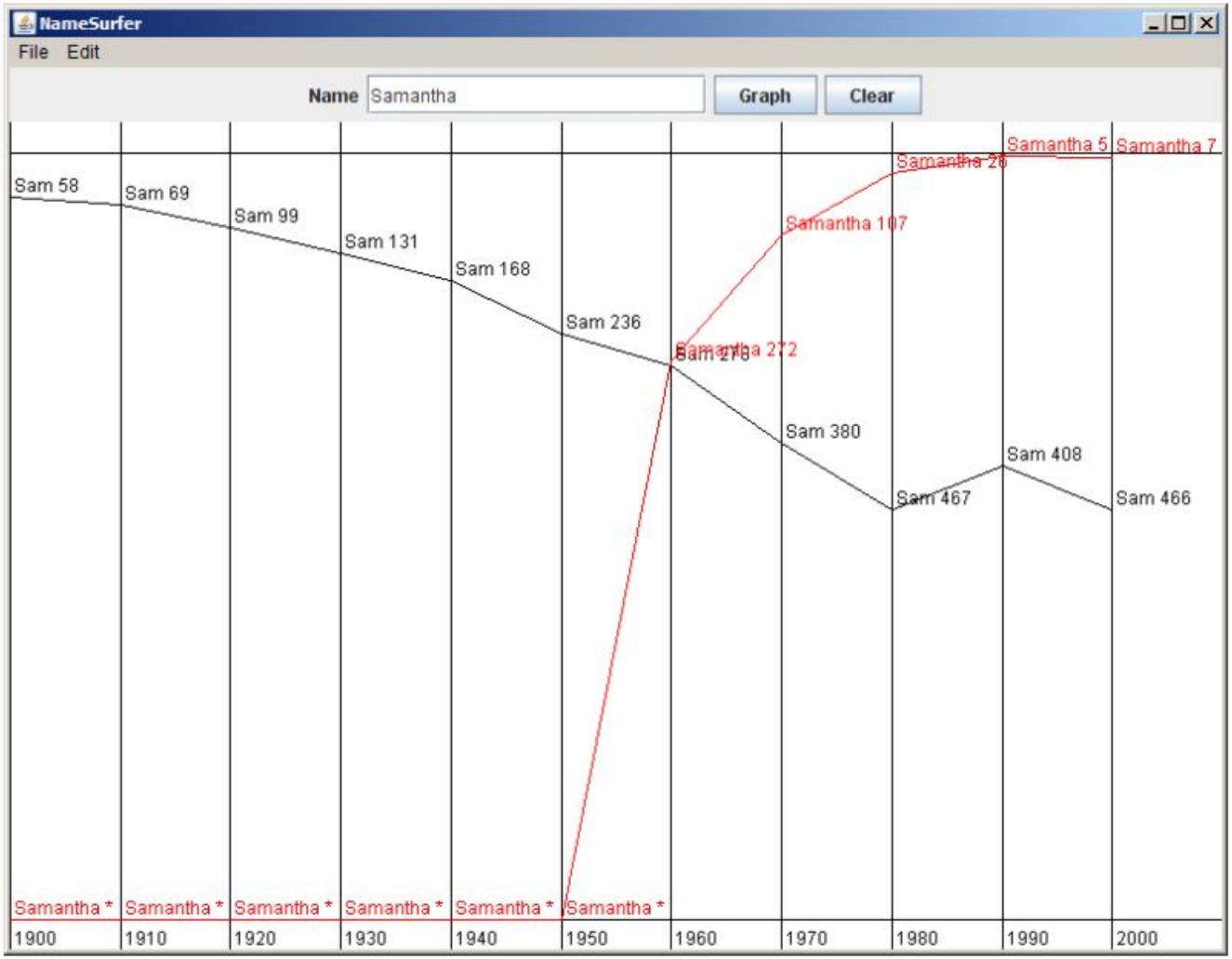
YEAH Session #6

March 1, 2017, 7:30 PM

Ashley Taylor and Ryan Eberhardt



YEAH Hours Schedule

Topic	Date	Time	Location
Assignment 6	Today!	Now!	Here!
Assignment 7	3/9 (Thurs)	7:30PM	Bishop Aud
Final Exam	3/20 (Mon)	8:30-11:30AM	TBD

Classes

How can we manage large programs?

Examples of Classes

- `GRect`: keeps track of x, y, width, height, color, visibility
- `ArrayList`: Maintains a list of elements (under the hood, it manages an array for you)
- `String`: stores all the characters that make up the String and provides methods for you to use those characters

Classes and Instances

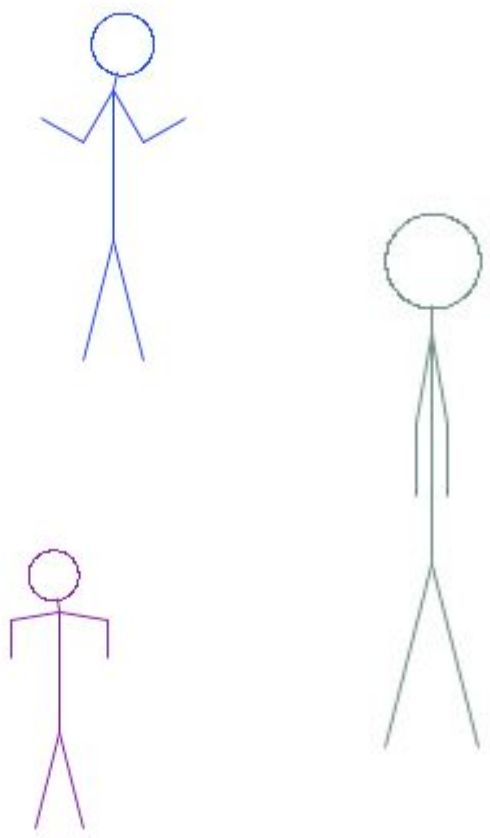
- A **class** is like a blueprint for something – it defines what something does and how it works
  - You can also think of classes like custom variable types
- An **instance** is an *actual copy* of what that entry describes

Parts of a Class

- Instance Variables: what information is part of each instance's identity (e.g. your name is part of you)
- Methods: things that the class can do (humans can age, grow, run, etc.)
  - Constructor: A special method that initializes all the instance variables (when you're born you get a name and an initial height/weight)

Dancing Stick Figures

- `StickFigure` Instance Variables
  - Size
  - Color
  - Position of limbs
- `StickFigure` Methods
  - Stand up
  - Sit down
  - Go to sleep
  - Dance!



