SWENG 421 - Project 1 Phase 1
Zachery Smith and Taren Lewis
zjs115 and tbl5256

Motivation for the development of this system:

The proposed system is based on a design that would allow easy access and visualization of empirical data such as stock prices, system measurements, or other experimental data. Data visualization is an important aspect in identifying mathematical relations, and communicating meaning from otherwise ambiguous data. The proposed system would implement a graphical user interface to allow a user to import different data sets, display the data, and customize the different settings for the displayed graph. These mentioned attributes could include different axes, data labels, titles, axis tick marks, gridlines, a legend, different data sets, etc. This will provide the user the freedom to create a rich and customizable visual representation of the data.

Current Problems:

Data visualization is important for both understanding underlying relations and communicating data, so the proposed system would present a seamless experience in doing both. Currently software exists to plot and display data, but it is not fully customizable to display the data in the visual format required by the user.

What problems will be resolved?:

Most data display programs do not allow full customization to serve the needs of the customer. They provide a base program which allows specific function overlays and calculations, but do not allow the user to customize the display itself. By providing a fully customizable data display, we are allowing the user full access to the options that they specifically want. This can be especially useful for the purposes of group presentations, or sharing the visual meaning of the data without cluttering it with something extraneous like axis tick marks if it is not necessary for that specific example.

Functional & Non-functional Features:

Functional: The implemented system would allow users to open graphs, edit graphs, and export graphs. The system will also be designed in a way to allow for future implementations of attributes and features to be added. It will implement basic x-y graphing capability and the ability to add or remove graph elements including different data sets, axes, titles, and a legend. Some of these elements will also be composite and allow the user to modify attributes of them, for example axes will be able to have major tick marks, minor tick marks, titles and gridlines. Data sets will be able to be displayed as a scatter plot, a line graph, or a connected scatter plot.

One additional feature to be implemented would be to allow a tracer of the data, meaning when the user hovers over the graph a vertical line will follow the mouse movement on the graph and display the current value of the data. If the data is represented as a continuous line graph, values will be linearly interpolated between data points, and if displayed as a scatter plot the tracer will snap to discrete data points. The equation used to interpolate a value between distinct points is shown below.

$$y = [(x - x_1) * (y_2 - y_1) / (x_2 - x_1)] + y_1$$

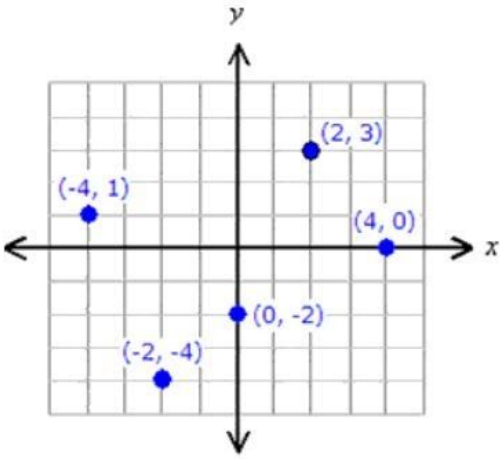
Non-Functional: The system must be able to properly deep copy objects, as well as disallow null or dangerous operations which could cause system instability such as the deletion of objects which do not exist, or the addition of objects when one of these objects already exists. The system will not allow duplication of certain objects, such as gridlines, minor tick marks, axis titles, major tick marks, legend, and graph title. The system must appropriately notify the user if an illegal operation is made, and prevent this operation instead of crashing. When the graph is displaying the data tracer, it must be displayed in real-time.

Justification of pattern choices:

From the creational patterns the system will utilize the **Prototype** pattern to allow the graphs and graph components to be deep copied because their composite nature of construction would be expensive. Prototype was also chosen to be able to copy different occurrences of the graph, for example a copy of the graph will be saved before any user input to allow the system to return to the original blank graph status on clear operation.

From the partitioning patterns the **Composite** pattern will be used to handle classes within the design, with the parent container class being Graph. Graph will be composed of multiple other classes including Data, Axis, Legend, and Title. In addition to that, the Data and Axis classes will be composite objects as well that are composed of Classes such as Point and Line for the Data class. Axis class will be composed of GrindLines, TickMarks, and Title.

From the structural patterns the **Decorator** pattern will be utilized to further emphasize the ability for growth and customization. Wrapper abstract classes can be implemented as subclasses in the composite pattern to allow different functionality to be added. The system will implement a wrapper for the Point classes, to allow implementation for a PointWithCoordiantes class that will call the Point class's draw method and then display the coordinates of the point graph, thus wrapping the Point class. This is an example of Wrapper because all Points have coordinates, but this wrapper class is a class which actively visually displays the coordinates alongside the Point itself, eg:



From the behavioral patterns the **Observer pattern** will be utilized where the GraphManager will be an observable object and all the data point objects will be considered observers. When the mouse hovers over the graphing area, the observable will notify the observers to see if the current x-value position of the mouse corresponds to the data point's x-value. If they match, then the graph will draw a red vertical line along the y axis of the data point's current x-value.

From the Concurrency patterns the **Future** design pattern will be utilized to render the graphic image displayed to the user prior to the user requesting it to display. The GUI will provide options to add, remove, and edit components of the graph and then allow the user to display the graph through a button click. The image graphic of the graph will be asynchronously created prior to the button click, as each of the components is selected within the dropdown menus. Doing so would alleviate the waiting for the graph's drawing computations to be completed on the button click, which will greatly increase the performance of the system and allow a smooth user experience.

Choice of programming language and reason:

Both of us are comfortable using C#, and working with Windows Forms which would provide a quick and easy way to deploy the GUI. Using a language that has a focus on OOP also would allow developing classes and class relationships to be much smoother than a more procedural focused language. We have also had some experience using C# to query APIs if needed, as well as working with graphics in C#.