DALHOUSIE UNIVERSITY

CSCI 3901: Software Development Concepts

Assignment 5

Name:Abhishek Latawa **Date:3 December, 2023**

Overview:

The primary objective of this assignment is to advance the bicycle sales database to include functionalities for order payments, returns, and store credits. The goal is to enable the company to effectively track payment statuses, handle customer returns, and manage store credits. This involves modifying the database structure, implementing an order transactions class with specified methods, and deploying the changes with minimal impact on existing operations and data integrity. The overarching aim is to enhance the company's system to better serve customer transactions and improve overall business efficiency.

Files and external data

The Implementation contains one OrderTransactions class , one DatabaseTester class , one OrderControl interface

OrderTransactions.Java:

The OrderTransactions class is a Java implementation of the orderControl interface, aiming to facilitate various transactional operations in the bicycle sales database. It connects to the database, executes SQL queries, and performs actions related to order payments, returns, and store credits. Below is a brief overview of the key functionalities:

applyCredit Method:

- Allows a customer to apply store credit to a specified order.
- Retrieves the customer ID, checks its validity, and inserts a new record into the StoreCredits table.

payOrder Method:

- Records a customer's payment for one or more unpaid orders.
- Iterates through the provided list of order IDs, processes payments, and updates the payment status in the Orders table.

returnItem Method:

- Manages the return of items from a specific order.
- Validates the order and customer association, processes returns for each item, and inserts corresponding records into the Returns table.

outstandingPayments Method:

- Retrieves a list of customers with outstanding payments.
- Executes a SQL query to calculate the total order value for unpaid orders and returns a map with customer names as keys and outstanding balances as values.

storeCredit Method:

- Retrieves a list of customers with available store credit.
- Executes a SQL query to calculate the total store credit for each customer and returns a map with customer names as keys and available credits as values.

returnWindow Method:

- Placeholder method for future implementation
- Intended to return a list of customers with average return periods, representing the average number of days between order date and return date.

UseDB Method:

Utility method to set the active database within the connection

DatabaseTester class:

The DatabaseTester class serves as a utility class for managing database connections within the bicycle sales system. It incorporates functionality to retrieve database connection details from a properties file, load them into the application, and establish a connection to the database.

Data structures and their relations to each other:

<u>Maps</u>:

HashMap was used to store the information for methods

- Outstandingpayments:Map<String, Double> was used to store the information
- Storecredit: Map<String, Double> was used to store the information of the credits remaining for the customers
- Returnwindow: Map<String, Double> was used to store the information for the avg days taken before returning the item

Connection and PreparedStatement Objects:

The OrderTransactions class uses java.sql.Connection and java.sql.PreparedStatement objects to interact with the database. These objects represent the connection to the database and precompiled SQL statements, respectively.

For database we used the SQL queries to to update the existing database with the new functionality which can be found in Database Document.pdf

Relations:

Composition with Connection and PreparedStatement The OrderTransactions class has a composition relationship with Connection and PreparedStatement objects. These objects are instantiated and used within methods to execute database operations.

Association with DatabaseTester:

The OrderTransactions class interacts with the DatabaseTester class to obtain a database connection. It calls the static getDatabaseConnection method from DatabaseTester to establish a connection.

The relationship between the use of maps in these classes is more sequential and functional. The OrderTransactions class utilizes maps to provide meaningful reports on outstanding payments and store credits, and the DatabaseTester class supports these operations by handling database connections. The maps serve as a data structure to organize and present customer-related financial information within the context of order transactions.

Assumptions:

- Payment can be made against the whole order not a single item

Choices:

Database Connection Management:

The DatabaseTester class is responsible for managing database connections. It uses properties from a configuration file to establish a connection with the MySQL database.

Use of Prepared Statements:

Prepared statements are employed in the OrderTransactions class to execute parameterized SQL queries. This helps prevent SQL injection attacks and enhances the security of database interactions.

Consistent Data Structure (Map):

The use of Map<String, Double> is consistent across the code, particularly in the methods related to outstanding payments and store credits. This data structure is employed to organize and represent financial information for customers.

Key algorithms and design elements:

Database Connection:

The getDatabaseConnection method in the DatabaseTester class implements the algorithm for establishing a connection to the MySQL database. It involves loading database connection details from a configuration file, registering the JDBC driver, and creating a connection.

I have used a separate file to store the details for the sensitive data for the DB connection which helps in protecting the sensitive data from exposure to the outside world.

Exception Handling:

Exception handling is implemented to capture and handle potential SQL exceptions, providing robustness against errors during database operations.

Input Validation:

The getCustomerID method in the OrderTransactions class performs input validation to check the existence of a customer before proceeding with database operations.

Dynamic Property Loading:

The DatabaseTester class dynamically loads database connection properties from a configuration file (sample.prop). This design element allows for easier configuration changes without modifying the source code.

Limitations:

- Using the customer name as the key in the hashmaps as in the real world scenario there could be multiple customers with same name

Q1.

Considerations for deployment with minimal impact

Have Development and Operations Collaborate

For a hassle-free deployment planning, development and operation have to go hand in hand. The operations team should be aware of every important aspect of development. Thus, you'll want to make sure that everyone in the company is on the same page.

Develop a Release Strategy

Developing a release strategy involves planning and executing the deployment of changes to a system while minimizing disruptions and ensuring a smooth transition.

Before you begin deployment, the team can come up with an effective strategy.

Deploy in Stages:

Consider deploying the changes in stages rather than all at once. This can involve deploying to a staging environment first, followed by gradual deployment to production. Staged deployment helps identify and address issues incrementally.

Database Changes:

Plan and execute any necessary database schema changes, and thoroughly test migration scripts in a staging environment.

Minimize Changes and Have a Rollback Plan

Too many sudden changes are usually what's responsible for a misbehaving application. Instead of bombarding an upgraded version with every change you want to make, add small changes with each release. That way, it’s easier for the team to search for the source of an issue.

Rigorous Testing:

Thoroughly test the new code in a staging environment that mirrors the production environment. Test all functionalities, including order payments, returns, and store credit applications. Address any issues before moving to production.

Q2. Argument as to why we should believe that your program operates as required

All the methods are implemented

I have implemented all the methods which are required to evolve the existing database to keep the track of the order payment , returns and store credits

Minimal DB changes to existing DB:

I have implemented the three new tables to store the information of the payments returns and store credits. As it becomes easier to query the db which also increases the query retrieval time.(*normalized DB*)

Testing Procedures:

The program has undergone thorough testing procedures, including unit tests, integration tests, and end-to-end tests. Test coverage has been designed to validate various aspects of the codebase, ensuring that each component functions correctly and that interactions between components are seamless.

Monitoring and Logging:

The program is equipped with robust monitoring and logging mechanisms. This allows for continuous tracking of system performance, identification of potential issues, and quick resolution of any anomalies that may arise in the live environment.

Updated DB

Database has been updated with these new tables and all the links has been made to the existing tables and all the sql commands has been written to update the existing table

References:

- ***[1]OpenAI, “ChatGPT,” chat.openai.com, Dec. 01, 2023. https://chat.openai.com***
- **[2]“GeeksforGeeks | A computer science portal for geeks,” *GeeksforGeeks*, 2019. https://www.geeksforgeeks.org**
- ***Youtube***
- ***Stack Overflow***