Dancing Stick Figures - Instance Variables

```
public class StickFigure extends GCompound {
    // Lots of constants omitted here...

    // What color is the stick figure?
    private Color color;
    // How big is the stick figure?
    private double scale;
    // How are its limbs positioned?
    private int rightShoulderAngle = NORMAL_SHOULDER_ANGLE;
    private int rightForearmAngle = NORMAL_FOREARM_ANGLE;
    private int leftShoulderAngle = NORMAL_SHOULDER_ANGLE;
    private int leftForearmAngle = NORMAL_FOREARM_ANGLE;
    private int headAngle = NORMAL_HEAD_ANGLE;

    private GOval head;
    private GLine neck;
    private GLine back;
    // Many more GObjects omitted...
```

Dancing Stick Figures -- Constructor and Getters

```
public class StickFigure extends GCompound { ...
    public StickFigure(Color figureColor, double scaleFactor) {
        scale = scaleFactor;
        color = figureColor;
        makeBodyParts();
    }

    public Color getColor() {
        return color;
    }

    public boolean isHeadUpright() {
        return headAngle == 0;
    }
}
```

Dancing Stick Figures -- Other methods

```
public class StickFigure extends GCompound { ...
    public void moveToDancingPosition(double position) {
        rightShoulderAngle = (int) interpolate(NORMAL_SHOULDER_ANGLE,
                DAB_RIGHT_SHOULDER_ANGLE, position);
        rightForearmAngle = (int) interpolate(NORMAL_FOREARM_ANGLE,
                DAB_RIGHT_FOREARM_ANGLE, position);
        leftShoulderAngle = (int) interpolate(NORMAL_SHOULDER_ANGLE,
                DAB_LEFT_SHOULDER_ANGLE, position);
        leftForearmAngle = (int) interpolate(NORMAL_FOREARM_ANGLE,
                DAB_LEFT_FOREARM_ANGLE, position);
        headAngle = (int) interpolate(NORMAL_HEAD_ANGLE, DAB_HEAD_ANGLE,
                position);
        updateShapes();
    }
}
```

Using `StickFigure`

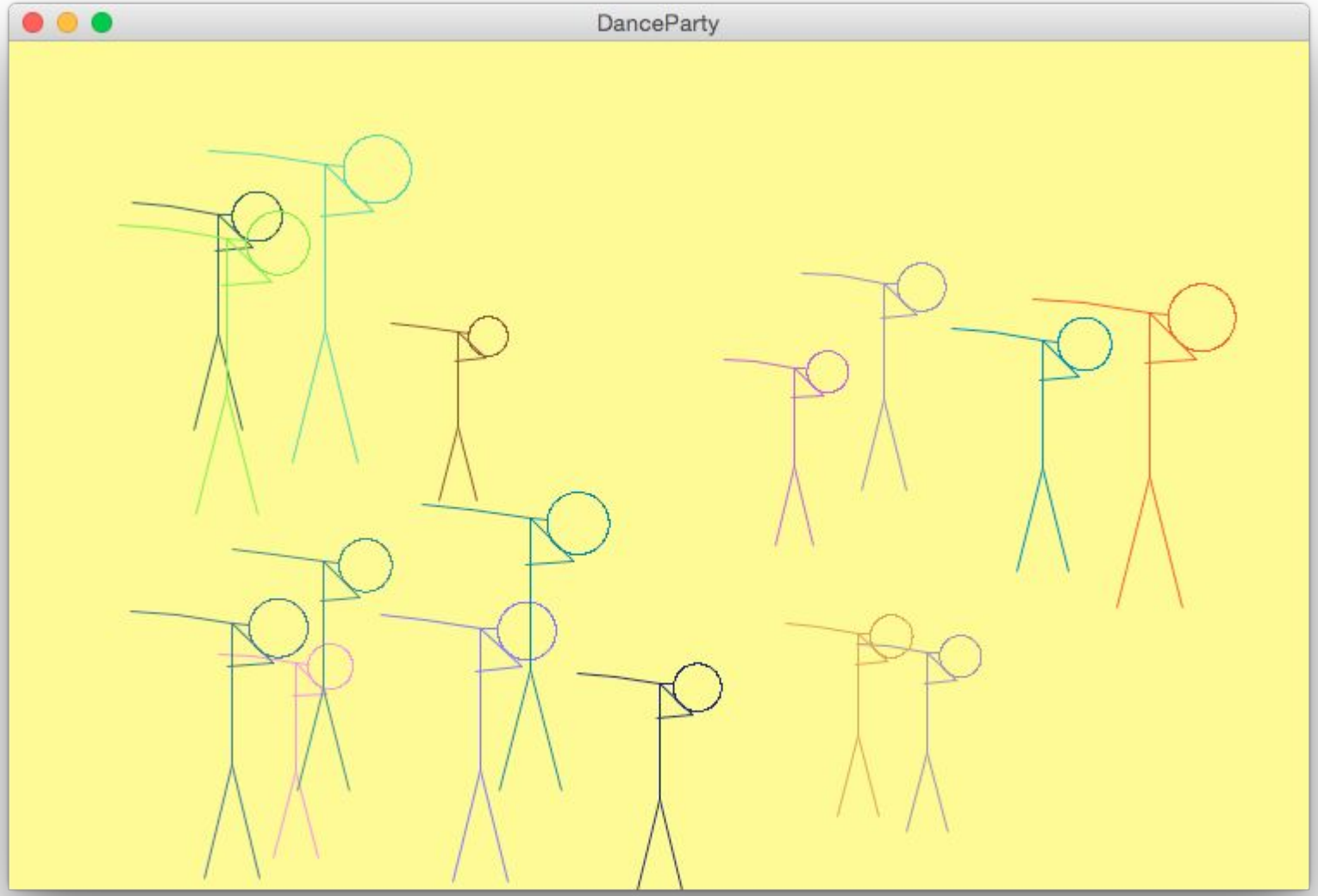
```
public void run() {
     ArrayList<StickFigure> figures = new ArrayList<StickFigure>();
     StickFigure firstFigure = new StickFigure(rgen.nextColor(),
                                               rgen.nextDouble(SCALE_MIN, SCALE_MAX));
     figures.add(firstFigure);
     add(firstFigure, getWidth() / 2, getHeight());

     // Go into an infinite loop that makes more figures and makes them dance
     while (true) {
          // Change background color
          this.setBackground(rgen.nextColor());
```

