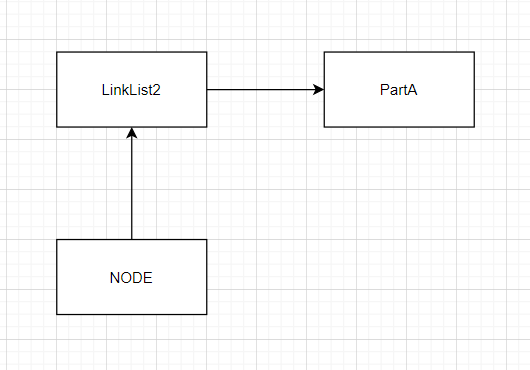
**Midterm Report**

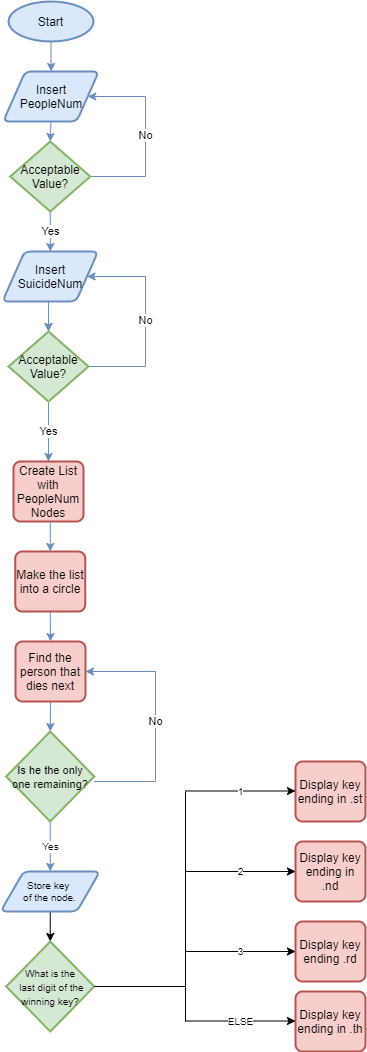
**Note: For the sake of concision the report will not include explanations of the class made code, unless changes are made. Furthermore, code that stays the same between parts will not be mentioned again.**

**Part A:**

* **LinkList2.java:**
  + **Methods:**
    - **Delete: Based on the delete method we created in class. Created a custom parameter to take into account the fact that our list is now a circle. Since there will never be a null value I created a counter (countDelete) as a switch, and changed the parameter of the while statement to stop the loop after we reach the .first node for the second time.**
    - **LinkCircle: Finds the final node of the list and makes it point to the first node, effectively making the list into a circle.**
    - **Display: Based on the Display method we created in class. Created a similar parameter as in the delete method to compensate for the circle list.**
* **PartA.java:** 
  + **Variables:**
    - **LinkList2 object: list**
    - **Integer: survivalPos**
    - **Integer: deleted**
    - **Integer: count**
  + **Methods**
    - **Selector:** 
      * **Parameters:**
        + **Int peopleNum: the amount of people participating.**
        + **Int SuicideNum: the number of steps in the suicide game.**
      * **Functionality:** 
        + **Step 1: The list is created using the insert first method, the for loop is reversed (i starts at peopleNum and decrements until it reaches 1) in order to easily create an ordered linked list, where the key of each node represent its position.**
        + **Step 2: Use LinkCircle to make the list into a circle.**
        + **Step 3: Create a temp node that is equal the the first node of the list.**
        + **Step 4: Create a for loop that repeats for every person in the list.**
        + **Step 5: Move the temp node forward in the list for the number of steps indicated by suicideNum.**
        + **Step 6: Store the key of the person about to die in survivalPos.**
        + **Step 7: If the temp node points to it self it means it is the only one left, return survivalPos since it is the key of the winning position and exit.**
        + **Step 7: Otherwise, delete the node and move temp to the next non-null position**
    - **Main:**
      * **Variables:**
        + **PartA object: test**
        + **Scanner object: userIn**
        + **Integer peopleNum**
        + **Integer suicideNum**
        + **Integer survivalPos**
      * **Functionality:**
        + **Simple UI that asks of the user to input the peopleNum and suicideNum values and traps for any erratic inputs. The values are then fed as parameters in the Selector function and the results are stored in survivalPos. Finally the result is tested and a display is outputted based on the number.**
* **Class Dependency:**

