Grazitti Academy

Fooddeck

Java Specification Document

Version 1.0

	Prepared By / Last Updated By
Name	Dr. Aswin Kumar. S
Role	Corporate Manager – Academy
Signature	
Date	12.02.2021

Table of Contents

1.0 Introduction

1.1 Purpose of this document

The purpose of this document is to define the Java class related implementation for Fooddeck project.

1.2 Definitions & Acronyms

Definition / Acronym	Description
N/A	**N/A**

1.3 Project Overview

Refer Fooddeck-use-case-specification.docx for understanding the functionality and features,and refer the Fooddeck-Class_diagrams.docx for class diagrams

1.4 Scope

Creation of model and data access object classes for Fooddeck application

1.5 Intended Audience

Product Owner

Scrum Master

Application Architect

Project Manager

Test Manager

Development Team

Testing Team

1.6 Hardware and Software Requirement

1. Hardware Requirement:
    a. Developer PC/Laptop with 4GB Ram

2. Software Requirement

    a. JDK 1.8

    b. Eclipse IDE for Enterprise Java Developers 2019-03 R

2.0Class Diagram

The classes specified in this document are the primary Java classes that are required for implementation of Fooddeck application. **Though JDBC module is covered, the actual database implementation details and role based functionalities are postponed to the respective module**. The classes in this specification are implemented with hardcoded values and will be consumed by the Servlets when implementing the next module.

2.1 Model package

Following are the real world objects identified for Fooddeck application. Menu Item refers to a menu item available for sale in Fooddeck. Cart will represent customer's cart to hold the selected menu items. Refer the diagram in the attached file and create classes accordingly.

Guidelines for understanding the class diagram: - ref the class diagram file attached.

1. "com.grazitti.Fooddeck.model" represents the package
2. MenuItem and Cart are classes
3. The content within MenuItem are instance variables
4. The hypen in each line represents private access specifier
5. For the sake of simplicity the constructors, getter and setter method are not included in the diagram. But it needs to be implemented in code. Code generation option in Eclipse can be used to generate code:
    a. Constructor with option to set all instance variables
    b. Getter and Setter method for each instance variable
    c. Generate toString() method
    d. Generate equals() method which checks for equality based on the 'id' attribute

2.2 Util Package

Common reusable classes and methods across Fooddeck application will be included in this package.

Guidelines for understanding the class diagram- ref : attached file:

1. "com.grazitti.Fooddeck.util" represents the package

2. DateUtil is a class
3. Underline denotes static method.

2.2.1 DateUtil.java

convertToDate(date: String): Date

This method is used to convert date entered in a form to be converted into a Date object.

1. Using SimpleDateFormat and parse() method convert the input String in 'dd/MM/yyyy' format into java.util.Date type.

2.3 Dao package

This package contains the list of classes that will code to manage the data for Fooddeck application. The methods in Dao classes will be tested using MenuItemDaoCollectionImplTest and CartDaoCollectionImplTest classes. The Dao interface classes will act as a contract for working with any database. In this specification the implementation of MenuItemDaoCollectionImpl and CartDaoCollectionImpl will be Collection framework based implementation of Dao interfaces MenuItemDao and CartDao.

Guidelines for understanding the class diagram - attached:

1. Identify the package, classes, access modifiers, methods and static methods from the above diagram.
2. MenuItemDao and CartDao are interfaces
3. MenuItemDaoCollectionImpl and CartDaoCollectionImpl are implementation classes for the interfaces as denoted by the dotted arrow line.
4. MenuItemDaoCollectionImplTest and CartDaoCollectionImplTest are implementation classes for testing MenuItemDaoCollectionImpl and CartDaoCollectionImpl.
5. **MenuItemDaoSqlImpl, CartDaoSqlImpl, MenuItemDaoSqlImplTest, CartDaoSqlImplTest classes will not be implemented in this module. Please ignore these classes for this module.**
6. CartEmptyException is an exception class that extends exception.
7. Highlighted classes will be implemented in this module.