Using StickFigure

```
public void run() { ...
     while (true) { ...
          // Make the figures dab
          for (double progress = 0; progress <= 1; progress += DAB_RESOLUTION) {
               for (StickFigure figure : figures) {
                    figure.moveToDancingPosition(progress);
               }
               pause(SLEEP_TIME);
          }
          for (double progress = 1; progress >= 0; progress -= DAB_RESOLUTION) {
               for (StickFigure figure : figures) {
                    figure.moveToDancingPosition(progress);
               }
               pause(SLEEP_TIME);
          }

          // Add a new figure in a random location
          StickFigure newFigure = new StickFigure(rgen.nextColor(), rgen.nextDouble(1, 2));
          double x = rgen.nextDouble(newFigure.getWidth() / 2.0,
                                     getWidth() - newFigure.getWidth() / 2.0);
          double y = rgen.nextDouble(newFigure.getHeight(), getHeight());
          figures.add(newFigure);
          add(newFigure, x, y);
     }
}
```


DanceParty

Other classes

- See problem 2 (for an example like the Human Class) and problem 3 (for an example of a GCanvas class) from this week's section handout
- See bouncing balls example from lecture Monday, 2/27 (timecode 37:15 in SCPD recording)

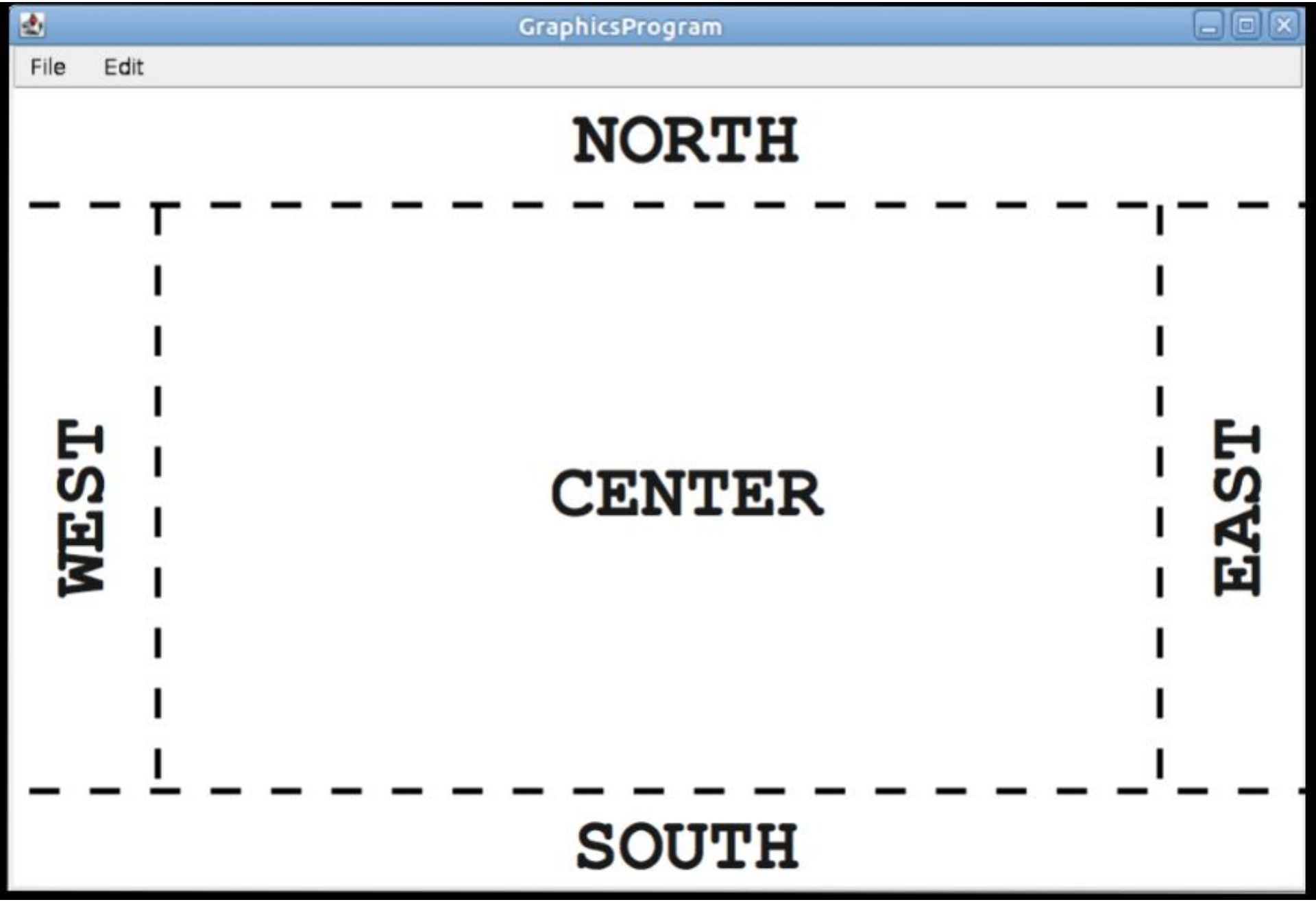
Interactors

Interactors

```
Jbutton button = new JButton("Add");
add(button, NORTH);
JTextField field = new JTextField(25);

// Listen for "ENTER" in text field
field.addActionListener(this);

add(field, NORTH);
```


GraphicsProgram
File
Edit
NORTH
WEST
CENTER
EAST
SOUTH

Interactors

- Add them in a specific *region* on screen (usually not CENTER! That's where the canvas goes)
- addActionListeners() in your main program
- Implement the actionPerformed method to respond to action events (just like you did for mouseMoved, mousePressed, etc.)
- JButton takes name on button as parameter – JTextField takes max text field length

Interactors

```
public void actionPerformed(ActionEvent e) {
    if(e.getActionCommand().equals("Add")) {
            …
    }
}
```

---- OR (BOTH EQUIVALENT) ----

```
public void actionPerformed(ActionEvent e) {
    if(e.getSource() == addButton) {
            …
    }
}
```

