Project Report

Sara McKim, Taha Motiwala, Mustafa Asghar

1. Final Project Overview and Description

January 1st is an interactive GUI designed as an ordering and POS (Point of Sale) platform. Designed with the following: order management, payment processing and promotional discounts, January 1st is an advanced system that is both user and business friendly.

To begin, January 1st is an ordering system that uses the application’s input to output it into a separate file (alternatively known as the bill). This innovative approach ensures seamless communication between the application’s classes, allowing the bill to be acknowledged and maintained accordingly. Specifically, and in application to the system, when the bill class identifies the user’s order from the order.txt, it calculates the total accumulated cost and any eligible discounts/promotions. Such functionality is not only vital to maintaining proper bookkeeping, but to also enhance the user’s experience.

Furthermore, dynamic arrays were used for the output from the one shared text file into the bill class.

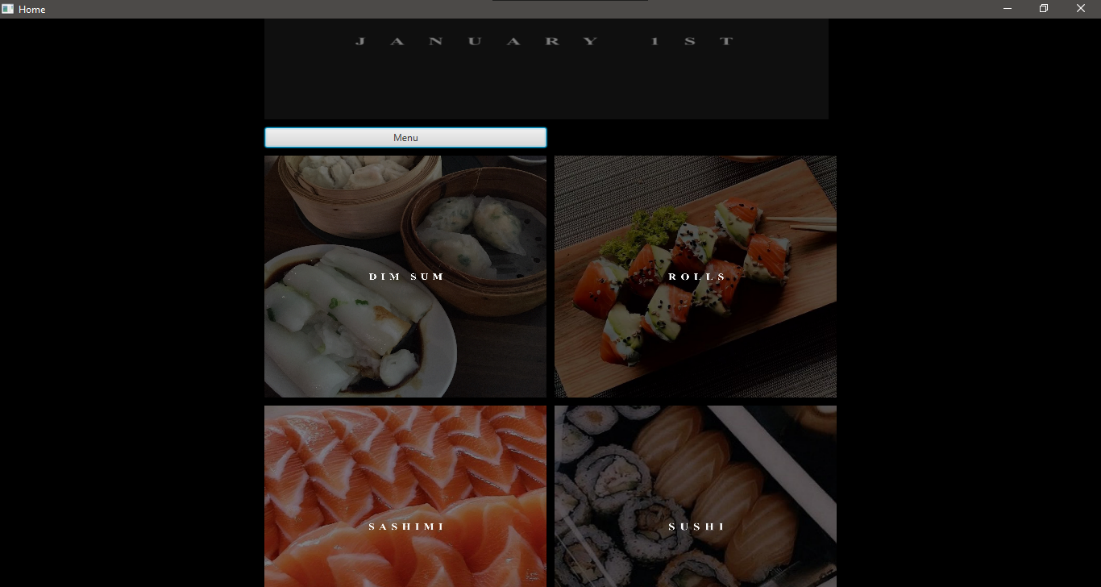
Finally, January 1st boasts a modern and aesthetic interface that is professional and simple. The visually appealing design not only enhances the customer experience, but elevates the overall representation of the business. Through its incorporation of contemporary design principles and user-centric elements, January 1st ensures ease in efficiently interacting and navigating with the application.

In conclusion, January 1st is an effective system with various capabilities including, but not limited to, order management, proper bookkeeping and an application that is both accessible and intuitive.

1. GUI Photos

CLASS HOME EXPLANATION:

This Java code describes the "Home" JavaFX application. When the application is run, the "January 1st" restaurant's home page is shown. A picture of the restaurant, a menu button, and some informative images of the food the restaurant serves are all featured on the homepage.   
The next line of code creates a button object called "menuButton" that, when clicked, will display the menu. The button's preferred width and height are then specified by the code, and it also gives the button an event listener. The menu button opens the "MenuApplication" JavaFX program, which shows the restaurant's menu when clicked.   
The menu button and all the educational images are then added to the grid pane, which has been given the object name "pane" in the code. The pictures are included in   
the row and column coordinates of each image into the grid pane using the GridPane class's add() method.   
The GridPane object is then placed within a Group object called "image" that is made. To centre the home page, a StackPane object named "stackpane" is created. The setBackground() function of the StackPane class is used to give the StackPane object a black background.   
The Stage object is then added to with a Scene object that was made with the StackPane object. When the application is launched, the Stage object is configured to display the home page. The JavaFX application is launched by calling the launch() method from the class's main() method.



