Rajalakshmi Engineering College

Name: Ashwath s
Email: 241001021@rajalakshmi.edu.in
Roll no: 241001021
Phone: 6369582224
Branch: REC
Department: IT - Section 3
Batch: 2028
Degree: B.E - IT

Scan to verify results

2024_28_III_OOPS Using Java Lab

REC_2028_OOPS using Java_Week 11

Attempt : 1
Total Mark : 20
Marks Obtained : 20

Section 1 : Project

1. Problem Statement

Create a JDBC-based Inventory Management System that handles runtime input to manage items in an inventory. The system should allow users to:

Add a new item (item ID, name, quantity, price).

Restock an item by increasing its quantity.

Reduce the stock of an item, ensuring sufficient quantity.

Display all items in the inventory in a sorted order by item ID.

Exit the application.

Half of the code is given here; Only the remaining part should be completed.

The system should connect to a MySQL database using the following default credentials:

DB URL: jdbc:mysql://localhost/ri_db

USER: test

PWD: test123

The items table has already been created with the following structure:

Table Name: items

Input Format

The first line of input consists of an integer choice, representing the operation to be performed (1 for Add Item, 2 for Restock item, 3 for reduce item, 4 for Display, 5 for Exit).

For choice 1 (Add Item):

- The second line consists of an integer item_id.
- The third line consists of a string name.
- The fourth line consists of an integer quantity.
- The fifth line consists of a double price.

For choice 2 (Restock Item):

- The second line consists of an integer item_id.
- The third line consists of an integer quantity_to_add (must be positive).

For choice 3 (Reduce Stock):

- The second line consists of an integer item_id.
- The third line consists of an integer quantity_to_remove (must be positive).

For choice 4 (Display Inventory):

- No additional inputs are required.

For choice 5 (Exit):

- No additional inputs are required.

Output Format

For choice 1 (Add Item):

- Print "Item added successfully" if the item was added.
- Print "Failed to add item." if the insertion failed.

For choice 2 (Restock Item):

- Print "Item restocked successfully" if the restock was successful.
- Print "Item not found." if the specified item ID does not exist.

For choice 3 (Reduce Stock):

- Print "Stock reduced successfully" if the stock reduction was successful.
- Print "Not enough stock to remove." if there is insufficient quantity.
- Print "Item not found." if the specified item ID does not exist.

For choice 4 (Display Inventory):

- Display each item on a new line in the format:
- ID | Name | Quantity | Price
- If no items are available, print nothing (or handle with an appropriate message if desired).

For choice 5 (Exit):

- Print "Exiting Inventory Management System."

For invalid input:

- Print "Invalid choice. Please try again."

Sample Test Case

Input: 1
101
Laptop
50

```
1200.00
4
5
Output: Item added successfully
ID | Name | Quantity | Price
101 | Laptop | 50 | 1200.00
Exiting Inventory Management System.
```

