1 AdminView.java

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import javax.swing.*;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Unit tests for the AdminView class.
 * This class validates the initialization and properties of components in AdminView.
 */
class AdminViewTest {

    private AdminView adminView; // Instance of AdminView to be tested

    /**
     * Sets up the AdminView instance before each test.
     */
    @BeforeEach
    void setUp() {
        adminView = new AdminView();
    }

    /**
     * Validates that the AdminView is initialized successfully.
     */
    @Test
    void testAdminViewInitialization() {
        assertNotNull(adminView, "AdminView should be initialized.");
    }

    /**
     * Validates that the search field is initialized with correct properties.
     */
    @Test
    void testSearchFieldInitialization() {
        JTextField searchField = adminView.getSearchField();
        assertNotNull(searchField, "Search field should be initialized.");
        assertEquals(20, searchField.getColumns(), "Search field should have 20 columns.");
    }

    /**
     * Validates that the search button is initialized with the correct text.
```

```
     */
    @Test
    void testSearchButtonInitialization() {
        JButton searchButton = adminView.getSearchButton();
        assertNotNull(searchButton, "Search button should be initialized.");
        assertEquals("Search", searchButton.getText(), "Search button text should be 'Search'.");
    }

    /**
     * Validates that the users table is initialized properly.
     */
    @Test
    void testUsersTableInitialization() {
        JTable usersTable = adminView.getUsersTable();
        assertNotNull(usersTable, "Users table should be initialized.");
    }

    /**
     * Validates that the add user button is initialized with the correct text.
     */
    @Test
    void testAddUserButtonInitialization() {
        JButton addUserButton = adminView.getAddUserButton();
        assertNotNull(addUserButton, "Add User button should be initialized.");
        assertEquals("Add User", addUserButton.getText(), "Add User button text should be 'Add
User'.");
    }

    /**
     * Validates that the delete user button is initialized with the correct text.
     */
    @Test
    void testDeleteUserButtonInitialization() {
        JButton deleteUserButton = adminView.getDeleteUserButton();
        assertNotNull(deleteUserButton, "Delete User button should be initialized.");
        assertEquals("Delete User", deleteUserButton.getText(), "Delete User button text should be
'Delete User'.");
    }

    /**
     * Validates that the manage attendance button is initialized with the correct text.
     */
    @Test
    void testManageAttendanceButtonInitialization() {
```

```
        JButton manageAttendanceButton = adminView.getManageAttendanceButton();
        assertNotNull(manageAttendanceButton, "Manage Attendance button should be
initialized.");
        assertEquals("Manage Attendance", manageAttendanceButton.getText(), "Manage
Attendance button text should be 'Manage Attendance'.");
    }

    /**
     * Validates that the update details button is initialized with the correct text.
     */
    @Test
    void testUpdateDetailsButtonInitialization() {
        JButton updateDetailsButton = adminView.getUpdateDetailsButton();
        assertNotNull(updateDetailsButton, "Update Details button should be initialized.");
        assertEquals("View/Update Details", updateDetailsButton.getText(), "Update Details button
text should be 'View/Update Details'.");
    }

    /**
     * Validates that the send message button is initialized with the correct text.
     */
    @Test
    void testSendMessageButtonInitialization() {
        JButton sendMessageButton = adminView.getSendMessageButton();
        assertNotNull(sendMessageButton, "Send Message button should be initialized.");
        assertEquals("Send Message", sendMessageButton.getText(), "Send Message button text
should be 'Send Message'.");
    }

    /**
     * Validates that the message area is initialized with the correct dimensions.
     */
    @Test
    void testMessageAreaInitialization() {
        JTextArea messageArea = adminView.getMessageArea();
        assertNotNull(messageArea, "Message area should be initialized.");
        assertEquals(5, messageArea.getRows(), "Message area should have 5 rows.");
        assertEquals(30, messageArea.getColumns(), "Message area should have 30 columns.");
    }

    /**
     * Validates that the save user button is initialized with the correct text.
     */
    @Test
```

```
    void testSaveUserButtonInitialization() {
        JButton saveUserButton = adminView.getSaveUserButton();
        assertNotNull(saveUserButton, "Save User button should be initialized.");
        assertEquals("Save User", saveUserButton.getText(), "Save User button text should be
'Save User'.");
    }

