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

E.g.

MinMaxFinder.findMinMax( new int[](56, 34,7,3, 54, 3, 34, 34, 53} ); should return a new array with min and max values (3,56) at oth and 1"index respectively

#### Mainclass

```sh

import java.util.Arrays;

public class MinMaxFinder {

public void main(String[] args) {

findMinMax(new int[] {56,34,7,3,54,3,34,34,53});

}

public int[] findMinMax(int[] is) {

Arrays.sort(is);

int min = is[0];

int max = is[is.length-1];

int res[]= {min,max};

System.out.println("(Minimum value index 0,Maximum value index 1) : "+res[0]+","+res[1]);

return res;

}

}

```

#### Testcases class

```sh

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

import java.util.Arrays;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestInstance;

import org.junit.jupiter.api.TestInstance.Lifecycle;

@TestInstance(Lifecycle.PER\_CLASS)

class MinMaxFinderTest {

MinMaxFinder minmax;

int[] arr1 = {56,34,7,3,58,3,34,1,53};

int[] arr2 = {56,37,7,99,31,34,8,10};

int[] arr3 = {11,89,29,121,12,55};

int[] expectedval1 = {1,58};

int[] expectedval2 = {7,99};

int[] expectedval3 = {11,121};

@BeforeAll

void init() {

minmax = new MinMaxFinder();

}

@Test

void arr1testcase() {

fun(arr1,expectedval1);

fun(arr2,expectedval2);

fun(arr3,expectedval3);

}

private void fun(int[] arr, int[] expectedval) {

// TODO Auto-generated method stub

int[] actualval = minmax.findMinMax(arr);

assertNotNull(Arrays.toString(actualval),"No data Found");

assertEquals(Arrays.toString(expectedval),Arrays.toString(actualval));

assertArrayEquals(expectedval ,actualval,"mismatch");

assertTrue(actualval.length==2);

assertTrue(actualval[0]<actualval[1]);

}

}

```

#### Output

```sh

(Minimum value index 0,Maximum value index 1) : 1,58

(Minimum value index 0,Maximum value index 1) : 7,99

(Minimum value index 0,Maximum value index 1) : 11,121

```

![screenshots](https://github.com/sayuyagami/Training\_assignments/blob/master/screenshots/Maxmintestcases.png)

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

#### Main class

```sh

import java.util.Arrays;

public class Objectarray {

public void main(String[] args) {

findMinMax(new int[] {56,34,7,3,54,3,34,34,53});

}

public Object[] findMinMax(int[] is) {

// TODO Auto-generated method stub

Arrays.sort(is);

int min = is[0];

int max = is[is.length-1];

Object res[]= {min,max};

System.out.println("(Minimum value,Maximum value) : "+Arrays.toString(res));

return res;

}

}

```

#### Testcase class

```sh

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

import java.util.Arrays;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestInstance;

import org.junit.jupiter.api.TestInstance.Lifecycle;

@TestInstance(Lifecycle.PER\_CLASS)

class ObjectarrayTest {

Objectarray objminmax;

int[] arr1 = {59\*2,34,11\*2,72,52,9,34-2,18,55};

int[] arr2 = {5622,3737-18,1897+1,9929,2456,1720,2898/2,2001};

int[] arr3 = {11000,89123,29890,12100,12567,55110};

Object expectedval1[] = {9,118};

Object expectedval2[] = {1449,9929};

Object expectedval3[] = {11000,89123};

@BeforeAll

void init() {

objminmax = new Objectarray();

}

@Test

void objtestcase1() {

fun(arr1,expectedval1);

fun(arr2,expectedval2);

fun(arr3,expectedval3);

}

private void fun(int[] arr, Object[] expval) {

Object actualval[] = objminmax.findMinMax(arr);

assertNotNull(actualval);

assertNotSame(expval,actualval);//varies in object reference

assertEquals(Arrays.toString(expval),Arrays.toString(actualval));

assertIterableEquals(Arrays.asList(expval),Arrays.asList(actualval));

}

}

```

#### Output

```sh

(Minimum value,Maximum value) : [9, 118]

(Minimum value,Maximum value) : [1449, 9929]

(Minimum value,Maximum value) : [11000, 89123]

```

![screenshots](https://github.com/sayuyagami/Training\_assignments/blob/master/screenshots/Objectarraytest.png)

>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 InsufficientFunds Expcetion 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.

