**Exercise 1: Setting Up MySQL Database Access Using Java**

**Objectives:**

* Set up a MySQL database.
* Connect to the MySQL database using Java.

**Business Scenario:**

You are developing an employee management system that requires storing and retrieving employee data from a MySQL database.

**Tasks:**

1. **Set Up MySQL Database:**
   * Install MySQL server on your system.
   * Create a new database named **employee\_db**.
   * Create a table named **employees** with columns **id (INT, primary key, auto-increment)**, **name (VARCHAR)**, **position (VARCHAR)**, and **salary (DECIMAL)**.
2. **Create a Java Project:**
   * Set up a new Java project in your preferred IDE.
   * Add the MySQL JDBC driver to your project dependencies.
3. **Establish a Connection:**
   * Create a new Java class named **DatabaseConnection**.
   * Write code to establish a connection to the **employee\_db** using JDBC.
   * Ensure the connection is successful by printing a success message.

**Exercise 2: JDBC Architecture and Driver Connection**

**Objectives:**

* Connect to the MySQL database using JDBC API.

**Business Scenario:**

You are tasked with writing a Java application that can connect to the MySQL database and perform basic operations.

**Tasks:**

1. **Write a Connection Class:**
   * Create a Java class named **JDBCConnection**.
   * Implement a method **getConnection()** that returns a **Connection** object.
   * Use the **DriverManager** to establish a connection to the **employee\_db**.

**Exercise 3: JDBC API Data Retrieval Using SELECT Query**

**Objectives:**

* Retrieve data from the MySQL database using JDBC API.
* Display the retrieved data in the console.

**Business Scenario:**

You need to retrieve and display the list of employees from the **employees** table.

**Tasks:**

1. **Retrieve Data:**
   * Create a Java class named EmployeeDataRetrieval.
   * Implement a method getEmployees() that retrieves all employee records from the employees table using a SELECT query.
   * Use a Statement object to execute the query.
2. **Display Data:**
   * Iterate over the ResultSet to display each employee's details (id, name, position, salary) in the console.

**Exercise 4: JDBC API CRUD Implementation and Demonstration**

**Objectives:**

* Perform CRUD (Create, Read, Update, Delete) operations using JDBC API.

**Business Scenario:**

You are tasked with implementing a full-featured employee management system that allows adding, updating, deleting, and retrieving employee records.

**Tasks:**

1. **Create Employee:**
   * In the **EmployeeDataRetrieval** class, implement a method **addEmployee(String name, String position, double salary)** that inserts a new employee record into the **employees** table.
   * Use a **PreparedStatement** for this operation.
2. **Update Employee:**
   * Implement a method **updateEmployee(int id, String name, String position, double salary)** that updates an existing employee record based on the id.
   * Use a **PreparedStatement** for this operation.
3. **Delete Employee:**
   * Implement a method **deleteEmployee(int id)** that deletes an employee record based on the id.
   * Use a **PreparedStatement** for this operation.
4. **Demonstrate CRUD Operations:**
   * Write a main method that demonstrates the CRUD operations:
     + Add a few employee records.
     + Update an employee record.
     + Delete an employee record.
     + Retrieve and display all employee records.

**Exercise 5: JDBC Application to Access Data from Table**

**Objectives:**

* Develop a JDBC application to access and manipulate data from a MySQL table.

**Business Scenario:**

You need to create a Java application that allows users to interact with the employee database, perform CRUD operations, and view employee data.

**Tasks:**

1. **User Interface:**
   * Create a simple console-based user interface that prompts the user to choose an operation (Add, Update, Delete, View).
2. **Add Employee:**
   * Prompt the user to enter employee details (name, position, salary).
   * Call the **addEmployee()** method to add the new employee to the database.
3. **Update Employee:**
   * Prompt the user to enter the employee ID and new details.
   * Call the **updateEmployee()** method to update the employee record.
4. **Delete Employee:**
   * Prompt the user to enter the employee ID.
   * Call the **deleteEmployee()** method to delete the employee record.
5. **View Employees:**
   * Call the **getEmployees()** method to retrieve and display all employee records.
6. **Loop and Exit:**
   * Implement a loop that allows the user to perform multiple operations until they choose to exit the application.