**Title: Restaurant Bill System Using Java and SQL**

**1. Problem Statement**

The restaurant needs a system to manage billing efficiently by automating customer and order data processing. It should collect customer details, store orders with itemized charges, and generate detailed bills, including a 10% discount. The system must integrate with a MySQL database for record-keeping. It should validate user inputs and handle errors gracefully. A user-friendly, console-based interface is required for seamless operation.

**2. Dataset Description**

**Database Name: restaurants**

**1. Table: customer1**

Stores customer details and billing date.

* **Columns**:
  + id (INT, Primary Key, Auto-Increment): Unique identifier for each customer.
  + name (VARCHAR): Name of the customer.
  + date (DATE): Date of the billing transaction.

**2. Table: order3**

Stores details of each order placed by a customer.

* **Columns**:
  + id (INT, Primary Key, Auto-Increment): Unique identifier for each order.
  + customer\_id (INT, Foreign Key): Links to the id in the customer1 table.
  + item\_name (VARCHAR): Name of the item ordered.
  + quantity (INT): Quantity of the item ordered.
  + price (FLOAT): Per-unit price of the item.
  + total (FLOAT): Total cost for the item (quantity \* price).

**Relationships:**

* **One-to-Many Relationship**:  
  Each customer (customer1) can have multiple associated orders (order3), linked via customer\_id.

**3. Language and Concepts Used**

**Programming Language**: Java

**Database**: SQL (structured in a relational database)

**Key Concepts**:

1. **Database Integration**:
   * Use of JDBC to connect and interact with a MySQL database.
   * Insert operations for customer and order data.
2. **Input Handling**:
   * Gather customer and order details using the Scanner class.
3. **Object-Oriented Programming (OOP)**:
   * Encapsulation of functionality in methods (insertCustomer, printBill).
   * Reusability of modular methods for tasks like formatting and calculations.
4. **Bill Formatting and Calculation**:
   * Structured layout for bills with headers and totals.
   * 10% discount applied to total bill calculation.
5. **Error Handling**:
   * Use of try-catch blocks for exceptions like SQL errors and missing JDBC driver.
6. **Dynamic Input Validation**:
   * Validation of user inputs, such as ensuring a non-empty customer name.
7. **Database Schema Assumptions**:
   * Tables: customer1 (customer info) and order3 (order details).
8. **JDBC Driver Loading**:
   * Explicit driver loading with Class.forName.
9. **Resource Management**:
   * Use of try-with-resources for database connections.
   * Closing the Scanner to avoid resource leaks.

**4. Code**

**Bill.java**

import java.sql.\*;

import java.time.LocalDate;

import java.util.Scanner;

public class Bill {

// Database connection details

private static final String DB\_URL = "jdbc:mysql://localhost:3306/restaurants?useSSL=false&serverTimezone=UTC";

private static final String DB\_USER = "root";

private static final String DB\_PASSWORD = "jan@#$54";

// Format for printing the bill header

public static void format(String date, String name) {

System.out.println(" ");

System.out.println("\t GREAT OBSERVER RESTAURANT CLUB ");

System.out.println("======================================================");

System.out.println(" Date :" + date);

System.out.print(" Invoice To :" + name);

System.out.println(" ");

System.out.println("---------------------------------------------------------------");

System.out.print(" Items ");

System.out.print(" \t\t\t QTY ");

System.out.println(" \t\t\t Total ");

System.out.println("---------------------------------------------------------------");

}

// Format for printing each bill item

public static void printBill(String item, int qty, float price) {

System.out.print(" " + item);

System.out.print(" \t\t\t " + qty);

System.out.println(" \t\t\t " + qty \* price);

System.out.println(" ");

}

// Calculate and print the final bill

public static void calculateBill(float[] prices, int[] quantities) {

float total = 0;

for (int i = 0; i < prices.length; i++) {

total += prices[i] \* quantities[i];

}

float discount = 0.1f \* total;

float netTotal = total - discount;

System.out.println("\n -------------------------------------------------------------------------\n");

System.out.println(" Discount: 10% \t\t\t " + discount);

System.out.println("\n -------------------------------------------------------------------------\n");

System.out.println(" Total Amount : \t\t\t\t\t\t " + netTotal);

System.out.println("\n -------------------------------------------------------------------------\n");

System.out.println(" ");

System.out.println(" \t\t Thank you and visit again!!! \n\n ");

}

// Insert customer into the database

public static int insertCustomer(String name) throws SQLException {

String query = "INSERT INTO customer1 (name, date) VALUES (?, ?)";

try (Connection conn = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD);

PreparedStatement ps = conn.prepareStatement(query, PreparedStatement.RETURN\_GENERATED\_KEYS)) {

ps.setString(1, name);

ps.setDate(2, java.sql.Date.valueOf(LocalDate.now()));

ps.executeUpdate();

var rs = ps.getGeneratedKeys();

if (rs.next()) {

return rs.getInt(1); // return the generated customer ID

}

}

return -1; // return -1 if insertion failed

}

// Insert an order into the database

public static void insertOrder(int customerId, String item, int quantity, float price) throws SQLException {

