Exercise Power Grid

Display and Web Interface

Advisor: Dr. David Braun

Term: Spring-Summer-Fall 2009

Student: Alex Chernetz

Major: Computer Engineering (CPE)
Abstract

The 2008-2009 expansion of the Recreation Center at Cal Poly includes three new rooms with cardiovascular fitness equipment. As part of its ongoing commitment to sustainable development, the new machines connect to the main power grid and generate power during a workout. This document explains the process of quantifying and expressing the power generated using two interfaces: an autonomous display designed for a television with a text size and amount of detail adaptable to multiple television sizes and viewing distances, and an interactive, more detailed Web interface accessible with any Java-capable computer system or browser.

Introduction

According to a 2002 report from the Centers for Disease Control and Prevention (CDC) [1], the average American man has a body mass index (BMI) of 27.8, and the average American woman has a BMI of 28.1. For reference, the World Health Organization (WHO) defines “overweight” as a BMI exceeding 25.0 and “obese” as a BMI exceeding 30.0 [2]. In other words, the average American (both men and women) needs to lose 19 pounds.

It comes as no surprise that out of the top ten reasons men and women give for exercising, “weight control” ranks first, chosen by 84.7% of men and women on average [3]. “Increased energy” occupies a close second place at 84.5%, but this reason carries a double entendre, and many people are not aware of it.

If all 307.5 million people in this country lost an average of 19 pounds of fat, it would translate into more than 6 billion kilowatt-hours of usable energy even if we factor in the paltry average muscle efficiency of 22% [4][5]. Recycling this energy would keep more than 4.8 million tons of CO$_2$ out of the atmosphere. In economic terms, it would open a billion-dollar revenue stream from electricity generation alone, not to mention the $117 billion this would save annually in healthcare costs [6]! Yes, these figures amount to a pipe dream, but the energy saved from modifying even 50 machines is far from trivial (see the line graph for more details).

Unfortunately, most exercise machines dissipate the energy they generate through a resistor bank. This not only wastes energy, but also results in the waste of even more energy when the machine heats the fitness room up and the gym thermostat uses air conditioning to compensate for it. However, by modifying exercise machines with a power inverter and connecting them to the power grid, these machines can generate electricity instead and minimize air conditioning usage.

This project seeks to quantify the renewable energy that this system will provide, and to raise the awareness of exercisers (as well as their calorie burn count) by presenting these statistics in a manner that captures eyes and heightens motivation.
**Project Overview**

This project consists of a software application developed with Java 1.6 that calculates the power generated in the new Recreation Center by Precor EFX-546i fitness machines (see Figure 1) modified with Enphase and standalone power inverters. While this project currently uses data provided by simulations and examples, it provides a foundation for connecting to the newly implemented EHFEM MySQL database running on Decade.

The application illustrates the power data in three different ways: an animated bar graph that shows current power output across groups of machines, an animated line graph that shows power output over the course of yesterday, and a table that converts power output over different times into offsets of polluting chemicals and energy sources.

Eventually, as this software continues to evolve, it will support at least two and probably three different methods of deployment. In addition to its current form (i.e., a desktop Java application), this project will function as a scripted, non-interactive program intended for use with a large television in the Recreation Center, and it will function as a Java applet accessible through the Decade server.

![Figure 1. A Precor EFX-546i exercise machine.](http://www.oneshotfitness.com/wp-content/uploads/2009/01/img_4097.jpg)

**Requirements and Specifications**

Table I contains the current working list of requirements and specifications for this project. Each requirement and/or specification in Table I has one or more ABET categories assigned to it, as well as a short justification that explains the rationale behind its inclusion in the list.
Summary of ABET Categories

The ABET categories used to classify requirements are all the principal elements of sustainability, which are as follows:

- **Economic**: Any criterion intended to save money or a significant amount of time.
  - **Manufacturability**: Criteria that ensure the usability of the final system.
- **Environmental**: Any criterion that has some type of ecological or environmental benefit.
- **Sociopolitical**: Encompasses socially and politically motivated criteria.
  - **Health and Safety**: Any criterion intended to prevent physical or mental harm.
  - **Ethical**: Criteria ensuring open standards compatibility or lawful mandates.

Each project requirement in Table I has either an orange or green text color to indicate its implementation status. Green requirements are complete; orange requirements are not. In addition, all requirements colored orange have recent updates explaining the cause of delay.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>ABET Category</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project will run as an interactive web page.</td>
<td>Sociopolitical</td>
<td>An interactive web page will make the project more informative, which will increase its popularity and exposure to the public.</td>
</tr>
<tr>
<td>Project will run on a graphical display controlled by a script.</td>
<td>Sociopolitical</td>
<td>In order to display the power statistics in the gym, the project must run independently without prompting for input. <strong>UPDATE 11/16/2009</strong>: While the foundation for this functionality exists with the animation classes, this will likely be postponed until the database is finalized.</td>
</tr>
<tr>
<td>The web page and graphical display will utilize the same graphing libraries.</td>
<td>Manufacturability, Economic</td>
<td>The web page form of the project allows the user to select units on the graphs and cycle through the graphs with buttons. The graphical display cycles through the graphs based on a timer. Nothing else is, or should be, different.</td>
</tr>
<tr>
<td>The web page will support all common browsers.</td>
<td>Sociopolitical, Ethical</td>
<td>The web page should not have any information in it that requires specific software (especially proprietary, closed-source software) to render.</td>
</tr>
<tr>
<td>The inference of power output data from any single individual will be impossible.</td>
<td>Health and Safety, Ethical, Economic</td>
<td>Allowing the exercise data from specific individuals to become public constitutes an invasion of privacy and may result in litigation.</td>
</tr>
<tr>
<td>The graphical display will support any operating system.</td>
<td>Sociopolitical, Ethical</td>
<td>The graphical display should operate with any computer, as long as the computer hardware can generate and output the graphics. Since the graphics will be basic, any system younger than 10 years should have this capability.</td>
</tr>
<tr>
<td>The graphical display will meet the FAA standards for text size.</td>
<td>Health and Safety</td>
<td>The Federal Aviation Administration (FAA) published a standard that specifies the recommended and minimum angular height for text in order to ensure its legibility. <strong>UPDATE 11/16/2009:</strong> The N in the 20:N vision scale (e.g., 20:20 vision) is 4 times the angular height; the FAA minimum standard (16’) is thus visible with 20:64 vision, so this requirement may not be practical given the state budget crisis and cost of large displays.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>The project will raise environmental awareness by being informative and interesting.</td>
<td>Environmental</td>
<td>Making the project interesting and informative will encourage viewers to make decisions more likely to benefit the environment.</td>
</tr>
<tr>
<td>The television used to show the power statistics in the gym will be as large as possible without being overpriced or inefficient.</td>
<td>Economic, Environmental, Sociopolitical</td>
<td>The TV market has a wide array of technologies and vendors. Some technologies have size limitations that will make a TV prohibitively expensive if exceeded, and some TVs can consume up to quadruple the power of other TVs. <strong>UPDATE 11/16/2009:</strong> The final selection and installation will take place in 2010. The code will support scripting with little modification. Deploying the Web interface first will also help determine the best information to show in the script.</td>
</tr>
<tr>
<td>The project and database will be compatible with Enphase power inverters.</td>
<td>Manufacturability</td>
<td>The exercise machines in the gym will generate power through Enphase power inverters, so any device made to display that information must be able to access it.</td>
</tr>
<tr>
<td>All of the code used in the project will be released and made publicly available under the GPL.</td>
<td>Ethical</td>
<td>In order to verify the accuracy of the statistics calculated by the project, the code used in the process should be open-source.</td>
</tr>
<tr>
<td>All of the code used in the project will be accompanied by comprehensive documentation.</td>
<td>Economic, Manufacturability</td>
<td>Documenting all the source code will allow the source code to be extended to support new features, comprehensible to anyone who needs to learn it, and extensible to applications outside the scope of the project.</td>
</tr>
<tr>
<td>The code will provide an easy interface for developers to create and control multiple execution threads.</td>
<td>Manufacturability, Sociopolitical, Ethical</td>
<td>The Java Swing libraries require multithreading in order to function properly, especially when animating data, retrieving data using a query, or interacting with the user.</td>
</tr>
</tbody>
</table>
**Project Planning Timeline**

Table II provides a Gantt chart that shows the timeline for each phase of the project.

<table>
<thead>
<tr>
<th>3/30 – 12/10</th>
<th>MAR</th>
<th>APR</th>
<th>MAY</th>
<th>JUN</th>
<th>JUL</th>
<th>AUG</th>
<th>SEP</th>
<th>OCT</th>
<th>NOV</th>
<th>DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Planning phase (4/1 - 9/20)</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposal (4/1 - 4/10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements (6/20 - 9/10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase 0 Design: Scope Definition (4/10 - 6/20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase 1 Design: Screen Layout (5/1 - 8/1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase 2 Design: Code Structuring (7/20 - 10/1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation (8/10 - 12/10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation (8/10 - 12/10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report Compilation (9/1 - 12/10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing (8/20 - 12/10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Design Process and Challenges**

The first phase of the design process consisted mainly of informal brainstorming and determining the scope of the project. This phase took place from the beginning of the spring term and ended around the beginning of July after determining that the programming language most suitable for the project was Java.

The second phase consisted of developing the foundations and prerequisite objects that turned out to be necessary in the implementation process, such as thread management systems for animation and specific types of datasets for graphing.
Consideration and Rejection of CSAFE

One of the first ideas involved the use of a protocol known as the **Communications Specification for Fitness Equipment, or CSAFE**. While CSAFE is an open standard, it is fairly obscure and would have required the assembly of additional hardware devices to process data from the exercise machines. In addition, different manufacturers implement different parts of CSAFE. Three models of exercise machines may have required three different methods of accessing the required data, and information on commercial deployment and compatibility turned out to be very scarce. This idea was scrapped before the end of spring quarter for two reasons; the first reason was the excessive time that would likely have been required to create the CSAFE hardware device, and the second reason was the risk involved. Having the success or failure of a project dependent essentially on luck does not make for a sustainable project.

Choosing Java as a Programming Language

Initially, I never considered Java as a viable choice; I thought a Java implementation would run too slowly to be useful in the least regard. I had a strong desire to use either C or C++, but the more I thought about potential issues such as cross-compatibility and browser integration that could arise with C or C++, Java looked more and more appealing. It had been about two years since I had last developed anything substantial with Java; in fact, the last program I had developed with Java had been a primitive version of a limitation hacking tool for a video game that was more than likely compiled with C++ (more information on this can be found [here](http://www.irrlicht3d.org/pivot/entry.php?id=446) and [here](http://www.irrlicht3d.org/pivot/entry.php?id=446)). Obviously, for that project, C++ would speed up the process of finding binary data within a file and manipulating pointers, since Java lacks pointers entirely.

For any other project that does not involve hacking video games, the difference in speed is negligible (and completely meaningless considering the savings in development time). The turning point in this decision process came when I decided to revisit benchmarks comparing Java to C++ and found results where the performance of Java tended to fall within a factor of 2 of C++ performance, such as the results in Table III.

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fibonacci</td>
<td>10767 ms</td>
<td>7468 ms</td>
</tr>
<tr>
<td>Heapsort</td>
<td>5454 ms</td>
<td>4344 ms</td>
</tr>
<tr>
<td>NestedLoops</td>
<td>8022 ms</td>
<td>3453 ms</td>
</tr>
<tr>
<td>CopyData</td>
<td>1204 ms</td>
<td>1125 ms</td>
</tr>
</tbody>
</table>

Source: [http://www.irrlicht3d.org/pivot/entry.php?id=446](http://www.irrlicht3d.org/pivot/entry.php?id=446)
Shortly after downloading and configuring the NetBeans IDE, I wrote my first class for the project: a number formatter that takes a double and formats it to maintain a constant number of significant figures in the output, along with an appropriate suffix based on SI notation (i.e., ‘m’ for $10^{-3}$, ‘K’ for $10^{3}$, ‘M’ for $10^{6}$, etc.).

After this point, the design and implementation of the project began to accelerate dramatically.

**Inclusion of Third-Party Libraries**

My first attempt at creating the graphs involved attempting to create the graphs from scratch, placing the graph labels and titles based on equations that took into account the size of the display (see Appendix B). However, this proved inadequate when I attempted to render the Y-axis title, which requires a $90^\circ$ counterclockwise rotation.

Unfortunately, Swing does not support text rotation without overriding the `paintComponent` method. Manipulating Swing components with Java 2D painting code proved difficult and prone to error. This led me to start looking for open source solutions rather than continuing to implement my design. Specifically, I wanted a tested and proven solution with an open source license, and found exactly that with the JFreeChart library. The next section discusses the implications of this discovery.

After discovering JFreeChart, I researched other libraries with the goal of reducing development time as much as possible, which led to the inclusion of SwingX and JGoodies. Table IV contains information regarding the third-party libraries utilized in the project, all of which carry open source licenses compatible with version 3 of the GPL.

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Version/Date</th>
<th>Java Import</th>
<th>License</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFreeChart</td>
<td>1.0.13 4/20/2009</td>
<td>org.jfree.chart</td>
<td>LGPL v2.1</td>
<td>Provides a wide array of dataset and graph types to simplify the graph drawing process.</td>
</tr>
<tr>
<td>SwingX</td>
<td>10/28/2009</td>
<td>javax.swing</td>
<td>LGPL v2.1</td>
<td>Provides several useful extensions to the Swing GUI and toolkit.</td>
</tr>
<tr>
<td>JGoodies</td>
<td>9/16/2009</td>
<td>com.jgoodies</td>
<td>BSD</td>
<td>Makes the project look “nicer” by providing various user interface extensions.</td>
</tr>
</tbody>
</table>
Starting Out with JFreeChart

JFreeChart eliminated the need to write hundreds of lines of code to draw graphs, which opened the door to developing additional features that would make the graphs far more interesting. With all the standard datasets, options, and graph types included with the library, creating a bar graph like the one shown below in Figure 2 became a straightforward task.

![Figure 2. A very simple bar graph similar to the first working version of Interface 1.](image)

The Power of Sustainable, Open-Source Code

The vertical Y-axis title on the left-hand side of the graph in Figure 2 appears there by default. In other words, the hundreds of lines of code and hours of design time previously required in order to render a Y-axis title just dropped to zero.

In addition, since JFreeChart has an LGPL open source license, we can eliminate all the code and development time without paying for a license. This illustrates the power of open-source code: it shortens development time, increases productivity, and costs nothing. The more popular the code gets, the better it gets. Higher usage results in faster bug fixing, more features, and faster code.
The developer guide can decrease costs further by greatly reducing the time it takes to learn the library. Even though the guide requires a commercial license, the time required to learn the library decreases enough to make the guide a worthwhile investment. It also provides a sustainable business opportunity based on an open-source model, which encourages more businesses to consider using open source software by countering skepticism about the economics of open source.

**Manipulating JFreeChart with Subclasses and Other Hacks**

While JFreeChart saved countless hours of development time, it also opened up new possibilities for addressing the requirement of making the display interesting. Even though the code that produced Figure 2 had solved all the problems that had existed before JFreeChart, the graph still suffered from monotonous color issues (i.e., too much red). The graph failed to do anything even remotely interesting, yet alone captivating, eye-catching, or cool. Fortunately, the path from Figure 2 to the far nicer graph shown in Figure 3 is neither long nor difficult.

![Figure 3. Interface 1 with custom annotations and vastly improved color scheme in current version.](image_url)

- Group 1 (Lobby): 2.1 K
- Group 2 (Front): 3.7 K
- Group 3 (Upstairs): 2.7 K
**BarRenderer3D → ColorfulBarRenderer3D**

At first, I attempted to address the color issues by searching for options that would change the color of each bar, but found nothing. Looking at the source code for `AbstractRenderer` revealed that colors only changed from one series to another, not from one value to another within the same series. To fix the color scheme, I created a new subclass extending `BarRenderer3D` and overrode the functions `getItemFillPaint` and `getItemPaint` to look up the `Paint` object based on the column instead of the row. In terms of length, the implementation of this subclass consists of two functions, each of which contains only one line of code. When it takes two lines of code to make an improvement this substantial, it illustrates the superior design of the library.

**CategoryItemLabelGenerator → SimpleCategoryItemLabelGenerator**

Creating the custom-formatted annotations at the top of each bar in Figure 3 also requires very few steps. The `SimpleCategoryItemLabelGenerator` implements a simple interface that prints data to 2 significant figures with the `SuffixedNumberFormat.format2` function. One extra step involves a simple call to `setSeriesItemLabelsVisible(true)` on the renderer to enable annotations. The last step involves controlling each annotation based on animation, a much more complex feature explained in the next section.

**Animation and Other Applications of Multithreading**

While JFreeChart supports a wide variety of graph types and options for them, the graphing library lacks one significant feature: direct support for animation. Since I already had substantial experience working with animation by taking CPE 474 with Dr. Chris Buckalew, I saw an opportunity to add animation to the project as an extra specification.

**Fundamentals and Implementation of Animation**

One essential component of animation consists of a parameter that updates in a manner dependent on time. In this project, the `AnimationController` controls the time parameter. By measuring the start time of the current frame and subtracting the start time of the previous frame, the animator obtains the time elapsed between the two frames. The animator then adds the elapsed time to a running total. To calculate the new time parameter, the animator divides this running total by the time desired between the beginning and end of this animation. After calculating the time parameter, a special dataset designed for data animation uses the parameter to control and manipulate the content of a graph.

In a simple animation system running in a single thread, the animator updates the time parameter and renders the corresponding data as quickly as possible. The data changes usually only take a few microseconds to calculate, but the human eye cannot perceive display changes faster than about once every 15 milliseconds. As a result, the data changes far more often than it needs to, leading to an inefficient allocation of processor resources and a substantial waste of power. To fix this problem, the animator and data renderer must run in two separate threads.
Future Implementation – JDBC Database Access

Additionally, implementing database support with JDBC will require adding a third thread due to asynchronous data retrieval; accessing data over a network in the Swing event dispatch thread will lag the user interface, which would lead to a lot of angry and impatient users that would have less interest in the project as a result of their substandard experience.

Overall Project Diagram

Figure 4 on page 15 contains the overall project diagram. Note that the Enphase power inverter connects to several machines at once; unlike the proposed standalone inverter, the Enphase inverter only addresses exercise machines by group.

In the diagram, boxes colored green represent complete parts of the project. Orange boxes represent incomplete, missing, or malfunctioning parts of the project. Each caption in the diagram starts with a dash (‘-’) and denotes either required or highly recommended areas to focus on for future work.

Design Methodology

The course of progress for an engineering project tends to follow one of two general models: the waterfall model and the iterative model.

The Waterfall Model

In the waterfall design model, each stage of development starts and finishes in a purely sequential order. This model characterizes most hardware development due to the high cost required to make changes to a previous step. If a given client changes their requirements during the implementation process in a hardware project, the physical mechanics of the project change. The hardware then requires redesign and reimplementation: a significant investment of time (and money).

The development process recommended on the Senior Projects webpage outlines four main steps that follow the waterfall model. Iterations do not receive mention until Step 4 of the process. For a hardware project, the process makes sense; for a software project such as this one, it led to months of project delays because of a chicken-and-egg problem (the design is the egg, the implementation is the chicken, and the two evolve in a manner quite similar to Darwinian evolution).

The diagram in Figure 5 on page 16 shows the basic idea behind the waterfall model.
Figure 4. Overall project diagram with project scope outlined within the dashed lines.
The Iterative Model

In the iterative model, requirements, design, implementation, and testing take place concurrently. It applies to software more than the waterfall model because changing a program requires nothing more than changing the source code and recompiling. For hardware, the “recompiling” process may take days, weeks, or even months. For software, it takes about 30 seconds.

This monumental difference in compilation time underlies the reason why it makes sense to spend months developing requirements and designs in advance for hardware. For software, it becomes a source of monumental project delays; much more can get done by implementing bits and pieces at a time and designing along the way according to the needs of the project. Figure 6 provides an illustration of the iterative model.

Interestingly, the first paper that first uses the term “waterfall model” to describe a design process also attacks it as risky and prone to failure. This paper constitutes ancient knowledge in the computing world; it has a publication date of 1970, which predates the microprocessor by 3 years.

The senior project guidelines would better serve students if this model received more emphasis. Any future students writing any form of code - even hardware code such as VHDL or assembly code – should not repeat my mistake of making an epic design (i.e., the graph layout in Appendix B) only to have it result in an epic failure after making a small design change.
Television Research

Calculating the Required Television Size

Using trigonometry, the height of a text label can be calculated based on the target distance and desired visual angle. The equation is:

\[ H_Y = 2D \tan\left(\frac{A}{120} \left(\frac{\pi}{180}\right)\right) \]

where \( H_Y \) is the text label height, \( A \) is the angle in arcminutes, and \( D \) is the distance.

Once \( H_Y \) is computed, the total height of the display can be computed by multiplying \( H_Y \) by the ratio calculated above based on the number of Y-axis labels desired for the graph. The full equation is this:

\[ H = (4.945 + 1.5N) (2D) \tan\left(\frac{A}{120} \left(\frac{\pi}{180}\right)\right) \]

Now, the diagonal length of the display can be computed based on the total height and the aspect ratio, which is standardized at 16 units of width to 9 units of height, or 16:9. This length is the minimum size of the television. The final equation is this:

\[ S = (337^{1/2}/9) (4.945 + 1.5N) (2D) \tan\left(\frac{A}{120} \left(\frac{\pi}{180}\right)\right) \]

Using \( N = 5 \), \( D = 480” \), and \( A = 20’ \), the value of \( S \) is equal to about 71”. This is not a small television.
Comparison of Television Types

There are three main types of large televisions: rear-projection (RPTV), plasma, and liquid crystal (LCD). Each has its own set of benefits and drawbacks in terms of price and maintenance.