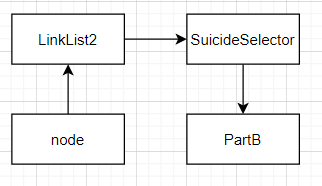


* **Flowchart:**

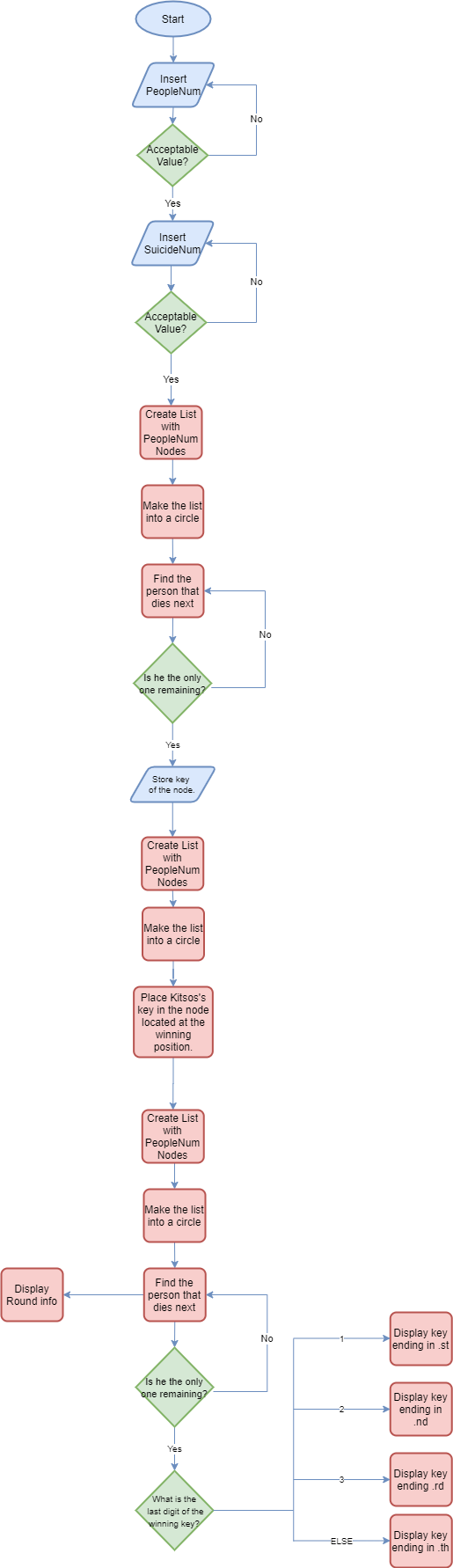
****

**Part B:**

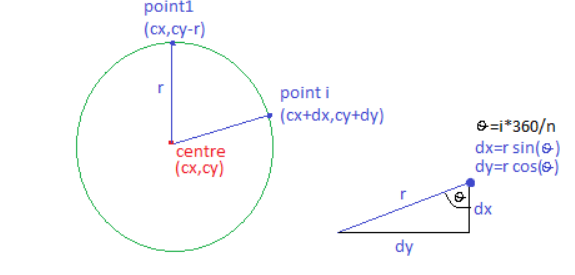
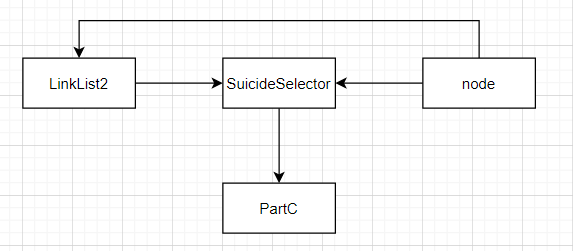
* **node.java:**
  + **Variables:**
    - **Integer: key**
    - **String array: nameArray 🡪 Contains 40 ancient Greek names.**
    - **Node: next**
  + **Methods:**
    - **Display: Instead of displaying the key, this method now displays the item of nameArray with the index of the key.**
* **SuicideSelector.java:**
  + **Variables:**
    - **LinkList2 object: list**
    - **LinkList2 object: listAnalyze**
    - **Integer survivalPos**
    - **Integer deleted**
    - **Integer count**
  + **Methods:**
    - **Selector: The same as in partA**
    - **AnalyticalDisplay:**
      * **Parameters:**
        + **Int peopleNum**
        + **Int suicideNum**
        + **Int survivalPos**
      * **Functionality:**
        + **Step 1: Create and link the list as in selector.**
        + **Step 2: Move temp until its key matches the survival position.**
        + **Step 3: Change the key of that node to 0, a key reserved for Kitsos.**
        + **Step 4: Display the full list. The display method is the same but the numbers have been substituted with the names already from the node class.**
        + **Step 5: Execute the suicide selection process the same as in selector, but this time create a display command for every “round” (every repeat of the peopleNum loop), that prints the round count and the dead person.**
        + **Step 6: Display the list without the dead person.**
        + **Step 7: Continue the process in the exact same way as in partA.   
            
