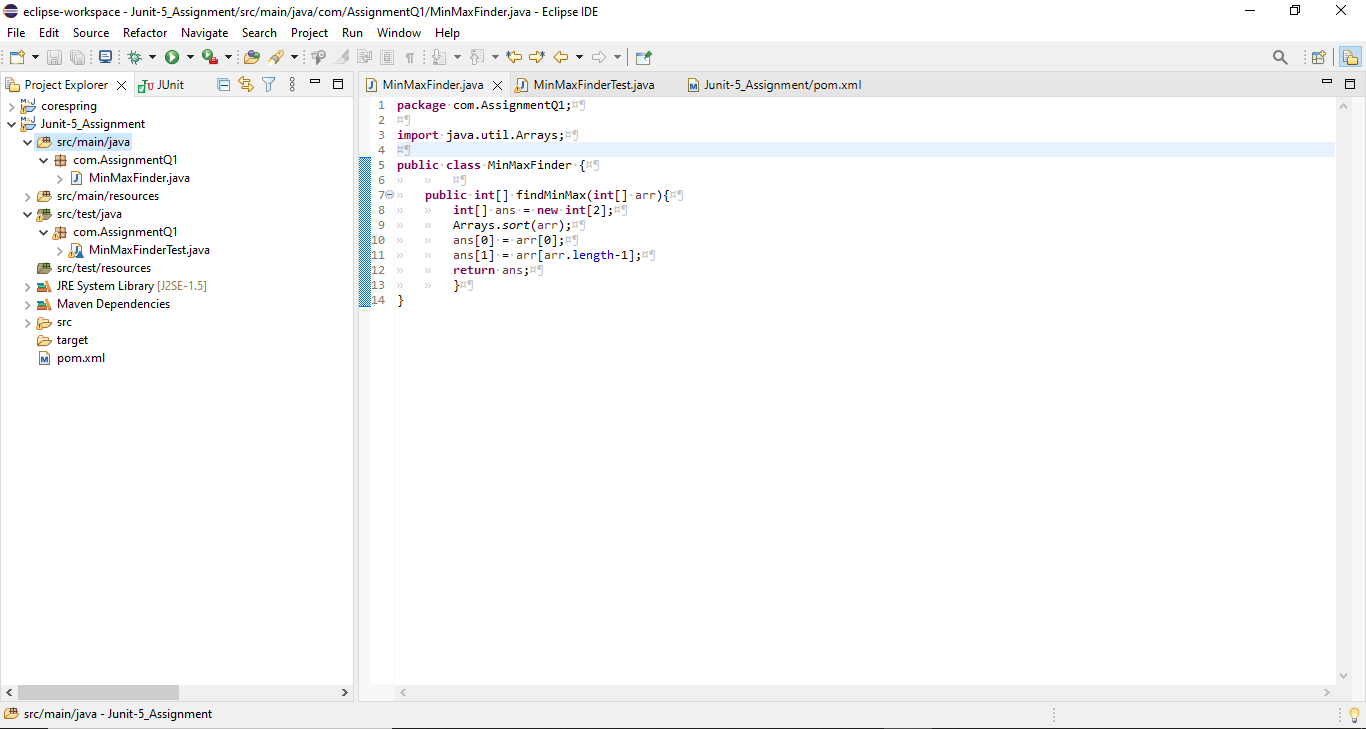
**Assignments on Junit**

**Q1.** 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.

**Created Main Class & Test class for the same:**

****

**Main Class Code:-**

**package com.AssignmentQ1;**

**import java.util.Arrays;**

**public 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;**

**}**

**}**

**Test Class Code:-**

**package AssignmentsQ1;**

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

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

