*package* com.jtdev.shape\_shift;  
  
  
*import* com.google.android.gms.tasks.Task;  
*import* com.google.android.gms.tasks.Tasks;  
*import* com.google.firebase.auth.*AuthResult*;  
*import* com.google.firebase.auth.FirebaseAuth;  
*import* com.google.firebase.auth.FirebaseAuthException;  
*import* com.jtdev.shape\_shift.fragments.Login;  
  
*import* org.junit.Before;  
*import* org.junit.Test;  
*import* org.mockito.Mock;  
*import* org.mockito.MockitoAnnotations;  
  
*import static* org.junit.Assert.*assertEquals*;  
*import static* org.junit.Assert.*assertTrue*;  
*import static* org.mockito.ArgumentMatchers.*eq*;  
*import static* org.mockito.Mockito.*verify*;  
*import static* org.mockito.Mockito.*when*;  
  
*import* android.content.Context;  
  
*import static* org.mockito.Mockito.*mock*;  
  
*public class* ExampleUnitTest {  
  
 *private* Login loginFrag;  
  
 @Mock  
 *private* FirebaseAuth mockFirebaseAuth;  
  
 @Mock  
 *private* Context mockContext;  
  
 @Before  
 *public void* setUp() {  
 MockitoAnnotations.*initMocks*(*this*);  
 loginFrag = *new* Login();  
 }  
  
 @Test  
 *public void* validSignIn()*throws* InterruptedException {  
 *// Mock successful sign-in  
 when*(mockFirebaseAuth.signInWithEmailAndPassword("1@gmail.com", "123123"))  
 .thenReturn(*mock*(Task.*class*));  
  
  
 loginFrag.checkUser("1@gmail.com", "123123", mockFirebaseAuth, mockContext);  
  
  
 Thread.*sleep*(1000);  
  
  
 *verify*(mockFirebaseAuth).signInWithEmailAndPassword("1@gmail.com", "123123");  
 }  
  
 @Test  
 *public void* notValidSignIn() {  
  
 String invalidEmail = "2@gmail.com";  
 String invalidPass = "invalid";  
  
 Task<*AuthResult*> failedTask = Tasks.*forException*(*new* FirebaseAuthException("123", "Wrong Credentials"));  
 *when*(mockFirebaseAuth.signInWithEmailAndPassword(*eq*(invalidEmail), *eq*(invalidPass)))  
 .thenReturn(failedTask);  
  
  
 loginFrag.checkUser(invalidEmail, invalidPass, mockFirebaseAuth, mockContext);  
  
  
 *verify*(mockFirebaseAuth).signInWithEmailAndPassword(invalidEmail, invalidPass);  
  
  
 }  
 @Test  
 *public void* signInWithEmptyCredentials() {  
 String invalidEmail ="";  
 String invalidPass = "";  
  
 Task<*AuthResult*> failedTask = Tasks.*forException*(*new* FirebaseAuthException("123", "Please enter your email and password"));  
 *when*(mockFirebaseAuth.signInWithEmailAndPassword(*eq*(invalidEmail), *eq*(invalidPass)))  
 .thenReturn(failedTask);  
  
  
 loginFrag.checkUser(invalidEmail, invalidPass, mockFirebaseAuth, mockContext);  
  
  
 *verify*(mockFirebaseAuth).signInWithEmailAndPassword(invalidEmail, invalidPass);  
 }  
  
 @Test  
 *public void* signInWithNullCredentials() {  
 String invalidEmail =*null*;  
 String invalidPass = *null*;  
  
 Task<*AuthResult*> failedTask = Tasks.*forException*(*new* FirebaseAuthException("123", "Complete all fields"));  
 *when*(mockFirebaseAuth.signInWithEmailAndPassword(*eq*(invalidEmail), *eq*(invalidPass)))  
 .thenReturn(failedTask);  
  
 loginFrag.checkUser(invalidEmail, invalidPass, mockFirebaseAuth, mockContext);  
  