    /**
     * Validates that all user detail fields are initialized properly.
     */
    @Test
    void testUserDetailsFieldsInitialization() {
        assertNotNull(adminView.getFirstNameField(), "First Name field should be initialized.");
        assertNotNull(adminView.getLastNameField(), "Last Name field should be initialized.");
        assertNotNull(adminView.getAgeField(), "Age field should be initialized.");
        assertNotNull(adminView.getDobField(), "DOB field should be initialized.");
        assertNotNull(adminView.getAddressField(), "Address field should be initialized.");
        assertNotNull(adminView.getPhoneField(), "Phone field should be initialized.");
        assertNotNull(adminView.getEmailField(), "Email field should be initialized.");
        assertNotNull(adminView.getRoleField(), "Role field should be initialized.");
        assertNotNull(adminView.getCourseField(), "Course field should be initialized.");
        assertNotNull(adminView.getBranchField(), "Branch field should be initialized.");
    }
}
```

2. LoginView.java

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import javax.swing.*;
import static org.junit.jupiter.api.Assertions.*;

//Tests for the LoginView class.
class LoginViewTest {
    private LoginView loginView; // Instance of the LoginView to test

    /**
     * Initialize LoginView before each test.
     */
    @BeforeEach
    void setUp() {
        loginView = new LoginView();
    }
```

```
    /**
     * Ensure LoginView is instantiated correctly.
     */
    @Test
    void testLoginViewInitialization() {
        assertNotNull(loginView, "LoginView instance should not be null.");
    }
     //Verify the email field is properly initialized and can accept text.
    @Test
    void testEmailFieldInitialization() {
        JTextField emailField = loginView.getEmailField();
        assertNotNull(emailField, "Email field should be initialized.");
        emailField.setText("user@example.com");
        assertEquals("user@example.com", emailField.getText(), "Email field should retain input
text.");
    }
    /**
     * Verify the password field is properly initialized and can accept text.
     */
    @Test
    void testPasswordFieldInitialization() {
        JPasswordField passwordField = loginView.getPasswordField();
        assertNotNull(passwordField, "Password field should be initialized.");
        passwordField.setText("securepassword");
        assertEquals("securepassword", new String(passwordField.getPassword()), "Password
field should retain input text.");
    }

    /**
     * Verify the login button is properly initialized with correct text.
     */
    @Test
    void testLoginButtonInitialization() {
        JButton loginButton = loginView.getLoginButton();
        assertNotNull(loginButton, "Login button should be initialized.");
        assertEquals("Login", loginButton.getText(), "Login button text should be 'Login'.");
    }
}
```

3. StudentView.java

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import javax.swing.*;
```

```
import static org.junit.jupiter.api.Assertions.*;

class StudentViewTest {

    private StudentView studentView; // Instance of StudentView to be tested
    // Sets up the StudentView instance before each test
    @BeforeEach
    void setUp() {
        studentView = new StudentView();
    }
    // Ensures StudentView is initialized successfully
    @Test
    void testStudentViewInitialization() {
        assertNotNull(studentView, "StudentView instance should not be null.");
    }

    // Validates the personal details button is properly initialized
    @Test
    void testViewPersonalDetailsButtonInitialization() {
        JButton button = studentView.getViewPersonalDetailsButton();
        assertNotNull(button);
        assertEquals("View Personal Details", button.getText());
    }

    // Validates the marks button is properly initialized
    @Test
    void testViewMarksButtonInitialization() {
        JButton button = studentView.getViewMarksButton();
        assertNotNull(button);
        assertEquals("View Marks", button.getText());
    }

    // Validates the attendance button is properly initialized
    @Test
    void testViewAttendanceButtonInitialization() {
        JButton button = studentView.getViewAttendanceButton();
        assertNotNull(button);
        assertEquals("View Attendance", button.getText());
    }

    // Validates the fees button is properly initialized
    @Test
    void testViewFeesButtonInitialization() {
        JButton button = studentView.getViewFeesButton();
```

```
        assertNotNull(button);
        assertEquals("View Fees", button.getText());
    }

    // Validates the marks table is properly initialized
    @Test
    void testMarksTableInitialization() {
        JTable table = studentView.getMarksTable();
        assertNotNull(table);
    }

    // Validates the attendance table is properly initialized
    @Test
    void testAttendanceTableInitialization() {
        JTable table = studentView.getAttendanceTable();
        assertNotNull(table);
    }

    // Validates the fees table is properly initialized
    @Test
    void testFeesTableInitialization() {
        JTable table = studentView.getFeesTable();
        assertNotNull(table);
    }

    // Validates the setPersonalDetails method updates labels correctly
    @Test
    void testSetPersonalDetails() {
        studentView.setPersonalDetails(
            "John Doe", 20, "2003-01-01", "123 Main St",
            "1234567890", "john@example.com", "Computer Science", "Engineering"
        );

