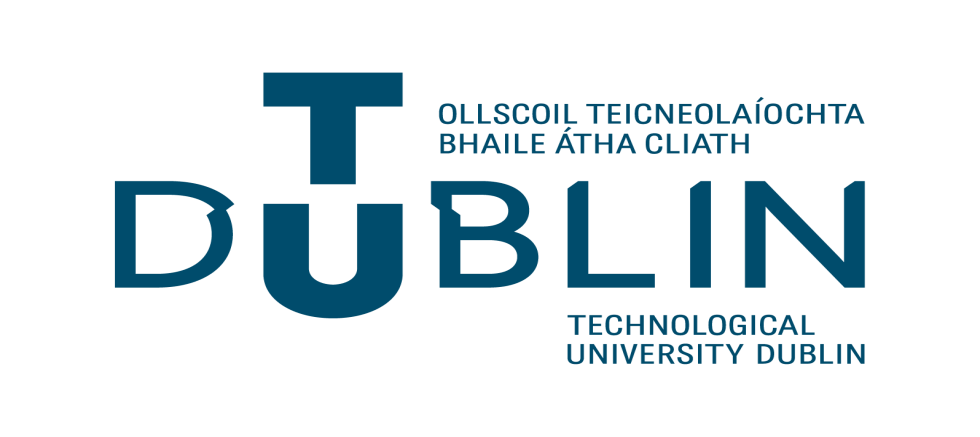
**

Robert Leniston

B00158877

Department of Informatics

School of Informatics and Engineering

TU Dublin Blanchardstown Campus

Dublin 15

Higher Certificate in Science in Computer Systems Management.

Software development

14/12/2024

**Declaration**

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, except where otherwise stated. I/We have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.

I/We understand that plagiarism, collusion, and copying are grave and serious offences and accept the penalties that would be imposed should I/we engage in plagiarism, collusion or copying. I acknowledge that copying someone else’s assignment, or part of it, is wrong, and that submitting identical work to others constitutes a form of plagiarism. I/We have read and understood the colleges plagiarism policy 3AS08 (available [here](https://www.itb.ie/AboutITB/QualityAssurancePolicies/3AS08%20Institute%20Procedural%20Guidelines%20for%20Dealing%20with%20Instances%20of%20Plagiarism%20in%20Assignments%20and%20Examinations%20%2013%20June%202014.pdf)).

This material, or any part of it, has not been previously submitted for assessment for an academic purpose at this or any other academic institution.

I have not allowed anyone to copy my work with the intention of passing it off as their own work.

Name: Robert Leniston Dated:14/12/2024

Contents

[Overview 4](#_Toc183451286)

[Application logic 4](#_Toc183451287)

[Application Components 4](#_Toc183451288)

[Main class 4](#_Toc183451289)

[DatabaseManager.Java 4](#_Toc183451290)

[Constructor – coffee.java 5](#_Toc183451291)

[Getters / Setters 5](#_Toc183451292)

[getAllCoffees 6](#_Toc183451293)

[addCoffee 6](#_Toc183451294)

[removeCoffeeById 7](#_Toc183451295)

[updateStockQuantityById 7](#_Toc183451296)

[searchCoffeeById 7](#_Toc183451297)

[Coffee.java 8](#_Toc183451298)

[Testing Procedure 9](#_Toc183451299)

[View Coffee 9](#_Toc183451300)

[Add Coffee 9](#_Toc183451301)

[Remove coffee by ID 9](#_Toc183451302)

[Update Stock Quantity by ID 9](#_Toc183451303)

[Search Coffee By ID 9](#_Toc183451304)

[Defensive checks 10](#_Toc183451305)

[Application Logic checks 10](#_Toc183451306)

[Invalid ID Tests: 10](#_Toc183451307)

[Empty Brand Tests: 10](#_Toc183451308)

[Price Tests: 10](#_Toc183451309)

[Quantity Tests: 10](#_Toc183451310)

[GUI Level checks 11](#_Toc183451311)

[Search Button 11](#_Toc183451312)

[Update Button 11](#_Toc183451313)

[Add Button 11](#_Toc183451314)

[Remove Button 11](#_Toc183451315)

[References 11](#_Toc183451316)

# Overview

The purpose of the application that I have designed is to allow the manager of a coffee shop to perform various database operations. These operations are to view, add, update, delete coffee records though a java app that connects to an SQL database.

# Application logic

Upon starting the application, the main method in CoffeeShopApp initialized the database connection by reading connection details from the db.properties file. This connection is passed to the GUI constructor.

The coffeeshopapp class creates a GUI instance and displays input fields for Coffee ID, Brand, Type, Price Per KG and quantity along with other buttons for adding, updating, viewing, removing and searching. Each button click on the GUI triggers an event in the form of an action listener. For example, when the users click the “Add Coffee” button, the ActionListener retrieces values from the input field and create a coffee object, the addCoffee method is called to inset the new record into the SQL Database.

