==> ./src/edu/gatech/earthquakes/components/Importer.java <==

```
package edu.gatech.earthquakes.components;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashSet;
import java.util.Locale;
import java.util.Map;
import java.util.Set;

import com.google.common.collect.Maps;
import com.sun.xml.internal.ws.api.streaming.XMLStreamReaderFactory.Default;

import edu.gatech.earthquakes.model.DataRow;
import edu.gatech.earthquakes.model.DataSet;

public class Importer {
	private final static String DATA_LOCATION = ".." + File.separator + "data" +
File.separator;
	private final static String FILENAME = "Catalog.csv";

	public static DataSet importData() {
		Set<DataRow> dataRows = new HashSet<DataRow>();
		try {
			final BufferedReader reader = new BufferedReader(
					new FileReader(new File(DATA_LOCATION
							+ FILENAME)));

			while (reader.ready()) {
				final String line = reader.readLine();
				final String[] data = line.split(",");
				try{
				   dataRows.add(readQuake(data));
				} catch(ArrayIndexOutOfBoundsException aiobe){
					System.err.println(line);
				}
			}
```

```
                reader.close();
        } catch (FileNotFoundException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
        return new DataSet(dataRows);
}

private static DataRow readQuake(final String[] data) throws IOException {
        Map<String, Object> curQuake = Maps.newHashMap();

        String date = data[0];
        curQuake.put(DataRow.DATE, createDate(date));

        String record = data[1];
        curQuake.put(DataRow.RECORD, record);

        Double lat = parseDoubleMissing(data[2]);
        curQuake.put(DataRow.LATTITUDE, lat);

        Double lon = parseDoubleMissing(data[3]);
        curQuake.put(DataRow.LONGITUDE, lon);

        String time = data[4];
        curQuake.put(DataRow.TIME, timeConvert(time));

        String continent = data[5];
        curQuake.put(DataRow.CONTINENT, continentConvert(continent));

        String depth = data[6];
        curQuake.put(DataRow.DEPTH, parseDoubleMissing(depth));

        String momentMagnitude = data[7];
        curQuake.put(DataRow.MOMENT_MAGNITUDE,
                        parseDoubleMissing(momentMagnitude));

        String bodyWaveMagnitude = data[8];
        curQuake.put(DataRow.BODY_WAVE_MAGNITUDE,
                        parseDoubleMissing(bodyWaveMagnitude));

        String surfaceWaveMagnitude = data[9];
        curQuake.put(DataRow.SURFACE_WAVE_MAGNITUDE,
                        parseDoubleMissing(surfaceWaveMagnitude));
```

```
                String localWaveMagnitude = data[10];
                curQuake.put(DataRow.LOCAL_WAVE_MAGNITUDE,
                                parseDoubleMissing(localWaveMagnitude));

                String eventDep = data[31];
                curQuake.put(DataRow.DEPENDENCY, findDependancy(eventDep));

                String mainEventDate = data[32];
                curQuake.put(DataRow.MAIN_DATE, createDate(mainEventDate));

                String eventType = data[33];
                curQuake.put(DataRow.TYPE, findType(eventType));

                return new DataRow(curQuake);
        }

        private static Date createDate(final String yearMonthDay) {
                Date date = null;
                if(("-").equals(yearMonthDay)){
                        date = null;
    } else {
        try {
            date = new SimpleDateFormat("yyyyMMdd", Locale.ENGLISH)
                    .parse(yearMonthDay);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }
                return date;
        }

        private static String timeConvert(final String time) {
                // the input --- is what is passed when data is not there
                return time;
        }


private static DataRow.Continent continentConvert(final String continent) {
    // the input --- is what is passed when data is not there
    switch (continent) {
    case "AF":
        return DataRow.Continent.AFRICA;
    case "AU":
```

```
            return DataRow.Continent.AUSTRALIA;
        case "AS":
            return DataRow.Continent.ASIA;
        case "EU":
            return DataRow.Continent.EURASIA;
        case "IN":
            return DataRow.Continent.INDIA;
        case "NA":
            return DataRow.Continent.NORTH_AMERICA;
        case "SA":
            return DataRow.Continent.SOUTH_AMERICA;
        default :
            return null;
        }
    }

        private static DataRow.Dependency findDependancy(String dep) {
                // the input --- is what is passed when data is not there
                switch (dep) {
                case "I":
                        return DataRow.Dependency.INDEPENDENT;
                case "D":
                        return DataRow.Dependency.DEPENDENT;
                case "P":
                        return DataRow.Dependency.POSSIBLY;
                default :
                   return null;
        }
            }

        private static DataRow.Type findType(String type) {
                // the input --- is what is passed when data is not there
                switch (type) {
                case "tect.":
                        return DataRow.Type.TECT;
                case "deep min.":
                        return DataRow.Type.DEEP_MINING;
                case "mining":
                        return DataRow.Type.MINING;
                case "reservoir":
                        return DataRow.Type.RESERVOIR;
                case "oil field":
                        return DataRow.Type.OIL_FEILD;
                case "-":
```

```
                        return null;
                default :
                   return DataRow.Type.TECT;
        }
           }

           private static Double parseDoubleMissing(String num) {
                Double parsedNumber = null;
                try {
                        parsedNumber = Double.parseDouble(num);
                } catch (Exception e) {
                        parsedNumber = null;
                }
                return parsedNumber;
           }
}

==> ./src/edu/gatech/earthquakes/components/Theme.java <==
package edu.gatech.earthquakes.components;

import java.awt.Color;

public final class Theme {
   public static final int HIGHLIGHTED_COLOR = 0xFFC9D94E;
   private static final int BASE_UI_COLOR = 0xFF00678B;
   private static final int BACKGROUND_COLOR = 0xFFEEEEEE;

   private Theme(){
      //Do nothing
   }

   /**
    * The change applied to values for highlight and darkened compliments, as
    * well as pallette generation.
    */
   private static final float DELTA_V = 0.3f;

   /**
    * Color pallette for drawing all the data.
    */
   private static final int[] pallette = new int[] {
         // continents
         0xff03a688, 0xffc9d94e, 0xfff23535, 0xfff2762e, 0xfff2ac29,
         0xff58838c, 0xff79c7d9,
```

```
        // dependency
        0xffbf996b, 0xfff28972, 0xfff2bc79,

        // types
        0xff0071bc, 0xffbc6f00,  0xff79637e, 0xff9bf2ea,

        0xffce1a53,

        // depth
        0xff1d8f49,

        0xff497358, 0xffd62bd9, 0xffd96aa3, 0xffffe800, };

public static float[] getHSB(final int rgbColor) {
    final float[] hsb = new float[3];
    final Color rgb = new Color(rgbColor);
    Color.RGBtoHSB(rgb.getRed(), rgb.getGreen(), rgb.getBlue(), hsb);
    return hsb;
}

private static int getRGB(float[] hsb) {
    final Color converted = Color.getHSBColor(hsb[0], hsb[1], hsb[2]);
    return converted.getRGB();
}

public static int getBackgroundColor() {
    return BACKGROUND_COLOR;
}

public static int getBaseUIColor() {
    return BASE_UI_COLOR;
}

public static int getDarkUIColor() {
    float[] hsb = getHSB(BASE_UI_COLOR);
    hsb[2] = hsb[2] - DELTA_V;
    if (hsb[2] < 0.0f){
        hsb[2] = 0.0f;
    }
    return getRGB(hsb);
}

public static int getBrightUIColor() {
```

```
    float[] hsb = getHSB(BASE_UI_COLOR);
    hsb[2] = hsb[2] + DELTA_V;
    if (hsb[2] > 1.0f){
        hsb[2] = 1.0f;
    }
    return getRGB(hsb);
}

public synchronized static int getPalletteColor(int userIndex) {
    if (userIndex < 0){
        throw new IllegalArgumentException("Can't have negative index.");
    }
    int index = userIndex % pallette.length;
    return pallette[index];
}

public static int rgba(int rgb, int alpha) {
    return rgb & ((alpha << 24) | 0xFFFFFF);
}

public static int changeSaturation(int rgb, float percentage, boolean relative) {
    float[] hsb = getHSB(rgb);
    if(relative){
        hsb[1] *= percentage;
    } else {
        hsb[1] = percentage;
    }
    if (hsb[1] < 0.0f){
        hsb[1] = 0.0f;
    }
    return getRGB(hsb);
}

public static int changeBrightness(int rgb, float percentage, boolean relative) {
    float[] hsb = getHSB(rgb);
    if(relative){
        hsb[2] *= percentage;
    } else {
        hsb[2] = percentage;
    }
    if (hsb[2] < 0.0f){
        hsb[2] = 0.0f;
    }
    return getRGB(hsb);
```

```
    }

}

==> ./src/edu/gatech/earthquakes/components/Controller.java <==
package edu.gatech.earthquakes.components;

import java.awt.Rectangle;

import processing.core.PApplet;

import com.google.common.eventbus.EventBus;

import edu.gatech.earthquakes.interfaces.Brushable;
import edu.gatech.earthquakes.interfaces.Drawable;
import edu.gatech.earthquakes.interfaces.Filterable;
import edu.gatech.earthquakes.interfaces.Interactable;
import edu.gatech.earthquakes.model.DataSet;
import edu.gatech.earthquakes.model.DeadEventCanary;
import edu.gatech.earthquakes.model.Interaction;
import edu.gatech.earthquakes.vises.AbstractVisualization;
import edu.gatech.earthquakes.vises.Slider;
import edu.gatech.earthquakes.vises.Workspace;

public class Controller {

	private final PApplet parentApplet;


	private final Slider dataslider;

	private final Workspace workspace;

	private int lastWidth, lastHeight;

	public static final EventBus BRUSH_BUS = new EventBus("Brushing Bus");
	public static final EventBus DRAW_BUS = new EventBus("Drawing Bus");
	public static final EventBus FILTER_BUS = new EventBus("Filtering Bus");
	public static final EventBus INTERACT_BUS = new EventBus("Interacting Bus");

	public Controller(final PApplet parent) {
		this.parentApplet = parent;
		final DataSet masterData = Importer.importData();
		lastWidth = lastHeight = 0;
```

```
		setUpCanary();

		int[] test = new int[2012 - 495];
		for (int i = 0; i < test.length; i++) {
			test[i] = i + 495;
		}

		workspace = new Workspace(10, 10, parent.getWidth() - 20,
				parent.getHeight() - 120, masterData);
		registerVisualization(workspace);

		dataslider = new Slider(50, 768 - 100, 924, 50, masterData);
		dataslider.setUndecorated(true);
		dataslider.setDrawInterval(250);
		registerVisualization(dataslider);
	}

	private void setUpCanary(){
		final DeadEventCanary dec = DeadEventCanary.getInstance();
		BRUSH_BUS.register(dec);
		DRAW_BUS.register(dec);
		FILTER_BUS.register(dec);
		INTERACT_BUS.register(dec);
	}

	public static void registerVisualization(final AbstractVisualization av) {
		if (av instanceof Brushable){
	BRUSH_BUS.register(av);
		}
		if (av instanceof Drawable){
		DRAW_BUS.register(av);
		}
		if (av instanceof Filterable){
		FILTER_BUS.register(av);
		}
		if (av instanceof Interactable){
		    INTERACT_BUS.register(av);
		}
	}

	/**
	 * Called at each loop of the animation thread
	 */
```

```
public void refresh() {
	handleInput();
	redrawAll();
}

public void redrawAll() {
	if (parentApplet.width != lastWidth
			|| parentApplet.height != lastHeight) {
		lastWidth = parentApplet.width;
		lastHeight = parentApplet.height;
		windowResized(lastWidth, lastHeight);
	}
	DRAW_BUS.post(parentApplet);
}

private boolean alreadyPressed;

public void handleInput() {
	boolean firstPress = false;
	boolean drag = false;
	boolean released = false;
	if (parentApplet.mousePressed) {
		if (!alreadyPressed) {
			firstPress = true;
		} else {
			drag = true;
		}
		alreadyPressed = true;
	} else {
		if (alreadyPressed == true) {
			released = true;
		}
		alreadyPressed = false;
	}

	Interaction interact = new Interaction(firstPress, drag, released,
			parentApplet);
	INTERACT_BUS.post(interact);
}

public static void applyFilter(final DataSet curDataSet) {
	FILTER_BUS.post(curDataSet);
}
```

```
	public static void applyBrushing(final DataSet curDataSet) {
		BRUSH_BUS.post(curDataSet);
	}

	public void windowResized(int width, int height) {