 *verify*(mockFirebaseAuth).signInWithEmailAndPassword(invalidEmail, invalidPass);  
 }  
  
 @Test  
 *public void* signInWithNullEmail() {  
  
 String invalidEmail =*null*;  
 String invalidPass = "123123";  
  
 Task<*AuthResult*> failedTask = Tasks.*forException*(*new* FirebaseAuthException("123", "Please enter your email"));  
 *when*(mockFirebaseAuth.signInWithEmailAndPassword(*eq*(invalidEmail), *eq*(invalidPass)))  
 .thenReturn(failedTask);  
  
 loginFrag.checkUser(invalidEmail, invalidPass, mockFirebaseAuth, mockContext);  
  
 *verify*(mockFirebaseAuth).signInWithEmailAndPassword(invalidEmail, invalidPass);  
  
 }  
 @Test  
 *public void* signInWithNullPassword() {  
 String invalidEmail ="1@gmail.com";  
 String invalidPass = *null*;  
  
 Task<*AuthResult*> failedTask = Tasks.*forException*(*new* FirebaseAuthException("123", "Please enter your password"));  
 *when*(mockFirebaseAuth.signInWithEmailAndPassword(*eq*(invalidEmail), *eq*(invalidPass)))  
 .thenReturn(failedTask);  
  
 loginFrag.checkUser(invalidEmail, invalidPass, mockFirebaseAuth, mockContext);  
  
 *verify*(mockFirebaseAuth).signInWithEmailAndPassword(invalidEmail, invalidPass);  
 }  
 @Test  
 *public void* testGetUnderweightRecommendation() {  
 class\_bmiCalculator bmiCalculator = *new* class\_bmiCalculator();  
 *// Given  
 float* underHeight = 155f; *//height in cm  
 float* underWeight = 40f; *//weight in kg* String expectedRecommendation = "You are underweight. It's important to maintain a balanced diet. " +  
 "We suggest you to try bulking with proper guidance using our workout plan.";  
 *// When  
 float* bmi = bmiCalculator.calculateBMI(underHeight, underWeight);  
 String recommendation = bmiCalculator.getRecommendation(bmi);  
  
 *// Then  
 assertEquals*(expectedRecommendation, recommendation);  
 }  
  
 @Test  
 *public void* testGetNormalWeightRecommendation() {  
 class\_bmiCalculator bmiCalculator = *new* class\_bmiCalculator();  
 *// Given  
 float* normalHeight = 175f; *//height in cm  
 float* normalWeight = 70f; *//weight in kg* String expectedRecommendation = "Your weight is within a healthy range. Keep up the good work and make sure you get frequent exercise and a healthy diet. " +  
 "Try our training regimen to keep a regular body.";  
 *// When  
 float* bmi = bmiCalculator.calculateBMI(normalHeight, normalWeight);  
 String recommendation = bmiCalculator.getRecommendation(bmi);  
  
 *// Then  
 assertEquals*(expectedRecommendation, recommendation);  
 }  
  
 @Test  
 *public void* testGetOverweightRecommendation() {  
 class\_bmiCalculator bmiCalculator = *new* class\_bmiCalculator();  
 *// Given  
 float* overweightHeight = 175f; *//height in cm  
 float* overweightWeight = 85f; *//weight in kg* String expectedRecommendation = "You have too much weight. To get a healthy weight, " +  
 "think about changing your lifestyle by consuming a healthier food and engaging in more physical activity. " +  
 "Try our exercise program to start your new lifestyle.";  
 *// When  
 float* bmi = bmiCalculator.calculateBMI(overweightHeight, overweightWeight);  
 String recommendation = bmiCalculator.getRecommendation(bmi);  
  
