Day 26

Assignment 3: Here's the modified Java program that uses a PreparedStatement to parameterize the SELECT query and prevent SQL injection:

A)

Java code:

```
import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.Scanner;


public class UserAuthentication {

  public static void main(String[] args) {

    // Database connection details

    String url = "jdbc:mysql://localhost:3306/mydatabase";

    String username = "username"; // Replace with your MySQL username

    String password = "password"; // Replace with your MySQL password


    // User input

    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter User ID: ");

    String userID = scanner.nextLine();
```

```
System.out.print("Enter Password: ");
String passwordInput = scanner.nextLine();

// Hash the password input
String hashedPassword = hashPassword(passwordInput);

try {
  // Establish connection
  Connection connection = DriverManager.getConnection(url, username, password);

  // Check if user access is allowed
  boolean accessAllowed = checkUserAccess(connection, userID, hashedPassword);

  if (accessAllowed) {
    System.out.println("Access granted. Welcome, " + userID + "!");
  } else {
    System.out.println("Access denied. Incorrect User ID or Password.");
  }

  // Close the connection
  connection.close();
} catch (SQLException e) {
  System.err.println("Failed to connect to the database");
  e.printStackTrace();
}
```

```
    }

    // Method to hash the password
    private static String hashPassword(String password) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hash = digest.digest(password.getBytes());
            StringBuilder hexString = new StringBuilder();
            for (byte b : hash) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) hexString.append('0');
                hexString.append(hex);
            }
            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
    }

    // Method to check if user access is allowed
    private static boolean checkUserAccess(Connection connection, String userID, String hashedPassword)
throws SQLException {
        String selectSQL = "SELECT * FROM User WHERE userID = ? AND password = ?";
        try (PreparedStatement statement = connection.prepareStatement(selectSQL)) {
            statement.setString(1, userID);
            statement.setString(2, hashedPassword);

            try (ResultSet resultSet = statement.executeQuery()) {

                return resultSet.next(); // Returns true if user with given ID and hashed password exists

            }

        }

    }

}
```

Explanation:

Import Statements: Import necessary classes from java.sql and java.security packages.

Database URL, Username, and Password: Replace the placeholders url, username, and password with your MySQL database connection details.

User Input: Accept user input for 'User ID' and 'Password'.

Hashing Password: The hashPassword() method hashes the password input using the SHA-256 algorithm. This hashed password will be used for authentication.

Connection Establishment: Establish a connection to the MySQL database.

Check User Access: The checkUserAccess() method uses a PreparedStatement with parameterized queries to prevent SQL injection. It checks if the provided 'User ID' and hashed password match any entries in the 'User' table. If a match is found, access is granted.

Closing Connection: Finally, close the connection using the close() method.

Using a PreparedStatement with parameterized queries is a best practice to prevent SQL injection attacks, making the application more secure.