**Liquid Crystal Displays (LCDs)**

While LCDs exist with dimensions sufficient to meet the needs of the project, they are very high in cost, and very limited in selection once the diagonal size exceeds 55”. LCDs require little maintenance over time, so the initial purchase comprises almost all of the cost. Since the application of LCD technology to large displays is still in a nascent phase, the power efficiency of these new displays is lacking substantially.

- **Price:** 1 out of 5 ($3.2k for 65”, $10.5k for 70”)
- **Maintenance:** 5 out of 5 ($0 in replacement lamps over 5 years)
- **Power efficiency:** 2 out of 5
- **Ease of mounting:** 4 out of 5

**Rear-projection Televisions (RPTVs)**

RPTVs have the lowest initial cost and highest power efficiency out of the three technologies. However, the maintenance on these televisions must occur far more frequently than on an LCD or plasma display. The short lamp life of 3000 to 6000 hours on average for RPTVs translates to a $150-200 lamp replacement every 9 months; 18 hours per day adds up quickly. When the cost of installation and bureaucratic red tape is factored in, the lamp replacement cost increases to $400+, and substantial downtime results. In addition, the DLP lamps contain mercury, which is hazardous.

In addition, this technology is in the process of being phased out. There is only one manufacturer currently making these televisions: Mitsubishi. When Mitsubishi stops making them, likely before the end-of-life of the television, the maintenance issues above will be exacerbated by an obsolete television.

Another issue these televisions have is the difficulty of wall mounting.
Plasma Displays

Plasma displays represent the best value. While their initial cost is twice that of an RPTV, their maintenance is comparable to that of an LCD. They are also over $1.0k cheaper than an LCD. The power efficiency ranks in the middle; plasma displays draw about 100 more watts than RPTVs, but still far less than an LCD of the same size. They are about as easy to mount on a wall as an LCD.

- **Price:** 3.5 out of 5 ($2.0k for 63”, $2.4k for 65”)
- **Maintenance:** 5 out of 5 ($0 in replacement lamps over 5 years)
- **Power efficiency:** 4 out of 5
- **Ease of mounting:** 4 out of 5

Total Cost by Television

Table V compares the cost of 7 different televisions, each of which has a diagonal length measuring 60” (1.52 m) or greater. For reference, a television measuring 60” on the diagonal implies dimensions (W × H) of 52.25” × 29.5” to the nearest quarter inch or 75 cm × 133 cm to the nearest centimeter since all of these televisions have aspect ratios of 16:9.

This table estimates the maintenance and power costs over 5 years. The power cost assumes that electricity costs 12¢/kWh and that the current gym hours (i.e., as of December 1, 2009) remain constant. Values in each column have been rounded to the nearest $25.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Model</th>
<th>Size</th>
<th>Tech</th>
<th>Price</th>
<th>Maint.</th>
<th>Power</th>
<th>Power Cost</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mitsubishi</td>
<td>WD-60735</td>
<td>60”</td>
<td>RPTV</td>
<td>1000</td>
<td>3150</td>
<td>240W</td>
<td>900</td>
<td>5050</td>
</tr>
<tr>
<td>Mitsubishi</td>
<td>WD-65735</td>
<td>65”</td>
<td>RPTV</td>
<td>1250</td>
<td>3150</td>
<td>220W</td>
<td>825</td>
<td>5225</td>
</tr>
<tr>
<td>Samsung</td>
<td>PN63B550</td>
<td>63”</td>
<td>Plasma</td>
<td>2000</td>
<td>0</td>
<td>347W</td>
<td>1300</td>
<td>3300</td>
</tr>
<tr>
<td>Panasonic</td>
<td>TCP65S1</td>
<td>65”</td>
<td>Plasma</td>
<td>2400</td>
<td>0</td>
<td>~325W</td>
<td>1225</td>
<td>3625</td>
</tr>
<tr>
<td>Sony</td>
<td>KDL70XBR7</td>
<td>70”</td>
<td>LCD</td>
<td>10500</td>
<td>0</td>
<td>700W</td>
<td>2625</td>
<td>13125</td>
</tr>
<tr>
<td>Mitsubishi</td>
<td>WD-73736</td>
<td>73”</td>
<td>RPTV</td>
<td>1700</td>
<td>3150</td>
<td>245W</td>
<td>925</td>
<td>5775</td>
</tr>
<tr>
<td>Sharp</td>
<td>LC-65D64U</td>
<td>65”</td>
<td>LCD</td>
<td>3200</td>
<td>0</td>
<td>525W</td>
<td>1975</td>
<td>5175</td>
</tr>
</tbody>
</table>
Table of Television Sizes and Scores

The two tables below assign relative quality scores from 1 to 5 based on two competing metrics: the number of labels on a given axis of a graph, and the angular height each of those labels would have if drawn from a distance of 40 feet.

<table>
<thead>
<tr>
<th>labels</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>4.5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1.5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>angle</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>4.5</td>
</tr>
<tr>
<td>18</td>
<td>3.5</td>
</tr>
<tr>
<td>16</td>
<td>1.5</td>
</tr>
<tr>
<td>15.5</td>
<td>1</td>
</tr>
</tbody>
</table>

Based on the evaluation of several television sizes with different parameters, an equation for optimum size is the following quadratic:

\[ S_A = -0.0909A^2 + 4.0254A - 39.576 \]

The final table for subjective scores sorted by size is shown on the next page.

Television Decision Matrix

The final decision matrix in Table IX rates 7 televisions from 1 to 5 according to economics, environmental friendliness, maintenance level, ease of wall mounting, and size.
Table VIII.
Subjective score calculated from the diagonal length of a television screen.

<table>
<thead>
<tr>
<th>Size</th>
<th>Labels</th>
<th>Angle</th>
<th>Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Angle</td>
</tr>
<tr>
<td>60&quot;</td>
<td>3</td>
<td>22.3053</td>
<td>5.00</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>19.2484</td>
<td>4.23</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>16.9284</td>
<td>2.52</td>
</tr>
<tr>
<td>63&quot;</td>
<td>4</td>
<td>20.211</td>
<td>4.65</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>17.775</td>
<td>3.26</td>
</tr>
<tr>
<td>65&quot;</td>
<td>4</td>
<td>20.852</td>
<td>4.84</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>18.339</td>
<td>3.67</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>16.3664</td>
<td>1.96</td>
</tr>
<tr>
<td>70&quot;</td>
<td>4</td>
<td>22.4564</td>
<td>5.00</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>19.7498</td>
<td>4.98</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>17.6254</td>
<td>4.47</td>
</tr>
<tr>
<td>73&quot;</td>
<td>4</td>
<td>23.4189</td>
<td>5.00</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>20.5962</td>
<td>4.84</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>18.3808</td>
<td>4.77</td>
</tr>
</tbody>
</table>

Table IX.
Final decision matrix for the television.

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Size</th>
<th>Tech.</th>
<th>Ratings</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mitsubishi</td>
<td>WD-60735</td>
<td>60&quot;</td>
<td>RPTV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mitsubishi</td>
<td>WD-65735</td>
<td>65&quot;</td>
<td>RPTV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Samsung</td>
<td>PN63B550</td>
<td>63&quot;</td>
<td>Plasma</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Panasonic</td>
<td>TCP65S1</td>
<td>65&quot;</td>
<td>Plasma</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sony</td>
<td>KDL70XBR7</td>
<td>70&quot;</td>
<td>LCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mitsubishi</td>
<td>WD-73736</td>
<td>73&quot;</td>
<td>RPTV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sharp</td>
<td>LC-65D64U</td>
<td>65&quot;</td>
<td>LCD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The final scores suggest that plasma technology has a superior value when compared with LCD and RPTV. Indeed, despite the fact that an RPTV initially costs less, it would cost more in maintenance over the life of the television than a plasma.

**Appendix A: Senior Project Analysis**

**Summary of Functionality**

This project provides a solid foundation for creating a web interface or scripted display when development progresses far enough in those areas and the final display purchase is made. It provides support for several advanced animation algorithms, user interfaces, and makes several difficult Java concepts such as multithreading, concurrency, and JavaBeans properties easier to work with.
Primary Constraints, Challenges, and Difficulties

Several challenges impeded progress on the project, including organization and planning issues as well as Java programming constructs such as multithreading. In many cases, a problem has more than one solution, but only one or two good and highly efficient solutions, and it can take a long time to find them.

This topic is addressed in more detail in Design Process and Challenges (pages 8-14).

Economic

The only material cost involved in this particular project is the cost of the display for the gym, and that phase of the project ended up delayed to a subsequent quarter. One part of the project consisted of research into the advantages and disadvantages of various types of televisions, which concluded that plasma sets were the most advantageous.

However, this project would have incurred substantial development costs and substantial delays. The total development time for this project is significantly longer than the original estimate, several times over. While the original time estimate specified 8 hours per week over a quarter (80 hours), the actual time taken greatly exceeded 300 hours and may have totaled close to 400 hours, just in the second quarter of the project.

Given that the cost of labor in software amounts to about $50/hour, a discrepancy like this in the business world carries a hefty penalty: anywhere from $11,000 to $16,000. This would add up very quickly for any sizable company, and the current economic depression only makes it worse.

Manufacturing

One commercial fitness center that has successfully utilized renewable energy from exercise is the Green Microgym, based out of Portland. It remains to be seen how much this market will actually grow. However, it is definitely plausible that a commercial venture could succeed in manufacturing interfaces and displays for gyms generating renewable energy.

If commercial demand started to form for this software, dual-licensing makes it possible to sell software licenses or services to companies while maintaining GPL licensing for non-profit organizations.

The principal mission, however, for this software is to educate people about how dangerous or safe different greenhouse gases and energy sources are, and to encourage weight loss by having a nice-looking display system.

Environmental

The project serves several purposes intended to benefit the environment, including:
• Raising awareness and recognition of ongoing environmental issues, including climate change caused by various air pollutants.
• Motivating gym users to exercise to increase their energy output (i.e., competing to try and beat various records, but on an anonymous basis).
• Providing accurate information about the hazards of various chemicals to the environment in an intuitive web interface.

**Appendix B: Unused Designs**

This section contains designs not currently in use but that may be useful to future students developing on the foundation of this project.

**Determining Graph Drawing Panel Height**

Before I discovered the JFreeChart graphing library, I had designed a graph from scratch and calculated the relative heights of various features of the graph. This section showcases this work, which may still be useful for comparing the height of the final graphs (a task which will not take place until the scripted display is implemented in 2010).

**Scripted Display – General Layout**

The general layout of the scripted display, from top to bottom, is as follows:

• Margin above graph title
• Graph title
• Margin below graph title
• Graph window
• Margin above X axis labels
• X axis labels
• Margin below X axis labels and above X axis title
• X axis title
• Margin below X axis title

**FAA Text Height Standards**

The Federal Aviation Administration (FAA) explains how the legibility of text changes based on the visual angle between the observer and the text, denoted by \( \theta \) in Figure 7. The visual angles and levels of legibility are summarized as follows:

• **20-22’ (2.91 – 3.20 mrad)**: Preferred height of dynamic or critical text.
• **18’ (2.62 mrad)**: Minimum height for good text legibility.
• **16’ (2.33 mrad)**: Minimum height for marginal text legibility.
**Height of the Graph Window (H<sub>GR</sub>)**

The graph window has a height that depends on the number of vertical subdivisions on the graph. Each subdividing line has a corresponding Y-axis label, which has a height equal to the height of an X-axis label. In addition, a minimum spacing of half the label height must be maintained between labels on the Y-axis. In addition, the top Y-axis label is centered at the top of the graph window, and the bottom Y-axis label is centered at the bottom of the graph window.

Based on these parameters, the height of the graph window can be calculated:

\[
H_{GR} = 0.5H_Y + 0.5(N - 1)H_Y + (N - 2)H_Y + 0.5H_Y = H_Y(1.5N - 1.5)
\]

where \( H_{GR} \) is the height of the graph window, \( H_Y \) is the height of a label on the X- or Y-axis, and \( N \) is the number of labels on the Y-axis (i.e., the number of vertical subdivisions, plus 2).

**Height of the Graph Title and Margins (H<sub>U</sub>)**

Next, we must determine the relative sizes of the axis labels, axis titles, and graph title. For clear differentiation, axis titles will be 15% larger than axis labels, and the graph title will be 15% larger than the axis titles (32.25% larger than the axis labels).

Based on this information, we can calculate the height of the graph title and its margins, which are equal to half the graph title height on both sides:

\[
H_U = 0.5H_T + H_T + 0.5H_T = 2.0H_T = 2.645H_Y
\]

where \( H_U \) is the total height of everything above the graph window, \( H_T \) is the height of the graph title, and \( H_Y \) is the same label height we used above.
**Height of the X Axis Title and Margins (H_L)**

Finally, we determine the height of the X-axis title and its margins. The margin above the X-axis labels is equal to half the label height \( H_Y \). The margin below the X-axis labels and above the X-axis title is equal to half the X-axis title height, and the margin below the X-axis title is also equal to half the X-axis title height.

Based on this information, we can calculate the height of the X-axis title and its margins. Then, we can calculate the height of everything under the graph window as follows:

\[
H_L = 0.5H_Y + H_Y + 0.5H_{XT} + H_{XT} + 0.5H_{XT} = 1.5H_Y + 2.0H_{XT} = 3.8H_Y
\]

where \( H_L \) is the total height of everything below the graph window, \( H_{XT} \) is the height of the X-axis title, and \( H_Y \) is the same label height we used in the first calculation.

**Height – Grand Total**

Now that each section of the display listed above has had its height calculated, the total height of the display in terms of the label height and the number of Y-axis labels can be calculated:

\[
H = H_Y (4.945 + 1.5N)
\]

A sufficiently informative graph should have at least five Y-axis labels: the bottom and top labels, and three or more labels between them. This places a lower bound on the ratio of label height to screen height: 12.445.

**Possible Design – Time by Academic Quarter**

Since the JFreeChart library is utilized to produce the graphs for the project, an interface will exist through which extra time periods can be defined. Since the start and end dates of quarters at Cal Poly follow a specific pattern, they can be predicted into the future with an accuracy of one week. In addition, the times of year at which this potential inaccuracy comes into play are the same times of year that few people are on campus (i.e., the week before WOW week).

Starting from the fall term, the pattern of start and end dates for each quarter is this:

- **Fall**
  - Start: first Monday after September 10
  - End: first Monday after December 10
- **Winter**
  - Start: first Monday after January 3
  - End: fourth Monday in March (after March 20)
- **Spring**
  - Start: first Monday of the week containing April 1
  - End: third Monday in June
- **Summer**
  - Start: third Monday in June
  - End: first Monday after September 3

These general date estimates allow the display of data by quarter without needing to update an internal calendar every year when the Board of Regents approves the next schedule. Whatever the case may be, the fall term in 2015 will likely start on either September 14 or September 21 unless radical changes to the schedule happen between now and then.

**Appendix C: Kyoto Chemical Table**

Interface 3 contains a popup menu that can select one of 63 greenhouse gas chemicals regulated under the Kyoto Protocol. The menu currently does not change any data, but future work for this project can invoke this table to translate CO\(_2\) offsets (0.718 kg per kilowatt-hour) into offsets for any of the 63 chemicals in the table.

The table will make more sense when meaningful data starts becoming available through the SQL database, a process that will probably take at least another term (Winter 2010).

**CO\(_2\) Equivalence for Various Noxious Chemicals, by Category**

The web interface will contain a drop-down list of 63 chemicals, including CO\(_2\), that are known to cause climate change according to the Intergovernmental Panel on Climate Change (IPCC). The purpose of this list is to raise awareness of different classes of chemicals and to encourage the use of chemicals that will mitigate environmental damage.

The chemicals and the categories they fall into are listed in Table X below. To convert a quantity of a given chemical into an equivalent quantity of CO\(_2\), the interface will multiply the amount of the given chemical by its 100-year global warming potential (GWP) value.

Out of the 63 chemicals on the list, SF\(_6\) has the highest GWP at 22.8k. In other words, releasing only one pound of SF\(_6\) into the atmosphere has the same effect as releasing more than 10 metric tons of CO\(_2\)!

<table>
<thead>
<tr>
<th>Chemical Formula</th>
<th>Lifetime</th>
<th>100-year GWP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carbon dioxide CO(_2)</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>Methane CH(_4)</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td>Nitrous oxide N(_2)O</td>
<td>114</td>
<td>298</td>
</tr>
<tr>
<td><strong>CFCs and other ozone-depleting chemicals</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFC-11 CCl(_3)F</td>
<td>45</td>
<td>4750</td>
</tr>
<tr>
<td>CFC-12 CCl(_2)F(_2)</td>
<td>100</td>
<td>10900</td>
</tr>
<tr>
<td>CFC-13 CClF(_3)</td>
<td>640</td>
<td>14400</td>
</tr>
<tr>
<td>CFC-113 CCl(_2)FCCIF (_2)</td>
<td>85</td>
<td>6130</td>
</tr>
<tr>
<td>CFC-114 CClIF(_3)CCIF (_2)</td>
<td>300</td>
<td>10000</td>
</tr>
<tr>
<td>CFC-115 CClIF(_2)CF (_3)</td>
<td>1700</td>
<td>7370</td>
</tr>
<tr>
<td>Chemical</td>
<td>Formula</td>
<td>Lifetime</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------</td>
<td>----------</td>
</tr>
<tr>
<td>Halon-1301</td>
<td>CBrF₃</td>
<td>65</td>
</tr>
<tr>
<td>Halon-1211</td>
<td>CBrClF₂</td>
<td>16</td>
</tr>
<tr>
<td>Halon-2402</td>
<td>CBrF₂CBrF₂</td>
<td>20</td>
</tr>
<tr>
<td>Carbon tetrachloride</td>
<td>CCl₄</td>
<td>26</td>
</tr>
<tr>
<td>Methyl bromide</td>
<td>CH₃Br</td>
<td>0.7</td>
</tr>
<tr>
<td>Methyl chloroform</td>
<td>CH₃CCl₂</td>
<td>5</td>
</tr>
<tr>
<td>HCFC-22</td>
<td>CHClF₂</td>
<td>12</td>
</tr>
<tr>
<td>HCFC-123</td>
<td>CHCl₂CF₃</td>
<td>1.3</td>
</tr>
<tr>
<td>HCFC-124</td>
<td>CHClF₃CF₃</td>
<td>5.8</td>
</tr>
<tr>
<td>HCFC-141b</td>
<td>CH₃CCl₂F</td>
<td>9.3</td>
</tr>
<tr>
<td>HCFC-142b</td>
<td>CH₂CClF₂</td>
<td>17.9</td>
</tr>
<tr>
<td>HCFC-225ca</td>
<td>CHCl₂CF₂CF₃</td>
<td>1.9</td>
</tr>
<tr>
<td>HCFC-225cb</td>
<td>CHClF₃CF₂CF₂</td>
<td>5.8</td>
</tr>
<tr>
<td><strong>HFCs (hydrofluorocarbons; non-ozone-depleting greenhouse gases)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HFC-23</td>
<td>CHF₃</td>
<td>270</td>
</tr>
<tr>
<td>HFC-32</td>
<td>CH₂F₂</td>
<td>4.9</td>
</tr>
<tr>
<td>HFC-125</td>
<td>CHF₂CF₃</td>
<td>29</td>
</tr>
<tr>
<td>HFC-134a</td>
<td>CH₂FCF₃</td>
<td>14</td>
</tr>
<tr>
<td>HFC-143a</td>
<td>CH₂CF₁</td>
<td>52</td>
</tr>
<tr>
<td>HFC-152a</td>
<td>CH₂CHF₂</td>
<td>1.4</td>
</tr>
<tr>
<td>HFC-227ea</td>
<td>CF₃CHFCF₁</td>
<td>34.2</td>
</tr>
<tr>
<td>HFC-236fa</td>
<td>CF₃CH₂CF₃</td>
<td>240</td>
</tr>
<tr>
<td>HFC-245fa</td>
<td>CHF₃CH₂CF₃</td>
<td>7.6</td>
</tr>
<tr>
<td>HFC-365mfc</td>
<td>CH₂CF₂CH₂CF₃</td>
<td>8.6</td>
</tr>
<tr>
<td>HFC-43-10mee</td>
<td>CF₃CHFCHFCF₂CF₃</td>
<td>15.9</td>
</tr>
<tr>
<td><strong>PFCs (perfluorinated compounds; non-ozone-depleting greenhouse gases)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sulfur hexafluoride</td>
<td>SF₆</td>
<td>3200</td>
</tr>
<tr>
<td>Nitrogen trifluoride</td>
<td>NF₃</td>
<td>740</td>
</tr>
<tr>
<td>PFC-14</td>
<td>CF₄</td>
<td>50000</td>
</tr>
<tr>
<td>PFC-116</td>
<td>C₅F₁₀</td>
<td>10000</td>
</tr>
<tr>
<td>PFC-218</td>
<td>C₆F₁₃</td>
<td>2600</td>
</tr>
<tr>
<td>PFC-318</td>
<td>C₇C₆F₈</td>
<td>3200</td>
</tr>
<tr>
<td>PFC-3-1-10</td>
<td>C₅F₁₀</td>
<td>2600</td>
</tr>
<tr>
<td>PFC-4-1-12</td>
<td>C₆F₁₂</td>
<td>4100</td>
</tr>
<tr>
<td>PFC-5-1-14</td>
<td>C₇F₁₄</td>
<td>3200</td>
</tr>
<tr>
<td>PFC-9-1-18</td>
<td>C₁₀F₁₈</td>
<td>1000</td>
</tr>
<tr>
<td>Trifluoromethyl sulfur pentafluoride</td>
<td>SF₃CF₂</td>
<td>800</td>
</tr>
<tr>
<td><strong>HFEs (hydrofluoroethers; non-ozone-depleting alternative chemicals)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HFE-125</td>
<td>CHF₂OCF₃</td>
<td>136</td>
</tr>
<tr>
<td>HFE-134</td>
<td>CHF₂OCHF₂</td>
<td>26</td>
</tr>
<tr>
<td>HFE-143a</td>
<td>CH₂OCF₃</td>
<td>4.3</td>
</tr>
<tr>
<td>HCFE-235da2</td>
<td>CHF₂OCHCICF₃</td>
<td>2.6</td>
</tr>
<tr>
<td>HFE-245cb2</td>
<td>CHF₂OCF₂CHF₂</td>
<td>5.1</td>
</tr>
<tr>
<td>HFE-245fa2</td>
<td>CHF₂OCH₃CF₃</td>
<td>4.9</td>
</tr>
<tr>
<td>HFE-254cb2</td>
<td>CH₂OCF₂CHF₂</td>
<td>2.6</td>
</tr>
<tr>
<td>HFE-347mcc3</td>
<td>CH₂OCF₂CF₂CF₄</td>
<td>5.2</td>
</tr>
<tr>
<td>HFE-347pcc2</td>
<td>CH₂OCF₂OCH₅CF₃</td>
<td>7.1</td>
</tr>
<tr>
<td>HFE-356pcc3</td>
<td>CH₂OCF₂CF₂CHF₂</td>
<td>0.33</td>
</tr>
<tr>
<td>HFE-449sl</td>
<td>C₆F₁₀OCH₃</td>
<td>3.8</td>
</tr>
<tr>
<td>HFE-569sf2</td>
<td>C₆F₁₀OC₃H₄</td>
<td>0.77</td>
</tr>
<tr>
<td>HFE-43-10pccc124</td>
<td>CHF₂OCF₂OC₂F₂OCHF₂</td>
<td>6.3</td>
</tr>
<tr>
<td>HFE-236ca12</td>
<td>CHF₂OCF₂OCHF₂</td>
<td>12.1</td>
</tr>
</tbody>
</table>
### Chemical Formula Lifetime 100-year GWP

