

Rajalakshmi Engineering College

Name: shyam ganesh
Email: 241801266@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: AI & DS - Section 4
Batch: 2028
Degree: B.E - AI & DS

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

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) {
    int itemId = scanner.nextInt();
    scanner.nextLine(); // Consume newline

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

    MenuItem menuItem = new MenuItem(itemId, name, category, price); //
    Using POJO

    String insertQuery = "INSERT INTO menu (item_id, name, category, price)
VALUES (?, ?, ?, ?)";
    try (PreparedStatement stmt = conn.prepareStatement(insertQuery)) {
        stmt.setInt(1, menuItem.getItemId());
        stmt.setString(2, menuItem.getName());
        stmt.setString(3, menuItem.getCategory());
        stmt.setDouble(4, menuItem.getPrice());

        int rowsInserted = stmt.executeUpdate();
        System.out.println(rowsInserted > 0 ? "Menu item added successfully" :
"Failed to add item.");
    } catch (SQLException e) {
        System.out.println("Error adding item: " + e.getMessage());
    }
}

```

```
}  
}
```

```
public static void updateItemPrice(Connection conn, Scanner scanner) {  
    int itemId = scanner.nextInt();  
    double newPrice = scanner.nextDouble();
```

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

```
        int rowsUpdated = stmt.executeUpdate();  
        System.out.println(rowsUpdated > 0 ? "Item price updated successfully" :  
"Item not found.");  
    } catch (SQLException e) {  
        System.out.println("Error updating price: " + e.getMessage());  
    }  
}
```

```
public static void viewItemDetails(Connection conn, Scanner scanner) {  
    int itemId = scanner.nextInt();
```

```
    String selectQuery = "SELECT * FROM menu WHERE item_id = ?";  
    try (PreparedStatement stmt = conn.prepareStatement(selectQuery)) {  
        stmt.setInt(1, itemId);
```

```
        ResultSet rs = stmt.executeQuery();  
        if (rs.next()) {  
            MenuItem menuItem = new MenuItem(  
                rs.getInt("item_id"),  
                rs.getString("name"),  
                rs.getString("category"),  
                rs.getDouble("price")  
            );
```

```
            System.out.printf("ID: %d | Name: %s | Category: %s | Price: %.2f%n",  
                menuItem.getItemId(),  
                menuItem.getName(),  
                menuItem.getCategory(),  
                menuItem.getPrice());  
        } else {
```



```

        System.out.println("Item not found.");
    }
} catch (SQLException e) {
    System.out.println("Error retrieving item details: " + e.getMessage());
}
}

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

        System.out.println("ID | Name | Category | Price");
        while (rs.next()) {
            MenuItem menuItem = new MenuItem(
                rs.getInt("item_id"),
                rs.getString("name"),
                rs.getString("category"),
                rs.getDouble("price")
            );

            System.out.printf("%d | %s | %s | %.2f%n",
                menuItem.getItemId(),
                menuItem.getName(),
                menuItem.getCategory(),
                menuItem.getPrice());
        }
    } catch (SQLException e) {
        System.out.println("Error displaying menu items: " + e.getMessage());
    }
}
}

```

```

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;
    }

    // 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

2. Problem Statement

Create a JDBC-based School Management System that handles runtime input to manage student records. The system should allow users to:

Add a new student (student ID, name, grade level, GPA).

Update a student's GPA, ensuring the GPA value is within the valid range (0.0 - 4.0).

View a specific student's record by student ID.

Display all students in the database.

Exit the application.

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 students table has already been created with the following structure:

Table Name: students

Input Format

The first line of input consists of an integer choice, representing the operation to be performed:

(1 for Add Student, 2 for Update GPA, 3 for View Student Record, 4 for Display All Students, 5 for Exit)

For choice 1 (Add Student):

- The second line consists of an integer student_id.
- The third line consists of a string name.
- The fourth line consists of a string grade_level.
- The fifth line consists of a double gpa (must be between 0.0 and 4.0).

For choice 2 (Update GPA):

- The second line consists of an integer student_id.
- The third line consists of a double new_gpa (must be between 0.0 and 4.0).

For choice 3 (View Student Record):

- The second line consists of an integer student_id.

For choice 4 (Display All Students):

- No additional inputs are required.

For choice 5 (Exit):

- No additional inputs are required.