 *// Then  
 assertEquals*(expectedRecommendation, recommendation);  
 }  
  
 @Test  
 *public void* testGetObesityRecommendation() {  
 class\_bmiCalculator bmiCalculator = *new* class\_bmiCalculator();  
 *// Given  
 float* obeseHeight = 175f; *//height in cm  
 float* obeseWeight = 100f; *//weight in kg* String expectedRecommendation = "You are overweight. It's critical to put your health first by making lifestyle adjustments. " +  
 "Using our exercise program can help you lower the health risks linked to obesity.";  
 *// When  
 float* bmi = bmiCalculator.calculateBMI(obeseHeight, obeseWeight);  
 String recommendation = bmiCalculator.getRecommendation(bmi);  
  
 *// Then  
 assertEquals*(expectedRecommendation, recommendation);  
 }  
}  
  
  
  
  
  
 */\*  
  
 @Test  
 // Sign-Up Test  
 public void signup\_isCorrect() {  
  
 AccountService service = new AccountService();  
  
 // Testing Sign-Up Valid Credentials  
 assertTrue(service.signup("newUsername", "StrongPassword123"));  
 // valid if the input text meets the password requirements  
  
 // Testing password requirements  
 assertFalse(service.signup("newUsername", "weak")); // Password too short  
 assertFalse(service.signup("newUsername", "weakpassword")); // No uppercase letters  
 assertFalse(service.signup("newUsername", "PASSWORD123")); // No lowercase letters  
 assertFalse(service.signup("newUsername", "weakpassword")); // No numbers  
 assertFalse(service.signup("newUsername", "WeakPassword")); // No special characters  
  
 // Sign up are invalid if one of the requirement did not meet the Password requirements.  
 }  
  
 @Test  
 // Forgot Password Test  
 public void forgotPassword\_isCorrect() {  
  
 AccountService service = new AccountService();  
 // Testing valid email for password reset  
 assertTrue(service.forgotPassword("user@example.com"));  
  
 // Testing invalid email for password reset  
 assertFalse(service.forgotPassword("invalid-email")); // Invalid email format  
 assertFalse(service.forgotPassword("nonexistent@example.com")); // Email not yet registered  
  
 }  
  
  
 @Test  
 //Test BMI Calculator  
 public void testCalculateBMI() {  
 BMICalculator bmi = new BMICalculator();  
  
 // Test1: Valid input  
 double height = 1.75; // in meters  
 double weight = 70.0; // in kilograms  
 double expectedBMI = 22.86; // expected BMI  
 assertEquals(expectedBMI, bmi.calculateBMI(height, weight), 0.01); // tolerance: 0.01  
  
 // Test2: Negative Height  
 double invalidHeight = -1.75; // invalid height  
 double validWeight = 70.0; // valid weight  
  
 try {  
 bmi.calculateBMI(invalidHeight, validWeight);  
 fail("Expected IllegalArgumentException for negative height");  
 } catch (IllegalArgumentException e) {  
 assertEquals("Height must be greater than zero", e.getMessage());  
 }  
  
 // Test3: Negative Weight  
 double validHeight = 1.75; // valid height  
 double invalidWeight = -70.9; // invalid weight  
  
 try {  
 bmi.calculateBMI(validHeight, invalidWeight);  
 fail("Expected IllegalArgumentException for negative weight");  
 } catch (IllegalArgumentException e) {  
 assertEquals("Weight must be greater than zero", e.getMessage());  
 }  
  
 // Test4: Zero Height  
 double zeroHeight = 0.0; // zero height  
 try {  
 bmi.calculateBMI(zeroHeight, validWeight);  
 fail("Expected IllegalArgumentException for zero height");  
 } catch (IllegalArgumentException e) {  
 assertEquals("Height must be greater than zero", e.getMessage());  
 }  
  
 // Test5: Zero Weight  
 double zeroWeight = 0.0; // zero weight  
 try {  
 bmi.calculateBMI(validHeight, zeroWeight);  
 fail("Expected IllegalArgumentException for zero weight");  
 } catch (IllegalArgumentException e) {  
 assertEquals("Weight must be greater than zero", e.getMessage());  
 }  
  
  
 }  
  
 @Test  
 // Test accuracy of BMI Status  
 public void testGetBMISResult() {  
 bmi\_calculator bmiCalculator = new bmi\_calculator();  
  
