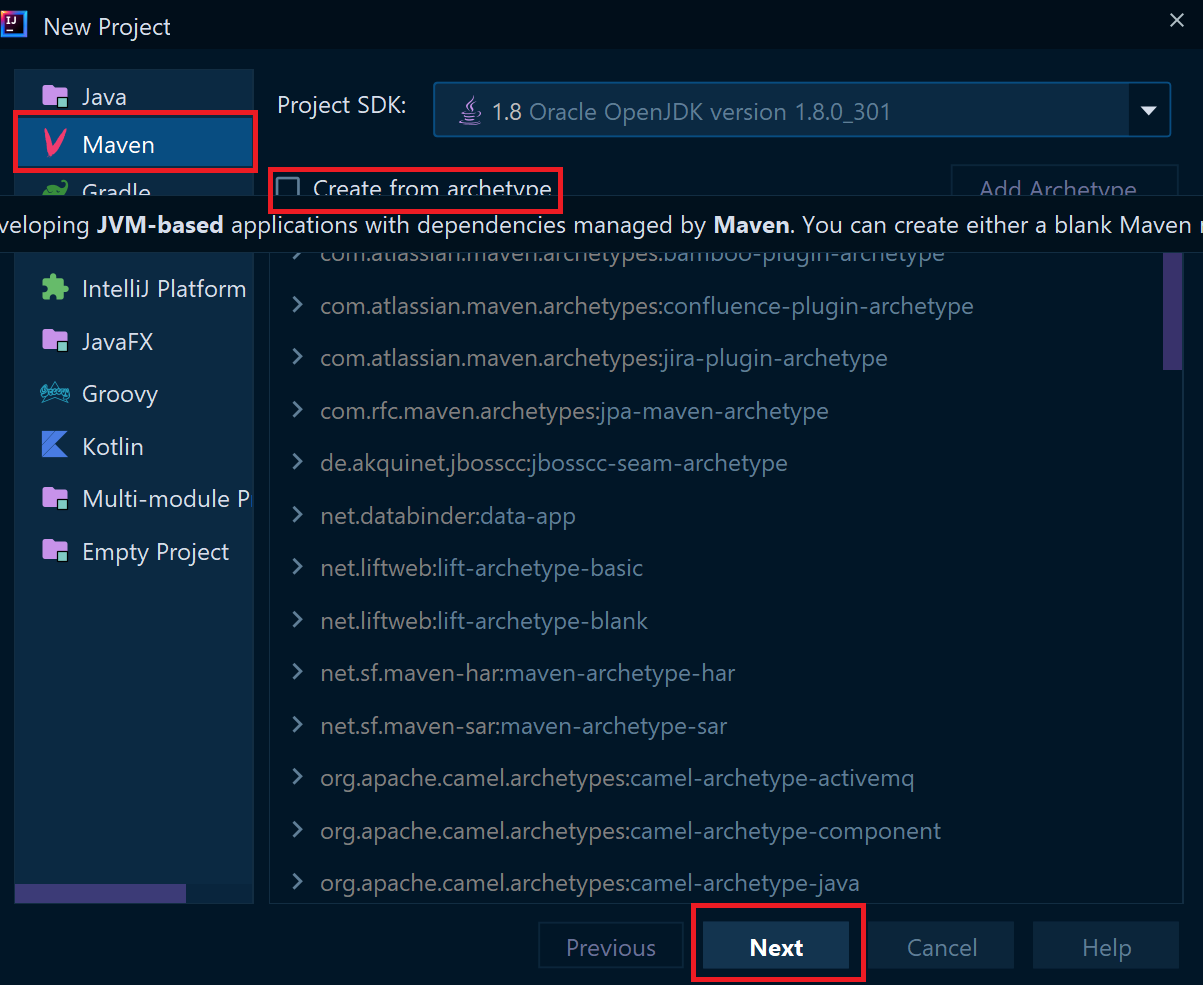
**Name: Ayush Agrawal**

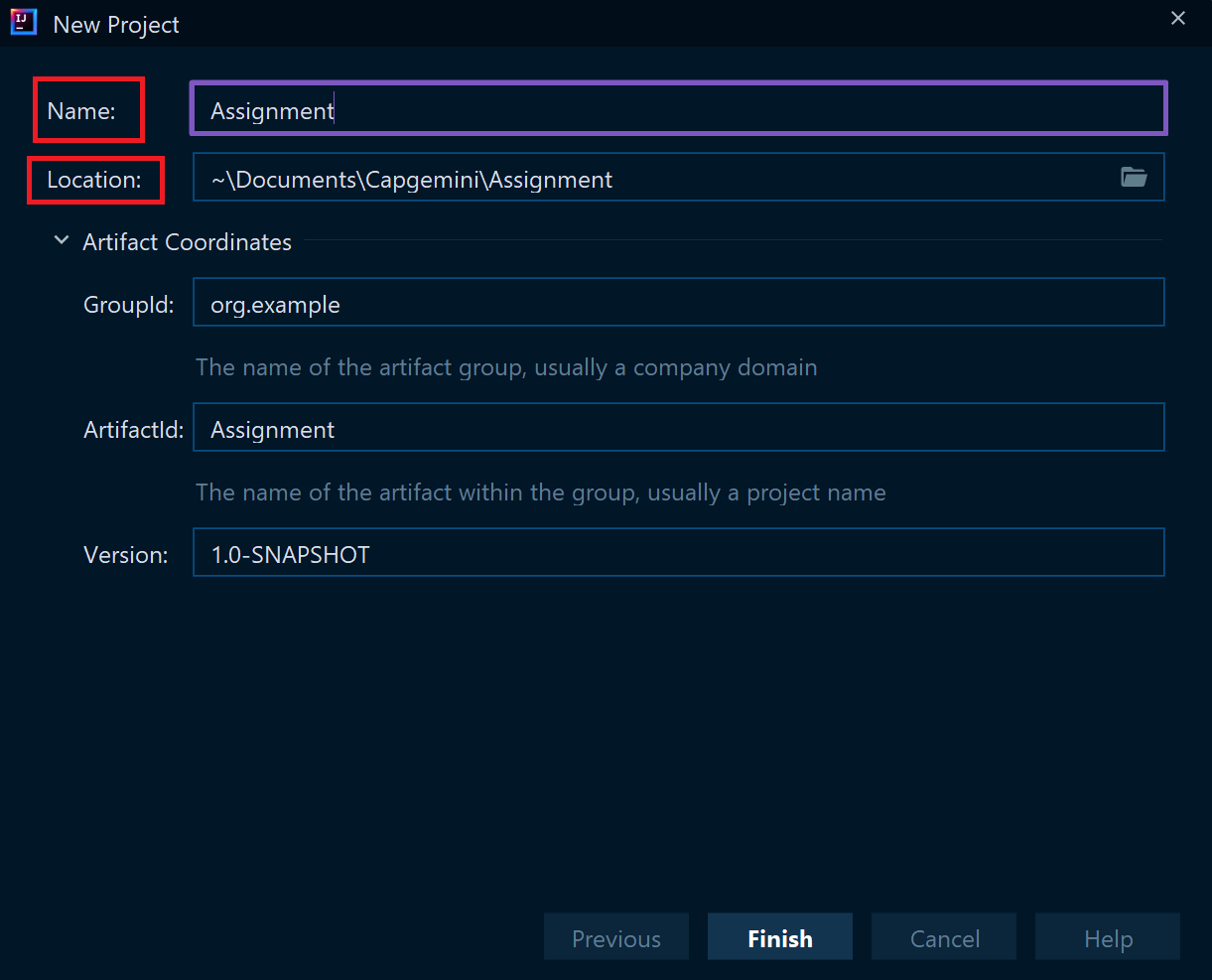
**Date: 29-01-2022**

**Step 1:-** Setup Maven Project in IntelliJ Idea Editor.

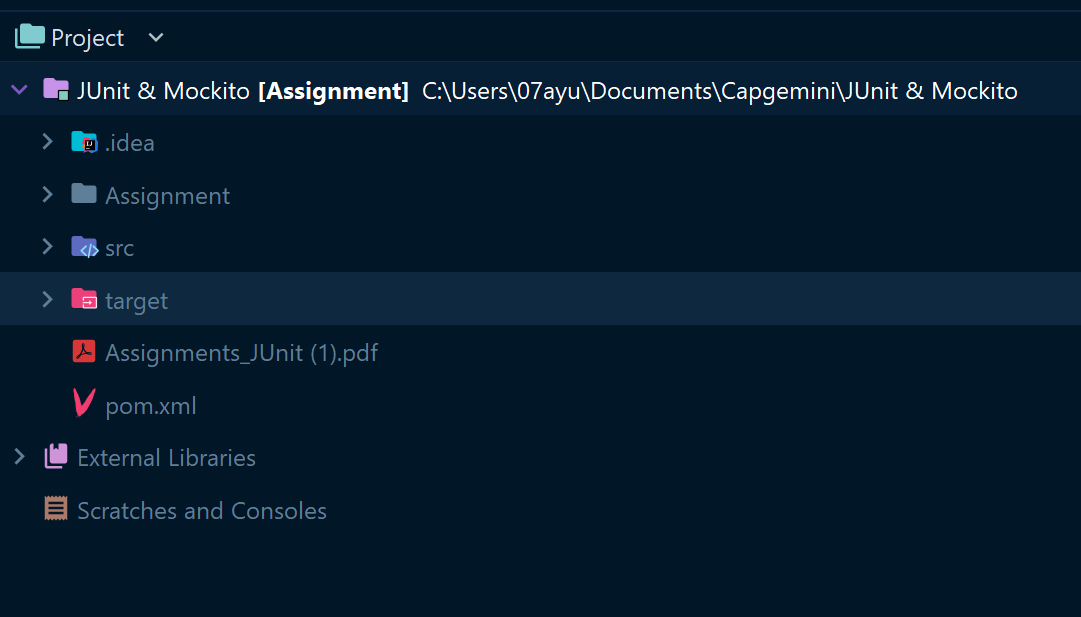


**Note:** Select Maven from the list on the left column of new project setup and it’s your choice to create from archetype (I have choose default settings) then click on next.

