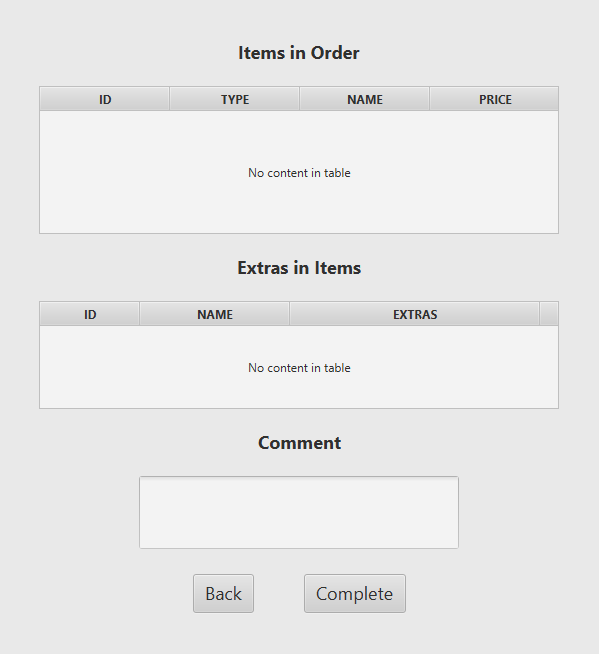
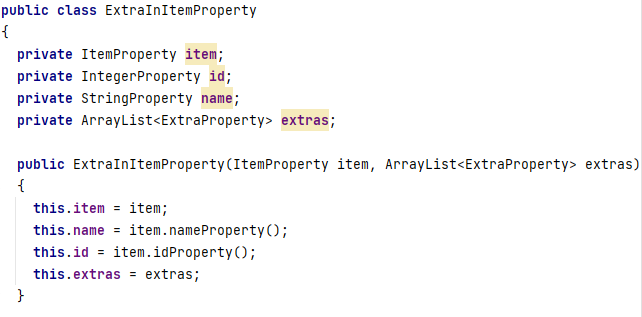
*Extras in Barista View - design*

The essential concern for the barista is to know if the customer put any extras for his items. To fulfill this case, the OrderDetailsView has to be modified accordingly. New TableView is introduced containing three columns - id and name of the item, and chosen extras. This provides necessary information for the person responsible for preparation of the order. 

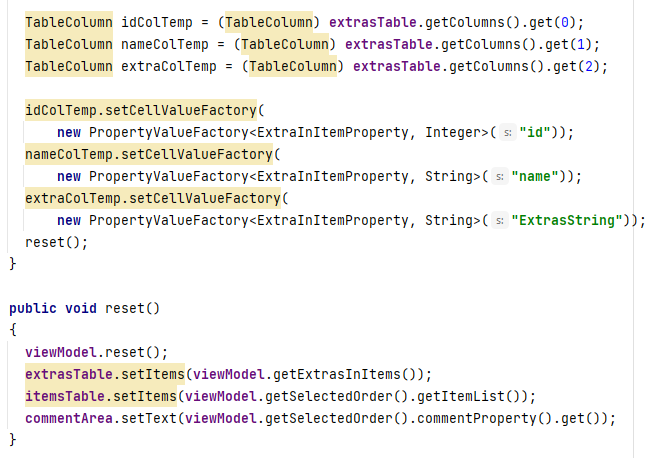
*Extras in Barista View - implementation*

To display the correct extras for the specific item, those two properties had to be combined. As a consequence, the ExtraInItemProperty class was introduced.



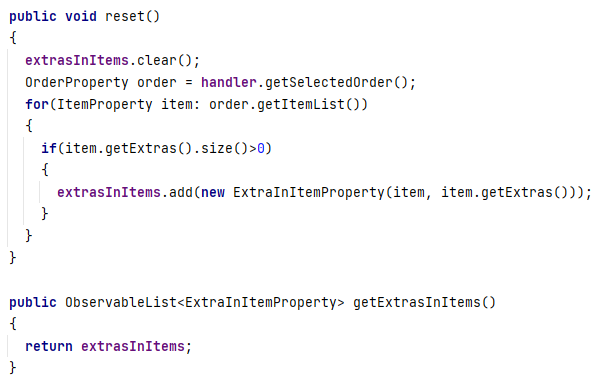
Inside the OrderDetailViewController the columns are assigned proper values and the reset method is invoked. The reset method calls the method with the same name from ViewModel and after that the items in the table are set using the ObservableList of type ExtraInItemProperty.

