Final Project - Documentation

COIS-2240H: Software Design and Modelling
Matthew Brown, #0648289
Seth Hannah, #0656551
Melissa Van Bussel, #0579124

Package: kiosk

Class: SplashController

`public class` **`SplashController`**

- The SplashController Class controls the splash screen, which is what the user sees when first opening the app. This class is associated with the splash.fxml file.
  - ***Field Detail***
    - **imgSplashScreen**
      - `private javafx.scene.image.ImageView imgSplashScreen`
      - The image displayed at the start of the app.
  - ***Constructor Detail***
    - **SplashController**
      - `public SplashController()`
  - ***Method Detail***
    - **triggerSplash**
      - `void triggerSplash()`
      - Switches the scene to the homeScreen. This method is called when the user taps anywhere on the splash screen.

Package: kiosk

Class: HomeController

`public class` **`HomeController`**

- The HomeController Class is the controller for the screen the user first sees. It is associated with the `home.fxml` file. It loads in several instances of `HomeItemController` to fill the screen with.
    - ***Field Detail***
        - **BANNER_HEIGHT**
            - `public static final int BANNER_HEIGHT`
            - The height of the banner on each screen. This is equal to 1/4 of the height of the app.
        - **OBJECT_HEIGHT**
            - `public static final int OBJECT_HEIGHT`
            - The height of the scroll panes on the home screen and category screens.
        - **OBJECT_SIDE_OFFSET**
            - `public static final int OBJECT_SIDE_OFFSET`
            - The side offset for objects on the home screen, category screens, and order screen.
        - **OBJECT_TOP_POS**
            - `public static final int OBJECT_TOP_POS`
            - The Y-position of objects below banners.
        - **GRID_GAP**
            - `public static final int GRID_GAP`
            - The spacing between items in the grid containing the 6 different categories.
        - **root**
            - `javafx.scene.layout.AnchorPane root`
            - The Anchor Pane which contains all of the other nodes, on the home screen.

- **typesGrid**
  - `javafx.scene.layout.GridPane typesGrid`
  - The Grid Pane which contains the 6 different categories on the home screen.
- **sideBar**
  - `static javafx.scene.layout.AnchorPane sideBar`
  - The Anchor Pane which contains the sidebar / navigation drawer.

- ***Constructor Detail***
  - **HomeController**
    - `public HomeController()`

- ***Method Detail***
  - **Initialize**
    - `public void initialize(java.net.URL location, java.util.ResourceBundle resources)`
    - Initializes the home screen.
  - **openSideMenu**
    - `public void openSideMenu()`
    - Opens the navigation drawer. This method is called when the user presses the hamburger button.

Package: kiosk

Class: NavigationController

`public class` **`NavigationController`**

- The NavigationController Class is the controller for the navigation drawer. It is associated with the navigationDrawer.fxml file.
  - ***Field Detail***
    - **NAV_BANNER_HEIGHT**
      - `public static final int NAV_BANNER_HEIGHT`
      - The height of the banner in the navigation drawer.
    - **WIDTH**
      - `public static final int WIDTH`
    - **HEIGHT**
      - `public static final int HEIGHT`
      - The height of the VBox in the navigation drawer. This is equal to the height of the app minus the height of the banner. This VBox contains the buttons in the navigation drawer.
    - **sideBar**
      - `javafx.scene.layout.AnchorPane sideBar`
      - The Anchor Pane which contains all of the other nodes.
  - ***Constructor Detail***
    - **NavigationController**
      - `public NavigationController()`
  - ***Method Detail***
    - **initialize**
      - `public void initialize(URL location, ResourceBundle resources)`

- Initializes the navigation drawer. Generates the buttons when the navigation drawer is opened.

- **closeSideMenu**

- `public void closeSideMenu()`
- This method closes the navigation drawer. It is called when the user pushes the "X" button (hamburger-close).