<table>
<thead>
<tr>
<th>Chemical</th>
<th>Formula</th>
<th>Lifetime</th>
<th>100-year GWP</th>
</tr>
</thead>
<tbody>
<tr>
<td>HFE-338pcc13</td>
<td>CHF₂OCF₂CF₂OCHF₂</td>
<td>6.2</td>
<td>1500</td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PFPMIE</td>
<td>CF₃OCF(CF₃)CF₂OCF₂OCF₃</td>
<td>800</td>
<td>10300</td>
</tr>
<tr>
<td>Dimethylether</td>
<td>CH₃OCH₃</td>
<td>0.015</td>
<td>1</td>
</tr>
<tr>
<td>Methylene chloride</td>
<td>CH₂Cl₂</td>
<td>0.38</td>
<td>8.7</td>
</tr>
<tr>
<td>Methyl chloride</td>
<td>CH₃Cl</td>
<td>1</td>
<td>13</td>
</tr>
</tbody>
</table>


### CO₂ Equivalence of Various Energy Sources

Carbon dioxide emission is expressible in terms of energy output as well. The EPA lists some of the most common energy expenses and their equivalent in terms of CO₂, which are summarized in Table XI.

Table X and Table XI can be used to express the power output from one or more exercise machines as any other form of energy or greenhouse gas emission.

**Table XI.**

<table>
<thead>
<tr>
<th>Energy expenditure</th>
<th>CO₂ equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 kWh of electricity</td>
<td>0.718 kg</td>
</tr>
<tr>
<td>1 gallon of gasoline</td>
<td>8.81 kg</td>
</tr>
<tr>
<td>1 barrel of crude oil</td>
<td>430 kg</td>
</tr>
</tbody>
</table>

Source: [http://www.epa.gov/RDEE/energy-resources/refs.html](http://www.epa.gov/RDEE/energy-resources/refs.html)

### Appendix D: Source Code

The latest version of the source code is available at [http://kenai.com/projects/ehfem](http://kenai.com/projects/ehfem). The current version as of December 14, 2009 is provided here as well.
edu.calpoly.ee.ehfem.MainInterface

/*
 * MainInterface.java
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 * This file is part of Exercise Power Grid Interface.
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */

package edu.calpoly.ee.ehfem;

import com.jgoodies.looks.plastic.Plastic3DLookAndFeel;
import edu.calpoly.ee.ehfem.bargraph.PowerBarGraph;
import edu.calpoly.ee.ehfem.linegraph.PowerHistoryLineGraph;
import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import static edu.calpoly.ee.ehfem.linegraph.PowerHistoryLineGraph.yesterday;

/**
 * This file contains the main interface. Running the <code>main()</code>
 * function here will run the entire project. Each interface can be accessed
 * using the previous and next display buttons in the navigation panel at the
 * bottom of each screen.
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */

public class MainInterface extends JFrame implements ActionListener {
    /** Serialization constant. */
    private static final long serialVersionUID = -8195372553470270015L;

    private NavigationPanel navPanel; //The navigation panel.
    private JPanel mainPanel; //The main panel holds the navigation panel and the graph panel.
    private JPanel graphPanel; //The panel that holds each graph.
    private int curDisp = 0; //The current display index.

    //The canonical names for each display or interface that can be drawn.
    private String [] displays = {"BarGraph", "LineGraph", "OffsetTable"};

    /**
     * Performs the actions tied to the buttons in the navigation panel (namely,
     * changing the display). If the source of an action cannot be determined, a
     * MajorIssueException will be thrown to indicate the presence of a problem.
     * @param e the event information related to the action that just happened
     */
public void actionPerformed(ActionEvent e) {
    if (e.getSource().equals(navPanel.getPrevButton())) {
        System.out.println("*** Going back to previous display ***");
        curDisp = (cur Disp == 0) ? displays.length - 1 : curDisp - 1;
        CardLayout clt = (CardLayout) graphPanel.getLayout();
        clt.show(graphPanel, displays[curDisp]);
    } else if (e.getSource().equals(navPanel.getNextButton())) {
        System.out.println("*** Going forward to next display ***");
        curDisp = (curDisp == displays.length - 1) ? 0 : curDisp + 1;
        CardLayout clt = (CardLayout) graphPanel.getLayout();
        clt.show(graphPanel, displays[curDisp]);
    } else {//"throw" an exception
        new MajorIssueException(e.getSource().getClass() + ": Navigation "+ "is undefined! (MainInterface.actionPerformed)!");
    }
}

/**
 * Creates the main interface. Sets the look and feel to Plastic 3D in the
 * JGoodies library (because it looks nice).
 */
public MainInterface() {
    super("Main Interface");
    try {
        UIManager.setLookAndFeel(new Plastic3DLookAndFeel());
    }
    catch (Exception e) {}
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    mainPanel = new JPanel();
    mainPanel.setLayout(new BorderLayout());

    graphPanel = new JPanel();
    graphPanel.setLayout(new CardLayout());
    graphPanel.add(new PowerBarGraph(), displays[0]);
    graphPanel.add(new PowerHistoryLineGraph(yesterday()), displays[1]);
    graphPanel.add(new OffsetTable(), displays[2]);

    navPanel = new NavigationPanel();
    navPanel.getPrevButton().addActionListener(this);
    navPanel.getNextButton().addActionListener(this);

    mainPanel.add(graphPanel, BorderLayout.CENTER);
    mainPanel.add(navPanel, BorderLayout.SOUTH);

    getContentPane(mainPanel);
    pack();
    setVisible(true);
}

/**
 * This main function executes first - all it does is start a new thread in
 * which the main interface is created and executed.
 * @param args any command-line arguments to pass (none are processed)
 */
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            // Interface creation and execution
        }
    });
}
```java
public void run()
{
    new MainInterface();
}
}

edu.calpoly.ee.ehfem.MajorIssueException

/**
 * MajorIssueException.java
 */
import javax.swing.*;

public class MajorIssueException extends Exception
{
    /** Serialization constant. */
    private static final long serialVersionUID = -7690824885840294877L;

    private static final String mWinLF;  //The class for Windows look and feel

    /**
     * Create a new MajorIssueException with the default error text, then quit
     * the program.
     */
    public MajorIssueException()
    {
        try
        {
            UIManager.setLookAndFeel(mWinLF);
        }
    }
```
catch (Exception ui) {}

JOptionPane.showMessageDialog(null, "A major issue occurred!\n\nProgram will now quit.", "Major Issue", JOptionPane.ERROR_MESSAGE);

System.exit();

/**
 * Create a new MajorIssueException with the given error text, then quit the
 * program. A dialog box will appear with a relevant error message. This is
 * used to bring major issues to the immediate attention of anyone in a
 * position to deal with and fix them.
 * @param errorText a String giving some information about the error that
 * occurred that might help solve the problem
 */

public MajorIssueException(String errorText)
{
    try
    {
        UIManager.setLookAndFeel(mWinLF);
    } catch (Exception ui) {}

    JOptionPane.showMessageDialog(null, "A major issue occurred with the " + 
    "following message:\n\n" + errorText + "\n\nProgram will now quit.", 
    "Major Issue", JOptionPane.ERROR_MESSAGE);
    System.exit();
}

/**
 * This main function is used to test MajorIssueException invocations. It
 * prints a summary of the purpose of the class before terminating.
 * @param args the arguments (there aren't any!)
 */

public static void main(String[] args)
{
    new MajorIssueException("Major issues can be thrown with this class\n" + 
    "to make themselves apparent instantly and hopefully shorten the\n" + 
    "time it takes to get rid of them!");
}

edu.calpoly.ee.ehfem.NavigationPanel

/*
 * NavigationPanel.java
 *
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 * This file is part of Exercise Power Grid Interface.
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */

package edu.calpoly.ee.ehfem;
import java.awt.Dimension;
import java.awt.GridLayout;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * The NavigationPanel is a simple JPanel that contains two buttons to navigate
 * between each display. It appears at the bottom of the screen when in
 * interactive (i.e., not scripted) mode.
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */
public class NavigationPanel extends JPanel
{
    /** Serialization constant. */
    private static final long serialVersionUID = -4805270146500861376L;

    private JButton prevButton; //the Previous Display button
    private JButton nextButton; //the Next Display button

    /**
     * Creates the navigation panel with a grid layout and border space to keep
     * the buttons from getting too close together or too big, since the widths
     * of the buttons add up to far less than the width of the interface.
     */
    public NavigationPanel()
    {
        prevButton = new JButton("Previous Display");
        nextButton = new JButton("Next Display");

        GridLayout gl = new GridLayout(0, 2);
        setLayout(gl);

        setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
        add(prevButton);
        add(nextButton);
        gl.setHgap(20); //Set a horizontal gap of 20 pixels between the buttons.
        gl.setLayoutContainer(this);
        setPreferredSize(new Dimension(750, 52));
    }

    /**
     * Gets a reference to the previous display button.
     * @return the previous display button
     */
    public JButton getPrevButton()
    {
        return this.prevButton;
    }

    /**
     * Gets a reference to the next display button.
     * @return the next display button
     */
    public JButton getNextButton()
    {
        return this.nextButton;
    }

    /**
     * A main() function to test and make sure the navigation panel is working
     * as expected.
     * @param args the arguments to this function (currently none are needed)
     */
public static void main(String[] args) {
    JFrame frame = new JFrame("test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    NavigationPanel navPanel = new NavigationPanel();
    frame.setContentPane(navPanel);
    frame.pack();
    frame.setVisible(true);
}

edu.calpoly.ee.ehfem.OffsetTable

/*
 * OffsetTable.java
 *
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 *
 * This file is part of Exercise Power Grid Interface.
 *
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */
package edu.calpoly.ee.ehfem;

import edu.calpoly.ee.ehfem.chemical.ChemLabel;
import edu.calpoly.ee.ehfem.chemical.DateLabel;
import edu.calpoly.ee.ehfem.image.PollutionFileList;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.Image;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import org.jdesktop.swingx.JXFrame;
import org.jdesktop.swingx.swingx.JXGlassBox;
import org.jdesktop.swingx.swingx.JXImagePanel;
import org.jdesktop.swingx.swingx.JXLabel;
import org.jdesktop.swingx.swingx.JXPanel;

public class OffsetTable extends JXImagePanel {
    /** Serialization constant. */
    private static final long serialVersionUID = 1459337220074653631L;

    private ChemLabel[] mChemLabels = new ChemLabel[3];
    private DateLabel mDateLabel = new DateLabel();
    private JXLabel mTitleLabel;
    private JXPanel mLabelPanel;
    private JXLabel[] mTableLabels = new JXLabel[6];
    private double[] mCarbonOffsets = new double[] { 1060000, 165000, 226000 };
private ImageIcon mImageIcon;

void initializeLabels()
{
    mChemLabels[0] = new ChemLabel();
    mChemLabels[0].setForeground(new Color(0, 0, 255));
    mChemLabels[1] = new ChemLabel();
    mChemLabels[1].setForeground(new Color(0, 0, 255));
    mChemLabels[2].setForeground(new Color(0, 0, 255));
    mTitleLabel = new JXLabel("Chemical and Energy Offsets");
    mTitleLabel.setFont(new Font("Arial", Font.BOLD, 16));
    mTableLabels[0] = mChemLabels[0];
    mTableLabels[1] = new JXLabel("Electricity Total: 1.48 MWh");
    mTableLabels[2] = mDateLabel;
    mTableLabels[3] = new JXLabel("Electricity Since 10/12/2009: 230 kWh");
    mTableLabels[4] = mChemLabels[2];
    mTableLabels[5] = new JXLabel("Electricity Today: 32.9 kWh");
    mLabelPanel = new JXGlassBox(0.0f);
    mLabelPanel.setLayout(new GridLayout(3, 2));
    for (int i = 0; i < 6; i++)
    {
        mLabelPanel.add(mTableLabels[i]);
    }
}

public OffsetTable()
{
    super();
    initializeLabels();
    setLayout(new BorderLayout());
    setEditable(false);
    setAlpha(0.4f);
    setPreferredSize(new Dimension(750, 420));
    setStyle(Style.SCALED);
    setDoubleBuffered(false);  

    String fileName = PollutionFileList.getFileName();
    try
    {
        mImageIcon = new ImageIcon(fileName);
        Image image = mImageIcon.getImage();
        this.setImage(image);
    }
    catch (Exception e)
    {
        String mName = "OffsetTable (constructor)";
        String issue = String.format("%s
Major issue with loading %s
", mName, fileName);
        new MajorIssueException(issue);
    }

    add(mTitleLabel, BorderLayout.NORTH);
    add(mLabelPanel, BorderLayout.CENTER);
}

public static void main(String[] args)
{
    SwingUtilities.invokeLater(new Runnable()
    {
        public void run()
        {
            JXFrame frame = new JXFrame("Test Interface 3");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           OffsetTable table = new OffsetTable();
            frame.setContentPane(table);
            frame.pack();
            frame.setVisible(true);
        }
    });
}
edu.calpoly.ee.ehfem.animation

edu.calpoly.ee.ehfem.animation.Animatable

/*
 * Animatable.java
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 * This file is part of Exercise Power Grid Interface.
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */

package edu.calpoly.ee.ehfem.animation;

/**
 * Animatable objects include any objects whose data changes rapidly as a
 * function of the standard time parameter, most commonly ranging from 0 to 1.
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */

public interface Animatable
{
  /**
   * Animate the object to the given time parameter.
   * @param timeParam the time parameter
   */
  void animate(double timeParam);
}

edu.calpoly.ee.ehfem.animation.AnimatedCategoryDataset

/*
 * AnimatedCategoryDataset.java
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 * This file is part of Exercise Power Grid Interface.
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 */
You should have received a copy of the GNU General Public License along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.

package edu.calpoly.ee.ehfem.animation;

import edu.calpoly.ee.ehfem.data.SimpleCategoryDataset;
import org.jfree.data.category.CategoryDataset;

/**
 * The AnimatedCategoryDataset integrates all previous bar graph animation functionality into a single (@link CategoryDataset)-based dataset, which significantly reduces the complexity of animations down to the implementation of a single interface, (@link Animatable), and the presence of a (@link AnimationController), which invokes a separate thread for any logic operations required by the animation and keeps it out of the Swing event dispatch thread and avoids lagging the interface.
 * There is a bit of trickery involved in creating the animations - this class actually contains two separate datasets. One dataset holds the current values of the animation; the other dataset holds the actual data values. All the getter methods are then overridden to get their data from the animated dataset rather than from the super class.
 * To get the actual data values in this dataset, call <code>getLabelValue</code> instead.
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */
public class AnimatedCategoryDataset extends SimpleCategoryDataset implements Animatable {

/** Serialization constant. */
private static final long serialVersionUID = -7668673185780539084L;

/** Dataset with time-dependent values updated each frame. */
private SimpleCategoryDataset animatedDataset;

/** The animation controller to which this dataset attaches. */
private AnimationController animationController;

/** The value of the time parameter, which is updated on every call to the <code>animate()</code> function. */
private double mTimeParam;

/** Create a new animated category dataset, attaching a default animation controller to it. */
public AnimatedCategoryDataset()
{
    super();
    animatedDataset = new SimpleCategoryDataset();
    animationController = new AnimationController(this);
}

/**
 * Animate this dataset by using the new <code>timeParam</code> to calculate the new dataset values. For the power bar graph, this just computes a simple linear interpolation.
 * @param timeParam the new time parameter
 */
public void animate(double timeParam)
{
    if (timeParam < this.mTimeParam)
    {
        animatedDataset.clear();
    }
}
this.mTimeParam = timeParam;
int count = super.getItemCount();
for (int i = 0; i < count; i++) {
    double value = super.getValue(i).doubleValue();
    double high = super.getRangeUpperBound(true);
    double low = super.getRangeLowerBound(true);
    double barLevel = (high - low) * timeParam + low;
    value = Math.min(barLevel, value);
    animatedDataset.insertValue(i, super.getKey(i), value);
}
fireDatasetChanged(); //get all listeners to update with the new frame

/**
 * Get the current value in the animated dataset. Overrides the default
 * getter function to trick related classes into using the animated value.
 * @param item the index of the animated item
 * @return the current value associated with the item
 */
@override
public Number getValue(int item)
{
    return animatedDataset.getValue(item);
}

/**
 * Get the current value in the animated dataset. Overrides the default
 * getter function to trick related classes into using the animated value.
 * @param key the key for the animated item
 * @return the current value associated with the key
 */
@override
public Number getValue(Comparable key)
{
    return animatedDataset.getValue(key);
}

/**
 * Get the current value in the animated dataset. Overrides the default
 * getter function to trick related classes into using the animated value.
 * @param row any integer - placeholder for backwards compatibility
 * @param column the index for the animated item
 * @return the current value associated with the index
 */
@override
public Number getValue(int row, int column)
{
    return this.getValue(column);
}

/**
 * Get the current value in the animated dataset. Overrides the default
 * getter function to trick related classes into using the animated value.
 * @param rowKey any value - placeholder for backwards compatibility
 * @param colKey the key associated with the value
 * @return the current value associated with the index
 */
@override
public Number getValue(Comparable rowKey, Comparable colKey)
{
    return this.getValue(colKey);
}

/**
 * Get the actual value (i.e., not the temporary animated one) from the
 * dataset at the given index. Note that this function calls the superclass.
public Number getLabelValue(int item)
{
    return super.getValue(item);
}

/**
 * Get the actual value (i.e., not the temporary animated one) from the
 * dataset at the given index. Note that this function calls the superclass.
 * @param key the key that the value is associated with
 * @return the actual value in the dataset
 */
public Number getLabelValue(Comparable key)
{
    return super.getValue(key);
}
private double mTimeParam; //the current time parameter (updated each frame)

private int mItemCount; //the current item count

/**
 * Gets the current time parameter.
 * @return the current time parameter
 */
public synchronized double getTimeParam()
{
    return this.mTimeParam;
}

/**
 * Sets the time parameter to a new value.
 * @param timeParam the new time parameter
 */
public synchronized void setTimeParam(final double timeParam)
{
    this.mTimeParam = timeParam;
}

/**
 * Sets the current item count to a new value.
 * @param itemCount the new item count
 */
public synchronized void setItemCount(final int itemCount)
{
    this.mItemCount = itemCount;
}

/**
 * Returns the current item count. The series index is currently ignored
 * (only index 0 is being animated to maximize performance)
 * @param series the index of the series, currently meaningless
 * @return the current item count for series 0
 */
@Override
public synchronized int getItemCount(int series)
{
    return this.mItemCount;
}

/**
 * Creates a new AnimatedTimeSeriesCollection with an empty data array.
 */
public AnimatedTimeSeriesCollection()
{
    super();
    animationController = new AnimationController(this);
}

/**
 * Creates a new AnimatedTimeSeriesCollection by placing the given
 * TimeSeries into index 0 on the underlying TimeSeriesCollection.
 * @param series the TimeSeries to animate
 */
public AnimatedTimeSeriesCollection(TimeSeries series)
{
    super(series);
    animationController = new AnimationController(this);
}

