SOFTWARE QUALITY ENGINEERING

(Course Project)

FAST NUCES ISLAMABAD

Group Members

Zain Ali Khan

Mohammad Abdullah Khan Durrani

Noor ul Baseer

1. **Add Employee GUI Test Script**

```
package hotel.management.system.automatedTests;

import hotel.management.system.AddEmployee;
import org.assertj.swing.core.matcher.JButtonMatcher;
import org.assertj.swing.core.matcher.JTextComponentMatcher;
import org.assertj.swing.fixture.FrameFixture;
import org.junit.*;

import javax.swing.*;

public class AddEmployeeGUITest {

    private FrameFixture window;

    // Setup method to initialize the GUI frame and make it ready for
testing
    @Before
    public void setUp() {
        AddEmployee frame = new AddEmployee(); // Instantiate the
AddEmployee frame
        window = new FrameFixture(frame); // Bind the frame with
AssertJ Swing's FrameFixture
        window.show(); // Display the GUI frame
    }

    // Cleanup method to close the window and release resources
    @After
    public void tearDown() {
        window.cleanUp(); // Close the frame and clean up the test
environment
    }

    // Test to verify that the name field accepts input correctly
    @Test
```

```
    public void testNameField() {
        window.textBox("nameField").enterText("John Doe"); // Simulate
typing "John Doe" in the name field
        Assert.assertEquals("John Doe",
window.textBox("nameField").text()); // Assert the field contains the
expected text
    }

    // Test to verify the age field accepts input correctly
    @Test
    public void testAgeField() {
        window.textBox("ageField").enterText("25"); // Simulate typing
"25" in the age field
        Assert.assertEquals("25", window.textBox("ageField").text());
// Assert the field contains the expected text
    }

    // Test to ensure gender selection via radio buttons works
correctly
    @Test
    public void testGenderSelection() {
        window.radioButton("maleRadioButton").click(); // Simulate
selecting the male radio button

Assert.assertTrue(window.radioButton("maleRadioButton").target().isSel
ected()); // Verify the male radio button is selected
    }

    // Test to ensure job role selection from the combo box works as
expected
    @Test
    public void testJobSelection() {
        window.comboBox("jobComboBox").selectItem("Manager"); //
Simulate selecting "Manager" from the combo box
        Assert.assertEquals("Manager",
window.comboBox("jobComboBox").selectedItem()); // Assert the selected
item is "Manager"
    }
```

```
    // Test to verify the Save button functionality after filling out
all fields
    @Test
    public void testSaveButton() {
        // Fill out all required fields
        window.textBox("nameField").enterText("John Doe");
        window.textBox("ageField").enterText("30");
        window.radioButton("maleRadioButton").click();
        window.comboBox("jobComboBox").selectItem("Manager");
        window.textBox("salaryField").enterText("50000");
        window.textBox("phoneField").enterText("1234567890");
        window.textBox("aadharField").enterText("123456789012");

window.textBox("emailField").enterText("john.doe@example.com");

        // Simulate clicking the Save button
        window.button(JButtonMatcher.withName("saveButton")).click();

        // Optional delay to ensure the dialog appears
        try {
            Thread.sleep(1000); // Adjust this value as needed based
on system performance
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Retrieve the dialog's message
        String dialogMessage = window.dialog().label().text();
        System.out.println("Dialog message: " + dialogMessage);

        // Assert that the dialog contains the success message
        Assert.assertTrue(dialogMessage.contains("Employee Added
Successfully"));
    }

}
```

Explanation of the Script

This Java script automates the testing of the `AddEmployee` GUI in a hotel management system using the AssertJ Swing framework. Each test focuses on verifying the correctness and functionality of individual GUI components:

1. **Setup and Teardown**: The `setUp` method initializes the `AddEmployee` frame for testing, and the `tearDown` method ensures proper cleanup after each test.
2. **Field Input Verification**:
    - **`testNameField` and `testAgeField`**: Validate that the name and age fields accept user inputs correctly by simulating text entry and comparing the actual input with the expected values.
3. **Component Interaction**:
    - **`testGenderSelection`**: Ensures the gender selection functionality works by checking if a radio button is correctly selected upon user interaction.
    - **`testJobSelection`**: Verifies the selection of job roles from a combo box and confirms the selected item.
4. **Save Button Test**:
    - **`testSaveButton`**: Simulates filling all required fields, selecting options, and clicking the save button. It then checks if the system responds with a success message dialog.

