Rajalakshmi Engineering College

Name: sowmya A
Email: 240701523@rajalakshmi.edu.in
Roll no: 240701523
Phone: 9841749965
Branch: REC
Department: CSE - Section 9
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 : 1
Total Mark : 20
Marks Obtained : 10

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 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();
    }
}

// --- JDBC DAO Methods ---

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();
        scanner.nextLine();

        // Using lowercase 'itemid' for compatibility
        String sql = "INSERT INTO menu (itemid, name, category, price) VALUES
(?, ?, ?, ?)";
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, itemId);
            pstmt.setString(2, name);
            pstmt.setString(3, category);
            pstmt.setDouble(4, price);

            int rowsAffected = pstmt.executeUpdate();

            if (rowsAffected > 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) {
    try {
        int itemId = scanner.nextInt();
        scanner.nextLine();
        double newPrice = scanner.nextDouble();
        scanner.nextLine();

        // Using lowercase 'itemid' for compatibility
        String sql = "UPDATE menu SET price = ? WHERE itemid = ?";
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setDouble(1, newPrice);
            pstmt.setInt(2, itemId);

            int rowsAffected = pstmt.executeUpdate();

            if (rowsAffected > 0) {
                System.out.println("Item price updated successfully");
            } else {
                System.out.println("Item not found.");
            }
        }
    } catch (SQLException e) {
        System.out.println("An error occurred during price update.");
    } catch (Exception e) {
        System.out.println("An error occurred during price update.");
    }
}

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

```
        int itemId = scanner.nextInt();
        scanner.nextLine();

        // Using lowercase 'itemid' for compatibility
        String sql = "SELECT itemid, name, category, price FROM menu WHERE
itemid = ?";
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, itemId);

            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    // Using lowercase 'itemid' when reading from ResultSet for
consistency
                    MenuItem item = new MenuItem(
                        rs.getInt("itemid"),
                        rs.getString("name"),
                        rs.getString("category"),
                        rs.getDouble("price")
                    );
                    System.out.printf("ID: %d | Name: %s | Category: %s | Price: %.2f%n",
                            item.getItemId(), item.getName(), item.getCategory(),
item.getPrice());
                } else {
                    System.out.println("Item not found.");
                }
            }
        }
    } catch (SQLException e) {
        System.out.println("An error occurred while viewing item details.");
    } catch (Exception e) {
        System.out.println("An error occurred while viewing item details.");
    }
  }

  public static void displayAllMenuItems(Connection conn) {
    try {
        // Using lowercase 'itemid' for compatibility and ordering
        String sql = "SELECT itemid, name, category, price FROM menu ORDER BY
itemid";
        try (Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {
```

```
            boolean hasRecords = false;
            while (rs.next()) {
                if (!hasRecords) {
                    System.out.println("ID | Name | Category | Price");
                    hasRecords = true;
                }
                // Using lowercase 'itemid' when reading from ResultSet
                MenuItem item = new MenuItem(
                    rs.getInt("itemid"),
                    rs.getString("name"),
                    rs.getString("category"),
                    rs.getDouble("price")
                );
                System.out.printf("%d | %s | %s | %.2f%n",
                        item.getItemId(), item.getName(), item.getCategory(),
item.getPrice());
            }
        }
    } catch (SQLException e) {
        System.out.println("An error occurred while displaying all menu items.");
    }
}

// --- MenuItem Class Implementation (static class for compilation) ---
static 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) {
        this.itemId = itemId;
        this.name = name;
        this.category = category;
        this.price = price;
    }

    // Getters
    public int getItemId() {
        return itemId;
```

```
    }

    public String getName() {
        return name;
    }

    public String getCategory() {
        return category;
    }

    public double getPrice() {
        return price;
    }

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

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

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

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

//
```

Status : Wrong **Marks : 0/10**

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 {
      // Read inputs for choice 1
      int studentId = scanner.nextInt();
      scanner.nextLine(); // Consume newline after reading int
      String name = scanner.nextLine();
      String gradeLevel = scanner.nextLine();
      double gpa = scanner.nextDouble();
      scanner.nextLine(); // Consume newline after reading double

      if (gpa < 0.0 || gpa > 4.0) {
        // Although the prompt doesn't explicitly mention this check for
addStudent,
        // a valid GPA range is specified. We'll proceed with insertion as the
output
        // format only has success/failure messages for the database
operation.
        // However, a real-world system would validate it here.
      }

