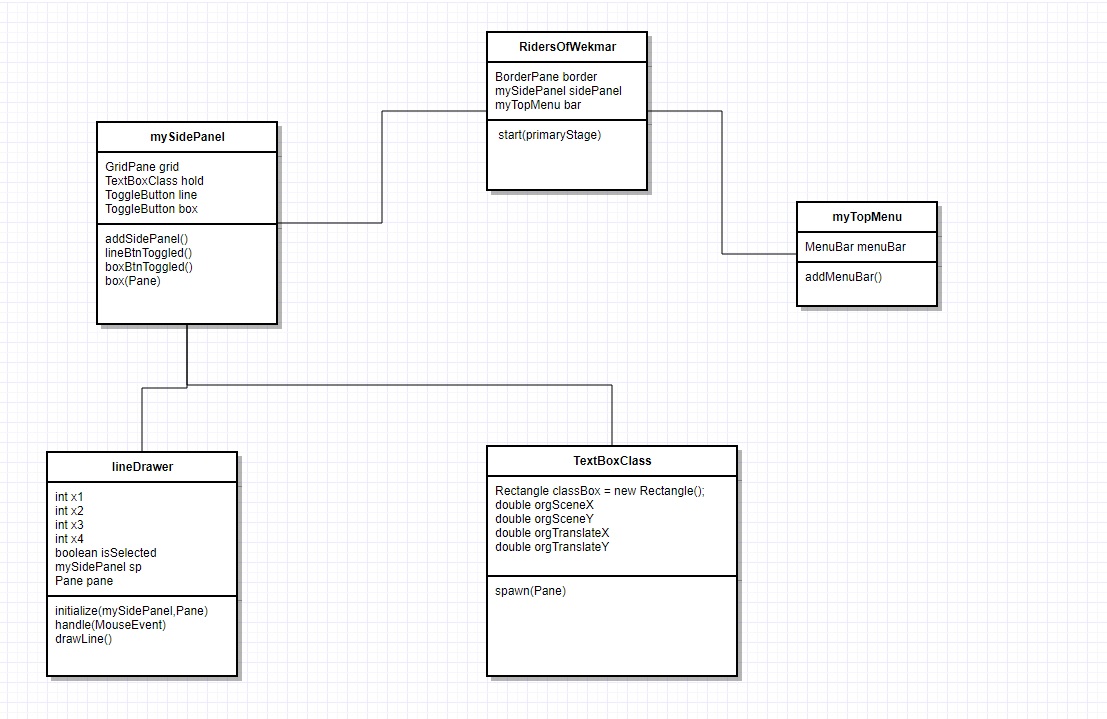
**Riders of Wekmar Graphical Editor Design Implementation**

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

******

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

**1.1 Purpose**

This system version's purpose is to allow creation of TextBoxClass objects onto a GridPane to be used abstractly as a UML diagram creation system.

**1.2 System Overview**

The system is using Java language with focus on object oriented programming.

**1.2.1 System Description**

The system is designed to be used as a graphical interface with JavaFx libraries as support.

**1.2.2 Assumptions and Constraints**

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

**Overall Design: The project we are trying to build is a UML diagram builder. The UML diagram will have many different pieces that will rely on each other to function correctly. When our software is successfully executed, you should see a window with three main components. A blank section in the middle, a side bar menu, and then top menu. Considering factors like reusability, and maintainability, the group concluded it would be best to have these feature in their own class. Currently, there are 2 operations that functions in our software, creating and moving textboxes, and drawing lines. For the same reasons, we again concluded to have those functionalities happen in a separate classes. The way those operations will occur is very straight forward. When the buttons on side Panel are clicked, the intended method will be called; an object will be created, and finally added to the grid.**

**Features we want:**

|  |  |
| --- | --- |
| **Components we want** | **How we are added them** |
| **Adding a grid that has Snap to grid Functionality** |  |
| **Adding textboxes to the main grid and moving them around** | **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** | **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.** |