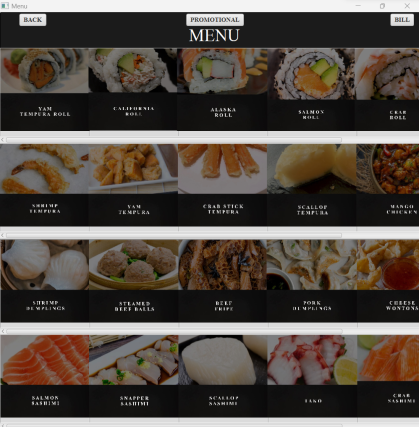
CLASS MENU EXPLANATION:

The Back button facilitates a function that, upon activation, returns the user to the home class through the use of the backButton method.

Similarly, the Promotional button serves the purpose of directing the user to the promotional class through the activation of the PromotionalButton method. Upon arrival, the user is greeted with various promotions that the restaurant offers.

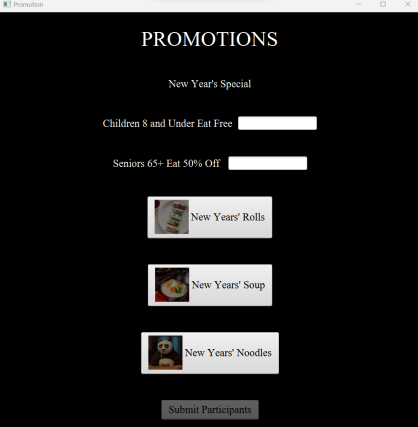
The Bill button also has a specific function. When selected, it takes the user to the Bill class by utilizing the billButton method. Here, the user can view the items they have chosen.

The remaining buttons are identified by a numeric name and possess the capability of adding data to a file named Order.txt. This data is subsequently employed by the Bill class to identify each item and allocate the appropriate price. For instance, the eighthbutton method represents the eighth item in the menu, Yam Tempura, and is responsible for appending the corresponding name to the Order.txt file. Once recognized by the Bill class, the item is allocated the specific price and added to the table, alongside any other items the user has chosen.



CLASS PROMOTION EXPLANATION:   
TextField: The createNumericTextField() method creates a new TextField instance that only accepts numeric input. A TextFormatter is used to set a filter that allows only digits (0-9) to be entered in the TextField. If the input entered by the user is not an integer, an alert message is displayed.

Button: The submitButton is used to submit the number of participants for a promotion. It is initially disabled until at least one of the TextFields (textField1 and textField2) has a value entered in it. The setDisable() method is used to enable or disable the button based on the content of the TextFields. The textProperty() method is used to listen for changes in the TextField and call the setDisable() method accordingly.

When the submitButton is clicked, the handleParticipantsSubmitted() method is called from the DataHandler class. This method takes two parameters, promotion and participants, and adds them to a file called "order.txt" using the Files.write() method. Before writing to the file, it checks if the participants parameter is not empty. If the participants parameter is empty, the data is not written to the file. If there is an IOException, the printStackTrace() method is called to print the stack trace of the exception.  


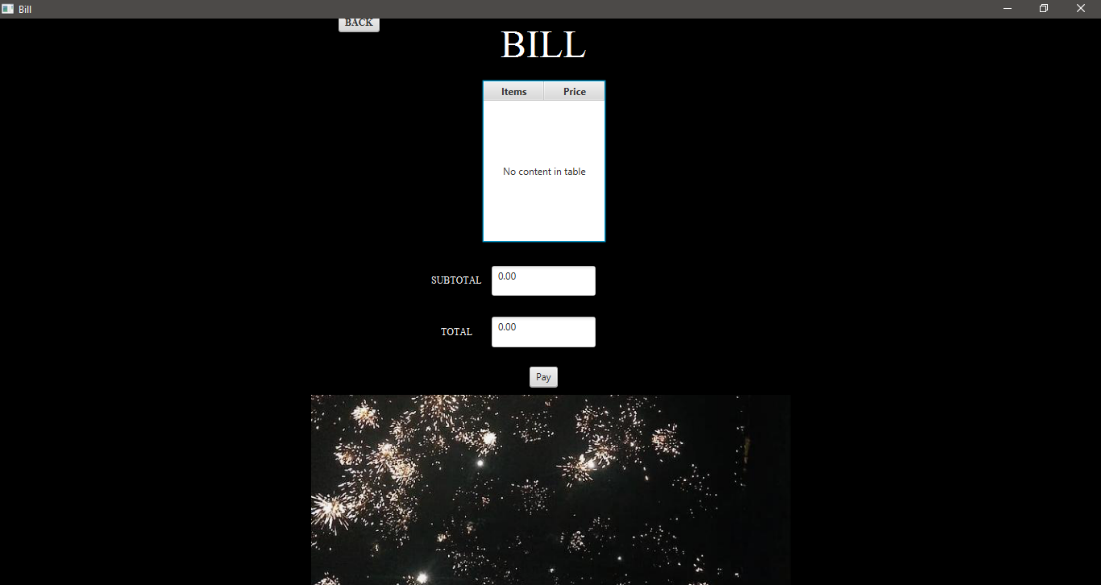
CLASS BILL EXPLANATION:   
The BillController class is the main class that controls the behavior of the GUI. It has several @FXML-annotated fields that represent the different components of the GUI, such as the TableView, TableColumn, TextArea, GridPane, and Button.