Interactors

- Can use e.getSource() to get the interactor that the user interacted with (but need all your interactors as instance variables to check equality)
- Can use e.getActionCommand() to get the *name* of the interactor that the user interacted with (don't need all your interactors as instance variables – only need to know their name! But still need text field as instance variable if you want to access its text)
- If you listen for ENTER on text fields, and want ENTER to be equivalent to pressing a button, name text field and button the SAME!
- Name of JButton is button's text. Use .setActionCommand(*name*) to set "name" of text fields

```
private JTextField field;
public void run() {
    field = new JTextField(25);
    field.addActionListener(this);
    field.setActionCommand("Add");
    add(field, NORTH);
    JButton button = new JButton("Add");
    add(button, NORTH);
    addActionListeners();
}
public void actionPerformed(ActionEvent e) {
    if(e.getActionCommand().equals("Add")) {
          // Will be true if user types ENTER
       // in text field OR clicks "Add"!
       String text = field.getText(); // get text
       }
}
```

NameSurfer!

- Due at 10:30AM on Wednesday, Mar. 8
- Practice with arrays, ArrayLists, HashMaps
- Practice with multiple classes/code files
- Interactors!

Key Functionality

- Read data from a file
- Interact with the user
- Plot the data

Key Functionality

- Read data from a file
  - Use BufferedReader to read raw text
  - Look through each line to get names and popularities
  - Store that data
- Interact with the user

- Plot the data

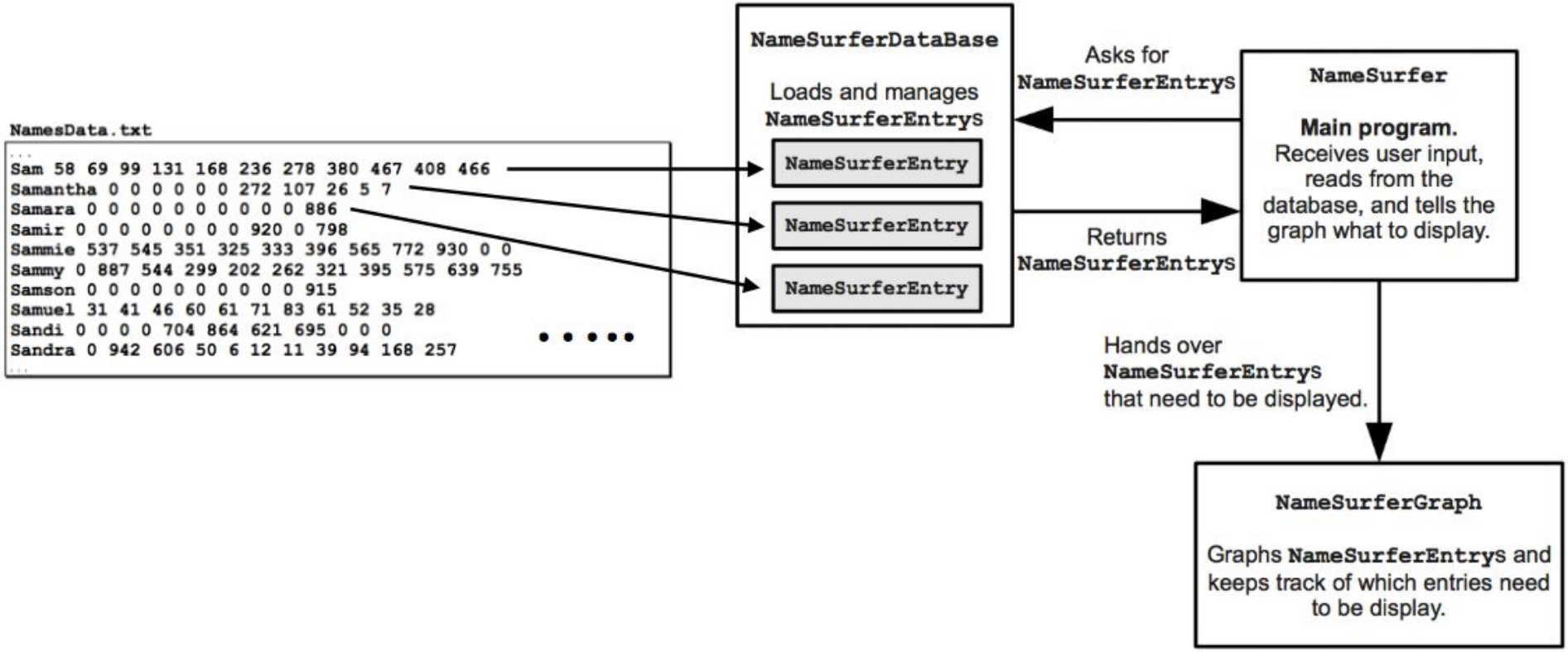
Key Functionality

- Read data from a file
  - Use BufferedReader to read raw text
  - Look through each line to get names and popularities
  - Store that data
- Interact with the user
  - Let user enter names to look up
  - Allow user to clear canvas
- Plot the data