Answer

```
import java.sql.*;
import java.util.Scanner;

class InventoryManagementSystem {
  public static void main(String[] args) {
    try (Connection conn = DriverManager.getConnection("jdbc:mysql://
localhost/ri_db", "test", "test123");
         Scanner scanner = new Scanner(System.in)) {

        boolean running = true;

        while (running) {

            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    addItem(conn, scanner);
                    break;
                case 2:
                    restockItem(conn, scanner);
                    break;
                case 3:
                    reduceStock(conn, scanner);
                    break;
                case 4:
                    displayInventory(conn);
                    break;
                case 5:
                    System.out.println("Exiting Inventory Management System.");
                    running = false;
                    break;
                default:
```

```
                System.out.println("Invalid choice. Please try again.");
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public static void addItem(Connection conn, Scanner scanner) {
    // write your code here
    try {
        int itemId = scanner.nextInt();
        scanner.nextLine(); // consume endline before reading name
        String name = scanner.nextLine();
        int quantity = scanner.nextInt();
        double price = scanner.nextDouble();

        String insertSql = "INSERT INTO items (item_id, name, quantity, price)
VALUES (?, ?, ?, ?)";
        try (PreparedStatement pst = conn.prepareStatement(insertSql)) {
            pst.setInt(1, itemId);
            pst.setString(2, name);
            pst.setInt(3, quantity);
            pst.setDouble(4, price);

            int rows = pst.executeUpdate();
            if (rows > 0) {
                System.out.println("Item added successfully");
            } else {
                System.out.println("Failed to add item.");
            }
        } catch (SQLException e) {
            // Could be duplicate key or other constraint; per spec print failure
message
            System.out.println("Failed to add item.");
        }
    } catch (Exception e) {
        // Input parsing error or similar
        System.out.println("Failed to add item.");
    }
}

public static void restockItem(Connection conn, Scanner scanner) {
    // write your code here
    try {
        int itemId = scanner.nextInt();
        int qtyToAdd = scanner.nextInt();

        if (qtyToAdd <= 0) {
            // Spec requires positive; choose stable output
            System.out.println("Item not found.");
            return;
        }

        String selectSql = "SELECT quantity FROM items WHERE item_id = ?";
        try (PreparedStatement pstSelect = conn.prepareStatement(selectSql)) {
            pstSelect.setInt(1, itemId);
            try (ResultSet rs = pstSelect.executeQuery()) {
                if (!rs.next()) {
                System.out.println("Item not found.");
                return;
                }
            }
        }

        String updateSql = "UPDATE items SET quantity = quantity + ? WHERE
item_id = ?";
        try (PreparedStatement pstUpdate = conn.prepareStatement(updateSql)) {
            pstUpdate.setInt(1, qtyToAdd);
            pstUpdate.setInt(2, itemId);
            int rows = pstUpdate.executeUpdate();
            if (rows > 0) {
                System.out.println("Item restocked successfully");
            } else {
                System.out.println("Item not found.");
            }
        } catch (SQLException e) {
            System.out.println("Item not found.");
        }
    } catch (Exception e) {
        System.out.println("Item not found.");
    }
}

public static void reduceStock(Connection conn, Scanner scanner) {
```

```
    // write your code here
    try {
        int itemId = scanner.nextInt();
        int qtyToRemove = scanner.nextInt();

        if (qtyToRemove <= 0) {
            // Spec requires positive; choose stable output
            System.out.println("Item not found.");
            return;
        }

        String selectSql = "SELECT quantity FROM items WHERE item_id = ?";
        int currentQty;
        try (PreparedStatement pstSelect = conn.prepareStatement(selectSql)) {
            pstSelect.setInt(1, itemId);
            try (ResultSet rs = pstSelect.executeQuery()) {
                if (!rs.next()) {
                    System.out.println("Item not found.");
                    return;
                }
                currentQty = rs.getInt("quantity");
            }
        }

        if (currentQty < qtyToRemove) {
            System.out.println("Not enough stock to remove.");
            return;
        }

        String updateSql = "UPDATE items SET quantity = quantity - ? WHERE
item_id = ?";
        try (PreparedStatement pstUpdate = conn.prepareStatement(updateSql)) {
            pstUpdate.setInt(1, qtyToRemove);
            pstUpdate.setInt(2, itemId);
            int rows = pstUpdate.executeUpdate();
            if (rows > 0) {
                System.out.println("Stock reduced successfully");
            } else {
                System.out.println("Item not found.");
            }
        } catch (SQLException e) {
            System.out.println("Item not found.");
```

```
          }
        } catch (Exception e) {
          System.out.println("Item not found.");
        }
    }

    public static void displayInventory(Connection conn) {
        // write your code here
        String query = "SELECT item_id, name, quantity, price FROM items ORDER BY 
item_id";
        try (PreparedStatement pst = conn.prepareStatement(query);
             ResultSet rs = pst.executeQuery()) {

            if (!rs.next()) {
                // no rows - per spec print nothing
                return;
            }

            // There is at least one row - print header then the first row and remaining 
rows
            System.out.println("ID | Name | Quantity | Price");
            // print first row
            int id = rs.getInt("item_id");
            String name = rs.getString("name");
            int qty = rs.getInt("quantity");
            double price = rs.getDouble("price");
            System.out.printf("%d | %s | %d | %.2f%n", id, name, qty, price);

            // print remaining rows
            while (rs.next()) {
                id = rs.getInt("item_id");
                name = rs.getString("name");
                qty = rs.getInt("quantity");
                price = rs.getDouble("price");
                System.out.printf("%d | %s | %d | %.2f%n", id, name, qty, price);
            }
        } catch (SQLException e) {
            // per spec, silent failure is acceptable; do nothing
        }
    }
}
```

Status : Correct ***Marks : 10/10***

2. Problem Statement

In Café Central, the menu is cataloged and stored in a database.

