**Design Overview**

First, I gathered data from the client on what needs they have and what limitations they place on the software. Then I decided on how to efficiently store and process data in order to meet the client's needs. Then I re-met with the client to see if my proposed solution was acceptable to them. Then I used the gathered feedback to further refine my design.

**User-Defined Data Structures**

* Data Type: Coordinate
  + Description: A coordinate pair for storing intercept values
  + Fields:
    - double xCoord: holds the x value of the point
    - double yCoord: holds the y value of the point
  + Methods:
    - Getters and Setters
* Data Type: Line
  + Description: A linear line to be used to store the value of the lines in the graphs
  + Fields:
    - double yIntercept: To store the line’s height when graphing it.
    - double slope: To store the line’s slope to graph it
  + Methods:
    - .intercept(Line):
      * Functionality: calculate the intercept point of the two lines
      * Preconditions: Two lines
      * Postconditions: A returned Coordinate with the intercept of the two lines
      * Pseudocode: Coordinate c = new coordinate; c.setX((intercept1 – intercept2)/(slope2 – slope1)); c.setY((intercept1 \* slope 2 – intercept2 \* slope1)/(slope2 – slope1)); return c;
    - Getters and Setters
* Interface: Graph
  + Description: An interface that has the required methods for all of the graphs
  + Fields:
    - None
  + Methods:
    - .takeInput() :
      * Functionality: to allow the graph to take input and calculate new line values
      * Preconditions: A graph with old data and a user ready to type in data
      * Postconditions: The data will be sorted to the right location and the graph update
      * Pseudocode: graph.getline().setslope(input);
    - .display() :
      * Functionality: to allow the graph to be graphed
      * Preconditions: The graph has all necessary values
      * Postconditions: The graph is showed to the GUI
      * Pseudocode: window1.set(graph);
    - .recalculate() :
      * Functionality: change line values in response to other parameters changing
      * Preconditions: New data has been put into the important variables
      * Postconditions: The graph’s data is now correct
      * Pseudocode: graph.getLine.setSlope(slope formula); graph.getLine.setIntercept(intercept formula);
* Data Type: ASAD implements Graph
  + Description: A graph of the aggregate supply/aggregate demand model. The x axis is real GDP and the y axis is price level. It has an upward sloping short run AS, a downward sloping AD, and a vertical long run AS
  + Fields:
    - Line SRAS : the short run AS line
    - Line AD: the AD line
    - double LRAS: the x intercept of the vertical long run AS line
  + Methods:
    - Graph methods
    - Getters to allow for data to be returned and analyzed
    - Setters to allow data to be changed
* Data Type: MDGraph implements Graph
  + Description: A graph that shows how much money people wish to hold based on the interest rate. The y-axis is the nominal interest rate and the x-axis is quantity of money. It has a vertical money supply, and a money demand which consists of a vertical portion and a downward sloping portion.
  + Fields:
    - double baseMoneyNeeds: the x intercept of the vertical part of money demand
    - Line liquidityPreference: the sloping portion of the money demand
    - double moneySupply: the x intercept of the vertical money supply
  + Methods:
    - Graph methods
    - Getters to allow for data to be returned and analyzed
    - Setters to allow data to be changed
* Data Type: ConsFunc implements Graph
  + A graph that shows how much households consume at various disposable income levels. The x-axis is quantity of consumption; the y-axis is quantity of current disposable income. The slope of the function is the marginal propensity to consume (MPC) and the y intercept is the autonomous consumption.
  + Fields:
    - Line consumption: the line of the consumption function
  + Methods:
    - Graph methods
    - Getters to allow for data to be returned and analyzed
    - Setters to allow data to be changed
* Data Type: LFMarket implements Graph
  + The loanable funds market that shows the supply and demand of loanable funds. The x axis is quantity of loanable funds and the y axis is the interest rate. The graph has an upward sloping LF supply and a downward sloping LF demand.
  + Fields:
    - Line supplyLF: the loanable funds supply line
    - Line demandLF: the loanable funds demand line
  + Methods:
    - Graph methods
    - Getters to allow for data to be returned and analyzed
    - Setters to allow data to be changed
* Data Type: FEMarket implements Graph
  + The graph that shows a currency’s value in relation to another currency. The x-axis is quantity of currency exchanged and the y-axis is the price in terms of another currency of the currency. The graph consists of an upward sloping supply and a downward sloping demand.
  + Fields:
    - Line dollarDemand: the line for the demand of the dollar
    - Line dollarSupply: the line for the supply of the dollar
  + Methods:
    - Graph methods
    - Getters to allow for data to be returned and analyzed
    - Setters to allow data to be changed