Package: kiosk

Class: CategoryController

`public class` **`CategoryController`**

- The CategoryController Class is the controller for the category screens. It is associated with the `category.fxml` file. It loads several instances of `CategoryItemController` to fill the screen with each category.
  - ***Field Detail***
    - **root**
      - `javafx.scene.layout.AnchorPane root`
      - The AnchorPane which contains all of the other nodes.
    - **hamburger**
      - `javafx.scene.control.Button hamburger`
      - The button which allows the user to open the navigation drawer.
    - **orderButton**
      - `javafx.scene.control.Button orderButton`
      - The button which allows the user to view their order, by switching to the order screen.
    - **typesGrid**
      - `javafx.scene.layout.GridPane typesGrid`
      - The grid which contains the 6 categories.
    - **numberPane**
      - `javafx.scene.layout.StackPane numberPane`
      - The Pane which displays the number of items the user has added to their order.
    - **orderNumber**
      - `javafx.scene.text.Text orderNumber`
      - Contains the number of food or drink items currently in the user's order.

- **banner**
  - `javafx.scene.image.ImageView banner`
  - The image at the top of the screen.
- **category**
  - `private java.lang.String category`
  - Each instance of CategoryController corresponds to one of 6 categories, given by the `category` field.
- **sideBar**
  - `static javafx.scene.layout.AnchorPane sideBar`
  - The Anchor Pane which contains the sidebar / navigation drawer.

- ***Constructor Detail***
  - **CategoryController**
    - `CategoryController(java.lang.String category)`
    - This constructor is used to create an instance of CategoryController for each category.
    - **Parameters:**
    - `category - The food category (e.g., Salad, Burgers, etc)`

- ***Method Detail***
  - **initialize**
    - `public void initialize(URL location, ResourceBundle resources)`
    - Initializes the Category screen for the given category.
  - **openSideMenu**
    - `private void openSideMenu()`
    - Opens the navigation drawer. This method is called when the user presses the hamburger button.
  - **incrementOrderIcon**
    - `public void incrementOrderIcon()`

- Increases the number in `numberPane` on top of the "View Order" button. This method is invoked any time a user adds to their order.

- **viewOrder**

- `private void viewOrder()`
- Opens the order screen. This method is called when the user presses the "View Order" button.

Package: kiosk

Class Order

`public class Order`

- `Order` is the class which handles the user's order of food and drink items. An `Order` object contains a list of the food and drink items currently in the user's order. Food and drink items can be added to the order, removed from the order, or the order can be reset.
  - ***Field Detail***
    - **items**
      - `private javafx.collections.ObservableList<Item> items`
      - The list of food and drink items currently in the user's order. An `Observable List` of `Item` objects.
  - ***Constructor Detail***
    - **Order**
      - `Order()`
      - Constructs an empty `observableArrayList` of `Item` objects. An instance of `Order` is empty until the user adds an item to their order.
  - ***Method Detail***
    - **getItems**
      - `javafx.collections.ObservableList<Item> getItems()`
      - Gets the `items` field.
      - **Returns:**
      - `ObservableList<Item> The list of food and drink items currently in the user's order.`
    - **resetOrder**
      - `void resetOrder()`
      - Resets the user's order, by clearing all `Item` objects from the `items` field.
    - **addToOrder**

- `public void addToOrder(java.lang.String name, float price)`
- Adds food / drink item to the user's order. If the user has already ordered at least one item with the given name, the quantity is increased by 1. If the user has not ordered the item yet, a new `Item` object is created, and is added to the `items` list. For example, `addToOrder("Little Leaf Meal", 3.69F)` would increase the quantity of "Little Leaf Meal"s in the user's order if the quantity before method invocation was at least one. Otherwise, a new `Item` object would be created, and this object would be added to `items`.
- **Parameters:**
- `name - The name of the food or drink item to be added to the order.`
- `price - The price of the food or drink item to be added to the order.`

- **calculateSubtotal**

- `float calculateSubtotal()`
- Calculates the cost of the user's order before tax (the subtotal).
- **Returns:**
- `float The subtotal of the user's order, before taxes are added.`

- **getLength**

- `int getLength()`
- Gets the total number of food items in the user's order, by adding up the quantities of each Item in this instance. This method is used for determining which number to display in the red circle on the "View Order" button on the category screens.
- **Returns: int** `The number of items in the user's order.`

