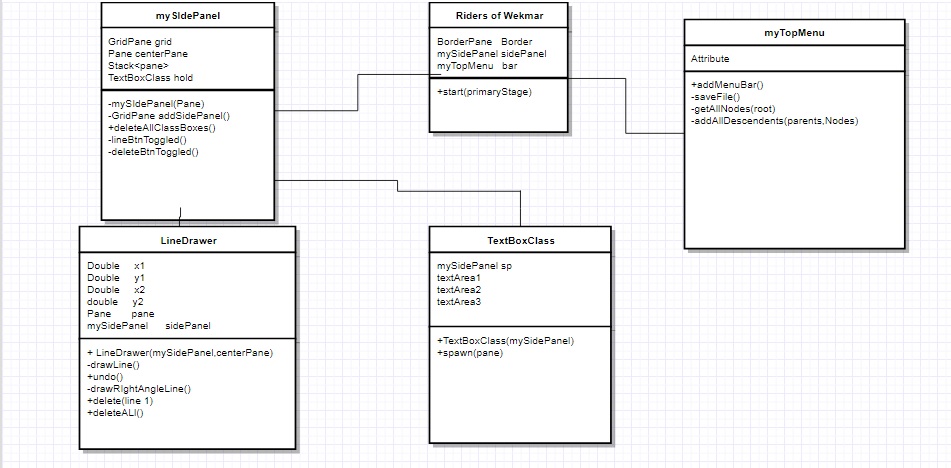
**Riders of Wekmar Graphical Editor**

**Iteration 2 Design**

***Peter Dawson, Ashely Camacho, Sam Aungst, Kailash Sayal, Kyle Martin***

**Additional files: application.css –style and control objects on the menu**

**Testing Files – exports TextFile that displays coordinates off all lines and TextBoxes**

**1.1 Purpose**

The purpose of iteration two was to build off of the objects we built for iteration 1. We tried to improve all the functionalities to all the existing objects like the textboxes, lines and the grid. The group analyzed the testing from the first iteration and that steered us in the direction where we wanted to go. We did not really add any completely new features, instead we tried to enhance what we already had before moving forward to iteration 3.

**1.2 System Overview**

Still only exclusively using Java Fx , we manage to build a simple UML graphical editor. The editor is only in its second iteration. Still a few more features we want to add and a couple of problems that should still be addressed. After using our second iteration program you should be able to create simple UML diagrams that show basic associations . The textboxes are resizable, and moveable. There are delete and undo buttons. There is still some problems with the line drawing function. The line will not snap to a textbox so if you move the textbox you will have to re draw the line. We are currently working on that fix.

**1.2.2 Assumptions and Constraints**

Using a team of 5 members the system will be implemented over 3 iterations from August to December**.**

**Overall Design:**

**Iteration 1 was a success. We built the entire layout to our program, which was 3 different sections, the TopPanel, SidePanel, and the actual grid. We also had some functionality like drawing lines and creating moveable textboxes. As Previous stated from the last design iteration, we thought it would be a good idea to put each major object in its own class. That makes it easier for updating and reading. Iteration 2 we decided to build off what we made of iteration 1 with some additional features. We analyzed the testing results from the first iteration and tried to fix a couple of those errors. A couple of us decided to fix the errors, while the rest simply looked at adding new functionalities and improving what we already had working.**

**Below are the specifics of how we implemented certain parts to our code**

**Features**

|  |  |
| --- | --- |
| **Components** | **How we added them** |
| **Adding a grid that has Snap to grid Functionality (iteration 1)** |  |
| **Adding textboxes to the main grid and moving them around**  **(iteration 1)** | **Using the textArea libraries in java, we created 3 textArea objects . Using the text area method setLayoutY and setLayoutX I was able to position all the text areas so they would stack on top of each other.**  **To move the textboxes, we first added a black box on top of the textbox that is supposed to be the area the user clicks to drag around a text box.**  **Using Mouse Event.Mouse\_Dragged, we are able to drag around the a pane. Using setTranslateX and setTranslate Y we are able to move an object around to different locations. Placing the object in a new location relys on our snap to grid functionality.** |
| **Adding the ability to drawlines on the grid**  **(iteration 1)** | **lineDrawer adds event handlers to deal with the click and release of the left mouse button on the center pane (centerPane). The mouse click handlers store the coordinates of the mouse as they are triggered and “snaps” the coordinates to a 25x25 px grid. The handlers are nearly identical, except the mouse release handler uses the drawLine() method to tell the lineDrawer to create the line object. Upon creation of the line object, the x and y coordinates are checked for being above 0, if not they are set to 0, to ensure the line remains restricted to the center panel.**  **More on “snapping”:**  **On mouse click, the x and y coordinates of the click are locally stored in the handler as int variables (named x and y).**  **In order to snap the coordinates to a grid, the remainders of x and y divided by 25 are evaluated to see if it they are closer to 0 or 25. For example, if x = 462, then the remainder of 462 divided by 25 (462 % 25 = 12) is 12. Because 12 is less than half of 25, therefore closer to 0, that remainder is subtracted from x (462 - 12 = 450). If the remainder were 13, it would be greater than half of 25, therefore the remainder is still subtracted from x, but only after 25 is added on top of it (463 + 25 - 13 = 475). In this way the coordinates are “snapped” to a 25 x 25 grid.** |
| **Undo Button**  **(iteraton 2)** |  |
| **Delete Line & Textbox Button**  **(iteration 2)** |  |
| **Resizable Textbox**  **(iteration 2)** | **There are 3 buttons that are added to the top of the textbox. A grow Button,shrink buttons, and default button. When the grow button is clicked each text area will increase by 5px by height and width. The opposite will occur when clicked the shrink button. The default button will set the text box back to its normal size. There is an if statement on the shrink and grow buttons so control that it doesn’t get too small or to big** |
| **Save Function** |  |