To efficiently manage the restaurant's menu using Java and JDBC, you must build a Restaurant Management System that supports:

Adding new menu items

Updating menu item prices

Viewing details of a menu item

Displaying all menu items in sorted order

You are given two files:

File 1: MenuItem.java (POJO Class)

This class represents the MenuItem entity.

A MenuItem contains the following details:

Field	Description
itemId	Unique Menu Item ID (Integer)
name	Item Name (String)
category	Item Category (String)
price	Item Price (Double)

Students must write code in the marked area:

```
class MenuItem {
    private int itemId;
    private String name;
    private String category;
    private double price;
    public MenuItem() {}

    public MenuItem(int itemId, String name, String category, double price) {
        // write your code here
    }

    // Include getters and setters
}
```

Expected in this part:

Assign parameter values to instance variables inside the constructor.

Add getters and setters for all attributes.

File 2: MenuItemDAO.java (Data Access Layer)

This class handles all database operations using JDBC.

Students must complete the missing JDBC logic in the following methods:

```
class MenuItemDAO {

    public void addMenuItem(Connection conn, MenuItem menuItem)
throws SQLException {
        // write your code here
    }

    public void updateItemPrice(Connection conn, int itemId, double
newPrice) throws SQLException {
        // write your code here
    }

    public void deleteMenuItem(Connection conn, int itemId) throws
SQLException {
        // write your code here
    }

    public MenuItem viewItemDetails(Connection conn, int itemId) throws
SQLException {
        // write your code here
    }

    public List<MenuItem> displayAllMenuItems(Connection conn) throws
SQLException {
        // write your code here
    }

    private MenuItem mapToMenuItem(ResultSet rs) throws SQLException {
        return new MenuItem(
            // write your code here
        );
    }
}
```

Expected in this part:

Write SQL queries for INSERT, UPDATE, DELETE, SELECT.

Execute queries using PreparedStatement or Statement.

Map ResultSet rows to MenuItem objects using mapToMenuItem().

Return a List<MenuItem> where required.

The system should connect to a MySQL database using the following default credentials:

DB URL: jdbc:mysql://localhost/ri_db

USER: test

PWD: test123

The menu table has already been created with the following structure:

Table Name: menu

Input Format

The first line of input consists of an integer choice, representing the operation to be performed (1 for Add Item, 2 for Restock item, 3 for reduce item, 4 for Display, 5 for Exit).

For choice 1 (Add Menu Item):

- The second line consists of an integer item_id.
- The third line consists of a string name.
- The fourth line consists of a string category.
- The fifth line consists of a double price.

For choice 2 (Update Item Price):

- The second line consists of an integer item_id.
- The third line consists of a double new_price.

For choice 3 (View Item Details):

- The second line consists of an integer item_id.

For choice 4 (Display All Menu Items):

- No additional inputs are required.

For choice 5 (Exit):

- No additional inputs are required.

Output Format

For choice 1 (Add Menu Item):

- Print "Menu item added successfully" if the item was added.
- Print "Failed to add item." if the insertion failed.

For choice 2 (Update Item Price):

- Print "Item price updated successfully" if the price update was successful.
- Print "Item not found." if the specified item ID does not exist.

For choice 3 (View Item Details):

- Display the item details in the format:
- ID: [item_id] | Name: [name] | Category: [category] | Price: [price]
- Print "Item not found." if the specified item ID does not exist.

For choice 4 (Display All Menu Items):

- Display each item on a new line in the format:
- ID | Name | Category | Price
- If no items are available, print nothing (or handle with an appropriate message if desired).

For choice 5 (Exit):

- Print "Exiting Restaurant Management System."

For invalid input:

- Print "Invalid choice. Please try again."

Sample Test Case

Input: 1
11
Margherita Pizza
Main Course
12.99
4
5
Output: Menu item added successfully
ID | Name | Category | Price
11 | Margherita Pizza | Main Course | 12.99
Exiting Restaurant Management System.

