Hyeon Oh, Nikhil Agarwal

**Project 4 Test Specifications**

**Class Name: MainController**

**View Name: StoreFrontView.fxml**

**Method Signature: protected void displayDonutView() {} //opens up a new window for ordering donuts when clicking on Donuts image button**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the Donuts image button which will show a new window for ordering donuts. | * Start the program   and click on the Donuts image button. | Will show a new window for ordering donuts. |

**Method Signature: protected void displayCoffeeView() {} //opens up a new window for ordering coffee when clicking on Coffee image button**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the Coffee image button which will show a new window for ordering coffee. | * Start the program and click on the Coffee image button. | Will show a new window for ordering coffee. |

**Method Signature: protected void displayBasketView() {} //opens up a new window for the order basket when clicking on “Your Basket” image button**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the “Your Basket” image button which will show a new window for the order basket. | * Start the program and click on the “Your Basket” image button | Will show a new window for the order basket. |

**Method Signature: protected void displayStoreView() {} //opens up a new window for all the orders currently when clicking on “Store Orders” image button**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the “Store Orders” image button which will show a new window for the store orders which show all the orders placed currently. | * Start the program and click on the “Store Orders” image button | Will show a new window for the store orders. |

**Class Name: OrderDonutsController**

**View Name: donutsView.fxml**

**Method Signature: public void setInitialBasket() {} //the user cannot make a donut order until he/she clicks the “Load Basket” button**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the “Donuts” image button which will show a new window for the order basket. Then to make a donut order, the user must click on the “Load Basket” button first. | * The user must click on the “Load Basket” button first in order to make a donut order * Test Data: Click on “Load Basket” button | The user can now make a donut order after the “Load Basket” button has been clicked. |

**Method Signature: public void addToBasket() {} //the user can add the order to the donuts order basket which is shown on the ListView to the right before adding the order to the order basket view**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the “Donuts” image button which will show a new window for the order basket. Then to make a donut order, the user must click on the “Load Basket” button first.  The user can now make a donut order and once the “Add to Basket” button is clicked, the order will be shown on the right ListView which can then be added to the order basket view/window. | * The user must be in the donuts order view window in order to order donuts. The user must click on the “Load Basket” button to be able to make an order. After choosing the donut type and flavor, the user can add the donut to the donut check out listview, where the user can then finally add the order to the order basket view. * Test Data:   Type of Donut: Yeast Donut  Amount: 2  Flavor: Jelly | After clicking on “Add to Basket”, the ListView on the right side will show the donut order.  In this specific example, the ListView will show  2 x Yeast Donut - Jelly |

**Method Signature: public void removeButtonClicked() {} //the specific donut order row that is shown in the right side ListView will be removed**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the “Donuts” image button which will show a new window for the order basket. Then to make a donut order, the user must click on the “Load Basket” button first.  The user can now make a donut order and once the “Add to Basket” button is clicked, the order will be shown on the right ListView which can then be added to the order basket view/window. The right side ListView will now show the donut orders that can be added to the order basket. When clicking on a donut order row and clicking on the remove button, the order will be deleted. | * The user must be in the donuts order view window in order to order donuts. The user must click on the “Load Basket” button to be able to make an order. After choosing the donut type and flavor, the user can add the donut to the donut check out listview, where the user can then finally add the order to the order basket view. Once the order is added, the right side ListView will contain the donut orders. The row can then be selected and removed if the user wants to get rid of the order. * Test Data:   Type of Donut: Yeast Donut  Amount: 2  Flavor: Jelly | The right side ListView which is essentially the shopping cart will remove the donut order row. |

