

Testing ConcreteItemDAO

The intro to this chapter was written by Laura when testing the OrderDAO.

createItem(Item item)

- Should never throw anything, insert the Item object into the database.
- It will never be called with a null object therefore it is unnecessary to check it.

JUnit test method

```
@Test void createItem()
{
    Item item = new Item( name: "Black tea", type: "coffee", price: 15, description: "Earl gray made in China.");
    assertDoesNotThrow(() -> ConcreteItemDAO.getInstance().createItem(item));
}
```

JUnit output

```
✓ Tests passed: 1 of 1 test – 1 sec 295 ms
"C:\Program Files\Java\jdk-11.0.2\bin\java.exe" ...
The adding of an item is completed.
```

The changes in the database

	item_id	name	type	price	description
1	1	Coffee	coffee	20	:)
2	2	Tea	tea	7	chilly tea
3	3	Irish	coffee	35	Coffee with rum
4	4	Mint tea	tea	15	Green tea with a dried mint.
5	5	Black tea	coffee	15	Earl gray made in China.

Documentation of AddItemViewController

Each view requires a ViewController so that it can be connected with the functionality of the system. All of the specific ViewControllers extend the superclass ViewController, which is simply responsible for initializing ViewHandler, ViewModelFactory and root. There are also getters for all of the fields. The important part of this abstract class is that it has a *protected abstract void init()* method. This method is essential for the specific ViewController as this is where all the necessary initializations are occurring.

ViewController superclass

```
6 public abstract class ViewController
7 {
8     private Region root;
9     private ViewHandler viewHandler;
10    private ViewModelFactory viewModelFactory;
11
12    protected abstract void init();
13
14    public void init(ViewHandler viewHandler, ViewModelFactory viewModelFactory,
15                    Region root)
16    {
17        this.root=root;
18        this.viewHandler=viewHandler;
19        this.viewModelFactory=viewModelFactory;
20        init();
21    }
22
23    public void reset()
24    {
25        //To be overridden?
26    }
27
28    public Region getRoot() { return root; }
32
33    public ViewModelFactory getViewModelFactory() { return viewModelFactory; }
37
38    public ViewHandler getViewHandler() { return viewHandler; }
42 }
```

Fields and init method

```
12 public class AddItemViewController extends ViewController
13 {
14     @FXML private TextField nameField;
15     @FXML private ChoiceBox<String> typeChoiceBox;
16     @FXML private TextField priceField;
17     @FXML private TextArea descriptionArea;
18     @FXML private Label errorLabel;
19     private AddItemViewModel addItemViewModel;
20
21     @Override protected void init()
22     {
23         addItemViewModel = getViewModelFactory().getAddItemViewModel();
24         nameField.textProperty().bindBidirectional(addItemViewModel.nameProperty());
25         priceField.textProperty()
26             .bindBidirectional(addItemViewModel.priceProperty());
27         descriptionArea.textProperty()
28             .bindBidirectional(addItemViewModel.descriptionProperty());
29         errorLabel.textProperty()
30             .bindBidirectional(addItemViewModel.errorProperty());
31         typeChoiceBox.getItems().addAll(addItemViewModel.getTypes());
32         //Putting items inside the choice box
33         typeChoiceBox.valueProperty()
34             .bindBidirectional(addItemViewModel.chosenProperty());
35         //Binding the chosen item with the view model
36     }
```

The mentioned *init()* method initializes the fields of the class and binds them with the *StringProperty* inside the *AddItemViewModel* class. This *ViewModel*'s value is assigned by calling a *getViewModelFactory()* from the superclass. After this the *AddItemViewModel* is retrieved by a simple getter. Line 31 is also worth mentioning as this is the way to fill the choice box with the determined values. This is done by getting the *ArrayList* of types from *AddItemViewModel*. Moreover, method *valueProperty()* is used to obtain the value chosen by the user from the choice box. Subsequently, it is bound with the corresponding *StringProperty* inside the *AddItemViewModel*.

OnAction methods

```
38  @FXML private void submitButton()  
39  {  
40      addItemViewModel.submit();  
41  }  
42  
43  @FXML private void backButton()  
44  {  
45      getViewHandler().openView(id: "DatabaseView.fxml");  
46      addItemViewModel.clear();  
47  }
```

The method *submitButton()* is delegating the work to the *AddItemViewModel* and all the information inside the *ViewModel* is fetched from the bound values. The method *backButton()* solely opens the previous View and calls a *clear* method inside the *AddItemViewModel*.