/**
 * Animate the dataset to the given time parameter.
 *
public void animate(double timeParam)
{
    if (timeParam < this.mTimeParam)
    {
        this.mItemCount = 0;
    }
    this.mTimeParam = timeParam;
    double first = super.getStartXValue(0, 0);
    double last = super.getStartXValue(0, super.getItemCount(0) - 1);
    double enablePos = Math.min(timeParam, 1.0);
    double curLast = first + (last - first) * enablePos;
    this.mItemCount = (timeParam < 1) ? findItemCount(curLast) :
                        super.getItemCount(0);
    //System.out.println(String.format("Item count: %d", this.mItemCount));
    fireDatasetChanged();
}

public int findItemCount(double finalTime)
{
    int start = getItemCount(0);
    int end = super.getItemCount(0);
    for (int i = start; i < end; i++)
    {
        if (finalTime < super.getXValue(0, i))
        {
            return i;
        }
    }
    return start;
}

public int getParentItemCount(int series)
{
    return super.getItemCount(series);
}

public double getRangeUpperBound(boolean nada)
{
    double maxValue = 0;
    int count = super.getItemCount(0);
    for (int i = 0; i < count; i++)
    {
        double nextValue = super.getYValue(0, i);
        maxValue = Math.max(nextValue, maxValue);
    }
    double maxDigits = Math.log10(maxValue);
    double maxDigit1 = maxDigits - Math.floor(maxDigits);
    double maxPower10 = Math.pow(10, Math.floor(maxDigits));
}
double maxMult = Math.ceil(Math.pow(10, maxDigit1));
double maxBound = maxPower10 * maxMult;
return maxBound;

/**
  * Since no negative power can ever be generated, the lower range bound will
  * always be equal to zero.
  * @param nada included for backwards compatibility, does nothing
  * @return the lowest power possible, which is zero
  */
public double getRangeLowerBound(boolean nada)
{
    return 0;
}

/**
  * Returns the Range from the lower bound (0) to the calculated upper bound.
  * @param nada included for backwards compatibility, does nothing
  * @return the range of power values
  */
public Range getRangeBounds(boolean nada)
{
    return new Range(getRangeLowerBound(nada), getRangeUpperBound(nada));
}

edu.calpoly.ee.ehfem.animation.AnimationController

/*
 * AnimationController.java
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 * This file is part of Exercise Power Grid Interface.
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */
package edu.calpoly.ee.ehfem.animation;

import edu.calpoly.ee.ehfem.data.SuffixedNumberFormat;
import java.util.Date;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import static java.util.concurrent.TimeUnit.*;
import static java.lang.System.*;

/**
 * An AnimationController helps manage animation threads, allowing for smoother
 * frame transitions and avoidance of interference across threads and lag caused
 * by routines being executed in the wrong place (e.g. the Swing event dispatch
 * thread - execute animations there and you will experience hella lag!).
 * */
public class AnimationController {
    class AnimationTask implements Runnable {
        public void run() {
            // The total animation cycle time includes the time spent animating
            // plus the time spent halted at the end.
            double totalTime = mAnimatedLength + mConstantLength;

            // The maximum time parameter is not 1.0 - when it goes above 1.0,
            // it's a signal that the last frame should remain on the screen.
            double maxTimeParam = totalTime / mAnimatedLength;

            // Calculate the time difference as accurately as possible (in ns)
            long nanosNow = System.nanoTime();
            double timeDiff = (nanosNow - mLastTimeNanos) / mAnimatedLength;
            mLastTimeNanos = nanosNow;

            // Update the time parameter - if over the maximum, reset it to zero.
            mTimeParam += timeDiff / 1.0e9;
            mTimeParam = (mTimeParam > maxTimeParam) ? 0 : mTimeParam;

            // Finally, animate the attached Animatable object.
            mTargetAnimation.animate(mTimeParam);

            // Count up the frame and note the time accumulated.
            mCount++;
            mTimeAcc += timeDiff / 1.0e9;

            // Every 25th frame, print the time difference and current FPS. The
            // frame rate should be quite high; in my tests it got about 180.
            if (mCount == 25) {
                double fps = mCount / mTimeAcc;
                // out.println(String.format("Time diff: \%s\"FPS: \%.3f",
                // SuffixNumberFormat.format2(timeDiff/1.0e9, 4, false), fps));
                mCount = 0;
                mTimeAcc = 0;
            }
        }
    }

    private AnimationTask mAnimator; // the animation task being controlled
    private ScheduledExecutorService mScheduler; // the animation task scheduler
    private ScheduledFuture<?> mHandle; // the handle to the animation scheduler
    private double mAnimatedLength = 3.0; // the animation time period in seconds
    private double mConstantLength = 2.0; // the time to halt at the end in sec.
    private double mTimeParam = 0; // the current time parameter
    private long mLastTimeNanos; // the last time value, in nanoseconds
    private Animatable mTargetAnimation; // the target of animation (the dataset)
    private long mCount = 0; // the running frame count
    private double mTimeAcc = 0.0; // the time accumulated since last FPS count

    /**
     * Create a new AnimationController for the given Animatable object and use
     * the default times for animation length and time to halt at the end; these
     * values are 3.0 and 2.0 seconds, respectively.
     */
@param animation an Animatable dataset

public AnimationController(Animatable animation)
{
    this(animation, 3.0, 2.0);
}

/**
 * Create a new AnimationController for the given Animatable object and use
 * the given animation length and given time to halt at the end.
 *
 * @param animation an Animatable dataset
 * @param animationTime the time it should take to get from the first frame
 * to the last frame
 * @param haltTimeAtEnd the time to hold the last frame before restarting
 * the animation
 */
public AnimationController(Animatable animation, double animationTime, double haltTimeAtEnd)
{
    this.mTargetAnimation = animation;
    this.mTimeParam = 0;
    this.mAnimator = new AnimationTask();
    this.mScheduler = Executors.newScheduledThreadPool(1);
    this.mLastTimeNanos = System.nanoTime();
    this.mHandle = mScheduler.scheduleAtFixedRate(mAnimator, 16, 16, MILLISECONDS);
    this.mAnimatedLength = animationTime;
    this.mConstantLength = haltTimeAtEnd;
}

/**
 * Stops the animation by shutting down the thread.
 */
public void stop()
{
    mScheduler.shutdown();
}

/**
 * Starts the animation again by restarting the thread.
 */
public void start()
{
    if (mScheduler.isShutdown() || mScheduler.isTerminated())
    {
        mHandle = mScheduler.scheduleWithFixedDelay(mAnimator, 16, 16, MILLISECONDS);
    }
}

edu.calpoly.ee.ehfem.bargraph

edu.calpoly.ee.ehfem.bargraph.ColorfulBarRenderer3D

/*
 * ColorfulBarRenderer3D.java
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 * This file is part of Exercise Power Grid Interface.
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 */
package edu.calpoly.ee.ehfem.bargraph;

import java.awt.Paint;
import org.jfree.chart.renderer.category.BarRenderer3D;

/**
 * A more colorful version of the standard BarRenderer3D. The standard
 * renderer will only change the color for separate series rather than for each
 * category of data. Since there is only one series in a given bar graph, the
 * standard implementation only uses one color, generating a boring graph.
 *
 * ColorfulBarRenderer3D changes the implementation by looking up the proper
 * color based on the column (which changes from category to category) instead
 * of the row (which always has index 0).
 *
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */
public class ColorfulBarRenderer3D extends BarRenderer3D {
    /** Serialization constant. */
    private static final long serialVersionUID = -197393936113247112351;

    /**
     * Get the Paint fill value for the current data value. Row is ignored as it
     * is moot for the graph.
     *
     * @param row whatever row the function was called with
     * @param column the index value of the desired category
     * @return Paint a Paint object set to a color, which is specified through
     * internal mappings in the superclass.
     */
    @Override
    public Paint getItemFillPaint(int row, int column) {
        return lookupSeriesFillPaint(column);
    }

    /**
     * Get the Paint item value for the current data value. Row is ignored as it
     * is moot for the graph.
     *
     * @param row whatever row the function was called with
     * @param column the index value of the desired category
     * @return Paint a Paint object set to a color, which is specified through
     * internal mappings in the superclass.
     */
    @Override
    public Paint getItemPaint(int row, int column) {
        return lookupSeriesPaint(column);
    }
}

edu.calpoly.ee.ehfem.bargraph.PowerBarGraph

/*
 * PowerBarGraph.java
 *
 * Copyright © 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 *
 * This file is part of Exercise Power Grid Interface.
 */
* Exercise Power Grid Interface is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
* Exercise Power Grid Interface is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
* You should have received a copy of the GNU General Public License
* along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
*
package edu.calpoly.ee.ehfem.bargraph;
import edu.calpoly.ee.ehfem.animation.AnimatedCategoryDataset;
import edu.calpoly.ee.ehfem.data.SuffixedNumberFormat;
import java.awt.Dimension;
import java.awt.Font;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.CategoryAxis;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;
/**
 * The PowerBarGraph display consists of a bar graph which shows the current
 * power generation across each group of exercise machines. This display will
 * roughly take the form of <a href=original_design/gui_bar_graph.png>this
 * concept design</a> when fully implemented.
 *
 * Current functionality consists of the following:
 * - Rendering of the sample group names and values.
 * - Functional previous and next display buttons when run from the main
 *   class, MainInterface.java.
 * - The bar graph itself is fully functional; each bar is rendered with a
 *   colorful, animated, three-dimensional look and feel provided by a
 *   special dataset and renderer derived from the JFreeChart library.
 *
 * Remaining functionality consists of the following:
 * TODO: Link the values and groups to serial numbers provided by the MySQL
 * database that will run on <a href=decade.ee.calpoly.edu>Decade</a>.
 * TODO: Calculate the maximum instantaneous power generated by all groups over
 * the total lifespan of each group and show the value on the chart as a
 * horizontal line.
 *
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */
public class PowerBarGraph extends JPanel
{
    /** Serialization constant. */
    private static final long serialVersionUID = -6172144917100318529L;

    /** Default names and values. */
    private String[] names = {"Lobby", "Front", "Upstairs"};
    private int[] maxValues = {2100, 3700, 2700};

    /** Graph and axis titles. */
    private String title = "Current Power by Room";
    private Font titleFont = new Font("Arial", Font.BOLD, 16);
    private String yTitle = "Power (W)";
    private String xTitle = "Groups";
/** The dataset, renderer, label generator, and plot. */
private AnimatedCategoryDataset dataset = new AnimatedCategoryDataset();
private ColorfulBarRenderer3D renderer = new ColorfulBarRenderer3D();
private SimpleCategoryItemLabelGenerator generator = new SimpleCategoryItemLabelGenerator(2);
private CategoryPlot plot;

/** Temporary dataset initializer - obsolete when values are provided */
{
    for (int i = 0; i < names.length; i++)
    {
        dataset.setValue("Group " + (i + 1) + ", " + names[i] + ", " + maxValues[i]);
    }
}

/** Gets the minimum height value (this is obsolete now) */
private int getMinHeightValue()
{
    return (int) dataset.getRangeLowerBound(true);
}

/** Gets the maximum height value (this is obsolete now) */
private int getMaxHeightValue()
{
    return (int) dataset.getRangeUpperBound(true);
}

/**
 * Generate the chart using default parameters.
 */
private JFreeChart generateChart()
{
    CategoryAxis xAxis = new CategoryAxis(xTitle);
    NumberAxis yAxis = new NumberAxis(yTitle);
    yAxis.setRange(getMinHeightValue(), getMaxHeightValue());
    SuffixedNumberFormat fmt = new SuffixedNumberFormat(2);
    this.plot = new CategoryPlot(this.dataset, xAxis, yAxis, this.renderer);
    this.plot.setOrientation(PlotOrientation.VERTICAL);
    renderer.setBaseItemLabelGenerator(generator);
    renderer.setSeriesItemLabelsVisible(0, true);
    //PollutionSelector psel = new PollutionSelector();
    //BufferedImage bufimage = psel.getRandomImage();
    //Image image = Toolkit.getDefaultToolkit().createImage(bufimage.getSource());
    JFreeChart chart = new JFreeChart(title, titleFont, plot, false);
    //chart.setBackgroundImage(image);
    return chart;
}

/**
 * Generates a chart and a ChartPanel to hold it, then adds the ChartPanel
 * to this class.
 */
public PowerBarGraph()
{
    JFreeChart chart = generateChart();
    ChartPanel panel = new ChartPanel(chart);
    panel.setPreferredSize(new Dimension(750, 420));
    add(panel);
    //animator = new DataAnimator(40);
    //animator.start();
}
/*
 * A separate main() function for testing only the bar graph.
 * @param args No arguments are needed for this function.
 */
public static void main(String[] args)
{
    SwingUtilities.invokeLater(new Runnable()
    {
        public void run()
        {
            PowerBarGraph graph = new PowerBarGraph();
            ApplicationFrame frame = new ApplicationFrame("Power Bar Graph");
            frame.setContentPane(graph);
            frame.pack();
            RefineryUtilities.centerFrameOnScreen(frame);
            frame.setVisible(true);
        }
    });
}

edu.calpoly.ee.ehfem.bargraph.SimpleCategoryItemLabelGenerator

/**
 * The SimpleCategoryItemLabelGenerator generates the labels on the bar graph
 * when each of the bars reaches its maximum during the animation.
 * TODO: Add units to the end of the labels for watts.
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */
public class SimpleCategoryItemLabelGenerator
    implements CategoryItemLabelGenerator, Serializable
{
    /** Serialization constant. */
    private static final long serialVersionUID = 654665631969067081L;

    import edu.calpoly.ee.ehfem.animation.AnimatedCategoryDataset;
    import java.io.Serializable;
    import org.jfree.chart.labels.CategoryItemLabelGenerator;
    import org.jfree.data.category.CategoryDataset;
    import org.jfree.util.PublicCloneable;
    import static edu.calpoly.ee.ehfem.data.SuffixedNumberFormat.format2;
/** The number of significant figures desired in the label. Default: 2 */
private int mSigfigs = 2;

/**
 * Create a new SimpleCategoryItemLabelGenerator with the given number of
 * significant figures desired for each value.
 * @param sigfigs the number of significant figures
 */
public SimpleCategoryItemLabelGenerator(int sigfigs)
{
  this.mSigfigs = sigfigs;
}

/**
* Checks the value at the given column to see if it has reached its maximum
* or if it is still increasing. If it has reached its maximum, the value
* will be annotated toward the top of the appropriate bar on the graph.
* If the dataset is not an animated dataset, it prints the annotated value
* (since the value will not change and it has reached its maximum by
* default).
* @param dataset the {@link CategoryDataset} to check and/or annotate
* @param column the index of the column to check and/or annotate
* @return String the annotated value (if ready to print), or a blank string
*/
public String generateColumnLabel(CategoryDataset dataset, int column)
{
  if (dataset instanceof AnimatedCategoryDataset)
  {
    AnimatedCategoryDataset animSet = (AnimatedCategoryDataset) dataset;
    double labelValue = animSet.getLabelValue(column).doubleValue();
    double animValue = animSet.getValue(column).doubleValue();
    if (animValue >= labelValue)
    {
      return format2(labelValue, mSigfigs, false);
    }
  }
  double value = dataset.getValue(0, column).doubleValue();
  return format2(value, mSigfigs, false);
}

/**
* Generates the label for the given column (there are no rows in the Simple
* Category data structures since there is only one series). In other words,
* the row parameter is simply ignored/dropped.
* @param dataset the dataset to generate the label for if it has been
* determined that a label should be printed
* @param row the row, which only exists for backward compatibility with the
* JFreeChart library
* @param column the column, which is the index of the value being checked
* @return String the label if the value is ready for one, or a blank string
*/
public String generateLabel(CategoryDataset dataset, int row, int column)
{
  return generateColumnLabel(dataset, column);
}

/**
* Returns an empty string as there are no row labels. This function only
* exists to be backward compatible with the CategoryItemLabelGenerator
* interface.
* @param dataset the dataset to try generating the label for
* @param row the index of the row (highly unlikely to be unequal to 0)
* @return an empty string
*/
public String generateRowLabel(CategoryDataset dataset, int row) {
    return "";
}

edu.calpoly.ee.ehfem.chemical

edu.calpoly.ee.ehfem.chemical.Chemical

/**************************************************
 * A Chemical consists of a chemical name, a chemical formula, and a 100-year
 * global warming potential (GWP) value used to express its equivalent in
 * CO$_2$. For example, one of the 63 chemicals in the Kyoto chemical
 * list has the chemical name "CFC-11", the formula "CCl$_3$F", and a
 * GWP of 4750 (4.75 kg CO$_2$ to 1.00 g CCl$_3$F; in American
 * units, 1 ounce of CFC-11 equates to about 300 pounds of CO$_2$!).
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 **************************************************

public class Chemical {
    private String mName; //this chemical's name
    private String mFormula; //this chemical's formula, in HTML format
    private double mGWP; //the number of units of CO2 to 1 unit of this chemical

    //This holds the dimensions of each field (name, formula, GWP) when JLabel
    //objects are created with them.
    private Dimension [] mLabelSizes = new Dimension[4];

    /**
     * Constructs a new Chemical given a regular name in English, a formula in
     * standard format, and a GWP value, which should be the 100-year GWP. See
     * pages 10-12 of the Senior Project Report for the reference table.
     * @param name - the standard chemical name
     * @param formula - the chemical formula, which we will convert to HTML
     * @param gwp - the global warming potential (GWP) of this chemical
     */

public Chemical(String name, String formula, double gwp)
{
    this.mName = name;
    this.mFormula = formula;
    this.mGWP = gwp;
    calculateSizes();
}

/**
 * Calculate the size of each field (name, formula, GWP) and store it for
 * future reference as a private variable local to this Chemical instance.
 * The last field, mLabelSizes[3], contains the total width and maximum
 * height for this Chemical object.
 */
private void calculateSizes()
{
    mLabelSizes[0] = ChemUtils.getLabelSizeHTML(mName);
    String formulaHTML = ChemUtils.toHTMLFormula(mFormula, true);
    mLabelSizes[1] = ChemUtils.getLabelSize(formulaHTML);
    String gwpLabel = getGWPText();
    mLabelSizes[2] = ChemUtils.getLabelSizeHTML(gwpLabel);
}

/**
 * Get the standard English chemical name associated with this chemical.
 *
 * @return String - the standard chemical name
 */
public String getName()
{
    return this.mName;
}

/**
 * Get the standard ASCII-formatted version of the chemical formula.
 *
 * @return String - the standard ASCII-formatted version of the chemical
 * formula
 */
public String getFormula()
{
    return this.mFormula;
}

/**
 * Get the HTML-formatted version of the chemical formula. The HTML version
 * has subscripted numbers, like a chemical formula should when written.
 * This version allows the entire HTML string to be constructed for the
 * purpose of determining the width of the formatted chemical formula.
 *
 * @param createHeaders - true if the entire HTML string is to be
 * constructed, otherwise false
 * @return String - the HTML-formatted version of the chemical formula with
 * subscripted numbers
 */
public String getFormulaHTML(boolean createHeaders)
{
    return ChemUtils.toHTMLFormula(getFormula(), createHeaders);
}

/**
 * Get the HTML-formatted chemical formula, but don't put in the headers.
 *
* @return String - the subscripted chemical formula in HTML format
 */
public String getFormulaHTML()
{
    return ChemUtils.toHTMLFormula(getFormula(), false);
}

/**
 * Get the global warming potential (GWP) of this chemical.
 *
 * @return double - the GWP of the chemical
 */
public double getGWP()
{
    return this.mGWP;
}

/**
 * Get the global warming potential (GWP) of this chemical and format it to
 * 3 significant figures in exponential notation with the appropriate SI
 * suffix.
 *
 * @return String - the formatted GWP of the chemical
 */
public String getGWPText()
{
    SuffixedNumberFormat fmt = new SuffixedNumberFormat(3);
    String gwpLabel = fmt.format(mGWP);
    return gwpLabel;
}

/**
 * Get the maximum height of all the fields in this Chemical.
 *
 * @return int - the maximum height
 */
public int getHeight()
{
    return mLabelSizes[3].height;
}

/**
 * Get the total width of the 3 fields that comprise a Chemical object when
 * each field is assembled into a JLabel.
 *
 * @return int - the total width of all the fields
 */
public int getTotalWidth()
{
    return mLabelSizes[3].width;
}

/**
 * Get the width of the field on the right that contains the GWP of the
 * chemical.
 *
 * @return int - the width of the GWP value when formatted and assembled
 * into a JLabel
 */
public int getGWPWidth()
{
    return mLabelSizes[2].width;
}

/**
 * Get the width of the formula when subscripted, formatted into HTML, and
 * placed into a JLabel object.
 *
 * @return int - the width of the formatted chemical formula when formatted
 * and assembled into a JLabel
 */
public int getFormulaWidth() {
    return mLabelSizes[1].width;
}

/**
   * Get the width of the chemical name - for consistency, this is also
   * formatted into HTML although the difference here should be negligible.
   * @return int  - the width of the standard chemical name when converted to
   *    HTML and assembled into a JLabel
   */
public int getNameWidth() {
    return mLabelSizes[0].width;
}

---

```
edu.calpoly.ee.ehfem.chemical.ChemicalList

/*
 * ChemicalList.java
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 * This file is part of Exercise Power Grid Interface.
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */

package edu.calpoly.ee.ehfem.chemical;

import java.awt.Dimension;
import java.util.Vector;

/**
    * A ChemicalList incorporates a list of Chemicals in a Vector object, along
    * with a category (e.g., "Ozone-depleting chemicals" for the CFCs).
    * @author Alex Chernetz <achernet-AT-calpoly.edu>
    */
public class ChemicalList extends Vector<Chemical> {

    /** Serialization constant. */
    private static final long serialVersionUID = -8904951593775985000L;

    private String mCategory; //the category that encompasses these chemicals
    private Dimension mCategorySize; //the size of a JLabel with this category

    private int mTargetWidth = 0; //the target width for a given ChemicalList
    
    /**
     * Constructs a new ChemicalList with the given category name.
     * @param category the new category name
     */
```
public ChemicalList(String category)
{
    super();
    this.mCategory = category;
    this.mCategorySize = ChemUtils.getLabelSizeHTML(category);
}