		dataslider.resizeTo(new Rectangle(50, height - 100, width - 100, 50));
		workspace.resizeTo(new Rectangle(10, 10, width - 10, height - 120));
	}
}

==> ./src/edu/gatech/earthquakes/interfaces/Drawable.java <==
package edu.gatech.earthquakes.interfaces;

import com.google.common.eventbus.Subscribe;

import processing.core.PApplet;

public interface Drawable {

	@Subscribe
	public void drawComponent(PApplet parent);

}

==> ./src/edu/gatech/earthquakes/interfaces/Brushable.java <==
package edu.gatech.earthquakes.interfaces;

import com.google.common.eventbus.Subscribe;

import edu.gatech.earthquakes.model.DataSet;


public interface Brushable {

	@Subscribe
	public void brushData(DataSet newDataSet);

}

==> ./src/edu/gatech/earthquakes/interfaces/Filterable.java <==
package edu.gatech.earthquakes.interfaces;

import com.google.common.eventbus.Subscribe;
```

```
import edu.gatech.earthquakes.model.DataSet;

public interface Filterable {

	@Subscribe
	public void filterBy(DataSet filteredData);

}

==> ./src/edu/gatech/earthquakes/interfaces/Interactable.java <==
package edu.gatech.earthquakes.interfaces;

import com.google.common.eventbus.Subscribe;

import edu.gatech.earthquakes.model.Interaction;

public interface Interactable {

	@Subscribe
	public void handleInput(Interaction interaction);

}

==> ./src/edu/gatech/earthquakes/interfaces/Resizable.java <==
package edu.gatech.earthquakes.interfaces;

import java.awt.Rectangle;

public interface Resizable {

	public void resizeTo(Rectangle bounds);

}

==> ./src/edu/gatech/earthquakes/EarthquakesMain.java <==
package edu.gatech.earthquakes;

import java.awt.GraphicsEnvironment;
import java.awt.Insets;
import java.awt.Rectangle;
import java.io.File;
import java.io.InputStream;

import javax.swing.JFrame;
```

```
import processing.core.PApplet;
import processing.core.PFont;
import edu.gatech.earthquakes.components.Controller;
import edu.gatech.earthquakes.components.Theme;

public class EarthquakesMain extends PApplet {

	private static final long serialVersionUID = 1L;
	private Controller cont;

	public static void main(String[] args) {
		PApplet.main(new String[] { "--present", "EarthquakesMain" });
	}

	public void setup() {
		smooth();

		PFont font = null;
		String fileName = ".." + File.separator + "data" + File.separator
				+ "fonts" + File.separator + "Quicksand-Regular.ttf";
		InputStream i = createInput(fileName);
		font = createFont(fileName, 24);

		textFont(font);
		Rectangle bounds = GraphicsEnvironment.getLocalGraphicsEnvironment()
				.getMaximumWindowBounds();
		JFrame sample = new JFrame();
		sample.pack();
		Insets insets = sample.getInsets();
		int wwidth = (bounds.width - insets.left - insets.right);
		int wheight = (bounds.height - insets.top - insets.bottom);
		sample.dispose();
		size(wwidth, wheight - 48); // Offset applet bottom and top
		cont = new Controller(this);
	}

	public void draw() {
		background(Theme.getBackgroundColor());
		cont.refresh();
	}
}

==> ./src/edu/gatech/earthquakes/util/UIUtils.java <==
```

```
package edu.gatech.earthquakes.util;

import java.lang.reflect.Field;

import processing.core.PApplet;
import processing.core.PGraphics;

public final class UIUtils {

    private UIUtils(){
        //do nothing
    }

    public static void roundRect(final int x, final int y, final int w,
            final int h, final int radius, final PApplet parent) {
        int modifiedRadius = radius;
        final PApplet p = parent;
        if (p == null) {
            return;
        }

        final int stroke = UIUtils.getStrokeColor(p);

        // Sanitize data
        if (radius > w / 2){
            modifiedRadius = w / 2;
        }
        if (radius > h / 2){
            modifiedRadius = h / 2;
        }

        // Draw all rectangles first
        parent.noStroke();
        // Center
        p.rect(x + modifiedRadius, y + modifiedRadius, w - 2 * modifiedRadius,
                h - 2 * modifiedRadius);
        // Top
        p.rect(x + modifiedRadius, y, w - 2 * modifiedRadius, modifiedRadius);
        // Bottom
        p.rect(x + modifiedRadius, y + h - modifiedRadius, w - 2
                * modifiedRadius, modifiedRadius);
        // Left
        p.rect(x, y + modifiedRadius, modifiedRadius, h - 2 * modifiedRadius);
        // Right
```

```
    p.rect(x + w - modifiedRadius, y + modifiedRadius, modifiedRadius, h
            - 2 * modifiedRadius);

    // Draw all Corners
    parent.stroke(stroke);
    // Top Left
    p.arc(x + modifiedRadius, y + modifiedRadius, 2 * modifiedRadius,
            2 * modifiedRadius, PApplet.PI, PApplet.PI + PApplet.HALF_PI);
    // Top Right
    p.arc(x + w - modifiedRadius, y + modifiedRadius, 2 * modifiedRadius,
            2 * modifiedRadius, PApplet.PI + PApplet.HALF_PI,
            2 * PApplet.PI);
    // Bottom Left
    p.arc(x + modifiedRadius, y + h - modifiedRadius, 2 * modifiedRadius,
            2 * modifiedRadius, PApplet.HALF_PI, PApplet.PI);
    // Bottom Right
    p.arc(x + w - modifiedRadius, y + h - modifiedRadius,
            2 * modifiedRadius, 2 * modifiedRadius, 0, PApplet.HALF_PI);

    // Draw all Lines
    // Top
    p.line(x + modifiedRadius, y, x + w - modifiedRadius, y);
    // Bottom
    p.line(x + modifiedRadius, y + h, x + w - modifiedRadius, y + h);
    // Left
    p.line(x, y + modifiedRadius, x, y + h - modifiedRadius);
    // Right
    p.line(x + w, y + modifiedRadius, x + w, y + h - modifiedRadius);
}

public static int getFillColor(PApplet papplet) {
    try {
        Field graphicsField = PApplet.class.getDeclaredField("g");
        PGraphics graphics = (PGraphics) graphicsField.get(papplet);

        return graphics.fillColor;
    } catch (NoSuchFieldException nsfe) {
        nsfe.printStackTrace();
        return 0;
    } catch (IllegalAccessException iae) {
        iae.printStackTrace();
        return 0;
    }
}
```

```
    public static int getStrokeColor(PApplet papplet) {
        try {
            Field graphicsField = PApplet.class.getDeclaredField("g");
            PGraphics graphics = (PGraphics) graphicsField.get(papplet);

            return graphics.strokeColor;
        } catch (NoSuchFieldException nsfe) {
            nsfe.printStackTrace();
            return 0;
        } catch (IllegalAccessException iae) {
            iae.printStackTrace();
            return 0;
        }
    }

}

==> ./src/edu/gatech/earthquakes/vises/DetailedInfo.java <==
package edu.gatech.earthquakes.vises;

import java.awt.Rectangle;
import java.io.IOException;
import java.net.UnknownHostException;
import java.security.NoSuchAlgorithmException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.Locale;
import java.util.Locale.Category;

import processing.core.PApplet;

import com.google.common.eventbus.Subscribe;

import edu.gatech.earthquakes.components.Theme;
import edu.gatech.earthquakes.interfaces.Brushable;
import edu.gatech.earthquakes.model.DataRow;
import edu.gatech.earthquakes.model.DataSet;
import edu.gatech.earthquakes.web.CustomSearch;

public class DetailedInfo extends Individual implements Brushable {
    // Display Strings
    private static final String NUMBER_OF_RESULTS = "Number of Results";
```

```
private static final String TITLE = "Top result title";

// Calculated Data
private volatile int numResults;
private volatile String title;

// To keep from spamming custom search
private volatile boolean searching;

// Formatting
private int xPadding;
private int yPadding;
private int textSize;

public DetailedInfo(int x, int y, int w, int h, DataRow displayData) {
   super(x, y, w, h, displayData, "Detailed Information");
   setFormatting(w, h);
   recalculateNumResults();
   searching = false;
}

private void setFormatting(int width, int height) {
   xPadding = width / 50;
   yPadding = width / 50;
   textSize = Math.min(width / 20, height / 20);
}

private void recalculateNumResults() {
   if (!searching) {
      new Thread(new Runnable() {

         @Override
         public void run() {
            try {
               numResults = CustomSearch.getTotalCount(CustomSearch
                     .getInstance().getQuery(getWebQuery()));
               title = CustomSearch.getTitles(0, CustomSearch
                     .getInstance().getQuery(getWebQuery()));
               searching = false;
            } catch (UnknownHostException uhe) {
               numResults = -1;
               title = "";
               System.err.println("Unknown Host: " + uhe.getMessage());
            } catch (NoSuchAlgorithmException | IOException e) {
```

```
                    numResults = -1;
                    title = "";
                    e.printStackTrace();
                } catch (Exception e){
                    numResults = -1;
                    title = "";
                } finally {
                    searching = false;
                }
            }
        }).run(); // Should be threaded in the end, but having I/O issues
                  // right now.
    }
  }

  public void drawComponent(PApplet parent) {
    super.drawComponent(parent);
    PApplet p = parent;
    p.stroke(Theme.getBaseUIColor());
    p.strokeWeight(2);

    p.textAlign(PApplet.LEFT);
    p.textSize(textSize);
    String displayOutput = getDisplayString();
    // magical 2's are to keep text from drawing outside the box
    p.text(displayOutput, x + xPadding, y + yPadding, x + w - 2*xPadding, y + h - 2*yPadding -
textSize);
  }

  private String getDisplayString() {
    StringBuilder sb = new StringBuilder();

    sb.append(DataRow.MOMENT_MAGNITUDE);
    sb.append(": ");
    sb.append(displayData.getValue(DataRow.MOMENT_MAGNITUDE));
    sb.append('\n');

    sb.append(DataRow.LATTITUDE);
    sb.append(": ");
    sb.append(displayData.getValue(DataRow.LATTITUDE));
    sb.append('\n');

    sb.append(DataRow.LONGITUDE);
    sb.append(": ");
```

```
    sb.append(displayData.getValue(DataRow.LONGITUDE));
    sb.append('\n');

    sb.append(DataRow.CONTINENT);
    sb.append(": ");
    sb.append(displayData.getValue(DataRow.CONTINENT));
    sb.append('\n');

    sb.append(DataRow.DEPTH);
    sb.append(": ");
    sb.append(displayData.getValue(DataRow.DEPTH));
    sb.append("\n");

    sb.append(DataRow.DATE);
    sb.append(": ");

    Calendar cal = Calendar.getInstance();
    Locale loc = Locale.getDefault(Category.DISPLAY);
    cal.setTime((Date) displayData.getValue(DataRow.DATE));
    String displayMonth = cal.getDisplayName(Calendar.MONTH,
          Calendar.SHORT, loc);
    sb.append(displayMonth);
    sb.append(" ");
    sb.append(cal.get(Calendar.DAY_OF_MONTH));
    sb.append(", ");
    sb.append(cal.get(Calendar.YEAR));
    sb.append("\n");

    if(numResults >=0){
       sb.append("Google Search Results for Earthquake: ");
       sb.append(numResults);
       sb.append('\n');
    }
    if(title.length() > 0){
       sb.append(TITLE);
       sb.append(": ");
       sb.append(title);
       sb.append('\n');
    }

    return sb.toString();
}

private String getWebQuery() {
```

```
        StringBuilder sb = new StringBuilder();
        Date d = (Date) displayData.getValue(DataRow.DATE);
        Calendar cal = Calendar.getInstance();
        cal.setTime(d);
        sb.append(new SimpleDateFormat("MMM").format(d));
        sb.append(" ");
        sb.append(cal.get(Calendar.DAY_OF_MONTH));
        sb.append(" ");
        sb.append(cal.get(Calendar.YEAR));
        sb.append(" ");
        sb.append(displayData.getValue(DataRow.CONTINENT));
        sb.append(" ");
        sb.append("Earthquake");
        return sb.toString();
    }

    @Override
    @Subscribe
    public void brushData(DataSet ds) {
        if (!ds.getDatum().isEmpty() && ds.getDatum().size() == 1) {
            // do my things
            displayData = ds.getDatum().iterator().next();
            recalculateNumResults();
        }
    }

    @Override
    public void resizeTo(Rectangle bounds) {
        super.resizeTo(bounds);
        setFormatting(bounds.width, bounds.height);
    }

}

==> ./src/edu/gatech/earthquakes/vises/AbstractVisualization.java <==
package edu.gatech.earthquakes.vises;

import java.awt.Rectangle;

import processing.core.PApplet;
import edu.gatech.earthquakes.components.Theme;
import edu.gatech.earthquakes.interfaces.Drawable;
import edu.gatech.earthquakes.interfaces.Resizable;
```

```
