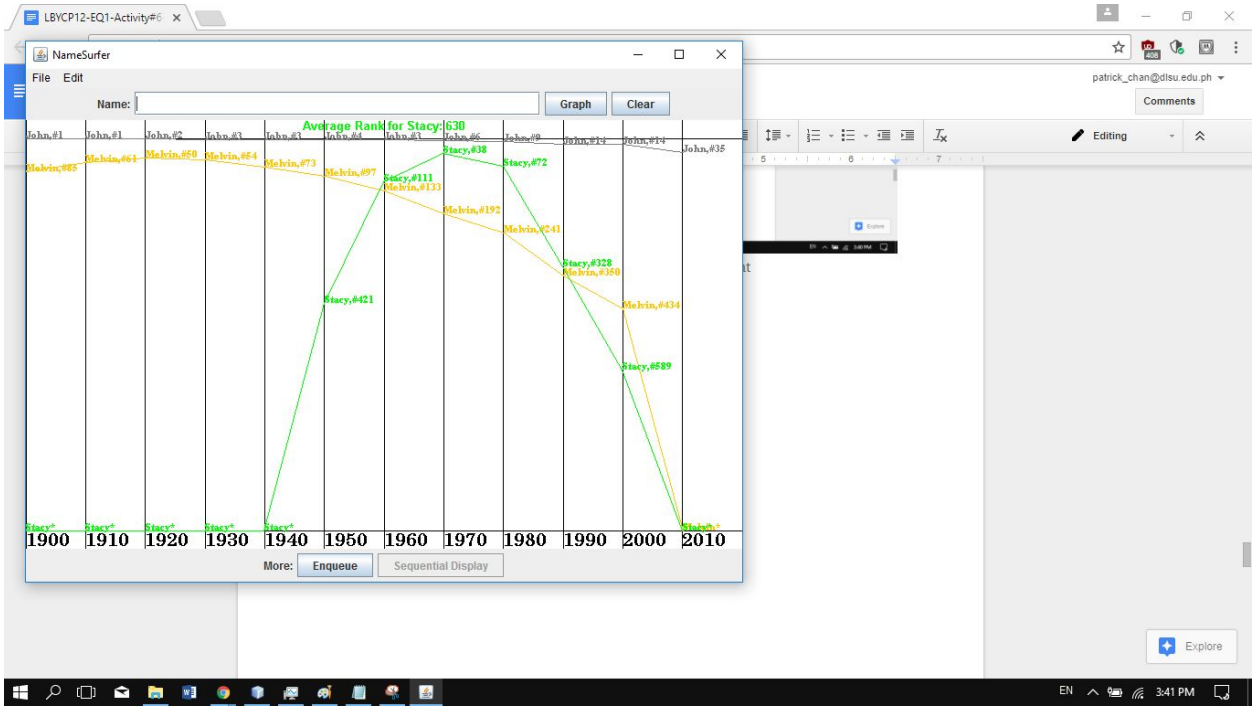
Starting Display, showcasing the different interactors and the empty graph.



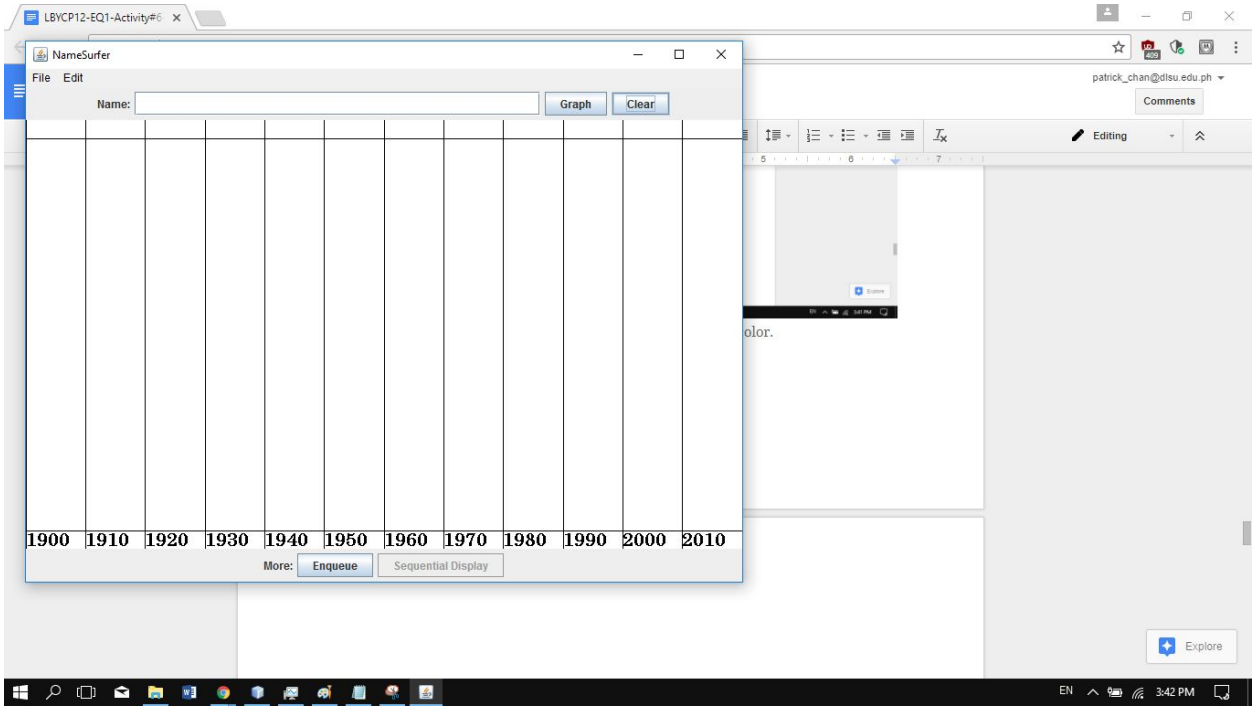
Case insensitive for inputs



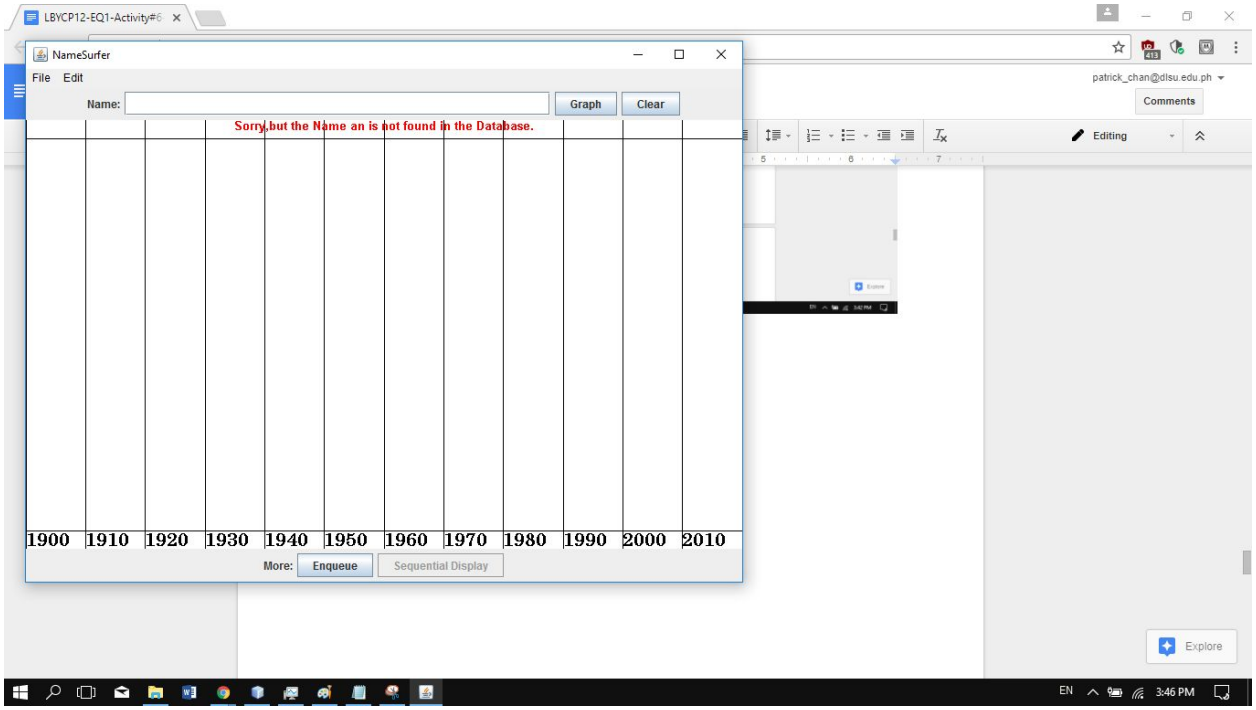
Animation: new graph enters from the right. Any inputs during this animation are noted and are displayed afterwards in order.



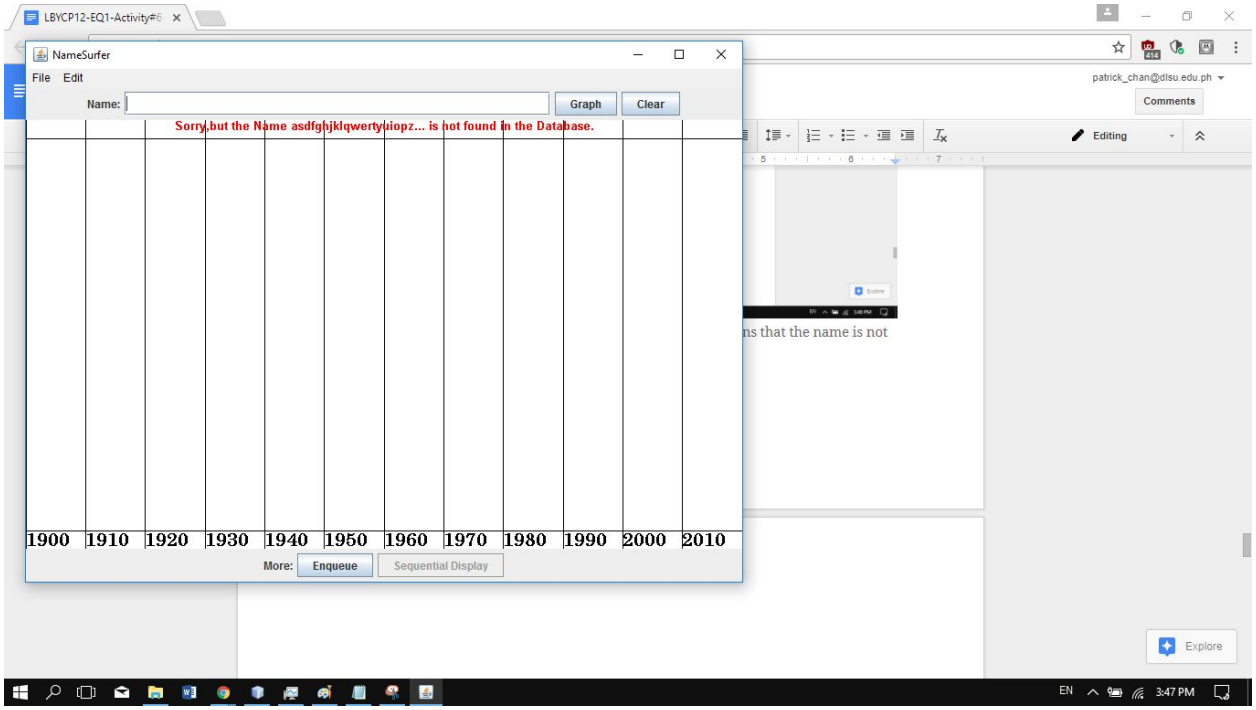
The next graph is always assigned a different color. The average rank is also displayed with the same color as its graph.



