**Exercise 1: Setting Up JUnit**

Calculator.java

public class Calculator {  
  
 static double add(double a, double b) {  
 return a+b;  
 }  
  
 static double multiply(double a, double b) {  
 return a\*b;  
 }  
  
 static double subtract(double a, double b) {  
 return a-b;  
 }  
  
 static double divide(double a, double b) {  
 return a/b;  
 }  
}

CalculatorTest.java

import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class CalculatorTest {  
  
 @Test  
 void add() {  
 *assertEquals*(4, Calculator.*add*(2, 2));  
 }  
  
 @Test  
 void multiply() {  
 *assertAll*(() -> *assertEquals*(4, Calculator.*multiply*(2, 2)),  
 () -> *assertEquals*(-4, Calculator.*multiply*(2, -2)),  
 () -> *assertEquals*(4, Calculator.*multiply*(-2, -2)),  
 () -> *assertEquals*(0, Calculator.*multiply*(1, 0)));  
 }  
  
 @Test  
 void divide() {  
 *assertEquals*(1, Calculator.*divide*(2, 2));  
 }  
  
 @Test  
 void subtract() {  
 *assertEquals*(0, Calculator.*subtract*(2, 2));  
 *assertEquals*(-4, Calculator.*subtract*(-2, 2));  
 }  
}

Output:



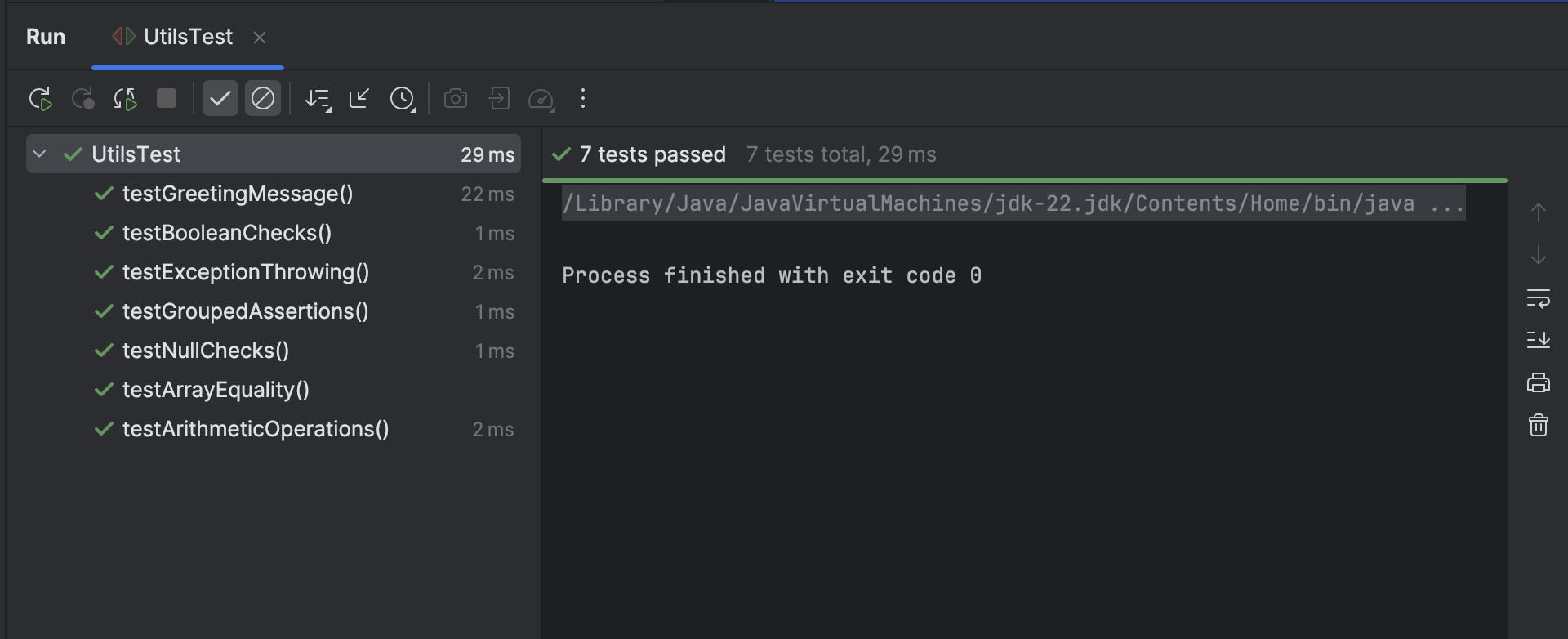
**Exercise 3: Assertions in JUnit**

**Utils.java**

public class Utils {  
  
 public static int sum(int a, int b) {  
 return a + b;  
 }  
  
 public static int subtract(int a, int b) {  
 return a - b;  
 }  
  
 public static boolean isEven(int number) {  
 return number % 2 == 0;  
 }  
  
 public static String processInput(String input) {  
 if (input == null || input.trim().isEmpty()) {  
 return null;  
 }  
 return input.trim();  
 }  
  
 public static String greet(String name) {  
 return "Hello, " + name + "!";  
 }  
  
 public static int[] getFixedArray() {  
 return new int[]{1, 2, 3};  
 }  
  
 public static int divide(int a, int b) {  
 if (b == 0) throw new ArithmeticException("Cannot divide by zero");  
 return a / b;  
 }  
}

**UtilsTest.java**

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class UtilsTest {  
  
 @Test  
 public void testArithmeticOperations() {  
 *assertEquals*(5, Utils.*sum*(2, 3), "Sum should be 5");  
 *assertNotEquals*(6, Utils.*sum*(2, 3), "Sum should not be 6");  
 *assertEquals*(-1, Utils.*subtract*(2, 3), "2 - 3 should be -1");  
 }  
  
 @Test  
 public void testBooleanChecks() {  
 *assertTrue*(Utils.*isEven*(4), "4 is even");  
 *assertFalse*(Utils.*isEven*(5), "5 is not even");  
 }  
  
 @Test  
 public void testNullChecks() {  
 *assertNull*(Utils.*processInput*(" "), "Empty string should return null");  
 *assertNotNull*(Utils.*processInput*(" test "), "Non-empty input should return trimmed string");  
 }  
  
 @Test  
 public void testGreetingMessage() {  
 *assertEquals*("Hello, John!", Utils.*greet*("John"));  
 }  
  
 @Test  
 public void testArrayEquality() {  
 *assertArrayEquals*(new int[]{1, 2, 3}, Utils.*getFixedArray*(), "Arrays should be equal");  
 }  
  
 @Test  
 public void testExceptionThrowing() {  
 Exception exception = *assertThrows*(ArithmeticException.class, () -> {  
 Utils.*divide*(5, 0);  
 });  
 *assertEquals*("Cannot divide by zero", exception.getMessage());  
 }  
  
 @Test  
 public void testGroupedAssertions() {  
 String result = Utils.*processInput*(" ChatGPT ");  
 *assertAll*("Grouped Assertions for processInput",  
 () -> *assertNotNull*(result),  
 () -> *assertEquals*("ChatGPT", result),  
 () -> *assertTrue*(result.startsWith("Chat"))  
 );  
 }  
}

