**JUNIT-5 AND MOCKITO ASSIGNMENT**

**Assignment on Junit**

1) Write a class called MinMaxFinder. Define a method in it called find MinMax() which accepts an int array and returns new array of size 2, wherein the 0th index will have the min value of the array and 1st index will have max value of the array. Perform Junit testing of the method find Min Max with as many test cases you can think of (min 3 test cases)

E.g.

MinMaxFinder.find Min Max( new int[]{56, 34, 7,3, 54, 3, 34, 34, 53}); should return a new array with min and max values {3, 56} at 0th and 1st index respectively

**Solution:**

**MinMaxFinder.java**

public class MaxMinFinder {

public static int[] findMaxMin(int[] inputArr)

{

int[] minMaxValue = new int[2];

int max = inputArr[0];

int min = inputArr[0];

for(int i = 1 ; i < inputArr.length ; i++)

{

if(inputArr[i] > max) //for max value

max=inputArr[i];

if(inputArr[i] < min) //for min value

min=inputArr[i];

}

minMaxValue[0] = min;

minMaxValue[1] = max;

return minMaxValue; //returning array

}

}

**MinMaxFinderTest.java**

**i**mport static org.junit.jupiter.api.Assertions.\*;

import java.util.Arrays;

import org.junit.jupiter.api.Test;

class MaxMinFinderTest {

int[] result = new int[2];

@Test

void test1() {

result = MaxMinFinder.findMaxMin(new int[] {1,3,56,26,32,755,0,4535,42,21});

int[] expectedResult = {0,4535};

assertEquals(Arrays.toString(expectedResult), Arrays.toString(result));

}

@Test

void test2() {

result = MaxMinFinder.findMaxMin(new int[] {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0});

int[] expectedResult = {0,0};

assertEquals(Arrays.toString(expectedResult), Arrays.toString(result));

}

@Test

void test3() {

result = MaxMinFinder.findMaxMin(new int[] {1,3,4,5,6,723,563,121231,545,2,56,6});

int[] expectedResult = {1,121231};

assertEquals(Arrays.toString(expectedResult), Arrays.toString(result));

}

@Test

void test4() {

result = MaxMinFinder.findMaxMin(new int[] {0,324,234,23,521,55,555,55666,555,77});

int[] expectedResult = {0,55666};

assertEquals(Arrays.toString(expectedResult), Arrays.toString(result));

}

@Test

void test5() {

result = MaxMinFinder.findMaxMin(new int[] {333,33,333,333,333,3333,3333333,333,33});

int[] expectedResult = {33,3333333};

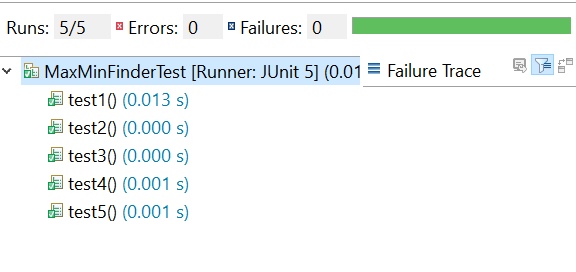
assertEquals(Arrays.toString(expectedResult), Arrays.toString(result));

}

}

**Output:**

**Testing output:**



2) Modify the above method to return a single object representing min and max value of the pass array. Define new sets of Junit Test cases of this modified method.

**Solution:**

**MinMax.java**

package MinMaxFromArray;

public class MinMax {

private int[] minMax = new int[2];

public int[] getMinMax() {

return minMax;

}

public void setMinMax(int min , int max) {

this.minMax[0] = min;

this.minMax[1] = max;

}

}

**FindMinMax.java**

**package** MinMaxFromArray;

**public** **class** FindMinMax {

**public** **static** MinMax maxMinInArray ( **int**[] inputArray )

{ MinMax obj = **new** MinMax(); //created a object to store min max from input array

**int** min = inputArray[0]; //min

**int** max = inputArray[0]; //max

**for**(**int** i = 1 ; i < inputArray.length ; i++)

{ **if**( inputArray[i] > max ) //for max value

max=inputArray[i];

**if**( inputArray[i] < min ) //for min value

min=inputArray[i];