public abstract class AbstractVisualization implements Drawable, Resizable {

    private Rectangle fb; // FrameBounds
    protected int x, y, w, h;
    protected int buffer = 20;
    private String title;
    private boolean undecorated;

    private final int FRAME_TOP = 30, FRAME_BOTTOM = 2, FRAME_LEFT = 2,
            FRAME_RIGHT = 2, CORNER_RADIUS = 25, BASE_INSET = 2;

    public AbstractVisualization(int x, int y, int w, int h) {
        this(x, y, w, h, "Abstract Vis - FIX ME");
    }

    public AbstractVisualization(int x, int y, int w, int h, String title) {
        this(x, y, w, h, title, false);
    }

    public AbstractVisualization(int x, int y, int w, int h, String title,
            boolean undecorated) {
        fb = new Rectangle(x, y, w, h);
        this.title = title;
        this.undecorated = undecorated;
        recalculateInsets();
    }

    public void drawComponent(PApplet parent) {
        if (!undecorated) {
            parent.noStroke();
            parent.fill(Theme.rgba(Theme.getBrightUIColor(), 0x66));
            parent.rect(fb.x + BASE_INSET + CORNER_RADIUS, fb.y + BASE_INSET,
                    fb.width - 2 * BASE_INSET - CORNER_RADIUS, CORNER_RADIUS);
            parent.rect(fb.x + BASE_INSET, fb.y + BASE_INSET + CORNER_RADIUS,
                    fb.width - 2 * BASE_INSET, FRAME_TOP - CORNER_RADIUS);

            parent.stroke(Theme.getBaseUIColor());
            parent.strokeWeight(2);
            parent.arc(fb.x + BASE_INSET + CORNER_RADIUS, fb.y + BASE_INSET
                    + CORNER_RADIUS, 2 * CORNER_RADIUS, 2 * CORNER_RADIUS,
                    PApplet.PI, PApplet.PI + PApplet.HALF_PI);

            parent.noFill();
            parent.strokeCap(PApplet.ROUND);
```

```
        parent.strokeJoin(PApplet.ROUND);
        // border around left, bottom, right;
        parent.beginShape();
        parent.vertex(fb.x + BASE_INSET, fb.y + BASE_INSET + CORNER_RADIUS);
        parent.vertex(fb.x + BASE_INSET, fb.y + fb.height - BASE_INSET);
        parent.vertex(fb.x + fb.width - BASE_INSET, fb.y + fb.height
                - BASE_INSET);
        parent.vertex(fb.x + fb.width - BASE_INSET, fb.y + BASE_INSET);
        parent.vertex(fb.x + BASE_INSET + CORNER_RADIUS, fb.y + BASE_INSET);
        parent.endShape();
        parent.line(fb.x + BASE_INSET, fb.y + FRAME_TOP, fb.x + fb.width
                - 2 * BASE_INSET, fb.y + FRAME_TOP);

        parent.fill(Theme.getDarkUIColor());
        parent.textSize(FRAME_TOP - 8);
        parent.textAlign(PApplet.LEFT);
        parent.text(title, fb.x + BASE_INSET + CORNER_RADIUS, fb.y,
                fb.width - 2 * BASE_INSET - CORNER_RADIUS, FRAME_TOP);
    }
}

@Override
public void resizeTo(Rectangle bounds) {
    fb = (Rectangle) bounds.clone();
    recalculateInsets();
}

public void setUndecorated(boolean undecorated) {
    this.undecorated = undecorated;
}

private void recalculateInsets() {
    if (undecorated) {
        this.x = fb.x;
        this.y = fb.y;
        this.w = fb.width;
        this.h = fb.height;
    } else {
        this.x = fb.x + FRAME_LEFT;
        this.y = fb.y + FRAME_TOP;
        this.w = fb.width - (FRAME_LEFT + FRAME_RIGHT);
        this.h = fb.height - (FRAME_TOP + FRAME_BOTTOM);
    }
}
```

```
    public String getTitle() {
        return title;
    }
}

==> ./src/edu/gatech/earthquakes/vises/NestedCirclePlot.java <==
package edu.gatech.earthquakes.vises;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import java.util.TreeSet;

import processing.core.PApplet;
import edu.gatech.earthquakes.components.Theme;
import edu.gatech.earthquakes.interfaces.Filterable;
import edu.gatech.earthquakes.model.DataComparator;
import edu.gatech.earthquakes.model.DataRow;
import edu.gatech.earthquakes.model.DataSet;

public class NestedCirclePlot extends Aggregate implements Filterable {
    // the thing that will be grouped by after location
    private String dataType;
    private DataComparator dataComp;
    private Map<String, Set<TypeCount>> computedValues;
    private double maxVal = 0;
    private int offset = 15;
    private TreeSet<TypeCount> totals;
    private int numTotalQuakes;

    public NestedCirclePlot(int x, int y, int w, int h, DataSet displayData,
            String dataType, int numTotalQuakes) {
        super(x, y, w, h, displayData, "Type by Continent");
        this.dataType = dataType;
        this.numTotalQuakes = numTotalQuakes;

        DataComparator.CompareCategories category = null;
        switch (dataType) {
            case DataRow.DEPENDENCY:
                category = DataComparator.CompareCategories.DEPENDENCY;
```

```
            break;
        case DataRow.TYPE:
            category = DataComparator.CompareCategories.TYPE;
            break;
    }

    dataComp = new DataComparator(
            DataComparator.CompareCategories.CONTINENT, category);

    computeNominalData();

}

public void drawComponent(PApplet parent) {
    super.drawComponent(parent);

    float drawY = y + h - buffer;
    float maxCircleRadius = getCircleRadius(maxVal);

    boolean right = false;
    parent.noStroke();

    parent.textSize(offset * 3 / 4);
    parent.textAlign(PApplet.CENTER);

    for (String country : computedValues.keySet()) {

        //draw the colored square for the country and write the country name
        parent.noStroke();
        parent.fill(DataRow.getColorFor(country));
        if (right) {
            parent.rect(x + w / 2 + offset / 2,
                    drawY - maxCircleRadius * 2, maxCircleRadius * 2,
                    maxCircleRadius * 2);
            parent.fill(0);
            parent.text(country, x + w / 2 + offset / 2 + maxCircleRadius,
                    drawY + offset * 3 / 4);
        } else {
            parent.rect(x + w / 2 - offset / 2 - maxCircleRadius * 2, drawY
                    - maxCircleRadius * 2, maxCircleRadius * 2,
                    maxCircleRadius * 2);
            parent.fill(0);
            parent.text(country, x + w / 2 - offset / 2 - maxCircleRadius,
                    drawY + offset * 3 / 4);
```

```
        }

        parent.strokeWeight(1);
        //draw the circles for the country
        for (TypeCount t : computedValues.get(country)) {

            parent.fill(Theme.changeSaturation(
                    DataRow.getColorFor(t.getType()), .5f, true));

            parent.stroke(0xff555555);

            float radius = getCircleRadius(t.getCount());
            if (right)
                parent.ellipse(x + w / 2 + offset / 2 + maxCircleRadius, drawY - radius, radius * 2,
radius * 2);
            else
                parent.ellipse(x + w / 2 - offset / 2 - maxCircleRadius, drawY - radius, radius * 2,
radius * 2);
        }

        if (right)
            drawY -= maxCircleRadius * 2 + offset;
        right = !right;
    }
    //draw the outline of the percentage square
    parent.noFill();
    parent.noStroke();
    parent.rect(x + w / 2 + offset / 2, drawY - maxCircleRadius * 2, maxCircleRadius * 2,
maxCircleRadius * 2);

    //draw the total percentages (upper right corner square)
    for (TypeCount t : totals) {
        int color = Theme.changeSaturation(
                DataRow.getColorFor(t.getType()), .5f, true);
        parent.fill(color);
        float heightPercent = t.getCount() / (float) numTotalQuakes;
        parent.rect(x + w / 2 + offset / 2, drawY - heightPercent
                * maxCircleRadius * 2, maxCircleRadius * 2, heightPercent
                * maxCircleRadius * 2);
        drawY -= heightPercent * maxCircleRadius * 2;
    }

  }
```

```
/**
 * Calculates the radius of the circle based on the number of earthquakes that
 * circle represents
 *
 * @param count
 * @return
 */
private float getCircleRadius(double count) {
    float maxDiameter = Math.min(
            (float) ((h - buffer * 2 - offset
                    * (Math.ceil(computedValues.size() / 2.0) - 1)) / (Math
                    .ceil(computedValues.size() / 2.0))),
            (w - buffer * 2 - offset * 2) / 2);

    double maxArea = Math.PI * Math.pow(maxDiameter / 2, 2);

    float area = (float) (maxArea * count / maxVal);
    return (float) (Math.sqrt(area / Math.PI));
}

private void computeMaxVal() {
    for (Set<TypeCount> countryData : computedValues.values()) {
        for (TypeCount t : countryData) {
            if (t.getCount() > maxVal)
                maxVal = t.getCount();
        }
    }
}

private void calculateTotals() {
    HashMap<String, Integer> counts = new HashMap<String, Integer>();

    for (Set<TypeCount> countryData : computedValues.values()) {
        for (TypeCount t : countryData) {
            if (counts.containsKey(t.getType())) {

                int count = counts.get(t.getType());
                counts.put(t.getType(), t.getCount() + count);
            } else
                counts.put(t.getType(), t.getCount());
        }
    }

    totals = new TreeSet<TypeCount>();
```

```
    for (String type : counts.keySet()) {
        totals.add(new TypeCount(type, counts.get(type)));
    }
}

private void computeNominalData() {
    // populate the computed values with the continents and the set
    computedValues = new TreeMap<String, Set<TypeCount>>();
    for (DataRow.Continent c : DataRow.Continent.values()) {
        computedValues.put(c.toString(), new TreeSet<TypeCount>());
    }

    ArrayList<DataRow> list = new ArrayList<DataRow>(displayData.getDatum());
    Collections.sort(list, dataComp);

    String type = null;
    String continent = null;
    if (list.size() > 0) {
        type = list.get(0).getValue(dataType).toString();
        continent = list.get(0).getValue(DataRow.CONTINENT).toString();
    }

    int count = 0;

    for (DataRow quake : list) {
        // get continent and type out of the current quake
        String curContinent = quake.getValue(DataRow.CONTINENT).toString();
        String curType = quake.getValue(dataType).toString();

        // if we've come to data from a new continent
        if (!curContinent.equals(continent)) {
            computedValues.get(continent).add(new TypeCount(type, count));
            count = 1;
            type = curType;
            continent = curContinent;
        } else {
            if (curType.equals(type)) {
                count++;
            } else {
                computedValues.get(curContinent).add(new TypeCount(type, count));
                count = 1;
                type = curType;
            }
```

```
            }
        }
        if(continent != null)
            computedValues.get(continent).add(new TypeCount(type, count));

        calculateTotals();
        computeMaxVal();
    }

    @Override
    public void filterBy(DataSet filteredData) {
        this.displayData = filteredData;
        computeNominalData();
    }

    private class TypeCount implements Comparable<TypeCount> {
        private String type;
        private int count;

        public TypeCount(String type, int count) {
            this.type = type;
            this.count = count;
        }

        public int getCount() {
            return count;
        }

        public int compareTo(TypeCount t) {
            return t.count - count;
        }

        public String getType() {
            return type;
        }
    }

}

==> ./src/edu/gatech/earthquakes/vises/Aggregate.java <==
package edu.gatech.earthquakes.vises;

import edu.gatech.earthquakes.model.DataSet;
```

```
public abstract class Aggregate extends Multi{

	//the data variable we are organizing the quakes based off of
	@SuppressWarnings("unused")
	private String aggregator;

	public Aggregate(int x, int y, int w, int h, DataSet displayData) {
		this(x, y, w, h, displayData, "Aggregate - FIX ME");
	}

	public Aggregate(int x, int y, int w, int h, DataSet displayData, String title) {
		super(x, y, w, h, displayData, title);
	}
}

==> ./src/edu/gatech/earthquakes/vises/DepthPlot.java <==
package edu.gatech.earthquakes.vises;

import java.awt.Rectangle;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashSet;

import processing.core.PApplet;

import edu.gatech.earthquakes.components.Controller;
import edu.gatech.earthquakes.components.Theme;
import edu.gatech.earthquakes.interfaces.Filterable;
import edu.gatech.earthquakes.interfaces.Interactable;
import edu.gatech.earthquakes.model.DataRow;
import edu.gatech.earthquakes.model.DataSet;