        assertEquals("John Doe", studentView.nameLabel.getText());
        assertEquals("20", studentView.ageLabel.getText());
        assertEquals("2003-01-01", studentView.dobLabel.getText());
        assertEquals("123 Main St", studentView.addressLabel.getText());
        assertEquals("1234567890", studentView.phoneLabel.getText());
        assertEquals("john@example.com", studentView.emailLabel.getText());
        assertEquals("Computer Science", studentView.courseLabel.getText());
        assertEquals("Engineering", studentView.branchLabel.getText());
    }
}
```

4. TeacherView.java

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import javax.swing.*;
import static org.junit.jupiter.api.Assertions.*;

class TeacherViewTest {

    private TeacherView teacherView; // Instance of TeacherView for testing
    // Setup the TeacherView instance before each test
    @BeforeEach
    void setUp() {
        teacherView = new TeacherView();
    }

    // Ensure TeacherView is initialized properly
    @Test
    void testTeacherViewInitialization() {
        assertNotNull(teacherView);
    }

    // Test if view personal details button is initialized correctly
    @Test
    void testViewPersonalDetailsButton() {
        JButton button = teacherView.getViewPersonalDetailsButton();
        assertNotNull(button);
        assertEquals("View Personal Details", button.getText());
    }

    // Test if update personal details button is initialized correctly
    @Test
    void testUpdatePersonalDetailsButton() {
        JButton button = teacherView.getUpdatePersonalDetailsButton();
        assertNotNull(button);
        assertEquals("Update Personal Details", button.getText());
    }

    // Test if manage attendance button is initialized correctly
    @Test
    void testManageAttendanceButton() {
        JButton button = teacherView.getManageAttendanceButton();
        assertNotNull(button);
        assertEquals("Update Attendance", button.getText());
```

```
}

// Test if save marks button is initialized correctly
@Test
void testSaveMarksButton() {
    JButton button = teacherView.getEnterMarksButton();
    assertNotNull(button);
    assertEquals("Save Marks", button.getText());
}

// Test if attendance table is initialized properly
@Test
void testAttendanceTable() {
    JTable table = teacherView.getAttendanceTable();
    assertNotNull(table);
}

// Test if marks table is initialized properly
@Test
void testMarksTable() {
    JTable table = teacherView.getMarksTable();
    assertNotNull(table);
}

// Test if subject field is initialized properly
@Test
void testSubjectField() {
    JTextField field = teacherView.getSubjectField();
    assertNotNull(field);
}

// Test if marks field is initialized properly
@Test
void testMarksField() {
    JTextField field = teacherView.getMarksField();
    assertNotNull(field);
}

// Test if student ID field is initialized properly
@Test
void testStudentIdField() {
    JTextField field = teacherView.getStudentIdField();
    assertNotNull(field);
}
```

```
    // Test if first name field is initialized properly
    @Test
    void testFirstNameField() {
        JTextField field = teacherView.getFirstNameField();
        assertNotNull(field);
    }

    // Test if last name field is initialized properly
    @Test
    void testLastNameField() {
        JTextField field = teacherView.getLastNameField();
        assertNotNull(field);
    }

    // Test if age field is initialized properly
    @Test
    void testAgeField() {
        JTextField field = teacherView.getAgeField();
        assertNotNull(field);
    }

    // Test if date of birth field is initialized properly
    @Test
    void testDobField() {
        JTextField field = teacherView.getDobField();
        assertNotNull(field);
    }

    // Test if save details button is initialized correctly
    @Test
    void testSaveDetailsButton() {
        JButton button = teacherView.getSaveDetailsButton();
        assertNotNull(button);
        assertEquals("Save Details", button.getText());
    }
}
```

Model Classes:

1. Attendance.java

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.sql.*;
```

```
import java.util.List;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

class AttendanceTest {

    private Connection mockConnection; // Mocked database connection
    private PreparedStatement mockStatement; // Mocked PreparedStatement
    private ResultSet mockResultSet; // Mocked ResultSet

    @BeforeEach
    void setUp() throws SQLException {
        // Initialize mock objects before each test
        mockConnection = mock(Connection.class);
        mockStatement = mock(PreparedStatement.class);
        mockResultSet = mock(ResultSet.class);
    }

    // Test for marking attendance
    @Test
    void testMarkAttendance() throws SQLException {
        Attendance attendance = new Attendance();
        attendance.setUserId(1); // Set test user ID
        attendance.setDate(Date.valueOf("2023-12-01")); // Set test date
        attendance.setStatus("present"); // Set test status

        // Mock database behavior
        when(mockConnection.prepareStatement(anyString())).thenReturn(mockStatement);

        // Execute the method under test
        attendance.markAttendance(mockConnection);

        // Verify the database interactions
        verify(mockConnection).prepareStatement("INSERT INTO attendance (user_id, date,
status) VALUES (?, ?, ?)");
        verify(mockStatement).setInt(1, 1);
        verify(mockStatement).setDate(2, Date.valueOf("2023-12-01"));
        verify(mockStatement).setString(3, "present");
        verify(mockStatement).executeUpdate();
    }

    // Test for viewing attendance
    @Test
```

```
    void testViewAttendance() throws SQLException {
        int userId = 1; // Test user ID

        // Mock database behavior
        when(mockConnection.prepareStatement(anyString())).thenReturn(mockStatement);
        when(mockStatement.executeQuery()).thenReturn(mockResultSet);