**Method Signature: public void addToOrderButtonClicked () {} //Adds the donut order/s from the right side ListView (shopping cart) to the order basket view/window**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the “Donuts” image button which will show a new window for the order basket. Then to make a donut order, the user must click on the “Load Basket” button first.  The user can now make a donut order and once the “Add to Basket” button is clicked, the order will be shown on the right ListView which can then be added to the order basket view/window. The right side ListView will now show the donut orders that can be added to the order basket.  When clicking on “Add Basket To Order”, the user will add the donut orders to the order basket view/window. | * The user must be in the donuts order view window in order to order donuts. The user must click on the “Load Basket” button to be able to make an order. After choosing the donut type and flavor, the user can add the donut to the donut check out listview, where the user can then finally add the order to the order basket view. Once the order is added, the right side ListView will contain the donut orders. The user can then click on “Add Basket To Order” to add the donut orders to the order basket view/window. * Test Data:   Type of Donut: Yeast Donut  Amount: 2  Flavor: Jelly | When adding the basket to order, the right side ListView which shows the donut orders is removed from the ListView and added to the order basket view/window. The user can continue to add more donuts if he/she wishes to do so. |

**Method Signature: public void displaySelected() {} //the user is able to pick from three different donut flavors which are yeast donuts, cake donuts, and donut holes. When clicking on these different donut flavors, different flavors for each specified donuts can be chosen and a different image of the donut will appear in the GUI**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the “Donuts” image button which will show a new window for the order basket. Then to make a donut order, the user must click on the “Load Basket” button first. Now, the user is able to use the combobox to see which donut flavors are available. Selecting the yeast donut type will allow for 6 different flavors of yeast donuts which are classic, glazed, jelly, cinnamon sugar, boston cream, and powdered sugar. The imageview will show an image of a yeast donut. | * The user must be in the donuts order view window in order to order donuts. The user must click on the “Load Basket” button to be able to make an order. In this test case, the user will choose the yeast donut in the combobox. * Test Data:   Type of Donut: Yeast Donut | Selecting the yeast donut type will allow for 6 different flavors of yeast donuts which are classic, glazed, jelly, cinnamon sugar, boston cream, and powdered sugar. The imageview will show an image of a yeast donut. |
| 2 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the “Donuts” image button which will show a new window for the order basket. Then to make a donut order, the user must click on the “Load Basket” button first.  Now, the user is able to use the combobox to see which donut flavors are available. Selecting the cake donut type will allow for 3 different flavors of cake donuts which are Plain Vanilla, Chocolate Cake, and Strawberry Short-Cake. The imageview will show an image of a cake donut. | * The user must be in the donuts order view window in order to order donuts. The user must click on the “Load Basket” button to be able to make an order. In this test case, the user will choose the cake donut in the combobox. * Test Data:   Type of Donut: Cake donut | Selecting the cake donut type will allow for 3 different flavors of yeast donuts which are plain vanilla, chocolate, and strawberry short-cake. The imageview will show an image of a cake donut. |
| 3 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the “Donuts” image button which will show a new window for the order basket. Then to make a donut order, the user must click on the “Load Basket” button first.  . Now, the user is able to use the combobox to see which donut flavors are available. Selecting the donut holes type will allow for 3 different flavors of donut holes which are glazed, chocolate, and jelly. The imageview will show an image of a donut hole. | * The user must be in the donuts order view window in order to order donuts. The user must click on the “Load Basket” button to be able to make an order. In this test case, the user will choose the donut holes donut type in the combobox. * Test Data:   Type of Donut: Donut holes | Selecting the donut holes donut type will allow for 3 different flavors of donut holes are glazed, chocolate, and jelly. The imageview will show an image of donut holes. |