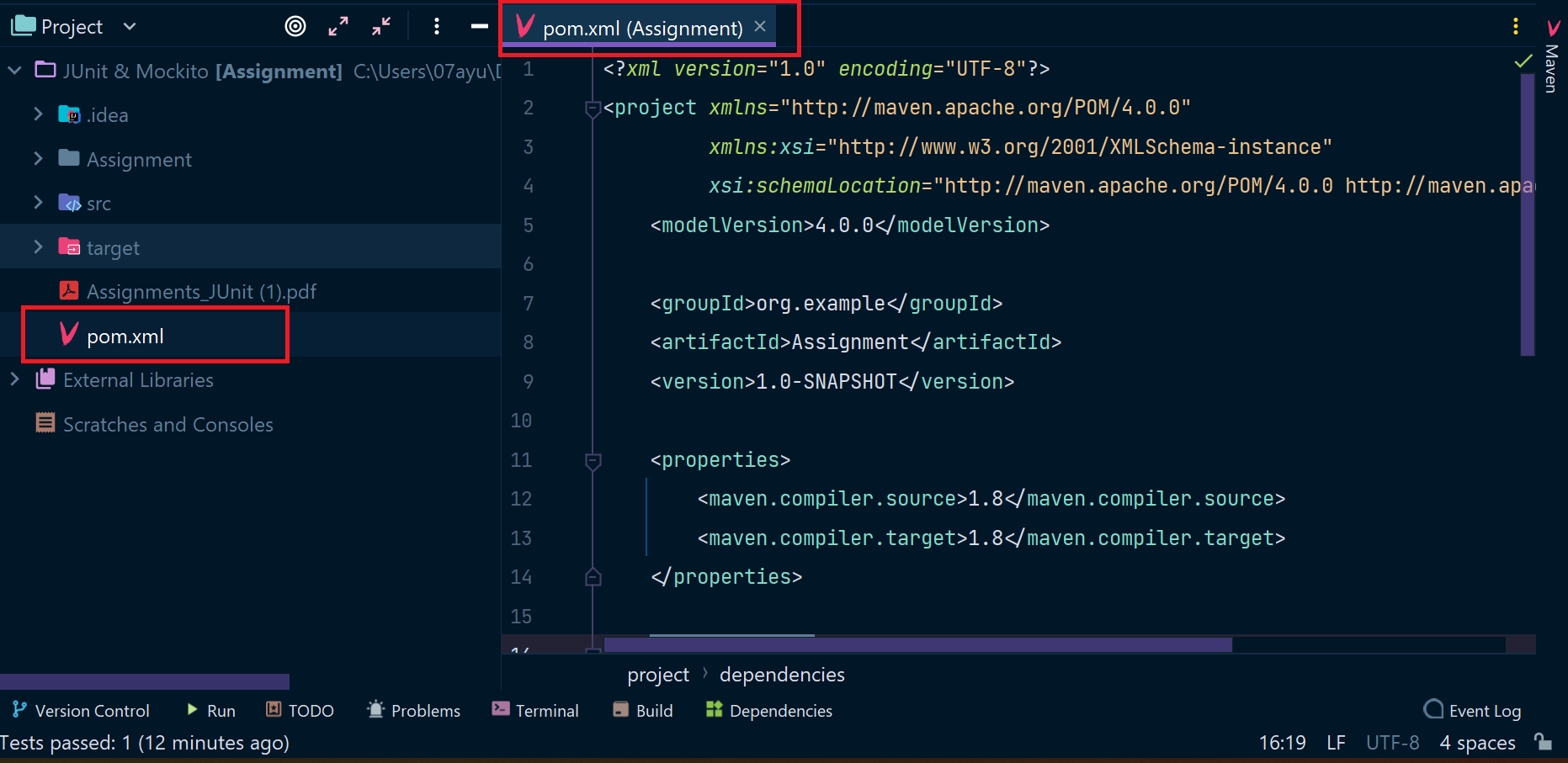
**Step 2**: Write the name of your project in my case it is “Assignment” and provide the path of your project in which directory you want to create the project. Then click on finish and your maven project has been setup in your editor.



**Step 3:** Project will look like this.



**Step 4:** Open pom.xml file to add dependencies and properties for Junit-5 framework.

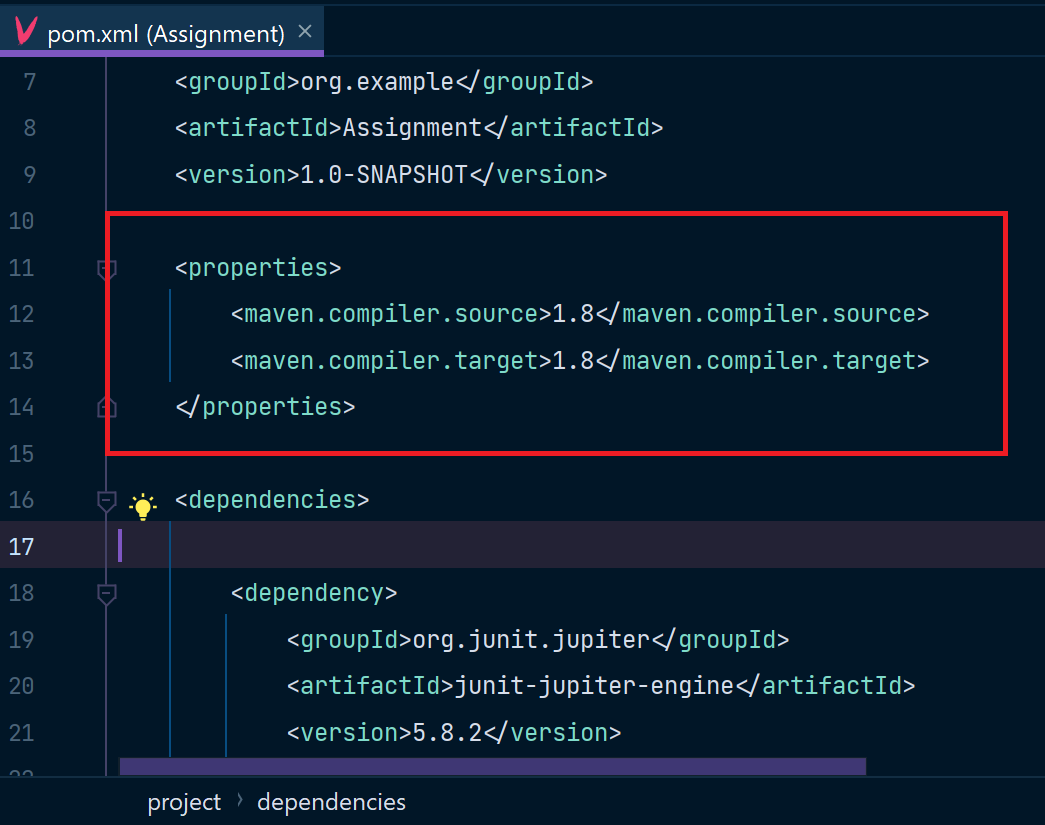


**Step 5:** Add properties for maven compiler.

**Code:**

<properties>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
</properties>

**Add the above code segment in pom.xml as below:**



**Note: I am using Java 1.8 so I have added 1.8 compiler, checked your compiler version and added the substitute compiler version.**