/**
 * Get the category name.
 * @return the category name
 */
public String getCategory()
{
    return this.mCategory;
}

/**
 * Set the category name to some new name.
 * @param category the new category name
 */
public void setCategory(String category)
{
    this.mCategory = category;
    this.mCategorySize = ChemUtils.getLabelSizeHTML(category);
}

/**
 * Get the width of the category encompassing this ChemicalList.
 * @return the width of the category
 */
public int getCategoryWidth()
{
    return this.mCategorySize.width;
}

/**
 * Get the widest chemical name in this list.
 * @return the length of the widest name in the list
 */
public int getWidestName()
{
    int widest = 0;
    for (int i = 0; i < this.size(); i++)
    {
        Chemical next = this.get(i);
        int nextWidth = next.getNameWidth();
        widest = Math.max(nextWidth, widest);
    }
    return widest;
}

/**
 * Get the widest chemical formula in this list.
 * @return the length of the widest chemical formula in the list
 */
public int getWidestFormula()
{
    int widest = 0;
    for (int i = 0; i < this.size(); i++)
    {
        Chemical next = this.get(i);
        int nextWidth = next.getFormulaWidth();
        widest = Math.max(nextWidth, widest);
    }
    return widest;
}
/**
 * Get the widest GWP value (when rendered as a String).
 * @return the length of the widest GWP value in the list
 */
public int getWidestGWP()
{
    int widest = 0;
    for (int i = 0; i < this.size(); i++)
    {
        Chemical next = this.get(i);
        int nextWidth = next.getGWPWidth();
        widest = Math.max(nextWidth, widest);
    }
    return widest;
}

/**
 * Calculates and returns the maximum text height of all the chemicals in
 * this list.
 * @return the maximum height of all the chemicals
 */
public int getMaxHeight()
{
    int height = 0;
    for (int i = 0; i < this.size(); i++)
    {
        Chemical next = this.get(i);
        int nextHeight = next.getHeight();
        height = Math.max(nextHeight, height);
    }
    return height;
}

/**
 * Get the maximum width of all the chemicals in the list by adding the
 * widest chemical name, the widest chemical formula, and the widest GWP
 * value and returning the sum.
 * @return the sum of the widest chemical name, formula, and GWP
 */
public int getMaxWidth()
{
    int maxWidthTotal = getMaxNameWidth() + getMaxFormulaWidth() + getWidestGWP();
    mTargetWidth = Math.max(maxWidthTotal, mTargetWidth);
    return maxWidthTotal;
}

/**
 * Sets the target width to render the chemical list at if the given target
 * width is greater than the maximum width for this list.
 * @param targetWidth the target width at which to render this chemical list
 */
public void setTargetWidth(int targetWidth)
{
    this.mTargetWidth = Math.max(targetWidth, mTargetWidth);
}

/**
 * Gets the current target width value.
 * @return the current target width value
 */
public int getTargetWidth()
{
    return this.mTargetWidth;
}

/**
 * Build a table in HTML at the given index, which will be used when the
 * chemical list is being displayed in the popup menu.
 * @param index the index at which to build the next HTML table
 * @return String the text to place into the menu item that displays this
 * chemical, in full HTML format.
 */
public String buildHTMLTable(int index)
{
    //The smallest width the table can have without jacking it up is equal
    //to the largest widths in each of the 3 table columns.
    int maxNameWidth = getWidestName();
    int maxFormulaWidth = getWidestFormula();
    int maxGWPWidth = getWidestGWP();
    int totalWidth = maxNameWidth + maxFormulaWidth + maxGWPWidth;
    int targetWidth = getTargetWidth();
    if (targetWidth > totalWidth) {
        int deltaWidth = targetWidth - totalWidth;
        double nameRatio = (double) maxNameWidth / (double) totalWidth;
        double formulaRatio = (double) maxFormulaWidth / (double) totalWidth;
        double gwpRatio = (double) maxGWPWidth / (double) totalWidth;
        maxNameWidth += deltaWidth * nameRatio;
        maxFormulaWidth += deltaWidth * formulaRatio;
        maxGWPWidth += deltaWidth * gwpRatio;
        totalWidth = targetWidth;
    }

    Chemical chem = this.get(index);
    String name = chem.getName();
    String formula = ChemUtils.toHTMLFormula(chem.getFormula(), false);
    String gwp = chem.getGWPText();
    String htmlTable = String.format("<html><body><table border="0" width="%d"><tr>
" +
    "<td align="left" valign="middle" width="%d" nowrap>%s</td>
" +
    "<td align="left" valign="middle" width="%d" nowrap>%s</td>
" +
    "<td align="right" valign="middle" width="%d" nowrap>%s</td>
" +
    "</tr></table></body></html>", totalWidth, maxNameWidth, name,
    maxFormulaWidth, formula, maxGWPWidth, gwp);
    return htmlTable;
}

edu.calpoly.ee.ehfem.chemical.ChemicalTestMenu2

/*
 * ChemicalTestMenu2.java
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 * This file is part of Exercise Power Grid Interface.
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 */
package edu.calpoly.ee.ehfem.chemical;

import com.jgoodies.looks.Options;
import com.jgoodies.looks.common.PopupMenuLayout;
import com.jgoodies.looks.plastic.PlasticLookAndFeel;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JMenuItem;
import javax.swing.JPopupMenu;
import javax.swing.JTextField;
import javax.swing.UIManager;

/**
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */
public class ChemicalTestMenu2
{
    private static void initJGoodiesUI()
    {
        try
        {
            UIManager.setLookAndFeel(new PlasticLookAndFeel());
        }
        catch (Exception e) {}
    }

    public static JPopupMenu getChemicalMenu()
    {
        JPopupMenu chemPopup = new JPopupMenu("Chemicals");
        PopupMenuLayout l = new PopupMenuLayout(chemPopup, BorderLayout.X_AXIS);
        chemPopup.putClientProperty(Options.NO_ICONS_KEY, Boolean.TRUE);
        chemPopup.putClientProperty(Options.NO_MARGIN_KEY, Boolean.TRUE);
        chemPopup.add(new JMenuItem(toHTML("Carbon dioxide", "CO<sub>2</sub>", "1.00")));
        chemPopup.add(new JMenuItem(toHTML("Methane", "CH<sub>4</sub>", "25.0")));
        chemPopup.add(new JMenuItem(toHTML("Nitrous oxide", "N<sub>2</sub>O", "298")));
        chemPopup.updateUI();
        l.invalidateLayout(chemPopup);
        return chemPopup;
    }

    public static void main(String[] args)
    {
        initJGoodiesUI();
        final JFrame frame = new JFrame("Test 2");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        final JTextField tf = new JTextField();
        frame.add(tf, BorderLayout.CENTER);
        final JPopupMenu theMenu = getChemicalMenu();
        ActionListener al = new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                theMenu.show(frame, 10, 10);
            }
        };
    }
}
KeyStroke keystroke = KeyStroke.getKeyStroke(KeyEvent.VK_PERIOD, 0, false);
tf.registerKeyboardAction(al, keystroke, JComponent.WHEN_FOCUSED);
frame.pack();
frame.setSize(420, 200);
frame.setVisible(true);
}

public static String toHTML(String name, String form, String gwp)
{
    String htmlTable = String.format("<html><body><table border=0 " + "width=420%"+"tr"><td align=left nowrap> %s </td><td align=right " + "nowrap> %s </td><td align=right nowrap> %s </td></tr></table>" + "/</body></html>", 100, name, form, gwp);
return htmlTable;
}

edu.calpoly.ee.ehfem.chemical.ChemicalTestMenu

/*
 * ChemicalTestMenu.java
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 * This file is part of Exercise Power Grid Interface.
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */
package edu.calpoly.ee.ehfem.chemical;
import com.jgoodies.looks.plastic.Plastic3DLookAndFeel;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.plaf.ComponentUI;
import javax.swing.plaf.MenuItemUI;

public class ChemicalTestMenu
    extends JPopupMenu
{
    public ChemicalTestMenu()
    {
        super("Chemicals");
        this.putClientProperty("jgoodies.noIcons", Boolean.TRUE);
        JPanel n = new JPanel(new GridLayout(1, 3));
        JMenuItem next = new JMenuItem("*");
        n.add(new JLabel("Carbon dioxide"));
        n.add(new JLabel("CO\u2082"));
        n.add(new JLabel("1.00"));
        add(new JMenuItem("Carbon dioxide\t CO\u2082\t 1.00"));
        add(new JMenuItem("Methane\t CH\u2084\t 25.0"));
        JMenuItem firstMenu = new JMenuItem("Chemical Category 1");
        JMenuItem chem1 = new JMenuItem("Chemical 1");
        chem1.setActionCommand("sel_chem_1");
        chem1.addActionListener(panel);
    }
}
firstMenu.add(chem1);

firstMenu.addSeparator();

JMenuItem chem2a = new JMenuItem("Chemical 2a");
//chem2a.setActionCommand("sel_chem_2a");
//chem2a.addActionListener(panel);
firstMenu.add(chem2a);
JMenuItem chem2b = new JMenuItem("Chemical 2b");
//chem2b.setActionCommand("sel_chem_2b");
//chem2b.addActionListener(panel);
firstMenu.add(chem2b);
this.add(firstMenu);

public static void main(String[] args)
{
    try
    {
        UIManager.setLookAndFeel(new Plastic3DLookAndFeel());
    }
    catch (Exception e) {}

    final JFrame frame = new JFrame("Carbon Acid Test");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    final JTextField tf = new JTextField();
    frame.add(tf, BorderLayout.CENTER);

    final ChemicalTestMenu theMenu = new ChemicalTestMenu();
    ActionListener al = new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            theMenu.show(frame, 10, 10);
        }
    };
    KeyStroke keystroke = KeyStroke.getKeyStroke(KeyEvent.VK_PERIOD, 0, false);
    tf.registerKeyboardAction(al, keystroke, JComponent.WHEN_FOCUSED);
    frame.setSize(420, 200);
    frame.setVisible(true);
}
edu.calpoly.ee.ehfem.chemical.ChemLabel

/*
 * ChemLabel.java
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 * This file is part of Exercise Power Grid Interface.
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */
package edu.calpoly.ee.ehfem.chemical;

import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeEvent;
import org.jdesktop.beans.AbstractBean;
import org.jdesktop.swingx.JXLabel;
import edu.calpoly.ee.ehfem.data.SuffixedNumberFormat;
import java.awt.Color;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JMenuItem;
import javax.swing.SwingConstants;

/**
 * A (JLabel) subclass designed to compare and contrast the severity of
 * 63 greenhouse gases regulated under the Kyoto Protocol. Right-clicking on
 * the label will bring up a menu containing several categories of chemicals,
 * including chemicals that deplete the ozone layer. Since some of these gases
 * are thousands of times more potent than carbon dioxide (CO<sub>2</sub>),
 * raising awareness about them is crucial in the fight to keep anthropogenic
 * climate change from spiraling out of control.
 *
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */
public class ChemLabel extends JXLabel
{

    //Serialization constant.
    private static final long serialVersionUID = 1000L;

    /**
     * First off, a "bean", or JavaBeans object, connects class data objects
     * to property change events, which take place in a separate background
     * thread called the EDT (event dispatch thread). They can also get very
     * complicated, very quickly.
     * The AbstractBean provides a simple and easy way to work with JavaBeans
     * properties; it comes from the org.jdesktop.beans utility library
     * packaged with SwingX. The ChemLabel class holds their first project
     * implementation; BeanGWP is used for the global warming potential, and
     * BeanFormula is used for the chemical formula.
     */
    public class BeanGWP extends AbstractBean
    {
        private Double mGWP; //the GWP value as a Double object

        /**
         * Sets a new GWP and generates a signal to all currently-listening
         * PropertyChangeListeners that the GWP has changed.
         */
    }
public void setGWP(Double newGWP)
{
    double oldGWP = getGWP();
    this.mGWP = newGWP;
    firePropertyChange("GWP", oldGWP, getGWP());
}

/**
* Gets the current GWP value stored in this object.
* @return the GWP value contained in a Double object
*/
public Double getGWP()
{
    return this.mGWP;
}

/**
* Constructs a new BeanGWP based on the GWP for carbon dioxide
* (CO$_2$), or 1.00 by definition.
*/
public BeanGWP()
{
    this.mGWP = Double.valueOf(1.00);
}

/**
* The BeanFormula handles changes in the chemical formula (which is tied
* to the GWP) initiated by the chemical popup menu (and perhaps scripts
* when the scripted display is implemented).
* @constructor
*/
public class BeanFormula extends AbstractBean
{
    private String mFormula; //the chemical formula
    /**
    * Gets the chemical formula contained in this object.
    * @return the chemical formula contained in this object
    */
    public String getFormula()
    {
        return mFormula;
    }
    /**
    * Sets the chemical formula to a new String and generates a signal
    * to any PropertyChangeListener that are listening.
    * @param formula the new chemical formula
    */
    public void setFormula(String formula)
    {
        String oldFormula = getFormula();
        this.mFormula = formula;
        firePropertyChange("formula", oldFormula, getFormula());
    }
    /**
    * Constructs a new BeanFormula with carbon dioxide as the default.
    */
    public BeanFormula()
    {
        this.mFormula = "CO2";
    }
}
/**
 * The PopupHiliteOpener does exactly that: it highlights and opens popups
 * when the mouse is over the label or when the label is right-clicked,
 * respectively. It has a reference to the main ChemLabel so that the Kyoto
 * popup menu knows where to signal after its final selection (and also so
 * that it remembers what the last choices were).
 */

public class PopupHiliteOpener extends MouseAdapter
{
    //A reference to the ChemLabel that was clicked.
    private ChemLabel mThisLabel;

    /**
     * Called when the mouse button has been pressed. Checks to see if it's
     * the right mouse button; if it is, it brings up the menu.
     * @param e the information needed to process the mouse press
     */
    @Override
    public void mousePressed(MouseEvent e)
    {
        System.out.println("Mouse pressed");
        if (e.getButton() == MouseEvent.BUTTON3)
        {
            KyotoChemicalTable.attach(mThisLabel);
            mPopup.show(mThisLabel, 10, 10);
        }
        e.consume();
    }

    /**
     * Called when the mouse enters inside the label area. Highlights the
     * label and turns it white.
     * @param e some information to help process the mouse entrance event
     */
    @Override
    public void mouseEntered(MouseEvent e)
    {
        System.out.println("Mouse entered");
        setBackground(new Color(0, 0, 255));
        setForeground(new Color(255, 255, 255));
    }

    /**
     * Called when the mouse exits out of the label area. Turns the label
     * back to blue.
     * @param e some information to help process the mouse exit event
     */
    @Override
    public void mouseExited(MouseEvent e)
    {
        System.out.println("Mouse exited");
        setBackground(null);
        setForeground(new Color(0, 0, 255));
    }

    /**
     * Constructs a new PopupHiliteOpener and links it to the given
     * ChemLabel.
     * @param label - the ChemLabel to link the popup handler to,
     * usually &lt;code&gt;this&lt;/code&gt;
     */
}
public PopupHiliteOpener(ChemLabel thisLabel)
{
    this.mThisLabel = thisLabel;
}

/**
 * The ChemChanger is the PropertyChangeListener designed to handle GWP and
 * chemical formula changes. It updates the text and values in the label
 * when activated.
 */
public class ChemChanger implements PropertyChangeListener
{
    /**
     * Called when a property changes. This checks to see what property
     * changed so that the appropriate data can be changed to reflect it.
     *
     * @param event contains the property name and the old and new values
     */
    public void propertyChange(PropertyChangeEvent event)
    {
        if (event.getPropertyName().equals("GWP"))
        {
            Double gwpObj = (Double) event.getNewValue();
            mGWP = gwpObj.doubleValue();
            System.out.println(String.format("GWP has changed to: %s",
                                            SuffixedNumberFormat.format2(mGWP, 3, true)));
        } else if (event.getPropertyName().equals("formula"))
        {
            mFormula = (String) event.getNewValue();
            setText(ChemUtils.toHTMLFormula(mFormula, true));
            System.out.println("Formula is now " + mFormula);
        }
    }
}

private BeanGWP mGWPBean; //the JavaBean for the GWP
private BeanFormula mFormulaBean; //the JavaBean for the formula
private double mGWP = 1.00; //the current GWP (CO_2 = 1.00)
private String mFormula = "CO2"); //the current chemical formula (no HTML)
private PopupHiliteOpener mPopupOpener; //the popup handler
private ChemChanger mChemChanger; //the chemical changer
private final JPopupMenu mPopup; //a link to the popup to display

/**
 * This block executes before the constructor gets called (initialization).
 */
{
    mPopup = KyotoChemicalTable.createMenu();
    mGWPBean = new BeanGWP();
    mFormulaBean = new BeanFormula();
    mPopupOpener = new PopupHiliteOpener(this);
    mChemChanger = new ChemChanger();
}

/**
 * Gets the GWP for the chemical in this label.
 * @return the current GWP
 */
public double getGWP()
{
    return this.mGWP;
}

/**
 * Sets the GWP to a new value and triggers a property change event.
 * @param gwp the new GWP value
 */

public void setGWP(double gwp) {
    mGWPBean.setGWP(Double.valueOf(gwp));
}

/**
* Gets the chemical formula being displayed by the label.
* @return the current chemical formula
*/
public String getFormula() {
    return this.mFormula;
}

/**
* Sets the chemical formula to a new value and triggers a property change event.
* @param formula the new chemical formula
*/
public void setFormula(String formula) {
    mFormulaBean.setFormula(formula);
}

/**
* Constructs a new ChemLabel using carbon dioxide as the default chemical.
* Note that HTML conversion does not take place until the text of the label itself is actually being set.
*/
public ChemLabel() {
    super(ChemUtils.toHTMLFormula("CO2", true));
    mGWPBean.addPropertyChangeListener(mChemChanger);
    mFormulaBean.addPropertyChangeListener(mChemChanger);
    addMouseListener(mPopupOpener);
    addPropertyChangeListener(mChemChanger);
    setHorizontalAlignment(SwingConstants.CENTER);
    setVerticalAlignment(SwingConstants.CENTER);
    setToolTipText("Right-click to select a chemical...");
}

edu.calpoly.ee.ehfem.chemical.ChemUtils

/*
* ChemUtils.java
* Copyright © 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
* This file is part of Exercise Power Grid Interface.
* Exercise Power Grid Interface is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
* Exercise Power Grid Interface is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
* You should have received a copy of the GNU General Public License
* along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
*/
package edu.calpoly.ee.ehfem.chemical;

import java.awt.Dimension;
import javax.swing.JLabel;

/**
 * ChemUtils provides a static library of functions for frequent tasks such as
 * rendering a JLabel with a String and calculating the size of
 * the resulting JLabel, as well as several HTML conversion utilities.
 * This version eliminates many complicated parameters such as size calculation
 * metric (getPreferredSize() appears to have the highest accuracy).
 *
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */
public class ChemUtils
{
    /**
     * Given labelText, create a standard JLabel, calculate its size, and
     * return it in a Dimension object.
     *
     * @param labelText the text to render in the JLabel
     * @return the height and width of the new label object
     */
    public static Dimension getLabelSize(String labelText)
    {
        JLabel label = new JLabel(labelText);
        Dimension prefSize = label.getPreferredSize();
        return prefSize;
    }

    /**
     * Given labelText, create a HTML-enhanced JLabel, calculate its
     * size, and return it in a Dimension object.
     *
     * @param labelText the text to render in the JLabel.
     * @return the height and width of the new label object;
     */
    public static Dimension getLabelSizeHTML(String labelText)
    {
        JLabel label = new JLabel(toHTMLString(labelText));
        Dimension prefSize = new Dimension(label.getPreferredSize());
        return prefSize;
    }

    /**
     * Converts the given label text into a complete HTML-formatted version for
     * use with a Swing component.
     *
     * @param labelText - the text to convert to HTML formatting
     * @param createHeaders true if all the usual markup features are required,
     *  otherwise false.
     */
    public static String toHTMLString(String labelText)
    {
        String htmlStr = "<html><body><table border="0" width="100%"><tr>" +
            "<td align="left" valign="middle" nowrap>" + labelText + "</td>" +
            "</tr></table></body></html>";
        return htmlStr;
    }

    /**
     * Given a chemical formula (e.g., "C02"), return a String that will
     * subscript the numbers when rendered in HTML format (e.g.,
     * "<html>CO<sub>2</sub></html>").
     *
     * This function is declared static since it stands a good chance of being
     * useful in other classes and doesn’t require any Chemical class objects to
     * actually do its job.
     *
     * @param formula - the standard ASCII form of a chemical formula
     * @param createHeaders true if all the usual markup features are required,
     * otherwise false.
     */
public static String toHTMLFormula(String formula, boolean createHeaders) {
    String subScripted = formula.replaceAll("([0-9]+)", "<sub>$1</sub>";
    subScripted = (createHeaders) ? toHTMLString(subScripted) : subScripted;
    return subScripted;
}

edu.calpoly.ee.ehfem.chemical.DateLabel

public class DateLabel extends JXLabel {
    /**
     * The DateHiliteOpener does exactly that: it highlights and opens popups
     * when the mouse is over the label or when the label is right-clicked,
     * respectively. It has a reference to the main DateLabel.
     */
    public class DateHiliteOpener extends MouseAdapter {
        //A reference to the DateLabel that was clicked.
        private DateLabel mThisLabel;
    }
}
/**
 * Called when the mouse button has been clicked. Checks to see if it's
 * the right mouse button; if it is, it brings up the menu.
 * @param e the information needed to process the mouse click
 */