Package: kiosk

Class: OrderController

`public class` **`OrderController`**

- The OrderController Class controls the order screen. It is associated with the `order.fxml` file.
  - ***Field Detail***
    - **TABLE_HEIGHT**
      - `public static final int TABLE_HEIGHT`
      - The height of the table which displays the items currently in the user's order.
    - **SUBTOTAL_POS**
      - `public static final int SUBTOTAL_POS`
      - The Y-position of the "subtotal" text field on the order screen.
    - **HST_POS**
      - `public static final int HST_POS`
      - The Y-position of the "hst" text field on the order screen.
    - **TOTAL_POS**
      - `public static final int TOTAL_POS`
      - The Y-position of the "total" text field on the order screen.
    - **POPUP_WIDTH**
      - `public static final int POPUP_WIDTH`
      - The width of the confirmation popup.
    - **POPUP_HEIGHT**
      - `public static final int POPUP_HEIGHT`
      - The height of the confirmation popup.
    - **ANCHOR_SIDE_DISTANCE**

- `public static final int ANCHOR_SIDE_DISTANCE`
- The distance that the confirmation popup is from the sides of the window.

- **ANCHOR_TOP_BOTTOM_DISTANCE**
  - `public static final int ANCHOR_TOP_BOTTOM_DISTANCE`
  - The distance that the top and bottom of the confirmation popup are from the top and bottom respectively of the window. The confirmation popup is centered.

- **root**
  - `javafx.scene.layout.AnchorPane root`
  - The AnchorPane for the order screen in which all of the other nodes are contained.

- **orderTable**
  - `public javafx.scene.control.TableView<Item> orderTable`
  - The table which contains the items currently in the user's order.

- **hamburger**
  - `javafx.scene.control.Button hamburger`
  - The button which allows the user to open the navigation drawer.

- **submitOrder**
  - `javafx.scene.control.Button submitOrder`
  - The button which allows the user to send their order to the kitchen.

- **resetOrder**
  - `javafx.scene.control.Button resetOrder`
  - The button which allows the user to reset their order. Pushing this button clears the order table.

- **confirmation**
  - `javafx.scene.layout.AnchorPane confirmation`

- The pane which holds text telling the user their order number, informs them that their request was shipped off to the kitchen, and offers a button to reset the program back to the Splash Screen.

- **orderNumber**
  - `javafx.scene.text.Text orderNumber`
  - A reference to the `Text` object that holds the "Order #....." text.
- **continueButton**
  - `javafx.scene.control.Button continueButton`
  - A reference to the button that resets the program when the user is complete and has been shown their order number.
- **hst**
  - `javafx.scene.control.TextField hst`
  - The TextField which displays the amount of HST (Harmonized Sales Tax) for the user's order.
- **subtotal**
  - `javafx.scene.control.TextField subtotal`
  - The TextField which displays the subtotal (before tax) for the user's order.
- **total**
  - `javafx.scene.control.TextField total`
  - The TextField which displays the total (including tax) for the user's order.
- **sideBar**
  - `static javafx.scene.layout.AnchorPane sideBar`
  - The AnchorPane which contains the sidebar / navigation drawer.

- ***Constructor Detail***
  - **OrderController**
    - `public OrderController()`
- ***Method Detail***

- **Initialize**
  - `public void initialize(java.net.URL location, java.util.ResourceBundle resources)`
  - Initializes the order screen.
- **generateTable**
  - `private void generateTable(javafx.scene.control.TableView<Item> table)`
  - Generates the table which displays the items currently in the user's order.
  - **Parameters:**
  - `table - A TableView of Items to be filled with data from the oder.getItems() method.`
- **refreshLabels**
  - `public void refreshLabels()`
  - Refreshes the subtotal, HST, and total for the user's order, and redraws the text on-screen.
- **openSideMenu**
  - `private void openSideMenu()`
  - Opens the side bar / navigation drawer. This method is called when the user pushes the hamburger button.
- **displayConfirmation**
  - `private void displayConfirmation()`
  - Generates an order number for the user, then shows the pane holding the order number and the button which allows them to end their session.
- **clearOrder**
  - `private void clearOrder()`

- Clears the items in the user's order, then refreshes the table and text-fields for prices.