The initialize() method is called when the GUI is first loaded, and it sets up the different components of the GUI. The method first reads the items that were ordered from a file called order.txt and adds them to the TableView. It then checks if any of the items have a discount keyword in their name and calculates the subtotal based on the prices of the items and the discount factor. Finally, it sets the values of the Subtotal and Total TextArea fields.

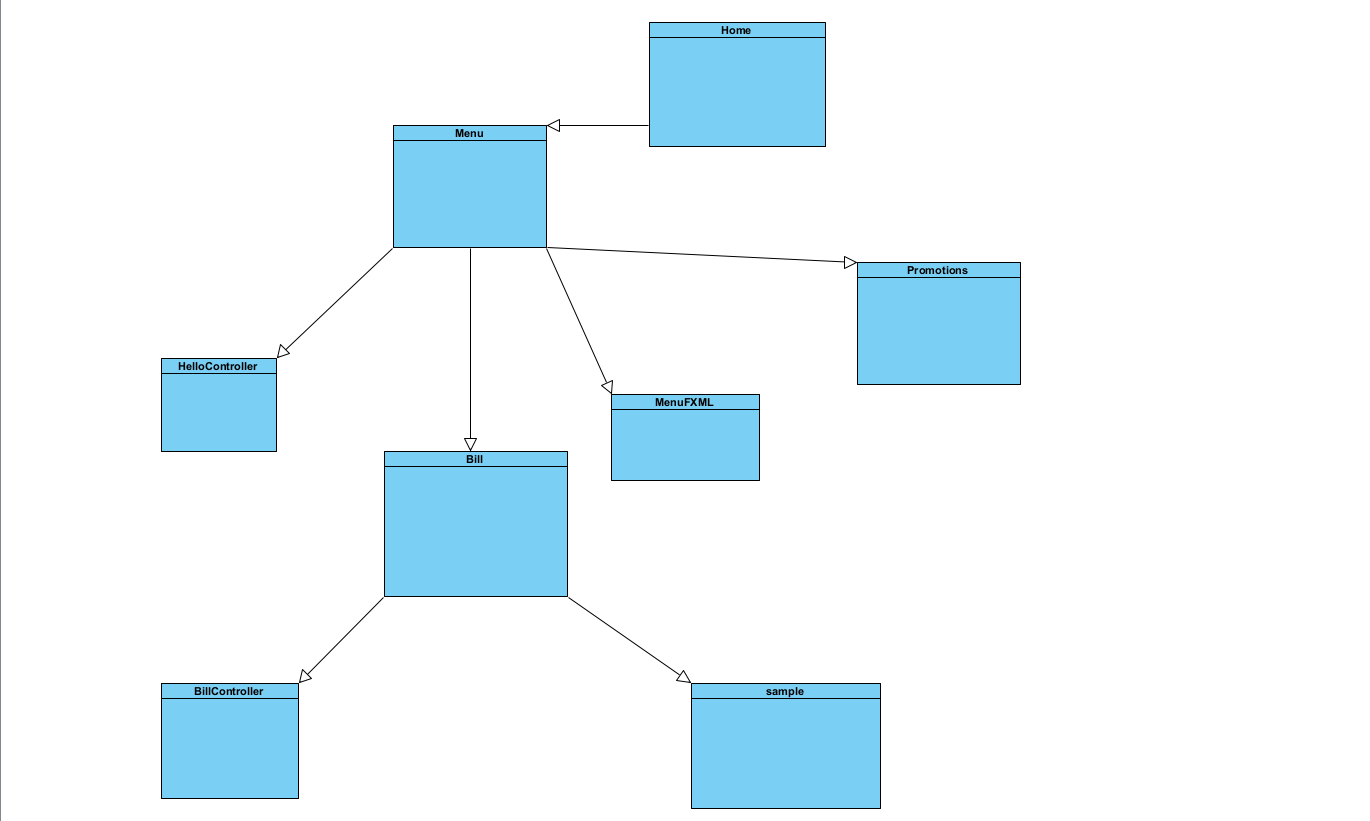
The Payment() method is called when the pay button is clicked. It clears the order.txt file and closes the GUI window.