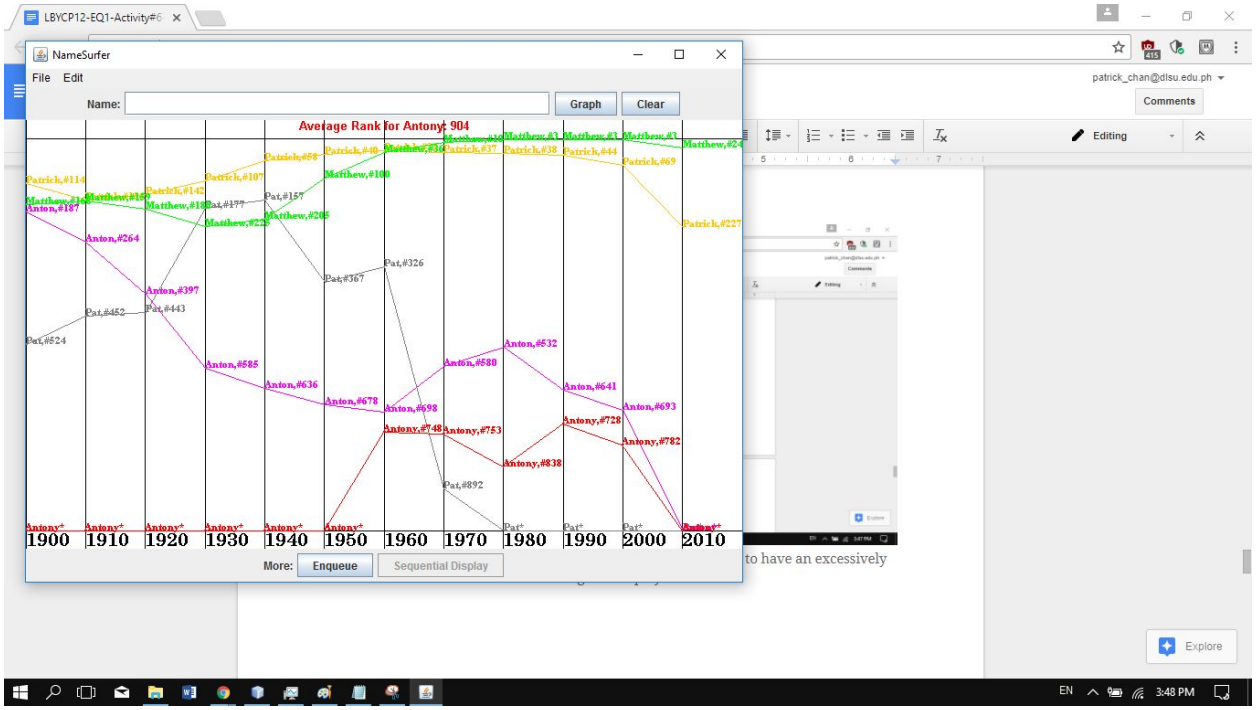
Clear button clears the graph display



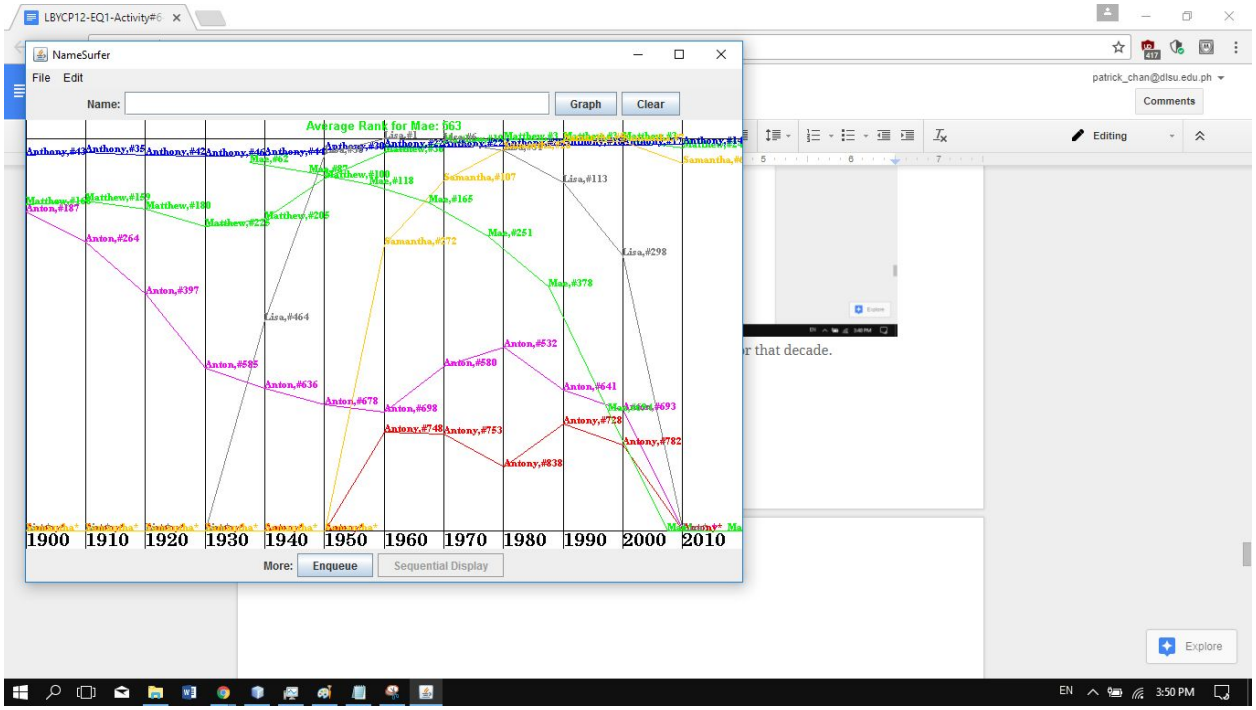
Prints a message if name is not found in the database. (This means that the name is not common enough)



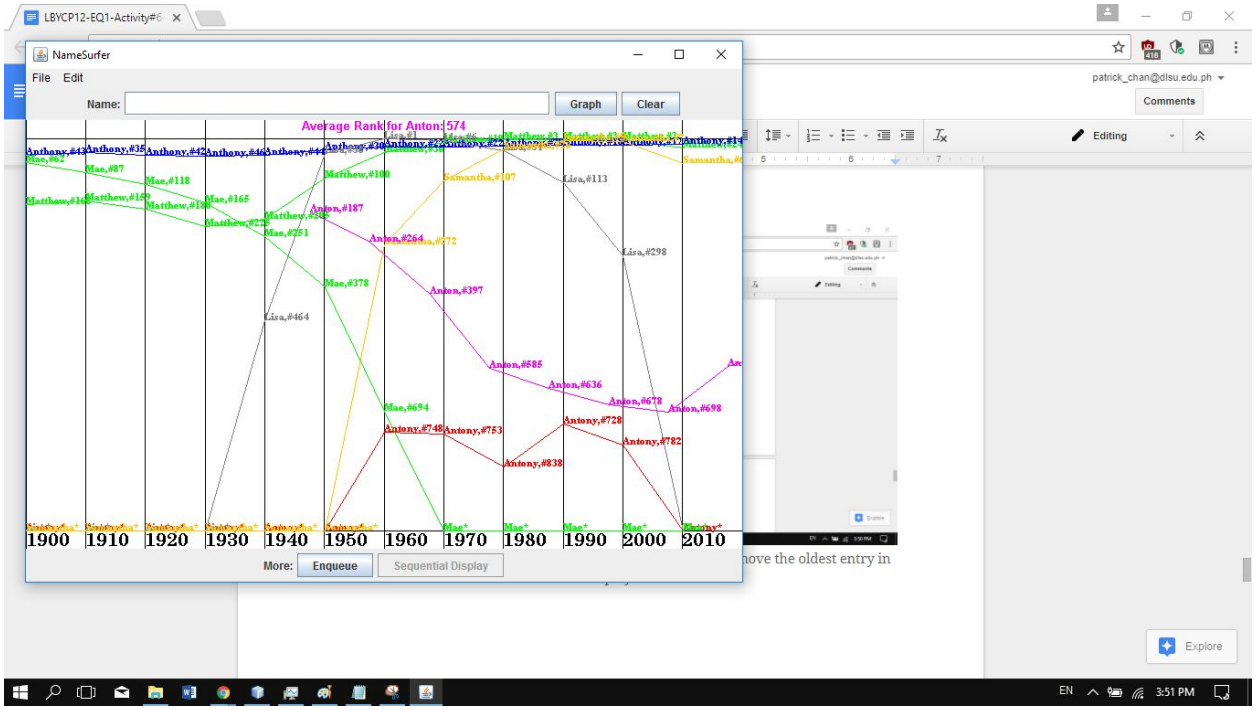
Long names, if not found, are trimmed in the message, so as not to have an excessively long line display.



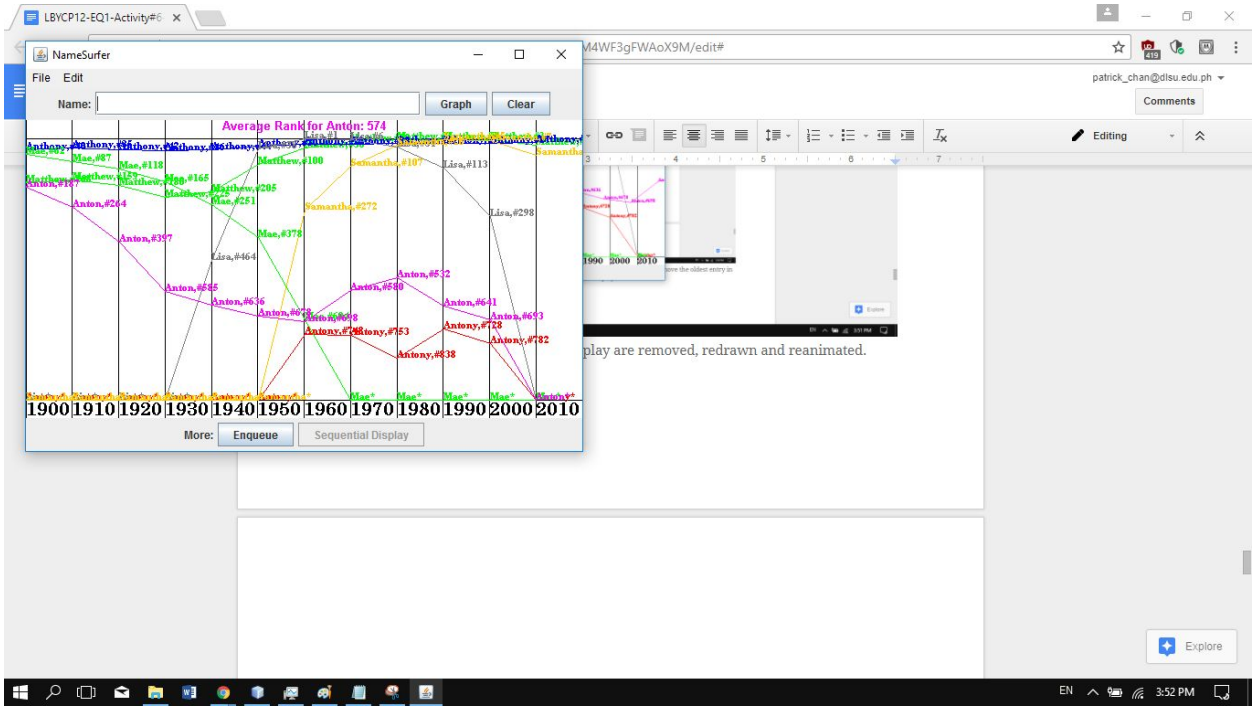
Asterisks are placed if the name is not in the top 1000 for that decade.



