```java
package edu.asu.ASUHelloWorldJavaFXMaven;

import static org.mockito.Mockito.*;

public class JUnitTest {

    private DatabaseHelper dbHelper;
    private Connection mockConnection;
    private PreparedStatement mockPreparedStatement;
    private Statement mockStatement;


   @BeforeEach
    void setUp() throws SQLException {
        // You can use H2 in-memory database for testing
        String jdbcUrl = "jdbc:h2:mem:testdb"; // H2 in-memory DB
        mockConnection = DriverManager.getConnection(jdbcUrl, "sa", "");

    try (Statement stmt = mockConnection.createStatement()){

     String userTable = "CREATE TABLE IF NOT EXISTS cse360users ("
      + "id INT AUTO_INCREMENT PRIMARY KEY, "
      + "password VARCHAR(255), "
      + "role VARCHAR(20), "
      + "access BOOLEAN, "
      + "email VARCHAR(255), "
      + "first VARCHAR(255), "
      + "middle VARCHAR(255), "
      + "last VARCHAR(255), "
      + "preferred VARCHAR(255), "
      + "USERNAME VARCHAR(255), "
      + "temp VARCHAR(255), "
      + "date VARCHAR(255))";

     stmt.execute(userTable);
    }

        try (Statement stmt = mockConnection.createStatement()) {
            // Insert test users
            stmt.executeUpdate("INSERT INTO cse360users (password, role, access, email, first,
                    + "VALUES ('pass123', 'admin', true, 'admin', 'First', 'Middle', 'Last',
            stmt.executeUpdate("INSERT INTO cse360users (password, role, access, email, first,
                    + "VALUES ('pass456', 'user', false, 'user', 'SFirst', 'SMiddle', 'SLast',
        }

        dbHelper = new DatabaseHelper(mockConnection);
    }

/*    @Test
    void testIsDatabaseEmpty_whenDatabaseIsEmpty() throws SQLException {
        // When
        boolean isEmpty = dbHelper.isDatabaseEmpty();

        // Then
        assertTrue(isEmpty, "Database should be empty initially");
    }
*/
   @Test
   void testAccess() throws SQLException {
       // Given
       String username = "testUser";  // The username you're testing for
```

```
70          boolean expectedAccess = true; // The expected access value
71
72          // When
73          dbHelper.access(username); // Call the method to test
74
75          // Then
76          String query = "SELECT access FROM cse360users WHERE username = ?";
77          try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
78              pstmt.setString(1, username); // Set the username in the query
79
80              try (ResultSet rs = pstmt.executeQuery()) {
81                  assertTrue(rs.next(), "User should be found in the database");
82                  assertEquals(expectedAccess, rs.getBoolean("access"), "Access value should match
83              }
84          }
85      }
86
87
88      @Test
89      void testRegister() throws SQLException {
90          // Given
91          String username = "testUser";
92          String password = "testPassword";
93          String role = "student";  // Example role
94
95          // When
96          dbHelper.register(username, password, role); // Call the method to test
97
98          // Then
99          String query = "SELECT * FROM cse360users WHERE username = ? AND role = ?";
100         try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
101             pstmt.setString(1, username); // Set the username in the query
102             pstmt.setString(2, role);     // Set the role in the query
103
104             try (ResultSet rs = pstmt.executeQuery()) {
105                 assertTrue(rs.next(), "User should be inserted into the database");
106                 assertEquals(username, rs.getString("username"));
107                 assertEquals(role, rs.getString("role"));
108                 // Optionally check the encrypted password:
109                 String encryptedPasswordFromDb = rs.getString("password");
110                 String expectedEncryptedPassword = Base64.getEncoder().encodeToString
111                 assertEquals(expectedEncryptedPassword, encryptedPasswordFromDb, "Passwords
112             }
113         }
114     }
115
116     @Test
117     void testInvitedata() throws SQLException {
118         // Given
119         String role = "role";
120         String temp = "temp123";
121         String date = "2024-11-20";
122
123         // When
124         dbHelper.invitedata(role, temp, date); // Call the method to test
125
126         // Then
127         String query = "SELECT * FROM cse360users WHERE role = ? AND temp = ? AND date = ?";
128         try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
129             pstmt.setString(1, role);
130             pstmt.setString(2, temp);
```

```
            pstmt.setString(3, date);

            try (ResultSet rs = pstmt.executeQuery()) {
                assertTrue(rs.next(), "User should be inserted into the database");
                assertEquals(role, rs.getString("role"));
                assertEquals(temp, rs.getString("temp"));
                assertEquals(date, rs.getString("date"));
                }
        }
    }

    @Test
    void testUpdate() throws SQLException {
        // Given
        String username = "testUser";
        String initialEmail = "initialEmail@example.com";
        String initialFirst = "InitialFirst";
        String initialMiddle = "InitialMiddle";
        String initialLast = "InitialLast";
        String initialPreferred = "InitialPreferred";

        // Insert initial data into the database for the given username
        String insertUser = "INSERT INTO cse360users (username, email, first, middle, last,
        try (PreparedStatement pstmt = mockConnection.prepareStatement(insertUser)) {
            pstmt.setString(1, username);
            pstmt.setString(2, initialEmail);
            pstmt.setString(3, initialFirst);
            pstmt.setString(4, initialMiddle);
            pstmt.setString(5, initialLast);
            pstmt.setString(6, initialPreferred);
            pstmt.executeUpdate();