**Step 6:** Add Junit Jupiter Engine Maven dependency in your pom.xml.

<dependencies>  
  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-engine</artifactId>  
 <version>5.8.2</version>  
 <scope>test</scope>  
 </dependency>  
</dependencies>

**Add the above code segment in pom.xml as below:**



**Then your maven project is ready for Unit testing by Junit framework.**

**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

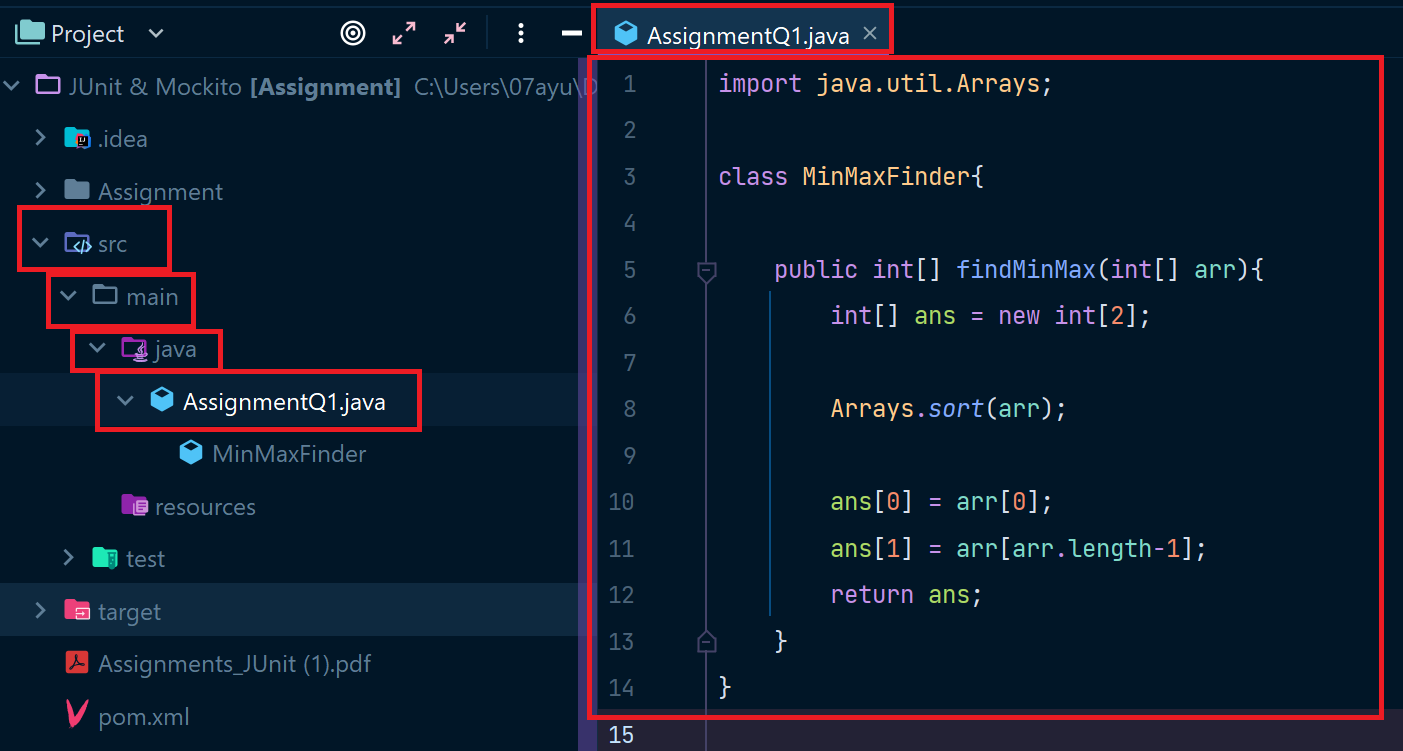
array with min and max values {3, 56} at 0th and 1st index respectively.