3.0 Design for View Menu Item List Admin (TYUC001)

3.1 Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined below - **The diagram file attached**.

3.2 MenuItemDao.java

Add the method getMenuItemListAdmin(): List<MenuItem> in the interface.

3.3 MenuItemDaoCollectionImpl.java

Class for managing data of menu items using Java Collections Framework.

Constructor

The objective of this constructor is to initialize the menu item data that will be displayed in MenuItem listing screen of Admin.

1. Check if menuItemList static variable is null or not
2. If it is null perform the steps below:
    a. Create an instance of ArrayList with MenuItem type
    b. Create multiple MenuItem instances and add them to menuItemList. Refer Menu Item List Admin screen shot from web interface specification and include sample data for menuItemList based on this sample data.

getMenuItemListAdmin(): List<MenuItem>

This method returns the list of menu items that will be displayed in the MenuItem listing screen for Admin.

1. Return the menuItemList

3.4 MenuItemDaoCollectionImplTest.java

main(args[]: String): void

1. Invoke testGetMenuItemListAdmin()

testGetMenuItemListAdmin(): void

1. Instantiate MenuItemDaoCollectionImpl and assign it MenuItemDao reference variable menuItemDao.

2. Invoke menuItemDao.getMenuItemListAdmin() and obtain the menuItemList

3. Iterate through the menuItemList and display all attributes of each menu item.

4.0 Design for View Menu Item List Customer (TYUC002)

4.1 Class Diagram

Refer attached file,

4.2 MenuItemDao.java

Add the method getMenuItemListCustomer(): List<MenuItem> in the interface.

4.3 MenuItemDaoCollectionImpl.java

This class manages the data related to Menu Items of Fooddeck application. A new method needs to be added for this use case.

getMenuItemListCustomer(): List<MenuItem>

This method returns the list of menu items that will be displayed in the Menu Item listing screen for Customer.

1. Initialize an ArrayList for type MenuItem

2. Iterate through menuItemList and perform the following steps:

    a. Is the launch date of the menu item is today or before today?

    b. Is the menu item available is active?

    c. If the above conditions satisfy, add the menu item into the ArrayList created in the first step.

3. Return the filtered ArrayList

4.4 MenuItemDaoCollectionImplTest.java

main(args[]: String): void

1. Invoke testGetMenuItemListCustomer()

testGetMenuItemListCustomer(): void

1. dInstantiate MenuItemDaoCollectionImpl and assign it MenuItemDao reference variable menuItemDao.

2. Invoke menuItemDao.getMenuItemListCustomer() and obtain the menuItemList

3. Iterate through the menuItemList and display all attributes of each menu item.

5.0 Design for Edit Menu Item (TYUC003)

5.1 Class Diagram

The dilagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined the diagram given in the attached file

5.2 MenuItemDao.java

1. Add method modifyMenuItem(menuItem: MenuItem): void in the interface.

2. Add method getMenuItem(menuItemId: long): MenuItem in the interface.

5.3 MenuItemDaoCollectionImpl.java

This class manages the data related to Menu Items of Fooddeck application. A new method needs to be added for this use case.

modifyMenuItem(menuItem: MenuItem): void

This method will be used to change the menu item data in the list of menu items. This method will be invoked when Customer submits the user form.

1. Iterate through the menuItemList and find the matching menu item

2. Update the matching menuItem in the ArrayList

getMenuItem(menuItemId: long): MenuItem

This method is used to retrieve a particular menu item's detail from the menu item list. This method will be invoked when user click on Edit link in menu item listing screen of Admin.

1. Iterate through menuItemList and find the matching menu item

2. Return the matching menuItem from the menuItemList

5.4 MenuItemDaoCollectionImplTest.java

main(args[]: String): void

1. Invoke testModifyMenuItem()

testModifyMenuItem(): void

1. Create an instance for Menu Item with id matching with one of the menu

item already added to the menuItemList.