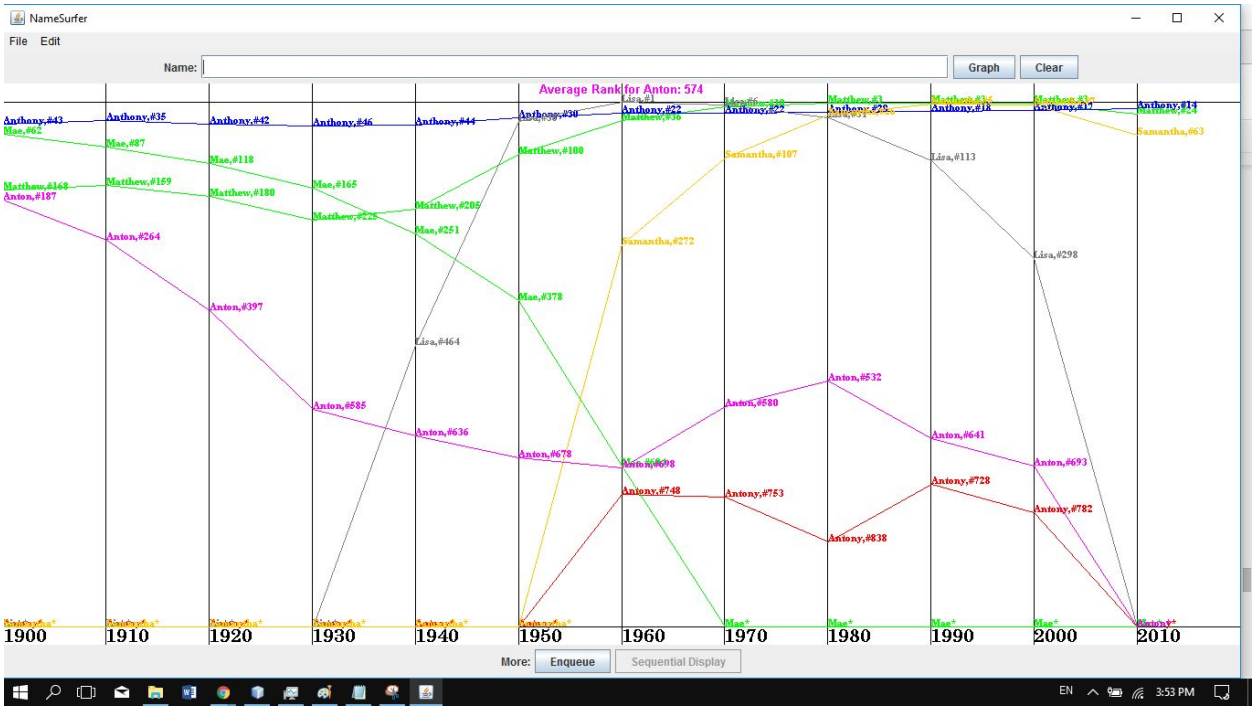
When 7 entries are displayed, inputting the next entry would remove the oldest entry in display.



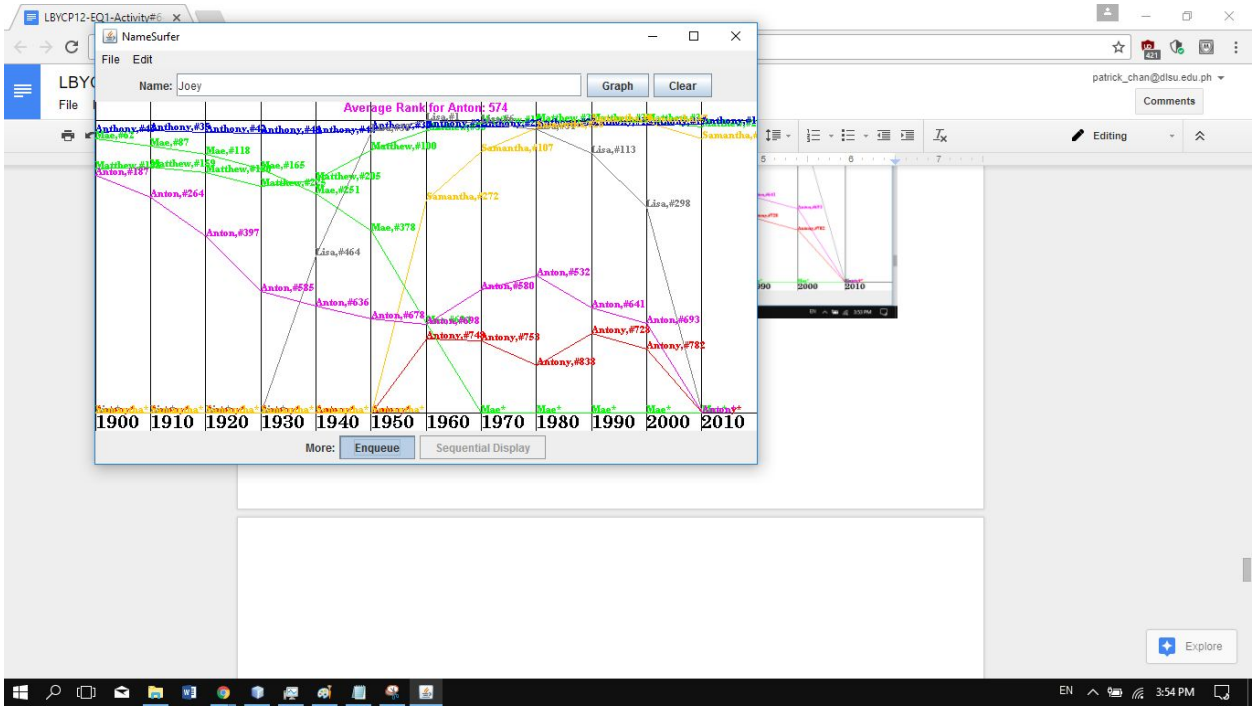
Input names that are already on display are removed, redrawn and reanimated.



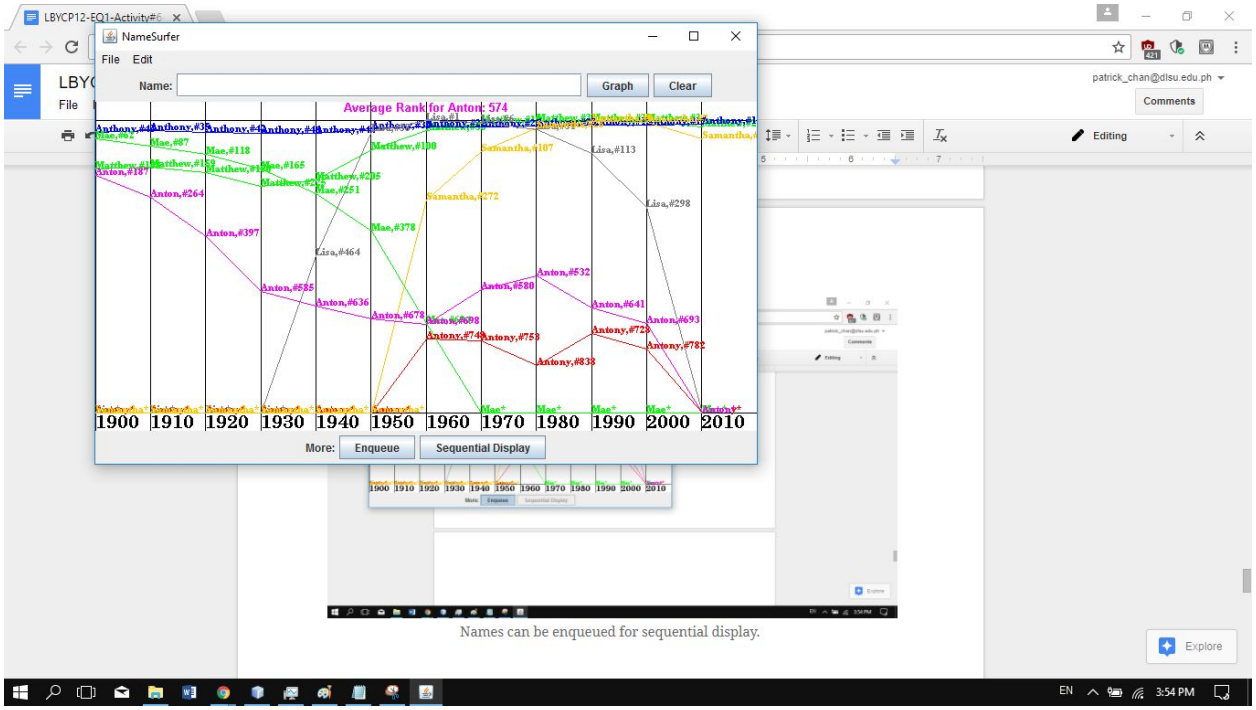
When the window is resized, the graph display scales accordingly, and so is the text field.