Output Format

The output displays:

For choice 1 (Add Student):

- Print "Student added successfully" if the student was added.
- Print "Failed to add student." if the insertion failed.

For choice 2 (Update GPA):

- Print "GPA updated successfully" if the GPA update was successful.
- Print "Student not found." if the specified student ID does not exist.
- Print "GPA must be between 0.0 and 4.0." if the provided GPA is out of the valid range.

For choice 3 (View Student Record):

- Display the student details in the format:
- ID: [student_id] | Name: [name] | Grade Level: [grade_level] | GPA: [gpa]
- Print "Student not found." if the specified student ID does not exist.

For choice 4 (Display All Students):

- Display each student on a new line in the format:
- ID | Name | Grade Level | GPA
- If there are no records, print nothing (or handle with an appropriate message if desired).

For choice 5 (Exit):

- Print "Exiting School Management System."

For invalid input:

- Print "Invalid choice. Please try again."

Sample Test Case

Input: 1

101

Alice Johnson

10

3.8

5

Output: Student added successfully
Exiting School Management System.

Answer

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

class SchoolManagementSystem {
    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:
                        addStudent(conn, scanner);
                        break;
                    case 2:
                        updateGrades(conn, scanner);
                        break;
                    case 3:
                        viewStudentRecord(conn, scanner);
                        break;
                    case 4:
                        displayAllStudents(conn);
                        break;
                    case 5:
                        System.out.println("Exiting School Management System.");
                        running = false;
                        break;
                    default:
                        System.out.println("Invalid choice. Please try again.");
                }
            }
        }
    }
}
```

```

    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public static void addStudent(Connection conn, Scanner scanner) {
    try {
        int id = scanner.nextInt();
        String name = scanner.next();
        String grade = scanner.next();
        double gpa = scanner.nextDouble();

        String sql = "INSERT INTO students (student_id, name, grade_level, gpa)
VALUES (?, ?, ?, ?)";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, id);
            ps.setString(2, name);
            ps.setString(3, grade);
            ps.setDouble(4, gpa);
            int rows = ps.executeUpdate();
            if (rows > 0)
                System.out.print("Student added successfully ");
            else
                System.out.print("Failed to add student. ");
        }
    } catch (Exception e) {
        System.out.print("Failed to add student. ");
    }
}

public static void updateGrades(Connection conn, Scanner scanner) {
    try {
        int id = scanner.nextInt();
        double newGpa = scanner.nextDouble();
        if (newGpa < 0.0 || newGpa > 4.0) {
            System.out.print("GPA must be between 0.0 and 4.0. ");
            return;
        }
        String sql = "UPDATE students SET gpa=? WHERE student_id=?";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setDouble(1, newGpa);
            ps.setInt(2, id);
            int rows = ps.executeUpdate();

```

```

        if (rows > 0)
            System.out.print("GPA updated successfully ");
        else
            System.out.print("Student not found. ");
    }
} catch (Exception e) {
    System.out.print("Student not found. ");
}
}

public static void viewStudentRecord(Connection conn, Scanner scanner) {
    try {
        int id = scanner.nextInt();
        String sql = "SELECT * FROM students WHERE student_id=?";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, id);
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                System.out.printf("ID: %d | Name: %s | Grade Level: %s | GPA: %.2f ",
                    rs.getInt("student_id"),
                    rs.getString("name"),
                    rs.getString("grade_level"),
                    rs.getDouble("gpa"));
            } else {
                System.out.print("Student not found. ");
            }
        }
    } catch (Exception e) {
        System.out.print("Student not found. ");
    }
}
}

```

```

public static void displayAllStudents(Connection conn) {
    String sql = "SELECT student_id, name, grade_level, gpa FROM students
ORDER BY student_id";
    try (Statement stmt = conn.createStatement(); ResultSet rs =
stmt.executeQuery(sql)) {
        boolean printedHeader = false;
        while (rs.next()) {
            if (!printedHeader) {
                System.out.print("ID | Name | Grade Level | GPA ");
                printedHeader = true;
            }
        }
    }
}

```

```
    }  
    System.out.printf("%d | %s | %s | %.2f ",  
        rs.getInt("student_id"),  
        rs.getString("name"),  
        rs.getString("grade_level"),  
        rs.getDouble("gpa"));  
    }  
} catch (Exception e) {  
    // ignore DB exceptions  
}  
}  
}
```

Status : Correct

Marks : 10/10