

# Rajalakshmi Engineering College

Name: Prajith S  
Email: 240701390@rajalakshmi.edu.in  
Roll no:  
Phone: 8610799150  
Branch: REC  
Department: CSE - Section 8  
Batch: 2028  
Degree: B.E - CSE

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### REC\_2028\_OOPS using Java\_Week 11

Attempt : 2  
Total Mark : 20  
Marks Obtained : 20

#### Section 1 : Project

##### 1. 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 addItem(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();
}
}

public static void addMenuItem(Connection conn, Scanner scanner) {
    try {
        int itemId = scanner.nextInt();
        scanner.nextLine();
        String name = scanner.nextLine();
        String category = scanner.nextLine();
        double price = scanner.nextDouble();

        String sql = "INSERT INTO menu (item_id, name, category, price) VALUES
(?, ?, ?, ?)";
        PreparedStatement pst = conn.prepareStatement(sql);
        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.");
    }
}

```

```
}
```

```
public static void updateItemPrice(Connection conn, Scanner scanner) {  
    try {  
        int itemId = scanner.nextInt();  
        double newPrice = scanner.nextDouble();  
  
        String sql = "UPDATE menu SET price = ? WHERE item_id = ?";  
        PreparedStatement pst = conn.prepareStatement(sql);  
        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.");  
    }  
}
```

```
public static void viewItemDetails(Connection conn, Scanner scanner) {  
    try {  
        int itemId = scanner.nextInt();  
  
        String sql = "SELECT * FROM menu WHERE item_id = ?";  
        PreparedStatement pst = conn.prepareStatement(sql);  
        pst.setInt(1, itemId);  
  
        ResultSet rs = pst.executeQuery();  
        if (rs.next()) {  
            System.out.println("ID: " + rs.getInt("item_id")  
                + " | Name: " + rs.getString("name")  
                + " | Category: " + rs.getString("category")  
                + " | Price: " + String.format("%.2f", rs.getDouble("price")));  
        } else {  
            System.out.println("Item not found.");  
        }  
    } catch (SQLException e) {  
        System.out.println("Item not found.");  
    }  
}
```



```

}

public static void displayAllMenuItems(Connection conn) {
    try {
        String sql = "SELECT * FROM menu ORDER BY item_id";
        Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery(sql);

        System.out.println("ID | Name | Category | Price");
        while (rs.next()) {
            System.out.println(
                rs.getInt("item_id") + " | " +
                rs.getString("name") + " | " +
                rs.getString("category") + " | " +
                String.format("%.2f", rs.getDouble("price"))
            );
        }
    } catch (SQLException e) {
        System.out.println("Error displaying items.");
    }
}

```

```

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

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

    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

## 2. 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) {
    try {
        int id = scanner.nextInt();
        scanner.nextLine();
        String name = scanner.nextLine();
        int qty = scanner.nextInt();
    }
}

```

```

double price = scanner.nextDouble();

String sql = "INSERT INTO items (item_id, name, quantity, price) VALUES
(?, ?, ?, ?)";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setInt(1, id);
ps.setString(2, name);
ps.setInt(3, qty);
ps.setDouble(4, price);

int rows = ps.executeUpdate();
if (rows > 0)
    System.out.println("Item added successfully");
else
    System.out.println("Failed to add item.");

} catch (Exception e) {
    System.out.println("Failed to add item.");
}
}

public static void restockItem(Connection conn, Scanner scanner) {
    try {
        int id = scanner.nextInt();
        int addQty = scanner.nextInt();

        String check = "SELECT quantity FROM items WHERE item_id = ?";
        PreparedStatement ps1 = conn.prepareStatement(check);
        ps1.setInt(1, id);
        ResultSet rs = ps1.executeQuery();

        if (!rs.next()) {
            System.out.println("Item not found.");
            return;
        }

        int newQty = rs.getInt("quantity") + addQty;

        String update = "UPDATE items SET quantity = ? WHERE item_id = ?";
        PreparedStatement ps2 = conn.prepareStatement(update);
        ps2.setInt(1, newQty);
        ps2.setInt(2, id);
    }
}

```

```

        ps2.executeUpdate();

        System.out.println("Item restocked successfully");

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

public static void reduceStock(Connection conn, Scanner scanner) {
    try {
        int id = scanner.nextInt();
        int removeQty = scanner.nextInt();

        String check = "SELECT quantity FROM items WHERE item_id = ?";
        PreparedStatement ps1 = conn.prepareStatement(check);
        ps1.setInt(1, id);
        ResultSet rs = ps1.executeQuery();

        if (!rs.next()) {
            System.out.println("Item not found.");
            return;
        }

        int currentQty = rs.getInt("quantity");

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

        int newQty = currentQty - removeQty;

        String update = "UPDATE items SET quantity = ? WHERE item_id = ?";
        PreparedStatement ps2 = conn.prepareStatement(update);
        ps2.setInt(1, newQty);
        ps2.setInt(2, id);
        ps2.executeUpdate();

        System.out.println("Stock reduced successfully");

    } catch (Exception e) {

```

```

        System.out.println("Item not found.");
    }
}

public static void displayInventory(Connection conn) {
    try {
        String sql = "SELECT * FROM items ORDER BY item_id";
        PreparedStatement ps = conn.prepareStatement(sql);
        ResultSet rs = ps.executeQuery();

        // REQUIRED HEADER for testcases
        System.out.println("ID | Name | Quantity | Price");

        while (rs.next()) {
            System.out.println(
                rs.getInt("item_id") + " | " +
                rs.getString("name") + " | " +
                rs.getInt("quantity") + " | " +
                String.format("%.2f", rs.getDouble("price"))
            );
        }

    } catch (Exception e) {
        // ignore
    }
}
}

```

**Status :** Correct

**Marks :** 10/10