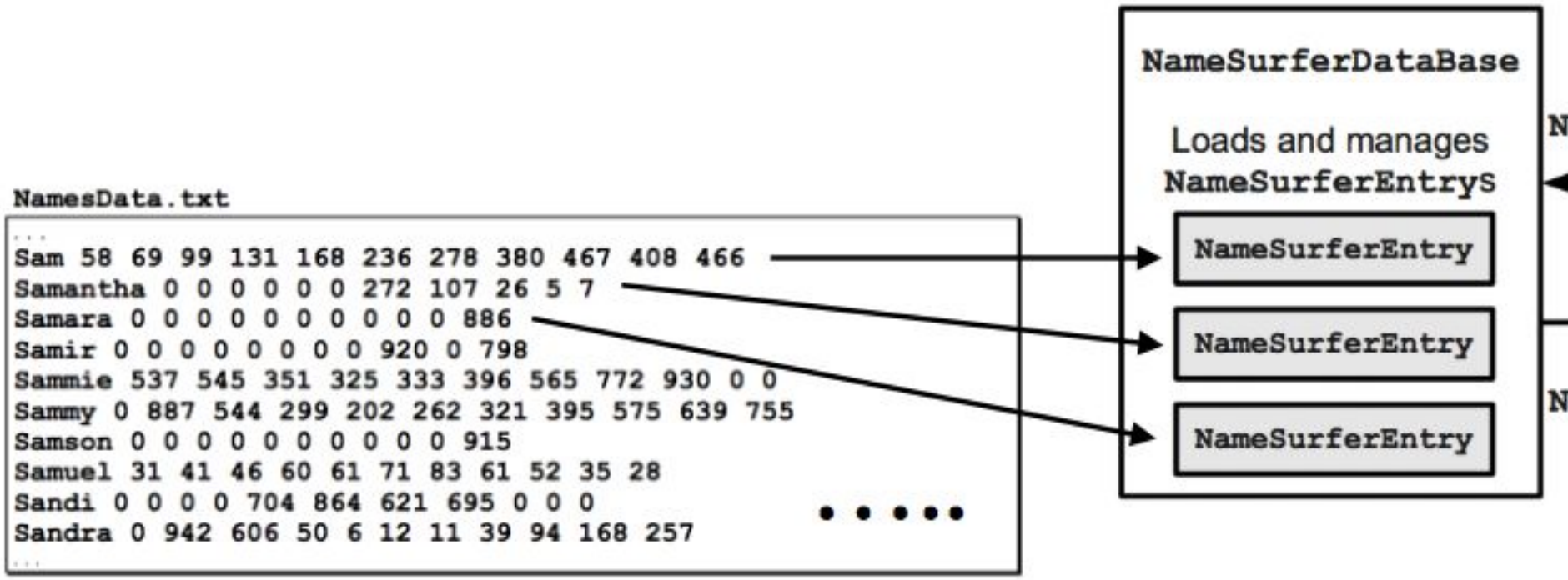
Key Functionality

- Read data from a file
  - Use BufferedReader to read raw text
  - Look through each line to get names and popularities
  - Store that data
- Interact with the user
  - Let user enter names to look up
  - Allow user to clear canvas
- Plot the data
  - Drawing with lots of GLines!

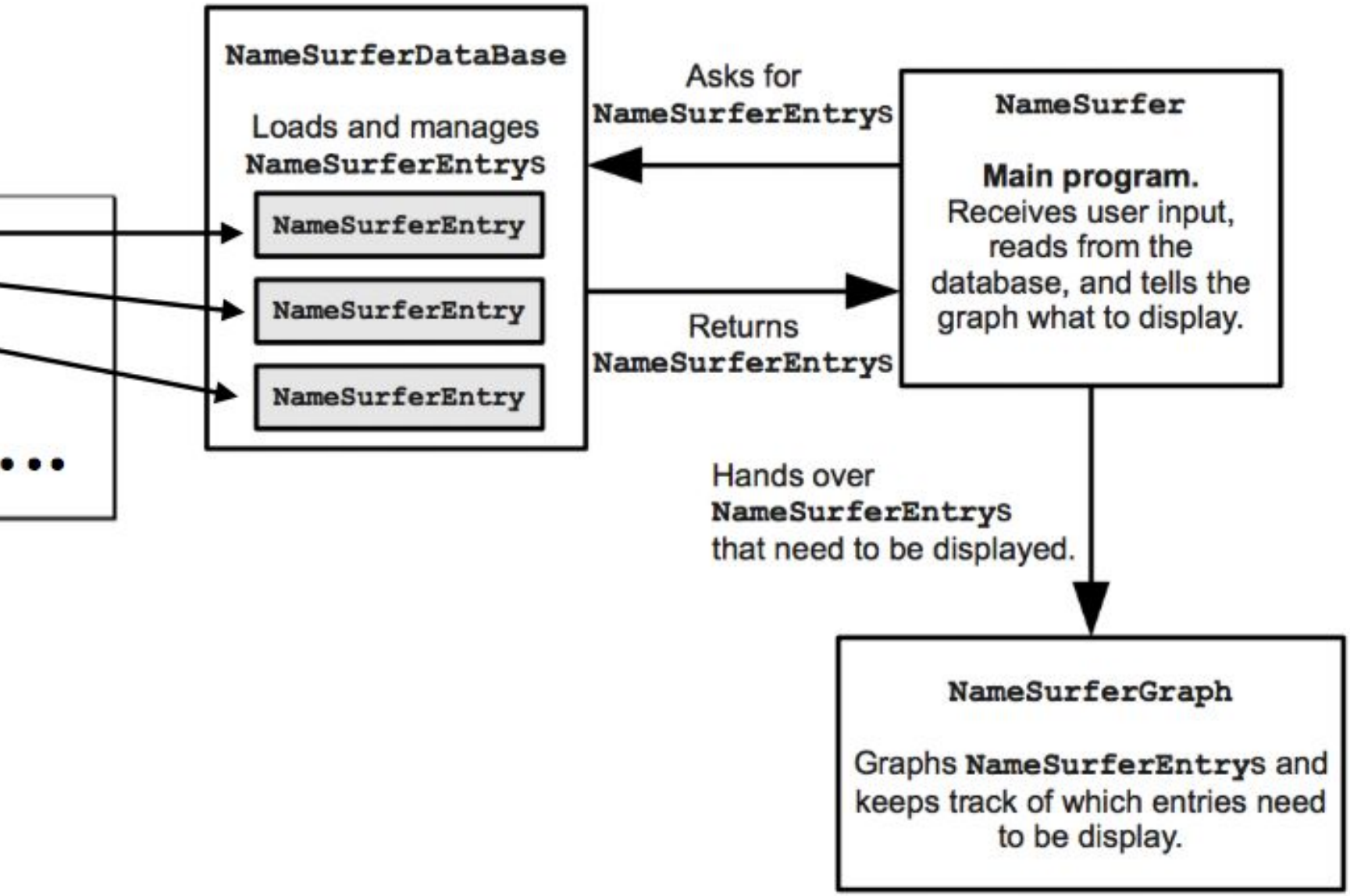
NameSurfer Overview



NameSurfer Overview



NameSurfer Overview



NameSurferEntry

- Contains data for *one name/one line in text file*
- Stores name and popularity ranks for 1900-2000

```
Sam 58 69 99 131 168 236 278 380 467 408 466
```

NameSurferEntry.java

```java
/* 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.
 */
    public NameSurferEntry(String line) {
        // You fill this in //
    }

/* Method: getName() */
/**
 * Returns the name associated with this entry.
 */
    public String getName() {
        // You need to turn this stub into a real implementation //
        return null;
    }

/* 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.
 */
    public int getRank(int decade) {
        // You need to turn this stub into a real implementation //
        return 0;
    }

/* Method: toString() */
/**
 * Returns a string that makes it easy to see the value of a
 * NameSurferEntry.
 */
    public String toString() {
        // You need to turn this stub into a real implementation //
        return "";
    }
```

Parse text line from file to get name and ranks

Return name

Return the rank for the given **number of decades after START_DECADE**.

