PatientDashboard Unit Test Specifications

Document Purpose: This document outlines the test specifications for the PatientDashboard class unit tests in the gp_booking_system package. These tests ensure the functionalities of the PatientDashboard class work as intended.

1. Overall Test Strategy:

The test strategy utilizes JUnit and Mockito frameworks for mocking objects and verifying interactions.

- Mocks are used for objects like Connection, Statement, ResultSet, Patient, and Doctor.
- Tests cover functionalities like arranging booking, sending confirmation emails, retrieving doctor information, and updating patient doctors.

2. Individual Test Cases:

Test Case 1: testArrangeBooking

- Mocks Patient, Doctor, and PatientDashboard objects.
- Verifies that showDateTimePrompt is called with the mocked doctor when the arrangeBooking method is called.
- Result: Passed

Test Case 2: testShowDateTimePrompt

- Mocks Doctor, Patient, and PatientDashboard objects.
- Stubs methods like isValidDateTime, isDoctorAvailable, bookAppointment, and sendConfirmationMessages to avoid side effects.
- Verifies that internal methods are called with the expected arguments.
- Result: Passed

Test Case 3: testIsDoctorAvailable

- Mocks Connection, PreparedStatement, ResultSet, and Patient objects.
- Simulates a database connection and result set to test whether a doctor is available for a specific date and time.
- Verifies the method returns true when a doctor is available and falseotherwise.
- Result: Passed

Test Case 4: testSendConfirmationMessages

- Mocks PatientDashboard object.
- Verifies that the sendConfirmationEmailToPatient method is called with the expected arguments.
- Result: Passed

Test Case 5: testBookAppointment

- Mocks Patient, Doctor, Connection, and PreparedStatement objects.
- Simulates database interactions and prepared statement execution.
- Verifies that the prepared statement is executed with the expected parameters.
- Result: Passed

Test Case 6: testGetPatientDoctor

- Mocks Connection, PreparedStatement, ResultSet, and Patient objects.
- Simulates a database connection and result set to retrieve a patient's assigned doctor.
- Verifies that the retrieved doctor object has the expected ID.
- Result: Passed

Test Case 7: testGetDoctor (Similar to Test Case 8)

- Mocks Connection, PreparedStatement, ResultSet, and Patient objects.
- Simulates a database connection and result set to retrieve a doctor's information based on their ID.
- Verifies that the retrieved doctor object has the expected name and email.

Test Case 8: testViewBookingsWithInvalidInputFormat

- Mocks JOptionPane to simulate user input.
- Tests the behavior when the user enters an invalid date format for viewing bookings.

Test Case 9: testChangeDoctor

- Mocks Patient, Doctor, and PatientDashboard objects.
- Simulates selecting a new doctor through a mocked dialog.
- Verifies that the updatePatientDoctor method is called with the expected arguments.

Group D(Patient) Theophilus Ogieva: tio5, thpr2, Ws247 and Km769

Test Case 10: testDisplayDoctorSelectionDialog

- Mocks Patient, Doctor, JOptionPane, and JComboBox objects.
- Simulates selecting a doctor from a list displayed in a dialog.
- Verifies that the selected doctor's information is retrieved correctly.

Test Case 11: testUpdatePatientDoctor (Test outcome not provided)

- Mocks Doctor object.
- Tests the logic for updating a patient's doctor (outcome not shown).

Test Case 12: testGetAllDoctors

- Mocks Doctor, Connection, PreparedStatement, and ResultSet objects.
- Simulates retrieving a list of doctors from the database.
- Verifies that the retrieved doctor objects match the expected list.

Test Case 13: testIsDoctorAvailable_NoDoctorAvailable

- Mocks ResultSet object.
- Simulates an empty result set indicating no

Test Results

All tests passed successfully.

-