Package: kiosk

Class: Main

`public class Main`

- ***Field Detail***

  - **DB**
    - `public static Database DB`
    - The instance of Database which allows the app to connect to master.db.
  - **order**
    - `public static Order order`
    - The instance of Order which keeps track of the items in the user's order.
  - **homeScreen**
    - `private static javafx.scene.Scene homeScreen`
  - **splashScreen**
    - `private static javafx.scene.Scene splashScreen`
  - **orderScreen**
    - `private static javafx.scene.Scene orderScreen`
  - **orderController**
    - `private static OrderController orderController`
  - **WIDTH**
    - `public static final int WIDTH`
  - **HEIGHT**
    - `public static final int HEIGHT`
  - **HALF_WIDTH**
    - `public static final int HALF_WIDTH`
  - **QUARTER_HEIGHT**

- `static final int QUARTER_HEIGHT`

- ***Method Detail***
  - **getHomeScreen**
    - `static javafx.scene.Scene getHomeScreen()`
  - **getSplashScreen**
    - `static javafx.scene.Scene getSplashScreen()`
  - **getOrderScreen**
    - `static javafx.scene.Scene getOrderScreen()`
  - **getOrderController**
    - `public static OrderController getOrderController()`
  - **start**
    - `public void start(Stage primaryStage throws java.lang.Exception`
  - **selectCategory**
    - `public static void(Node root, String category) java.lang.String category)`
  - **main**
    - `public static void main(java.lang.String[] args)`
  - **resetKiosk**
    - `static void resetKiosk(javafx.stage.Stage primaryStage)`
    - Simply triggers the splash screen back onto the display. Nothing else is required to restart the program.

Package: kiosk.backend

Class: Database

`public class **Database**`

- The `Database` class allows connections to the database to be opened and closed. A connection to `master.db` is created at the start of the program, and is closed at the end of the program.
    - ***Field Detail***
        - **connection**
            - `private java.sql.Connection connection`
            - The connection to the database.
    - ***Constructor Detail***
        - **Database**
            - `public Database(java.lang.String filePath)`
            - Constructor which creates a `Connection` to the database specified by the `filepath`.
            - **Parameters:**
            - `filePath - A String containing the filepath to the desired database file.`
    - ***Method Detail***
        - **closeConnection**
            - `public void closeConnection()`
            - Closes the connection to the database.
        - **makeStatement**
            - `java.sql.Statement makeStatement()`
            - Creates a `Statement` object for the database connection. A `Statement` object is created any time the database is queried, which occurs in the `Menu` class.

- **Returns:**
- `Statement The generated Statement object for the connection field.`

Package: kiosk.backend

Class: Item

`public class` **`Item`**

- Objects of the `Item` class are food or drink items in the user's order.
  - ***Field Detail***
    - **name**
      - `private java.lang.String name`
      - The name of the food or drink item in the user's order.
    - **price**
      - `private float price`
      - The price of the food or drink item in the user's order.
    - **quantity**
      - `private int quantity`
      - The quantity of the food or drink item in the user's order.
    - **increaseQuantityButton**
      - `private javafx.scene.control.Button increaseQuantityButton`
      - The button that is visible in the table on the order screen, which allows user to increase quantity of this food or drink item in their order by 1.
    - **decreaseQuantityButton**
      - `private javafx.scene.control.Button decreaseQuantityButton`
      - The button that is visible in the table on the order screen, which allows user to decrease quantity of this food or drink item in their order by 1.
  - ***Constructor Detail***
    - **Item**

- `public Item(String name, float price)`
- Constructor which creates a food or drink `Item` with a default quantity of 1. This constructor is called when the user adds a new food or drink item to their order.
- **Parameters:**
- `name - The name of the food or drink item.`
- `price - The price of the food or drink item.`

- ***Method Detail***
  - **getName**
    - `public java.lang.String getName()`
    - Gets the name of the food or drink item.
    - **Returns:**
    - `String The name of the food or drink item.`
  - **getPriceString**
    - `public java.lang.String getPriceString()`
    - Gets the price of the food or drink item. Called by the CellValueFactory in `OrderController.java` to format the price properly.
    - **Returns:**
    - `String The price of the food or drink item, formatted as local currency.`
  - **getPrice**
    - `public float getPrice()`
  - **getQuantity**
    - `public int getQuantity()`
    - Gets the quantity of the food or drink item in the user's order.
    - **Returns:**
    - `int The number of items in the user's order with name this.name.`
  - **setQuantity**