2. Instantiate MenuItemDaoCollectionImpl and assign it MenuItemDao reference variable menuItemDao.

3. Invoke MenuItemDao.modifyMenuItem(menuItem) by passing the menu item created in the first step.

4. Invoke menuItemDao.getMenuItem(productId) to read and check if the menu item details are modified.

6.0 Design for Add to Cart (TYUC004)

6.1 Class Diagram

The diagram (attached) denotes the methods that needs to be implemented for this use case. Method wise specification is defined below.

6.2 CartDao.java

1. Add method addCartItem(userId: long, menuItemId: long): void in the interface.

6.3 CartDaoCollectionImpl.java

This class manages the data related to Cart of all users of Fooddeck application. A new method needs to be added for this use case.

Constructor CartDaoCollectionImpl()

Data for all users will be stored in the HashMap available in Cart instance. This constructor initialized the Cart as well as the HashMap within the Cart, so that the class instance is ready to store values in the HashMap when Customer adds items into the Cart.

1. Check if the userCarts instance variable is null or not

2. If userCarts is null then create a new instance of HasMap with type Long and Cart and assign it to userCarts instance variable.

3. The userCarts instance variable will hold the cart details for each user in a HashMap. The key of this HashMap will have the userId. Each value in the HashMap will be an ArrayList of MenuItem.

addCartItem(userId: long, menuItemId: long): void

This method is invoked when Customer clicks Add to Cart link in menu item listing screen. This method gets the menu item list from the HashMap for the specific user and adds the menu item into the menu item list. If there is no such user in the HashMap, then a new entry needs to be added in the HashMap with userId as key and new ArrayList of Menu Items as value.

1. Instantiate MenuItemDaoCollectionImpl and assign it MenuItemDao reference variable menuItemDao.

2. Get the menuItem using menuItemDao.getMenuItem(menuItemId) method

3. Check existence of user in userCarts based on userId

4. If user exists in userCarts, perform the steps below:

a. Get the menuItemList from the userCarts

b. Add the menuItem obtained in previous step into menuItemList

5. If user does not exist in userCarts, perform the steps below:

a. Create a new Cart instance with new ArrayList

b. Add the menu item obtained in step one and add it to menuItemList created in previous step

c. Put the userId and ArrayList of MenuItem into the userCarts

7.0 Design for View Cart (TYUC005)

7.1 Class Diagram

The diagram attached denotes the methods that needs to be implemented for this use case. Method wise specification is defined below.

7.2 CartDao.java

1. Add method getAllCartItems(userId: long): List<MenuItem> in the interface.

7.3 CartEmptyException.java

1. Extend this class from java.lang.Exception and include an empty constructor.

7.4 CartDaoCollectionImpl.java

This class manages the data related to Cart of all users of Fooddeck application. A new method needs to be added for this use case.

getAllCartItems(userId: long): Cart throws CartEmptyException

Method to get list of menu items added by a customer to Cart.

1. Get the menuItemList based on userId from the HashMap of userCarts
2. If the returned list is empty
    a. Create new CartEmptyException and throw it
3. If the returned list is not empty
    a. Iterate through the menuItemList and add up the prices.
    b. Set the total instance variable of cart with the added up menu item prices.
    c. return cart

7.5 CartDaoCollectionImplTest.java

main(args[]: String): void

1. Invoke testAddCartItem()

testAddCartItem(): void

1. Instantiate CartDaoCollectionImpl and assign it to CartDao reference variable cartDao.

2. Invoke cartDao.addCartItem() method with following parameters

    a. userId: 1

    b. menuItemId: one of existing menuItemId in MenuItemDaoCollectionImpl

3. Invoke cartDao.getAllCartItems() with userId as 1

4. Display the contents of MenuItemList returned in previous step and check if the added cart item is present or not.

testGetAllCartItems(): void

5. Instantiate CartDaoCollectionImpl and assign it to CartDao reference variable cartDao.

6. Invoke cartDao.getAllCartItems() method passing the userId as 1 and display the resulting list of menu items.

8.0 Design for Remove Cart Item (TYUC006)

8.1 Class Diagram

The attached diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined below.