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

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

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

**import AssignmentsQ1.MinMaxFinder;**

**class MinMaxFinderTest {**

**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**

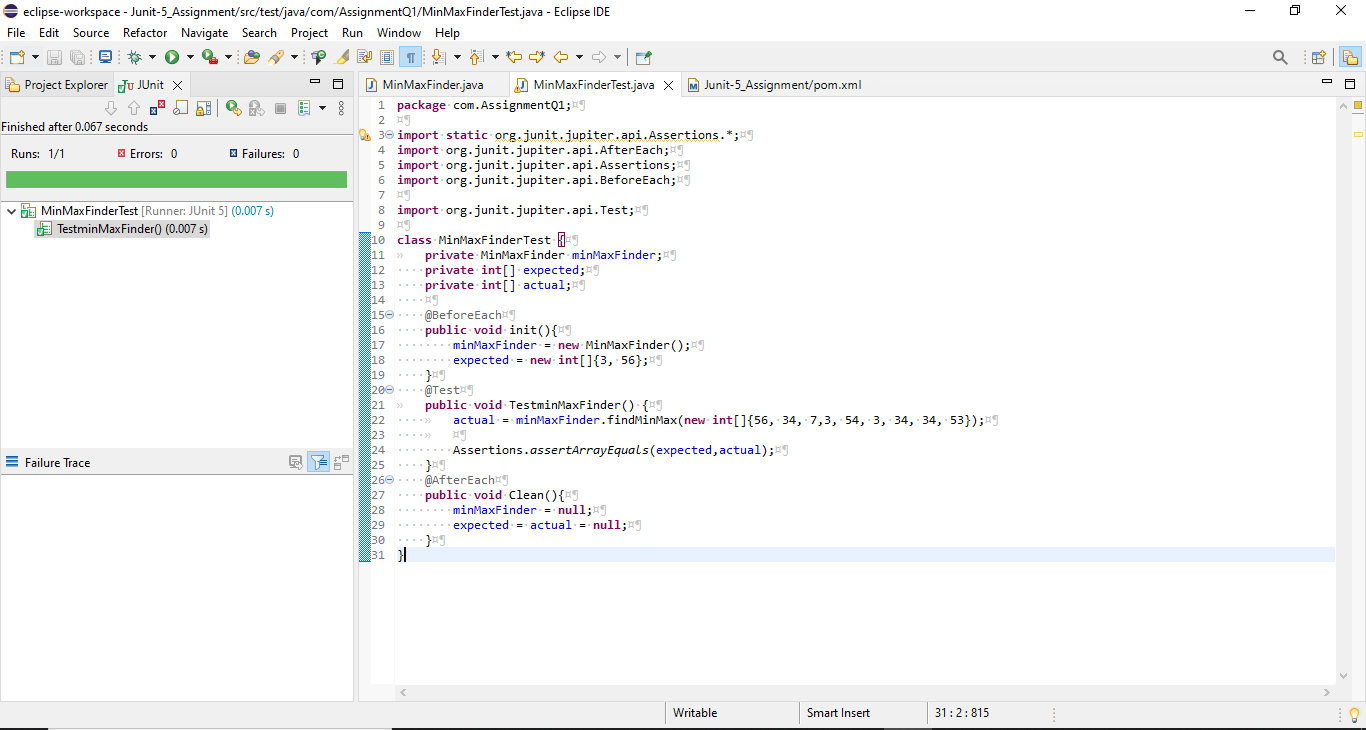
**public void Clean(){**

**minMaxFinder = null;**

**expected = actual = null;**

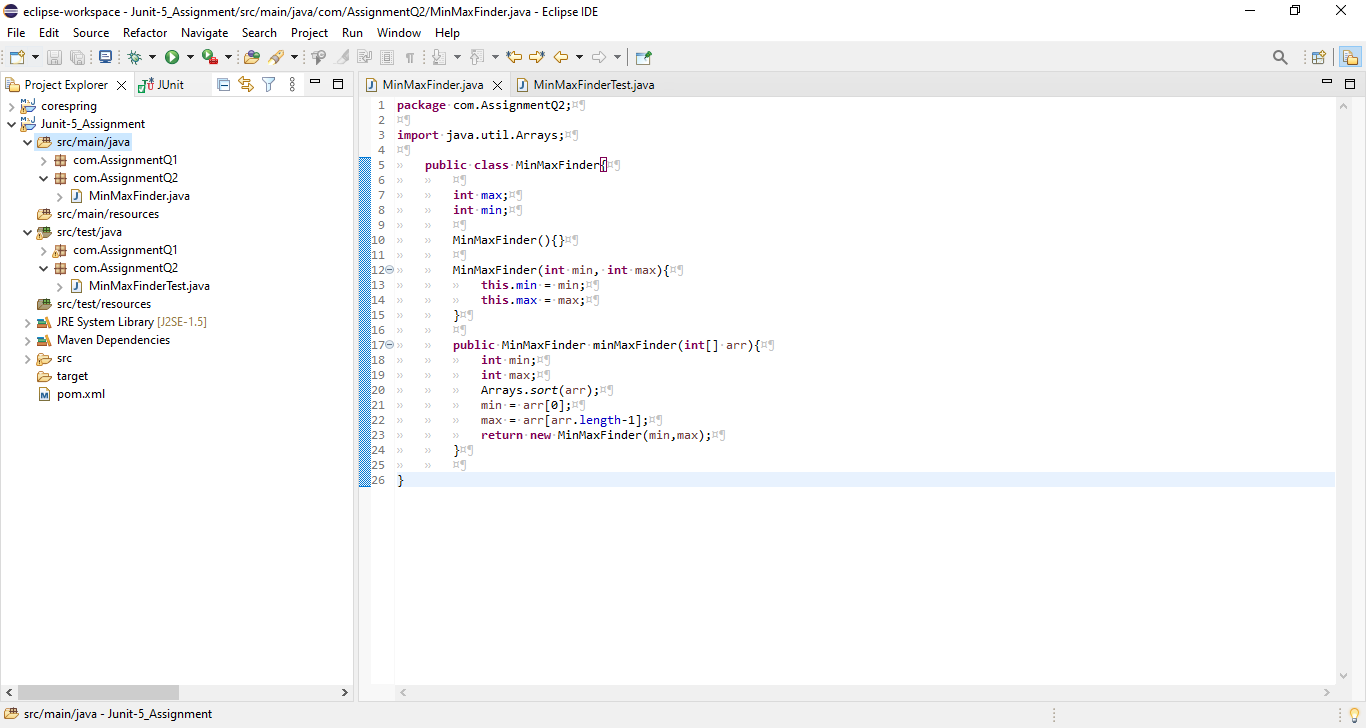
**}**

**}**

**Testing Output :**- 

**Q2.** 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.