      String sql = "INSERT INTO students (student_id, name, grade_level, gpa)
VALUES (?, ?, ?, ?)";
      try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
```

```
            pstmt.setInt(1, studentId);
            pstmt.setString(2, name);
            pstmt.setString(3, gradeLevel);
            pstmt.setDouble(4, gpa);

            int rowsAffected = pstmt.executeUpdate();

            if (rowsAffected > 0) {
                System.out.println("Student added successfully");
            } else {
                System.out.println("Failed to add student.");
            }
        }
    } catch (SQLException e) {
        // Check for duplicate key error (student_id is likely primary key)
        if (e.getSQLState().startsWith("23")) { // SQLState for Integrity Constraint
Violation
            System.out.println("Failed to add student."); // Assuming failure due to
constraint (like duplicate ID)
        } else {
            e.printStackTrace();
            System.out.println("Failed to add student.");
        }
    } catch (java.util.InputMismatchException e) {
        // Handle cases where non-numeric input is given for ID or GPA
        System.out.println("Failed to add student.");
    }
}

public static void updateGrades(Connection conn, Scanner scanner) {
    try {
        // Read inputs for choice 2
        int studentId = scanner.nextInt();
        scanner.nextLine(); // Consume newline after reading int
        double newGpa = scanner.nextDouble();
        scanner.nextLine(); // Consume newline after reading double

        if (newGpa < 0.0 || newGpa > 4.0) {
            System.out.println("GPA must be between 0.0 and 4.0.");
            return;
        }
```

```
        String sql = "UPDATE students SET gpa = ? WHERE student_id = ?";
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setDouble(1, newGpa);
            pstmt.setInt(2, studentId);

            int rowsAffected = pstmt.executeUpdate();

            if (rowsAffected > 0) {
                System.out.println("GPA updated successfully");
            } else {
                System.out.println("Student not found.");
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
        // Assuming the system should only print the specified messages.
        // A more robust system might differentiate error types.
        System.out.println("An error occurred during GPA update.");
    } catch (java.util.InputMismatchException e) {
        // Handle cases where non-numeric input is given for ID or GPA
        System.out.println("An error occurred during GPA update.");
    }
}

public static void viewStudentRecord(Connection conn, Scanner scanner) {
    try {
        // Read input for choice 3
        int studentId = scanner.nextInt();
        scanner.nextLine(); // Consume newline after reading int

        String sql = "SELECT student_id, name, grade_level, gpa FROM students
WHERE student_id = ?";
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, studentId);

            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    int id = rs.getInt("student_id");
                    String name = rs.getString("name");
                    String gradeLevel = rs.getString("grade_level");
                    double gpa = rs.getDouble("gpa");
```

```
                // Output format: ID: [student_id] | Name: [name] | Grade Level:
[grade_level] | GPA: [gpa]
                // Using printf to format GPA to two decimal places as seen in
sample output.
                System.out.printf("ID: %d | Name: %s | Grade Level: %s | GPA: %.2f
%n",
                        id, name, gradeLevel, gpa);
            } else {
                System.out.println("Student not found.");
            }
          }
        }
    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("An error occurred while viewing student record.");
    } catch (java.util.InputMismatchException e) {
        // Handle cases where non-numeric input is given for ID
        System.out.println("An error occurred while viewing student record.");
    }
  }

  public static void displayAllStudents(Connection conn) {
    try {
        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 hasRecords = false;
            while (rs.next()) {
                if (!hasRecords) {
                    // Print header only if there is at least one student
                    System.out.println("ID | Name | Grade Level | GPA");
                    hasRecords = true;
                }

                int id = rs.getInt("student_id");
                String name = rs.getString("name");
                String gradeLevel = rs.getString("grade_level");
                double gpa = rs.getDouble("gpa");

                // Output format: ID | Name | Grade Level | GPA
```

```
                // Using printf to format GPA to two decimal places as seen in
sample output.
                System.out.printf("%d | %s | %s | %.2f%n", id, name, gradeLevel, gpa);
            }
            // If hasRecords is false, no output is printed (matching the
requirement: "If there are no records, print nothing")
        }
    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("An error occurred while displaying all students.");
    }
  }

}
```

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