import edu.gatech.earthquakes.model.Interaction;

public class DepthPlot extends Multi implements Filterable, Interactable {
	private float[] quakeRadii;
	private float[][] drawingCoordinates;

	private Date[] timeRange;
	private double[] depthRange;
	private double[] magRange;

	private int highlightedIndex;
	private int numberMissing;
```

```
        private int textSize;

        public DepthPlot(int x, int y, int w, int h, DataSet displayData) {
                super(x, y, w, h, displayData, "Depth vs Time");
                calculateRanges();
                calculateDrawingValues();
                textSize = Math.min(w/20, h/20);
        }

public void drawComponent(PApplet parent) {
    super.drawComponent(parent);
    parent.strokeWeight(1);
    drawAxes(parent);
    for (int i = 0; i < drawingCoordinates.length; i++) {
        if (drawingCoordinates[i][1] != y) {
            int color = DataRow.getColorFor(DataRow.DEPTH);

            float loc = (drawingCoordinates[i][1] - y - buffer) / h;
            color = Theme.changeSaturation(color, 1 - loc, false);

            if (i == highlightedIndex) {
                parent.fill(Theme.rgba(Theme.HIGHLIGHTED_COLOR, 150));
                parent.stroke(Theme.HIGHLIGHTED_COLOR);
            } else {
                float brightness = PApplet.map(loc, 0f, 1f, 0f, 0.5f);
                color = Theme.changeBrightness(color, 0.75f - brightness,
                        false);
                parent.fill(Theme.rgba(color, 150));
                parent.stroke(color);
            }
            parent.ellipse(drawingCoordinates[i][0],
                    drawingCoordinates[i][1], quakeRadii[i] * 2,
                    quakeRadii[i] * 2);
        }
    }

    parent.textAlign(PApplet.LEFT);
    parent.textSize(textSize);
    parent.fill(Theme.getDarkUIColor());
    String displayOutput = "# missing depth: " + numberMissing;
    parent.text(displayOutput, x + buffer, y + h - 2*buffer, x + w - 2*buffer, y + h);
```

```
	}


	private float getCircleRadius(double mag) {
		float minDiameter = w/35;
		float maxDiameter = w/15;
		double maxArea = Math.PI*Math.pow(maxDiameter/2, 2);
		double minArea = Math.PI*Math.pow(minDiameter/2, 2);

		float area = (float) ((maxArea - minArea) * (mag - magRange[0]) / (magRange[1]-
magRange[0]) + minArea);
		return (float)(Math.sqrt(area/Math.PI));
	}

	private void calculateDrawingValues(){
	   numberMissing = 0;
		float xoffset = (w-2*buffer)/(float)displayData.getDatum().size();

		drawingCoordinates = new float[displayData.getDatum().size()][2];
		quakeRadii = new float[displayData.getDatum().size()];
		int index = 0;
		for(DataRow d: displayData){
			//calculate the x coordinate
			drawingCoordinates[index][0] = x+buffer+xoffset*index;

			//calculate the y coordinate
	if(d.getValue(DataRow.DEPTH) != null)
				drawingCoordinates[index][1] = y+calculateY((double)
d.getValue(DataRow.DEPTH));
			else{
				drawingCoordinates[index][1] = y;
				numberMissing++;
			}


			if(d.getValue(DataRow.MOMENT_MAGNITUDE) != null)
				quakeRadii[index] = getCircleRadius((double)
d.getValue(DataRow.MOMENT_MAGNITUDE));
			else{
				quakeRadii[index] = (float) magRange[0];
			}

			index++;
```

```
		}
	}

	private float calculateY(double depth){
		return (float) ((h-buffer*2)*(depth-depthRange[0])/(depthRange[1]-depthRange[0])
+ buffer);
	}

	private void calculateRanges(){
		timeRange = new Date[2];
		depthRange = new double[2];
		magRange = new double[2];

		for(DataRow d : displayData){
			//get the data from the current quake
			Date curDate = (Date) d.getValue(DataRow.DATE);
			double curDepth = depthRange[0];
			double curMag = magRange[0];

			if(d.getValue(DataRow.DEPTH)!= null){
				curDepth = (double) d.getValue(DataRow.DEPTH);
			}
			if(d.getValue(DataRow.MOMENT_MAGNITUDE) != null){
				curMag = (double)d.getValue(DataRow.MOMENT_MAGNITUDE);
			}
			//if this is the first thing we've hit, set everything to the current quake
			if(timeRange[0] == null){
				timeRange[0] = curDate;
				timeRange[1] = curDate;
				depthRange[0] = curDepth;
				depthRange[1] = curDepth;
				magRange[0] = curMag;
				magRange[1] = curMag;
			}

			//check the time ranges
			if(curDate.before(timeRange[0]))
				timeRange[0] = curDate;
			else if(curDate.after(timeRange[1]))
				timeRange[1] = curDate;

			//check the depth ranges
			if(curDepth < depthRange[0])
				depthRange[0] = curDepth;
```

```
                else if(curDepth > depthRange[1])
                        depthRange[1] = curDepth;

                //check the magnitude ranges
                if(curMag < magRange[0])
                        magRange[0] = curMag;
                else if(curMag > magRange[1])
                        magRange[1] = curMag;
        }
}

private void drawAxes(PApplet parent){
   int verticalOffset = h/20;
   int depthOffset = (int)(depthRange[1]-depthRange[0])/verticalOffset;
   parent.stroke(0xaa);
   parent.fill(0);
   parent.textSize(verticalOffset/3);
   parent.textAlign(PApplet.CENTER);
   for(int i=0; i< h-buffer*2; i+= verticalOffset ){
      parent.line(x+buffer-2, y+buffer+i, x+w-buffer, y+buffer+i);
      parent.pushMatrix();
      parent.translate(x+buffer/2, y+buffer+i);
      parent.rotate(-PApplet.PI / 2);
      parent.text(i*depthOffset+"", 0 , 0);
      parent.popMatrix();
   }
}

@Override
public void filterBy(DataSet filteredData) {
        this.displayData = filteredData;
        //calculateRanges();
        calculateDrawingValues();

}

public void resizeTo(Rectangle bounds) {
        super.resizeTo(bounds);

        //calculateRanges();
        calculateDrawingValues();
}

@Override
```

```
    public void handleInput(Interaction interaction) {

        int mX = interaction.getParentApplet().mouseX;
        int mY = interaction.getParentApplet().mouseY;
        //check if mouse in within the vis
        if (mX > x && mX < x + w && mY > y && mY < y + h) {

            float[] depths = getDepths();
            float[] mag = getMagnitudes();
            boolean found = false;
            int lastDist = Integer.MAX_VALUE;
            int distanceToMouse = Integer.MAX_VALUE;

            for (int i = 0; i < depths.length; i++) {
                distanceToMouse = (int) Math.round(Math.sqrt(
                        (Math.abs(mX - drawingCoordinates[i][0]) + Math.abs(mY -
drawingCoordinates[i][1]))
                ));
                //if the mouse is within radius of the current earthquake
                if(lastDist > distanceToMouse
                        && Math.abs(mY - drawingCoordinates[i][1]) < getCircleRadius(mag[i])
                        && Math.abs(mX - drawingCoordinates[i][0]) < getCircleRadius(mag[i])) {
                    highlightedIndex = i;
                    ArrayList<DataRow> rowList = new ArrayList<>(displayData.getDatum());
                    Controller.BRUSH_BUS.post(new DataSet(rowList.get(i)));
                    found = true;
                    lastDist = distanceToMouse;
                }

                if (!found) {
                    highlightedIndex = -1;
                    Controller.BRUSH_BUS.post(new DataSet(
                            new HashSet<DataRow>()));
                }

            }
        }
    }

    private float[] getDepths() {
        if (displayData != null) {
            float[] depths = new float[displayData.getDatum().size()];
            int i = 0;
            for (DataRow d : displayData) {
```

```
            if (d.getValue(DataRow.DEPTH) != null) {
                depths[i] = ((Double) d.getValue(DataRow.DEPTH)).floatValue();
            } else {
                depths[i] = 0.0f;
            }
            i++;
        }
        return depths;
    } else {
        System.err.println("all the things are broken");
        return null;
    }
  }

  private float[] getMagnitudes() {

    float[] mag = new float[displayData.getDatum().size()];
    int i = 0;
    for (DataRow quake : displayData) {
        mag[i] = ((Double) quake.getValue(DataRow.MOMENT_MAGNITUDE))
                .floatValue();
        i++;
    }
    return mag;
  }

}

==> ./src/edu/gatech/earthquakes/vises/Workspace.java <==
package edu.gatech.earthquakes.vises;

import java.awt.Rectangle;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.List;
import java.util.Locale;

import processing.core.PApplet;

import com.google.common.collect.Lists;
import com.google.common.eventbus.Subscribe;

import edu.gatech.earthquakes.components.Controller;
import edu.gatech.earthquakes.components.Theme;
```

```
import edu.gatech.earthquakes.interfaces.Drawable;
import edu.gatech.earthquakes.interfaces.Interactable;
import edu.gatech.earthquakes.model.DataRow;
import edu.gatech.earthquakes.model.DataSet;
import edu.gatech.earthquakes.model.Interaction;
import edu.gatech.earthquakes.util.UIUtils;

public class Workspace extends AbstractVisualization implements Interactable {

    private DataSet masterData;
    List<AbstractVisualization> allVises;
    List<AbstractVisualization> openVises;

    private final float MAX_ASPECT_RATIO = 2.0f;

    private int numHighlighted;

    private static final int BAR_WIDTH = 75, CORNER_RADIUS = 10,
            BUTTON_PADDING = 10;

    public Workspace(int x, int y, int w, int h, DataSet masterData) {
        super(x, y, w - BAR_WIDTH, h, "Primary Workspace", true);
        this.masterData = masterData;
        allVises = Lists.newArrayList();
        openVises = Lists.newArrayList();
        numHighlighted = -1;
        intantiateVises();
    }

    public void intantiateVises() {
        DataRow mainQuake = null;
        for (DataRow quake : masterData.getDatum())
            try {
                if (quake.getValue(DataRow.DATE).equals(
                        new SimpleDateFormat("yyyyMMdd", Locale.ENGLISH)
                                .parse("20010126"))
                        && quake.getValue(DataRow.DEPENDENCY).equals(
                                DataRow.Dependency.INDEPENDENT)) {
                    mainQuake = quake;
                }
            } catch (ParseException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
```

```
        Individual detail = new DetailedInfo(x, y, w, h, mainQuake);
        Controller.registerVisualization(detail);
        allVises.add(detail);

        Aggregate circles = new NestedCirclePlot(x, y, w, h, masterData,
                DataRow.TYPE, masterData.getDatum().size());
        Controller.registerVisualization(circles);
        allVises.add(circles);

        Individual aftershock = new AftershockMap(x, y, w, h, mainQuake,
                masterData);
        Controller.registerVisualization(aftershock);
        allVises.add(aftershock);

        Multi bars = new DepthPlot(x, y, w, h, masterData);
        Controller.registerVisualization(bars);
        allVises.add(bars);

        Multi nomiBars = new NominalBarGraph(x, y, w, h, masterData,
                DataRow.TYPE);
        Controller.registerVisualization(nomiBars);
        allVises.add(nomiBars);

        openVises.addAll(allVises);

    }

    @Override
    public void drawComponent(PApplet parent) {

        // Draw Sidebar Rectangles
        parent.noStroke();
        parent.fill(Theme.rgba(Theme.getBrightUIColor(), 0x33));

        parent.rect(x + w - BAR_WIDTH, y + CORNER_RADIUS, CORNER_RADIUS, h - 2
                * CORNER_RADIUS);
        parent.rect(x + w - (BAR_WIDTH - CORNER_RADIUS), y, BAR_WIDTH
                - CORNER_RADIUS, h);

        // Draw Sidebar Corners
        parent.stroke(Theme.getBaseUIColor());
        parent.strokeWeight(2);
        parent.arc(x + w - (BAR_WIDTH - CORNER_RADIUS), y + CORNER_RADIUS,
```

```
        2 * CORNER_RADIUS, 2 * CORNER_RADIUS, PApplet.PI, PApplet.PI
                + PApplet.HALF_PI);
parent.arc(x + w - (BAR_WIDTH - CORNER_RADIUS), y + h - CORNER_RADIUS,
        2 * CORNER_RADIUS, 2 * CORNER_RADIUS, PApplet.HALF_PI,
        PApplet.PI);

// Draw Sidebar Lines
parent.noFill();
parent.line(x + w, y, x + w - BAR_WIDTH + CORNER_RADIUS, y);

parent.line(x + w - BAR_WIDTH, y + CORNER_RADIUS, x + w - BAR_WIDTH, y
        + h - CORNER_RADIUS);

parent.line(x + w, y + h, x + w - BAR_WIDTH + CORNER_RADIUS, y + h);

// Draw Icons for each vis
parent.stroke(Theme.getBaseUIColor());
int buttonWidth = BAR_WIDTH - BUTTON_PADDING * 2;
int buttonHeight = h / allVises.size() - 2 * BUTTON_PADDING;
