Thneed Inc

By: Alaric Manning, Tian Li, Luke Solem

Table of Contents:

Login Page Controller

- **signInButtonClick(ActionEvent e):**
    - This method will go into the login_info.txt access the information line by line, then save it in an arraylist, then save it to a hashmap to enable to check username and password in pair.
    - it will go into the order_info.txt to load the orders
    - It also read the customer_info.txt file to create a customer object arraylist to enable display in the order history page.
    - Read in each line of the order file and split it on the commas, then use indexing to assign appropriate varaibles and recreate every order object
    - Iterate through the orderlist and if the order is filled add it to the filledOrderList and its toString version to the filledInfoList and do the same for unfilled orders
    - Convert the filledInfoList and unfilledInfoList to observable lists and push them to the filled order list view and unfilled order list view on the Order History Page Controller
    - Read in all of the customer information from the customer text file and recreate every customer obect and save them to an ArrayList
    - Take the user to the Order History Page if login information was correct and use the setter methods to set the list views
- **getOrderList()**
    - Returns the ArrayList of all orders
- **getFilledOrderList()**
    - Returns the ArrayList of all filled orders
- **getUnfilledOrderList()**
    - Return the ArrayList of all unfilled orders
- **getCustList()**
    - Return the ArrayList of all customer objects

Order History Page Controller

- **setCallingController(LoginController c)**
    - This method takes a LoginController object as an input and links to page
- **viewCustomerButtonClick(ActionEvent e)**
    - Creates ArrayList of customer objects
    - Gets the selected item from the list view as a string and splits it on the comma

    - Sets customer variables by indexing the listview items after split
    - Iterates through the ArrayList of customer objects and finds where the customer ID is the same as the item selected from the list view by the user and sets the customer variables to be pushed to the customerInfoPageController.
    - Creates a new FXMLLoader of the CustomerInfoPage and uses the setter methods in that controller to set the labels with the customer info of the customer that the user had selected
- **getUnfilledListView()**
    - Returns the unfilled listview object
- **getFilledListView()**
    - Returns the filled listview object
- **placeNewOrderButtonClick(ActionEvent e)**
    - Creates a new FXMLLoader of the Order Page
    - Displays the Order Page to the user and uses the customerInitialize method from the order page to initialize the customer ID combo box on the Order Page with an ArrayList of the customers
- **fillSelectedOrderButtonClick(ActionEvent event)**
    - Creates an ArrayList of orders by using the getOrderList function from the login controller
    - Gets the selected item in the listview, converts it to a string and splits is on the commas
    - Sets an orderID variable with indexing on the array of strings creates from the list view
    - Iterates through the ArrayList of orders and if the orderID is the same orderID that the user selected we use the setStatus method to set the status to true and fill the order
    - We create ArrayLists for unfilled orders and filled orders
    - Go through all of the orders and if they're status is true then add them to the filled order list otherwise add them to the unfilled order list
    - Add the toString method of each to an info list for filled order or unfilled orders then sort both of those lists, convert them to observable lists, and set the filledOrderListView and unfilledOrderListView contents with those two observable lists
- **saveOrderButtonClick(ActionEvent event)**
    - Creates a new Order_Info.txt file then opens a new PrintWriter and FileOutputStream
    - Then go through the order list and write the toString version of each order in the order list to the file along with a new line character and then close the PrintWriter and FileOutputStream
- **getPageStage()**
    - Returns the Order Page Stage

Order Page Controller

- **setCallingController(OrderHistoryController c)**
    - This method takes the OrderHistoryController and links it to the Order Page.
- **setLoginController(LoginController lc)**
    - This method takes the LoginController and links it to the Order Page.
- **intialize()**
    - Sets the values of the size combo box.
- **customerIntialize(ArrayList <customer> cusID)**
    - Populates ComboBox with customerIDs.
- **clearButtonClicked(ActionEvent event) throws IOException**
    - Clears the TextFields and Radio Buttons on the Order Page.
    - Sets RadioButtons to False, TextFields to clear using Clear function.
- **checkoutButtonClicked(ActionEvent event)**
    - Overview (Creates an Order object from the GUI based off what was entered by a user.)
    - Initialize variables to be added to an object.
    - Get customerID and Size from a combo box, get Radio Buttons from selected toggle.
    - Create a new order object, added it to the order list from the LoginController.
    - Use BufferedWriter to write Order object via toString method to the Order_Info text file.
- **addNewCustomerButtonClicked(ActionEvent event)**
    - Uses FXMLLoader to load the Register New Customer FXML GUI.
- **addButtonClicked(ActionEvent event)**
    - Overview (Creates a Thneed object from the GUI based off what was entered by a user.)
    - Initialize variables to be added to an object.
    - Get Size from a combo box, get Radio Buttons from selected toggle, Quantity from a TextField.
    - Create a new Thneed object with variables from GUI, add it to the order list.

Register New Customer Page Controller

- **cancelButtonClick (ActionEvent event)**
    - This method will hide the current page when it is clicked.
    - Set the previous page's stage to null to enable it run again.
- **setCallingController(OrderPageController c)**

    - This method enables to create an order page controller to link to that page and access its methods.
- **registerButtonClick (ActionEvent event)**
    - This method will create a new customer, get all the input from customer and save it to an Arraylist.
    - It will add the new customer ID to the order page customer id combo box.
    - It will hide the current page when it is clicked.
    - Set the previous page's stage to null to enable it run again.
    - This method will write to the customer_info.txt file.