        // Simulate result set data
        when(mockResultSet.next()).thenReturn(true, false); // First call returns true, second
returns false
        when(mockResultSet.getInt("attendance_id")).thenReturn(101);
        when(mockResultSet.getInt("user_id")).thenReturn(userId);
        when(mockResultSet.getDate("date")).thenReturn(Date.valueOf("2023-12-01"));
        when(mockResultSet.getString("status")).thenReturn("present");

        // Execute the method under test
        List<Attendance> attendanceList = Attendance.viewAttendance(mockConnection, userId);

        // Verify the database interactions
        verify(mockConnection).prepareStatement("SELECT * FROM attendance WHERE user_id
= ?");
        verify(mockStatement).setInt(1, userId);
        verify(mockStatement).executeQuery();

        // Assertions to validate the retrieved attendance data
        assertNotNull(attendanceList);
        assertEquals(1, attendanceList.size());
        Attendance attendance = attendanceList.get(0);
        assertEquals(101, attendance.getAttendanceId());
        assertEquals(userId, attendance.getUserId());
        assertEquals(Date.valueOf("2023-12-01"), attendance.getDate());
        assertEquals("present", attendance.getStatus());
    }
}
```

2. Fees.java

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.sql.*;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

class FeesTest {
```

```
    private Connection mockConnection; // Mocked database connection
    private PreparedStatement mockStatement; // Mocked PreparedStatement
    private ResultSet mockResultSet; // Mocked ResultSet

    @BeforeEach
    void setUp() throws SQLException {
        // Initialize mock objects before each test
        mockConnection = mock(Connection.class);
        mockStatement = mock(PreparedStatement.class);
        mockResultSet = mock(ResultSet.class);
    }

    // Test for inserting fees into the database
    @Test
    void testCalculateFees() throws SQLException {
        Fees fees = new Fees();
        fees.setUserId(1); // Test user ID
        fees.setTotalFees(1000.0); // Test total fees
        fees.setPaid(500.0); // Test paid amount
        fees.setDue(500.0); // Test due amount

        // Mock database behavior
        when(mockConnection.prepareStatement(anyString())).thenReturn(mockStatement);

        // Execute the method under test
        fees.calculateFees(mockConnection);

        // Verify the database interactions
        verify(mockConnection).prepareStatement("INSERT INTO fees (user_id, total_fees, paid,
due) VALUES (?, ?, ?, ?)");
        verify(mockStatement).setInt(1, 1);
        verify(mockStatement).setDouble(2, 1000.0);
        verify(mockStatement).setDouble(3, 500.0);
        verify(mockStatement).setDouble(4, 500.0);
        verify(mockStatement).executeUpdate();
    }

    // Test for fetching a fee record from the database
    @Test
    void testViewFees() throws SQLException {
        int userId = 1; // Test user ID

        // Mock database behavior
```

```
        when(mockConnection.prepareStatement(anyString())).thenReturn(mockStatement);
        when(mockStatement.executeQuery()).thenReturn(mockResultSet);

        // Simulate result set data
        when(mockResultSet.next()).thenReturn(true, false); // First call returns true, second
returns false
        when(mockResultSet.getInt("fee_id")).thenReturn(101);
        when(mockResultSet.getInt("user_id")).thenReturn(userId);
        when(mockResultSet.getDouble("total_fees")).thenReturn(1000.0);
        when(mockResultSet.getDouble("paid")).thenReturn(500.0);
        when(mockResultSet.getDouble("due")).thenReturn(500.0);

        // Execute the method under test
        Fees fees = Fees.viewFees(mockConnection, userId);

        // Verify the database interactions
        verify(mockConnection).prepareStatement("SELECT * FROM fees WHERE user_id = ?");
        verify(mockStatement).setInt(1, userId);
        verify(mockStatement).executeQuery();

        // Assertions to validate the retrieved fees data
        assertNotNull(fees);
        assertEquals(101, fees.getFeeId());
        assertEquals(userId, fees.getUserId());
        assertEquals(1000.0, fees.getTotalFees());
        assertEquals(500.0, fees.getPaid());
        assertEquals(500.0, fees.getDue());
    }

    // Test for non-existent user fee record
    @Test
    void testViewFeesNoRecord() throws SQLException {
        int userId = 999; // Non-existent user ID

        // Mock database behavior
        when(mockConnection.prepareStatement(anyString())).thenReturn(mockStatement);
        when(mockStatement.executeQuery()).thenReturn(mockResultSet);

        // Simulate empty result set
        when(mockResultSet.next()).thenReturn(false);

        // Execute the method under test
        Fees fees = Fees.viewFees(mockConnection, userId);
```

```
        // Verify the database interactions
        verify(mockConnection).prepareStatement("SELECT * FROM fees WHERE user_id = ?");
        verify(mockStatement).setInt(1, userId);
        verify(mockStatement).executeQuery();

        // Assert that no fees record is found
        assertNull(fees);
    }
}
```