@Override
public void mouseClicked(MouseEvent e)
{
    System.out.println("Mouse clicked");
    if (e.getButton() == MouseEvent.BUTTON3)
    {
        JXMonthView monthView = new JXMonthView();
        JFrame monthFrame = new JFrame();
        monthFrame.setContentPane(monthView);
        monthFrame.pack();
        monthFrame.setVisible(true);
    }
    e.consume();
}

/**
 * Called when the mouse enters inside the label area. Highlights the
 * label and turns it white.
 * FIXME: There should also be a blue box around the label, but for
 * some reason it isn't showing up.
 * @param e some information to help process the mouse entrance event
 */

@Override
public void mouseEntered(MouseEvent e)
{
    System.out.println("Mouse entered");
    setBackground(new Color(0, 0, 255));
    setForeground(new Color(255, 255, 255));
}

/**
 * Called when the mouse exits out of the label area. Turns the label
 * back to blue.
 * @param e some information to help process the mouse exit event
 */

@Override
public void mouseExited(MouseEvent e)
{
    System.out.println("Mouse exited");
    setBackground(null);
    setForeground(new Color(0, 0, 255));
}

/**
 * Constructs a new DateHiliteOpener and links it to the given
 * DateLabel.
 * @param thisLabel - the DateLabel to link the popup handler to,
 * usually &lt;code&gt;this&lt;/code&gt;
 */

public DateHiliteOpener(DateLabel thisLabel)
{
    this.mThisLabel = thisLabel;
}
public class DateBean extends AbstractBean {
    private Date mDate;

    public DateBean() {
        this.mDate = new Date();
    }

    public Date getDate() {
        Date theDate = (Date) mDate.clone();
        return theDate;
    }

    public void setDate(Date other) {
        Date oldDate = (Date) mDate.clone();
        this.mDate = (Date) other.clone();
        firePropertyChange("DateBean", oldDate, getDate());
    }
}

public class DateSaver implements PropertyChangeListener, ActionListener {
    public void propertyChange(PropertyChangeEvent event) {
        if (event.getPropertyName().equals("date")) {
            mDateBean.setDate(mDatePicker.getDate());
        } else if (event.getPropertyName().equals("DateBean")) {
            mDate = (Date) event.getNewValue();
            setText(mDateFormat.format(mDate));
            System.out.println("Date change: " + getText());
        }
    }

    public void actionPerformed(ActionEvent event) {
        if (JXDatePicker.COMMIT_KEY.equals(event.getActionCommand())) {
            Date datePick = mDatePicker.getDate();
            mDateBean.setDate(datePick);
        }
    }
}

private Date mDate;
private static JXDatePicker mDatePicker;
private DateSaver mDateSaver;
private DateFormat mDateFormat;
private DateBean mDateBean;
private DateHiliteOpener mDateMouseHandler;

public DateLabel() {
    super("10/28/2009");
    this.mDate = new Date();
    long weekAgo = this.mDate.getTime() - (1000 * 3600 * 24 * 7);
    this.mDatePicker = new JXDatePicker(new Date(weekAgo));
    this.mDateSaver = new DateSaver();
    this.mDateFormat = DateFormat.getDateInstance(DateFormat.MEDIUM);
    this.mDateBean = new DateBean();
    this.mDateMouseHandler = new DateHiliteOpener(this);
    addMouseListener(this.mDateMouseHandler);
    mDatePicker.addPropertyChangeListener(mDateSaver);
    mDatePicker.addActionListener(mDateSaver);
    mDateBean.addPropertyChangeListener(mDateSaver);
}
this.addPropertyChangeListener(mDateSaver);
mDatePicker.setFormats(this.mDateFormat);
setHorizontalTextPosition(SwingConstants.CENTER);
setVerticalTextPosition(SwingConstants.CENTER);
setHorizontalAlignment(SwingConstants.CENTER);
setVerticalAlignment(SwingConstants.CENTER);
}

edu.calpoly.ee.ehfem.chemical.KyotoChemicalTable

/**
 * KyotoChemicalTable.java
 *
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 *
 * This file is part of Exercise Power Grid Interface.
 *
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */

package edu.calpoly.ee.ehfem.chemical;
import edu.calpoly.ee.ehfem.data.SuffixedNumberFormat;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.ButtonGroup;
import javax.swing.JMenu;
import javax.swing.JPopupMenu;
import javax.swing.JRadioButtonMenuItem;

/**
 * Creates and stores the popup menu containing all the Kyoto chemicals.
 *
 * TODO: The space we save by making this a static class isn't worth the issues
 * it is causing with synchronization - this needs to be fixed
 *
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */

public final class KyotoChemicalTable
{
    public enum ChemCategory
    {
        INITIAL(0), CFC(1), HFC(2), PFC(3), HFE(4), OTHER(5);
        
        private final int mType;

        ChemCategory(int type)
        {
            this.mType = type;
        }
        
        public final int type()
        {
            return this.mType;
        }
    }
}
static class ChemListener implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        String sender = event.getActionCommand();
        int index = 0;
        for (int i = 0; i < mFormulae.length; i++)
        {  
            if (mFormulae[i].equals(sender))
            {
                index = i;
                break;
            }
        }
        mCurrentSelectedFormula = mFormulae[index];
        mCurrentSelectedGWP = mGWPs[index];
        if (mCurrentChemLabel != null)
        {
            mCurrentChemLabel.setFormula(mCurrentSelectedFormula);
            mCurrentChemLabel.setGWP(mCurrentSelectedGWP);
        }
        detach();
        SuffixedNumberFormat fmt = new SuffixedNumberFormat(3);
        System.out.println(String.format("Selected %s with GWP %s", mCurrentSelectedFormula, fmt.format(mCurrentSelectedGWP)));
    }
}

private static final ChemicalList[] mChemicalLists = new ChemicalList[6];
public static ButtonGroup mChemicalButtons = new ButtonGroup();
private static String[] mNames;
private static String[] mFormulae;
private static double[] mGWPs;
private static ChemListener mChemListener = new ChemListener();

private static double mCurrentSelectedGWP = 1.00;
private static String mCurrentSelectedFormula = "CO2";
private static JPopupMenu mInitializedPopupMenu;
private static ChemLabel mCurrentChemLabel;

public static double getSelectedGWP()
{
    return mCurrentSelectedGWP;
}

public static String getSelectedFormula()
{
    return mCurrentSelectedFormula;
}

static
{
    mNames = new String[]
    {
    };
}
mFormulae = new String []
{
};

mGWPs = new double []
{
    1, 25, 298, 4750, 10900, 14400, 6130, 10000, 7370, 7140, 1890, 1640, 1400, 5, 146, 1810, 77, 697, 725, 2310, 122, 595, 14800, 675, 3500, 1430, 4470, 124, 3220, 9810, 1030, 794, 1640, 22800, 17200, 7390, 12200, 8830, 10300, 8860, 9160, 9300, 7500, 17700, 14900, 6320, 756, 350, 708, 659, 359, 575, 580, 110, 297, 59, 1870, 2800, 1500, 10300, 1, 8.7, 13;
};

String [] categoryNames = new String []
{
    "", "CFCs and other ozone-depleting chemicals", "HFCs (" +
    "hydrofluorocarbons; non-ozone-depleting greenhouse gases)", "PFCs (perfluorinated compounds; non-ozone-depleting greenhouse " +
    "gases)", "HFEs (hydrofluorethers; non-ozone-depleting " +
    "alternative chemicals", "Other"
};

int [] categorySizes = new int [] {3, 19, 11, 11, 15, 4};
int total = 0;
for (int i = 0; i < categorySizes.length; i++)
{
    mChemicalLists[i] = new ChemicalList(categoryNames[i]);
    for (int j = 0; j < categorySizes[i]; j++, total++)
    {
        mChemicalLists[i].add(new Chemical(mNames[total], mFormulae[total], mGWPs[total]));
    }
}

public static final ChemicalList getChemicalList(ChemCategory category)
{
    return mChemicalLists[category.type()];
}

/**
 * When the KyotoChemicalTable is first activated, the parent menu is the
 * first menu that becomes visible. It is divided into two sections: the
 * top, which contains chemicals that are common points of reference (e.g.,
 * CO<sub>2</sub>, N<sub>2</sub>O), and the bottom, which
 * contains 5 nested lists of chemicals (one for every category of chemicals
 * regulated under the Kyoto Protocol).
 * This method calculates the required width of the parent menu by taking
 * the maximum widths of the top section and the bottom section and setting
 * the width to the maximum of the two.
 * @return int - the width of the parent menu
 */
public static final int getParentMenuWidth()
{
    ChemicalList chemList = getChemicalList(ChemCategory.INITIAL);
    int topWidth = chemList.getMaxWidth();
    int bottomWidth = 0;
    for (int i = 1; i < mChemicalLists.length; i++)
    {  
        String nextCategory = mChemicalLists[i].getCategory();
        int catWidth = ChemUtils.getLabelSizeHTML(nextCategory).width;
        bottomWidth = Math.max(bottomWidth, catWidth);
    }
    return Math.max(topWidth, bottomWidth);
}

public static final void attach(ChemLabel label)
{
    mCurrentChemLabel = label;
    mCurrentSelectedFormula = label.getFormula();
    mCurrentSelectedGWP = label.getGWP();
}

public static final void detach()
{
    if (mCurrentChemLabel != null)
    {
        mCurrentChemLabel = null;
    }
    mCurrentSelectedFormula = "CO2";
    mCurrentSelectedGWP = 1.00;
}

protected static final JPopupMenu createMenu()
{
    if (mInitializedPopupMenu != null)
    {
        return mInitializedPopupMenu;
    }
    mInitializedPopupMenu = new JPopupMenu("Select a chemical...";
    ChemicalList chemList = getChemicalList(ChemCategory.INITIAL);
    int parentMenuWidth = getParentMenuWidth();
    chemList.setTargetWidth(parentMenuWidth);
    for (int i = 0; i < chemList.size(); i++)
    {
        String nextRow = chemList.buildHTMLTable(i);
        JRadioButtonMenuItem nextItem = new JRadioButtonMenuItem(nextRow);
        nextItem.setSelected(i == 0); //set default to CO2 (at index 0)
        nextItem.setActionCommand(chemList.get(i).getFormula());
        mChemicalButtons.add(nextItem);
        nextItem.addActionListener(mChemListener);
        mInitializedPopupMenu.add(nextItem);
    }
    mInitializedPopupMenu.addSeparator();
    for (int i = 1; i < mChemicalLists.length; i++)
    {
        ChemicalList mChemicalLists[i];
        String nextCategory = chemList.getCategory();
        JMenu nextMenu = new JMenu(nextCategory);
        for (int j = 0; j < chemList.size(); j++)
        {
            String nextRow = mChemicalLists[i].buildHTMLTable(j);
            JRadioButtonMenuItem nextItem = new JRadioButtonMenuItem(nextRow);
            nextItem.setActionCommand(chemList.get(j).getFormula());
            mChemicalButtons.add(nextItem);
            nextItem.addActionListener(mChemListener);
            nextMenu.add(nextItem);
        }
        mInitializedPopupMenu.add(nextMenu);
    }
    return mInitializedPopupMenu;
}
package edu.calpoly.ee.ehfem.data;
import edu.calpoly.ee.ehfem.MajorIssueException;
import java.util.Date;
import org.jfree.data.time.Minute;
import org.jfree.data.time.TimeSeries;
import static java.lang.String.format;

/**
 * This class works with Catmull-Rom splines to interpolate a smooth cubic line
 * over a set of points. In addition, the area under the line can be solved with
 * calculus, providing a useful measure of the total amount of power that has
 * been generated over time.
 *
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */
public class Catmull {

    /** The Catmull-Rom matrix. */
    private static final double [][] CATM_MAT = {
        {-1, 3, -3, 1},  // first row
        {2, -5, 4, -1},  // second row
        {-1, 0, 1, 0},   // third row
        {0, 2, 0, 0}     // fourth row
    };

    /**
     * The length of a standard Catmull-Rom polynomial.
     */
    private static final int POLY_LEN = 4;

    /**
     * Private quick method to get a vector in print form "[ x^3 x^2 x 1]".
     * @param theVector the vector to print
     * @return the print form of the vector
     */
    protected static String printVector(double [] theVector) {
        String str = "[ ";
        for (int i = 0; i < theVector.length; i++)
            str += Long.valueOf(Math.round(theVector[i])).toString() + " ";
        str += " ]";
        return str;
    }
}
/**
 * Derives polyMatrix as a polynomial with coefficients [x^3, x^2, x, 0].
 * The code should work if more than 4 values are provided but it only
 * gets used with 4 values in this code. I will not guarantee bug-free
 * extension to random lists of doubles with random lengths!
 * @param polyMatrix an array of polynomial index values, e.g. [3 5 1 9]
 * @return the derivative of the polynomial, e.g., [9 10 1]
 */
protected static final double[] derive(final double[] polyMatrix) {

double[] derived = new double[polyMatrix.length - 1];
derived[0] = 0;
for (int i = 0; i < derived.length; i++) {
    derived[i] = polyMatrix[i] * (derived.length - i);
}
return derived;
}

/**
 * Multiplies a quad-term polynomial (cubic) with a triple-term polynomial
 * (quadratic) to get a quintic function and return its coefficients in a
 * vector of length 6.
 * @param quadVector the cubic vector with 4 components
 * @param triVector the quadratic vector with 3 components
 * @return an array of length 6 with the multiplied polynomial.
 */
protected static final double[] multiply(final double[] quadVector, final double[] triVector) {

    int multLength = quadVector.length + triVector.length - 1;
double[] multVector = new double[multLength];
for (int i = 0; i < quadVector.length; i++) {
    for (int j = 0; j < triVector.length; j++) {
        int indexX = quadVector.length - 1 - i;
        int indexY = triVector.length - 1 - j;
        int indexMult = multLength - 1 - indexX - indexY;
multVector[indexMult] += (quadVector[i] * triVector[j]);
    }
}
return multVector;
}

/**
 * Integrates the given polynomial from 0 to 1 and returns the coefficients
 * as appropriate. This just divides everything by (6, 5, 4, 3, 2, 1)
 * down the array.
 * @param theVector the vector to integrate
 * @return the definite integral of the given polynomial from T = 0 to 1.
 */
protected static final double[] integrate(final double[] theVector) {

double[] integVector = new double[theVector.length + 1];
for (int i = 0; i < theVector.length; i++) {
    double denom = theVector.length - i; //amount to divide the next coefficient by
    integVector[i] = theVector[i] / denom;
}
return integVector;
}
/**
 * Integrate the curve, add up all the coefficients, and then get the sum.
 *
 * @param theVector the vector to integrate and add up the coefficients on
 * @return the integral of the vector from T = 0 to 1
 */
protected static double integrateAndSum(double[] theVector)
{
    double[] integVector = integrate(theVector);
    double sum = 0;
    for (int i = 0; i < integVector.length; i++)
    {
        sum += integVector[i];
    }
    return sum;
}

/**
 * Given a list of 4 X values and 4 Y values, calculate the integral of the
 * area underneath the curve between the middle two points in both range and
 * domain. This can be used to calculate the amount of power used over time.
 *
 * @param xValues the list X = {X-1, X0, X1, X2}
 * @param yValues the list Y = {Y-1, Y0, Y1, Y2}
 * @return the integral from T = 0 to 1 of the given points
 */
protected static final double integrateCtrlPts(final double[] xValues,
                                               final double[] yValues)
{
    if (xValues == null || yValues == null)
    {
        new MajorIssueException("catmullIntegrate does NOT work with null" +
                               "values!\nSee edu.calpoly.ee.ehfem.data.Catmull.java for the " +
                               "source.");
    }
    if (xValues.length != 4 || yValues.length != 4)
    {
        new MajorIssueException("catmullIntegrate does NOT work with " +
                               "lengths other than 4!\nSee edu.calpoly.ee.ehfem.data.Catmull." +
                               "java for the source.");
    }
    double[] catmX = getCatmul(xValues); // get x(t)
    double[] catmY = getCatmul(yValues); // get y(t)
    double[] catmDX = derive(catmX); // get x'(t)
    double[] catmYDX = multiply(catmY, catmDX); // get x'(t)y(t)
    double catmIntegral = integrateAndSum(catmYDX); // int x'(t)y(t)dt, t, 0, 1
    return catmIntegral;
}

/**
 * Compute the dot product to the number of dimensions equal to the smallest
 * of vec1.length and vec2.length So if vec1.length = 5 and vec2.length = 4,
 * it returns a double[4] with the first 4 terms of vec1 and vec2.
 *
 * @param vec1 the first vector
 * @param vec2 the second vector
 * @return the dot product up to the maximum number of possible dimensions
 */
protected static final double dotP(final double[] vec1, final double[] vec2)
{
    int maxLength = Math.min(vec1.length, vec2.length);
    double answer = 0;
    for (int i = 0; i < maxLength; i++)
    {
        answer += (vec1[i] * vec2[i]);
    }
    return answer;
}
/**
 * Gets the Catmull-Rom cubic polynomial coefficients for the given set of
 * values by multiplying each row of the Catmull-Rom matrix by each of the
 * given <code>values</code> and adding the products together. In other
 * words, v[0]*m[0][0] + v[1]*m[0][1] + v[2]*m[0][2] + v[3]*m[0][3] repeated
 * for m[1], m[2], and m[3].
 * @param values the X or Y values to create a spline for
 * @return an array of 4 coefficient values in (T^3, T^2, T, 1) order
 */
protected static final double[] getCatmull(final double[] values)
{
    if ((values == null) || (values.length != POLY_LEN))
    {
        new MajorIssueException("Null value array or one with more than 4" +
        " values\nwas passed to getCatmull in Catmull.java!");
    }
    double[] catms = new double[POLY_LEN];
    for (int i = 0; i < POLY_LEN; i++)
    {
        catms[i] = dotP(CATM_MAT[i], values) * 0.5;
    }
    //System.out.println("Old array: " + printVector(values) + "\tNew array: " +
    //printVector(catms));
    return catms;
}

/**
 * Creates a new array in which the first and last indices of <code>vals
 * </code> are duplicated.
 * @param vals an array of control points in one dimension
 * @return a new array where vals[0] and vals[vals.length-1] occur twice
 */
protected static final double[] duplicateFirstAndLast(final double[] vals)
{
    int totalLen = vals.length + 2; //Make space for 2 extra values.
    double[] dupVals = new double[totalLen]; //Allocate new array.
    dupVals[0] = vals[0]; //Copy 1st value over from vals.
    //Copy all of vals to dupVals[1]; dupVals[0] now equals dupVals[1].
    System.arraycopy(vals, 0, dupVals, 1, vals.length);
    //Copy the last value in dupVals from the penultimate value in dupVals.
    dupVals[totalLen - 1] = dupVals[totalLen - 2];
    System.out.println(format("Old: %s\nNew: %s\n", vals, dupVals));
    return dupVals;
}

/**
 * Get the control points [index-1, index, index+1, index+2] from vals. If
 * index is equal to 0 or length - 2, the first or last value is duplicated.
 * @param index the index for the second control point (the T = 0 point)
 * @param vals the complete list of values; length must be 2 or more
 * @return the 4 control points
 * @throws MajorIssueException when vals is null or has a length under 2 or
 * when the index is invalid
 */
protected static final double[] getCtrlPtsFrom(final int index, final
double[] vals) throws MajorIssueException
{
    if (vals == null)
    {
        String prob = "getCtrlPtsFrom: No null arrays allowed!";
        throw new MajorIssueException(prob);
    }
    if (vals.length < 2)
    {
        String prob = "getCtrlPtsFrom: Length of vals must be at least 2!";
        throw new MajorIssueException(prob);
    }
    }
    throw new MajorIssueException(prob);
  }
  int maxIndex = vals.length - 2;
  if (index < 0 || index > maxIndex)
  {
    String prob = "getCtrlPtsFrom: Index %d is out of range (0-%d)!";
    throw new MajorIssueException(format(prob, index, maxIndex));
  }
  double[] ctrlPts = new double[POLY_LEN];
  ctrlPts[1] = vals[index];
  ctrlPts[2] = vals[index + 1];
  ctrlPts[0] = (index == 0) ? ctrlPts[1] : vals[index - 1];
  return ctrlPts;

