**Adisri Sarode**

**1)** Write a class called MinMaxFinder. Define a method in it called findMinMax() 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 findMinMax 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

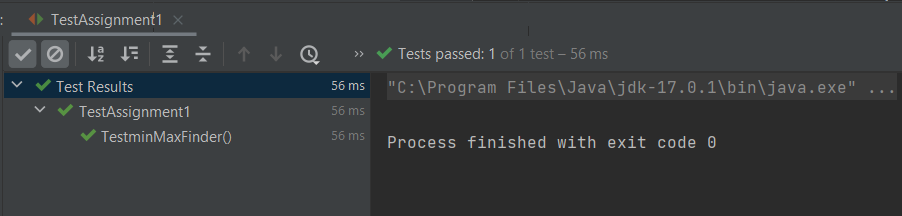
array with min and max values {3, 56} at 0th and 1st index respectively.

import java.util.Arrays;  
  
class MinMaxFinder{  
  
public int[] findMinMax(int[] arr){  
int[] ans = new int[2];  
  
Arrays.*sort*(arr);  
  
ans[0] = arr[0];  
ans[1] = arr[arr.length-1];  
return ans;  
 }  
}

**Code for Testing:**

import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.Assertions;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
public class TestAssignment1 {  
  
 private MinMaxFinder minMaxFinder;  
 private int[] expected;  
 private int[] actual;  
  
 @BeforeEach  
 public void init(){  
 minMaxFinder = new MinMaxFinder();  
 expected = new int[]{3, 56};  
 }  
  
 @Test  
 public void TestminMaxFinder(){  
 actual = minMaxFinder.findMinMax(new int[]{56, 34, 7,3, 54, 3, 34, 34, 53});  
 Assertions.*assertArrayEquals*(expected,actual);  
 }  
  
 @AfterEach  
 public void Clean(){  
 minMaxFinder = null;  
 expected = actual = null;  
 }  
}

**Output:** The below is showing that there is no error or failures while testing our code and all the defined tests passed successfully.

****

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

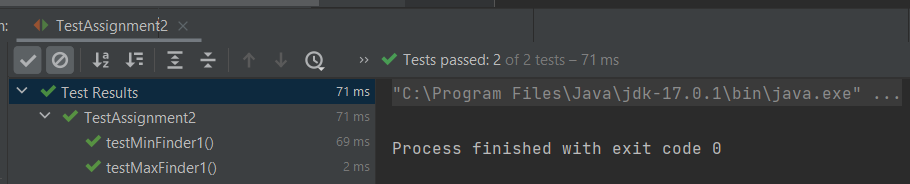
**Code:**

import java.util.Arrays;  
class MinMaxFinder1{  
 int max;  
 int min;  
 MinMaxFinder1(){}  
  
 MinMaxFinder1(int min, int max){  
 this.min = min;  
 this.max = max;  
  
 }  
 public MinMaxFinder1 minMaxFinder1(int[] arr){  
 int min;  
 int max;  
 Arrays.*sort*(arr);  
 min = arr[0];  
 max = arr[arr.length-1];  
  
 return new MinMaxFinder1(min,max);  
 }  
}

**Code for Testing:**

import org.junit.jupiter.api.\*;  
  
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)  
public class TestAssignment2 {  
 MinMaxFinder1 expected;  
 MinMaxFinder1 actual;  
 @BeforeEach  
 public void init(){  
 expected = new MinMaxFinder1(3,56);  
 actual = new MinMaxFinder1();  
 actual = actual.minMaxFinder1(new int[]{56, 34, 7,3, 54, 3, 34, 34, 53});  
  
 }  
  
 @Test()  
 @Order(1)  
 public void testMinFinder1(){  
 Assertions.*assertEquals*(expected.min,actual.min);  
 }  
  
 @Test  
 @Order(2)  
 public void testMaxFinder1(){  
 Assertions.*assertEquals*(expected.max,actual.max);  
 }  
  
 @AfterEach  
 public void Clean(){  
 actual = expected = null;  
 }  
}

**OUTPUT:**

****

**3)** Write a BankAccount class with method withdraw which accepts amount to be withdrawnfrom the account (amount to be deducted from the balance of the account). In case thereare insufficient funds a InsufficientFundsExpcetion should be raised. After defining themethod perform Junit testing to check whether the InsufficientFundsException is raisedwhen you try to withdraw amount that is over and above the account balance.bankAccount.withdraw(20,000); should raise the InsufficientFundsException ifthebalance in the account is less than 20,000.

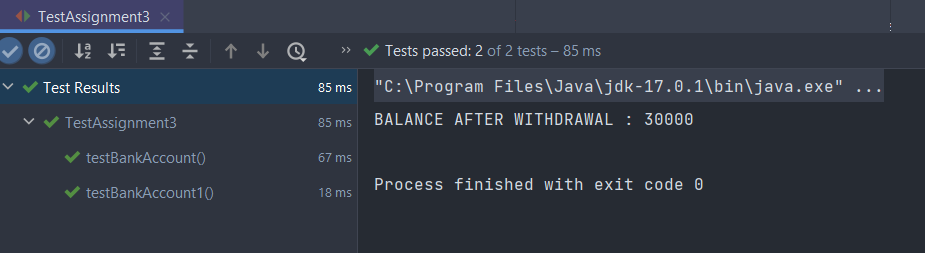
**Code:**

*class* BankAccount{  
 *private final long* amount;  
  