**Answer:**

**Code:**

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;  
 }  
}

**Add the above in src/main/java/AssignmentQ1.java file as below:**

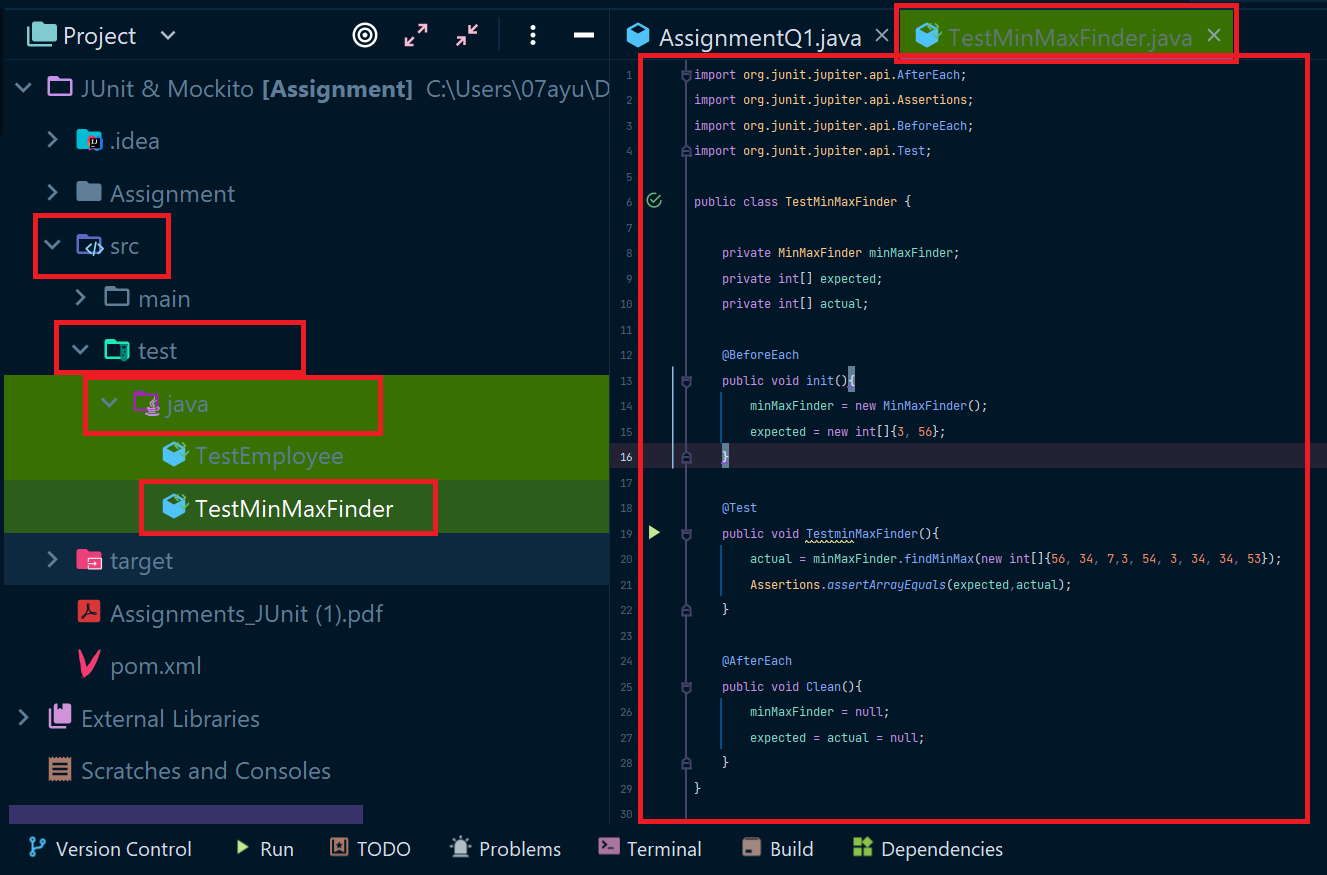


**For Testing above piece of code as per question:**

**Code for Testing:**

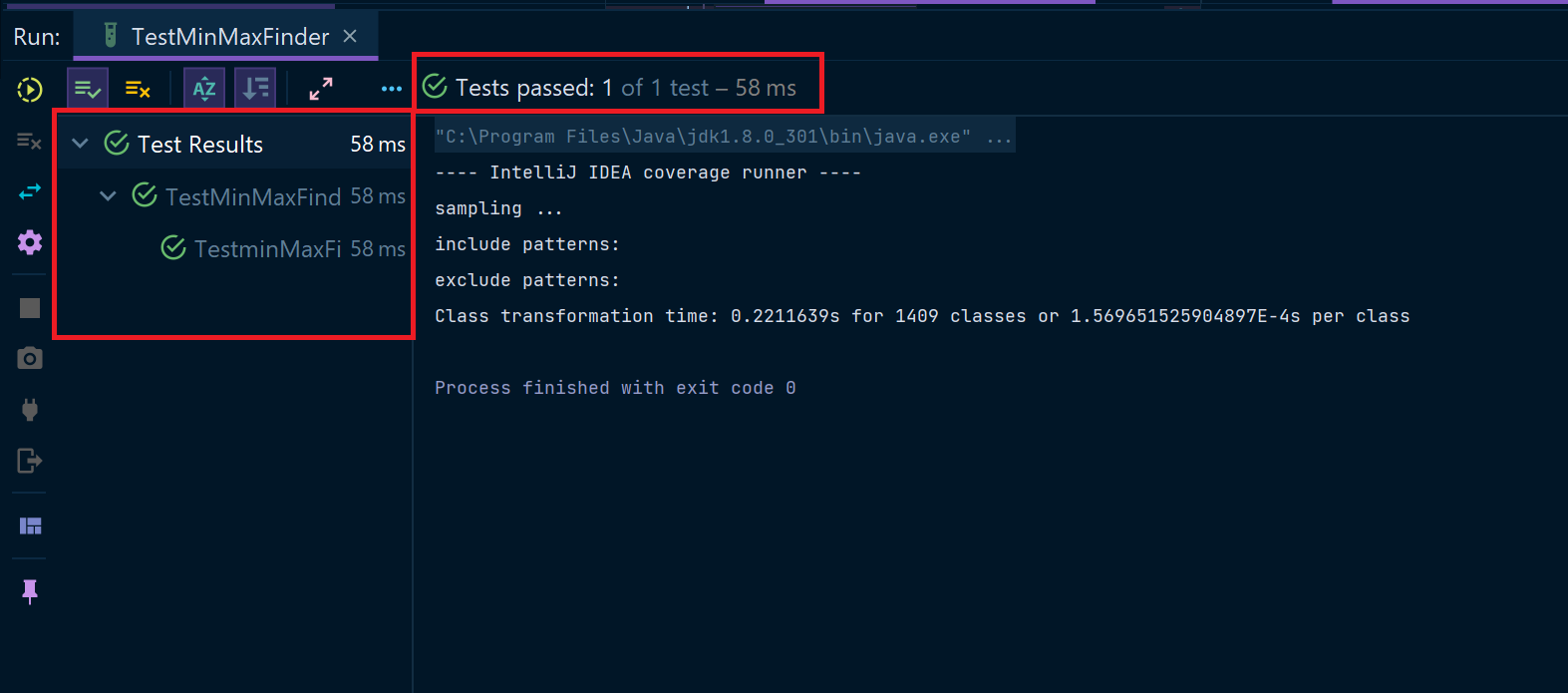
import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.Assertions;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
public class TestMinMaxFinder {  
  
 private MinMaxFinder minMaxFinder;  
 private int[] expected;  
 private int[] actual;  
  