  /**
   * Calculates and integrates the Catmull-Rom function that spans the given
   * X and Y values and returns the total after the integration process is
   * complete.
   * This function is used to calculate the total power output over a
   * continuous period of time given discrete points of known values for power
   * generation.
   * @param xValues the array of X values (i.e., the times)
   * @param yValues the array of Y values (i.e., the discrete power outputs)
   * @return the sum of the integral of the Catmull-Rom curve over each (X, Y)
   * @throws MajorIssueException when arrays are null or unequal in length
   */
  public static final double integrateList(final double[] xValues,
                                            final double[] yValues)
                              throws MajorIssueException
  {
    if (xValues == null || yValues == null)
    {
      throw new MajorIssueException("catmullIntegrateList: No null " +
                               "arrays allowed!");
    }
    if (xValues.length != yValues.length)
    {
      throw new MajorIssueException(format("catmullIntegrateList: " +
                                    "Arrays have unequal lengths (X=%d, Y=%d)!",
                                    xValues.length,
                                    yValues.length));
    }
    //Duplicate first and last X and Y values.
    //double[] xFirstLast2x = duplicateFirstAndLast(xValues);
    //double[] yFirstLast2x = duplicateFirstAndLast(yValues);
    //These will hold each group of X and Y control points.
    //double[] nextX = new double[POLY_LEN];
    //double[] nextY = new double[POLY_LEN];
    double total = 0;
    //Example: 15 original X and Y values translates to 14 iterations. Each
    //duplicated array has 17 entries since the first and last occur twice.
    for (int i = 0; i < (xValues.length - 1); i++)
    {
      //System.arraycopy(xFirstLast2x, i, nextX, 0, 4);
      //System.arraycopy(yFirstLast2x, i, nextY, 0, 4);
      double[] nextX = getCtrlPtsFrom(i, xValues);
      double[] nextY = getCtrlPtsFrom(i, yValues);
      double segment = integrateCtrlPts(nextX, nextY);
      total += segment;
      System.out.println("Segment "+i +": Power: " +
                         Math.round(segment) + " Running total: " + Math.round(total));
    }
    return total;
  }
/**
 * Evaluates a single T value (0.0 to 1.0) along the Catmull-Rom curve
 * given by the four control points in <code>vals</code>.
 * @param tValue the T parameter
 * @param vals the 4 control point values
 * @return the interpolated value at the given T value
 */

protected static final double evalSingleT(final double tValue, final double[] vals)
{
    double[] catmullValues = getCatmull(vals);
    double eval = catmullValues[0] * tValue * tValue * tValue +
                  catmullValues[1] * tValue * tValue +
                  catmullValues[2] * tValue +
                  catmullValues[3];
    return eval;
}

/**
 * Interpolates the given 4 Catmull-Rom control points with <code>precision</code> steps. Returns all <code>precision</code> values, going from T = 0 up to but not including T = 1.
 * @param vals the four control points
 * @param precision the desired precision of the interpolation
 * @return the interpolation from T = 0 to 1
 */

protected static final double[] interpCtrlPts(final double[] vals, final int precision)
{
    double[] interpVals = new double[precision];
    interpVals[0] = vals[1]; //Vals[1] is value at T = 0.
    double curT = 0.0;
    double tIncr = 1.0 / precision;
    int i = 1;
    for (i = 1, curT = tIncr; i < precision; i++, curT += tIncr)
    {
        interpVals[i] = evalSingleT(curT, vals);
    }
    System.out.println(format("%s{%d} = %s", vals, precision, interpVals));
    return interpVals;
}

/**
 * Interpolates each segment of the curve given by <code>vals</code> with <code>precision</code> steps. The new array will have a length equal to <code>(vals - 1) * precision + 1</code>.
 * @param vals the original time series
 * @param precision the desired precision of the interpolation
 * @return the new interpolated time series
 */

public static final double[] interpCurve(final double[] vals, final int precision) throws MajorIssueException
{
    int maxIndex = vals.length - 1;
    int newLen = precision * maxIndex + 1;
    double[] newVals = new double[newLen];
    for (int i = 0; i < maxIndex; i++)
    {
        double[] ctrlPts = getCtrlPtsFrom(i, vals);
        double[] interpPts = interpCtrlPts(ctrlPts, precision);
        int indexNewVals = precision * i;
        System.arraycopy(interpPts, 0, newVals, indexNewVals, precision);
    }
    newVals[newLen - 1] = vals[maxIndex];
    System.out.println(format("interpCurve: %s", newVals));
    return newVals;
}

/**
 * Creates a new time series by interpolating <code>precision</code> values and times over each original time segment. The new array will have the same time range as the original, but it will be far more amenable to animation.
 */
public static final TimeSeries interpTimeSeries(TimeSeries origSeries, int precision) throws MajorIssueException {
    int dataPts = origSeries.getItemCount();
    double[] timeVals = new double[dataPts];
    double[] powerVals = new double[dataPts];
    for (int i = 0; i < dataPts; i++)
    {
        timeVals[i] = origSeries.getTimePeriod(i).getFirstMillisecond();
        powerVals[i] = origSeries.getValue(i).doubleValue();
    }
    double[] newTimeVals = interpCurve(timeVals, precision);
    double[] newPowerVals = interpCurve(powerVals, precision);
    TimeSeries newSeries = new TimeSeries("Catmull-Rom Time Data");
    for (int i = 0; i < newTimeVals.length; i++)
    {
        long nextT = Math.round(newTimeVals[i]);
        newSeries.add(new Minute(new Date(nextT)), newPowerVals[i], false);
    }
    return newSeries;
}

edu.calpoly.ee.ehfem.data.DataTriplet

/*
 * DataTriplet.java
 *
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 * This file is part of Exercise Power Grid Interface.
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */

package edu.calpoly.ee.ehfem.data;

import org.jfree.chart.*;
import org.jfree.chart.axis.*;
import org.jfree.chart.plot.*;
import org.jfree.chart.renderer.xy.*;
import org.jfree.data.*;
import org.jfree.data.category.*;
import org.jfree.data.general.*;
import org.jfree.data.time.*;
import org.jfree.data.xy.*;
import org.jfree.date.*;
import org.jfree.ui.*;
import java.text.*;
import java.awt.*;
import java.util.*;
import javax.swing.*;
public class DataTriplet
{
    private static Random rand;
    private static double MACHINES = 50;

    private static class Watts
    {
        protected static final double MEAN = 100;
        protected static final double MIN_VALUE = 30;  //keep it within 2 standard deviations
        protected static final double MAX_VALUE = 170;  //keep it within 2 standard deviations
        protected static final double STDEV = 28;
    }

    static
    {
        rand = new Random();
    }

    /**
     * getFractionUsed - Taking a normal distribution curve with a mean of 0 and a standard
     * deviation of 1, we can multiply it by our desired standard deviation and add the mean.
     * The values are chosen so that 99% of the time, a value between the low and high will
     * be chosen (i.e., 2.5 standard deviations from the average of the two numbers).
     * Returns a fraction between 0 and 1. This value is used in such instances as guessing
     * the percentage of machines in use.
     * @param low: The target 1st percentile value to shape data to.
     * @param high: The target 99th percentile value to shape data to.
     * @return double A new value generated by the data simulator.
     */
    public static double getFractionUsed(double low, double high)
    {
        double stdev = 0.20 * (low + high);
        double mean = 0.5 * (low + high);
        double gauss = rand.nextGaussian();
        double value = gauss * stdev + mean;
        value = (value > 100) ? 100 : (value < 0) ? 0 : value;
        return value / 100;
    }

    /**
     * generateTotal - Simulates the total power being generated in the Rec Center. This
     * assumes N = 50 machines, which is multiplied by the simulated occupancy percentage.
     * @param low: The 5th percentile estimate of current occupancy.
     * @param high: The 95th percentile estimate of current occupancy.
     * @return double The simulated total instantaneous power in the Rec Center.
     */
    public static double generateTotal(double low, double high)
    {
        return getFractionUsed(low, high) * MACHINES * generateSingle();
    }

    /**
     * TODO: Nothing is in place to actually call this.
     */
    public static TimeSeriesDataItem generateTimeData(Calendar time, double low, double high)
    {
        double frac = getFractionUsed(low, high);
        double power = frac * MACHINES * generateSingle();
        return new TimeSeriesDataItem(new Minute(time.getTime()), power);
    }

    /**
     * generateSingle - Generates a simulated power data point estimating the output for
     * one exercise machine. The parameters used to control the normal distribution are in
     * the nested class Watts above (as of 10/13/2009, it assumes that people have an
     * average power output of 100W and that actual power output might fall under 30W or
     * above 170W for a total of 1% of the time.
     */

public static double generateSingle()
{
    double gauss = Math.abs(rand.nextGaussian() * Watts.STDEV + Watts.MEAN);
    gauss = (gauss < Watts.MIN_VALUE) ? Watts.MIN_VALUE :
             (gauss > Watts.MAX_VALUE) ? Watts.MAX_VALUE : gauss;
    return gauss;
}

private static final double[] defaultTimeList = {
    6, 7, 8, 8.5, 9, 11, 13, 15, 17, 19, 20, 21, 22, 23, 24};

private static final double[] lowRanges = {
    0, 5, 25, 30, 35, 40, 45, 50, 40, 35, 25, 15, 5, 0};

private static final double[] highRanges = {
    5, 15, 35, 45, 40, 55, 60, 70, 55, 50, 40, 30, 15, 10};

//Some sample data I generated that looked reasonable.
//TODO: Make it so that ridiculous variances in the simulated data trigger a discard
private static final double[] samples = {
    17, 1181, 1708, 2214, 1478, 1589, 3293, 2929, 2166, 1811, 2082, 444, 468, 556, 53};

private static final GregorianCalendar today;
private GregorianCalendar date;

static
{
    today = new GregorianCalendar();
    today.setLenient(true);
}

date = (GregorianCalendar) today.clone();

/**
 * Generate an array of simulated power measurements. The data generator will iterate through
 * the times given in the array and generate one power measurement for each time, based on
 * estimated power output values for a single machine.
 *
 * TODO: The measurement generator has a habit of generating data that looks like it came
 *       from a kangaroo on acid. If the first derivative is taken and it changes more than
 * 25% in different directions 5+ times, drop it.
 */
private double[] generateSingleValues(double[] times)
{
    double [] values = new double[times.length];
    for (int i = 0; i < times.length; i++)
    {
        values[i] = DataTriplet.getFractionUsed(lowRanges[i], highRanges[i]) *
                    DataTriplet.generateSingle();
    }
    return values;
}

private double[] generateTotalValues(double[] times)
{
    double [] values = new double[times.length];
    for (int i = 0; i < times.length; i++)
    {
        values[i] = DataTriplet.generateTotal(lowRanges[i], highRanges[i]);
    }
    return values;
}

private void getTimeLists(int startHour, int endHour)
{
    double [] defaults = new double[defaultTimeList.length];
    double defaultLength = defaultTimeList[defaults.length - 1] - defaultTimeList[0];
    double newLength = (double) endHour - startHour;
    double ratio = newLength / defaultLength;
    double oldOffset = defaultTimeList[0];
    for (int i = 0; i < defaults.length; i++)
    {
        defaults[i] = (defaultTimeList[i] - oldOffset) * ratio + startHour;
    }
    double [] values = generateTotalValues(defaults);
    for (int i = 0; i < defaults.length; i++)
    {
        int hours = (int) defaults[i];
        double min = (defaults[i] - hours) * 60;
        String minutes = (min < 10) ? "0" + (int) min : ":" + (int) min;
        System.out.println("Time: " + hours + ":" + minutes + ":Value: " + values[i]);
    }
}

public DataTripletDay()
{
    int weekDay = date.get(Calendar.DAY_OF_WEEK);

    //The gym opens at 6:00 Monday through Friday and 8:00 on weekend days.
    int startHour = (weekDay == Calendar.SATURDAY || weekDay == Calendar.SUNDAY) ? 8 : 6;
// The gym closes at midnight on every day except Saturday, when it
closes at 22:00.
    int endHour = (weekDay == Calendar.SATURDAY) ? 22 : 24;
    getTimeLists(startHour, endHour);
}

public DataTripletDay(int month, int day, int year)
{
    date.set(year, month - 1, day);
    int weekDay = date.get(Calendar.DAY_OF_WEEK);
    int startHour = (weekDay == Calendar.SATURDAY || weekDay == Calendar.SUNDAY) ? 8 : 6;
    int endHour = (weekDay == Calendar.SATURDAY) ? 22 : 24;
    getTimeLists(startHour, endHour);
}

public static void main(String[] args)
{
    DataTripletDay todaysDay = new DataTripletDay();
    DataTripletDay lastSaturday = new DataTripletDay(10, 31, 2009);
}

edu.calpoly.ee.ehfem.data.SimpleCategoryDataset

/*
 * SimpleCategoryDataset.java
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 * This file is part of Exercise Power Grid Interface.
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */
package edu.calpoly.ee.ehfem.data;

import java.io.Serializable;
import java.util.List;
import org.jfree.data.Range;
import org.jfree.data.RangeInfo;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.general.DefaultKeyedValuesDataset;
import org.jfree.util.PublicCloneable;

/**
 * A SimpleCategoryDataset is an optimized CategoryDataset with only one series
 * of values. Because there is only one series, there is only one row, and
 * every inherited method dealing with rows has been overridden to reflect this.
 * The end result is a much faster and cleaner interface.
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */
public class SimpleCategoryDataset extends DefaultKeyedValuesDataset implements CategoryDataset, PublicCloneable, Serializable, RangeInfo {
    private static final long serialVersionUID = 131137923281611405L;

    private static final int ROW_COUNT = 1; //row count is always 1
    private static final int ROW_INDEX = 0; //row index is always 0
    private static final String ROW_KEY = ""; //row key is always empty string

    /** Holds the max bound value - only change it when the dataset changes. */
    public double mMaxBound;

    /**
     * Returns the row count, which is always equal to 1. Rows are obsolete in
     * this dataset.
     *
     * @return the row count, equal to one
     */
    public int getRowCount() {
        return ROW_COUNT;
    }

    /**
     * The column count is equal to the number of groups in the dataset.
     *
     * @return the number of groups
     */
    public int getColumnCount() {
        return super.getItemCount();
    }

    /**
     * Returns the numerical value at the given column. Rows are obsolete but
     * required as a parameter for interfacing and backwards compatibility.
     *
     * @param row formerly the series number (now obsolete)
     * @param column the group number
     *
     * @return the value (i.e. wattage) at the given column
     */
    public Number getValue(int row, int column) {
        return super.getValue(column);
    }

    /**
     * Returns a blank (@link String) since rows are obsolete.
     *
     * @param row formerly the series number (now obsolete)
     * @param key formerly the row key (now obsolete)
     *
     * @return the value ""
     */
    public Comparable getRowKey(int row)
    {
        return ROW_KEY;
    }

    /**
     * Returns 0 - there is never more than one row, and that row has index 0.
     *
     * @param key formerly the row key (now obsolete)
     *
     * @return the row index, equal to zero
     */
    public int getRowIndex(Comparable key) {
        return ROW_INDEX;
    }

    /**
     * Returns a list with one item, which consists of an empty string.
     */
}
* @return the list {""}
*
**
public List getRowKeys()
{
    return Arrays.asList(ROW_KEY);
}

/**
 * Returns the {@link String} mapped to the given column as a Comparable.
 * @param column the column to get the key from
 * @return the key
 */
public Comparable getColumnKey(int column)
{
    return super.getKey(column);
}

/**
 * Returns the index that corresponds to the given key.
 * @param key the key to map to an index
 * @return the index
 */
public int getColumnIndex(Comparable key)
{
    return super.getIndex(key);
}

/**
 * Returns a list with the keys from all the columns in ascending order.
 * @return the list of keys
 */
public List getColumnKeys()
{
    return super.getKeys();
}

/**
 * Returns the value associated with the given (column) key. Once again,
 * rows are ignored.
 * @param rowKey the row key, which is obsolete
 * @param columnKey the column key, which is the actual key
 * @return the value
 */
public Number getValue(Comparable rowKey, Comparable columnKey)
{
    return super.getValue(columnKey);
}

/**
 * Returns the lower bound of the possible range of power output values.
 * Power output is always graphed using zero as the baseline, and there is
 * no negative power, so this is always zero.
 * @param includeInterval can be true or false, does not affect result
 * @return the lower bound of the power output range, which is always zero
 */
public double getRangeLowerBound(boolean includeInterval)
{
    return 0;
}

/**
 * Notify listeners that the dataset has changed, which means that the range
 * of values may have also changed.
 * */
```java
protected void fireDatasetChanged()
{
    this.calculateMaximumBound();
    super.fireDatasetChanged();
}

/**
 * Calculates and saves the upper bound based on the current range of power output values (which is equal to the maximum power in the category, rounded up to the next highest place).
 */
protected void calculateMaximumBound()
{
    double maxValue = 0;
    int count = super.getItemCount();
    if (count == 0) // there's no data to iterate over
    {
        return;
    }
    for (int i = 0; i < count; i++)
    {
        double nextValue = super.getValue(i).doubleValue();
        maxValue = Math.max(nextValue, maxValue);
    }
    double maxDigits = Math.log10(maxValue);
    double maxDigit1 = maxDigits - Math.floor(maxDigits);
    double maxPower10 = Math.pow(10, Math.floor(maxDigits));
    double maxMult = Math.ceil(Math.pow(10, maxDigit1));
    this.mMaxBound = maxPower10 * maxMult;
}

/**
 * Returns the upper bound of the possible range of power output values. This is equal to the maximum power with the most significant digit rounded up.
 * @param includeInterval can be true or false, does not affect result
 * @return the maximum power to the next most significant digit
 */
public double getRangeUpperBound(boolean includeInterval)
{
    return this.mMaxBound;
}

/**
 * Returns a Range with the low bound of zero along with the high bound.
 * @param includeInterval can be true or false, does not affect result
 * @return a Range which runs from 0 past the highest power output value
 */
public Range getRangeBounds(boolean includeInterval)
{
    return new Range(0, this.getRangeUpperBound(true));
}

@Override
public int hashCode()
{
    return super.hashCode();
}

@Override
public Object clone() throws CloneNotSupportedException
{
    SimpleCategoryDataset clone = (SimpleCategoryDataset) super.clone();
    return clone;
}

/**
 * Constructor that doesn't actually do anything.
 */
```
public SimpleCategoryDataset()
{
    super();
}

/*@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final SimpleCategoryDataset other = (SimpleCategoryDataset) obj;
    return true;
}*/

edu.calpoly.ee.ehfem.data.SuffixedNumberFormat

/**
 * SuffixedNumberFormat.java
 *Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 *This file is part of Exercise Power Grid Interface.
 *Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 *it under the terms of the GNU General Public License as published by
 *the Free Software Foundation, either version 3 of the License, or
 *(at your option) any later version.
 *Exercise Power Grid Interface is distributed in the hope that it will be useful,
 *but WITHOUT ANY WARRANTY; without even the implied warranty of
 *MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 *GNU General Public License for more details.
 *You should have received a copy of the GNU General Public License
 *along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */

package edu.calpoly.ee.ehfem.data;

import java.text.*;

public class SuffixedNumberFormat extends NumberFormat
{
    private enum Exactness
    {
        EXACT, INEXACT
    }

    private int sigfigs = 2;
    private String infinity = "\u221E";
    private String nan = "\u00D8";
    private int [] exponents = {-15, -12, -9, -6, -3, 0, 3, 6, 9, 12, 15};

    private String checkDoubleLimits(double number)
    {
        if (number == Double.NaN)
        {
            return nan;
        }
    }
else if (number == Double.POSITIVE_INFINITY)
  {
    return infinity;
  }
else if (number == Double.NEGATIVE_INFINITY)
  {
    return "-" + infinity;
  }
else if (number == 0)
  {
    return "0";
  }
return "";

private DecimalFormat getEngFormatter(Exactness ex)
{
  DecimalFormat df = new DecimalFormat("0E0");
  df.setMaximumIntegerDigits(3);
  int intDigits = Math.min(2, sigfigs);
  df.setMinimumIntegerDigits(intDigits);
  df.setMaximumFractionDigits(sigfigs - intDigits);
  df.setMinimumFractionDigits(sigfigs - intDigits);
  return df;
}

private DecimalFormat getSciFormatter(Exactness ex)
{
  DecimalFormat df = new DecimalFormat("0E0");
  df.setMaximumIntegerDigits(1);
  df.setMinimumIntegerDigits(1);
  df.setMaximumFractionDigits(sigfigs - 1);
  df.setMinimumFractionDigits(sigfigs - 1);
  return df;
}

private double getMinExp()
{
  return Math.pow(10, exponents[0]) - 0.5 * Math.pow(10, exponents[0] - sigfigs);
}

private double getMaxExp()
{
  int len = exponents.length;
  return Math.pow(10, exponents[len-1] + 3) - 0.5 * Math.pow(10, exponents[len-1] - sigfigs);
}

private String getSufStr(String engStr)
{
  String [] splitStr = engStr.split("[eE]\"\"");
  if (splitStr.length != 2)
  {
    System.out.println("Error parsing " + splitStr.toString());
    return "\u00D8";
  }
  int exponent = Integer.parseInt(splitStr[1]);
  if (exponent < exponents[0])
  {
    System.out.println("The exponent " + exponent + " is hella small");
    return "";
  }
  else if (exponent > exponents[exponents.length - 1])
  {
    System.out.println("The exponent " + exponent + " is hella big");
    return "";
  }
  if (splitStr[0].endsWith("."))
  {
    splitStr[0] = splitStr[0].substring(0, splitStr[0].length() - 1);
  }
}
```java
int index = (exponent - exponents[0]) / 3;
splitStr[0] += suffixes[index];
return splitStr[0];
}

public SuffixedNumberFormat()
{
    this.sigfigs = 2;
}

public SuffixedNumberFormat(int sigfigs)
{
    this.sigfigs = sigfigs;
}

public static String format2(double value, int digits, boolean numeric)
{
    int suffixStart = -24;
    int suffixEnd = suffixStart + (suffix.length - 1) * 3;

    String ans = (value == Double.POSITIVE_INFINITY) ? "\u221E" :
        (value == Double.NEGATIVE_INFINITY) ? "-\u221E" :
            (value == Double.NaN) ? "\u03D5" : "";
    if (ans.length() > 0)
    {
        return ans;
    }
    if (value < 0)
    {
        ans.concat("-");
        value = -value;
    }
    if (value == 0)
    {
        return "0";
    }
    int exp10 = (int) Math.log10(value);
    exp10 = (exp10 > 0) ? (exp10 / 3) * 3 : (-exp10 + 3) / 3 * (-3);
    value *= Math.pow(10, -exp10);
    if (value >= 1000)
    {
        value /= 1000;
        exp10 += 3;
    }
    else if (value >= 100)
    {
        digits -= 2;
    }
    else if (value >= 10)
    {
        digits -= 1;
    }
    digits = (digits <= 1) ? 1 : digits;
    boolean noSuf = numeric || (exp10 < suffixStart) || (exp10 > suffixEnd);
    String fmt = "%s" + Integer.toString(digits - 1) + "f";
    fmt += (noSuf) ? "e%d" : "%s";
    String valStr = "";
    if (noSuf)
    {
        valStr = String.format(fmt, value, exp10);
    }
    else
    {
        valStr = String.format(fmt, value, suffix[(exp10-suffixStart) / 3]);
    }
    ans += valStr;
    return valStr;
}
```
public StringBuffer format(double number, StringBuffer toAppendTo, FieldPosition pos)
{
    return format(number, toAppendTo, pos, Exactness.INEXACT);
}

public StringBuffer format(long number, StringBuffer toAppendTo, FieldPosition pos)
{
    return format((double) number, toAppendTo, pos, Exactness.EXACT);
}

public StringBuffer format(double number, StringBuffer toAppendTo, FieldPosition pos,
                Exactness ex)
{
    StringBuffer target = (toAppendTo == null) ? new StringBuffer() : toAppendTo;
    /*String limit = checkDoubleLimits(number);
    if (limit.compareTo("") != 0)
        {*/
        target.append(limit);
    return target;
    /*}
    if (number < 0)
        {*/
        target.append("-");
        number *= -1;
    /*}
    if (number < getMinExp() || number > getMaxExp())
        {
        target.append(getSciFormatter(ex).format(number));
    } else
        {
        String engStr = getEngFormatter(ex).format(number);
        String engSuffixed = getSufStr(engStr);
        target.append(engSuffixed);
    }*/
    return target;
}

public Number parse(String source, ParsePosition parsePosition)
{
    ParsePosition myParsePos = (parsePosition == null) ? new ParsePosition(0) : parsePosition;
    String parseThis = source.substring(myParsePos.getIndex());
    for (int i = 0; i < suffixes.length; i++)
        { parseThis.replaceFirst(suffixes[i], "E+" + exponents[i]); }
    DecimalFormat df = new DecimalFormat();
    return df.parse(parseThis, myParsePos);
}

public static void main(String [] args)
{
    double [] testValues = {69, 420, 8675309, 92960000, -1700, 3200};
    int [] testExactVals = {69, 420, 8675309, 92960000, 3700, -1000};
    SuffixedNumberFormat sf = new SuffixedNumberFormat();
    for (int i = 0; i < testValues.length; i++)
        { String str = sf.format(testValues[i]);
        String intstr = sf.format(testExactVals[i]);
        System.out.println("double: " + str + "\integer: " + intstr);
        str = format2(testValues[i], 4, false);
        intstr = format2(testExactVals[i], 4, false);
        System.out.println("double: " + str + "\integer: " + intstr);
    }
}
package edu.calpoly.ee.ehfem.image;

import java.io.File;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.filechooser.FileNameExtensionFilter;