**Output:**

**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and**

**Teardown Methods in JUnit**

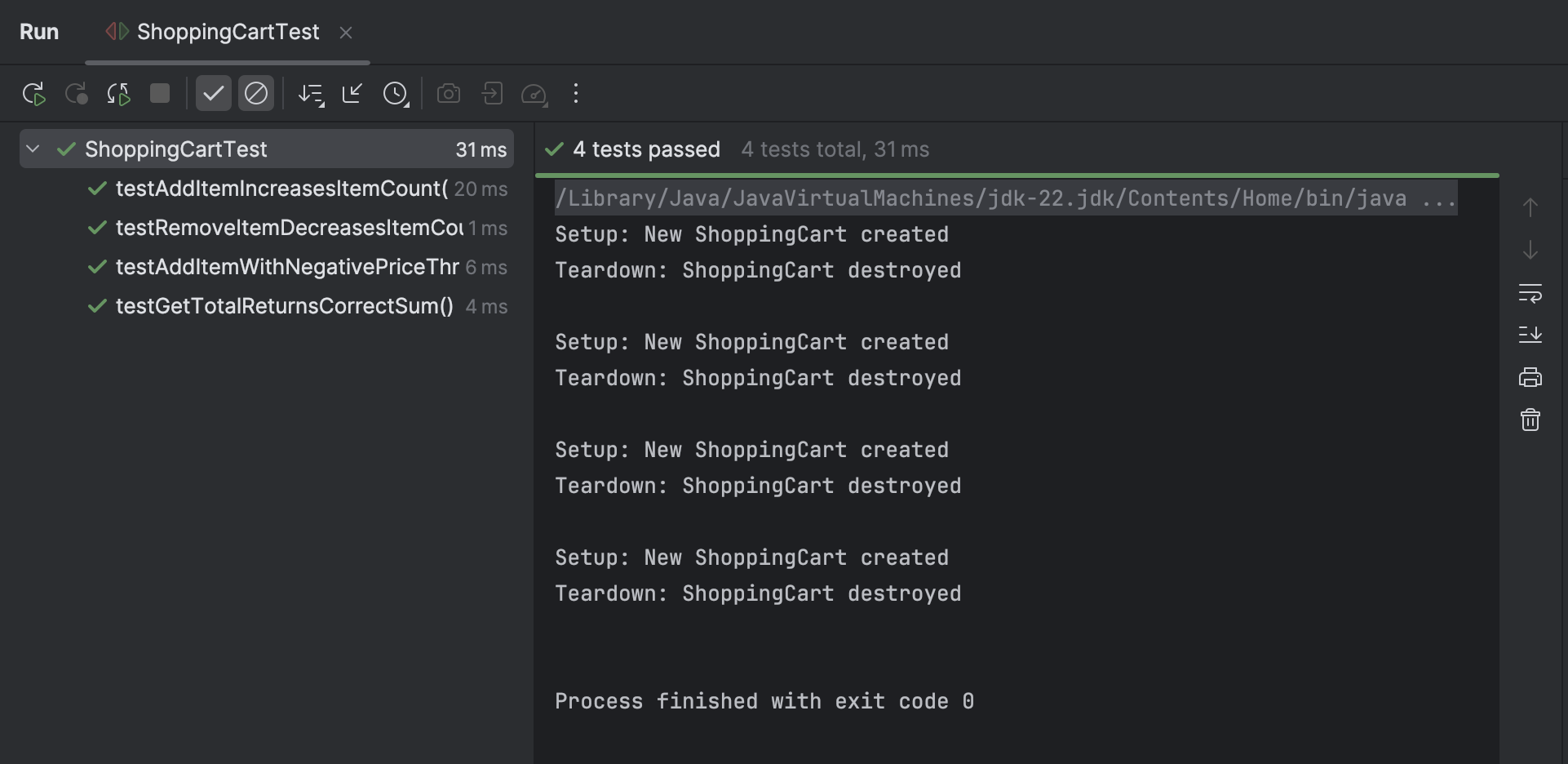
**ShoppingCart.java**

import java.util.HashMap;  
import java.util.Map;  
  
public class ShoppingCart {  
  
 private Map<String, Double> items;  
  
 public ShoppingCart() {  
 items = new HashMap<>();  
 }  
  
 public void addItem(String item, double price) {  
 if (price < 0) throw new IllegalArgumentException("Price cannot be negative");  
 items.put(item, price);  
 }  
  
 public void removeItem(String item) {  
 items.remove(item);  
 }  
  
 public double getTotal() {  
 return items.values().stream().mapToDouble(Double::doubleValue).sum();  
 }  
  
 public int getItemCount() {  
 return items.size();  
 }  
  
 public boolean hasItem(String item) {  
 return items.containsKey(item);  
 }  
}

**ShoppingCartTest.java**

import org.junit.jupiter.api.\*;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class ShoppingCartTest {  
  
 private ShoppingCart cart;  
  