# Application Components

## Main class

public class coffeeshopapp {

public static void main(String[] args) {

The coffeeshopapp class is the main entry point of the app. This manages the database connection using the db.properties file. It ensured the database connection is successfully established before any further operations are performed.

## DatabaseManager.Java

The DatabaseManager class is maintains a connection instance variable and handles the connection, it uses several components.

Connection variables – URL, User & password variables store the database connection details

The class has members such as URL, user, password, these are variables used to store the URL, password and user account which are pulled from the db.properties file.

A BufferedReader is used in conjunction with a try/catch method to read the db.properties file and handle any potential errors that might occur   
  
BufferedReader is used to read the db.properties file where the information used to connect to the SQL database exists. an IOException is used to catch any errors with reading the db.properties folder and SQLException is used to handle connection errors, such as incorrect login details.

catch (IOException e) {

System.***err***.println("Error reading properties file: " + e.getMessage());

return;

}

try (Connection conn = DriverManager.*getConnection*(url, user, password)) {

System.***out***.println("Connected Successfully!");

} catch (SQLException e) {

System.***err***.println("Error connecting to database: " + e.getMessage());

}

Once the database connection information is loaded the app uses DriverManager.*getConnection*  (Oracle, n.d.) to establish the connection.

## Constructor – coffee.java

The coffee.java class uses multiple variables to store data that is pulled from the SQL database. The purpose of this class is to pull data and store it in variables, the data can be modified and then entered back into the SQL database.

When we retrieve data from the SQL and store it in the variables, we will use this constructor to create instances of the coffee class for each record we pull from the database.

public coffee(int id, String brand, String type, double pricePerKg, double quantityInStock) {

this.id = id;

this.brand = brand;

this.type = type;

this.pricePerKg = pricePerKg;

this.quantityInStock = quantityInStock;

}

## Getters / Setters

We use getters and setters for interactive with private fields like Id, Brand, Type, this is an integral part of data encapsulation. The getters are methods that retrieve values of private fields I the class. These fields are private and cannot be accessed from outside the class.

public int getId() {

return id;

}

Setters are used to modify the value of the private fields. These setters enable us to change the value of a field outside the class while also keeping the feel private. For example, if we update the quantityInStock we use the setter method to modify that value. The naming convention of getX / setX was chosen as it is best practice and ensures consistency and readability. (Hamilton, n.d.)

## getAllCoffees

This instance method is responsible for pulling all of the coffee records from the coffee table in the SQL database. It will create a coffee object for each record and store it in a list. The name for the class getAllCoffee was chosen as it simply reflects the purpose of the method

Connection conn is an active database connection, this enables the application to execute SQL queries.

A prepared statement (“PreparedStatement (Java Platform SE 8 ),” n.d.) is used to execute the SQL query SELECT \* FROM coffee, this will retrieve all columns from the table, the ResultSet object is then used to hold the results of the query. For every row in ResultSet a new coffee object is create and populated with values from the corresponding columns.

coffee coffeeRecord = new coffee(

rs.getInt("ID"),

rs.getString("brand"),

rs.getString("type"),

rs.getDouble("pricePerKg"),

rs.getDouble("quantityInStock")

);

The coffee objects are then added to list<coffee> to store the records retrieved from the database. Once rows have been processed the method will return the completed list of coffee objects.

A try/catch block is used to retrieve all coffee records from the database. The value returned by the query are then stored in a ResultSet object. If there are issues with retrieving the data the code catches an SQLException error and prints a message informing the user “error retrieving coffee records”

## addCoffee

The addCoffee method is used to inset a new coffee record into the coffee table in the SQL database. Connection conn is used to connect to the database and execute the SQL query to inset the new coffee record.

The coffee newCoffee object contains the coffee details such as ID, brand, type, price Per Kg and quantity. These details will be added to the database.

The method uses an INSERT INTO SQL query (“Java JDBC Insert Example: How to insert data into a SQL table | alvinalexander.com,” n.d.) to insert new coffee records into the database. If there is an error during the insertion the method will catch an SQLEXCEPTION and prints an error message.

The method will also check for duplicate coffee ID’s, this is done by verifying if the ID exists in the database with the doesCoffeeExist method. If the ID exists in the database a warning message is displayed on screen informing the user of the duplicate value, the method will exit without updating the database. (“Swing Examples - Show Warning message Dialog,” n.d.)

## removeCoffeeById

The removeCoffeeById method is responsible for removing a coffee record from the SQL database based off its ID. Before deleting the record, it will do a check to ensure that the record actually exists and will then prompt the user for confirmation. The method first calls on doesCoffeeExist to verify that the record ID given exists. If the record does not exist, the method will give a pop-up message informing the user to that.