- `public void setQuantity(int quantity)`

- **getIncreaseQuantityButton**
  - `public javafx.scene.control.Button getIncreaseQuantityButton()`

- **getDecreaseQuantityButton**
  - `public javafx.scene.control.Button getDecreaseQuantityButton()`

Package: kiosk.backend

Class Menu

`public class **Menu**`

- The `Menu` class is a static class which contains methods for querying the Menu Table in the database.
  - ***Constructor Detail***
    - **Menu**
      - `public Menu()`
  - ***Method Detail***
    - **generateTypes**
      - `public static java.util.ArrayList<java.lang.String> generateTypes()`
      - Gets a list of 'types' (Breakfast, Burgers, etc.) from the Menu Table in the database.
      - **Returns:**
      - `ArrayList<String> A list of all food 'types' in the Menu Table in the database.`
    - **getItemsByType**
      - `public static java.util.ArrayList<java.lang.String> getItemsByType(java.lang.String type)`
      - Gets a list of 'items' (Vanilla Cone etc.) from the Menu Table in the database, by the type (Snacks and Treats etc.)
      - **Parameters:**
      - `type - The type of item (e.g., Breakfast, Burgers, Salads, etc.).`
      - **Returns:**
      - `ArrayList<String> A list containing the names of all items in the Menu Table for the specified type.`

- **getFilepath**
  - `public static java.lang.String getFilepath(java.lang.String name)`
  - Gets the filepath of the image from the Menu Table in the database, for the food item named `name`.
  - **Parameters:**
  - `name - The name of the food or drink item for which the filepath is desired.`
  - **Returns:**
  - `String The filepath of the image corresponding to the food item named name.`
- **getPrice**
  - `public static float getPrice(java.lang.String name)`
  - Gets the price from the Menu Table in the database, for the food item named `name`.
  - **Parameters:**
  - `name - The name of the food or drink item for which the price is desired.`
  - **Returns:**
  - `float The price of the food or drink item named name.`

Package: kiosk.loadIns

Class: CategoryItemController

`public class` **`CategoryItemController`**

- The controller for each food-item the user can choose from. An instance of `CategoryController` will load as many of these (along with it's associated `.fxml` file) into the program as there are items in its category. This controller controls the button and image for each food item.
  - ***Field Detail***
    - **root**
      - `public javafx.scene.layout.StackPane root`
    - **image**
      - `public javafx.scene.image.ImageView image`
    - **button**
      - `public javafx.scene.control.Button button`
    - **itemName**
      - `public java.lang.String itemName`
    - **parentController**
      - `private CategoryController parentController`
  - ***Constructor Detail***
    - **CategoryItemController**
      - `public CategoryItemController(java.lang.String itemName, CategoryController parent)`
      - Constructor used to tell the `.fxml` file which items to load into place.
      - **Parameters:**
      - `itemName - The official name of the food item in question.`

- `parent - The CategoryController which loaded in this CategoryItem. this is used to trigger a method inside that category's screen.`

- ***Method Detail***
  - **setImageURL**
    - `public void setImageURL(java.lang.String URL)`
    - Sets the image of the CategoryItem.
    - **Parameters:**
    - `URL - The location of the image.`
  - **addToOrder**
    - `public void addToOrder()`
    - Adds the item that this `CategoryItem` references to the Main order.

Package: kiosk.loadIns

Class: HomeItemController

`public class` **`HomeItemController`**

- This controller (related to `homeItem.fxml`)
  - ***Field Detail***
    - **IMAGE_WIDTH**
      - `public static final int IMAGE_WIDTH`
      - The width of images on the home screen.
    - **IMAGE_HEIGHT**
      - `public static final int IMAGE_HEIGHT`
      - The height of images on the home screen.
    - **category**
      - `private java.lang.String category`
    - **root**
      - `public javafx.scene.layout.StackPane root`
    - **image**
      - `public javafx.scene.image.ImageView image`
    - **button**
      - `public javafx.scene.control.Button button`
  - ***Constructor Detail***
    - **HomeItemController**
      - `public HomeItemController(java.lang.String category)`
  - ***Method Detail***
    - **setImageURL**
      - `public void setImageURL(java.lang.String URL)`

- **selectCategory**
  - `public void selectCategory()`