Customer Information Page Controller

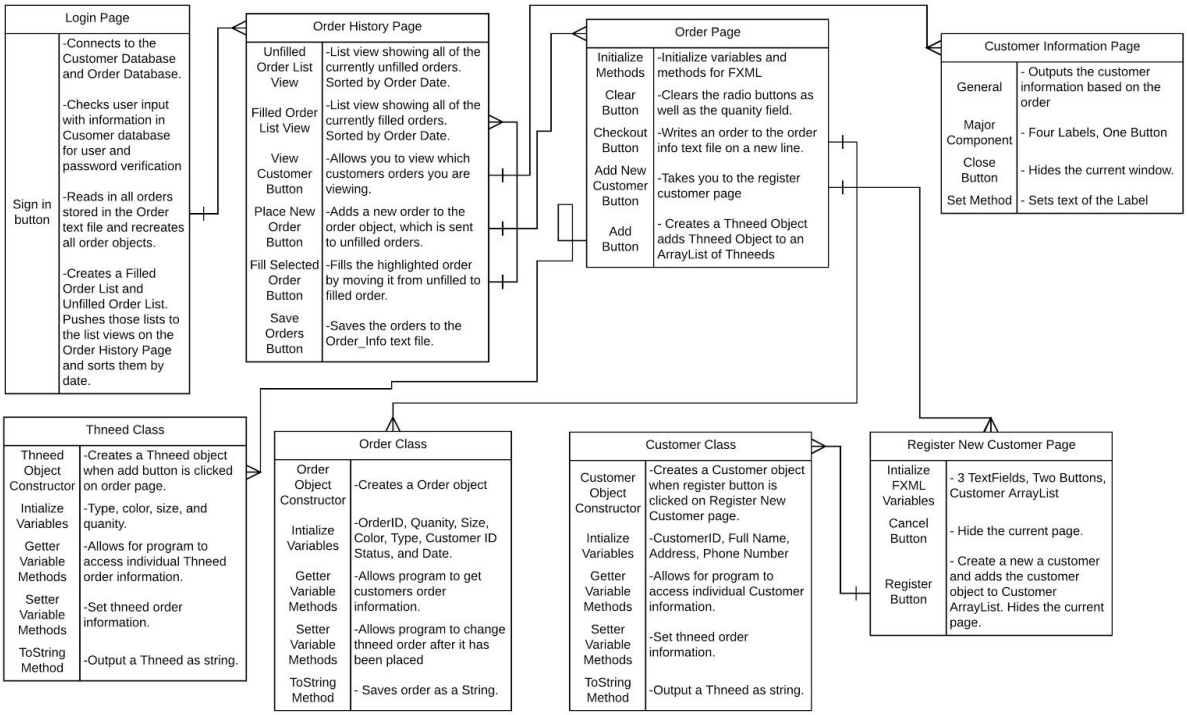
- **setController(OrderHistorycontroller c)**
    - Sets the controller to the order history page controller allowing us to interact with that controller
- **setCustNameLabel(String custName)**
    - Sets the customer name label on the GUI
- **setCustAddressLabel(String custAddress)**
    - Sets the customer address label on the GUI
- **setCustPhoneLabel(String custPhone)**
    - Sets the customer phone label on the GUI
- **setCustIDLabel(String custID)**
    - Sets the customer ID label on the GUI
- **closeButtonClick(ActionEvent event)**
    - Closes the customer information page

Order Class

- **Order(int orderID, int customerID, int qty, String size, String color, String type, Date date, boolean status)**
    - This order constructor will enable to create a order object with orderID, customerID, qty, size, color, type, date, status.
- **compareTo(Order o)**
    - This order constructor will enable to create a order object with orderID, customerID, qty, size, color, type, date, status.
- **getOrderID()**
    - Getter method to return the orderID for other class to get the orderID
- **getqty()**
    - Getter method to return the qty for other class to get the quantity
- **getSize()**

    - Getter method to return the Size for other class to get the Size.
- **getColor()**
    - Getter method to return the color for other class to get the color.
- **getCustomerID()**
    - Getter method to return the customerID for other class to get the customerID.
- **getDate()**
    - Getter method to return the date for other class to get the date.
- **getStatus()**
    - Getter method to return the status for other class to get the status.
- **setOrderID()**
    - setter method to enable to change the orderID.
- **setqty()**
    - setter method to enable to change the quantity.
- **setSize()**
    - setter method to enable to change the Size.
- **setColor()**
    - setter method to enable to change the color.
- **setCustomerID()**
    - setter method to enable to change the customerID.
- **setDate()**
    - setter method to enable to change the date.
- **setStatus()**
    - setter method to enable to change the status.

Architecture Diagram



Alaric Manning
Chris Li
Luke Solem

Daily Backlog

10/24/17 Documentation:

Built Login, Register page, & Welcome page GUIs - Chris
Built Login Page & Register Initial Controllers - Chris
Built Order History GUI, Order Class, & Order Text File - Alaric
Built Order Page GUI, Initial Controller, & linked Login Controller to Order Page - Luke

10/26/17 Documentation:

Login page completed
Remodeled GUIs & Controllers to fit the employee use case rather than customer.\

10/31/17 Documentation:
Reordered navigation of pages to open to view orders first rather than place orders.
Worked on reading in saved orders, edit GUIs and order Class.
Changed register GUI to register new customer within the order page GUI.

11/2/17 Documentation:
Edited OrderPageController to simplifiy methods of getting the thneed information using toggle groups rather than if statements.
Worked on reading in the unfilled and filled orders.

11/3/17 Documentation:
Worked on reading in customer orders based off of orders placed.
Added button to order page GUI and changed menu bar to combo box.

11/7/17 Documentation:
Orders are now able to be filled.
Save data button has been added and works.
Adding an exit window stop method which will write to file. Still a work in progress.

11/9/17 Documentation:
Filled orders now transition from filled to unfilled at the click of a button.
Fixed clear button to clear all fields.
Comments for documentation added to the code.
Created powerpoint for presentation and lucidchart for architecture diagram.