 @BeforeEach  
 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;  
 }  
}

**Add the above in src/test/java/TestMinMaxFinder.java file as below:**



**Testing Output:**

The below is showing that there is no error or failures while testing our code and all the defined tests passed successfully. (shows build success).



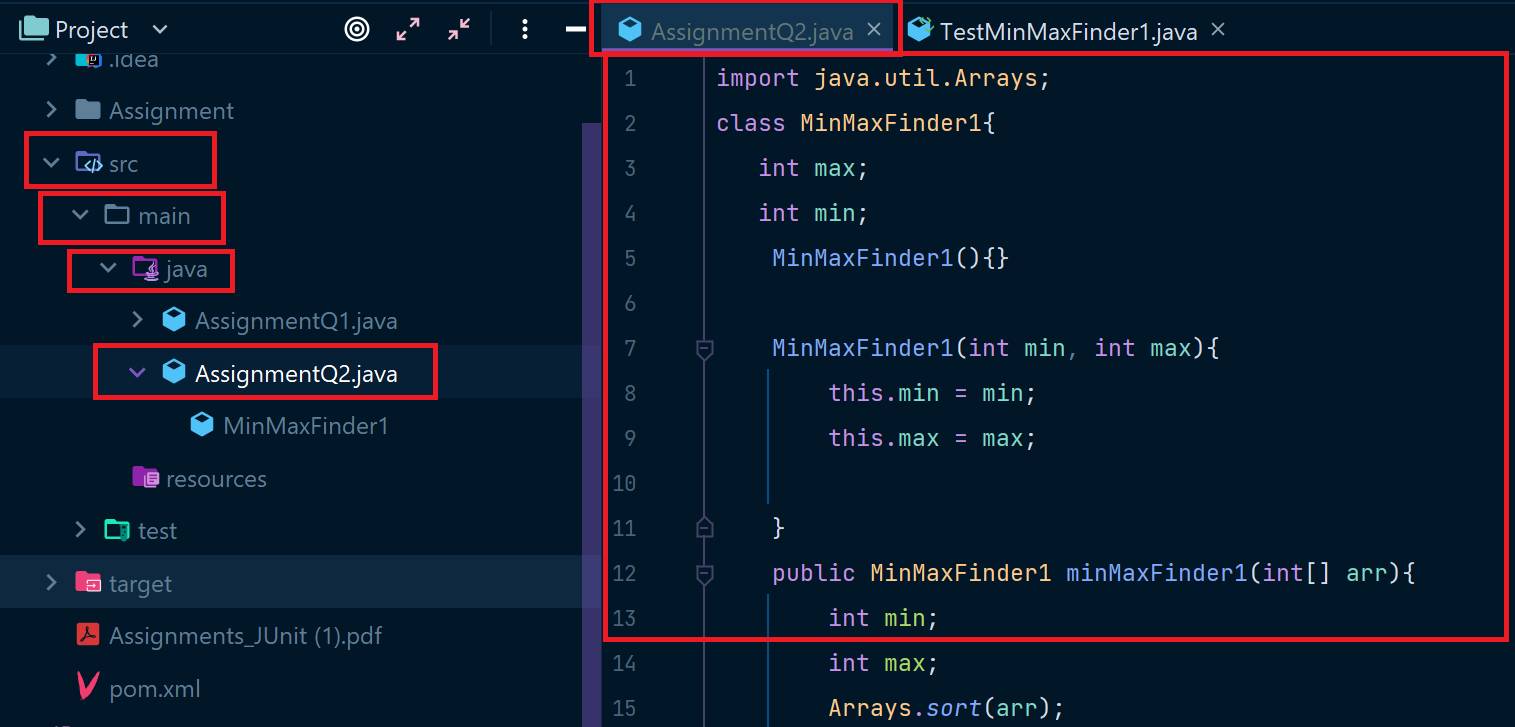
**2)** Modify the above method to return a single object representing min and max value of the pass array. Define new sets of Junit Test cases of this modified method.