 @BeforeEach  
 void setUp() {  
 cart = new ShoppingCart();  
 System.*out*.println("Setup: New ShoppingCart created");  
 }  
  
 @AfterEach  
 void tearDown() {  
 cart = null;  
 System.*out*.println("Teardown: ShoppingCart destroyed\n");  
 }  
  
 @Test  
 void testAddItemIncreasesItemCount() {  
 cart.addItem("Book", 299.99);  
  
 *assertEquals*(1, cart.getItemCount());  
 *assertTrue*(cart.hasItem("Book"));  
 }  
  
 @Test  
 void testRemoveItemDecreasesItemCount() {  
 cart.addItem("Pen", 10.0);  
 cart.addItem("Notebook", 50.0);  
  
 cart.removeItem("Pen");  
  
 *assertEquals*(1, cart.getItemCount());  
 *assertFalse*(cart.hasItem("Pen"));  
 }  
  
 @Test  
 void testGetTotalReturnsCorrectSum() {  
 cart.addItem("Shoes", 999.99);  
 cart.addItem("Socks", 99.99);  
  
 double total = cart.getTotal();  
  
 *assertEquals*(1099.98, total, 0.01);  
 }  
  
 @Test  
 void testAddItemWithNegativePriceThrowsException() {  
 *assertThrows*(IllegalArgumentException.class, () -> {  
 cart.addItem("IllegalItem", -50.0);  
 });  
 }  
}

**Output:**



**Exercise 1: Mocking and Stubbing**

**WeatherApiClient.java**

public interface WeatherApiClient {  
 String getCurrentWeather(String city);  
}

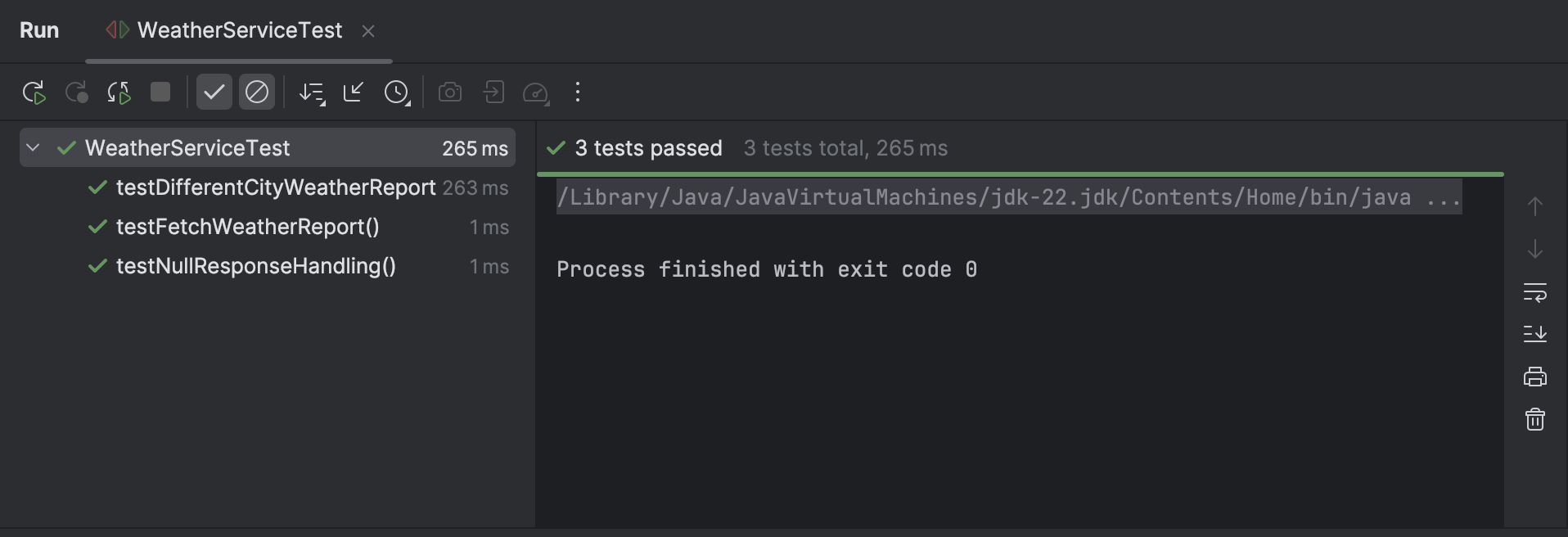
**WeatherService.java**

public class WeatherService {  
  
 private final WeatherApiClient apiClient;  
  