8.2 CartDao.java

1. Add method removeCartItem(userId: long, menuItemId: long): void in the interface.

8.3 CartDaoCollectionImpl.java

This class manages the data related to Cart of all users of Fooddeck application. A new method needs to be added for this use case.

removeCartItem(userId: long, menuItemId: long): void

Method to remove a menu item from the cart. This will be invoked when Customer clicks Delete link in the Cart screen.

1. Get the List<MenuItem> from userCarts based on userId
2. Iterate through the List of MenuItem and perform the below steps
    a. Check if the menuItemId of each menu item from the list matches with this methods input parameter
    b. If menuItemId matches then remove the menu item from the list

8.4 CartDaoCollectionImplTest.java

main(args[]: String): void

1. Invoke testRemoveCartItem()

testRemoveCartItem(): void

1. Instantiate CartDaoCollectionImpl and assign it CartDao reference variable cartDao.
2. Invoke cartDao.removeCartItem() method with following parameters
    a. userId: 1
    b. menuItemId: Same menuItemId as what was provided when testing add

cart item.

3. Invoke cartDao.getAllCartItems() with userId as 1

4. Enclose the above method within try catch block with catch block handling CartEmptyException. Check if the catch block is executed, which means that the cart item added during testAddCartItem() is removed now and the cart is empty, due to which the CartEmptyException is thrown.

9.0 Standards and Guidelines

9.1 Java

1. Ensure that the class names, method names and variable names are followed exactly as specified in the class diagram
2. Ensure that access modifier are in line with the class diagram specification
3. Naming standards to be followed:
   a. Variable
      i. Should be in mixed case with the first letter lowercase and with the first letter of each internal word capitalized (Example: firstName, dateOfBirth)
      ii. Variable names should be short, but meaningful
      iii. Variable name defined should indicate the purpose to a casual observer
      iv. Single character variable names should be avoided except for temporary variables
      v. Temporary variables include i, j, k and m
   b. Class
      i. Class name should be a noun
      ii. Class name should be in mixed case with the first letter uppercase and with the first letter of each internal word capitalized
      iii. Must use whole words and should not have acronyms or abbreviations

      Examples: Employee, TaxCalculator
   c. Method
      i. Method names should be verbs
      ii. Method names should be in mixed case with the first letter lowercase and with the first letter of each internal word capitalized

      Example: changeGear(), calculateBalance()

4. Code Formatting
    a. Class Structure
        i. Place the elements of a class in the following order:
            1. Static variables
            2. Instance variables
            3. Constructors
            4. Methods and Getter/Setters
            5. hashCode(), equals(), toString,
    b. Spacing
        i. A space before and after an operator is required
        ii. A space before curly braces is required
        iii. A space after a comma is required
        iv. A space after semicolon in for loop is required
        v. A single line space after a method is required
    c. Curly braces position
        i. Opening curly braces should be in the same line
        ii. Closing curly braces should always be in a new line
    d. Tab spacing
        i. Use 4 spaces instead of tab character
        ii. Increase a tab character in the lines after opening curly braces
        iii. Reduce a tab character on the of closing curly braces
        iv. Include one more tab in the wrapped line
    e. Line Width
        i. Width of a line should not exceed 100 characters

10.0 Change Log

<table>
<tr><th></th><th colspan="4">Changes Made</th></tr>
<tr><td>V1.0.0</td><td colspan="4">Initial baseline created on 12-02-2021 by Dr. Aswin Kumar. S</td></tr>
<tr><td rowspan="3">Vx.y.z</td><td colspan="4">Please refer the configuration control tool / change item status form if the details of changes are maintained separately. If not, the template given below needs to be followed – Not Applicable</td></tr>
<tr><td>Section No.</td><td>Changed By</td><td>Effective Date</td><td>Changes Effected</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
</table>