 // Test 1: Underweight BMI less than 18.5  
 UserModel user1 = new UserModel(1.75f, 55.00f); // Set height in meters and weight in kg  
 user1.setBmi(bmiCalculator.calculateBMI(user1.getHeight(), user1.getWeight())); // Calculate BMI  
 assertEquals("Underweight", bmiCalculator.getBMISResult(user1.getBmi()));  
  
 // Test 2: Normal weight BMI 18.5 to 24.9  
 User user2 = new User(1.75, 70.0); // Set height in meters and weight in kg  
 user2.setBmi(bmiCalculator.calculateBMI(user2.getHeight(), user2.getWeight())); // Calculate BMI  
 assertEquals("Normal weight", bmiCalculator.getBMISResult(user2.getBmi()));  
  
 // Test 3: Overweight BMI 25.0 to 29.9  
 User user3 = new User(1.75, 85.0); // Set height in meters and weight in kg  
 user3.setBmi(bmiCalculator.calculateBMI(user3.getHeight(), user3.getWeight())); // Calculate BMI  
 assertEquals("Overweight", bmiCalculator.getBMISResult(user3.getBmi()));  
  
 // Test 4: Obesity BMI 30.0 to 34.9  
 User user4 = new User(1.75, 100.0); // Set height in meters and weight in kg  
 user4.setBmi(bmiCalculator.calculateBMI(user4.getHeight(), user4.getWeight())); // Calculate BMI  
 assertEquals("Obesity", bmiCalculator.getBMISResult(user4.getBmi()));  
  
 // Test 5: Borderline between Normal weight and Overweight  
 User user5 = new User(1.75, 68.0); // Set height in meters and weight in kg  
 user5.setBmi(bmiCalculator.calculateBMI(user5.getHeight(), user5.getWeight())); // Calculate BMI  
 assertEquals("Normal weight", bmiCalculator.getBMISResult(user5.getBmi()));  
  
 // Test 6: Borderline between Overweight and Obesity  
 User user6 = new User(1.75, 83.0); // Set height in meters and weight in kg  
 user6.setBmi(bmiCalculator.calculateBMI(user6.getHeight(), user6.getWeight())); // Calculate BMI  
 assertEquals("Overweight", bmiCalculator.getBMISResult(user6.getBmi()));  
  
 // Test 7: Very low BMI value  
 User user7 = new User(1.75, 40.0); // Set height in meters and weight in kg  
 user7.setBmi(bmiCalculator.calculateBMI(user7.getHeight(), user7.getWeight())); // Calculate BMI  
 assertEquals("Underweight", bmiCalculator.getBMISResult(user7.getBmi()));  
  
 // Test 8: Very high BMI value  
 User user8 = new User(1.75, 150.0); // Set height in meters and weight in kg  
 user8.setBmi(bmiCalculator.calculateBMI(user8.getHeight(), user8.getWeight())); // Calculate BMI  
 assertEquals("Extreme obesity", bmiCalculator.getBMISResult(user8.getBmi()));  
  
 // Test 9: Negative BMI value (must be invalid)  
 User user9 = new User(1.75, -70.0); // Set height in meters and weight in kg  
 try {  
 user9.setBmi(bmiCalculator.calculateBMI(user9.getHeight(), user9.getWeight())); // Calculate BMI  
 fail("Expected IllegalArgumentException for negative BMI value");  
 } catch (IllegalArgumentException e) {  
 assertEquals("Height and weight must be positive values", e.getMessage());  
 }  
  
 // Test 10: Zero BMI value (must be invalid)  
 User user10 = new User(1.75, 0.0); // Set height in meters and weight in kg  
 try {  
 user10.setBmi(bmiCalculator.calculateBMI(user10.getHeight(), user10.getWeight())); // Calculate BMI  
 fail("Expected IllegalArgumentException for zero BMI value");  
 } catch (IllegalArgumentException e) {  
 assertEquals("Height and weight must be positive values", e.getMessage());  
 }  
 }  
  
  
  
  
 \*/*