int i = 0;
for (AbstractVisualization av : allVises) {
    if (openVises.contains(av)) {
            if (numHighlighted == i) {
                parent.fill(Theme.changeBrightness(
                        Theme.rgba(Theme.getBrightUIColor(), 0xaa), 1.1f,
                        true));
            } else {
                parent.fill(Theme.rgba(Theme.getBrightUIColor(), 0xaa));
            }
    } else {
            if (numHighlighted == i) {
                parent.fill(Theme.changeBrightness(Theme.changeSaturation(
                        Theme.getBrightUIColor(), 0.0f, true), 0.75f, true));
            } else {
                parent.fill(Theme.changeSaturation(
                        Theme.getBrightUIColor(), 0.0f, true));
            }
    }
    int bx = x + w - BAR_WIDTH + BUTTON_PADDING;
    int by = y + BUTTON_PADDING
                + (i * (BUTTON_PADDING * 2 + buttonHeight));

    UIUtils.roundRect(bx, by, buttonWidth, buttonHeight, CORNER_RADIUS,
                parent);
```

```
            parent.rectMode(PApplet.CORNER);
            parent.fill(Theme.getDarkUIColor());
            parent.textAlign(PApplet.CENTER, PApplet.CENTER);

            parent.textSize(16);

            parent.pushMatrix();
            parent.translate(bx + buttonWidth, by + 5);
            parent.rotate(PApplet.PI / 2);
            parent.text(av.getTitle(), 0, 0, buttonHeight - 10, buttonWidth);
            parent.popMatrix();
            i++;
        }
    }

    @Override
    public void resizeTo(Rectangle bounds) {
        super.resizeTo(bounds);
        int index = 0;

        if (openVises.size() > 0) {
            // Determine vis aspect ratio
            int width = (bounds.width - BAR_WIDTH - 10) / openVises.size();
            int height = bounds.height;
            int numRows = 1;
            int maxPerRow = openVises.size();

            while (height / (float) width > MAX_ASPECT_RATIO) {
                numRows++;
                maxPerRow = openVises.size() / numRows;
                if (openVises.size() % numRows != 0) {
                    maxPerRow++;
                }
                width = (bounds.width - BAR_WIDTH - 10) / maxPerRow;
                height = bounds.height / numRows;
            }

            int indexInRow = 0;
            int rowCount = 0;
            int lastRowCount = -1;

            while (index < openVises.size()) {
                int drawX = 0;
                int drawWidth = 0;
```

```
                if (rowCount < numRows - 1) {
                    drawX = bounds.x
                            + (indexInRow * (bounds.width - BAR_WIDTH - 10) / maxPerRow);
                    drawWidth = (bounds.width - BAR_WIDTH - 10) / maxPerRow;
                } else {
                    if (lastRowCount == -1) {
                        lastRowCount = openVises.size() - index;
                    }
                    drawX = bounds.x
                            + (indexInRow * (bounds.width - BAR_WIDTH - 10) / lastRowCount);
                    drawWidth = (bounds.width - BAR_WIDTH - 10) / lastRowCount;
                }

                openVises.get(index).resizeTo(
                        new Rectangle(drawX, y + bounds.height * rowCount
                                / numRows, drawWidth, height));

                indexInRow++;
                if (indexInRow == maxPerRow) {
                    rowCount++;
                    indexInRow = 0;
                }
                index++;
            }
        }
}

@Subscribe
public void handleInput(Interaction interaction) {
        PApplet pa = interaction.getParentApplet();

        // Handle highlighting detection
        boolean isOverOne = false;
        for (int i = 0; i < allVises.size(); i++) {
            if (isOverVisButton(pa.mouseX, pa.mouseY, i)) {
                if (interaction.isFirstPress()) {
                    toggleVis(i);
                }
                numHighlighted = i;
                isOverOne = true;
            }
        }
        if (!isOverOne) {
            numHighlighted = -1;
```

```
        }
    }

    private boolean isOverVisButton(int px, int py, int visCount) {
        int bw = BAR_WIDTH - BUTTON_PADDING * 2;
        int bh = h / allVises.size() - 2 * BUTTON_PADDING;
        int bx = x + w - BAR_WIDTH + BUTTON_PADDING;
        int by = y + BUTTON_PADDING + (visCount * (BUTTON_PADDING * 2 + bh));

        if (px > bx && px < (bx + bw) && py > by && py < (by + bh)) {
            return true;
        } else {
            return false;
        }
    }

    private void toggleVis(int i) {
        AbstractVisualization vis = allVises.get(i);
        if (openVises.contains(vis)) {
            openVises.remove(vis);
            if (vis instanceof Drawable)
                Controller.DRAW_BUS.unregister(vis);
            if (vis instanceof Interactable)
                Controller.INTERACT_BUS.unregister(vis);
        } else {
            openVises.add(Math.min(i, openVises.size()), vis);
            if (vis instanceof Drawable)
                Controller.DRAW_BUS.register(vis);
            if (vis instanceof Interactable)
                Controller.INTERACT_BUS.register(vis);
        }
        resizeTo(new Rectangle(x, y, w, h));
    }

}

==> ./src/edu/gatech/earthquakes/vises/AftershockMap.java <==
package edu.gatech.earthquakes.vises;

import java.awt.Rectangle;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashSet;
```

```
import java.util.Set;

import processing.core.PApplet;
import edu.gatech.earthquakes.components.Controller;
import edu.gatech.earthquakes.components.Theme;
import edu.gatech.earthquakes.interfaces.Filterable;
import edu.gatech.earthquakes.interfaces.Interactable;
import edu.gatech.earthquakes.model.DataRow;
import edu.gatech.earthquakes.model.DataSet;
import edu.gatech.earthquakes.model.Interaction;

public class AftershockMap extends Individual implements Interactable,
        Filterable {

    private double[] latRange;
    private double[] lonRange;
    private double[] magRange;

    private double[] highlightedPos;
    private DataSet aftershocks;
    private static int mainColor = Theme.getPalletteColor(16);
    private static int shockColor = Theme.getPalletteColor(17);

    public AftershockMap(int x, int y, int w, int h, DataRow displayData,
            DataSet filterData) {
        super(x, y, w, h, displayData, "Aftershock Locations");
        filterBy(filterData);
    }

    public void drawComponent(PApplet parent) {
        super.drawComponent(parent);
        parent.textAlign(PApplet.CENTER);

        double[][] coords = getCoordinates();
        double[] m = getMagnitudes();
        double[] mainCoords = getDrawingCoord(
                (double) displayData.getValue(DataRow.LONGITUDE),
                (double) displayData.getValue(DataRow.LATTITUDE));
        // System.out.println(mainCoords[0]);
        for (int i = 0; i < coords.length; i++) {

            // for magnitude, min and max are assumed to be 3 and 7 based on
            // moment magnitude numbers
            double[] c = getDrawingCoord(coords[i][0], coords[i][1]);
```

```
            // System.out.println(c[0]);

            if (highlightedPos != null && coords[i][0] == highlightedPos[0]
                    && coords[i][1] == highlightedPos[1]) {
                parent.fill(Theme.rgba(Theme.HIGHLIGHTED_COLOR, 0x55));
                parent.stroke(Theme.rgba(Theme.HIGHLIGHTED_COLOR, 0xbb));
            } else if (c[0] == mainCoords[0] && c[1] == mainCoords[1]) {

                parent.fill(Theme.rgba(mainColor, 0x88));
                parent.stroke(Theme.rgba(mainColor, 0xdd));
            } else {
                parent.fill(Theme.rgba(shockColor, 0x55));
                parent.stroke(Theme.rgba(shockColor, 0xbb));
            }

            parent.ellipse((float) c[0], (float) c[1],
                    (float) getCircleRadius(m[i]) * 2,
                    (float) getCircleRadius(m[i]) * 2);

        }

        drawAxes(parent);
    }

    private void drawAxes(PApplet parent) {
        parent.stroke(Theme.getDarkUIColor());
        parent.noFill();

        parent.rect(x + buffer, y + buffer, w - buffer * 2, h - buffer * 2);

        parent.fill(Theme.getDarkUIColor());

        parent.textSize(Math.min(w / 30, 12));

        // make and label the latitude tick marks
        double lon = 0;
        for (int i = 0; i < (w - buffer * 2); i += 50) {
            lon = (((lonRange[1] - lonRange[0]) * i) / (w - buffer * 2))
                    + lonRange[0];
            parent.line(x + buffer + i, y + h - buffer - 2, x + buffer + i, y
                    + h - buffer + 2);
            parent.text(formatDegrees(lon), x + buffer + i, y + h - buffer / 4);
        }
```

```
        // make and label the longitude tick marks
        double lat = 0;
        for (int i = 50; i < (h - buffer * 2); i += 50) {
            lat = (((latRange[1] - latRange[0]) * i) / (h - buffer * 2))
                    + latRange[0];
            parent.line(x + buffer - 2, y + h - buffer - i, x + buffer + 2, y
                    + h - buffer - i);

            parent.pushMatrix();
            parent.translate(x + 2 * buffer / 3, y + h - buffer - i);
            parent.rotate(-PApplet.PI / 2);
            parent.text(formatDegrees(lat), 0, 0);
            parent.popMatrix();
        }
    }

    private String formatDegrees(double lat) {
        String formatted = "";
        int degree = (int) lat;
        double minutes = (lat - degree) * 60;
        formatted += degree + "\u00B0 " + (int) minutes + "'";
        return formatted;
    }

    private double[][] getCoordinates() {
        double[][] coords = new double[aftershocks.getDatum().size()][2];

        int i = 0;
        for (DataRow quake : aftershocks) {
            coords[i][0] = (Double) quake.getValue(DataRow.LONGITUDE);
            coords[i++][1] = (Double) quake.getValue(DataRow.LATTITUDE);
        }

        return coords;
    }

    private double[] getMagnitudes() {

        double[] mag = new double[aftershocks.getDatum().size()];
        int i = 0;
        for (DataRow quake : aftershocks) {
            mag[i++] = (Double) quake.getValue(DataRow.MOMENT_MAGNITUDE);
        }
```

```
        return mag;
    }

    private void calculateRanges() {
        double[][] coords = getCoordinates();

        double[] lon = new double[coords.length];
        double[] lat = new double[coords.length];

        for (int i = 0; i < coords.length; i++) {
            lon[i] = coords[i][0];
            lat[i] = coords[i][1];
        }

        Arrays.sort(lon);
        Arrays.sort(lat);

        double dif = 0;
        double buffer = .1;

        if (aftershocks.getDatum().size() > 2) {
            if (lat[lat.length - 1] - lat[0] > lon[lon.length - 1] - lon[0]) {
                dif = lat[lat.length - 1] - lat[0];

                latRange = new double[] { lat[0] - buffer,
                        lat[lat.length - 1] + buffer };
                lonRange = new double[] { lon[0] - buffer,
                        lon[0] + dif + buffer };
            } else {
                dif = lon[lon.length - 1] - lon[0];

                lonRange = new double[] { lon[0] - buffer,
                        lon[lon.length - 1] + buffer };
                latRange = new double[] { lat[0] - buffer,
                        lat[0] + dif + buffer };
            }
        } else {
            double mainLat = (Double) displayData.getValue(DataRow.LATTITUDE);
            double mainLon = (Double) displayData.getValue(DataRow.LONGITUDE);

            latRange = new double[] { mainLat - 1 - buffer,
                    mainLat + 1 + buffer };
            lonRange = new double[] { mainLon - 1 - buffer,
                    mainLon + 1 + buffer };
```

```
        }

        double[] mags = getMagnitudes();
        Arrays.sort(mags);
        magRange = new double[] { mags[0] - .5, mags[mags.length - 1] - .5 };
    }

    @Override
    public void handleInput(Interaction interaction) {

        int mx = interaction.getParentApplet().mouseX;
        int my = interaction.getParentApplet().mouseY;

        if (mx > x && mx < x + w && my > y && my < y + h) {

            double[][] coords = getCoordinates();
            double[] mag = getMagnitudes();
            boolean found = false;

            for (int i = 0; i < coords.length && !found; i++) {