**Created Main Class & Test class for the same:**

****

**Main Class Code:-**

**package com.AssignmentQ2;**

**import java.util.Arrays;**

**public class MinMaxFinder {**

**int max;**

**int min;**

**MinMaxFinder(){}**

**MinMaxFinder(int min, int max){**

**this.min = min;**

**this.max = max;**

**}**

**public MinMaxFinder minMaxFinder(int[] arr){**

**int min;**

**int max;**

**Arrays.*sort*(arr);**

**min = arr[0];**

**max = arr[arr.length-1];**

**return new MinMaxFinder(min,max);**

**}**

**}**

**Test Class Code:**

**package com.AssignmentQ2;**

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

**@TestMethodOrder(MethodOrderer.OrderAnnotation.class)**

**public class TestMinMaxFinder {**

**MinMaxFinder expected;**

**MinMaxFinder actual;**

**@BeforeEach**

**public void init(){**

**expected = new MinMaxFinder(3,56);**

**actual = new MinMaxFinder();**

**actual = actual.minMaxFinder(new int[]{56, 34, 7,3, 54, 3, 34, 34, 53});**

**}**

**@Test()**

**@Order(1)**

**public void testMinFinder(){**

**Assertions.*assertEquals*(expected.min,actual.min);**

**}**

**@Test**

**@Order(2)**

**public void testMaxFinder(){**

**Assertions.*assertEquals*(expected.max,actual.max);**

**}**

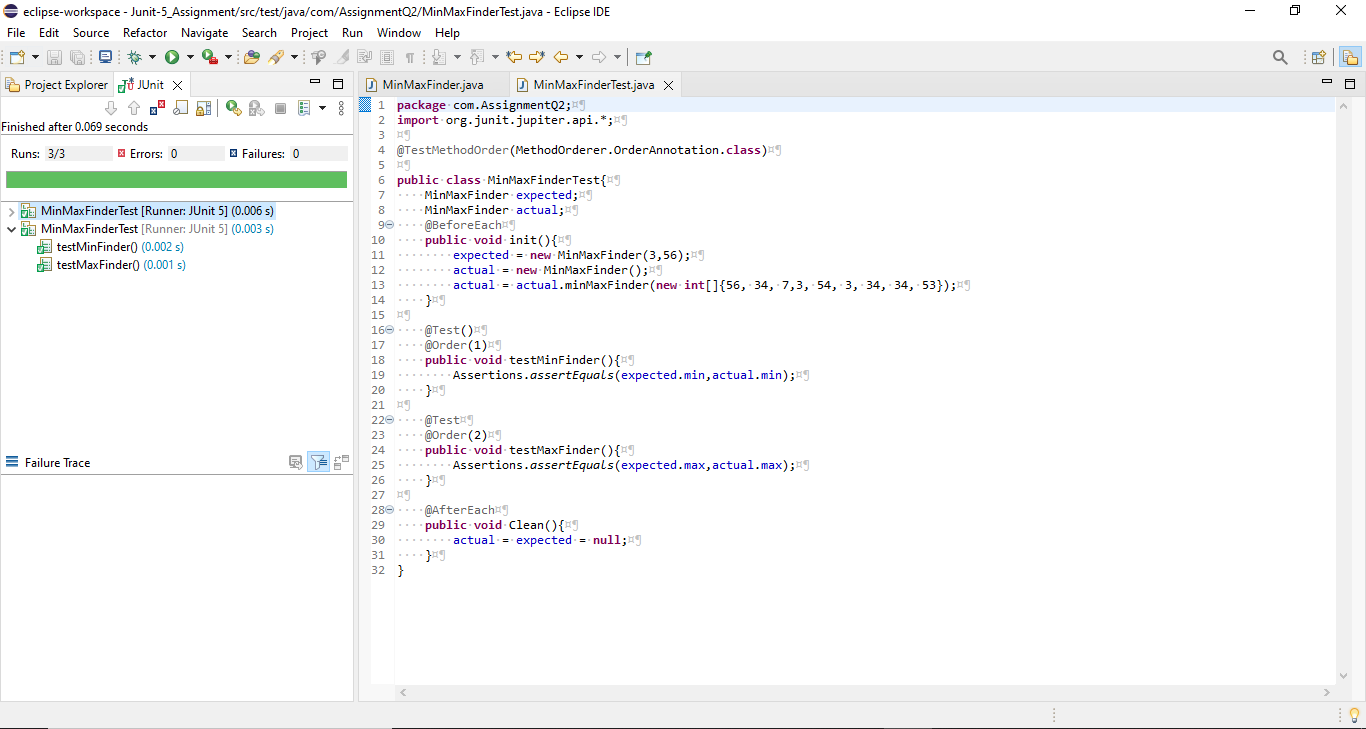
**@AfterEach**

**public void Clean(){**

**actual = expected = null;**

**}**

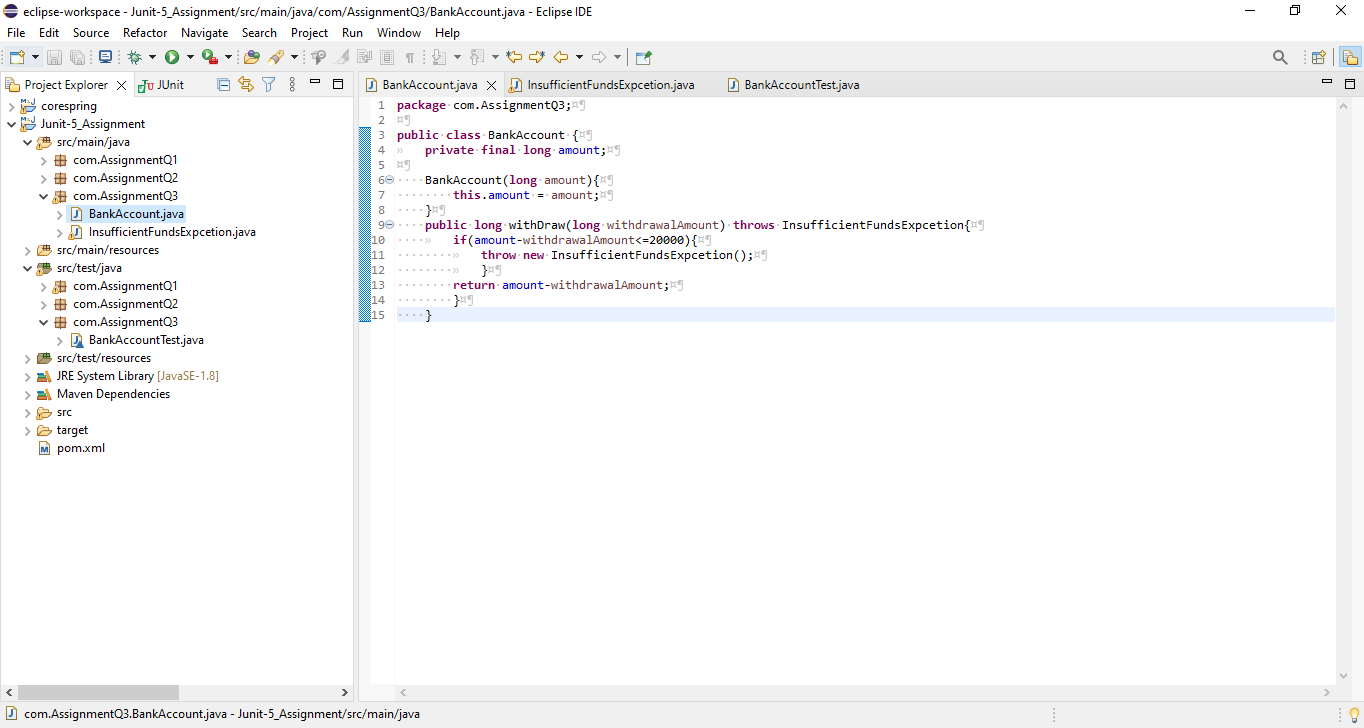
**}**

**Testing Output:- **

**Q3.** 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 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 InsufficientFundsException if the balance in the account is less than 20,000.