3. Marks.java

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import static org.mockito.Mockito.*;

class MarksTest {

    private Connection mockConnection; // Mocked database connection
    private PreparedStatement mockStatement; // Mocked PreparedStatement

    @BeforeEach
    void setUp() throws SQLException {
        // Initialize mock objects before each test
        mockConnection = mock(Connection.class);
        mockStatement = mock(PreparedStatement.class);

        // Mock the behavior of the connection's prepareStatement method
        when(mockConnection.prepareStatement(anyString())).thenReturn(mockStatement);
    }

    @Test
    void testEnterMarks() throws SQLException {
        Marks marks = new Marks();
        marks.setUserId(1); // Test user ID
        marks.setSubject("Mathematics"); // Test subject
        marks.setMarks(95); // Test marks

        // Execute the method under test
        marks.enterMarks(mockConnection);
```

```
        // Verify the interactions with the database
        verify(mockConnection).prepareStatement("INSERT INTO marks (user_id, subject, marks)
VALUES (?, ?, ?)");
        verify(mockStatement).setInt(1, 1);
        verify(mockStatement).setString(2, "Mathematics");
        verify(mockStatement).setInt(3, 95);
        verify(mockStatement).executeUpdate();
    }
}
```

4.User.java

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import static org.mockito.Mockito.*;

class UserTest {

    private Connection mockConnection; // Mocked database connection
    private PreparedStatement mockStatement; // Mocked PreparedStatement
    private ResultSet mockResultSet; // Mocked ResultSet

    @BeforeEach
    void setUp() throws SQLException {
        // Initialize mock objects before each test
        mockConnection = mock(Connection.class);
        mockStatement = mock(PreparedStatement.class);
        mockResultSet = mock(ResultSet.class);

        // Mock the behavior of the connection's prepareStatement method
        when(mockConnection.prepareStatement(anyString(),
anyInt())).thenReturn(mockStatement);
        when(mockConnection.prepareStatement(anyString())).thenReturn(mockStatement);
        when(mockStatement.getGeneratedKeys()).thenReturn(mockResultSet);
    }

    @Test
    void testAddUser() throws SQLException {
        // Setup
```

```
        User user = new User();
        user.setFirstName("John");
        user.setLastName("Doe");
        user.setAge(30);
        user.setDob(java.sql.Date.valueOf("1993-05-20"));
        user.setAddress("123 Main St");
        user.setPhone("1234567890");
        user.setEmail("john.doe@example.com");
        user.setRole("student");
        user.setCourse("Computer Science");
        user.setBranch("Software Engineering");

        when(mockResultSet.next()).thenReturn(true);
        when(mockResultSet.getInt(1)).thenReturn(101); // Simulating generated user ID

        // Execute
        user.addUser(mockConnection);

        // Verify
        verify(mockConnection).prepareStatement(anyString(),
eq(PreparedStatement.RETURN_GENERATED_KEYS));
        verify(mockStatement).setString(1, "John");
        verify(mockStatement).setString(2, "Doe");
        verify(mockStatement).setInt(3, 30);
        verify(mockStatement).setDate(4, java.sql.Date.valueOf("1993-05-20"));
        verify(mockStatement).setString(5, "123 Main St");
        verify(mockStatement).setString(6, "1234567890");
        verify(mockStatement).setString(7, "john.doe@example.com");
        verify(mockStatement).setString(8, "student");
        verify(mockStatement).setString(9, "Computer Science");
        verify(mockStatement).setString(10, "Software Engineering");
        verify(mockStatement).executeUpdate();
        verify(mockResultSet).next();
    }

    @Test
    void testUpdateUser() throws SQLException {
        // Setup
        User user = new User();
        user.setUserId(101); // Existing user ID
        user.setFirstName("Jane");
        user.setLastName("Doe");
        user.setAge(29);
        user.setDob(java.sql.Date.valueOf("1994-06-15"));
```

```
        user.setAddress("456 Another St");
        user.setPhone("0987654321");
        user.setEmail("jane.doe@example.com");
        user.setCourse("Data Science");
        user.setBranch("AI");

        // Execute
        user.updateUser(mockConnection);

        // Verify
        verify(mockConnection).prepareStatement(anyString());
        verify(mockStatement).setString(1, "Jane");
        verify(mockStatement).setString(2, "Doe");
        verify(mockStatement).setInt(3, 29);
        verify(mockStatement).setDate(4, java.sql.Date.valueOf("1994-06-15"));
        verify(mockStatement).setString(5, "456 Another St");
        verify(mockStatement).setString(6, "0987654321");
        verify(mockStatement).setString(7, "jane.doe@example.com");
        verify(mockStatement).setString(8, "Data Science");
        verify(mockStatement).setString(9, "AI");
        verify(mockStatement).setInt(10, 101);
        verify(mockStatement).executeUpdate();
    }

    @Test
    void testDeleteUser() throws SQLException {
        // Setup
        User user = new User();
        user.setUserId(102); // User ID to delete

        // Execute
        user.deleteUser(mockConnection);

        // Verify
        verify(mockConnection).prepareStatement(anyString());
        verify(mockStatement).setInt(1, 102);
        verify(mockStatement).executeUpdate();
    }
}
```