                double[] c = getDrawingCoord(coords[i][0], coords[i][1]);

                if (Math.abs(interaction.getParentApplet().mouseX - c[0]) <
getCircleRadius(mag[i])
                        && Math.abs(interaction.getParentApplet().mouseY - c[1]) <
getCircleRadius(mag[i])) {
                    highlightedPos = new double[] { coords[i][0], coords[i][1] };
                    ArrayList<DataRow> rowList = new ArrayList<>(
                            aftershocks.getDatum());
                    Controller.BRUSH_BUS.post(new DataSet(rowList.get(i)));
                    found = true;
                }

                if (!found) {
                    highlightedPos = null;
                    Controller.BRUSH_BUS.post(new DataSet(
                            new HashSet<DataRow>()));
                }

            }
        }
    }

    /*
```

```
 * All of th scaling is done with the formula of:
 *
 * f(x) = (b-a)(x-min)/(max-min) + a
 *
 * where [min,max] maps to [a,b]
 */

private float getCircleRadius(double mag) {
	float minDiameter = w / 12;
	float maxDiameter = w / 10;
	double maxArea = Math.PI * Math.pow(maxDiameter / 2, 2);
	double minArea = Math.PI * Math.pow(minDiameter / 2, 2);

	float area = (float) ((maxArea - minArea) * (mag - magRange[0])
		/ (magRange[1] - magRange[0]) + minArea);
	return (float) (Math.sqrt(area / Math.PI));
}

private double[] getDrawingCoord(double lon, double lat) {
	double qx = ((w - buffer * 2) * (lon - lonRange[0]))
		/ (lonRange[1] - lonRange[0]);
	double qy = ((h - buffer * 2) * (lat - latRange[0]))
		/ (latRange[1] - latRange[0]);

	return new double[] { x + qx + buffer, y + h - qy - buffer };
}

public void resizeTo(Rectangle bounds) {
	super.resizeTo(bounds);
	if (aftershocks.getDatum().size() > 0) {
	   calculateRanges();
	}
}

@Override
public void filterBy(DataSet filteredData) {
	Date date = (Date) displayData.getValue(DataRow.DATE);
	// System.out.println(date);
	Set<DataRow> shocks = new HashSet<DataRow>();

	for (DataRow quake : filteredData) {
	   if (quake.getValue(DataRow.DEPENDENCY).equals(
			DataRow.Dependency.DEPENDENT)
			&& quake.getValue(DataRow.MAIN_DATE).equals(date))
```

```
                shocks.add(quake);
            if (quake.getValue(DataRow.LATTITUDE).equals(
                    displayData.getValue(DataRow.LATTITUDE))
                    && quake.getValue(DataRow.LONGITUDE).equals(
                            displayData.getValue(DataRow.LONGITUDE))) {
                shocks.add(displayData);
                // System.out.println("added");
            }
        }
        aftershocks = new DataSet(shocks);
        if (aftershocks.getDatum().size() > 0) {
            calculateRanges();
        }
    }
}

==> ./src/edu/gatech/earthquakes/vises/Individual.java <==
package edu.gatech.earthquakes.vises;

import edu.gatech.earthquakes.model.DataRow;

public abstract class Individual extends AbstractVisualization {

    // the current quake that is displayed by this particular individual vis
    protected DataRow displayData;

    public Individual(int x, int y, int w, int h, DataRow displayData) {
        this(x, y, w, h, displayData, "Individual - FIX ME");
    }

    public Individual(int x, int y, int w, int h, DataRow displayData,
            String title) {
        super(x, y, w, h, title);
        this.displayData = displayData;
    }
}

==> ./src/edu/gatech/earthquakes/vises/NominalBarGraph.java <==
package edu.gatech.earthquakes.vises;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Hashtable;
import java.util.Set;
```

```
import processing.core.PApplet;
import edu.gatech.earthquakes.components.Theme;
import edu.gatech.earthquakes.model.DataRow;
import edu.gatech.earthquakes.model.DataSet;

public class NominalBarGraph extends BarGraph
{
	private Hashtable<String, Integer> bars;
	private volatile String longestTitle = "";

	public NominalBarGraph(int x, int y, int w, int h, DataSet displayData, String dataType) {
		super(x, y, w, h, displayData, dataType, "Earthquake " + dataType);
		createBars();
	}

	@Override
	public void drawComponent(PApplet parent) {
		super.drawComponent(parent);

		parent.noStroke();

		//parent.fill(Theme.getDarkUIColor());

		//width of the bars - scales based on the number of bars that we have
		int barW;
		if(numDivisions > 0){
	      barW = (w - buffer*2 - 2*numDivisions)/numDivisions;
		} else {
		   barW = w - buffer*2;
		}
		int barX = x+buffer+4;
		//the scale factor for the height of the bars
		float heightScale = (h-buffer*2.0f)/calcMax();

		parent.textAlign(PApplet.CENTER);
		int textSize = barW/4;
		parent.textSize(textSize);

		while(parent.textWidth(longestTitle) > (h - buffer - 10)){
		   textSize--;
		   parent.textSize(textSize);
		}
```

```
		ArrayList<String> sortedKeys = new ArrayList<>(bars.keySet());
		Collections.sort(sortedKeys);
		for(String key : sortedKeys){
			parent.fill(DataRow.getColorFor(key));
			parent.stroke(Theme.rgba(DataRow.getColorFor(key), 100));
			parent.rect(barX, y+(h-buffer-bars.get(key)*heightScale), barW,
bars.get(key)*heightScale);
			barX += barW+2;
			parent.fill(Theme.getDarkUIColor());
			parent.textAlign(PApplet.LEFT);
			parent.pushMatrix();
			parent.translate(barX-barW/2, y+h-(buffer) - 5);
			parent.rotate(-1 * PApplet.HALF_PI);
			parent.text(key.toString(),0,0);
			parent.popMatrix();
		}

		drawAxes(parent);
	}


	private void drawAxes(PApplet parent){
		parent.stroke(Theme.getDarkUIColor());
		parent.fill(Theme.getDarkUIColor());

		parent.line(x+buffer, y+h-buffer, x+ w-buffer, y+h-buffer); //bottom
		parent.line(x+buffer, y+buffer, x+ buffer, y+h-buffer); //left

		int numTicks = 10;
		int tickVal = calcMax()/numTicks;
		int tickLabel = 0;

		parent.textSize(8);
		parent.textAlign(PApplet.CENTER);
		for(int i=0; i<= h-buffer*2; i+= (h-buffer*2)/numTicks){
		   parent.pushMatrix();
			parent.line(x+buffer-2, y+h-buffer-i, x+buffer+2, y+h-buffer-i);
			parent.translate(x+buffer/2, y+h-buffer-i);
			parent.rotate(-PApplet.PI/2);
			parent.text(tickLabel  + "", 0, 0);
			tickLabel += tickVal;
			parent.popMatrix();
		}
	}
```

```
	/**
	 * Finds the value of the highest bar
	 *
	 * @return
	 */
	private int calcMax(){
		int max = 0;
		for(Object key : bars.keySet()){
			if(max < bars.get(key))
				max = bars.get(key);
		}
		return max;
	}

	private void createBars(){
		bars = new Hashtable<String, Integer>();

		for(DataRow row: displayData){
			if(row.getValue(dataType)!=null){
				if(bars.containsKey(row.getValue(dataType).toString()))
					bars.put(row.getValue(dataType).toString(),
bars.get(row.getValue(dataType).toString()) +1);

				else
					bars.put(row.getValue(dataType).toString(), 1);
			}
		}
		//to stop divide by zero
		numDivisions = Math.max(bars.size(), 1);

		Set<String> keySet = bars.keySet();
		int maxSize = 0;
		for(String key : keySet){
		   if(key.length() > maxSize){
			maxSize = key.length();
			longestTitle = key;
		   }
		}
	}

	@Override
	public void filterBy(DataSet ds){
		super.filterBy(ds);
```

```
                createBars();
        }

}

==> ./src/edu/gatech/earthquakes/vises/Slider.java <==
package edu.gatech.earthquakes.vises;

import java.awt.Rectangle;
import java.util.Calendar;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;

import processing.core.PApplet;

import com.google.common.collect.Sets;

import edu.gatech.earthquakes.components.Controller;
import edu.gatech.earthquakes.components.Theme;
import edu.gatech.earthquakes.interfaces.Interactable;
import edu.gatech.earthquakes.interfaces.Resizable;
import edu.gatech.earthquakes.model.DataRow;
import edu.gatech.earthquakes.model.DataSet;
import edu.gatech.earthquakes.model.Interaction;

public class Slider extends AbstractVisualization implements Interactable{
        float left, right;
        int goalLeft, goalRight;
        int snappedLeft, snappedRight;
        int rangeMin, rangeMax;

        int drawInterval;

        DataSet data;
        int[] years;
        int[] fullYears;

        boolean moveLeft, moveRight, moveAll;

        public static final int OUTSIDE = 0, INSIDE = 1, LEFTHANDLE = 2,
                        RIGHTHANDLE = 3;

        public Slider(int x, int y, int w, int h, DataSet data) {
```

```
		super(x, y, w, h, "Slider", true);
		this.left = x;
		this.right = w + x;
		goalLeft = (int) (left + 0.5f);
		goalRight = (int) (right + 0.5f);
		snappedLeft = goalLeft;
		snappedRight = goalRight;
		drawInterval = 100;
		this.data = data;

		grabDates();

		rangeMin = years[0];
		rangeMax = years[years.length - 1];
		fullYears = new int[rangeMax - rangeMin];
		for (int i = 0; i < fullYears.length; i++) {
			fullYears[i] = i + years[0];
		}

		moveLeft = moveRight = moveAll = false;
	}

	public void grabDates() {
		Set<Date> dates = Sets.newTreeSet();
		for (DataRow dr : data) {
			dates.add((Date) dr.getVariables().get(DataRow.DATE));
		}
		Date[] dateArray = dates.toArray(new Date[] {});
		years = new int[dateArray.length];
		Calendar cal = Calendar.getInstance();
		for (int i = 0; i < years.length; i++) {
			cal.setTime(dateArray[i]);
			years[i] = cal.get(Calendar.YEAR);
		}
	}

	public void changeWidthTo(int newWidth) {
		// Updates the slider width to a new value;
		float ratioR = (right - x) / (float) w, ratioL = (left - x) / (float) w;
		float ratioGR = (goalRight - x) / (float) w, ratioGL = (goalLeft - x)
				/ (float) w;
		w = newWidth;
		right = x + (int) (ratioR * w + 0.5);
		left = x + (int) (ratioL * w + 0.5);
```

```
        goalRight = x + (int) (ratioGR * w + 0.5);
        goalLeft = x + (int) (ratioGL * w + 0.5);
        snapGoals();
}

public void setDrawInterval(int drawInterval) {
        this.drawInterval = drawInterval;
}

public int whereIs(int mX, int mY) {
        int ret = OUTSIDE;
        int handleWidth = 10;
        if (mX >= fuzzLeft(left, this.x)
                        && mX <= fuzzRight(right, this.x + this.w) && mY > this.y
                        && mY < this.y + h) {
                ret = INSIDE;
        } else if (mX > fuzzLeft(left, this.x) - handleWidth
                        && mX < fuzzLeft(left, this.x) && mY > this.y
                        && mY < this.y + h) {
                ret = LEFTHANDLE;
        } else if (mX > fuzzRight(right, this.x + this.w)
                        && mX < fuzzRight(right, this.x + this.w) + handleWidth
                        && mY > this.y && mY < this.y + h) {
                ret = RIGHTHANDLE;
        }
        return ret;
}

public void dragAll(int nx, int px) {
        goalLeft += nx - px;
        goalRight += nx - px;
        if (goalLeft < x) {
                goalRight += x - goalLeft;
                goalLeft += x - goalLeft;
        }
        if (goalRight > x + w) {
                goalLeft -= (goalRight - (x + w));
                goalRight -= (goalRight - (x + w));
        }
}

public void dragLH(int nx, int px) {
        goalLeft += nx - px;
        if (goalLeft < x) {
```

```
                goalLeft += x - goalLeft;
        } else if (goalLeft > goalRight - w / fullYears.length) {
                goalLeft = goalRight - w / fullYears.length;
        }
}

public void dragRH(int nx, int px) {
        goalRight += nx - px;
        if (goalRight > x + w) {
                goalRight -= (goalRight - (x + w));
        } else if (goalLeft > goalRight - w / fullYears.length) {
                goalRight = goalLeft + w / fullYears.length;
        }
}

public void snapGoals() {
        int leftX = goalLeft - x;
        float ratioL = leftX / (float) w;
        int index = (int) Math.min(ratioL * fullYears.length + 0.5,
                        fullYears.length - 1);
        snappedLeft = x + w * index / fullYears.length;
        if (index == 0)
                snappedLeft = x;
        rangeMin = fullYears[index];

        int rightX = goalRight - x;
        float ratioR = rightX / (float) w;
        index = (int) Math.max(ratioR * fullYears.length + 0.5, 0);
        if (index == fullYears.length)
                snappedRight = x + w;
        snappedRight = x + w * index / fullYears.length;
        if (index != 0)
                rangeMax = fullYears[index - 1];
}

public int getLeftBound() {
        int leftX = (int) (left + 0.5) - x;
        float ratioL = leftX / (float) w;
        int index = (int) (ratioL * fullYears.length + 0.5);
        return fullYears[index];