The JOptionPane.showconfirmDialog method is used to create a dialog box that prompts the user for confirmation before a record is deleted, which will assist in preventing accidental deletion (“Show Confirm Dialog with yes no buttons,” n.d.)

If an error occurs during the deletion of a record, the method will catch an SQLException and print the error message “Error deleting coffee record”

## updateStockQuantityById

The updateStockQuantityById method is used to update the stock quantity in the SQL database, records are updated based on the records ID. The method will ensure the record exists and that the quantity of coffee that is being entered is valid.

The method will first call the doesCoffeeExist to check if the coffee record is a valid ID within the database. If the record does not exist, then the method will display a pop-up message informing the user. The method will then return without making any changes to the database.

Before the stock quantity is updated, the method will check whether the newQuantity is greater than 0, ensuing invalid values are not entered. If the quantity is invalid, a pop-up message will appear and ask the user to enter a valid stock quantity that is greater than 0. (“Java Operators,” n.d.)

Once a record is updated, the executeUpdate method will run the SQL query which will update the stock quantity. If the update is successful, the method will then display a popup message that will inform the user that the quantity was updated successfully.

## searchCoffeeById

This method allows the user to search for a coffee in the database based on its ID. The method will prompt the user to enter the ID of the coffee they are searching for. It will validate the input and check If the coffee exists. If the record does not exist a message is displayed to the user. If the input is empty or invalid, for example, if the user enters a none numeric value, the method displays the error message using JOptionPane.showMessageDialog() (“Java JOptionPane - javatpoint,” n.d.) - (“Tutorial,” n.d.) (“Layout Managers - JavaBitsNotebook.com,” n.d.)

After the user enters a numeric ID the method doesCoffeeExist is called upon to verify the record in the database. The method passes the active Connection object (conn) and the coffee ID (coffeeId) to the doesCoffeeExist method. The doesCoffeeExist method prepares an SQL query to count the number of records that match the ID provided by the user. PreparedStatement is used to interest the ID into the query, the method executes the query and gets the result using ResultSet.

If the record is valid, the method executes an SQL query using the PreparedStatement function to retrieve the record.

## Coffee.java

The Coffee class servers as an enetity class that helps to represent a coffee product in the application. The class implements validation to ensure the integrity of the data before any specific object is created or changed.

The class defines constant messages used for the validation. These constants ensure there is consistent error messages throughout the application.

# Testing Procedure

Testing was broken up into 5 parts, testing of the View Coffee, Search Coffee, Update, Remove and Add.

## View Coffee

The expected result for this function should display all coffee in the database. To test, I would click the button and note the result, in my first test, the information in the database was populated into the text box in the application. I carried out further tests by updating the information in the database and testing the functionality once more. After an update of the information in the database, the View Coffee function would display the updated database information.

## Add Coffee

The expected result of the Add Coffee function would inset information entered into the fields of the application in the database.

To test, I started off by properly inserting the information, once done, I would add the information and view the database contents. Upon seeing the data was successfully entered I began testing with invalid data entries. Test were done by leaving fields blank and entering duplicate ID’s. The code would detect duplicate values and throw up an error message if one was detected.

## Remove coffee by ID

The expected result would be that the application would remove a Coffee from the database based off the ID entered. Testing was done by entering an ID that existed, once entered the information would be successfully removed from the database. Further testing was done by removing a non-existent ID and using non-numerical values in the ID field. If an non-existent ID is entered the app will ignore the value and not update the database.

## Update Stock Quantity by ID

The expected result of this method is to update stock information based off the ID used. This method was tested by firstly entering a new stock quantity and then updating the database. If successful, a message would appear on screen informing the user that the stock was successfully updated. If a negative value was entered, a message would appear on screen asking the user to enter a value that is greater than 0. If a non-numerical value was also entered, the user would get an error message prompting for numerical values only.

## Search Coffee By ID

The expected result of this method is to click the button, enter an ID when prompted and then view the information in the database pertaining to what was just searched. Testing was first done by clicking the button and entering a known valid ID. Once entered, the information would be pulled from the database and present on screen for the user. Further testing was carried out by entering an invalid ID. If an invalid ID was entered a message would inform the user that the ID does not exist

# Defensive checks

The defensive checks are broken up into two parts, GUI level checks and checks in the applications logic. The GUI level checks will check for things like empty fields, whereas the checks in the application logic will check for things like negative values in fields.

## Application Logic checks

Defensive checks are carried out in the Coffee.java class, these checks will ensure data is valid before being parsed to their respective variables.