**Created Main Class, Test class and A Child Class Of Exception for the same:**

****

**Main Class Code:**

**package com.AssignmentQ3;**

**public 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;**

**}**

**}**

**Test Class Code: package com.AssignmentQ3;**

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

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

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

**public class BankAccountTest {**

**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));**

**});**

**}**

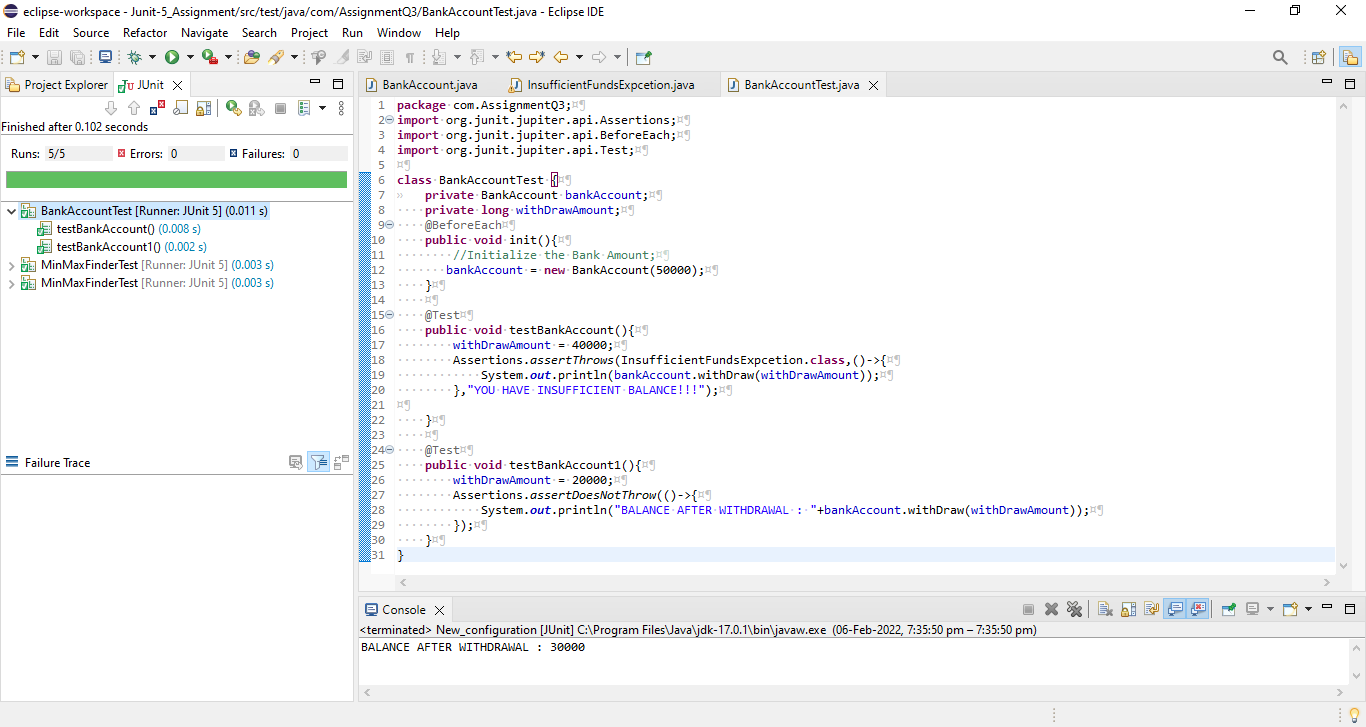
**}**

**Child Extend Exception Class Code:**

**package com.AssignmentQ3;**

**public class InsufficientFundsExpcetion extends Exception {**

**}**

**Testing Output: **

**Q4.** Write a Junit Testing to show the use of Lifecycle hooks annotation such as @BeforeAll, @BeforeEach @AfterEach and @AfterAll