}

obj.setMinMax(min,max); // stored min max of array in the object

**return** obj; //returning the object

}

}

**FindMinMaxTest.java**

package MinMaxFromArray;

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

import java.util.Arrays;

import org.junit.jupiter.api.Test;

class FindMinMaxTest {

MinMax testObject;

@Test

void test1() {

testObject = FindMinMax.maxMinInArray(new int[] {1,3,56,26,32,755,0,4535,42,21});

int[] expected = {0,4535};

int[] actual = testObject.getMinMax();

assertEquals(Arrays.toString(expected),Arrays.toString(actual));

}

@Test

void test2() {

testObject = FindMinMax.maxMinInArray(new int[] {12,46,78,123,7,2325,3232,7644,211235});

int[] expected = {7,211235};

int[] actual = testObject.getMinMax();

assertEquals(Arrays.toString(expected),Arrays.toString(actual));

}

@Test

void test3() {

testObject = FindMinMax.maxMinInArray(new int[] {14,62,632,6344,776,2345,45232,4331});

int[] expected = {14,45232};

int[] actual = testObject.getMinMax();

assertEquals(Arrays.toString(expected),Arrays.toString(actual));

}

@Test

void test4() {

testObject = FindMinMax.maxMinInArray(new int[] {1});

int[] expected = {1,1};

int[] actual = testObject.getMinMax();

assertEquals(Arrays.toString(expected),Arrays.toString(actual)); }

@Test

void test5() {

testObject = FindMinMax.maxMinInArray(new int[] {1234,12});

int[] expected = {12,1234};

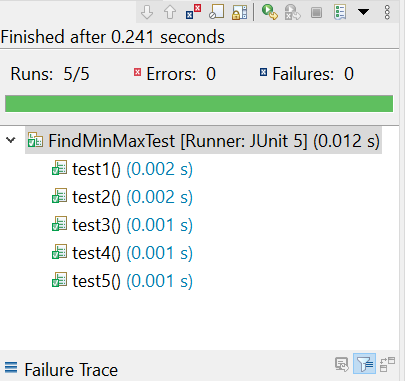
int[] actual = testObject.getMinMax();

assertEquals(Arrays.toString(expected),Arrays.toString(actual));

}

}

**Output:**

****

3) Write a Bank Account class with method withdraw which accepts amount to be withdrawn from the account (amount to be deducted from the balance of the account). In case there are insufficient funds a InsufficientFundsExpcetion should be raised. After defining the method perform Junit testing to check whether the insufficientFundsException is raised when you try to withdraw amount that is over and above the account balance.

bankAccount.withdraw(20,000); should raise the Insufficient Funds Exception if the balance in the account is less than 20,000.

**Solution:**

**BankAccount.java**

**package** BankAccount;

**public** **class** BankAccount {

**private** **double** balance = 20000;

**public** **double** withdraw(**double** amount) **throws** InsufficientFundsExpcetion

{

**if**(amount > balance)

{

**throw** **new** InsufficientFundsExpcetion();

}

**else** {

balance-=amount;

}

**return** balance;

}

}

**InsufficientFundsException.java**

**package** BankAccount;

**public** **class** InsufficientFundsExpcetion **extends** Exception{

InsufficientFundsExpcetion() {

}

}

**BankAccountTest.java**

package BankAccount;

import org.junit.jupiter.api.Test;

class BankAccountTest {

BankAccount account = new BankAccount();

@Test

void test1() throws InsufficientFundsExpcetion{

account.withdraw(12000);

}

@Test

void test2() throws InsufficientFundsExpcetion{

account.withdraw(30000);

}

@Test

void test3() throws InsufficientFundsExpcetion{

account.withdraw(1000);

}

@Test

void test4() throws InsufficientFundsExpcetion{

account.withdraw(10000);

}

@Test

void test5() throws InsufficientFundsExpcetion{

account.withdraw(21000);