String query = "INSERT INTO order3 (customer\_id, item\_name, quantity, price, total) VALUES (?, ?, ?, ?, ?)";

try (Connection conn = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD);

PreparedStatement ps = conn.prepareStatement(query)) {

ps.setInt(1, customerId);

ps.setString(2, item);

ps.setInt(3, quantity);

ps.setFloat(4, price);

ps.setFloat(5, quantity \* price);

ps.executeUpdate();

}

}

public static void main(String[] args) {

// Load MySQL JDBC driver

try {

Class.forName("com.mysql.cj.jdbc.Driver");

} catch (ClassNotFoundException e) {

System.out.println("MySQL JDBC Driver not found!");

e.printStackTrace();

return;

}

System.out.println("\t\n Welcome to the restaurant billing code using Java\n");

Scanner sc = new Scanner(System.in);

String inputName = "";

while (inputName.isEmpty()) {

System.out.print("ENTER CUSTOMER NAME: ");

inputName = sc.nextLine().trim();

if (inputName.isEmpty()) {

System.out.println("Please enter a valid name.");

}

}

System.out.print("Enter the Number Of orders requested by the customer: ");

int inputNumber = sc.nextInt();

String[] items = new String[inputNumber];

int[] quantities = new int[inputNumber];

float[] prices = new float[inputNumber];

for (int i = 0; i < inputNumber; i++) {

System.out.println(" ");

System.out.println("Enter Item " + (i + 1) + ": ");

System.out.print("Please Enter The Item Name: ");

sc.nextLine(); // Consume newline from the previous input

items[i] = sc.nextLine();

System.out.print("Enter Quantity of Items: ");

quantities[i] = sc.nextInt();

System.out.print("Enter the Per Unit Charge of Item: ");

prices[i] = sc.nextFloat();

System.out.println(" ");

}

// Insert customer data

try {

int customerId = insertCustomer(inputName);

if (customerId == -1) {

System.out.println("Error inserting customer data.");

return;

}

format("1/1/2024", inputName);

// Insert orders and print bill

for (int i = 0; i < inputNumber; i++) {

insertOrder(customerId, items[i], quantities[i], prices[i]);

printBill(items[i], quantities[i], prices[i]);

}

calculateBill(prices, quantities);

} catch (SQLException e) {

e.printStackTrace();

System.out.println("Error connecting to the database.");

} finally {

sc.close(); // Close scanner to avoid resource leak

}

}

}

**5. Result**

**Example Output**:

1. **Customer Data Insertion**:  
   The program inserts customer information (name and billing date) into the customer1 table and retrieves a unique customer\_id.
2. **Order Data Insertion**:  
   For each item ordered, the program inserts the details (item name, quantity, price, and total) into the order3 table, linking it to the respective customer\_id.
3. **Formatted Bill Generation**:  
   A detailed bill is displayed in the console, including:
   * Customer name and Order.
   * Itemized charges with quantity and total cost.
   * 10% discount applied and net total shown.
   * Thank you message.
4. **Database Records**:  
   The database now contains:
   * A new entry in the customer1 table for the customer.
   * Multiple entries in the order3 table for the associated orders.
5. **Output Example** (Console):

Welcome to the restaurant billing code using Java

ENTER CUSTOMER NAME: jan

Enter the Number Of orders requested by the customer: 4

Enter Item 1:

Please Enter The Item Name: idli

Enter Quantity of Items: 3

Enter the Per Unit Charge of Item: 5

Enter Item 2:

Please Enter The Item Name: Dosa

Enter Quantity of Items: 4

Enter the Per Unit Charge of Item: 10

Enter Item 3:

Please Enter The Item Name: Vadai

Enter Quantity of Items: 5

Enter the Per Unit Charge of Item: 6

Enter Item 4:

Please Enter The Item Name: poori

Enter Quantity of Items: 5

Enter the Per Unit Charge of Item: 12

GREAT OBSERVER RESTAURANT CLUB

======================================================

Date :1/1/2024

Invoice To :jan

---------------------------------------------------------------

Items QTY Total

---------------------------------------------------------------

idli 3 15.0

Dosa 4 40.0

Vadai 5 30.0

poori 5 60.0

-------------------------------------------------------------------------

Discount: 10% 14.5

-------------------------------------------------------------------------

Total Amount : 130.5

-------------------------------------------------------------------------

Thank you and visit again!!!

**6. Conclusion**

The Restaurant Bill System offers an efficient solution for managing customer details and order transactions using a database-driven approach. By automating the process of adding, updating, removing, and displaying customer information and orders, the system helps streamline the billing process. Integrating with MySQL ensures data integrity and scalability. The user-friendly flowchart and system design make it easier to handle multiple operations while ensuring accurate calculations and record management. This system ultimately contributes to faster service, improved customer experience, and reliable data storage for restaurant operations.