**Test Class Code:**

**package com.AssignmentQ4;**

**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.Order;**

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

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

**@TestInstance(TestInstance.Lifecycle.*PER\_CLASS*)**

**class DemoTest {**

**@BeforeAll**

**static void m1(){**

**System.*out*.println("BeforeAll - This will execute only once and before all the test methods. \n");**

**}**

**@BeforeEach**

**public void m2(){**

**System.*out*.println("BeforeEach - This will always execute before all the test methods. ");**

**}**

**@Test**

**@Order(1)**

**@DisplayName("Method 5")**

**public void m5(){**

**System.*out*.println("HELLO FROM TEST METHOD-1\n");**

**}**

**@Test**

**@Order(2)**

**@DisplayName("Method 6")**

**public void m6(){**

**System.*out*.println("HELLO FROM TEST METHOD-2\n");**

**}**

**@Test**

**@Order(3)**

**@DisplayName("Method 7")**

**public void m7(){**

**System.*out*.println("HELLO FROM TEST METHOD-3\n");**

**}**

**@AfterEach**

**public void m3(){**

**System.*out*.println("AfterEach - This will always execute after all the test methods. \n");**

**}**

**@AfterAll**

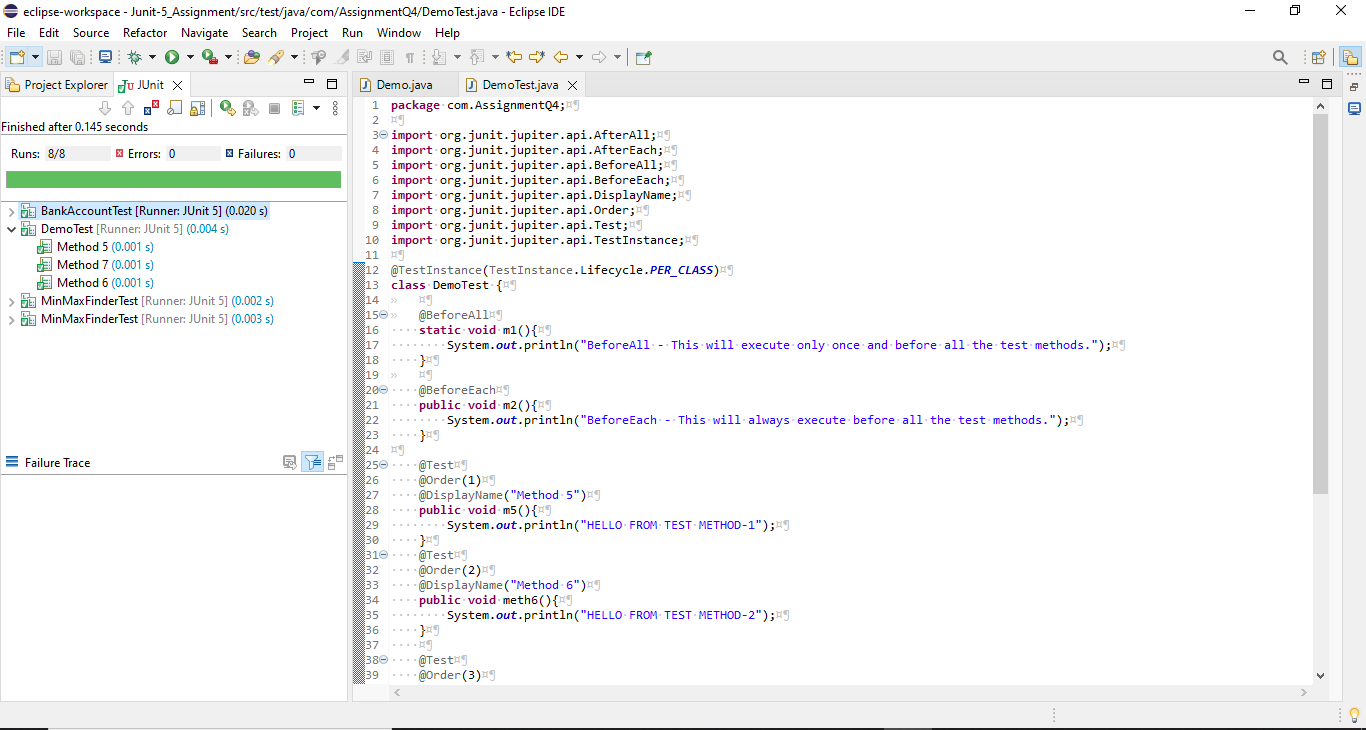
**static void m4(){**

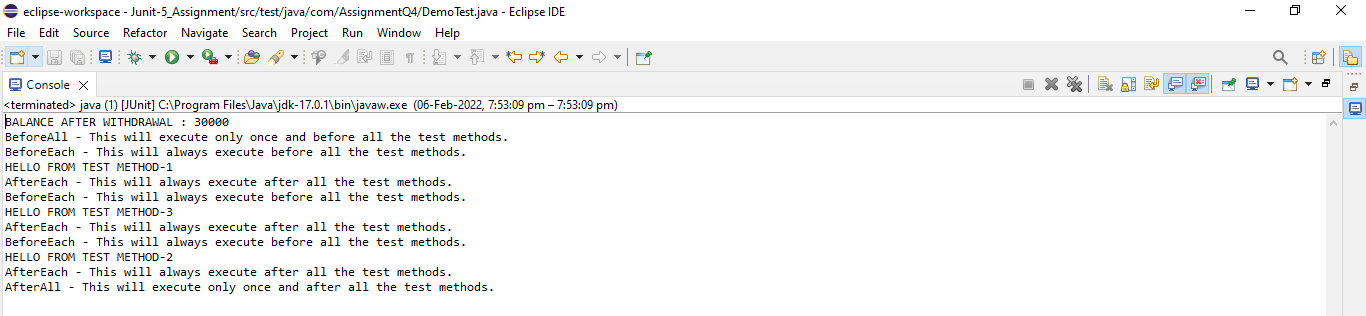
**System.*out*.println("AfterAll - This will execute only once and after all the test methods. \n");**

**}**

**}**

**Testing Output:**

****

**Console output:- **