**iii) System Design Diagrams**

1. Use Case Diagram
2. This diagram conveys which parts of the program a user interacts with. The user is able to load a file and view graphs and data, but unable to interact with how the program prepares this information for the user. The parts of the program the user is not able to interact with is represented in a very abstract and high-level format.
3. The rationale for the design of this diagram is based heavily on the requirements for the program and our design and implementation of the program. The customer’s requirements list that the user be able to load a selected file into the program, view data (in +/- expandable list), and view graphs. Each of these direct interactions by the user were visually extended with abstract processes of the program that allowed each interaction to happen.
4. Class Diagram
5. This diagram conveys the design of all classes in the program and how these classes are related to one another. From a high level, we see in the diagram that there is a Main class that is used to launch the MainWindow. From the MainWindow, two different actions are taken. The first action is the work done by the program to parse the “.csv” file, create objects from the rows in this file and subsequently add these objects to a map. The second action is the post processing completed on this map to form objects for the +/- expandable list and graphs.
6. The design for this program was based on the Builder Design developed by The Gang of Four. This allowed use to break the tasks of file parsing and user interface into levels. On the file parsing side, the Director class “pulls the strings” of building row objects from a high level. The Director will parse the “.csv” file and pass each parsed row to a RowBuilder class, which is at a level below it. The RowBuilder is responsible for building the row object by calling the AttributeReceiver which performs one level lower operations by retrieving a specific attribute from a given row. For example, the AttributeRetreiver will remove a name, date, string or integer from a “.csv” row and return it to the RowBuilder so that it can be assigned to a row object member variable. From here, the row object is passed back to the Director and inserted into a map.

The creation of this map allowed for simpler processing down the line to form objects which could be passed to a generic graphing tool, or generic +/- expandable list tool. The benefit of this strategy was regardless of whichever of the 4 file types being used, one GUI class (either for graphs or the +/- expandable lists) could display all of them without having specific implementation for each file type. At the implementation level, this was done by the ListBuilder which creates a ListClass object from a row object map and the GraphClass which creates a List<List<BarValue > > \*. These two objects are then passed to the PlusMinusList and graph dialog classes respectively for display in the MainWindow.

1. Sequence Diagram for a Scenario
2. *what* the diagram conveys
3. the *rationale* for the design (e.g., the choices made for the classes; modularity; the correspondence between the use case and customer’s requirements; how the functions interconnect to satisfy the scenario; criteria used to create packages; etc.).
4. Package Diagram
5. *what* the diagram conveys
6. the *rationale* for the design (e.g., the choices made for the classes; modularity; the correspondence between the use case and customer’s requirements; how the functions interconnect to satisfy the scenario; criteria used to create packages; etc.).