**Class Name: OrderCoffeeController**

**View Name: orderingCoffee.fxml**

**Method Signature: public void addCoffeeToOrderClicked() {} //adds the coffee order to the order basket**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the Coffee image button which will show a new window for ordering coffee. Then, the user must complete some actions before clicking on the ”Add to Order” button. | * The user must first pick a size, then he/she can pick add-ons to the coffee and the number of coffee to order. Then the user will be able to add the coffee to the order basket. * Test Data:   Size: Short  Add-Ons: Mocha  Amount: 2 | The running total for the coffee will display $4.38.  The order will be added to the order basket.  In the order basket view, the basket will show that the order has the Mocha add-on. |
| 2 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the Coffee image button which will show a new window for ordering coffee. Then, the user must complete some actions before clicking on the ”Add to Order” button. When adding no add-ons, there will be text that specifies that no add-ons were selected in the order basket view. | * The user must first pick a size, then he/she can pick add-ons to the coffee and the number of coffee to order. Then the user will be able to add the coffee to the order basket.   This time, the user will select no add-ons.   * Test Data:   Size: Short  Add-Ons:   * Amount: 2 | The running total for the coffee will display $3.78. The order will be added the order basket. The order basket view will show that this order did not have any add-ons to the coffee. |
| 3 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the Coffee image button which will show a new window for ordering coffee. Then, the user must complete some actions before clicking on the ”Add to Order” button. A user can add multiple add-ons to his/her coffee if they please. | * The user must first pick a size, then he/she can pick add-ons to the coffee and the number of coffee to order. Then the user will be able to add the coffee to the order basket.   This time, the user will select multiple add-ons.   * Test Data:   Size: Short  Add-Ons: Mocha, Caramel  Amount: 2 | The running total for the coffee will display $4.98.  The order will be added to the order basket. The order basket view will show that this order had multiple add-ons for the coffee requested. |

**Class Name: OrderBasketController**

**View Name: orderBasket.fxml**

**Method Signature: public void setLoadBasket() //loads the orders that were added to the orders from the donuts and coffee orders**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the “Your Basket” image button which will show a new window for the order basket. The user can click on the load basket button to see the orders placed in the basket. If the user has no orders placed in the basket, there will be an error message in the ListView. | * The user can click on load basket button to see the orders placed in the basket. For this test case, the user clicks on the load basket without an order being placed in the basket. * Test Data: Click on load basket button without placing an order into the basket. | The ListView will display “Basket is Empty!” |
| 2 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the “Your Basket” image button which will show a new window for the order basket. The user can click on the load basket button to see the orders placed in the basket. If the user has an order placed, it will show the orders and the running sub-total, sales tax, and the total amount. | * The user can click on load basket button to see the orders placed in the basket. For this test case, the user clicks on the load basket after placing an order into the basket. * Test Data: The user will add a coffee order to the basket.   For coffee:  Size: Short  Add-Ons: Mocha  Amount: 2  Then in basket window, click on load basket button. | 2 x Coffee – Size: Short, Add-Ons: [Mocha]  Sub-Total: $4.38  Sales Tax: $0.29  Total Amount: $4.67 |

**Method Signature: public void removeSelectedItem() {} //removes a specific order from the order basket**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the “Your Basket” image button which will show a new window for the order basket. The user can click on the load basket button to see the orders placed in the basket. If there is an order(s), the user can click on the order row and click on the “Remove Selected Item” button to remove that order from the order basket. | * After the user adds an order to the order basket, the user can select the row order and click on the “Remove Selected Item” button to remove that order from the order basket. * Test Data: The user will add a coffee order to the basket.   For coffee:  Size: Short  Add-Ons: Mocha  Amount: 2  Then in basket window, click on load basket button. The user will then click on the order row and click on the “Remove Selected Item” button to remove that order from the order basket. | The order will be removed and new sub-total, sales tax, and total amount of the orders will be calculated. If there was only one order in the basket, the ListView will say “Basket is Empty!” |

**Method Signature: public void placeOrder() {} //user can place the order which will be sent to the store orders**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | The user must start the program for the StoreFrontView view to be displayed. From there, the user must click on the “Your Basket” image button which will show a new window for the order basket. The user can click on the load basket button to see the orders placed in the basket. The user can click on the “Place Order” button to place the order which will be sent to the store orders. | * After the user adds an order to the order basket, the user can load the basket and place the orders into the store orders by clicking on “Place Order” * Test Data: The user will add a coffee order to the basket.   For coffee:  Size: Short  Add-Ons: Mocha  Amount: 2  Then in basket window, click on load basket button. The user will then click on the ”Place Order” button to place the order which will be sent to the store orders. | Order placed!  The Store Orders view will receive the order from the order basket. |