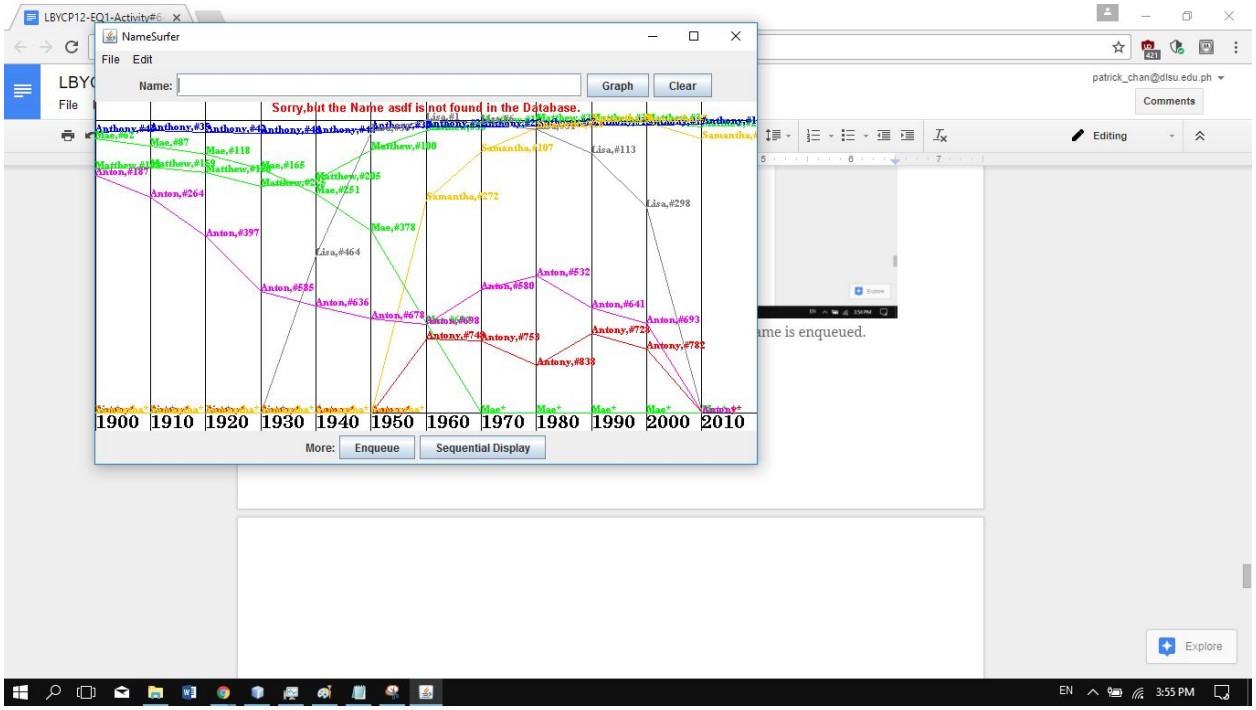
The display when the window is enlarged.



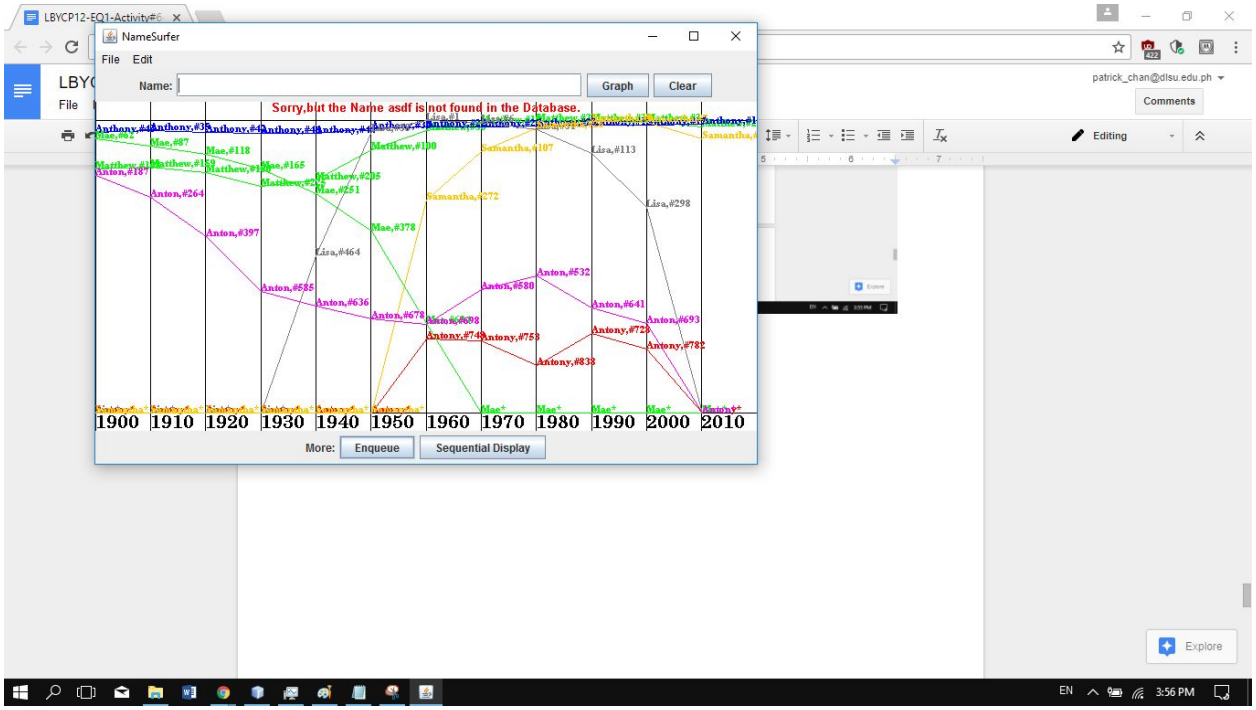
Names can be enqueued for sequential display.



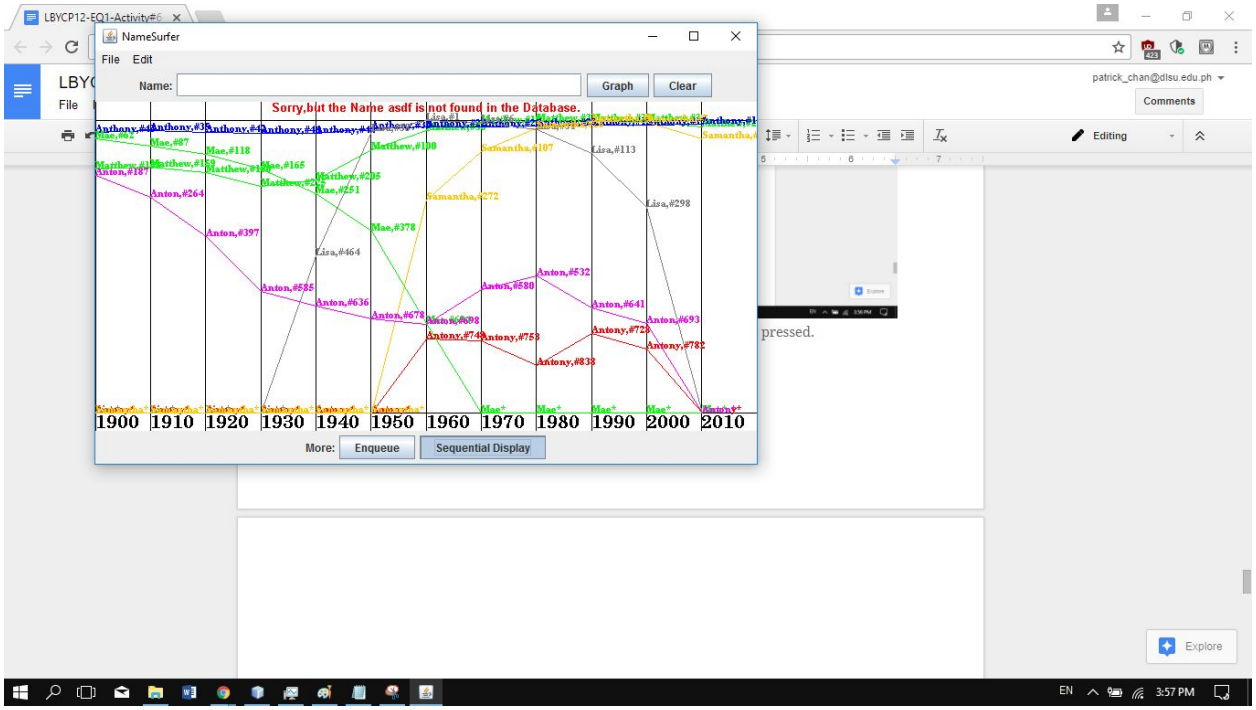
Sequential Display button is only available when at least one name is enqueued.



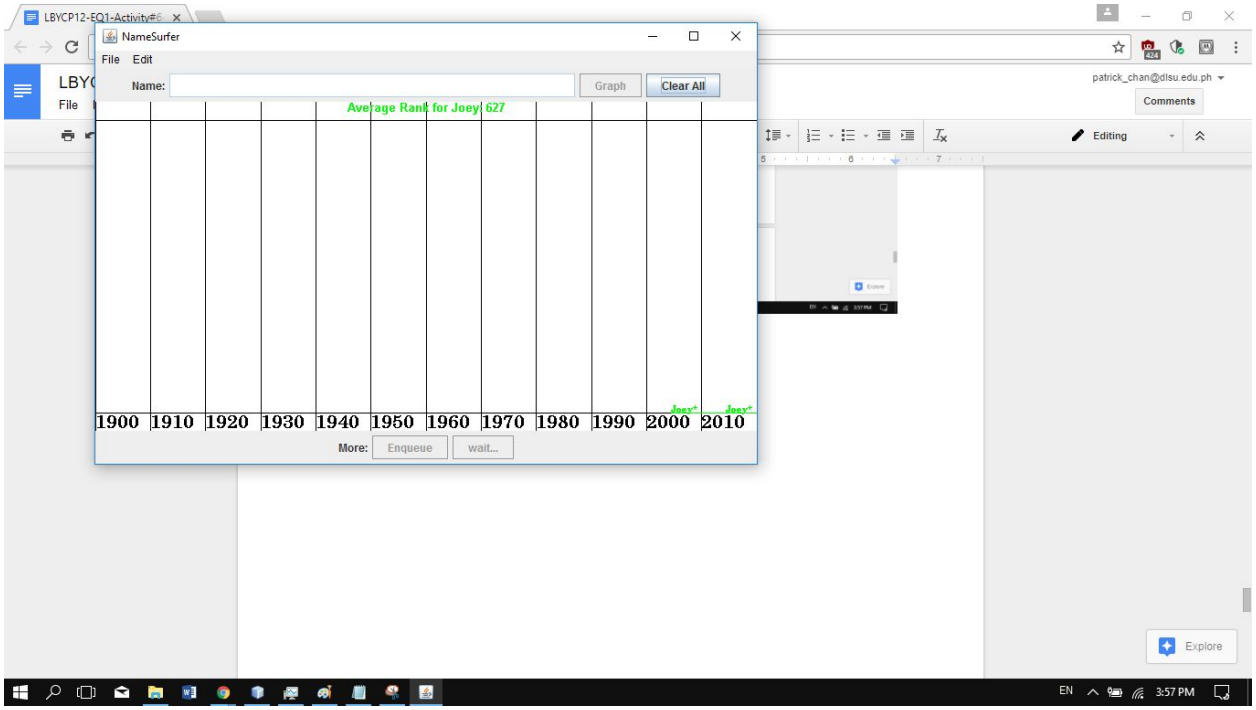
Names not found are not enqueued.