Return something like:
“Sam [58 60 13 36 36 135 734 3 4 1 2]”

Parsing with split()!

```
Sam 58 69 99 131 168 236 278 380 467 408 466
```

```
// line is a String containing "Sam 58 69 ..."
String[] tokens = line.split(" ");
// tokens[0] = "Sam"
// tokens[1] = "58"
// tokens[2] = "69"
// tokens[3] = "99"
// ...
```

NameSurferDatabase

- Collection of NameSurferEntries
- Responsible for reading in text file and creating NameSurfer entry for each line in the text file
- Responsible for storing all entries, and being able to look up entries by *name* (appropriate data structure? – array, ArrayList, HashMap?)

```
public class NameSurferDataBase implements NameSurferConstants {

/* 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) {
        // You fill this in //
    }

/* 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) {
        // You need to turn this stub into a real implementation //
        return null;
    }
}
```

NameSurferDatabase.java

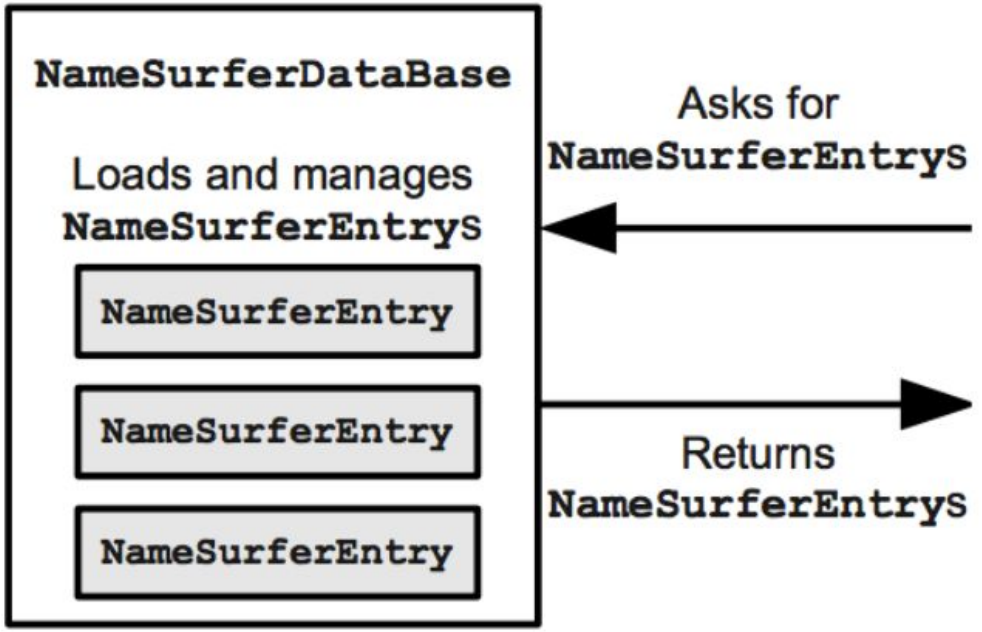
NameSurferDatabase.java

```
// constructor: for each line in the file, create a new
// NameSurferEntry:
String line = rd.readLine();
NameSurferEntry entry = new NameSurferEntry(line);
//Store this NameSurferEntry so it can be retrieved
```

NamesData.txt

```
...
Sam 58 69 99 131 168 236 278 380 467 408 466 997
Samantha 0 0 0 0 0 0 272 107 26 5 7 63
Samara 0 0 0 0 0 0 0 0 0 0 886 0
Samir 0 0 0 0 0 0 0 0 920 0 798 0
Sammie 537 545 351 325 333 396 565 772 930 0 0 0
Sammy 0 887 544 299 202 262 321 395 575 639 755 0
Samson 0 0 0 0 0 0 0 0 0 0 915 0
Samuel 31 41 46 60 61 71 83 61 52 35 28 32
Sandi 0 0 0 0 704 864 621 695 0 0 0 0
Sandra 0 942 606 50 6 12 11 39 94 168 257 962
...
```

rank 0 means the name did not appear in the top 1000 names for that year



NameSurferGraph

- Similar to problem 3 from this week's section
- Responsible for graphing entries
- Resizes when window resizes! (automatic – update() is called whenever window resized)
- Stores all entries currently being graphed so graph can be redrawn when the window is resized
- Different colors for each plot – color sequence cycles around!
- Rank 0 -> use * instead of 0 in graph label
- Rank 0 is at bottom of graph!!