        }

        // New data to update
        String newEmail = "newEmail@example.com";
        String newFirst = "NewFirst";
        String newMiddle = "NewMiddle";
        String newLast = "NewLast";
        String newPreferred = "NewPreferred";

        // When: Call the method to update the user details
        dbHelper.update(newEmail, newFirst, newMiddle, newLast, newPreferred, username);

        // Then: Check if the data is updated in the database
        String query = "SELECT * FROM cse360users WHERE username = ?";
        try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
            pstmt.setString(1, username);

            try (ResultSet rs = pstmt.executeQuery()) {
                assertTrue(rs.next(), "User should be updated in the database");
                // Check updated fields
                assertEquals(newEmail, rs.getString("email"));
                assertEquals(newFirst, rs.getString("first"));
                assertEquals(newMiddle, rs.getString("middle"));
                assertEquals(newLast, rs.getString("last"));
                assertEquals(newPreferred, rs.getString("preferred"));
                // Check if access is set to true
                assertTrue(rs.getBoolean("access"));
            }
        }
    }
```

```
192
193     @Test
194     void testResetUser() throws SQLException {
195         // Given: Setup a user with a specific username and temp value to test
196         String username = "testUser";
197         String temp = "temp123"; // This is the temp value we will use to find the user
198         String date = "2024-11-20";  // The new date value to update
199
200         // Insert the user with a specific username and temp value
201         String insertUser = "INSERT INTO cse360users (username, temp, password) VALUES (?, ?,
202         try (PreparedStatement pstmt = mockConnection.prepareStatement(insertUser)) {
203             pstmt.setString(1, username);
204             pstmt.setString(2, temp);
205             pstmt.setString(3, "oldPassword");  // Set an initial password
206             pstmt.executeUpdate();
207         }
208
209         // Confirm the user is inserted
210         String checkInsertQuery = "SELECT username, temp, password FROM cse360users WHERE
211         try (PreparedStatement pstmt = mockConnection.prepareStatement(checkInsertQuery)) {
212             pstmt.setString(1, username);
213             try (ResultSet rs = pstmt.executeQuery()) {
214                 assertTrue(rs.next(), "User should be inserted into the database");
215                 assertEquals(temp, rs.getString("temp"));
216                 assertNotNull(rs.getString("password"), "Password should not be null
217             }
218         }
219
220         // When: Call the resetuser method to reset the password, update the temp and date
221         dbHelper.resetuser(username, temp, date);
222
223         // Then: Check that the password is null, the temp is updated, and the date is set
224         String query = "SELECT password, temp, date FROM cse360users WHERE username = ?";
225         try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
226             pstmt.setString(1, username);
227             try (ResultSet rs = pstmt.executeQuery()) {
228                 assertTrue(rs.next(), "User should be found in the database with the specified
229                 // Check that the password is set to null
230                 assertNull(rs.getString("password"), "Password should be set to null");
231                 // Check that the temp is still the same
232                 assertEquals(temp, rs.getString("temp"), "Temp value should remain the same");
233                 // Check that the date is set correctly
234                 assertEquals(date, rs.getString("date"), "Date should be updated");
235             }
236         }
237     }
238
239     @Test
240     void testUpdatePass() throws SQLException {
241         // Given: Setup a user with a specific username and password to test
242         String username = "testUser";
243         String oldPassword = "oldPassword123"; // Old password to insert
244         String newPassword = "newPassword456";  // The new password to update
245         String encryptedOldPassword = Base64.getEncoder().encodeToString(oldPassword.getBytes
246         String encryptedNewPassword = Base64.getEncoder().encodeToString(newPassword.getBytes
247
248         // Insert the user with an initial password
249         String insertUser = "INSERT INTO cse360users (username, password) VALUES (?, ?)";
250         try (PreparedStatement pstmt = mockConnection.prepareStatement(insertUser)) {
251             pstmt.setString(1, username);
252             pstmt.setString(2, encryptedOldPassword);  // Insert the old password
```

```
253             pstmt.executeUpdate();
254         }
255
256         // Confirm the user is inserted and the password is correct
257         String checkInsertQuery = "SELECT username, password FROM cse360users WHERE username =
258         try (PreparedStatement pstmt = mockConnection.prepareStatement(checkInsertQuery)) {
259             pstmt.setString(1, username);
260             try (ResultSet rs = pstmt.executeQuery()) {
261                 assertTrue(rs.next(), "User should be inserted into the database");
262                 assertEquals(encryptedOldPassword, rs.getString("password"), "Password should
263             }
264         }
265
266         // When: Call the updatepass method to update the password
267         dbHelper.updatepass(newPassword, username); // Update the password
268
269         // Then: Check that the password is updated to the new encrypted password
270         String query = "SELECT password FROM cse360users WHERE username = ?";
271         try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
272             pstmt.setString(1, username);
273             try (ResultSet rs = pstmt.executeQuery()) {