}

public int getRightBound() {
        int rightX = (int) (right + 0.5) - x;
```

```
        float ratioR = rightX / (float) w;
        int index = (int) (ratioR * fullYears.length + 0.5);
        return fullYears[index - 1];
}

public void updateGoals() {
        goalLeft = snappedLeft;
        goalRight = snappedRight;
}

public void updateAnim(int slowness) {
        boolean changed = false;
        if (Math.abs(snappedLeft - left) > 0) {
                left += (snappedLeft - left) / slowness;
                if (Math.abs(snappedLeft - left) == 1) {
                        left = snappedLeft;
                }
                changed = true;
        }
        if (Math.abs(snappedRight - right) > 0) {
                right += (snappedRight - right) / slowness;
                if (Math.abs(snappedRight - right) == 1) {
                        right = snappedRight;
                }
                changed = true;
        }
        if (changed) {
                HashSet<DataRow> filtered = new HashSet<>();
                Calendar cal = Calendar.getInstance();
                for (DataRow dr : data) {
                        Date d = (Date) dr.getVariables().get(DataRow.DATE);
                        cal.setTime(d);
                        if (cal.get(Calendar.YEAR) >= getLeftBound()
                                        && cal.get(Calendar.YEAR) <= getRightBound()) {
                                filtered.add(dr);
                        }
                }
                DataSet ds = new DataSet(filtered);
                Controller.FILTER_BUS.post(ds);
        }
}

@Override
public void drawComponent(PApplet parent) {
```

```
            PApplet p = parent;
                p.stroke(Theme.getBaseUIColor());
                p.strokeWeight(2);
                p.noFill();
                p.strokeJoin(PApplet.ROUND);
                p.beginShape();
                p.vertex(x, y + h);
                p.vertex(x, y);
                p.vertex(x + w, y);
                p.vertex(x + w, y + h);
                p.endShape();

                // Draw underlying data

                // Draw mini graph
                p.stroke(Theme.getPalletteColor(2));
                p.strokeWeight(2);
                p.strokeCap(PApplet.ROUND);
                int prevYear = 0;
                double prevMag = 0;
                for (DataRow r : data) {
                        Date date = (Date) r.getVariables().get(DataRow.DATE);
                        Calendar cal = Calendar.getInstance();
                        cal.setTime(date);
                        int year = cal.get(Calendar.YEAR);
        double mag = (double) r.getVariables().get(
              DataRow.MOMENT_MAGNITUDE);
        if (year > prevYear || mag > prevMag) { // Optimization, dont draw line unless new year
or magnitude is larger
            float xLocation = xLocationMap(year, fullYears[0],
                  fullYears[fullYears.length - 1], x, x + w, left, right);
            float height = PApplet.map((float) mag, 4.0f, 8.0f, 0f,
                  (float) h);
            p.line(xLocation, y + h, xLocation, y + h - height);
        }
                }

                p.fill(Theme.getDarkUIColor());
                p.strokeWeight(2);
                p.stroke(Theme.getDarkUIColor());
                p.line(x, y + h, x + w, y + h);
                for (int i = 0; i < fullYears.length; i++) {
                        int xpos = (int) xLocationMap(i, 0, fullYears.length, x, x + w,
                                        left, right);
```

```
		if (fullYears[i] % drawInterval == 0) {
			p.textAlign(PApplet.CENTER);
			p.textSize(12);
			p.text(fullYears[i], xpos, y + h + 12);
		}

		// Draw ruler ticks
		if (fullYears[i] % 100 == 0) {
			p.line(xpos, y + h, xpos, y + h - 15);
		} else if (fullYears[i] % 10 == 0) {
			p.line(xpos, y + h, xpos, y + h - 10);
		} else {
			// p.line(xpos, y + h, xpos, y + h - 5);
		}
}

p.textSize(24);
p.text("" + getLeftBound() + " - " + getRightBound(), x + w / 2, y + h
			+ 36);

// Draw main bar
p.fill(0, 0, 0, 0);
for (int i = 0; i < h; i++) {
		p.stroke(Theme.rgba(Theme.getBaseUIColor(), i * 127 / h));
		p.line(fuzzLeft(left, x), y + i, fuzzRight(right, x + w), y + i);
}
p.stroke(Theme.getBaseUIColor());
p.rect(fuzzLeft(left, x), y, fuzzRight(right, x + w)
			- fuzzLeft(left, x), h);

// Draw left handle
int handleWidth = 10;
p.stroke(0, 0, 0, 0);
if (whereIs(p.mouseX, p.mouseY) == LEFTHANDLE) {
		p.fill(Theme.rgba(Theme.getBrightUIColor(), 127));
} else {
		p.fill(Theme.rgba(Theme.getBaseUIColor(), 127));
}
p.arc(fuzzLeft(left, x), y + handleWidth, 20, 20, PApplet.PI,
			3 * PApplet.PI / 2);
p.arc(fuzzLeft(left, x), y + h - handleWidth, 20, 20, PApplet.PI / 2,
			PApplet.PI);
p.rect(fuzzLeft(left, x) + 0.5f - handleWidth, y + handleWidth,
			handleWidth, h - 20);
```

```
		p.fill(Theme.getDarkUIColor());
		p.ellipse(fuzzLeft(left, x) - 5, y + (h / 2) - 5, 4, 4);
		p.ellipse(fuzzLeft(left, x) - 5, y + (h / 2), 4, 4);
		p.ellipse(fuzzLeft(left, x) - 5, y + (h / 2) + 5, 4, 4);

		// Draw right handle
		p.stroke(0, 0, 0, 0);
		if (whereIs(p.mouseX, p.mouseY) == RIGHTHANDLE) {
			p.fill(Theme.rgba(Theme.getBrightUIColor(), 127));
		} else {
			p.fill(Theme.rgba(Theme.getBaseUIColor(), 127));
		}
		p.arc(fuzzRight(right, x + w), y + 10, 20, 20, 3 * PApplet.PI / 2,
				2 * PApplet.PI);
		p.arc(fuzzRight(right, x + w), y + h - 10, 20, 20, 0, PApplet.PI / 2);
		p.rect(fuzzRight(right, x + w) + 0.5f, y + 10, 10, h - 20);

		p.fill(Theme.getDarkUIColor());
		p.ellipse(fuzzRight(right, x + w) + 5, y + (h / 2) - 5, 4, 4);
		p.ellipse(fuzzRight(right, x + w) + 5, y + (h / 2), 4, 4);
		p.ellipse(fuzzRight(right, x + w) + 5, y + (h / 2) + 5, 4, 4);

		updateAnim(2);
	}

	private static float xLocationMap(int datm, int dataMin, int dataMax,
			float leftEdge, float rightEdge, float sliderLeft, float sliderRight) {
		float calcuated = 0;
		float linear = PApplet.map(datm, dataMin, dataMax, leftEdge, rightEdge);
		if (linear < sliderLeft) {
			calcuated = fuzzLeft(linear, leftEdge);
		} else if (linear > sliderRight) {
			calcuated = fuzzRight(linear, rightEdge);
		} else {
			float sliderLeftOffset = fuzzLeft(sliderLeft, leftEdge);
			float sliderRightOffset = ((rightEdge - sliderRight) / 2)
					+ sliderRight;
			calcuated = PApplet.map(linear, sliderLeft, sliderRight,
					sliderLeftOffset, sliderRightOffset);
		}
		return calcuated;
	}
```

```
private static float fuzzLeft(float point, float leftEdge) {
	float factor = .5f;
	return ((point - leftEdge) * factor) + leftEdge;
}

private static float fuzzRight(float point, float rightEdge) {
	float factor = .5f;
	return rightEdge - ((rightEdge - point) * factor);
}

@Override
public void handleInput(Interaction interaction) {
   //FIXME allow user to select a single earthquakes
	if (interaction.isFirstPress()) {
		int location = whereIs(interaction.getParentApplet().mouseX,
				interaction.getParentApplet().mouseY);
		switch (location) {
		case LEFTHANDLE:
			moveLeft = true;
			break;
		case RIGHTHANDLE:
			moveRight = true;
			break;
		case INSIDE:
			moveAll = true;
			break;
		}
	} else if (interaction.isDragged()) {
		if (moveLeft) {
			dragLH(interaction.getParentApplet().mouseX,
					interaction.getParentApplet().pmouseX);
			snapGoals();
		}
		if (moveRight) {
			dragRH(interaction.getParentApplet().mouseX,
					interaction.getParentApplet().pmouseX);
			snapGoals();
		}
		if (moveAll) {
			dragAll(interaction.getParentApplet().mouseX,
					interaction.getParentApplet().pmouseX);
			snapGoals();
		}
	} else if (interaction.isReleased()) {
```

```
                updateGoals();
                moveLeft = moveRight = moveAll = false;
            }
        }

        @Override
        public void resizeTo(Rectangle bounds) {
            // This approach makes the date range scale closely enough, though due
            // to integer division the dates can change. Ideally this will be
            // changed in the future, to have the location determined by the
            // selected dates, not the other way around.
            double goalLeftRatio = (goalLeft - x) / (double) w;
            double goalRightRatio = (goalRight - x) / (double) w;
            double leftRatio = (left - x) / (double) w;
            double rightRatio = (right - x) / (double) w;
            double snapLeftRatio = (snappedLeft - x) / (double) w;
            double snapRightRatio = (snappedRight - x) / (double) w;
            super.resizeTo(bounds);
            goalLeft = (int) (x + goalLeftRatio * w);
            goalRight = (int) (x + goalRightRatio * w);
            left = (float) (x + leftRatio * w);
            right = (float) (x + rightRatio * w);
            snapLeftRatio = (int) (x + snapLeftRatio * w);
            snapRightRatio = (int) (x + snapRightRatio * w);
            snapGoals();
        }
}

==> ./src/edu/gatech/earthquakes/vises/BarGraph.java <==
package edu.gatech.earthquakes.vises;

import edu.gatech.earthquakes.interfaces.Filterable;
import edu.gatech.earthquakes.model.DataSet;

public abstract class BarGraph extends Aggregate implements Filterable{
        protected String dataType;
        protected int buffer = 20;
        protected int numDivisions;


        public BarGraph(int x, int y, int w, int h, DataSet displayData, String dataType) {
            this(x, y, w, h, displayData, dataType, "Bargraph - FIX ME");
        }
```

```
	public BarGraph(int x, int y, int w, int h, DataSet displayData, String dataType, String title)
{
		super(x, y, w, h, displayData, title);

		this.dataType = dataType;
	}

	@Override
	public void filterBy(DataSet filteredData) {
		displayData = filteredData;
	}

}

==> ./src/edu/gatech/earthquakes/vises/Multi.java <==
package edu.gatech.earthquakes.vises;

import edu.gatech.earthquakes.components.Controller;
import edu.gatech.earthquakes.model.DataSet;

/**
 * A visualization that displays more than one earthquake
 *
 * @author Elizabeth
 *
 */
public abstract class Multi extends AbstractVisualization {

   protected DataSet displayData;

   public Multi(int x, int y, int w, int h, DataSet displayData) {
	super(x, y, w, h, "Multi - FIX ME");

	this.displayData = displayData;

   }

   public Multi(int x, int y, int w, int h,
		DataSet displayData, String title) {
	super(x, y, w, h, title);
	this.displayData = displayData;
   }

   protected final void applyFilterGlobally() {
```

```
            Controller.applyFilter(displayData);
        }
    }

==> ./src/edu/gatech/earthquakes/model/DataComparator.java <==
package edu.gatech.earthquakes.model;

import java.util.Comparator;
import java.util.Date;
import java.util.HashMap;

public class DataComparator implements Comparator<DataRow> {

    private static final DataComparator baseCompare = new DataComparator(
            DataComparator.CompareCategories.DATE,
            DataComparator.CompareCategories.MAGNITUDE,
            DataComparator.CompareCategories.DEPTH);

    public static final HashMap<String, CompareCategories> categoryMap;
    static {
        categoryMap = new HashMap<>();
        categoryMap.put(DataRow.CONTINENT, CompareCategories.CONTINENT);
        categoryMap.put(DataRow.DEPENDENCY, CompareCategories.DEPENDENCY);
        categoryMap.put(DataRow.TYPE, CompareCategories.TYPE);
    }

    public enum CompareCategories {
        DATE(false), DATE_REVERSE(true), MAGNITUDE(false), MAGNITUDE_REVERSE(
                true), CONTINENT(false), DEPENDENCY(false), TYPE(false), DEPTH(
                false), DEPTH_REVERSED(true);

        private boolean reversed;

        private CompareCategories(boolean reversed) {
            this.reversed = reversed;
        }

        private boolean isReversed() {
            return reversed;
        }
    }

    public static DataComparator getDefaultComparator() {
        return baseCompare;
```

```
}

private CompareCategories[] categories;

public DataComparator(CompareCategories... compareCategories) {
	this.categories = compareCategories;
}

@Override
public int compare(DataRow arg0, DataRow arg1) {
	return compareLeveled(arg0, arg1, 0);
}

public int compareLeveled(DataRow arg0, DataRow arg1, int index) {
	if (index >= categories.length)
	   return 0;
	switch (categories[index]) {
	// The enum contains a boolean as for when it's reversed. Otherwise,
	// logic is the same.