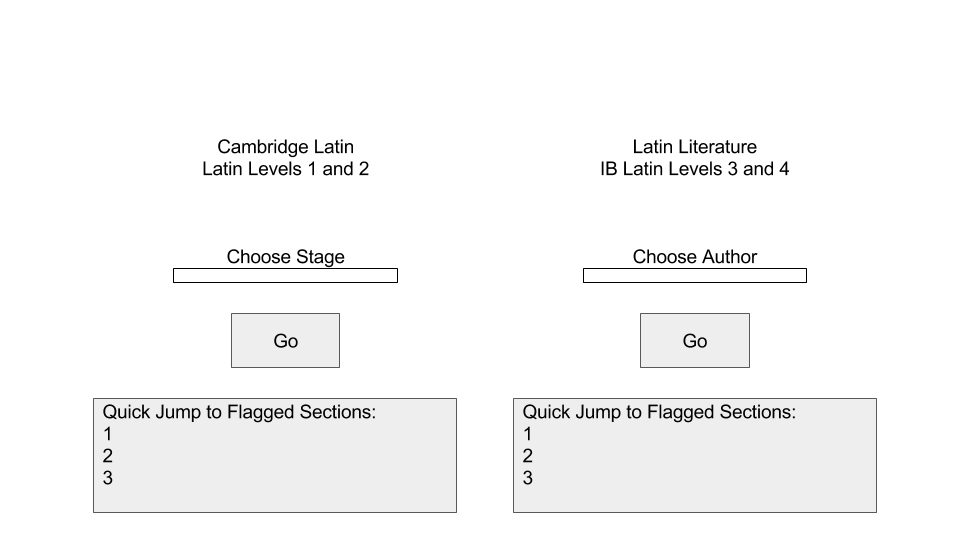
**File Structures**

* File Name: [Scenario Name]
  + Description: Used to store scenario data between sessions. Stores all the economic data so all the graphs can be regenerated.
  + Data Organization: The data is listed as a plaintext file that lists all the economic data, with a “\*” if no data was inputted for that field. When written to it will list the field name, a “: ”, and then the field value. When read for data, it would parse the string and take in the values.
* Folder Name: [Scenario Name] Graph Export
  + Description: A file that holds pictures of all the graphs that can be formed with the given data. Used to allow the user to have access to exported versions of these graphs for use elsewhere.
  + File Names: [Graph Name]
  + Data Organization: The folder would have .png images of every graph inside of it. These will be blank if the graph is missing data. These are right only; they cannot be imported for information.