274                 assertTrue(rs.next(), "User should be found in the database with the specified
275                 // Check that the password is updated to the new encrypted password
276                 assertEquals(encryptedNewPassword, rs.getString("password"), "Password should be
277             }
278         }
279     }
280
281     @Test
282     void testSetRole() throws SQLException {
283         // Given: Create a user with an initial role
284         String username = "testuser";
285         String initialRole = "student";
286         dbHelper.register(username, "password", initialRole); // Assuming you have a register
287
288         // When: Update the user's role
289         String newRole = "admin";
290         dbHelper.setrole(newRole, username);
291
292         // Then: Verify the role is updated in the database
293         String query = "SELECT role FROM cse360users WHERE username = ?";
294         try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
295             pstmt.setString(1, username);
296             try (ResultSet rs = pstmt.executeQuery()) {
297                 assertTrue(rs.next(), "User should be found in the database");
298                 assertEquals(newRole, rs.getString("role"), "Role should be updated to the new
299             }
300         }
301     }
302
303     @Test
304     void testLogin() throws SQLException {
305         // Given: Create a user with known credentials
306         String username = "testuser";
307         String password = "password123";  // This will be encrypted in the login method
308         dbHelper.register(username, password, "student"); // Assuming register method is
309
310         // When: Attempt to log in with correct credentials
311         boolean loginSuccessful = dbHelper.login(username, password);
312
313         // Then: The login should be successful
```

```
314         assertTrue(loginSuccessful, "Login should be successful with correct username and
    password");
315     }
316
317     void testHelpTemp() throws SQLException {
318         // Given: Insert a user with a known temp value
319         String temp = "temp123";
320         String username = "testuser";
321
322         dbHelper.register(username, "password123", "student"); // Assuming the register method
323
324         // Insert a row with the known temp value (directly using SQL for this test)
325         String insertTemp = "INSERT INTO cse360users (username, temp) VALUES (?, ?)";
326         try (PreparedStatement pstmt = mockConnection.prepareStatement(insertTemp)) {
327             pstmt.setString(1, username);
328             pstmt.setString(2, temp);
329             pstmt.executeUpdate();
330         }
331
332         // When: Check if the temp value exists
333         boolean result = dbHelper.helptemp(temp);
334
335         // Then: The result should be true since the temp exists in the database
336         assertTrue(result, "Temp should exist in the database");
337     }
338
339     @Test
340     void testDoesUserExistWithExistingUser() throws SQLException {
341         // Given: A user "testuser" exists in the database
342
343         String username = "testuser"; // Known user
344
345         // When: Check if the user exists
346         boolean result = dbHelper.doesUserExist(username);
347
348         // Then: The result should be true since the user exists
349         assertTrue(result, "User should exist in the database");
350     }
351
352     @Test
353     void testDoesUserExistWithNonExistingUser() throws SQLException {
354         // Given: A user "nonexistentuser" does not exist in the database
355
356         String username = "nonexistentuser"; // Non-existing user
357
358         // When: Check if the user exists
359         boolean result = dbHelper.doesUserExist(username);
360
361         // Then: The result should be false since the user does not exist
362         assertFalse(result, "User should not exist in the database");
363     }
364
365
366     @Test
367     void testPrefnameWithExistingUserAndPreferredName() throws SQLException {
368         // Given: A user "testuser" with a preferred name "Pref"
369         String username = "testuser";
370
371         // When: Call the prefname method
372         String result = dbHelper.prefname(username);
373
```

```
374         // Then: The result should be the preferred name "Pref"
375         assertEquals("Pref", result, "The preferred name should be 'Pref'");
376     }
377
378     @Test
379     void testDisplayUsersByAdmin() throws SQLException {
380         // Given: Two users are inserted into the database.
381
382         // When: The displayUsersByAdmin method is called
383         dbHelper.displayUsersByAdmin();
384
385         // Then: Verify that the data for these users exists in the database.
386         String query = "SELECT * FROM cse360users WHERE id = ?";
387
388         try (PreparedStatement pstmt = mockConnection.prepareStatement(query)) {
389             pstmt.setInt(1, 1);  // Check for first user (ID = 1)
390             try (ResultSet rs = pstmt.executeQuery()) {
391                 assertTrue(rs.next(), "First user should be present in the database");
392                 assertEquals("test1@example.com", rs.getString("email"));
393                 assertEquals("password1", rs.getString("password"));
394                 assertEquals("admin", rs.getString("role"));
395             }
396
397             pstmt.setInt(1, 2);  // Check for second user (ID = 2)
398             try (ResultSet rs = pstmt.executeQuery()) {
399                 assertTrue(rs.next(), "Second user should be present in the database");
400                 assertEquals("test2@example.com", rs.getString("email"));
401                 assertEquals("password2", rs.getString("password"));
402                 assertEquals("user", rs.getString("role"));
403             }
404         }
405     }
406
407
408 }
```