1. DatabaseConnection.java

```
import org.junit.jupiter.api.Test;
import java.sql.Connection;
```

```
import java.sql.DriverManager;
import java.sql.SQLException;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

class DatabaseConnectionTest {

    @Test
    void testGetConnectionSuccess() {
        try {
            // Mocking DriverManager to simulate successful connection
            Connection mockConnection = mock(Connection.class);
            mockStatic(DriverManager.class);
            when(DriverManager.getConnection("jdbc:mysql://localhost:3306/university_db", "root",
"9328199824"))
                    .thenReturn(mockConnection);

            Connection connection = DatabaseConnection.getConnection();

            assertNotNull(connection, "Connection should not be null");
            verifyStatic(DriverManager.class);
            DriverManager.getConnection("jdbc:mysql://localhost:3306/university_db", "root",
"9328199824");
        } catch (SQLException e) {
            fail("SQLException was not expected");
        }
    }

    @Test
    void testGetConnectionDriverNotFound() {
        try {
            // Mocking Class.forName to throw ClassNotFoundException
            mockStatic(Class.class);
            doThrow(ClassNotFoundException.class).when(Class.class);
            Class.forName("com.mysql.cj.jdbc.Driver");

            RuntimeException exception = assertThrows(RuntimeException.class,
DatabaseConnection::getConnection);
            assertEquals("MySQL Driver not found!", exception.getMessage());
        } catch (ClassNotFoundException e) {
            fail("ClassNotFoundException was not expected");
        }
    }
```

```
    @Test
    void testGetConnectionSQLException() {
        try {
            // Mocking DriverManager to throw SQLException
            mockStatic(DriverManager.class);
            when(DriverManager.getConnection("jdbc:mysql://localhost:3306/university_db", "root",
"9328199824"))
                    .thenThrow(SQLException.class);

            RuntimeException exception = assertThrows(RuntimeException.class,
DatabaseConnection::getConnection);
            assertEquals("Unable to connect to the database!", exception.getMessage());
        } catch (SQLException e) {
            fail("SQLException was not expected");
        }
    }
}
```

1. AdminController.java

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import view.AdminView;
import model.User;
import model.Attendance;

import javax.swing.*;
import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

class AdminControllerTest {
    private AdminView mockView;
    private Connection mockConnection;
    private PreparedStatement mockPreparedStatement;
    private ResultSet mockResultSet;
    private AdminController adminController;
```

```
    private final int adminUserId = 1;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
        adminController = new AdminController(mockView, mockConnection, adminUserId);
    }

    @Test
    void testAddOrUpdateUser_Success() throws Exception {
        // Mock user input fields
        when(mockView.getFirstNameField().getText()).thenReturn("John");
        when(mockView.getLastNameField().getText()).thenReturn("Doe");
        when(mockView.getAgeField().getText()).thenReturn("25");
        when(mockView.getDobField().getText()).thenReturn("2000-01-01");
        when(mockView.getAddressField().getText()).thenReturn("123 Main St");
        when(mockView.getPhoneField().getText()).thenReturn("1234567890");
        when(mockView.getEmailField().getText()).thenReturn("john.doe@example.com");
        when(mockView.getRoleField().getText()).thenReturn("student");
        when(mockView.getCourseField().getText()).thenReturn("CS");
        when(mockView.getBranchField().getText()).thenReturn("SE");

        when(mockConnection.prepareStatement(anyString(),
eq(PreparedStatement.RETURN_GENERATED_KEYS))).thenReturn(mockPreparedStatement)
;

        adminController.addOrUpdateUser();

        verify(mockPreparedStatement, times(1)).executeUpdate();
        verify(mockView, times(1)).getFirstNameField();
        verify(mockView, times(1)).getLastNameField();
    }

    @Test
    void testDeleteUser_Success() throws Exception {
        // Mock table selection
        JTable mockTable = mock(JTable.class);
        when(mockView.getUsersTable()).thenReturn(mockTable);
        when(mockTable.getSelectedRow()).thenReturn(0);
        when(mockTable.getValueAt(0, 0)).thenReturn(2); // User ID 2

when(mockConnection.prepareStatement(anyString())).thenReturn(mockPreparedStatement);

        adminController.deleteUser();
```

```
        verify(mockPreparedStatement, times(1)).executeUpdate();
        verify(mockTable, times(1)).getSelectedRow();
    }