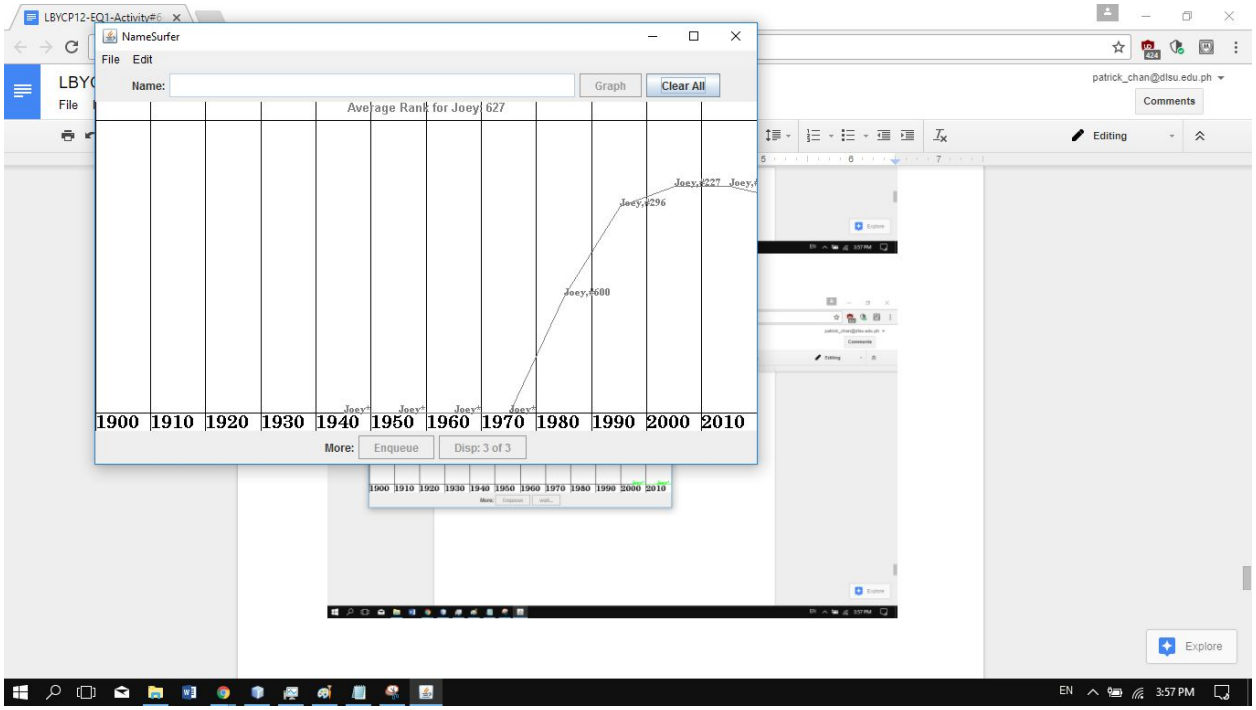
Focus is retained in the text field if enqueue or graph is pressed.



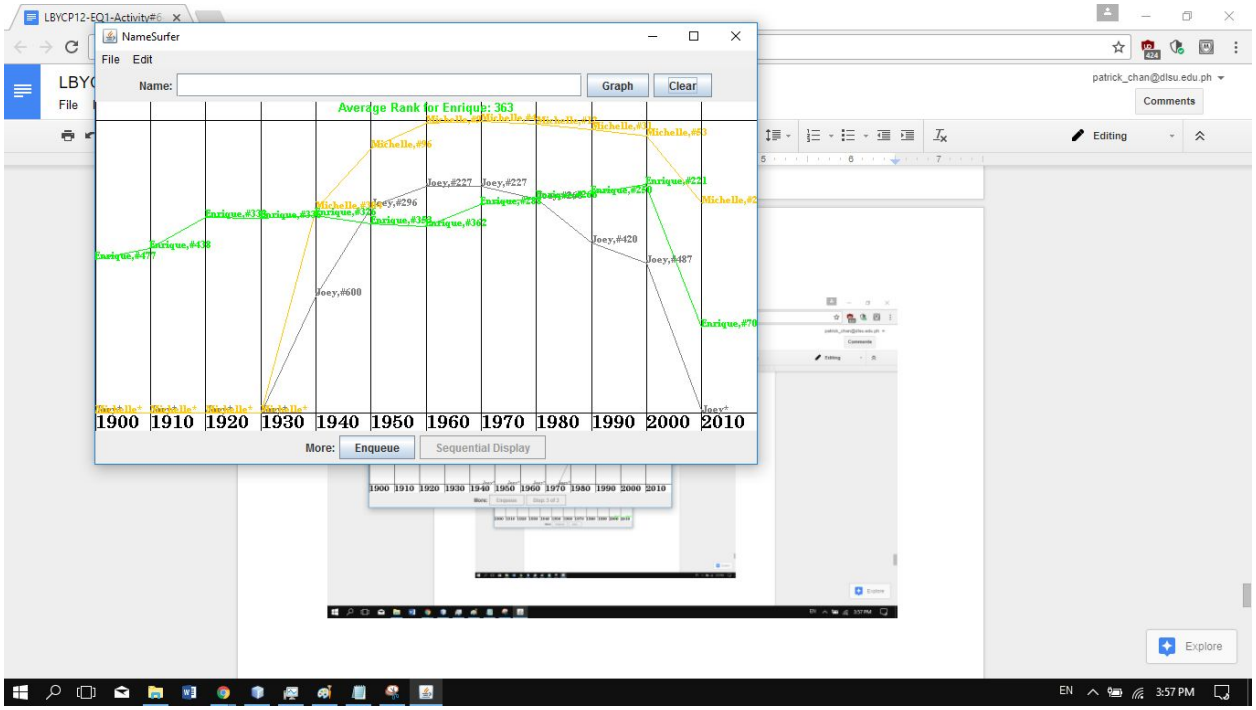
Sequential Display will clear the graph first.



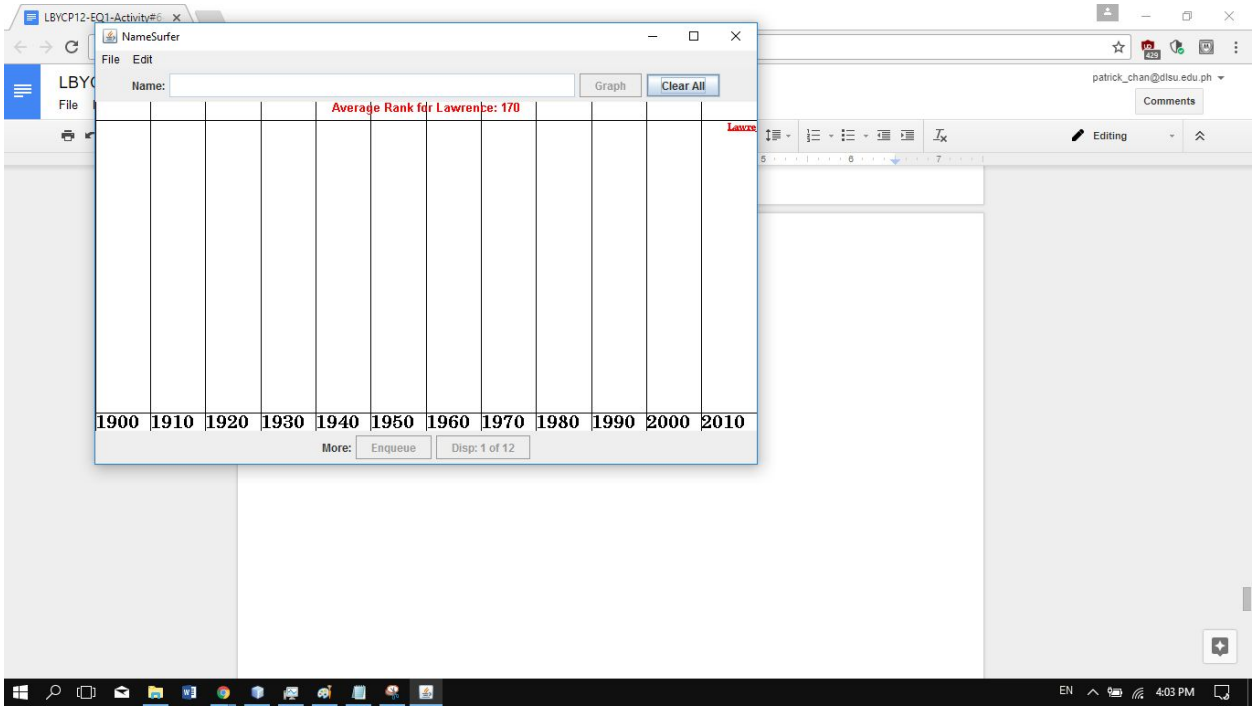
Note how the labels in the interactors changed, and how some have become disabled.



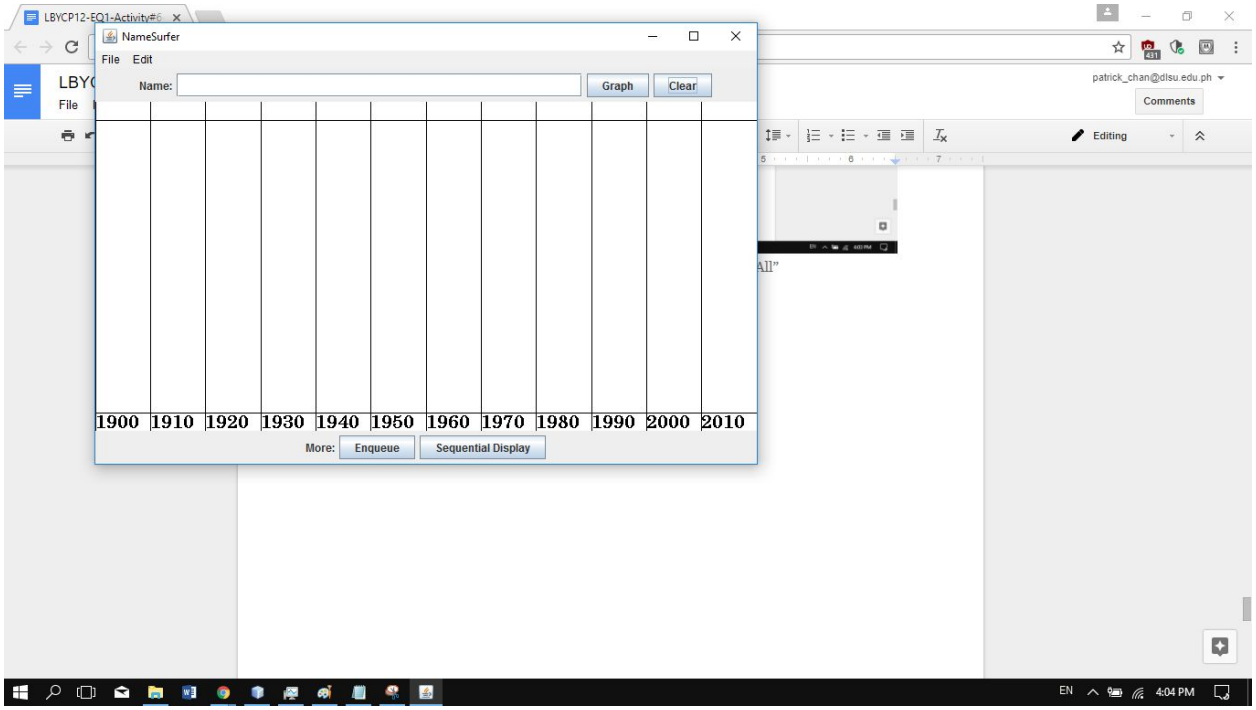
Each enqueued graph is displayed one by one, and the number of names still in queue are shown in place of the “Sequential Display” button.