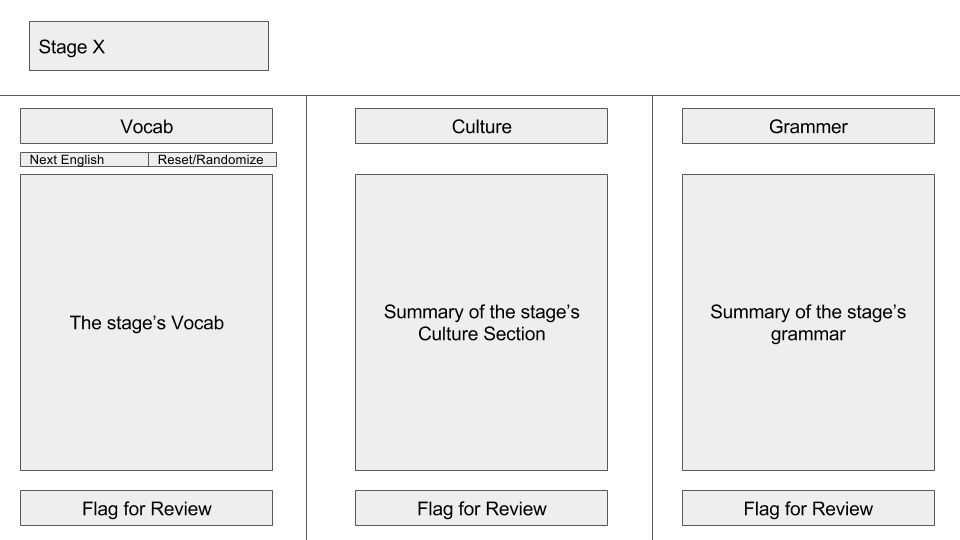
**Test Plan**

* As/AD Model
  + Test when data is normal: point of fractional business full capacity and the point of full business capacity are both in the first quadrant, and the latter is above and to the right of the former. ex: former = (100, 100), latter = (150, 150)
  + When data is abnormal:
    - 1 of the above rules is false. ex (100, 100), (50, 50)
    - the other rule is broken. ex (-100, -100), (0, 0)
    - Both rules are broken. ex (-25, 0), (-3, -3)
* Consumption Function
  + When Data is normal: Autonomous consumption is positive and the MPC is between 0 and 1 exclusive. ex 5000 and .5
  + When data is abnormal
    - Autonomous consumption is zero or negative
    - MPC is an edge case (0 or 1)
    - MPC is negative
    - MPC is greater than 1
* Money Demand Graph
  + When Data is Normal: Money supply is > 0 and Money demand has a positive y intercept, a negative slope, and intersects the money supply line in the first quadrant. ex: MS = 100, MD y intercept = 100, MD slope = -.5
  + When data is abnormal:
    - MS is < 0, ex -100
    - MD has a negative or zero y intercept, ex -100
    - MD has a positive or zero slope, ex .5
    - MD doesn’t intersect MS, ex if MS = 10000000 and MD = 100 - x
* Loanable Funds Market
  + When Data is normal: LF supply has a positive slope, LF demand has a negative slope and positive y intercept, and the two intercept in the first quadrant. ex LFS = 25 + x, LFD = 50 - x
  + When Data is abnormal:
    - LFS has a non-positive slope. ex -1
    - LFD has a non-negative slope. ex 0
    - LFD has a non-positive y intercept, ex -10
    - LFD and LFS do not intercept in the first quadrant. ex LFD = 1 - x LFS = 10 + x
* Foreign exchange market
  + When Data is normal: currency supply has a positive slope, currency demand has a negative slope and positive y intercept, and the two intercept in the first quadrant. ex CS = 25 + x, CD = 50 - x
  + When Data is abnormal:
    - CS has a non-positive slope. ex -1
    - CD has a non-negative slope. ex 0
    - CD has a non-positive y intercept, ex -10
    - CD and CS do not intercept in the first quadrant. ex CD = 1 - x CS = 10 + x

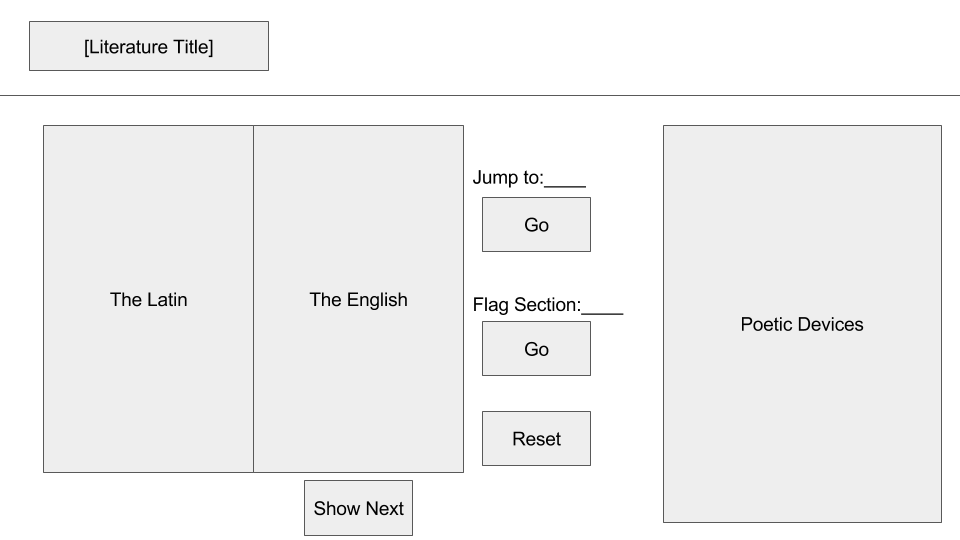
**User Interface Design**



This is the title screen of the program. It shows the options to open whatever the user needs to study. The boxes under Choose Stage and Choose Author are dropdown menus, when the user has selected their desired section they can press go to be taken to that section. The Quick Jumps allow the user to quickly access the sections that they have indicated are necessary for them to review.



This is the UI for the stages. It shows the three main sections for each stage and a summary of the information in each. The vocab secion starts with the English hidden and the next English button will incrementally revleal the English definitions; this allows for the user to quiz themslevs. The reset/randomize button hides all the English and rndomizes the



**Input**

The values the user inputs will always be doubles or ints and will be inputted either by keyboard or from file. They are used to calculate data about the graphs. The user will input two kinds of values, values that only tell the program about the slope and intercept of a line and are not saved separately from the line, or important economic data that could affect other economic graphs and are saved separately.

An example of the less important information is the slope of the AD curve. This value is not used anywhere else so it will be stored in the AD line then not stored elsewhere. When this value is changed, the line is directly .set() with the new value.

An example of data that is more important and stored elsewhere is the slope of the consumption function, because it is the MPC. Since the MPC effects changes in the money supply of the money demand graph, and elsewhere, it is stored in its own separate variable. This kind of data, when changed, does not directly .set() the line, but instead updates the dedicated variable and causes *all* graphs to recalculate because they may have been changed.

**Output**

The primary outputs for this program will be graphs. Using the line values stored in the various graphs, as well as rules for each graph, the graph can be precisely determined. These graphs will then be printed to a Java GUI.

The program also outputs files. The first type of file output is to save the scenario so that it can be returned to later. This file is a plaintext file and will have the name of the parameter, a “: “, and then the value, with a \* if there is no value.

The second type of file is a folder of .png’s that are pictures of the charts generated by the program. The chart will be blank if there is not sufficient data. This cannot be read from.