**JUNIT ASSIGNMENT**

**1]ANS:-**

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;

}

}

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

**2ANS:-**

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;

}

}

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]

**3]ANS:-**

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));

}

}

}

public class InsufficientFundsException extends Exception {

public void InsufficientFundsException() {

}

}

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

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

**4]ANS:-**

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

import org.junit.jupiter.api.Test;

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;

public class LifecylceTest

{

public void LifecycleTest() {

System.out.println("Constructor");

}

@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