Then, every name is again plotted, but on a single graph, while the interactors have returned to their original states.



Notice how “Clear” button has changed to “Clear All”



“Clear All” stops the Sequential Display Operation, and clears all data.

SUMMARY

First of all, this activity has helped me recall how to load data from a file, and store them, entry per entry, in a database. What's new about this, is that I also learned how to use the split method for a String, which made it much easier to parse the different data in every line; as each line followed a similar format, and each data was separated by the same delimiter.

Another thing that I was able to explore during this activity, was the usage of interactors. From my background with event-driven programming back in the Breakout activity, I was able to extend this knowledge and apply it to the way I implemented the actionListener methods for the JField and the JButtons. One thing I liked about the addition of interactors in the program in this activity, was how the it became easier to add these interactors to the display, since I could simply state in which part of the display I wanted each interactor to be placed (NORTH, SOUTH, WEST, EAST), and the libraries would automatically arrange and format it for me. In addition, speaking of eventListeners, I didn't think that creating a resizable display was just as simple as adding a componentListener to the object, so as for it to respond when the user resizes the display.

Lastly, for the integration of graphs into this activity, I initially had trouble coming up with ideas on how the Queue data structure can be used in the implementation of this activity, as it seemed to be less efficient if I simply replaced my lists with queues. Then, as I asked for some ideas, I learned how I could use a Queue for extensions on the activity, and realized the usefulness of a Queue data Structure. Moreover, to further take advantage of the features of a Queue data structure, when I saw how my graph animation causes any inputs made during the animation to not be recorded, I immediately thought of using a queue as a buffer, so that I could retain all of the user's inputs, while still being able to show the animation for each one.

APPENDIX

(The code for all classes are also attached in the Google Drive Folder)

The Code:

A) Namesurfer.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.namesurfer;
import acm.graphics.*;
import acm.io.*;
import acm.program.*;
import acm.util.*;
import java.io.*;
import java.applet.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;
import javax.swing.*;
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
//public class NameSurfer extends ConsoleProgram implements NameSurferConstants
{
public class NameSurfer extends Program implements NameSurferConstants {
   //~~~~~~~~~~~~~~~~~~~~~~~~~~~ Main Classes ~~~~~~~~~~~~~~~~~~~~~~~~~~~//
   //main classes
   public static void main(String[] args) {
      new NameSurfer().start(args);
   }
   //This method has the responsibility for reading in the data base
   //and initializing the interactors at the top of the window.
   public void init(){
      //other init
      add(gr);
      //interactors-add
      add(nlab,NORTH);
      add(nfie,NORTH);
      add(nb1,NORTH);
      add(nb2,NORTH);
      add(slab1,SOUTH);
      add(sb1,SOUTH);
      add(sb2,SOUTH);
      sb2.setEnabled(false);
```

```
//interactors-performed
nfie.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      //graph(e.getActionCommand());
      inputBuffer.push(e.getActionCommand());
      //isG=true;
      nfie.setText("");
   }
});
nb1.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      //graph(jfie.getText());
      inputBuffer.push(nfie.getText());
      //isG=true;
      nfie.setText("");
      nfie.requestFocusInWindow();
   }
});
nb2.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      clear();
      isInterrupt=true;
      nfie.setText("");
   }
});
sb1.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      if(db.findEntry(nfie.getText())==null){
         gr.notFound(nfie.getText());
      } else {
         inQ.push(nfie.getText());
      }
      nfie.setText("");
      sb2.setEnabled(!inQ.isEmpty());
      nfie.requestFocusInWindow();
   }
});
sb2.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      clear();
      isSeqDisp=true;
   }
```

```
    });
    //MORE interactors
    addComponentListener(new ComponentAdapter() {
        @Override
        public void componentResized(ComponentEvent e) {
            nfie.setColumns((getWidth()/nlab.getWidth()*4)/2);
        }
    });
}
public void run(){
    while(true){
        pause(0);
        if(isSeqDisp){
            seqDisp();
            isSeqDisp=false;
        } else if(!inputBuffer.isEmpty()){
            graph(inputBuffer.front(),false);
            inputBuffer.pop();
            //isG=false;
        }
    }
}
//~~~~~~~~~~~~~~~~~~~~~~~~~ Debugging & Misc ~~~~~~~~~~~~~~~~~~~~~~~~~//
public static void p(Object a){//debug
    System.out.print(a+"");
}
public static void pl(Object a){//debug
    System.out.println(a+"");
}
public static void pl(){//debug
    System.out.println();
}
//application size  //alternative method is setSize(x,y)
//public static final int APPLICATION_WIDTH = 400;
//public static final int APPLICATION_HEIGHT = 650;



//~~~~~~~~~~~~~~~~~~~~~~~~~ Global Variables ~~~~~~~~~~~~~~~~~~~~~~~~~//
JLabel nlab=new JLabel("Name:");
JTextField nfie=new JTextField(40);
JButton nb1=new JButton("Graph");
JButton nb2=new JButton("Clear");
NameSurferDataBase db=new NameSurferDataBase(NAMES_DATA_FILE);
NameSurferGraph gr=new NameSurferGraph();
//MORE
JLabel slab1=new JLabel("More:");
```

```
JButton sb1=new JButton("Enqueue");
JButton sb2=new JButton("Sequential Display");
Queue<String> inQ=new Queue<>();
private boolean isSeqDisp=false;
boolean isInterrupt=false;//if clear all is pressed during seqDisp
//shift?
//boolean isG=false;//i dont think this is needed
Queue<String> inputBuffer=new Queue<>();
//NameSurferEntry aaa= new NameSurferEntry("hi 0 1 2 3 4 5 6 7 8 9 0 1");
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Methods ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
private void graph(String input,boolean isRandColor){
   NameSurferEntry get=db.findEntry(input);
   if(get==null){pl("not found");
      gr.notFound(input);
      pause(300);
      //println("Name not found.");
   } else {//pl(get.toString());
      gr.addEntry(get,isRandColor);
   }
}
private void clear(){
   gr.clear();
}
private void seqDisp(){
   inputBuffer.clr();
   String aa=sb2.getText();
   String bb=nb2.getText();
   isInterrupt=false;
   //disp handa
   nfie.setEnabled(false);
   sb2.setEnabled(false);
   sb2.setText("wait...");
   sb1.setEnabled(false);
   nb1.setEnabled(false);
   nb2.setText("Clear All");
   for(int i=1;i<=inQ.size() && !isInterrupt;i++){
      graph(inQ.front(),true); //pl(inQ+"--test");
      sb2.setText("Disp: "+i+" of "+inQ.size());
      pause(700);
      inQ.push(inQ.front());
      inQ.pop();
      clear();
   }
   while(!inQ.isEmpty() && !isInterrupt){
      graph(inQ.front(),false);
      inQ.pop();
   }
```

```
      //balik
      nfie.setEnabled(true);
      sb2.setText(aa);
      nb2.setText(bb);
      sb2.setEnabled(!inQ.isEmpty());
      sb1.setEnabled(true);
      nb1.setEnabled(true);
      nb2.setEnabled(true);
      if(isInterrupt){
         clear();
         inQ.clr();
         sb2.setEnabled(!inQ.isEmpty());
      }
   }
}
```

B) JGraph.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.namesurfer;
import acm.graphics.*;
import java.awt.*;
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class JGraph extends GCompound implements NameSurferConstants{
   private NameSurferEntry src;
   private GLine[] pt=new GLine[NDECADES-1];
   private GLabel[] lab=new GLabel[NDECADES];
   private double grHt;//graph height
   private double spc;//just in case
   private static final Font FONT=new Font("Times New Roman",1,11);
   public Color color;

   //constructor
   //add the jgraph to 0,0 of graph height (so exclude top offset)
   public JGraph(NameSurferEntry source,double graphWidth,double
graphHeight,Color color){
      //init
      src=source;
      spc=graphWidth/NDECADES;
      grHt=graphHeight;
      this.color=color;
      //parts
      for(int i=0;i<NDECADES-1;i++){
         //line
```

```
            pt[i]=new GLine(spc*i,placeRank(src.getRank(i)),
                  spc*(i+1),placeRank(src.getRank(i+1)));
            pt[i].setColor(color);
            add(pt[i]);
            //text
            if(src.getRank(i)==0){
               lab[i]=new GLabel(src.getName()+"*");
            } else {
               lab[i]=new GLabel(src.getName()+",Rank "+src.getRank(i));
            }
            lab[i].setFont(FONT);
            lab[i].setColor(color);
            add(lab[i],pt[i].getStartPoint());
         }
            //text-last decade
         if(src.getRank(NDECADES-1)==0){
            lab[NDECADES-1]=new GLabel(src.getName()+"*");
         } else {
            lab[NDECADES-1]=new GLabel(src.getName()+",Rank
"+src.getRank(NDECADES-1));
         }
         lab[NDECADES-1].setFont(FONT);
         lab[NDECADES-1].setColor(color);
         add(lab[NDECADES-1],pt[NDECADES-2].getEndPoint());
      }

      ////////////////other methods/////////////////////
      private double placeRank(int rank){//requires grHt initialized
         if(rank>MAX_RANK || rank<0){
            throw new RuntimeException("placeRank: Invalid Rank--"+rank
               +"\nrange is 0 to "+MAX_RANK);
         } else if (rank==0){
            return grHt;//since origin on top
         } else {
            double divSize=grHt/(MAX_RANK-1);
            //1 nakadikit sa taas, 1000 nakadikit sa ilalim
            return (rank-1)*divSize;
         }
      }
      //reposition
      public void redraw(double graphWidth,double graphHeight){
         removeAll();
         spc=graphWidth/NDECADES;
         grHt=graphHeight;
         //parts
         for(int i=0;i<NDECADES-1;i++){
            //line
```

```
            pt[i]=new GLine(spc*i,placeRank(src.getRank(i)),
                  spc*(i+1),placeRank(src.getRank(i+1)));
            pt[i].setColor(color);
            add(pt[i]);
            //text
            if(src.getRank(i)==0){
               lab[i]=new GLabel(src.getName()+"*");
            } else {
               lab[i]=new GLabel(src.getName()+",#"+src.getRank(i));
            }
            lab[i].setFont(FONT);
            lab[i].setColor(color);
            add(lab[i],pt[i].getStartPoint());
         }
            //text-last decade
         if(src.getRank(NDECADES-1)==0){
            lab[NDECADES-1]=new GLabel(src.getName()+"*");
         } else {
            lab[NDECADES-1]=new GLabel(src.getName()+",#"+src.getRank(NDECADES-1));
         }
         lab[NDECADES-1].setFont(FONT);
         lab[NDECADES-1].setColor(color);
         add(lab[NDECADES-1],pt[NDECADES-2].getEndPoint());
      }

      public String getName(){
         return src.getName();
      }
}
```