**Answer:**

**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);  
 }  
}

**Add the above in src/main/java/AssignmentQ2.java file as below:**

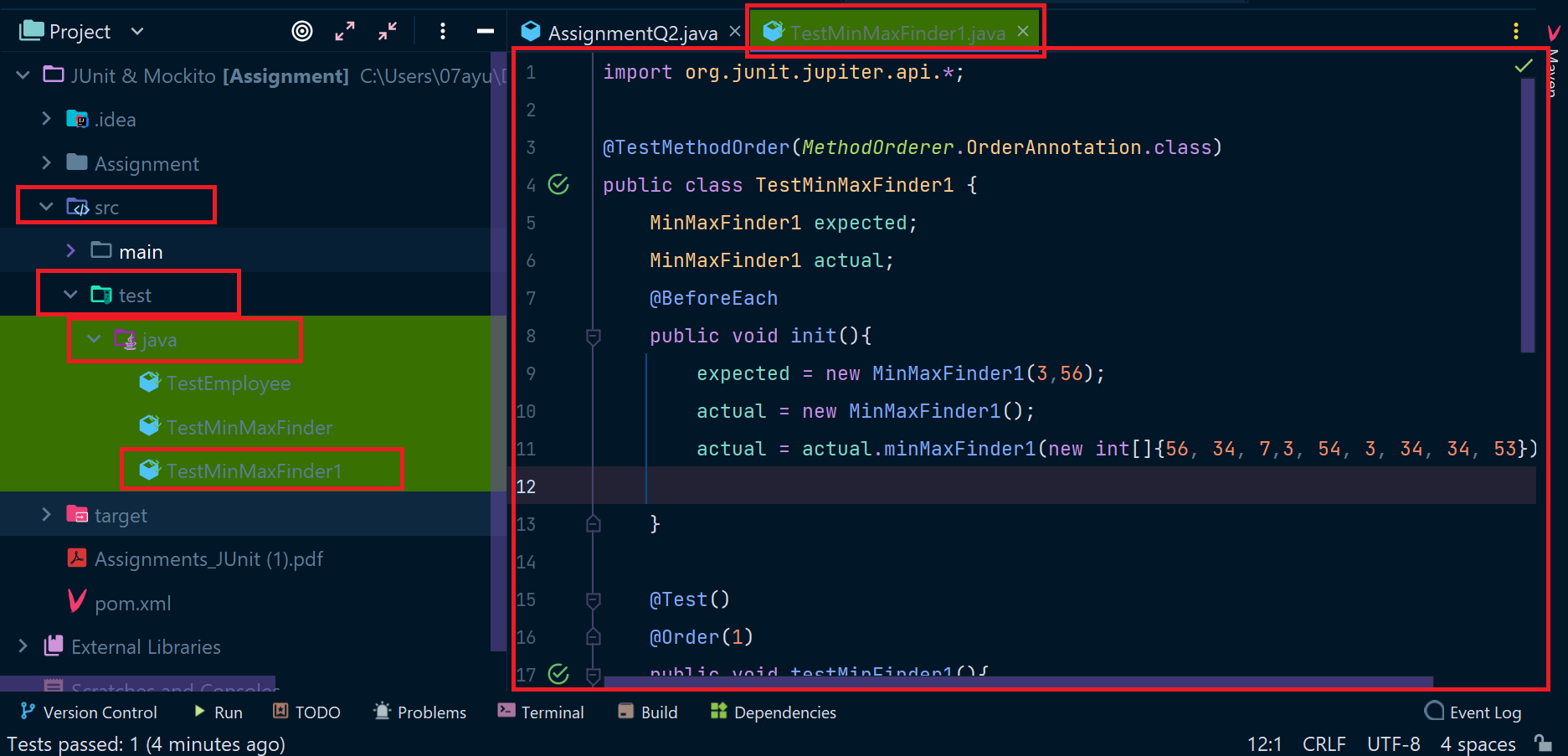


**For Testing above piece of code as per question:**

**Code for Testing:**

import org.junit.jupiter.api.\*;  
  
@TestMethodOrder(*MethodOrderer*.OrderAnnotation.class)  
public class TestMinMaxFinder1 {  
 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;  
 }  
}

**Add the above in src/test/java/TestMinMaxFinder1.java file as below:**



**Testing Output:**

The below is showing that there is no error or failures while testing our code and all the defined tests passed successfully. (shows build success).



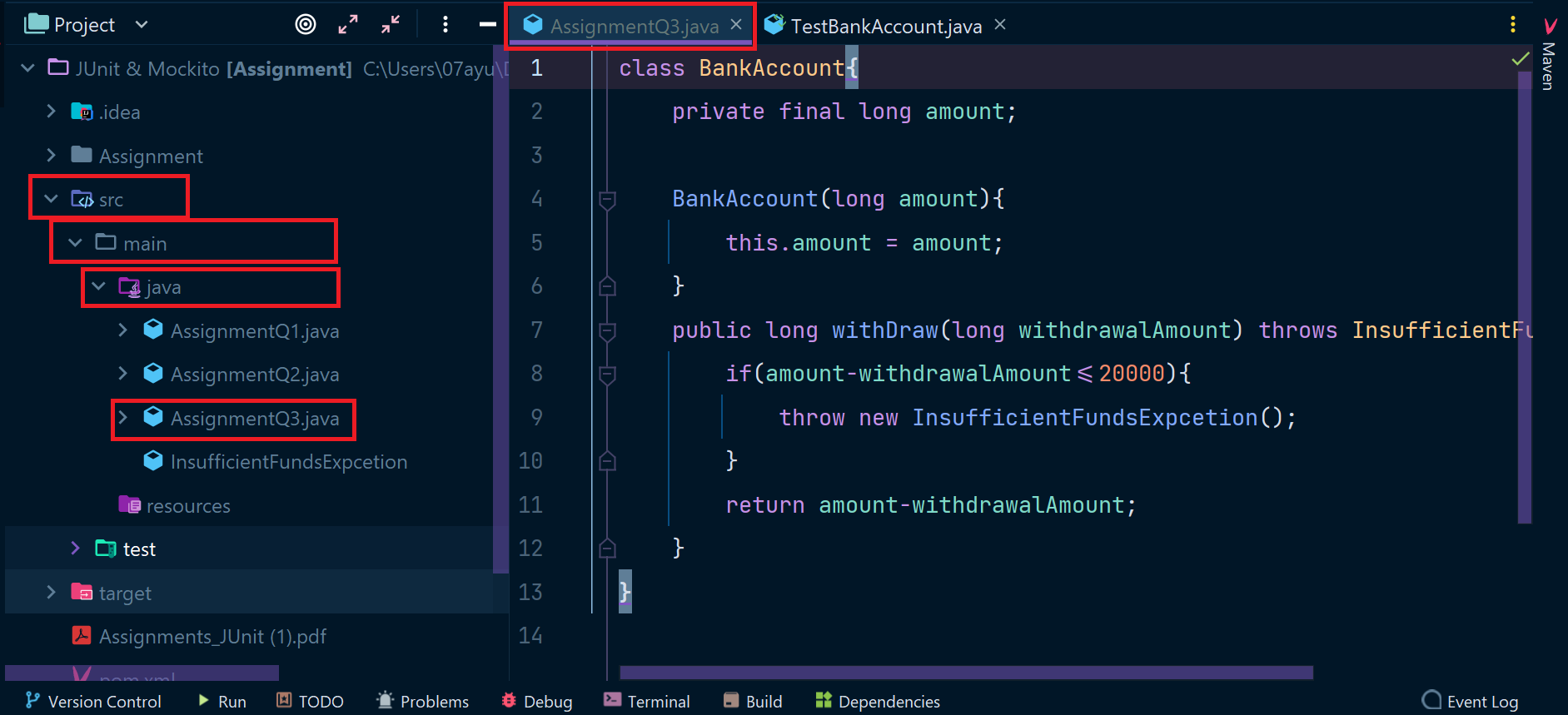
**3)** Write a BankAccount class with method withdraw which accepts amount to be withdrawn from the account (amount to be deducted from the balance of the account). In case there are insufficient funds a InsufficientFundsExpcetion should be raised. After defining the method perform Junit testing to check 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.