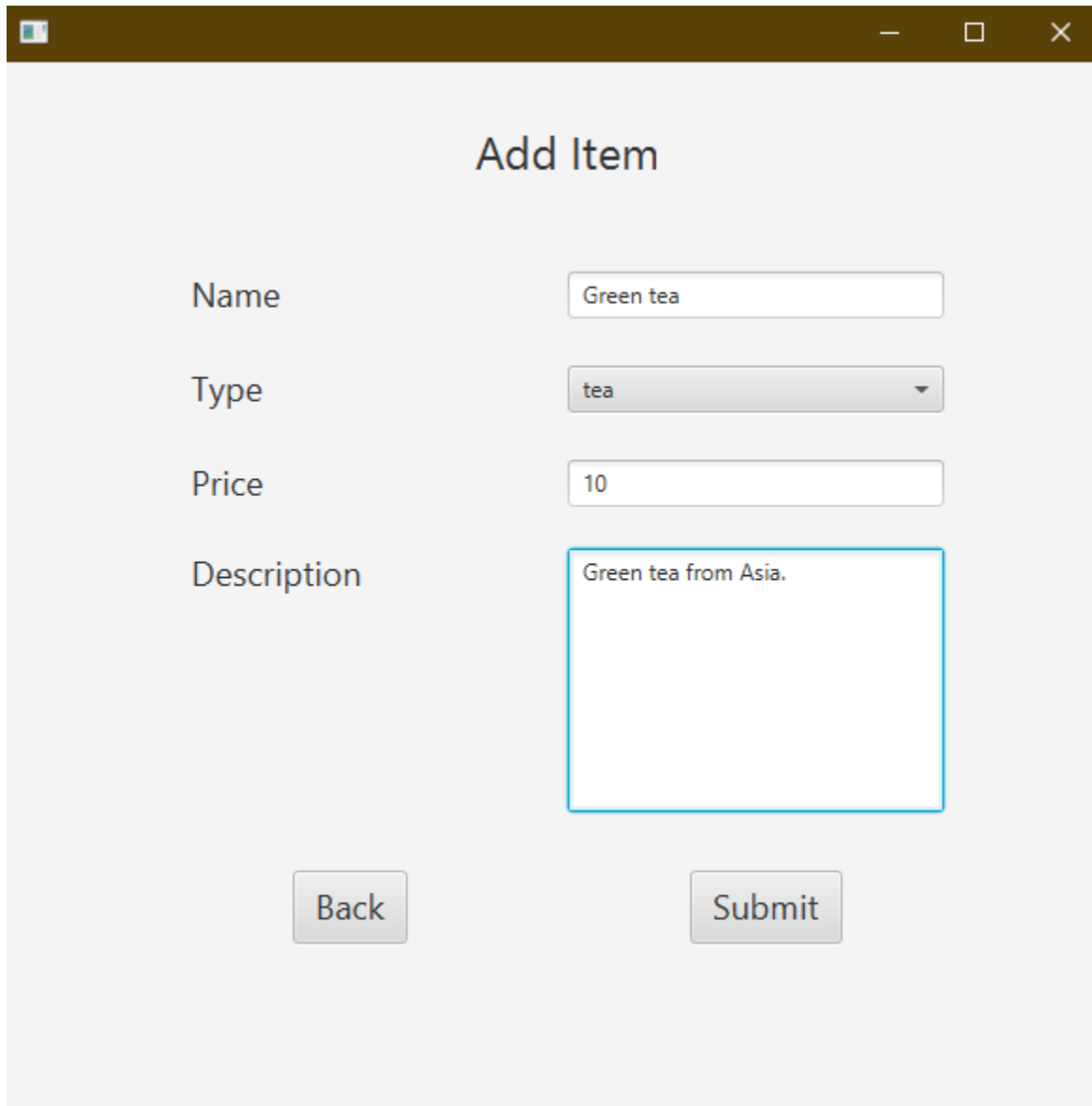
StartView, LoginView, DatabaseView Documentation

The initial view presented to the users is the start view. Two choices are granted: the first one is to start ordering; this would open the *CustomerView* where the items can be added to the order, while the second option is to log in: this would open the *LoginView* where the employees can enter their access key in order to be granted permission to see the *DatabaseView*. In this version they can select *Add Item*, which will take them to the *AddItemView* where they can add new items to the database.

Documentation of AddItem integration testing

For the purpose of testing the Add Item case the integration test was organized. This will test if the information is being transferred correctly from the View up to the database going through ViewModel, Model, RMI connection and the Database Persistence.

AddItemView



The screenshot shows a JavaFX window titled "Add Item". It contains four input fields: "Name" with the text "Green tea", "Type" with a dropdown menu showing "tea", "Price" with the text "10", and "Description" with a text area containing "Green tea from Asia.". At the bottom of the window are two buttons: "Back" and "Submit".

Console output on the Client side

```
"C:\Program Files\Java\jdk-11.0.2\bin\java.exe" ...  
maj 10, 2022 9:51:38 AM com.sun.javafx.application.PlatformImpl startup  
WARNING: Unsupported JavaFX configuration: classes were loaded from 'unnamed module @43f3ec2f'  
Item{id=-1, name='Green tea', type='tea', price=10.0, description='Green tea from Asia.', extras=[]}  
ModelManager: Item sent to the mediator.
```

As it can be seen, the Item object is created properly from the given values and then sent to the mediator package which is tested by the output in the console.

Console output on the Server side

```
"C:\Program Files\Java\jdk-11.0.2\bin\java.exe" ...
```

```
Registry started...
```

```
Server started...
```

```
The adding of an item is completed.
```

The item in the database

	item_id	name	type	price	description
1	1	Coffee	coffee	20	:)
2	2	Tea	tea	7	chilly tea
3	3	Irish	coffee	35	Coffee with rum
4	4	Mint tea	tea	15	Green tea with a dried mint.
5	5	Black tea	coffee	15	Earl gray made in China.
6	6	Green tea	tea	10	Green tea from Asia.

Adding of an item is completed without any errors which is checked by the proper console output and the existence of the specific item in the database.