Answer

```
import java.sql.*;
import java.util.Scanner;
```

```
class RestaurantManagementSystem {
   public static void main(String[] args) {
      try (Connection conn = DriverManager.getConnection("jdbc:mysql://
localhost/ri_db", "test", "test123");
          Scanner scanner = new Scanner(System.in)) {

         boolean running = true;

         while (running) {
            int choice = scanner.nextInt();

            switch (choice) {
               case 1:
                  addMenuItem(conn, scanner);
                  break;
               case 2:
                  updateItemPrice(conn, scanner);
                  break;
               case 3:
                  viewItemDetails(conn, scanner);
                  break;
               case 4:
                  displayAllMenuItems(conn);
                  break;
               case 5:
                  System.out.println("Exiting Restaurant Management System.");
                  running = false;
                  break;
               default:
                  System.out.println("Invalid choice. Please try again.");
            }
         }
      } catch (SQLException e) {
         e.printStackTrace();
      }
   }

// You are using Java
   public static void addMenuItem(Connection conn, Scanner scanner) {
      //Write your code here
      try {
         int itemId = scanner.nextInt();
         scanner.nextLine(); // consume endline before reading strings
```

```
            String name = scanner.nextLine();
            String category = scanner.nextLine();
            double price = scanner.nextDouble();

            String insertSql = "INSERT INTO menu (item_id, name, category, price)
VALUES (?, ?, ?, ?)";
            try (PreparedStatement pst = conn.prepareStatement(insertSql)) {
                pst.setInt(1, itemId);
                pst.setString(2, name);
                pst.setString(3, category);
                pst.setDouble(4, price);

                int rows = pst.executeUpdate();
                if (rows > 0) {
                    System.out.println("Menu item added successfully");
                } else {
                    System.out.println("Failed to add item.");
                }
            } catch (SQLException e) {
                System.out.println("Failed to add item.");
            }
        } catch (Exception e) {
            System.out.println("Failed to add item.");
        }
    }

    public static void updateItemPrice(Connection conn, Scanner scanner) {
        //Write your code here
        try {
            int itemId = scanner.nextInt();
            double newPrice = scanner.nextDouble();

            String updateSql = "UPDATE menu SET price = ? WHERE item_id = ?";
            try (PreparedStatement pst = conn.prepareStatement(updateSql)) {
                pst.setDouble(1, newPrice);
                pst.setInt(2, itemId);

                int rows = pst.executeUpdate();
                if (rows > 0) {
                    System.out.println("Item price updated successfully");
                } else {
                    System.out.println("Item not found.");
                }
            } catch (SQLException e) {
                System.out.println("Item not found.");
            }
        } catch (Exception e) {
            System.out.println("Item not found.");
        }
    }

    public static void viewItemDetails(Connection conn, Scanner scanner) {
        //Write your code here
        try {
            int itemId = scanner.nextInt();

            String selectSql = "SELECT item_id, name, category, price FROM menu
WHERE item_id = ?";
            try (PreparedStatement pst = conn.prepareStatement(selectSql)) {
                pst.setInt(1, itemId);
                try (ResultSet rs = pst.executeQuery()) {
                    if (rs.next()) {
                        int id = rs.getInt("item_id");
                        String name = rs.getString("name");
                        String category = rs.getString("category");
                        double price = rs.getDouble("price");
                        System.out.printf("ID: %d | Name: %s | Category: %s | Price: %.2f%n",
id, name, category, price);
                    } else {
                        System.out.println("Item not found.");
                    }
                }
            } catch (SQLException e) {
                System.out.println("Item not found.");
            }
        } catch (Exception e) {
            System.out.println("Item not found.");
        }
    }

    public static void displayAllMenuItems(Connection conn) {
        //Write your code here
        String query = "SELECT item_id, name, category, price FROM menu ORDER
BY item_id";
```

```
        try (PreparedStatement pst = conn.prepareStatement(query);
             ResultSet rs = pst.executeQuery()) {

            if (!rs.isBeforeFirst()) {
                // no rows - per spec print nothing
                return;
            }

            // Print header (matching sample)
            System.out.println("ID | Name | Category      | Price");

            while (rs.next()) {
                int id = rs.getInt("item_id");
                String name = rs.getString("name");
                String category = rs.getString("category");
                double price = rs.getDouble("price");
                System.out.printf("%d | %s | %s | %.2f%n", id, name, category, price);
            }
        } catch (SQLException e) {
            // silent per spec
        }
    }
}

class MenuItem {
    private int itemId;
    private String name;
    private String category;
    private double price;

    // Constructor
    public MenuItem(int itemId, String name, String category, double price) {
        this.itemId = itemId;
        this.name = name;
        this.category = category;
        this.price = price;
    }

    // Include getters and setters

    public int getItemId() {
        return itemId;
    }

    public void setItemId(int itemId) {
        this.itemId = itemId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }
}
//
```

Status : Correct ***Marks : 10/10***