*OrderDetailViewController - init and reset methods*



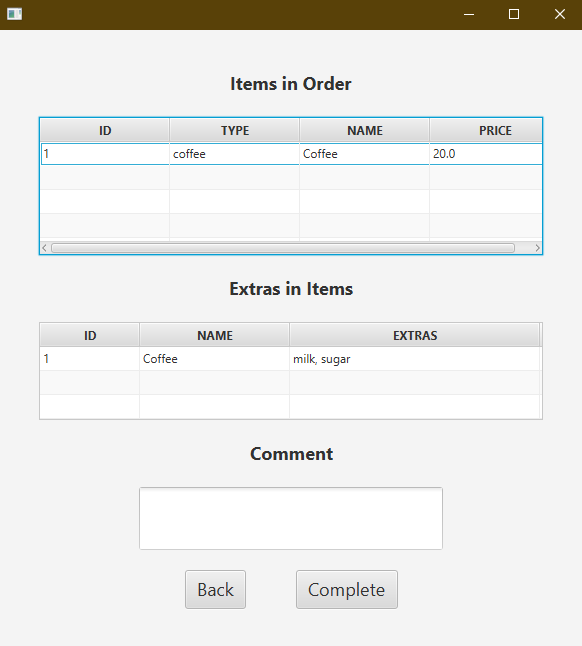
First of all, the list is cleared to avoid duplicates. The next step is to loop through the ItemList and fill the extrasInItems with the items and their extras. The method getExtrasInItems simply returns this variable.

*OrderDetailViewModel - reset method*



To sum up this part, the window displaying the extras for the item is shown.

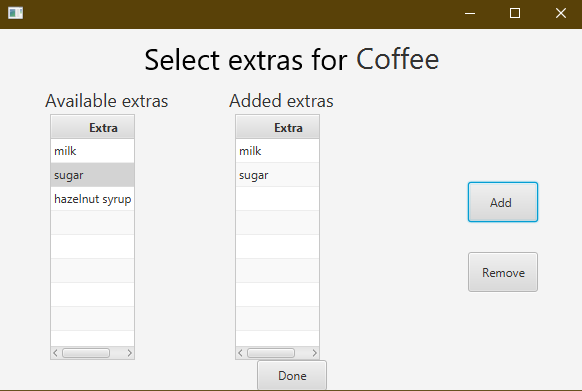
*OrderDetailView*



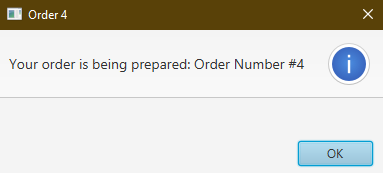
*Extras in Barista View - testing*

Integration testing was used to test the correctness of display of extras for a specific item in the OrderDetailView. To start the process the customer adds the desired extras for the item and pays for the order. This is crucial to test whether the extras added by the customer are accurately reflected in the OrderDetailView.

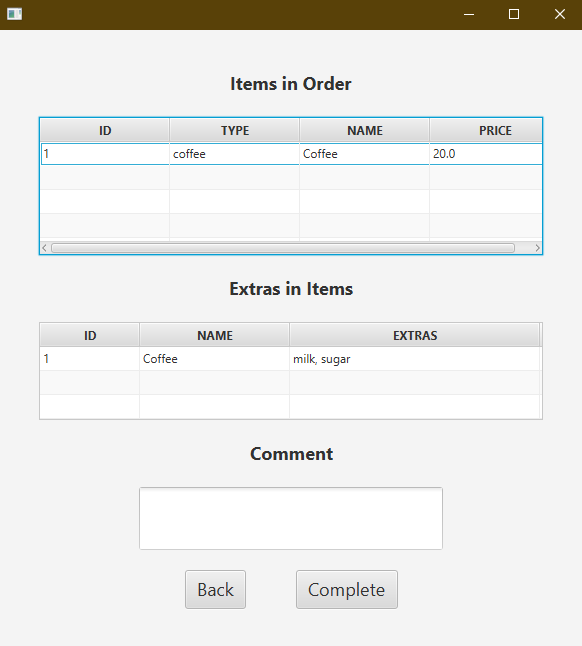
*ExtraView*



*Alert with the order number.*



Extras are correctly fetched from the database further down to the Model and eventually displayed in the OrderDetailView. This use case has passed the integration test.

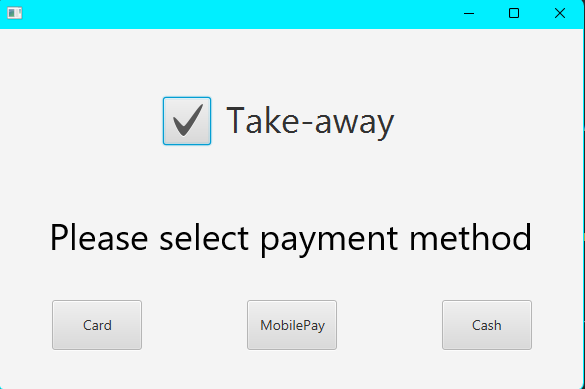


PAYMENT VIEW

# Design

The PaymentView was designed to satisfy the customers wishes according to the Product Backlog items 13 and 18. The first wants to give the customers the ability of choosing the payment method they would like to proceed with three choices: Card, MobilePay or Cash. The latter is for the customers to be able to choose whether they want their order to be prepared for on-site consumption or take-away.