2. Add Room GUI Test Script

```
package hotel.management.system.automatedTests;

import hotel.management.system.AddRoom;

import org.assertj.swing.core.matcher.JButtonMatcher;
import org.assertj.swing.fixture.FrameFixture;
import org.assertj.swing.fixture.JComboBoxFixture;
import org.assertj.swing.fixture.JTextComponentFixture;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class AddRoomGUITest {
    private FrameFixture window;

    // Setup method to initialize and display the AddRoom GUI for testing
```

```
    @Before
    public void setUp() {
        AddRoom frame = new AddRoom(); // Instantiate the AddRoom frame
        frame.setName("frame3"); // Assign a name for easier identification
        frame.setVisible(true); // Make the frame visible for interaction
        window = new FrameFixture(frame); // Bind the frame to AssertJ Swing's FrameFixture
    }

    // Cleanup method to close the window and release resources after each test
    @After
    public void tearDown() {
        window.cleanUp(); // Close the GUI and cleanup resources
    }

    // Test to verify the Room Number field behavior
    @Test
    public void testRoomNumberField() {
        JTextComponentFixture roomNumberField = window.textBox("roomNumberField"); //
Reference the Room Number field
        roomNumberField.requireEnabled(); // Ensure the field is enabled
        roomNumberField.requireEmpty(); // Ensure the field is initially empty

        // Simulate entering text in the Room Number field
        roomNumberField.enterText("101");
        roomNumberField.requireText("101"); // Verify the entered text is correct
    }

    // Test to validate the Availability ComboBox behavior
    @Test
    public void testAvailabilityComboBox() {
        JComboBoxFixture availabilityComboBox = window.comboBox("availabilityComboBox"); //
Reference the Availability ComboBox
        availabilityComboBox.requireEnabled(); // Ensure the combo box is enabled
        availabilityComboBox.requireSelection("Available"); // Verify the default selection is
"Available"

        // Simulate selecting a different option
        availabilityComboBox.selectItem("Occupied");
        availabilityComboBox.requireSelection("Occupied"); // Verify the selected item is
"Occupied"
    }

    // Test to validate the Cleaning Status ComboBox functionality
    @Test
```

```
    public void testCleaningStatusComboBox() {
        JComboBoxFixture cleaningStatusComboBox =
window.comboBox("cleaningStatusComboBox"); // Reference the Cleaning Status ComboBox
        cleaningStatusComboBox.requireEnabled(); // Ensure the combo box is enabled
        cleaningStatusComboBox.requireSelection("Cleaned"); // Verify the default selection is
"Cleaned"

        // Simulate selecting a different option
        cleaningStatusComboBox.selectItem("Dirty");
        cleaningStatusComboBox.requireSelection("Dirty"); // Verify the selected item is "Dirty"
    }

    // Test to verify the Price field behavior
    @Test
    public void testPriceField() {
        JTextComponentFixture priceField = window.textBox("priceField"); // Reference the Price
field
        priceField.requireEnabled(); // Ensure the field is enabled
        priceField.requireEmpty(); // Ensure the field is initially empty

        // Simulate entering a price
        priceField.enterText("5000");
        priceField.requireText("5000"); // Verify the entered price is correct
    }

    // Test to validate the Bed Type ComboBox functionality
    @Test
    public void testBedTypeComboBox() {
        JComboBoxFixture bedTypeComboBox = window.comboBox("bedTypeComboBox"); //
Reference the Bed Type ComboBox
        bedTypeComboBox.requireEnabled(); // Ensure the combo box is enabled
        bedTypeComboBox.requireSelection("Single Bed"); // Verify the default selection is "Single
Bed"

        // Simulate selecting a different bed type
        bedTypeComboBox.selectItem("Double Bed");
        window.robot().waitForIdle(); // Wait for UI updates if needed
        bedTypeComboBox.requireSelection("Double Bed"); // Verify the selected item is "Double
Bed"

        // Debugging: Print the selected item
        System.out.println("Selected item after change: " + bedTypeComboBox.selectedItem());
    }
```

```
    // Test to verify the Add button functionality
    @Test
    public void testAddButton() {
        window.button("addButton").requireEnabled().requireVisible(); // Ensure the Add button is
enabled and visible

        // Simulate clicking the Add button
        window.button("addButton").click();

        // Additional checks or mock verifications can be added if needed
    }

    // Test to verify the Back button functionality
    @Test
    public void testBackButton() {
        window.button("backButton").requireEnabled().requireVisible(); // Ensure the Back button is
enabled and visible

        // Simulate clicking the Back button
        window.button("backButton").click();

        // Additional checks or mock verifications can be added if needed
    }
}
```

Explanation of the Script

This script automates GUI testing for the `AddRoom` module of a hotel management system. It utilizes the AssertJ Swing framework to simulate user interactions with the `AddRoom` form and verify its behavior:

1. **Setup and Teardown**:
    - The `setUp` method initializes the `AddRoom` GUI frame and binds it to the `FrameFixture` for testing.
    - The `tearDown` method ensures that the GUI window is closed and resources are released after each test.
2. **Field and Component Tests**:
    - **`testRoomNumberField`**: Ensures the Room Number field is enabled, initially empty, and accepts correct input.