C) LList.java

```
package ph.edu.dlsu.chan.namesurfer;
class LList<E> implements ListInterface<E>{
      private Node<E> head;
      private int numItems;

   public void createList(){
      numItems = 0;
      head = null;
   } //end constructor

   public void add(int index, E item) throws ListIndexOutOfBoundsException {
      if ( index > 0 && index <= numItems + 1){
```

```
          if (index == 1){ // Create head
                    Node<E> newNode = new Node<E>(item);
                    newNode.setNext(head);
                    head = newNode;
          }
             else
          {
                    Node<E> newNode = new Node<E>(item);
                    Node<E> previous = find(index-1);
                    newNode.setNext(previous.getNext());
                    previous.setNext(newNode);
          }
             numItems++;
             }
      else
          throw new ListIndexOutOfBoundsException("ADD ERROR: List Index Out Of
Bounds");
   }

   public void remove(int index) throws ListIndexOutOfBoundsException{
      if ( index > 0 && index <= numItems + 1){
        if (index == 1){
           head = head.getNext();
        }
        else{
          Node<E> previous = find(index-1);
          Node<E> current = previous.getNext();
          previous.setNext(current.getNext());
         }
         numItems--;
      }
     else
          throw new ListIndexOutOfBoundsException("REMOVE ERROR: List Index Out Of
Bounds");
   }




   public boolean isEmpty(){
         return numItems == 0;
   }
```

```
  public E get(int index) throws ListIndexOutOfBoundsException{
     if ( index > 0 && index <= numItems + 1){
          Node<E> current = find(index);
          E item = current.getItem();
          return item;
     }
     else
         throw new ListIndexOutOfBoundsException("GET ERROR: List Index Out Of
Bounds");
  }

  public int size(){
        return numItems;
  }

  /// Locate a specified node in a linked list:
  private Node<E> find(int index){
  // precondition: index is the number of the desired node,
  // precondition: assume 1 ≤ index ≤ numItems+1;
  // postcondition: returns a reference to the desired node.
  Node<E> current = head;

  for (int i = 1; i < index; i++)
  {
    current = current.getNext();
  }

  return current;
  } //end find



  public String toString(){
     StringBuilder a=new StringBuilder("[");

     for(int i=1;i<=size();i++){
        a.append(find(i).getItem().toString()+",");
     }
      a.append("\b]");
      return a.toString();
  }
}//end class
```

D) ListEmptyException.java

```
/*
 * File: ListEmptyException.java
 * ----------------
 * This class handles a Empty list
 */
package ph.edu.dlsu.chan.namesurfer;
public class ListEmptyException extends RuntimeException{
   public ListEmptyException(String s){
      super(s);
   }//end constructor
} //end ListException
```

E) ListFullException.java

```
/*
 * File: ListFullException.java
 * ----------------
 * This class handles a full list
 */
package ph.edu.dlsu.chan.namesurfer;
public class ListFullException extends RuntimeException{
   public ListFullException(String s){
      super(s);
   }//end constructor
} //end ListException
```

F) ListIndexOutOfBoundsException.java

```
/*
 * File: ListIndexOutOfBoundsException.java
 * ----------------
 * This class handles the case when list index given is out of bounds
 */
package ph.edu.dlsu.chan.namesurfer;

public class ListIndexOutOfBoundsException extends IndexOutOfBoundsException{
     public ListIndexOutOfBoundsException(String s){
       super(s);
     }//end constructor
```

```
} //end ListIndexOutofBoundsException
```

G) ListInterfaace.java

```
/*
 * File: List.java
 * ----------------
 * This is the List ADT definition
 */
package ph.edu.dlsu.chan.namesurfer;

public interface ListInterface<E>{


  public void createList();
  // precondition: none
  // postcondition: Create an empty list


  public void add(int index, E item) throws ListIndexOutOfBoundsException;
// precondition: index (to be added) is within the position of the list of items,
1<=index<=size()+1
// postcondition: Insert item at position index of a list
// if 1<=index<= size()+1. If index <= size(), items
// at position index onwards are shifted one position
// to the right. Throws an exception when index is out of range.



  public void remove(int index) throws ListIndexOutOfBoundsException;
// precondition: index (to be removed) is within the position of the list of items,
1<=index<=size()
// postcondition: Remove item at position index of a list
// if 1<=index<= size(). Items at position
// index+1 onwards are shifted one position to the left
// Throws an exception when index is out of range, or if list is empty.



 public boolean isEmpty();
// precondition: none
// postcondition: Determine if a list is empty
```

```
public E get(int index) throws ListIndexOutOfBoundsException;
// precondition: index is within the position of the list of items, 1<=index<=size()
// postcondition: Returns item at position index of
// a list if 1<=index<=size(). Throws an exception if index is out of range.



public int size();
// precondition: none
// postcondition: Returns number of items in a list

}
```

H) NameSurferConstants.java

```
package ph.edu.dlsu.chan.namesurfer;
/*
 * File: NameSurferConstants.java
 * ------------------------------
 * This file declares several constants that are shared by the
 * different modules in the NameSurfer application.  Any class
 * that implements this interface can use these constants.
 */

public interface NameSurferConstants {

	/** The width of the application window */
	public static final int APPLICATION_WIDTH = 800;

	/** The height of the application window */
	public static final int APPLICATION_HEIGHT = 600;

	/** The name of the file containing the data */
	public static final String NAMES_DATA_FILE = "names-data.txt";

	/** The first decade in the database */
	public static final int START_DECADE = 1900;

	/** The number of decades */
	public static final int NDECADES = 12;

	/** The maximum rank in the database */
	public static final int MAX_RANK = 1000;
```

```
	/** The number of pixels to reserve at the top and bottom */
	public static final int GRAPH_MARGIN_SIZE = 20;

}
```

I) NameSurferDatabase.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.namesurfer;
import java.io.*;
import java.util.Arrays;

/*
 * File: NameSurferDataBase.java
 * -----------------------------
 * This class keeps track of the complete database of names.
 * The constructor reads in the database from a file, and
 * the only public method makes it possible to look up a
 * name and get back the corresponding NameSurferEntry.
 * Names are matched independent of case, so that "Eric"
 * and "ERIC" are the same names.
 */

public class NameSurferDataBase implements NameSurferConstants {
	private LList<NameSurferEntry> entries=new LList<>();
       private FileReader in;
       private BufferedReader br;


	/* Constructor: NameSurferDataBase(filename) */
	/**
	 * Creates a new NameSurferDataBase and initializes it using the
	 * data in the specified file.  The constructor throws an error
	 * exception if the requested file does not exist or if an error
	 * occurs as the file is being read.
	 */
	public NameSurferDataBase(String filename) {
	String abPath=new File("").getAbsolutePath();
	try{
	   in=new FileReader(abPath+filename);//if relative given
	} catch (FileNotFoundException e){
	   try{
	      in=new FileReader(filename);//if absolute given
```

```
    } catch (FileNotFoundException f){
        System.err.println("NameSurferDataBase: File not "
                + "Found!\nfilename may be absolute or"
                + "relative path...\nabPath="+
                abPath+"\n"+f.getMessage());
    }
}
br=new BufferedReader(in);
entries.createList();
System.out.println("Loading...\n");
System.out.println("Names Loaded:");
String a="";
while(true){//break if a==null after read
    try{
        a=br.readLine();
    } catch(IOException e){
        System.err.print("Encountered IOException\n"
                + "previous entry:");
        if(entries.isEmpty()){
            System.err.print("(null)");
        } else {
            System.err.print(entries.get(entries.size()).toString());
        }
        System.err.println("\n"+e.getMessage());
    }
    if(a==null){
        break;
    }
    entries.add(entries.size()+1,new NameSurferEntry(a));
    System.out.print(entries.size()+"\r");
}
System.out.println("\nfile read DONE!");
 }

 /* Method: findEntry(name) */
 /**
  * Returns the NameSurferEntry associated with this name, if one
  * exists.  If the name does not appear in the database, this
  * method returns null.
  */
 public NameSurferEntry findEntry(String name) {
int index=binSe(entries,name);
if(index==-1){
    return null;
    /*index=linSe(entries,name);
    if(index==-1){
        return null;
```

```
        }
        System.err.println("Note: your text file is not "
              + "alphabetically arranged.");*/
    }
    return entries.get(index);
     }



/* Method: binSe(list,query) */
/**
 * Does binary search.
 * Assumes list entries are lexicographically arranged.
 * Method is case-insensitive.
 *
 * @param l NameSurferEntry LList
 * @param q query (name)
 * @return -1 if not found, index if found (1 based, since list)
 */
private static int binSe(LList<NameSurferEntry> l,String q){
    int low = 1;
    int high = l.size();
    int mid;

    while (low <= high) {
        mid = (low + high) / 2;

        if (l.get(mid).getName().compareToIgnoreCase(q) < 0) {
            low = mid + 1;
        } else if (l.get(mid).getName().compareToIgnoreCase(q) > 0) {
            high = mid - 1;
        } else {
            return mid;
        }
    }

    return -1;
}


/* Method: linSe(list,query) */
/**
 * Does linear search.
 * binSe possible fallback (backup).
 * Method is case-insensitive.
 *
 * @param l NameSurferEntry LList
```

```
     * @param q query (name)
     * @return -1 if not found, index if found (1 based, since list)
     */
    private static int linSe(LList<NameSurferEntry> l,String q){
       for(int i=1;i<=l.size();i++){
          if(l.get(i).getName().equalsIgnoreCase(q)){
             return i;
          }
       }
       return -1;
    }
}
```

J) NameSurferEntry.java

```
/* © 2017 by Patrick Matthew Chan */
/* This class represents a single entry in the database.  Each
 * NameSurferEntry contains a name and a list giving the popularity
 * of that name for each decade stretching back to 1900. */
package ph.edu.dlsu.chan.namesurfer;
import acm.util.*;
import java.util.*;
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class NameSurferEntry implements NameSurferConstants {
    private int[] data;//ranks
    private String name;

    private Object err(String s){
       throw new NameSurferEntryException(s+"\n data:"+
             Arrays.toString(data)+"\n name:"+name+
             "\n NDECADES:"+NDECADES);
    }

       /* Constructor: NameSurferEntry(line) */
       /**
        * Creates a new NameSurferEntry from a data line as it appears
        * in the data file.  Each line begins with the name, which is
        * followed by integers giving the rank of that name for each
        * decade.
     *
     * @exception  NameSurferEntryException if amount of data does
     * not match NDECADES, incorrect format, non-int data
        */
```

```
    public NameSurferEntry(String line) {
    String[] split=line.split(" ");
       if(split.length!=NDECADES+1){
          err("wrong format/size\nread:"+line+"\narr"+
                Arrays.toString(split));
       }
    data=new int[split.length-1];
    for(int i=1;i<split.length;i++){
       try{
          data[i-1]=Integer.parseInt(split[i]);
       } catch(NumberFormatException e){
          err("non int data\n"+e.getMessage());
       }
    }
    name=split[0];
    }