NameSurferGraph.java

```
public class NameSurferGraph extends GCanvas
      implements NameSurferConstants, ComponentListener {
      /**
       * Creates a new NameSurferGraph object that displays the data.
       */
      public NameSurferGraph() {
            addComponentListener(this);
            // You fill in the rest //
      }

      /**
       * Clears the list of name surfer entries stored inside this class.
       */
      public void clear() {
            // You fill this in //
      }

      /* 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) {
            // You fill this in //
      }

      /**
       * 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() {
            // You fill this in //
      }

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

Clear list of graphed entries

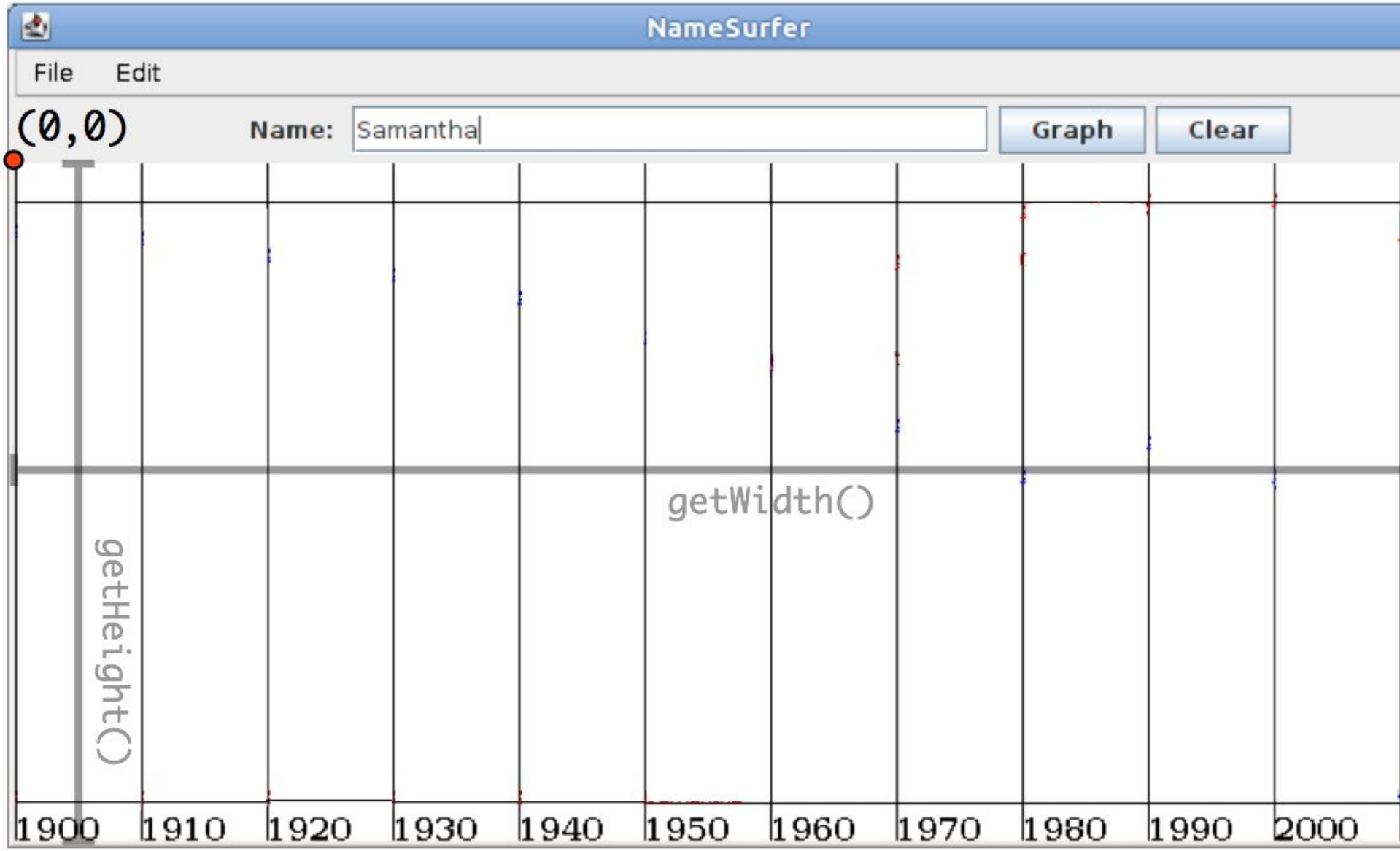
Adds the given entry to the list of graphed entries. Note: DOES NOT ACTUALLY GRAPH IT! update() does that.

Clears screen, then draws grid and all entries.

NameSurferGraph: update()

- Must also call update() when clearing or adding a new item. update() should be doing the drawing! (Why? We need to be able to reconstruct the entire graph)
- in NameSurfer.java (with graph as an instance variable):

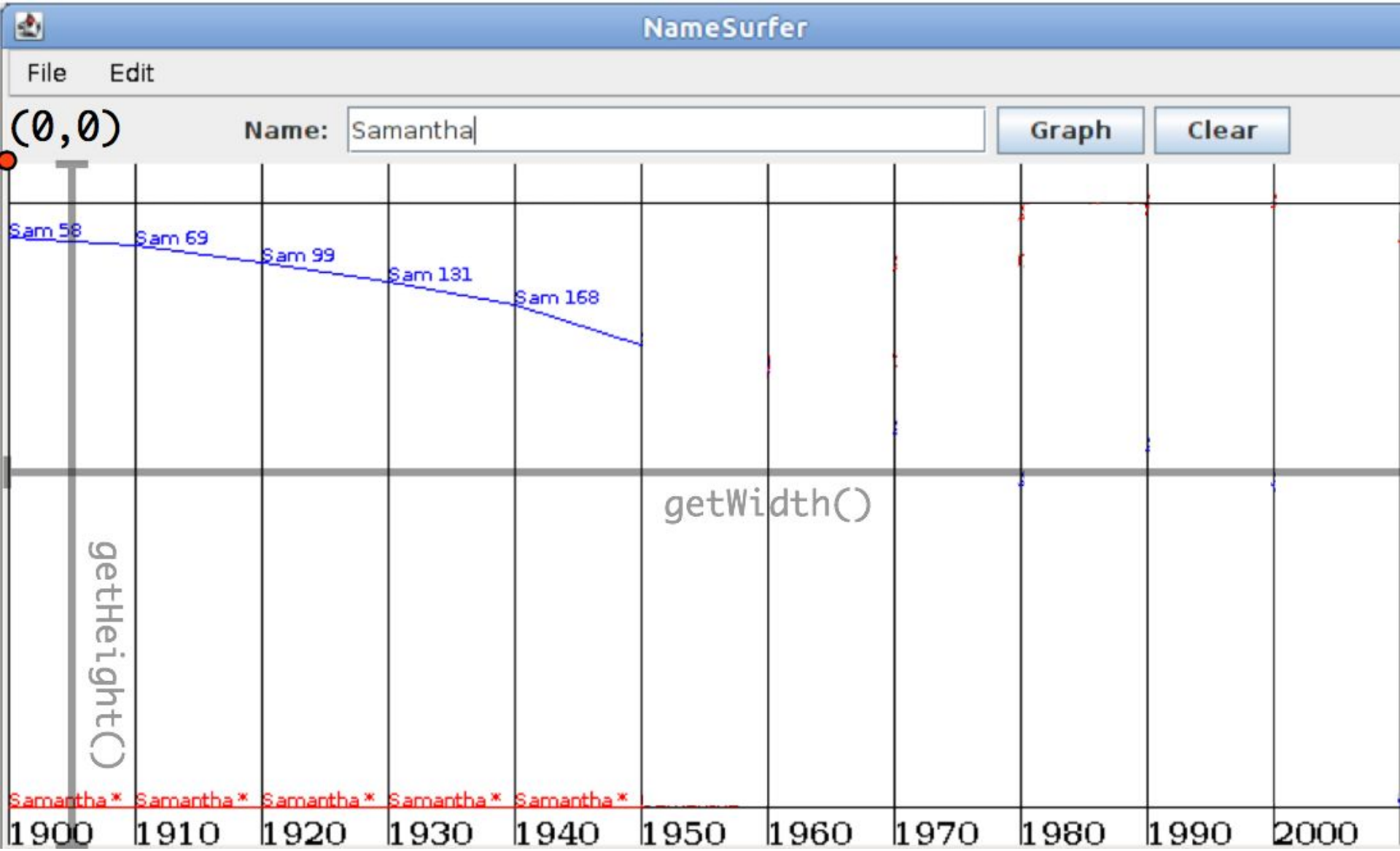
```
graph = new NameSurferGraph(); // in init!
add(graph); // in init!
// later...
graph.add(entry); // graph entry!
graph.update(); // actually draws it!
```


NameSurfer
File
Edit
(0,0)
Name:
Samantha
Graph
Clear
getWidth()
getHeight()
1900
1910
1920
1930
1940
1950
1960
1970
1980
1990
2000

NameSurferGraph: drawing

- Draw lines + GLabels labeling each point
- Remember, rank 0 should be graphed like MAX_RANK! Also, use * instead of rank for the GLabel
- MAX_RANK drawn at bottom of graph, rank 1 drawn at top. All other ranks drawn, equally spaced (e.g. rank MAX_RANK / 2 halfway down the screen)

Partially-drawn Example



Use Constants!