          The main class (PartB) contains only a main method that is exactly the same as the main method of PartA, barring the inclusion of a suicideSelector object and the execution of the selector and analyticalDisplay methods consecutively.   
            
          The differentiation of the classes in part B (the creation of the SuicideSelector class) serves only for improved readability of the code.**
* **Class Dependency:**

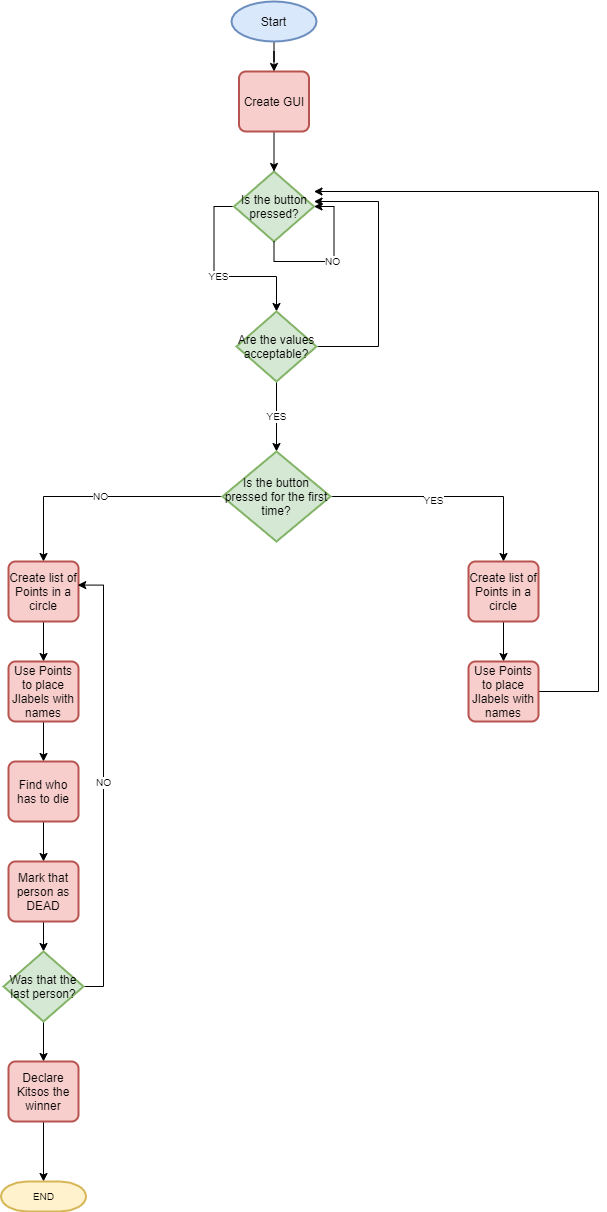


* **Flowchart:**

****

**Part C:**

* **Node.java:**
  + **Methods:**
    - **getKey:** 
      * **Parameters:**
        + **String: name**
      * **Functionality: This method accepts a string and returns the index of the item in nameArray that matches that string. If no match is found it returns -10.**
* **SuicideSelector.java:** 
  + **Variables:**
    - **JLabel: label**
  + **Methods:**
    - **StepSelector:**
      * **New parameters:**
        + **Int countButton**
      * **Functionality: The analyticalDisplay method has been modified as following.** 
        + **The people num loop has been removed and replaced by a loop based on countButton, a counter inserted by the PartC class, that increments every time the submitButton is pressed.   
          This is used to link the number of repeats of the search and delete commands, allowing us to move temp to the node that is to be deleted based on the round dictated by the gui. (User moves to the next round when he pushes the submitButton).**
        + **Once we have found the node that is to be deleted in this round, we return its key.**
    - **ListCreator:**
      * **Parameters:**
        + **Int PeopleNum**
        + **Int count**
        + **Int survivalPos**
      * **Functionality:** 
        + **Step 1: Create the linked list.**
        + **Step 2: Move temp according to the count parameter (This parameter is different than countButton. It is incremented every time a label is added to the main panel in the gui).**
        + **Step 3: Create a new JLabel object that contains the name associated to the temp.key index of nameArrays)**
        + **Step 4: Return the label to the main application.**
* **Part C:**
  + **Variables:**
    - **JTextfield object: peopleField**
    - **JTextField object: suicideField**
    - **TitledBorder object: suiNumBorder**
    - **TitledBorder object: peopleFieldBorder**
    - **Jlabel object: infoLabel**
    - **JButton object: submitButton**
    - **JPanel object: panelMain**
    - **JPanel object: infoPanel**
    - **JPanel object: inputPanel**
    - **Integer countButton**
    - **Integer peopleNum**
    - **Integer suicideNum**
    - **List exclusively of integers: excludedKeyList**
  + **Constructor:**
    - **Create and name the titled borders**
    - **Create and set up the text fields**
    - **Create and set up the infoLabel**
    - **Create and set up submitButton**
    - **Create panels**
    - **Fill and set up panels**
    - **Initialize countButton to zero**
    - **Initialize excludedKeyList**
    - **Add panels to content pane and define layout preferences**
  + **Action performed:**
    - **Trap text fields for erratic inputs (If the input is not valid a message will be shown through infopanel and the actionevent method will not continue).**
    - **Initialize peopleNum and SuicideNum through the input found in the text fields.**