/**
 * A file chooser specifically designed to load image files that have extensions
 * supported by Java by default.
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */
public class ImageFileChooser extends JFileChooser {
    /** Serialization constant. */
    private static final long serialVersionUID = -1760811005094377638L;

    /** A reference to the file that was chosen. */
    private File mChosenFile;

    /** Image file name extensions included with Java by default. */
    private static FileNameExtensionFilter mJPG;
    private static FileNameExtensionFilter mPNG;
    private static FileNameExtensionFilter mBMP;
    private static FileNameExtensionFilter mGIF;

    /**
     * Initializes the file extension filters in a way that looks clean.
     */
    static {
        mJPG = new FileNameExtensionFilter("JPG files", "jpg", "jpeg");
        mPNG = new FileNameExtensionFilter("PNG files", "png");
        mBMP = new FileNameExtensionFilter("BMP files", "bmp", "wbmp");
        mGIF = new FileNameExtensionFilter("GIF files", "gif");
    }

    /**
     * Construct a new ImageFileChooser.
     */
    public ImageFileChooser() {
        addChoosableFileFilter(mJPG);
        addChoosableFileFilter(mPNG);
    }
}
addChoosableFileFilter(mBMP);
addChoosableFileFilter(mGIF);
setAcceptAllFileFilterUsed(false);
int shown = showDialog(null, "Select");
if (shown == JFileChooser.APPROVE_OPTION)
{
  mChosenFile = getSelectedFile();
  System.out.println(mChosenFile.getName());
}
setSelectedFile(null); //Reset the selection and directory to default.

/**
 * Gets the currently chosen file.
 * @return the (link File) that this file chooser has chosen *
 */
public File getChosenFile()
{
  return this.mChosenFile;
}

/**
 * Resets the chosen file to <code>null</code>.
 */
public void resetChosenFile()
{
  this.mChosenFile = null;
}

/**
 * A <code>main()</code> class to test the file chooser. This has no args.
 * @param args nothing *
 */
public static void main(String [] args)
{
  JFrame frame = new JFrame("Image Chooser");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  new ImageFileChooser();
  frame.add(new JPanel());
  frame.pack();
  frame.setVisible(true);
}

edu.calpoly.ee.ehfem.image.ImageFileFilter

"
package edu.calpoly.ee.ehfem.image;

import java.io.File;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.io.FileFilter;

/**
 * An image file filter highly similar to the one used with the
 * {@link ImageFileChooser} but disallowing the selection of directories.
 * *
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */
public class ImageFileFilter extends javax.swing.filechooser.FileFilter
    implements java.io.FileFilter {
    /** Serializable constant. */
    private static final long serialVersionUID = -1085058092138082622L;

    /** The FileNameExtensionFilter is used for actual extension testing. */
    private FileNameExtensionFilter mImages = new FileNameExtensionFilter("Images", "jpg", "jpeg", "png", "gif", "bmp", "wbmp");

    /** True if directories are allowed, otherwise false. */
    private boolean mDirsAllowed = true;

    /**
     * Constructs a new ImageFileFilter that allows directories to be chosen.
     *
     */
    public ImageFileFilter() {
        this(true);
    }

    /**
     * Constructs a new ImageFileFilter where directories will be disallowed if
     * <code>acceptDirs</code> is false, or allowed otherwise.
     *
     * @param acceptDirs true if directories should be allowed, otherwise false
     */
    public ImageFileFilter(boolean acceptDirs) {
        this.mDirsAllowed = acceptDirs;
    }

    /**
     * If <code>theFile</code> is a directory, returns the boolean value of
     * <code>mDirsAllowed</code>. Otherwise, it will return the result of the
     * <code>accept</code> function call on <code>mImages</code>.  *
     * @param theFile the file to check
     * @return true if the file should be allowed, otherwise false
     */
    @Override
    public boolean accept(File theFile) {
        return theFile.isDirectory() ? mDirsAllowed : mImages.accept(theFile);
    }

    /**
     * The description according to this will always be "Images".
     *
     */
    @Override
    public String getDescription() {
        return "Images";
    }
}
package edu.calpoly.ee.ehfem.image;

import edu.calpoly.ee.ehfem.MajorIssueException;
import java.awt.Dimension;
import java.awt.Image;
import java.io.File;
import java.util.Random;
import java.util.Vector;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.filechooser.FileFilter;
import org.jdesktop.swingx.JXFrame;
import org.jdesktop.swingx.JXImagePanel;
import static java.lang.System.out;

/**
* Maintains a list of file names for air pollution images that can be loaded
* and displayed in the Chemical and Energy Offset Table.
* @author Alex Chernetz <achernet-AT-calpoly.edu>
*/
public class PollutionFileList
{
    /* FileFilter filter = new FileNameExtensionFilter("JPEG file", "jpg", "jpeg");
    JFileChooser fileChooser = ...;
    fileChooser.addChoosableFileFilter(filter);*/
    private static final String mDefaultDir = "airpollute";
    private static Vector<String> mFileList;
    private static final Random mRand = new Random();
    private static FileFilter mImageFilter;

    /**
     * Static initialization block - this finds all the images in the airpollute
     * subdirectory and adds them to the file list.
     */
    static
    {
        File dir = new File(mDefaultDir);
        File [] theFiles = dir.listFiles(new ImageFileFilter(false));
        if ((theFiles == null) || (theFiles.length == 0))
        {
            new MajorIssueException("PollutionFileList (static block)\n" +
            dir.getPath() + ": No such file or directory");
        }
        mFileList = new Vector<String>(theFiles.length);
    }
}
for (int i = 0; i < theFiles.length; i++)
{
    System.out.println(theFiles[i].toString());
    try
    {
        mFileList.add(i, theFiles[i].getCanonicalPath());
    }
    catch (Exception e) {}
}

/**
 * If the air pollution image list is valid (i.e., it has at least 1 entry),
 * this selects a random, valid index and returns the file name located at
 * that index. Otherwise, a [link MajorIssueException] is called up to
 * indicate that no valid images could be found in the <b>airpollute</b>
 * subdirectory.
 * <p>
 * This function is synchronized, so it will only be called by one thread at
 * a time.
 * <p>
 * TODO: This should either link to an Internet image, prompt the user for
 * another directory, or return something white or transparent rather than
 * end the program.
 * <p>
 * @return the randomly-chosen file name
 */
public static synchronized String getFileName()
{
    if ((mFileList == null) || (mFileList.size() == 0))
    {
        new MajorIssueException("PollutionFileList.getFileName
The list " +
        "of air pollution images is null or empty!\n");
    }
    int index = mRand.nextInt(mFileList.size());
    return mFileList.get(index);
}

//TESTING BELOW THIS LINE  -----------------------------------------------

protected static JXImagePanel createImagePanel(String fileName)
{
    JXImagePanel panel = new JXImagePanel();
    panel.setEditable(false);
    panel.setAlpha(0.4f);
    panel.setStyle(JXImagePanel.Style.SCALED);
    panel.setPreferredSize(new Dimension(750, 420));
    out.println(fileName);
    Image img;
    ImageIcon imgicon;
    try
    {
        imgicon = new ImageIcon(fileName);
        img = imgicon.getImage();
        panel.setImage(img);
    }
    catch (Exception e)
    {
        String mName = "PollutionFileList.createImagePanel";
        String issue = String.format("%s\nMajor issue with loading %s\n",
        mName, fileName);
        new MajorIssueException(issue);
    }
    return panel;
}

public static void main(String [] args)
{
    String testFile = getFileName();
    System.out.println(String.format("%s", testFile));
String[] formatNames = ImageIO.getReaderFormatNames();
for (int i = 0; i < formatNames.length; i++)
{
    System.out.println(formatNames[i].toString());
}
JXImagePanel panel = createImagePanel(testFile);
JFrame frame = new JFrame("Test Interface 3");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setContentPane(panel);
frame.pack();
frame.setVisible(true);
}

edu.calpoly.ee.ehfem.linegraph

edu.calpoly.ee.ehfem.linegraph.PowerHistoryLineGraph

/*
 * PowerHistoryLineGraph.java
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 * This file is part of Exercise Power Grid Interface.
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */

package edu.calpoly.ee.ehfem.linegraph;
import edu.calpoly.ee.ehfem.animation.AnimatedTimeSeriesCollection;
import edu.calpoly.ee.ehfem.*;
import edu.calpoly.ee.ehfem.data.Catmull;
import org.jfree.chart.*;
import org.jfree.chart.axis.*;
import org.jfree.chart.plot.*;
import org.jfree.chart.renderer.xy.*;
import org.jfree.data.time.*;
import org.jfree.data.*;
import java.text.*;
import java.awt.*;
import java.util.*;
import javax.swing.*;
import org.jfree.chart.axis.TickUnits;
import org.jfree.chart.title.TextTitle;
import static java.lang.System.out;
import static java.lang.String.format;

public class PowerHistoryLineGraph extends JPanel
{
    private static final long serialVersionUID = -7844313338816328484L;
    private String mTitle = "Power Generation for ";
    private Font mTitleFont = new Font("Arial", Font.BOLD, 16);
    private static Date mYesterday = yesterday();
public static Date yesterday()
{
    Date today = new Date();
    today.setTime(today.getTime() - (1000 * 3600 * 24));
    return today;
}

public AnimatedTimeSeriesCollection getDataset()
{
    TimeSeries series = new TimeSeries("Time Data");
    Day today = new Day();
    Day yesterday = (Day) today.previous();
    series.addData(new Minute(0, new Hour(6, yesterday)), 243);
    series.addData(new Minute(0, new Hour(7, yesterday)), 742);
    series.addData(new Minute(0, new Hour(8, yesterday)), 1676);
    series.addData(new Minute(0, new Hour(8, yesterday)), 2325);
    series.addData(new Minute(0, new Hour(9, yesterday)), 2228);
    series.addData(new Minute(0, new Hour(11, yesterday)), 2416);
    series.addData(new Minute(0, new Hour(13, yesterday)), 2744);
    series.addData(new Minute(0, new Hour(15, yesterday)), 2785);
    series.addData(new Minute(0, new Hour(17, yesterday)), 2815);
    series.addData(new Minute(0, new Hour(19, yesterday)), 3347);
    series.addData(new Minute(0, new Hour(20, yesterday)), 2233);
    series.addData(new Minute(0, new Hour(21, yesterday)), 1863);
    series.addData(new Minute(0, new Hour(22, yesterday)), 1536);
    series.addData(new Minute(0, new Hour(23, yesterday)), 683);
    series.addData(new Minute(0, new Hour(0, today)), 107);
    TimeSeries interpSeries = series;
    try
    {
        interpSeries = Catmull.interpTimeSeries(series, 10);
    }
    catch (Exception e) {}
    AnimatedTimeSeriesCollection coll = new AnimatedTimeSeriesCollection(
        interpSeries);
    coll.setXPosition(TimePeriodAnchor.MIDDLE);
    return coll;
}

public double computeKilowattHours(AnimatedTimeSeriesCollection dayOfPower)
throws MajorIssueException
{
    int totalMeasures = dayOfPower.getParentItemCount();
    out.println(format("Total measurements: %d", totalMeasures));
    out.println("Data points:");
    double[] xValues = new double[totalMeasures];
    double[] yValues = new double[totalMeasures];
    double millisToHours = 1 / 3.6e6;
    for (int i = 0; i < totalMeasures; i++)
    {
        xValues[i] = dayOfPower.getXValue(0, i) * millisToHours;
        yValues[i] = dayOfPower.getYValue(0, i);
        out.println(format("Point %d: (%.3f, %.3f)", i, xValues[i],
            yValues[i]));
    }

    /** The power in watts over the day. */
    double catmullPower = Catmull.integrateList(xValues, yValues);

    /** Multiply by 1/1000 to get kilowatt-hours. */
    catmullPower *= 1.0e-3;
    out.println(format("Total power computed: %.3f kWh", catmullPower));
    return catmullPower;
}

public PowerHistoryLineGraph(Date day)
{
    Calendar cal = Calendar.getInstance();
    cal.setTime(day);
    int year = cal.get(Calendar.YEAR);
public class PowerHistoryLineGraph {
    ApplicationFrame frame = new ApplicationFrame("Power History");
    frame.setContentPane(graph);
    frame.pack();

    int month = cal.get(Calendar.MONTH) + 1;
    int dayMth = cal.get(Calendar.DATE);
    Date thatMorning = DateUtilities.createDate(year, month, dayMth, 6, 0);
    Date thatNight = DateUtilities.createDate(year, month, dayMth + 1);
    DateRange dateRange = new DateRange(thatMorning, thatNight);

    TickUnits stdTimes = new TickUnits();
    stdTimes.add(new DateTickUnit(DateTickUnitType.MINUTE, 3));
    stdTimes.add(new DateTickUnit(DateTickUnitType.MINUTE, 5));
    stdTimes.add(new DateTickUnit(DateTickUnitType.MINUTE, 15));
    stdTimes.add(new DateTickUnit(DateTickUnitType.MINUTE, 30));
    stdTimes.add(new DateTickUnit(DateTickUnitType.MINUTE, 60));
    stdTimes.add(new DateTickUnit(DateTickUnitType.HOUR, 2));
    stdTimes.add(new DateTickUnit(DateTickUnitType.HOUR, 3));
    stdTimes.add(new DateTickUnit(DateTickUnitType.HOUR, 4));
    stdTimes.add(new DateTickUnit(DateTickUnitType.HOUR, 6));

    DateAxis timeAxis = new DateAxis("Time");
    timeAxis.setDateFormatOverride(new SimpleDateFormat("H:mm");
    timeAxis.setAutoRange(false);
    timeAxis.setLowerMargin(0.025);
    timeAxis.setUpperMargin(0.025);
    timeAxis.setRangeWithMargins(dateRange);
    timeAxis.setStandardTickUnits(stdTimes);
    //timeAxis.setTickUnit(new DateTickUnit(DateTickUnitType.MINUTE, 15));
    //timeAxis.
    AnimatedTimeSeriesCollection dataset = getDataset();
    double kilowattHours = 0.0;
    try {
        kilowattHours = computeKilowattHours(dataset);
    } catch (Exception e) {
        System.err.println("Exception caught!");
    }

    NumberAxis yPowerAxis = new NumberAxis("Power (W)");
    yPowerAxis.setAutoRange(false);
    yPowerAxis.setRange(dataset.getRangeBounds(true));
    //XYLineAndShapeRenderer renderer = new XYSplineRenderer(10);
    XYLineAndShapeRenderer renderer = new XYSplineRenderer(true, false);
    XYPlot plot = new XYPlot(dataset, timeAxis, yPowerAxis, renderer);
    DateFormat dateFmt = DateFormat.getDateInstance(DateFormat.MEDIUM);
    String datedTitle = mTitle + dateFmt.format(thatMorning);
    JFreeChart chart = new JFreeChart(datedTitle, mTitleFont, plot, false);
    TextTitle powerGenTitle = new TextTitle("Total power generated" + 
        ": %.2f kWh", kilowattHours);
    chart.addSubtitle(powerGenTitle);
    //DateAxis timeAxis = (DateAxis) plot.getDomainAxis();
    //timeAxis.setDateFormatOverride(new SimpleDateFormat("H:mm");
    //plot.getDomainAxis().setLowerMargin(1.0E-6);
    //plot.getDomainAxis().setUpperMargin(1.0E-6);
    ChartPanel panel = new ChartPanel(chart);
    panel.setPreferredSize(new Dimension(750, 420));
    add(panel);
}

public static void main(String[] args) {
    System.exit(0);
  }
}
RefineryUtilities.centerFrameOnScreen(frame);
frame.setVisible(true);
}
}

edu.calpoly.ee.ehfem.sql

edu.calpoly.ee.ehfem.sql.MachineGroup

/**
 * MachineGroup.java
 *
 * Copyright (c) 2009 Alex Chernetz <achernet-AT-calpoly.edu>. All rights reserved.
 *
 * This file is part of Exercise Power Grid Interface.
 *
 * Exercise Power Grid Interface is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Exercise Power Grid Interface is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Exercise Power Grid Interface. If not, see <http://www.gnu.org/licenses/>.
 */

package edu.calpoly.ee.ehfem.sql;

/**
 * @author Alex Chernetz <achernet-AT-calpoly.edu>
 */

public class MachineGroup implements Queryable
{
/**
 * A generic serial number, which can be an Enphase serial number, MAC ID,
 * or some other type of serial number that has not yet been conceived.
 */
private String mSerialNumber;

private int mMACHINECOUNT; //the number of machines

private String mGROUPNAME; //the name of this machine group

/**
 * Get the query used to search the database, which is the serial number.
 */
public String getQuery()
{
    return getSerialNumber();
}

/**
 * Get the serial number for this group from the Enphase inverter.
 */
public String getSerialNumber()
{
    return this.mSerialNumber;
}
public int getMachineCount()
{
    return this.mMachineCount;
}

/**
 * Set the number of machines represented by this group.
 * @param machineCount the new number of machines
 */
public void setMachineCount(final int machineCount)
{
    this.mMachineCount = machineCount;
}

/**
 * Get the group name.
 * @return the group name
 */
public String getGroupName()
{
    return this.mGroupName;
}

/**
 * Set the group name to another name.
 * @param groupName the new group name
 */
public void setGroupName(final String groupName)
{
    this.mGroupName = groupName;
}

/**
 * Create a new machine group with the given serial number, machine count,
 * and group name.
 * @param serialNumber the serial number
 * @param machineCount the machine count
 * @param groupName the name of the group to display in graphs
 */
public MachineGroup(String serialNumber, int machineCount, String groupName)
{
    this.mSerialNumber = serialNumber;
    this.mMachineCount = machineCount;
    this.mGroupName = groupName;
}

/**
 * Create a new machine group with the given serial number and machine
 * count.
 * @param serialNumber the serial number
 * @param machineCount the machine count
 */
public MachineGroup(String serialNumber, int machineCount)
{
    this(serialNumber, machineCount, "default");
}

/**
 * Create a new machine group with the given serial number and group name
 * containing 1 machine.
public MachineGroup(String serialNumber, String groupName)
{
    this(serialNumber, 1, groupName);
}

/**
 * Create a new machine group with one machine and the given serial number.
 *
 * @param serialNumber the serial number
 *
 */
public MachineGroup(String serialNumber)
{
    this(serialNumber, 1);
}

package edu.calpoly.ee.ehfem.sql;

/**
 * A Queryable object has a key that can be used to search through the MySQL
database on <a href=http://decade.ee.calpoly.edu>Decade</a>. Note that the
* program must be running on the Decade server in order for any query to
* succeed; this is a consequence of the fact that Decade does not allow
* external connections (it works using a named pipe for security reasons).
 *
 */
public interface Queryable
{
    String getQuery();
}