 BankAccount(*long* amount){  
 *this*.amount = amount;  
 }  
 *public long* withDraw(*long* withdrawalAmount) *throws* InsufficientFundsExpcetion{  
 *if*(amount-withdrawalAmount<=20000){  
 *throw new* InsufficientFundsExpcetion();  
 }  
 *return* amount-withdrawalAmount;  
 }  
}

**Code for testing:**

*import* org.junit.jupiter.api.Assertions;  
*import* org.junit.jupiter.api.BeforeEach;  
*import* org.junit.jupiter.api.Test;  
  
*public class* TestAssignment3 {  
 *private* BankAccount bankAccount;  
 *private long* withDrawAmount;  
 @BeforeEach  
 *public void* init(){  
*//Initialize the Bank Amount;* bankAccount = *new* BankAccount(50000);  
 }  
  
 @Test  
 *public void* testBankAccount(){  
 withDrawAmount = 40000;  
 Assertions.*assertThrows*(InsufficientFundsExpcetion.*class*,()->{  
 System.out.println(bankAccount.withDraw(withDrawAmount));  
 },"YOU HAVE INSUFFICIENT BALANCE!!!");  
  
 }  
  
 @Test  
 *public void* testBankAccount1(){  
 withDrawAmount = 20000;  
 Assertions.*assertDoesNotThrow*(()->{  
 System.out.println("BALANCE AFTER WITHDRAWAL : "+bankAccount.withDraw(withDrawAmount));  
 });  
 }  
}

**Output:**

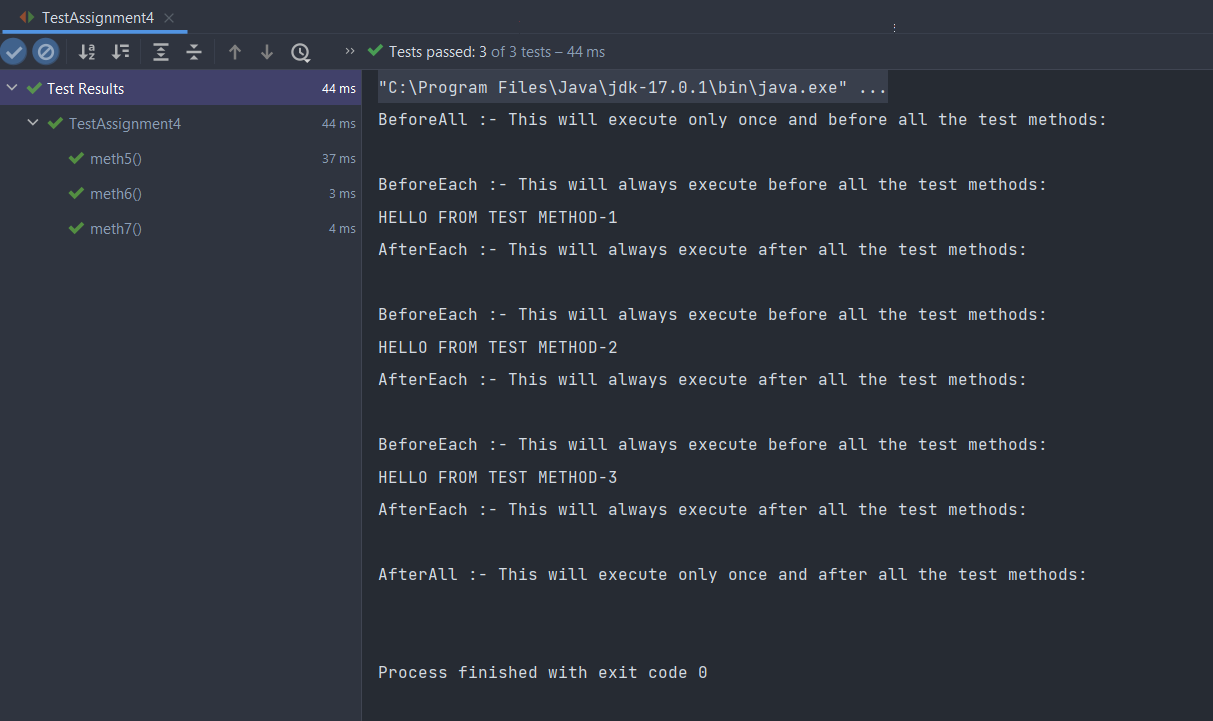
****

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

Code:

*import* org.junit.jupiter.api.\*;  
  
@TestMethodOrder(*MethodOrderer*.OrderAnnotation.*class*)  
*public class* TestAssignment4 {  
  
 @BeforeAll  
 *static void* meth1(){  
 System.out.println("BeforeAll :- This will execute only once and before all the test methods: \n");  
 }  
  
 @BeforeEach  
 *public void* meth2(){  
 System.out.println("BeforeEach :- This will always execute before all the test methods: ");  
 }  
  
 @Test  
 @Order(1)  
 *public void* meth5(){  
 System.out.println("HELLO FROM TEST METHOD-1");  
 }  
 @Test  
 @Order(2)  
 *public void* meth6(){  
 System.out.println("HELLO FROM TEST METHOD-2");  
 }  
 @Test  
 @Order(3)  
 *public void* meth7(){  
 System.out.println("HELLO FROM TEST METHOD-3");  
 }  
 @AfterEach  
 *public void* meth3(){  
 System.out.println("AfterEach :- This will always execute after all the test methods: \n");  
 }  
  
 @AfterAll  
 *static void* meth4(){  
 System.out.println("AfterAll :- This will execute only once and after all the test methods: \n");  
 }  
}

OUTPUT:

****