*Paymentview*



For this purpose, a checkbox was used. For selecting the payment method, the customer can click on one of the three buttons, each representing a different payment method.

# Implementation

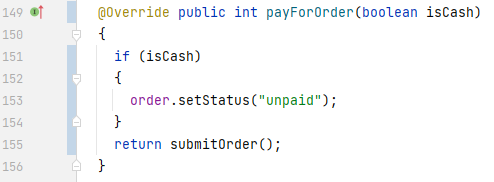
Payments done through Card and MobilePay are implemented in the same way: an if statement is checking the selectedProperty of the checkbox, and setting the order as Take-away if it is true. This feature was added in the final parts of the construction phase, therefore it was solved by adding “TAKE-AWAY” to the description of the order.

*Methods in the PaymentViewController class*



The methods from the viewmodel are calling the method payForOrder(boolean isCash). The parameter is set to true inside payWithCash() and to false inside (payWithCardOrMP).

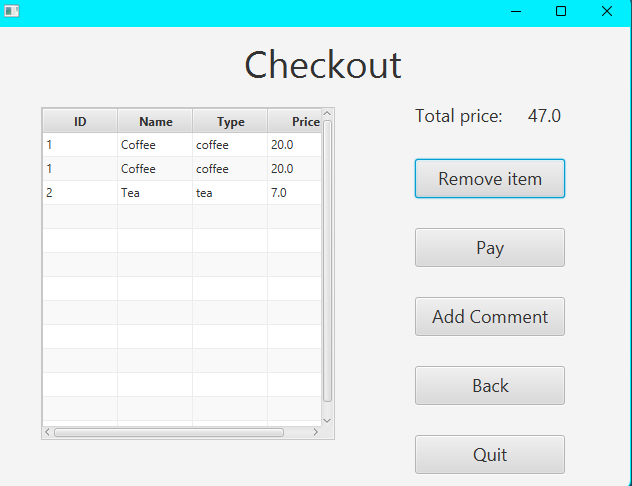
*payForOrder(boolean isCash) inside the ModelManager*

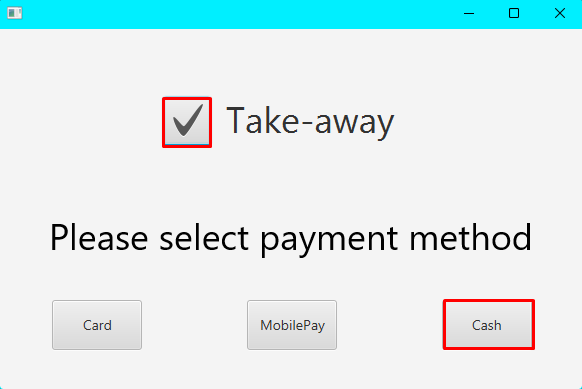


If the order is to be paid in cash, the status is set to unpaid and it goes to the cashier, otherwise the order is submitted.

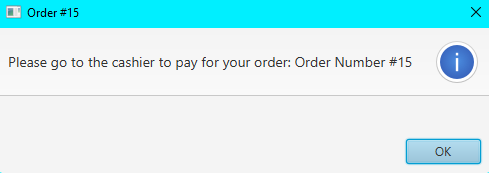
# Integration Testing

*The customer is adding three items to an order and clicking Pay*

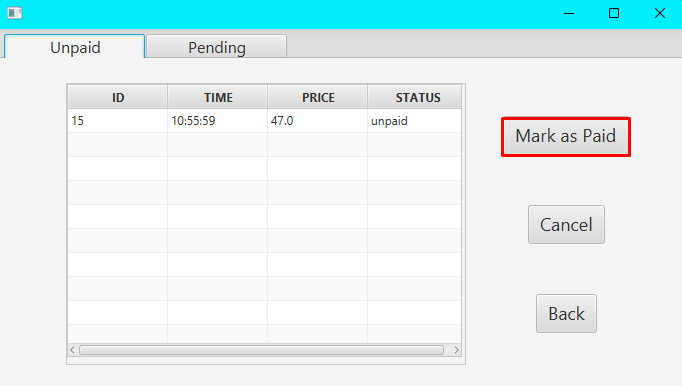


*The customer is selecting Take-away, and Cash payment*

*An alert is displayed for the customer with order ID #15*



*The cashier marks order #15 as paid after the customer handed him the cash*



*The barista receives the pending order #15. “TAKE-AWAY” is added to the description*



*Order #15 is in the database with the description*