	   case DATE:
	   case DATE_REVERSE:
	         Date d0 = (Date) arg0.getVariables().get(DataRow.DATE);
	         Date d1 = (Date) arg1.getVariables().get(DataRow.DATE);
	         if (d0.compareTo(d1) != 0)
	            return (!categories[index].isReversed() ? d0.compareTo(d1)
	                  : d1.compareTo(d0));
	         break;
	   case MAGNITUDE:
	   case MAGNITUDE_REVERSE:
	         Double m0 = (double) arg0.getVariables().get(
	               DataRow.MOMENT_MAGNITUDE);
	         Double m1 = (double) arg1.getVariables().get(
	               DataRow.MOMENT_MAGNITUDE);
	         if (m0.compareTo(m1) != 0)
	            return (!categories[index].isReversed() ? m0.compareTo(m1)
	                  : m1.compareTo(m0));
	         break;
	   case CONTINENT:
	         DataRow.Continent c0 = (DataRow.Continent) arg0.getVariables()
	               .get(DataRow.CONTINENT);
	         DataRow.Continent c1 = (DataRow.Continent) arg1.getVariables()
	               .get(DataRow.CONTINENT);

	         if (c0.compareTo(c1) != 0)
```

```
                return c0.compareTo(c1);
            break;
        case DEPENDENCY:
            DataRow.Dependency dependency0 = (DataRow.Dependency) arg0
                .getVariables().get(DataRow.DEPENDENCY);
            DataRow.Dependency dependency1 = (DataRow.Dependency) arg1
                .getVariables().get(DataRow.DEPENDENCY);
            if (dependency0.compareTo(dependency1) != 0)
                return dependency0.compareTo(dependency1);
            break;
        case DEPTH:
        case DEPTH_REVERSED:
            Object dObj0 = arg0.getVariables().get(DataRow.DEPTH);
            Object dObj1 = arg1.getVariables().get(DataRow.DEPTH);
            if (dObj0 == null) {
                if (dObj1 == null) {
                    break;
                } else {
                    return -1;
                }
            } else if (dObj1 == null) {
                return 1;
            }
            Double depth0 = (double) dObj0;
            Double depth1 = (double) dObj1;
            if (depth0.compareTo(depth1) != 0)
                return (!categories[index].isReversed() ? depth0
                        .compareTo(depth1) : depth1.compareTo(depth0));
            break;
        case TYPE:
            DataRow.Type t0 = (DataRow.Type) arg0.getVariables().get(
                DataRow.TYPE);
            DataRow.Type t1 = (DataRow.Type) arg1.getVariables().get(
                DataRow.TYPE);
            if (t0.compareTo(t1) != 0)
                return t0.compareTo(t1);
            break;
        }
        return compareLeveled(arg0, arg1, index + 1);
    }
}

==> ./src/edu/gatech/earthquakes/model/Interaction.java <==
package edu.gatech.earthquakes.model;
```

```
import processing.core.PApplet;

public class Interaction {

	final private boolean firstPress, dragged, released;
	final private PApplet parentApplet;

	public Interaction(final boolean firstPress, final boolean dragged,
		final boolean released, final PApplet parentApplet) {
		    super();
		    this.firstPress = firstPress;
		    this.dragged = dragged;
		    this.released = released;
		    this.parentApplet = parentApplet;
	}

	public boolean isFirstPress() {
		    return firstPress;
	}

	public boolean isDragged() {
		    return dragged;
	}

	public boolean isReleased() {
		    return released;
	}

	public PApplet getParentApplet() {
		    return parentApplet;
	}



}

==> ./src/edu/gatech/earthquakes/model/DataSet.java <==
package edu.gatech.earthquakes.model;

import java.util.Iterator;
import java.util.Set;
import java.util.TreeSet;
```

```
public class DataSet implements Iterable<DataRow> {

    private TreeSet<DataRow> datum;

    public DataSet(Set<DataRow> datum) {
        if (datum instanceof TreeSet<?>) {
            this.datum = (TreeSet<DataRow>) datum;
        } else {
            this.datum = new TreeSet<>();
            this.datum.addAll(datum);
        }
    }

    public DataSet(DataRow row) {
        TreeSet<DataRow> singleRow = new TreeSet<>();
        singleRow.add(row);
        this.datum = singleRow;
    }

    public TreeSet<DataRow> getDatum() {
        return datum;
    }

    public void setDatum(TreeSet<DataRow> datum) {
        this.datum = datum;
    }

    @Override
    public Iterator<DataRow> iterator() {
        return datum.iterator();
    }

}

==> ./src/edu/gatech/earthquakes/model/DataRow.java <==
package edu.gatech.earthquakes.model;

import java.util.Map;

import edu.gatech.earthquakes.components.Theme;

public class DataRow implements Comparable<DataRow>{

        public static int getColorFor(final String enumText){
```

```
                int result = -1;
                for(Continent c : DataRow.Continent.values()){
                        if(c.toString().equals(enumText)){
                                result = c.getColor();
                        }
                }
                for(Type t: Type.values()){
                        if(t.toString().equals(enumText)){
                                result = t.getColor();
                        }
                }
                for(Dependency d: Dependency.values())
                   if(d.toString().equals(enumText)){
                      result = d.getColor();
                   }

                if(enumText.equals(DEPTH)){
                        result = Theme.getPalletteColor(15);
                }
                if(result == -1){
                        throw new IllegalArgumentException("Name has no corresponding
enum");
                } else {
                        return result;
                }
        }

   public final static String DATE = "Date";
   public final static String RECORD = "Record";
   public final static String LATTITUDE = "Lattitude";
   public final static String LONGITUDE = "Longitude";
   public final static String TIME = "Time";
   public final static String CONTINENT = "Continent";
   public enum Continent {
      AFRICA("Africa", Theme.getPalletteColor(0)),
      AUSTRALIA("Australia", Theme.getPalletteColor(1)),
      ASIA("Asia", Theme.getPalletteColor(2)),
      EURASIA("Eurasia", Theme.getPalletteColor(3)),
      INDIA("India", Theme.getPalletteColor(4)),
      NORTH_AMERICA("North America", Theme.getPalletteColor(5)),
      SOUTH_AMERICA("South America", Theme.getPalletteColor(6));

      private String text;
      private int color;
```

```
        private Continent(final String text, final int color){
            this.text = text;
            this.color = color;
        }

        public String toString(){
            return text;
        }

        public int getColor(){
            return color;
        }
    }
    public final static String DEPTH = "Depth";
    public final static String MOMENT_MAGNITUDE = "Magnitude";
public final static String MOMENT_MAGNITUDE_UNCERTAINTY = "Magnitude_Uncertainty";
    public final static String BODY_WAVE_MAGNITUDE = "Body Wave Magnitude";
    public final static String SURFACE_WAVE_MAGNITUDE = "Surface Wave Magnitude";
    public final static String LOCAL_WAVE_MAGNITUDE = "Local Wave Magnitude";
    public final static String DEPENDENCY = "Dependency";
    public final static String MAIN_DATE = "Main Date";
    public enum Dependency{
        INDEPENDENT("Independent", Theme.getPalletteColor(7)),
        DEPENDENT("Dependent", Theme.getPalletteColor(8)),
        POSSIBLY("Possibly", Theme.getPalletteColor(9));

        private String text;
        private int color;

        private Dependency(final String text, final int color){
            this.text = text;
            this.color = color;
        }

        public String toString(){
            return text;
        }

        public int getColor(){
            return color;
        }
    }
    public final static String TYPE = "Type";
```

```
public enum Type{
   TECT("Tectonic", Theme.getPalletteColor(10)),
   DEEP_MINING("Deep Mining", Theme.getPalletteColor(11)),
   MINING("Mining", Theme.getPalletteColor(12)),
   RESERVOIR("Reservoir", Theme.getPalletteColor(13)),
   OIL_FEILD("Oil Field", Theme.getPalletteColor(14));

   private String text;
   private int color;

   private Type(String text, int color){
      this.text = text;
      this.color = color;
   }

   public String toString(){
      return text;
   }

   public int getColor(){
      return color;
   }
}

private Map<String, Object> variables;

      public DataRow(Map<String, Object> variables){
            this.variables = variables;
      }

      public Map<String, Object> getVariables() {
            return variables;
      }

      public void setVariables(Map<String, Object> variables) {
            this.variables = variables;
      }

      public Object getValue(String dataType){
            return variables.get(dataType);
      }

      @Override
   public int compareTo(DataRow arg0) {
```

```
            return DataComparator.getDefaultComparator().compare(this, arg0);
        }

}

==> ./src/edu/gatech/earthquakes/model/DeadEventCanary.java <==
package edu.gatech.earthquakes.model;

import com.google.common.eventbus.DeadEvent;
import com.google.common.eventbus.Subscribe;

import edu.gatech.earthquakes.components.Controller;

public class DeadEventCanary {

        private static DeadEventCanary instance;

        static{
                instance = new DeadEventCanary();
        }

        public static DeadEventCanary getInstance(){
                return instance;
        }

        @Subscribe
        public void respondToDeadEvent(final DeadEvent de){
                String busName = "";
                if(de.getSource().equals(Controller.BRUSH_BUS)){
                        busName = "brushing";
                }
                else if(de.getSource().equals(Controller.DRAW_BUS)){
                        busName = "drawing";
                }
                else if(de.getSource().equals(Controller.FILTER_BUS)){
                        busName = "filering";
                }
                else if(de.getSource().equals(Controller.INTERACT_BUS)){
                        busName = "interaction";
                }

                System.err.println("Dead Event " + de.getEvent() + " Dispatched on " + busName
+ " bus.");
        }
```

```
}

==> ./src/edu/gatech/earthquakes/web/CustomSearch.java <==
package edu.gatech.earthquakes.web;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLEncoder;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Properties;
import java.util.Scanner;

import org.jsoup.Jsoup;

import com.google.gson.JsonElement;
import com.google.gson.JsonParser;

public class CustomSearch {
   private final static String SEARCH_BASE = "https://www.googleapis.com/customsearch/v1";
   private final static String DATA_LOCATION = ".." + File.separator + "data" + File.separator;
   private final static String PROPERTIES_FILENAME = "config.properties";
   private final static String CACHE_LOCATION = DATA_LOCATION + "cache" +
File.separator;
   //custom search api key
   private String key;
   //custom search engine key
   private String cx;

   private static CustomSearch instance;
   static{
      instance = new CustomSearch();
   }
   public static CustomSearch getInstance(){
      return instance;
   }
```

```
    //usage is CustomSearch cs = new CustomSearch(); cs.getQuery("some_query");
    private CustomSearch(){
        this(DATA_LOCATION + PROPERTIES_FILENAME);
    }

    public CustomSearch(final String filepath_name) {
        final Properties prop = new Properties();
        try {
            //load a properties file
            prop.load(new FileInputStream(filepath_name));
            key = prop.getProperty("key");
            cx = prop.getProperty("cx");
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    private URL getUrl(final String query) throws MalformedURLException{
        String escaped_q = query;
        try {
            escaped_q = URLEncoder.encode(query, "UTF-8");
        } catch (UnsupportedEncodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        final String attempted_url = SEARCH_BASE + "?" + "key=" + key + "&cx=" + cx + "&q=" +
escaped_q;
        return new URL(attempted_url);
    }

    private static final String getOnlineContent(final URL url) throws IOException{
        //returns string of html page
        final InputStream in = url.openStream();
        final StringBuffer sb = new StringBuffer();

        final byte [] buffer = new byte[256];

        while(true){
            final int byteRead = in.read(buffer);
            if(byteRead == -1){
                break;
            }
            for(int i = 0; i < byteRead; i++){
                sb.append((char)buffer[i]);
```

```
            }
        }
        return sb.toString();
    }

    //TODO catch appropriate errors
    public String getQuery(String query) throws NoSuchAlgorithmException,
MalformedURLException, IOException{
        //this checks to see if the file is in the cache then returns the results as a string
        String result;
        final byte[] bytesOfMessage = query.getBytes("UTF-8");
        final MessageDigest md = MessageDigest.getInstance("MD5");
        byte[] thedigest = md.digest(bytesOfMessage);
        final String filename = bytesToPrintableString(thedigest);
        File f = new File(CACHE_LOCATION + filename);

        //TODO handle if query happens to be a directory
        if(f.exists() && !f.isDirectory()){
            //file is in cache
            Scanner s = new Scanner(f);
            //TODO handle no such element exception
            result = s.useDelimiter("\\Z").next();
            s.close();
        }
        else{
            //get content and write it to a file the return it
            BufferedWriter out = new BufferedWriter(new FileWriter(f));
            result = getOnlineContent(getUrl(query));
            out.write(result);
            out.close();
        }
        return result;
    }

    private String bytesToPrintableString(final byte[] thedigest) {
        //This is used to allow files to be saved in windows
        //Possible Chars are windows safe chars
        char[] possibleChars = {'0','1','2','3','4','5','6','7','8','9',
                'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z',
                'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z',
                };
        char[] writeableChars = new char[thedigest.length];
        for(int i=0; i< thedigest.length; i++){
            writeableChars[i] =  possibleChars[Math.abs(thedigest[i]) % possibleChars.length];
```

```
        }
        return new String(writeableChars);
    }

    public static int getTotalCount(final String jsonLine){
        //jsonline['queries']['request'][0]['totalResults']
        JsonElement jelement = new JsonParser().parse(jsonLine);
        String result = jelement.
                getAsJsonObject().
                getAsJsonObject("queries").
                getAsJsonArray("request").
                get(0).
                getAsJsonObject().
                get("totalResults").
                toString();
        //strips leading and trailing quotes
        result = result.replace("", ' ').trim();
        return Integer.parseInt(result);
    }

    public static String getTitles(int index, String jsonLine){
        //jdata['items'][1]['htmlTitle']
        JsonElement jelement = new JsonParser().parse(jsonLine);
        String result = jelement.
                getAsJsonObject().
                getAsJsonArray("items").
                get(index).
                getAsJsonObject().
                get("htmlTitle").
                toString();
        return Jsoup.parse(result).text();
    }
}
```