**Assignments on JUnit**

1. Write a class called MinMaxFinder. Define a method in it called findMinMax() which accepts an int array and return 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 findMinMax with as many test cases you can think of (min 3 test cases)

Eg.

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

**package** org.app;

**public** **class** MinMaxFinder {

**public** **int**[] minmaxFinder(**int**[] a) {

**int**[] arr=**new** **int**[2];

**int** min = a[0],max=0;

**for**(**int** i=0; i<a.length; i++ ) {

**if**(a[i]<min) {

min = a[i];

}

}

arr[0]=min;

**for**(**int** i=0; i<a.length; i++ ) {

**if**(a[i]>max) {

max = a[i];

}

}

arr[1]=max;

**int** r[] = {min,max};

System.***out***.println("Minimum:" + r[0] +" "+ "Maximum:" + r[1]);

**return** r;

}

}

package org.app;

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

import org.junit.jupiter.api.Test;

class MinMaxFinderTest {

int[] expected1= {4,21};

int[] expected2= {5,98};

int[] expected3= {1,6};

@Test

void test() {

MinMaxFinder mm=new MinMaxFinder();

int[] a= {20,10,4,15,21};

int[] min=mm.minmaxFinder(a);

assertArrayEquals(expected1,min); //WithArray

}

@Test

void test1() {

MinMaxFinder mm=new MinMaxFinder();

int[] a= {45,7,48,5,98};

int[] min=mm.minmaxFinder(a);

assertArrayEquals(expected2,min);

}

@Test

void test2() {

MinMaxFinder mm=new MinMaxFinder();

int[] a= {2,5,4,1,6};

int[] min=mm.minmaxFinder(a);

assertArrayEquals(expected3,min);

}

}

**Output:** Minimum:4 Maximum:21

Minimum:5 Maximum:98

Minimum:1 Maximum:6

1. 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.

**package** org.app;

**import** java.util.Arrays;

**public** **class** ObjArr {

**public** Object[] minmaxfinder(**int**[] a) {

**int**[] arr=**new** **int**[2];

**int** min = a[0],max=0;

**for**(**int** i=0; i<a.length; i++ ) {

**if**(a[i]<min) {

min = a[i];

}

}

arr[0]=min;

**for**(**int** i=0; i<a.length; i++ ) {

**if**(a[i]>max) {

max = a[i];

}

}

arr[1]=max;

Object r[] = {min,max};

System.***out***.println("Minimum, Maximum: "+Arrays.*toString*(r));

**return** r;

}

}

**package** org.app;

**import** **static** org.junit.Assert.*assertArrayEquals*;

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

**import** org.junit.jupiter.api.Test;

**class** ObjArrTest {

ObjArr objminmax;

Object[] expected1= {4,21};

Object[] expected2= {5,98};

Object[] expected3= {1,6};

@Test

**void** test() {

ObjArr mm=**new** ObjArr();

**int**[] a= {20+1,10-6,8,15,19};

Object[] min=mm.minmaxfinder(a);

*assertArrayEquals*(expected1,min);

}

@Test

**void** test1() {

ObjArr mm=**new** ObjArr();

**int**[] a= {45,7,48+2,5,98};

Object[] min=mm.minmaxfinder(a);

*assertArrayEquals*(expected2,min);

}

@Test

**void** test2() {

ObjArr mm=**new** ObjArr();

**int**[] a= {2,5,4,1,6};

Object[] min=mm.minmaxfinder(a);

*assertArrayEquals*(expected3,min);

}

}

**Output:** Minimum, Maximum: [4, 21]

Minimum, Maximum: [5, 98]

Minimum, Maximum: [1, 6]

1. Write a BankAccount class with method withdraw which accepts amount to be withdrawn from the account (amount to be deducted from the balance of the amount). In case there are sufficient funds a InsufficientFundsException 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 InSufficientFundsException if the balance in the amount is less than 20,000.

**package** org.app;

**public** **class** Banking {

**int** balance;

**public** **int** getBalance() {

**return** balance;

}

**public** **void** setBalance(**int** balance) {

**this**.balance = balance;

}

**public** **void** withdraw(**int** withdrawamt) **throws** InsufficientFundsException {

**if**(withdrawamt >=balance) {

**throw** **new** InsufficientFundsException();

}

**else** {

System.***out***.println("New balance amount is:"+(balance - withdrawamt));

}

}

}

**package** org.app;

**public** **class** InsufficientFundsException **extends** Exception {

**public** **void** InsufficientFundsException() {

}

}

**package** org.app;

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

//import static org.junit.Assert.assertArrayEquals;

//import org.app.Banking;

//import org.app.InsufficientFundsException;

**import** org.junit.jupiter.api.BeforeEach;

**import** org.junit.jupiter.api.DisplayName;

**import** org.junit.jupiter.api.Test;

**class** BankingTest {

Banking b;

@BeforeEach

**void** initialise() {

b=**new** Banking();

}

@Test

@DisplayName("checking InsufficientFundsException")

**public** **void** testWithdraw() {

b.setBalance(2000);

*assertThrows*(InsufficientFundsException.**class**,()->b.withdraw(25000),"your balance is less than withdraw amount are equals to withdraw amount");

}

@Test

@DisplayName("checking the balance ")

**public** **void** testWithdrawWithoutException() {

b.setBalance(2500);

**int** expected=2000;

**int** actual=2000;

*assertEquals*(expected,actual,"invalid balance");

}

}

**Output:** New balance amount is :1500

1. Write a JUnit Testinf to show the use of Lifecycle hooks annotation such as @BeforeAll, @BeforeEach, @AfterEach and @AfterAll.

**package** org.app;

**import** org.junit.runner.JUnitCore;

**public** **class** LifeCycle {

**public** **static** **void** main(String [] args) {

JUnitCore junit = **new** JUnitCore();

junit.run(LifeCycleTest.**class**);

}}

package org.app;

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

import org.junit.After;

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.Test;

class LifeCycleTest {

@BeforeAll

static void beforeTheEntireTestFixture() {

System.out.println("Before the entire test fixture");

}

@AfterAll

static void afterTheEntireTestFixture() {

System.out.println("After the entire test fixture");

}

@BeforeEach

void beforeEachTest() {

System.out.println("Before each test");

}

@AfterEach

void afterEachTest() {

System.out.println("After each test");

}

@Test

void firstTest() {

System.out.println("First test");

}

@Test

void secondTest() {

System.out.println("Second test");

}

}

**Output:** Before the entire test fixture

Before each test

First test

After each test

Before each test

Second test

After each test

After the entire test fixture