If invalid data is entered, the app will throw an IllegalArgumentException error message (“IllegalArgumentException (Java Platform SE 8 ),” n.d.), “Power, A. (2024) Week 10” . Testing of these defensive checks was carried out by testing each scenario possible, these tests consisted of the following: Testing of invalid ID, empty brand, negative price and negative quantity

Each defensive check was tested with the following scenario.

### Invalid ID Tests:

* + Attempted to enter ID: -1
  + Result: Error message "Coffee ID must be positive"
  + Status: Check successful

### Empty Brand Tests:

* + Attempted to enter empty brand name when adding coffee
  + Result: Error message "Brand name cannot be empty"
  + Status: Check successful

### Price Tests:

* + Attempted to enter price: -20.0
  + Result: Error message "Price must be positive"
  + Status: Check successful

### Quantity Tests:

* + Attempted to enter quantity: -100
  + Result: Error message "Quantity cannot be negative"
  + Status: Check successful

## GUI Level checks

GUI level checks are implemented throughout the action listeners in the Coffeeshopapp.java class. Try Catch methods are used throughout the Coffeeshopapp.java class for error checking at a GUI level

### Search Button

Defensive checks in this method will check if the ID field is empty and if the ID field contains a valid number using input.trim().isEmpty()(“Java String trim() Method,” n.d.) . If a field is left empty the user will be prompted to enter a value in the field with a message appearing on screen saying “Please enter an ID”

Non numerical values are handled similarly, a the user receiving a message stating “Please enter a valid numeric ID”

Once the error messages have run, the application returns without performing a search.

### Update Button

The update button defensive checks will check if the ID can be converted to a number and if the Quantity can be converted to a number. Invalid data is handled by displaying error messages, for invalid ID’s, the user is given an error message “Coffee ID must be a numeric value” and for invalid quantity “Numerical values only for quantity”

After the error messages, the application returns to without performing an update in the database.

### Add Button

The checks performed in the Add Button will ensure the ID can be converted to an integer, Price can be converted to double and Quantity can be converted to double.

Invalid data is handled with error messages. If numberic conversions fail, the user is given the error message “Please enter valid numerical values”. After the error message, the coffee object is not created or added.

### Remove Button

The remove button checks that only basic numerical values are parsed for ID. coffeeIdField.gettext gets whatever data is in the ID field, interger.parseInt will then try to convert that text into a number. If data is entered and it cannot be parsed to an int, the user will receive and error message stating “Please enter a valid numeric ID”.

# References

Hamilton, G., n.d. JavaBeans API Specification. URL http://download.oracle.com/otn-pub/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/beans.101.pdf

IllegalArgumentException (Java Platform SE 8 ) [WWW Document], n.d. URL https://docs.oracle.com/javase/8/docs/api/java/lang/IllegalArgumentException.html (accessed 11.25.24).

Java JDBC Insert Example: How to insert data into a SQL table | alvinalexander.com [WWW Document], n.d. URL https://alvinalexander.com/java/edu/pj/jdbc/jdbc0002/ (accessed 10.23.24).

Java JOptionPane - javatpoint [WWW Document], n.d. . www.javatpoint.com. URL https://www.javatpoint.com/java-joptionpane (accessed 10.24.24).

Java Operators [WWW Document], n.d. URL https://www.w3schools.com/java/java\_operators.asp (accessed 10.24.24).

Java String trim() Method [WWW Document], n.d. URL https://www.w3schools.com/java/ref\_string\_trim.asp (accessed 11.25.24).

Layout Managers - JavaBitsNotebook.com [WWW Document], n.d. URL https://mathbits.com/JavaBitsNotebook/GUIs/Layout.html (accessed 10.30.24).

Oracle, n.d. Class DriverManager [WWW Document]. URL https://docs.oracle.com/javase/7/docs/api/java/sql/DriverManager.html (accessed 10.23.24).

PreparedStatement (Java Platform SE 8 ) [WWW Document], n.d. URL https://docs.oracle.com/javase/8/docs/api/java/sql/PreparedStatement.html (accessed 10.24.24).

Show Confirm Dialog with yes no buttons [WWW Document], n.d. URL https://www.tutorialspoint.com/swingexamples/show\_confirm\_dialog\_with\_yesno.htm (accessed 10.24.24).

Swing Examples - Show Warning message Dialog [WWW Document], n.d. URL https://www.tutorialspoint.com/swingexamples/show\_warning\_message\_dialog.htm (accessed 10.30.24).

Tutorial: Java Swing Buttons - Layout | CodeHS [WWW Document], n.d. URL https://codehs.com/tutorial/david/java-swing-buttons-layout (accessed 10.30.24).

Power, A. (2024). Lecture Notes, Examples and Recordings – Week 1- 11. TU Dublin Blanchardstown Campus.