    /* Method: getName() */
    /**
     * Returns the name associated with this entry.
     */
    public String getName() {
              return name;
    }


    /* Method: getRank(decade) */
    /**
     * Returns the rank associated with an entry for a particular
     * decade.  The decade value is an integer indicating how many
     * decades have passed since the first year in the database,
     * which is given by the constant START_DECADE.  If a name does
     * not appear in a decade, the rank value is 0. {{The value of
  * decade ranges from 0 to NDECADES-1}}
  *
  * @param decade (0-based) number of decades passed after START_DECADE
     */
    public int getRank(int decade) {
    return data[decade];
    }


    /* Method: toString() */
    /**
     * Returns a string that makes it easy to see the value of a
     * NameSurferEntry.
```

```
         */
        public String toString() {
                return new String(name+Arrays.toString(data)+" <type:"
              +getClass().getName()+">");
        }




      /* Method: getNData() */
        /**
         * For checking.
       *
       * @return num of int data in this entry
         */
        /*public int getNData() {
                return data.length;
        }*/
}
```

K) NameSurferEntryException.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.namesurfer;
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class NameSurferEntryException extends RuntimeException{
   public NameSurferEntryException(String s){
      super("ERROR: "+s);
   }//end constructor
}
```

L) NameSurferGraph.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.namesurfer;
/*
 * File: NameSurferGraph.java
 * --------------------------
 * This class represents the canvas on which the graph of
 * names is drawn. This class is responsible for updating
 * (redrawing) the graphs whenever the list of entries changes
```

```
 * or the window is resized.
 */

import acm.graphics.*;
import static acm.util.JTFTools.pause;
import acm.util.RandomGenerator;
import java.awt.event.*;
import java.util.*;
import java.awt.*;

public class NameSurferGraph extends GCanvas
          implements NameSurferConstants, ComponentListener {

   private boolean isWorking=false;//so that update wont be called if unnecessary
   //^^ only add to methods w/ update in the end

   /**
    * Creates a new NameSurferGraph object that displays the data.
    */
   public NameSurferGraph() {
      isWorking=true;
      addComponentListener(this);
               //graphs.createList();//init
      grQ.clr();
      graphAddCount=0;
      update(false);
      isWorking=false;
   }


   /**
    * Clears the list of name surfer entries stored inside this class.
    */
   public void clear() {
      isWorking=true;
            //graphs.createList();//reset
      grQ.clr();
      graphAddCount=0;
      msgTop.setLabel("");
      update(false);
      isWorking=false;
   }


   /* Method: addEntry(entry) */
   /**
    * Adds a new NameSurferEntry to the list of entries on the display.
```

```
    * Note that this method does not actually draw the graph, but
    * simply stores the entry; the graph is drawn by calling update.
    */
   public void addEntry(NameSurferEntry entry, boolean isRandColor) {
      isWorking=true;
      Color curColor=null;
      //boolean isDuplicate=false;//apparently not needed anymore
               /*JGraph gr=new JGraph(entry,getWidth(),getHeight()-2*
                     GRAPH_MARGIN_SIZE,getCol(graphs.size()+1));
               graphs.add(graphs.size()+1,gr);*/
      for(int i=grQ.size();i>=1;i--){//duplicate check(grQ.size may vary,so decrement is
used)
         JGraph now=grQ.front();
         if(now.getName().equals(entry.getName())){
            curColor=now.color;
            grQ.pop();//can't break as queue has to be restored
         } else {
            grQ.push(grQ.front());
            grQ.pop();
         }
      }
      if(curColor==null){
         graphAddCount++;
         if(isRandColor){
            graphAddCount=RandomGenerator.getInstance().nextInt(6);
         }
         curColor=getCol(graphAddCount);
      }
      JGraph gr=new JGraph(entry,getWidth(),getHeight()-2*
            GRAPH_MARGIN_SIZE,curColor);
      if(grQ.size()>=MAX_GRAPHS){
         grQ.pop();
      }
      grQ.push(gr);
      //ave(++)
      double ave=0;
      for(int i=0;i<NDECADES;i++){
         ave=ave+entry.getRank(i);
         if(entry.getRank(i)==0){
            ave+=1001;
         }
      }
      ave=ave/NDECADES;
      msgTop.setColor(curColor);
      msgTop.setLabel("Average Rank for "+entry.getName()+": "+(int)ave);
      update(true);
      isWorking=false;
```

```
   }


   public static Color getCol(int index){
      switch(index%6){
         case 1:
            return Color.GRAY;
         case 3:
            return Color.GREEN;
         case 4:
            return Color.MAGENTA;
         case 2:
            return Color.ORANGE;
         case 5:
            return Color.RED;
         default://case 0 here also
            return Color.BLUE;
      }
   }


   /**
    * Updates the display image by deleting all the graphical objects
    * from the canvas and then reassembling the display according to
    * the list of entries. Your application must call update after
    * calling either clear or addEntry; update is also called whenever
    * the size of the canvas changes.
    */
   public void update(boolean isNewGraphAdded) {
      removeAll();
      //init
      for(int i=0;i<NDECADES;i++){
         div[i]=new GLine(0,0,0,getHeight());
         add(div[i],i*getWidth()/NDECADES,0);
         divYr[i]=new GLabel((START_DECADE+i*10)+"");
         divYr[i].setFont(FONT);
         add(divYr[i],i*getWidth()/NDECADES,getHeight()-FONT_Y_OFFSET);
      }
      horLimTop=new GLine(0,0,getWidth(),0);
      horLimBot=new GLine(0,0,getWidth(),0);
      add(horLimTop,0,/*FONT_Y_OFFSET*/GRAPH_MARGIN_SIZE);

add(horLimBot,0,getHeight()-/*(FONT_Y_OFFSET+FONT.getSize2D())*/GRAPH_MARGIN_
SIZE);
      //msg (++)
      msgTop.setFont(MSG_FONT);
      add(msgTop,(getWidth()-msgTop.getWidth())/2,msgTop.getAscent()-3);
```

```
    //draw graphs
            /*for(int i=1;i<=graphs.size();i++){//size=0, doesn't enter loop ("<"-necessary)
               graphs.get(i).redraw(getWidth(),getHeight()-2*GRAPH_MARGIN_SIZE);
               add(graphs.get(i),0,GRAPH_MARGIN_SIZE);
            }*/
    tempQ.clr();
    while(!grQ.isEmpty()){
       grQ.front().redraw(getWidth(),getHeight()-2*GRAPH_MARGIN_SIZE);
       add(grQ.front(),0,GRAPH_MARGIN_SIZE);
       if(grQ.size()==1 && isNewGraphAdded){anim(grQ.front());}
       if(!grQ.isEmpty()){//in case button is pressed in middle of animation
          tempQ.push(grQ.front());
          grQ.pop();
       }
    }
    while(!tempQ.isEmpty()){
       grQ.push(tempQ.front());
       tempQ.pop();
    }
}


/* Implementation of the ComponentListener interface */
public void componentHidden(ComponentEvent e) { }
public void componentMoved(ComponentEvent e) { }
public void componentResized(ComponentEvent e) {if(!isWorking){update(false);}}
public void componentShown(ComponentEvent e) { }



/**
 * Indicates that the name is not found.
 */
public void notFound(String input) {
    isWorking=true;
          //graphs.createList();//reset
    if(input.length()>20){
       input=input.substring(0,20).concat("...");
    }
    msgTop.setLabel("Sorry,but the Name "+input+
          " is not found in the Database.");
    msgTop.setColor(Color.RED);
    update(false);
    isWorking=false;
}
```

```
    private void anim(JGraph g){
        double x=g.getX();
        double y=g.getY();
        g.setLocation(getWidth(), y);
        //g.scale(.9*g.getX()-x);
        while(g.getX()>=x){
            g.move(-1,0);
            pause(1);
        }
    }




    //static final vars (global)
    private static final Font FONT=new Font("Century",1,20);
    private static final Font MSG_FONT=new Font("SansSerif",1,13);
    private static final int FONT_Y_OFFSET=3;
    private static final int MAX_GRAPHS=7;
    //global vars
    private GLine[] div=new GLine[NDECADES];
    private GLabel[] divYr=new GLabel[NDECADES];
    private GLine horLimTop;
    private GLine horLimBot;
    //private LList<JGraph> graphs=new LList<>();
    private Queue<JGraph> grQ=new Queue<>();
    private Queue<JGraph> tempQ=new Queue<>();
    private int graphAddCount=0;
    private GLabel msgTop=new GLabel("");
}
```

M) Node.java

```
package ph.edu.dlsu.chan.namesurfer;
public class Node<E>{

   private E item;
   private Node<E> next;

   public Node(E newItem, Node<E> nextNode){
        item = newItem; // DATA
        next = nextNode; //POINTER/LINK
   }
```

```
  public Node(E newItem){
       item = newItem;
       next = null;
  }

  public void setItem(E newItem){
        item = newItem;
  }

  public E getItem(){
        return item;
  }

  public void setNext(Node<E> nextNode){
        next = nextNode;
  }

  public Node<E> getNext(){
        return next;
  }
}// end class Node
```

N) Queue.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.namesurfer;
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class Queue<E> extends QueueBasic<E>{
   //constructor
   public Queue(){
      super();
   }

   //addtl methods
   public void clr(){
      items.createList();//reset
```

```
    }

    public String toString(){
        return new String("Q:"+items.toString());
    }
}
```

O) QueueBasic.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.namesurfer;
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class QueueBasic<E> implements QueueInterface<E>{
    protected LList<E> items=new LList<>();

    //constructor
    public QueueBasic(){
        items.createList();
    }

    //other methods
    public void push(E item){
        //if(!isFull()){
            items.add(items.size()+1,item);
        //} else {
         //   throw new QueueFullException("Full");
        //}
    }
    public E front(){
        if(!isEmpty()){
            return items.get(1);
        } else {
            throw new QueueEmptyException("Empty");
        }
    }
    public E rear(){
        if(!isEmpty()){
            return items.get(items.size());
        } else {
            throw new QueueEmptyException("Empty");
        }
    }
    public void pop(){
        if(!isEmpty()){
```

```
            items.remove(1);
        } else {
            throw new QueueEmptyException("Empty");
        }
    }
    public boolean isEmpty(){
        return items.isEmpty();
    }
    public int size(){
        return items.size();
    }
}
```

P) QueueEmptyException.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.namesurfer;
/* @author Patrick Matthew J. Chan [LBYCP12-EQ1]*/
public class QueueEmptyException extends RuntimeException{
    public QueueEmptyException(String s){
        super(s);
    }//end constructor
}
```

Q) QueueInterface.java

```
/* © 2017 by Patrick Matthew Chan */
package ph.edu.dlsu.chan.namesurfer;

/**
 *
 * @author Patrick Matthew J. Chan
 */
public interface QueueInterface<E> {
    public void push(E item);
    public E front();
    public E rear();
    public void pop();
    public boolean isEmpty();
    //public boolean isFull();
    public int size();
}
```

REFERENCES

1. E Roberts. *Art and Science of Java*. Pearson; 2013.
2. E Roberts, M Sahami, and M Stepp, *CS 106A: Programming Methodology (Java) Handouts*, Stanford University.