}

@Test

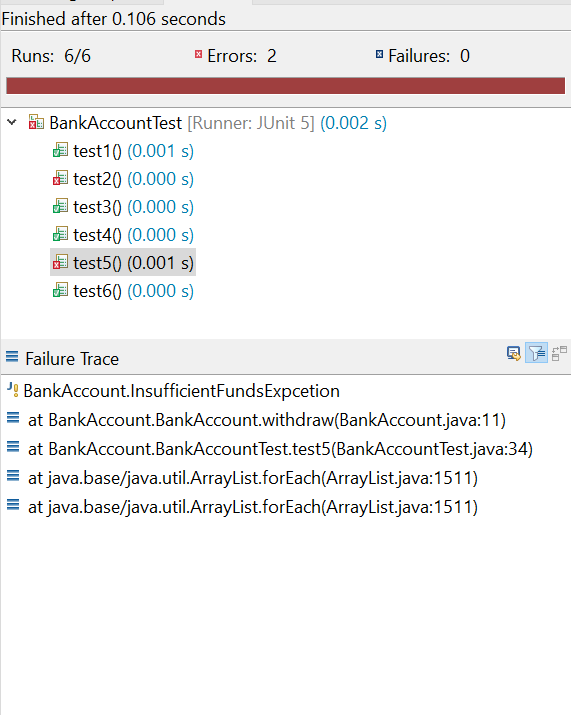
void test6() throws InsufficientFundsExpcetion{

account.withdraw(19000);

}

}

**Output:**



4) Write a Junit Testing to show the use of Lifecycle hooks annotation such as @BeforeAll, @BeforeEach @AfterEach and @AfterAll  
  
**Solution:  
  
Created a class with some basic methods.**

**DatabaseApplication.java**

**package** JunitHooks;

**public** **class** DatabaseApplication {

**public** **void** addData() {

System.***out***.println("added some data");

}

**public** **void** fetchData() {

System.***out***.println("fetched some data");

}

**public** **void** updateData() {

System.***out***.println("updated some data");

}

**public** **void** deleteData() {

System.***out***.println("deleted some data");

}

}

**DatabaseApplicationTest.java**

package JunitHooks;

import org.junit.jupiter.api.AfterAll;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Nested;

import org.junit.jupiter.api.Test;

class DatabaseApplicationTest {

DatabaseApplication dbApp ;

@BeforeAll

static void beforeAllinit(){

System.out.println("Started the Database Server");

}

@BeforeEach

void createInstance()

{

dbApp = new DatabaseApplication();

System.out.println("Database Instance Created");

}

@AfterEach

void commitChanges()

{

System.out.println("Changes Commited");

}

@AfterAll

static void turnoffServer(){

System.out.println("Database Server has been closed");

}

@Nested

class dbAppTests{

@Test

@DisplayName("Checking addData Method - Test1")

void addTest()

{

dbApp.addData();

}

@Test

@DisplayName("Checking fetchData Method - Test2")

void fetchTest()

{

dbApp.fetchData();

}

@Test

@DisplayName("Checking updateData Method - Test3")

void updateTest()

{

dbApp.updateData();

}

@Test

@DisplayName("Checking DeleteData Method - Test4")

void deleteTest()

{

dbApp.deleteData();

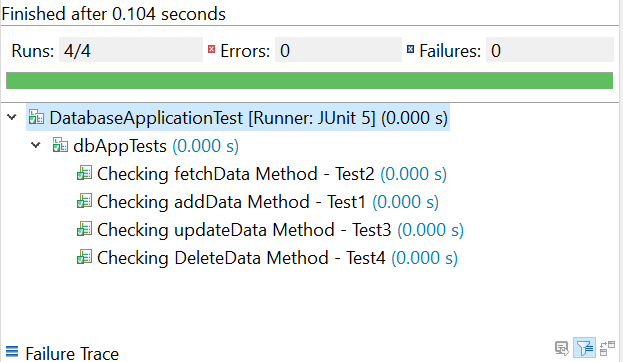
}

}

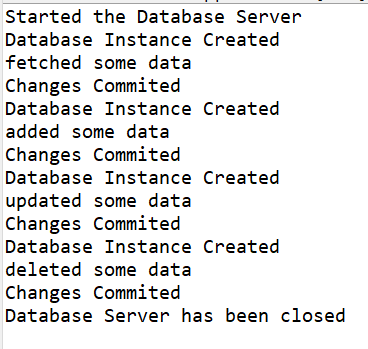
}

**Output:**

**Testing Output:**

****

**Console Output:**

****