    - **Make panelMain invisible.**
    - **Clear out all objects from within panelMain.**
    - **Make both text fields non-editable.**
    - **Initialize excludedKey to -1 so that it will not affect the initial creation of the labels.**
    - **Initialize count to zero.**
    - **Create a spring layout and apply it to panelMain**
    - **Create and Initialize a SuicideSelector object**
    - **The rest of the code is included in an if statement that checks if the user has inputted numbers withing the provided limits, running the rest of the program if he did, and informing him to retry if not.**
    - **Change the title of the submitButton**
    - **Create a point called center with set coordinates**
    - **Create a double called angle that contains the division of a circle by peopleNum, converted to radians since Java doesn’t understand degrees.**
    - **Create and initialize a radius integer that allows us to control how big the resulting circle will be.**
    - **Create and initialize a list made exclusively of points.**
    - **Run the selector method and store the result in survivalPos**
    - **If this is the first time the user pressed the button.**
      * **For as many times as indicated by peopleNum**
        + **This snippet was repurposed from stack overflow. The idea is that we calculate the coordinates of each point based on the coordinates of the center point. Dx is the sin of the angle and it states how much higher (or lower) our point is compared to the center. Like wise for Dy we use the cos of the angle and calculate how much on the right or the left of the center our point is.** 
        + **We repeat until we have populated our list with as many points as there are people in the circle.**
      * **For each loop. It will run once for every point in our list.**
        + **Create a label using the listCreator method.**
        + **Set up the preferences for our label.**
        + **Add the label to panelMain.**
        + **Increment count.**
    - **If this is not the first time the user pressed the button.**
      * **Create the point list as normal.**
      * **Run stepSelector and save the excluded key in the excludedKey variable.**
      * **If the excludedKey is not the survivalPos, add it to the excludedKeyList. This is to ensure we do not accidentally kill Kitsos.**
      * **For each point.**
        + **Create the label using listCreator.**
        + **Set the infoLabel text to inform you of the name of the person that got eliminated this round. (Used select.listAnalyze.first.nameArray[excludedKeyList.get(countButton - 1)] to find the name in order to avoid creating a node object)**
        + **If the index of the name contained in the label that was just created, matches any of the entries in the excludedKeyList entries, that person is declared dead.**
        + **If we repeat as many times as there are people (Minus Kitsos), the infoLabel declares that Kitsos is alive and the SubmitButton cannot be used anymore.**
      * **The countButton is incremented**
      * **PanelMain becomes visible again.**
  + **Main:**
    - **Creates and sets up the frame.**
    - **Sets a default closing operation to terminate when the user closes the window.**
    - **Makes the frame visible.**
* **Class Dependency:** 
* **Flowchart**

****