    @Test
    void testManageAttendance_Success() throws Exception {
        // Mock user data

when(mockConnection.prepareStatement(anyString())).thenReturn(mockPreparedStatement);
        when(mockPreparedStatement.executeQuery()).thenReturn(mockResultSet);
        when(mockResultSet.next()).thenReturn(true, false); // One user
        when(mockResultSet.getInt("user_id")).thenReturn(2);
        when(mockResultSet.getString("first_name")).thenReturn("John");
        when(mockResultSet.getString("last_name")).thenReturn("Doe");

        // Mock JOptionPane input
        JComboBox<String> mockComboBox = mock(JComboBox.class);
        when(mockComboBox.getSelectedItem()).thenReturn("present");

        JTextField mockDateField = mock(JTextField.class);
        when(mockDateField.getText()).thenReturn("2023-01-01");

        adminController.manageAttendance();

        verify(mockPreparedStatement, times(1)).executeUpdate();
        verify(mockResultSet, times(1)).next();
    }
}
```

2. LoginController.java

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import view.LoginView;
import view.AdminView;
import view.TeacherView;
import view.StudentView;
import model.User;
import javax.swing.*;
import java.sql.Connection;

import static org.mockito.Mockito.*;
```

```
class LoginControllerTest {

    @Mock
    private LoginView mockLoginView; // Mock LoginView

    @Mock
    private Connection mockConnection; // Mock Database Connection
    private LoginController loginController;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
        loginController = new LoginController(mockLoginView, mockConnection);
    }

    @Test
    void testLoginAsAdmin() throws Exception {
        // Mock user data for admin
        when(mockLoginView.getEmailField().getText()).thenReturn("admin@example.com");
        User mockUser = mock(User.class);
        when(mockUser.getRole()).thenReturn("admin");
        when(mockUser.getUserId()).thenReturn(1);
        when(User.getUserByEmail(mockConnection,
"admin@example.com")).thenReturn(mockUser);

        // Trigger login action
        loginController.actionPerformed(new ActionEvent(mockLoginView.getLoginButton(),
ActionEvent.ACTION_PERFORMED, "Login"));

        verify(mockLoginView).dispose();
        verify(mockUser, times(1)).getRole();
    }

    @Test
    void testLoginAsTeacher() throws Exception {
        // Mock user data for teacher
        when(mockLoginView.getEmailField().getText()).thenReturn("teacher@example.com");
        User mockUser = mock(User.class);
        when(mockUser.getRole()).thenReturn("teacher");
        when(mockUser.getUserId()).thenReturn(2);
        when(User.getUserByEmail(mockConnection,
"teacher@example.com")).thenReturn(mockUser);
```

```
        // Trigger login action
        loginController.actionPerformed(new ActionEvent(mockLoginView.getLoginButton(),
ActionEvent.ACTION_PERFORMED, "Login"));

        verify(mockLoginView).dispose();
        verify(mockUser, times(1)).getRole();
    }

    @Test
    void testLoginUserNotFound() throws Exception {
        // Mock no user found
        when(mockLoginView.getEmailField().getText()).thenReturn("unknown@example.com");
        when(User.getUserByEmail(mockConnection,
"unknown@example.com")).thenReturn(null);

        // Trigger login action
        loginController.actionPerformed(new ActionEvent(mockLoginView.getLoginButton(),
ActionEvent.ACTION_PERFORMED, "Login"));

        verify(mockLoginView, never()).dispose();
        verify(mockLoginView, times(1)).getEmailField();
    }
}
```

3. StudentController.java

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import view.StudentView;
import model.User;
import model.Marks;
import model.Attendance;
import model.Fees;

import javax.swing.*;
import java.sql.Connection;
import java.sql.Date;
import java.util.Arrays;
import java.util.List;

import static org.mockito.Mockito.*;
```

```
class StudentControllerTest {

    @Mock
    private StudentView mockView; // Mock StudentView
    @Mock
    private Connection mockConnection; // Mock Database Connection

    private StudentController studentController;
    private int studentUserId = 1;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
        studentController = new StudentController(mockView, mockConnection, studentUserId);
    }

    @Test
    void testViewPersonalDetails() throws Exception {
        // Mock User data
        User mockUser = mock(User.class);
        when(User.getUserById(mockConnection, studentUserId)).thenReturn(mockUser);
        when(mockUser.getFirstName()).thenReturn("John");
        when(mockUser.getLastName()).thenReturn("Doe");
        when(mockUser.getAge()).thenReturn(20);
        when(mockUser.getDob()).thenReturn(Date.valueOf("2003-01-01"));
        when(mockUser.getAddress()).thenReturn("123 Street");
        when(mockUser.getPhone()).thenReturn("1234567890");
        when(mockUser.getEmail()).thenReturn("john.doe@example.com");
        when(mockUser.getCourse()).thenReturn("CS");
        when(mockUser.getBranch()).thenReturn("IT");

        studentController.actionPerformed(new
ActionEvent(mockView.getViewPersonalDetailsButton(), ActionEvent.ACTION_PERFORMED,
"View Personal Details"));

        verify(mockView).setPersonalDetails("John Doe", 20, "2003-01-01", "123 Street",
"1234567890", "john.doe@example.com", "CS", "IT");
    }