- `testAvailabilityComboBox`: Verifies that the Availability ComboBox is functional and allows selecting between "Available" and "Occupied".
- `testCleaningStatusComboBox`: Checks that the Cleaning Status ComboBox behaves as expected, allowing selection between "Cleaned" and "Dirty".
- `testPriceField`: Confirms that the Price field accepts numeric input correctly.
- `testBedTypeComboBox`: Validates the Bed Type ComboBox functionality, ensuring it allows switching between "Single Bed" and "Double Bed".

3. **Button Functionality**:

   - `testAddButton`: Verifies that the Add button is visible, enabled, and clickable. Additional checks can confirm the expected outcome of the click.
   - `testBackButton`: Ensures the Back button is functional and performs the intended navigation or action.

3. Login GUI Test Script

```
package hotel.management.system.automatedTests;

import hotel.management.system.Login;

import org.assertj.swing.core.matcher.JButtonMatcher;
import org.assertj.swing.fixture.FrameFixture;
import org.assertj.swing.fixture.JTextComponentFixture;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import javax.swing.*;
import java.awt.event.ActionListener;

public class LoginGUITest {
    private FrameFixture window;

    // Setup method to initialize and display the Login GUI for testing
    @Before
    public void setUp() {
        Login frame = new Login(); // Instantiate the Login frame
        frame.setName("loginFrame"); // Assign a name for easier identification
        frame.setVisible(true); // Make the frame visible for interaction
```

```
        window = new FrameFixture(frame); // Bind the frame to AssertJ Swing's FrameFixture
    }

    // Cleanup method to close the window and release resources after each test
    @After
    public void tearDown() {
        window.cleanUp(); // Close the GUI and cleanup resources
    }

    // Test to verify the Username field behavior
    @Test
    public void testUsernameField() {
        JTextComponentFixture usernameField = window.textBox("usernameField"); // Reference
the Username field
        usernameField.requireEnabled(); // Ensure the field is enabled
        usernameField.requireEmpty(); // Ensure the field is initially empty

        // Simulate entering a username
        usernameField.enterText("testUser");
        usernameField.requireText("testUser"); // Verify the entered username is correct
    }

    // Test to verify the Password field behavior
    @Test
    public void testPasswordField() {
        JTextComponentFixture passwordField = window.textBox("passwordField"); // Reference
the Password field
        passwordField.requireEnabled(); // Ensure the field is enabled
        passwordField.requireEmpty(); // Ensure the field is initially empty

        // Simulate entering a password
        passwordField.enterText("testPass");
        passwordField.requireText("testPass"); // Verify the entered password is correct
    }

    // Test to verify the Login button functionality
    @Test
    public void testLoginButton() {
        window.button("loginButton").requireEnabled().requireVisible(); // Ensure the Login button
is enabled and visible

        // Simulate a login action
        window.textBox("usernameField").enterText("admin");
        window.textBox("passwordField").enterText("admin123");
```

```
        window.button("loginButton").click();

        // Additional assertions can verify the frame transition or mock database behavior
    }

    // Test to verify the Cancel button functionality
    @Test
    public void testCancelButton() {
        Login loginFrame = (Login) window.target(); // Get the Login frame instance
        JButton cancelButton = loginFrame.getCancelButton(); // Retrieve the Cancel button
reference

        // Mock the Cancel button's action to prevent System.exit()
        for (ActionListener listener : cancelButton.getActionListeners()) {
            cancelButton.removeActionListener(listener); // Remove original listeners
        }
        cancelButton.addActionListener(e -> System.out.println("Mock Cancel Action Triggered"));
// Add mock behavior

        // Simulate clicking the Cancel button
        window.button("cancelButton").click();

        // Validate expected behavior
        System.out.println("Cancel button test completed without affecting other tests.");
    }

    // Custom SecurityManager to prevent System.exit() from terminating the JVM during tests
    public static class NoExitSecurityManager extends SecurityManager {
        @Override
        public void checkPermission(java.security.Permission perm) {
            // Allow all permissions except System.exit()
            if (perm.getName().startsWith("exitVM")) {
                throw new SecurityException("Exit is not allowed in test");
            }
        }
    }
}
```

Key Points:

1. **Setup and Teardown**:
    - Initializes the `Login` frame for each test and ensures cleanup afterward.

2. **Field Tests**:
    - Verifies that username and password fields are enabled, initially empty, and accept input correctly.
3. **Button Tests**:
    - Ensures the Login button is visible, enabled, and functional.
    - Validates the Cancel button with a mocked action to avoid terminating the JVM via `System.exit`.
4. **Custom SecurityManager**:
    - Used to override `System.exit` calls during the tests, ensuring the JVM does not terminate when testing the Cancel button.