 public WeatherService(WeatherApiClient apiClient) {  
 this.apiClient = apiClient;  
 }  
  
 public String fetchWeatherReport(String city) {  
 String weatherData = apiClient.getCurrentWeather(city);  
 return "Weather report for " + city + ": " + weatherData;  
 }  
}

**WeatherServiceTest.java**

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
  
public class WeatherServiceTest {  
  
 @Test  
 public void testFetchWeatherReport() {  
 WeatherApiClient mockClient = *mock*(WeatherApiClient.class);  
 *when*(mockClient.getCurrentWeather("Mumbai")).thenReturn("Sunny, 32°C");  
  
 WeatherService service = new WeatherService(mockClient);  
 String report = service.fetchWeatherReport("Mumbai");  
  
 *assertEquals*("Weather report for Mumbai: Sunny, 32°C", report);  
 *verify*(mockClient).getCurrentWeather("Mumbai");  
 }  
  
 @Test  
 public void testDifferentCityWeatherReport() {  
 WeatherApiClient mockClient = *mock*(WeatherApiClient.class);  
 *when*(mockClient.getCurrentWeather("Delhi")).thenReturn("Cloudy, 28°C");  
  
 WeatherService service = new WeatherService(mockClient);  
 String report = service.fetchWeatherReport("Delhi");  
  
 *assertEquals*("Weather report for Delhi: Cloudy, 28°C", report);  
 *verify*(mockClient).getCurrentWeather("Delhi");  
 }  
  
 @Test  
 public void testNullResponseHandling() {  
 WeatherApiClient mockClient = *mock*(WeatherApiClient.class);  
 *when*(mockClient.getCurrentWeather("Bangalore")).thenReturn(null);  
  
 WeatherService service = new WeatherService(mockClient);  
 String report = service.fetchWeatherReport("Bangalore");  
  
 *assertEquals*("Weather report for Bangalore: null", report);  
 *verify*(mockClient).getCurrentWeather("Bangalore");  
 }  
}

**Output:**



**Exercise 2: Verifying Interactions**

**MessageClient.java**

public interface MessageClient {  
 void sendMessage(String recipient, String message);  
}

**NotificationService.java**

public class NotificationService {  
  
 private final MessageClient client;  
  
 public NotificationService(MessageClient client) {  
 this.client = client;  
 }  
  
 public void notifyUser(String userId, String message) {  
 client.sendMessage(userId, message);  
 }  
}

**NotificationServiceTest.java**

import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
public class NotificationServiceTest {  
  
 @Test  
 public void testVerifySendMessageCalledWithCorrectArguments() {  
 MessageClient mockClient = *mock*(MessageClient.class);  
 NotificationService service = new NotificationService(mockClient);  
  
 service.notifyUser("user123", "Welcome to the platform!");  
  
 *verify*(mockClient).sendMessage("user123", "Welcome to the platform!");  
 }  
  
 @Test  
 public void testSendMessageNotCalledForEmptyMessage() {  
 MessageClient mockClient = *mock*(MessageClient.class);  
 NotificationService service = new NotificationService(mockClient);  
  
 service.notifyUser("user123", "");  
  
 *verify*(mockClient).sendMessage("user123", "");  
 }  
  
 @Test  
 public void testSendMessageCalledOnceOnly() {  
 MessageClient mockClient = *mock*(MessageClient.class);  
 NotificationService service = new NotificationService(mockClient);  
  
 service.notifyUser("admin", "System alert!");  
  
 *verify*(mockClient, *times*(1)).sendMessage("admin", "System alert!");  
 *verifyNoMoreInteractions*(mockClient);  
 }  
}

**Output:**