**Class Name: StoreOrderController**

**View Name: storeOrders.fxml**

**Method Signature: public void makeList() {} //shows the orders that were placed that are in the ComboBox**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | Once a user puts their order into the order basket, the user can then place the order which they can now view in the store orders view/window. The user can then click on the combobox to check their order number and their order details in the listview below. If there is no order placed yet, there will be a message indicating so. | * After the user adds an order to the order basket, the user can load the basket and place the orders into the store orders by clicking on “Place Order”   The user can now navigate to the store orders view/window and click on the order numbers combobox to see their order number and their order details.   * Test Data:   There will be no orders placed. | No Orders Yet |
| 2 | Once a user puts their order into the order basket, the user can then place the order which they can now view in the store orders view/window. The user can then click on the combobox to check their order number and their order details in the listview below. | * After the user adds an order to the order basket, the user can load the basket and place the orders into the store orders by clicking on “Place Order”   The user can now navigate to the store orders view/window and click on the order numbers combobox to see their order number and their order details.   * Test Data:   The user will place two coffee orders separately, resulting in two coffee orders in the store orders view/window.  Coffee Order 1:  Size: Short  Amount: 1  Add-Ons: Mocha  Coffee Order 2:  Size: Tall  Amount: 2  Add-Ons: Caramel | In the Order Numbers combobox, there will be two orders listed with the order numbers randomized.  Order (Order Number):  945  895 |

**Method Signature: public void cancelSelectedOrder() {} //will cancel the specific order that is placed**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | Once a user puts their order into the order basket, the user can then place the order which they can now view in the store orders view/window. The user can then click on the combobox to check their order number and their order details in the listview below. Once they can see their order and order details in the ListView, the user can click on the “Cancel Order” button to cancel the order. | * After the user adds an order to the order basket, the user can load the basket and place the orders into the store orders by clicking on “Place Order”   The user can now navigate to the store orders view/window and click on the order numbers combobox to see their order number and their order details. The user can then click on the ”Cancel Order” button if he/she wishes to cancel the order.   * Test Data:   The user will place two coffee orders separately, resulting in two coffee orders in the store orders view/window.  Coffee Order 1:  Size: Short  Amount: 1  Add-Ons: Mocha  Coffee Order 2:  Size: Tall  Amount: 2  Add-Ons: Caramel  The user will cancel the first order | In the Order Numbers combobox, the user will cancel the first order so there will only be one order remaining in the combobox. The ListView will remove the order. |

**Method Signature: public void exportClicked() {} //will create a file that stores the user orders if it does not exist, will export the user orders to the file**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Requirement | Test Description and Input Data | Expected Result / Output |
| 1 | Once the user has his/her order placed, the user can wish to export their order which stores their order into a text file.  If the text file to export the orders do not exist, an orderSummary.txt file will be created which will store the orders. | * For this test case, the user will click on the export button without placing an order first. | Order Summary created.  Filename: orderSummary.txt  Please close all tabs to view this file.  In the orderSummary.txt  It will show this text:  Order Summary:  There are currently no orders.  Have a nice day! |
| 2 | Once the user has his/her order placed, the user can wish to export their order which stores their order into a text file. | * For this test case, the user will place a coffee order then click on the export button. * Test Data:   Coffee Size: Short  Amount: 1  Add-Ons: Mocha | Order Summary created.  Filename: orderSummary.txt  Please close all tabs to view this file.  In the orderSummary.txt  It will show this text:  Order Summary:  There are currently no orders.  Have a nice day! Order Summary:  Order Number: 504 1 x Coffee - Size: Short, AddOns: [Mocha]   Have a nice day! |