The backButton2() method is called when the back button is clicked. It returns the user to the restaurant menu.

The readCSV() method is a helper method that reads the items that were ordered from the order.txt file and returns a list of BillItem objects. It also sets the prices of the items based on their names using a series of if statements.



1. Final UML Diagrams



1. Conclusion
   1. Sara’s Conclusion.

In conclusion, the project proved to be an engaging and educational opportunity to strengthen my skills in JavaFX through the further practice of methods and effects, the difficulties encountered and my thoughts regarding future implementation of such skills.

The practice of JavaFX into a working application initially seemed a daunting task however, through research, studying and an insurmountable amount of testing, I was able to not only expand upon my knowledge of JavaFX but also create a fully functional application that both read user input and allowed it to be accessible elsewhere.

However, the testing also brought forth various difficulties. To begin, Eclipse ran into a bug regarding the graphics for the image views. Once encountered, it was remedied by switching to IntelliJ IDEA completely. Unfortunately, the difficulties continued with the image views as it was later discovered that if an image was covering a button completely, the user would be unable to click it, thus rendering our application obsolete and unusable. To fix this, the images had to be made smaller. However, many of the other difficulties were just learning experiences, furthering my knowledge and understanding of JavaFX.

In future creations, I will use the IDE IntelliJ to streamline my work and limit the encountering of any possible bugs. Furthermore, I will remember the challenges overcome and the knowledge gained from this endeavour to use it in future projects.

Thus, in conclusion, through the various challenges and experiences offered in this project, I expanded on my knowledge regarding JavaFX.

* 1. Taha’s Conclusion

To conclude, working on this project provided me with the opportunity to gain a lot of experience in JavaFX. Although, during the process of writing the code, I encountered several difficulties.

Firstly, I found it challenging to work with images since I was still learning JavaFX's ImageView class and the process of loading and displaying images. The images were not showing on the GUI until I used the absolute file path.

Another obstacle I faced was creating a TextField that only allows numeric input. I had to use JavaFX's TextFormatter class to create a filter to restrict the input, which was a bit challenging for me.

Furthermore, I found it difficult to write to a file since I was not familiar with Java's NIO library. Creating a new file if it did not exist and writing to it using the Files class was also confusing. Debugging the code was another challenge I faced. I spent a lot of time trying to get the code to work and was finally able to do so with the help of my groupmates.

Ultimately, these difficulties did broaden my understanding of JavaFX and therefore were beneficial to my learnings.

* 1. Mustafa’s Conclusion

After completing this assignment, I have come to the realization that it is beneficial to divide the code into three distinct classes: the main class, the controller class, and the fxml class. The main class contains the code necessary to run the GUI; the controller class contains all the functional code, and the fxml class contains the actual GUI code. Although one class could house the functionality of the GUI and the main method in the assignment, separating them helps avoid the issue of connecting too many classes to one another. Furthermore, I have gained knowledge on how to link classes, which I previously had difficulty with. With the help of my partner, Sara, I was able to connect two distinct classes flawlessly. The main issue I encountered was incorporating the information from the menu class and the promotional class into my Bill class. I used the File function to solve the issue. While my peers used file input to input data into a text file, I utilized file output. This involved reading data from the file into my program and displaying it on my table using dynamic ArrayList. The only challenge I faced was deleting the data in the shared text file amongst the three classes. If the user added new items to the bill, they would not be able to view the prior items that had been added to the table, so I had to clear the text file to allow them to do that. In conclusion, I have learned that proper linking of all classes is crucial once they have all been created. This ensures that the scripts are created without the need to dramatically alter anything due to an unforeseen error.