    @Test
    void testViewMarks() throws Exception {
        // Mock Marks data
        List<Marks> mockMarks = Arrays.asList(
                new Marks() {{ setSubject("Math"); setMarks(85); }},
```

```
            new Marks() {{ setSubject("Science"); setMarks(90); }}
        );
        when(Marks.viewAllMarksForUser(mockConnection,
studentUserId)).thenReturn(mockMarks);

        studentController.actionPerformed(new ActionEvent(mockView.getViewMarksButton(),
ActionEvent.ACTION_PERFORMED, "View Marks"));

        verify(mockView, times(1)).getViewMarksButton();
    }

    @Test
    void testViewFees() throws Exception {
        // Mock Fees data
        Fees mockFees = mock(Fees.class);
        when(Fees.viewFees(mockConnection, studentUserId)).thenReturn(mockFees);
        when(mockFees.getTotalFees()).thenReturn(1000.0);
        when(mockFees.getPaid()).thenReturn(700.0);
        when(mockFees.getDue()).thenReturn(300.0);

        studentController.actionPerformed(new ActionEvent(mockView.getViewFeesButton(),
ActionEvent.ACTION_PERFORMED, "View Fees"));

        verify(mockView, times(1)).getViewFeesButton();
        verify(mockView, never()).getViewPersonalDetailsButton();
    }
}
```

4. TeacherController.java

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import view.TeacherView;
import model.User;
import model.Attendance;
import model.Marks;

import javax.swing.*;
import java.sql.Connection;
import java.sql.Date;

import static org.mockito.Mockito.*;
```

```
class TeacherControllerTest {

    @Mock
    private TeacherView mockView; // Mock TeacherView
    @Mock
    private Connection mockConnection; // Mock database connection

    private TeacherController teacherController;
    private int teacherUserId = 1;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
        teacherController = new TeacherController(mockView, mockConnection, teacherUserId);
    }

    @Test
    void testViewPersonalDetails() throws Exception {
        // Mock User details
        User mockUser = mock(User.class);
        when(User.getUserById(mockConnection, teacherUserId)).thenReturn(mockUser);
        when(mockUser.getFirstName()).thenReturn("John");
        when(mockUser.getLastName()).thenReturn("Doe");
        when(mockUser.getAge()).thenReturn(35);
        when(mockUser.getDob()).thenReturn(Date.valueOf("1988-05-15"));
        when(mockUser.getAddress()).thenReturn("123 Teacher Street");
        when(mockUser.getPhone()).thenReturn("1234567890");
        when(mockUser.getEmail()).thenReturn("john.doe@school.com");
        when(mockUser.getCourse()).thenReturn("Mathematics");
        when(mockUser.getBranch()).thenReturn("High School");

        teacherController.actionPerformed(new
ActionEvent(mockView.getViewPersonalDetailsButton(), ActionEvent.ACTION_PERFORMED,
"View Personal Details"));

        verify(mockView).getFirstNameField().setText("John");
        verify(mockView).getLastNameField().setText("Doe");
        verify(mockView).getAgeField().setText("35");
        verify(mockView).getDobField().setText("1988-05-15");
        verify(mockView).getAddressField().setText("123 Teacher Street");
        verify(mockView).getPhoneField().setText("1234567890");
        verify(mockView).getEmailField().setText("john.doe@school.com");
        verify(mockView).getCourseField().setText("Mathematics");
```

```
        verify(mockView).getBranchField().setText("High School");
    }

    @Test
    void testManageAttendance() throws Exception {
        // Simulating attendance marking
        when(mockView.getManageAttendanceButton()).thenReturn(new JButton());
        Attendance mockAttendance = mock(Attendance.class);

        teacherController.actionPerformed(new
ActionEvent(mockView.getManageAttendanceButton(), ActionEvent.ACTION_PERFORMED,
"Manage Attendance"));

        verify(mockAttendance, never()).markAttendance(mockConnection); // GUI interactions
need simulation to test full behavior
    }

    @Test
    void testEnterMarks() throws Exception {
        // Mock marks entry
        when(mockView.getStudentIdField()).thenReturn(new JTextField("2"));
        when(mockView.getSubjectField()).thenReturn(new JTextField("Math"));
        when(mockView.getMarksField()).thenReturn(new JTextField("95"));

        Marks mockMarks = mock(Marks.class);

        teacherController.actionPerformed(new ActionEvent(mockView.getEnterMarksButton(),
ActionEvent.ACTION_PERFORMED, "Enter Marks"));

        verify(mockMarks, never()).enterMarks(mockConnection); // GUI interactions need
simulation to test full behavior
    }
}
```