```
/** 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 = 11;

/** 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;
```

Don't use! Use getWidth() and getHeight() instead!!

NameSurfer

- Main program
- Interactors/user input
- Reads from database, tells graph what to draw
- Name entered is **not** case sensitive

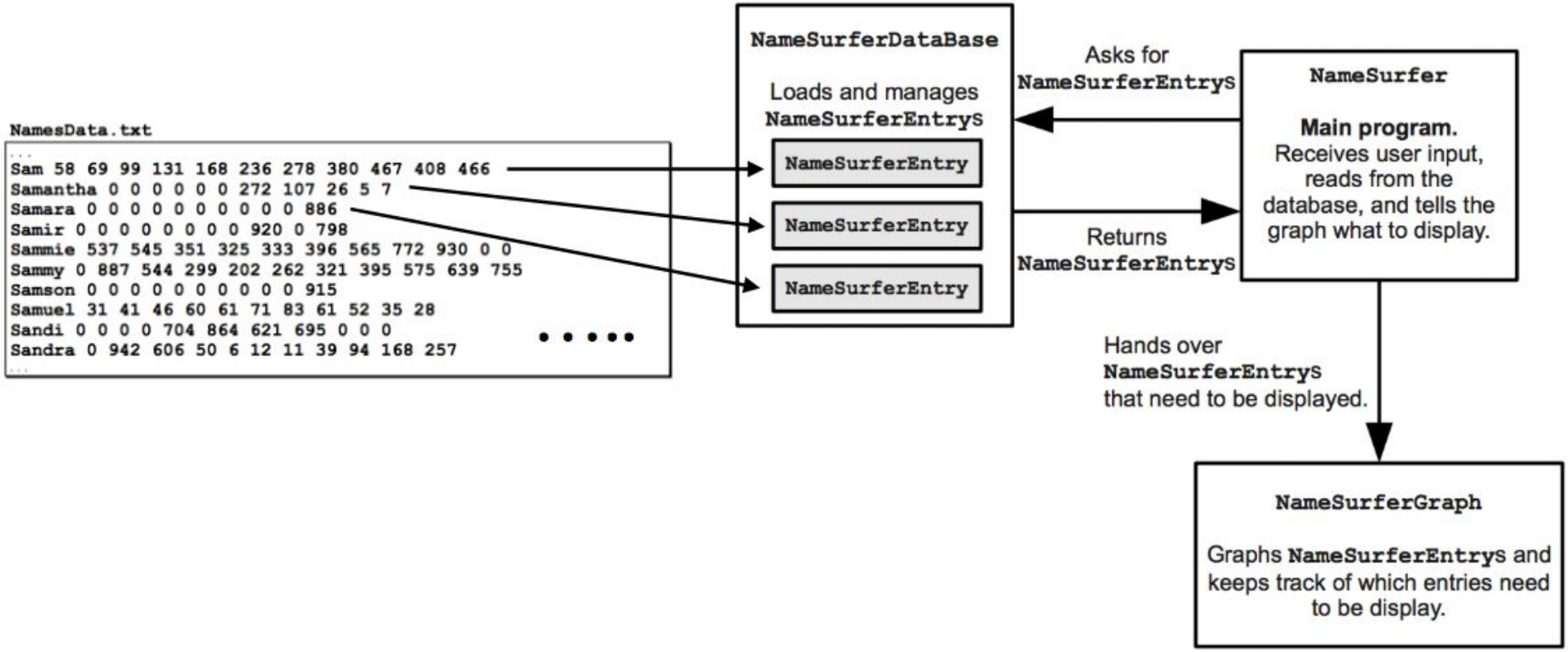
NameSurfer.java

```
public class NameSurfer extends Program implements NameSurferConstants {

/* Method: init() */
/**
 * This method has the responsibility for reading in the data base
 * and initializing the interactors at the top of the window.
 */
    public void init() {
        // You fill this in, along with any helper methods //
    }

/* Method: actionPerformed(e) */
/**
 * This class is responsible for detecting when the buttons are
 * clicked, so you will have to define a method to respond to
 * button actions.
 */
    public void actionPerformed(ActionEvent e) {
        // You fill this in //
    }
}
```

NameSurfer Overview



Tricky Parts

- Null pointer exceptions (use the debugger!)
- OutOfBoundsException
- Off-by-one drawing (notice – 11 decade lines and 11 graph GLabels, but only **10** plot lines for each entry!!)
- Don’t change any of the code given!
  – But add whatever you want :)

Final Tips

- Follow the specifications carefully
- Use suggested milestones
- Extensions!
- Comment!
- Go to the LaIR if you get stuck
- **Incorporate IG feedback!**

- Have fun!