#### BankAccount

```sh

public class BankAccount{

public static void withdraw(double withdamt, double balance) {

// TODO Auto-generated method stub

if(withdamt <= balance && withdamt > 20000) {

System.out.println("Withdraw done Successfully");

balance=balance-withdamt;

}else if( withdamt < 0)

{

throw new InvalidEntryException("InvalidEntry Fund RuntimeException");

}else if (withdamt > balance && withdamt > 20000)

{

throw new InsufficientFundsException("Insufficient Fund RuntimeException");

}else if(withdamt < 20000) {

throw new WithdrawLimitException("WithdrawLimitException RuntimeException");

}

//return balance;

}

}

class InvalidEntryException extends RuntimeException {

public InvalidEntryException(String message) {

super(message);

}

}

class InsufficientFundsException extends RuntimeException {

public InsufficientFundsException(String message) {

super(message);

}

}

class WithdrawLimitException extends RuntimeException {

public WithdrawLimitException(String message) {

super(message);

}

}

```

#### Testcases class

```sh

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

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

class BankAccountTest {

int i;

double bal1[]= {55000.0,100000.0,25000.0,55000.0,85000.0};

double bal2[]= {20000.0,18000.0,19000.0,17500.0,19900.0};

@ParameterizedTest

@ValueSource(doubles = {10000, 17300, 15000,18999,19999}) //limit values less than 20000

void testdrawLimitExpectedException(double w) {

for(i=0;i<bal1.length;i++) {

//throws WithdrawLimitException

assertThrows(WithdrawLimitException.class,

()->BankAccount.withdraw(w,bal1[i]));

}

}

@ParameterizedTest

@ValueSource(doubles = {21000, 30000, 50000, 39000,85000,60000}) //withdraw values more than 20000

void testInsuffExpectedException(double w) {

for(i=0;i<bal2.length;i++) {

//throws InsufficientFundsException

assertThrows(InsufficientFundsException.class,

()->BankAccount.withdraw(w,bal2[i]));

}

}

@ParameterizedTest

@ValueSource(doubles = {-10000, -17300, -15000,-18999,-99,-19999,-900}) //values less than 0

void InvalidEntryException(double w) {

for(i=0;i<bal1.length;i++) {

//throws InvalidEntryException

assertThrows(InvalidEntryException.class,

()->BankAccount.withdraw(w,bal1[i]));

assertThrows(InvalidEntryException.class,

()->BankAccount.withdraw(w,bal2[i]));

}

}

}

```

#### Output

![screenshots](https://github.com/sayuyagami/Training\_assignments/blob/master/screenshots/BankAccountTestcase.png)

>4.Write a Junit Testing to show the use of Lifecycle hooks annotation such as @BeforeAll,@BeforeEach @AfterEach and @After All

#### Mainclass

```sh

public class Calculator {

public static Integer multiply(int i, int j) {

// TODO Auto-generated method stub

return i\*j;

}

public static Integer add(int i, int j) {

// TODO Auto-generated method stub

return i+j;

}

}

```

#### Testcases Class

```sh

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;

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

public class HooksAnnotations {

@BeforeAll

static void befall(){

System.out.println("@BeforeAll executed");

}

@BeforeEach

void befeach(){

System.out.println("@BeforeEach executed");

}

@Test

void testCalcmultiply()

{

System.out.println("------TEST Multiply------");

assertEquals( 4 , Calculator.multiply(2, 2));

}

@Test

void testCalcadd()

{

System.out.println("------TEST Add------");

assertEquals( 6 , Calculator.add(2, 4));

}

@AfterEach

void afteach(){

System.out.println("@AfterEach executed");

}

@AfterAll

static void aftall(){

System.out.println("@AfterAll executed");

}

}

```

#### Output

```sh

@BeforeAll executed

@BeforeEach executed

------TEST Add------

@AfterEach executed

@BeforeEach executed

------TEST Multiply------

@AfterEach executed

@AfterAll executed

```