**Answer:**

**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;  
 }  
}

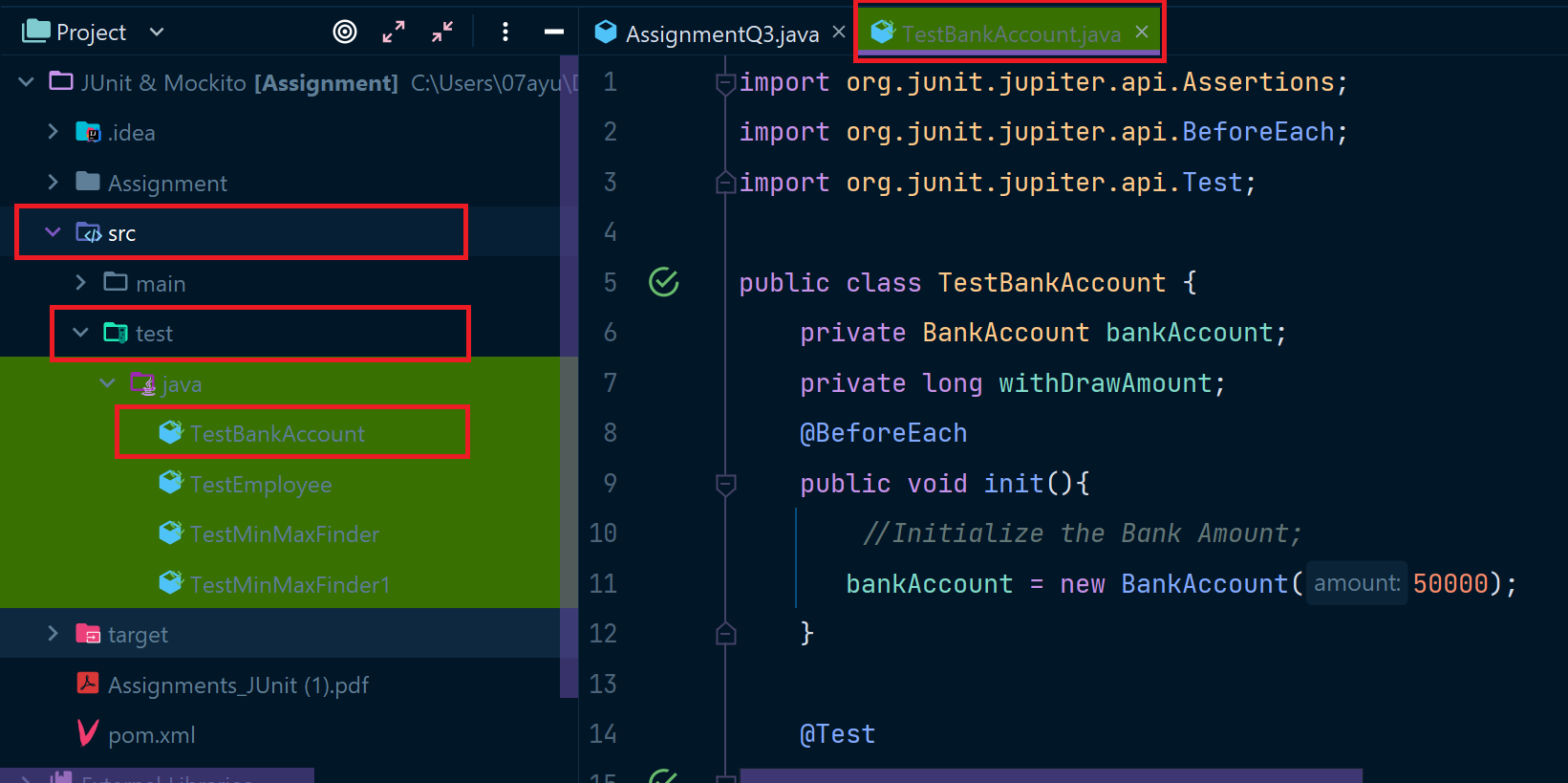
**Add the above in src/main/java/AssignmentQ3.java file as below:**



**For Testing above piece of code as per question:**

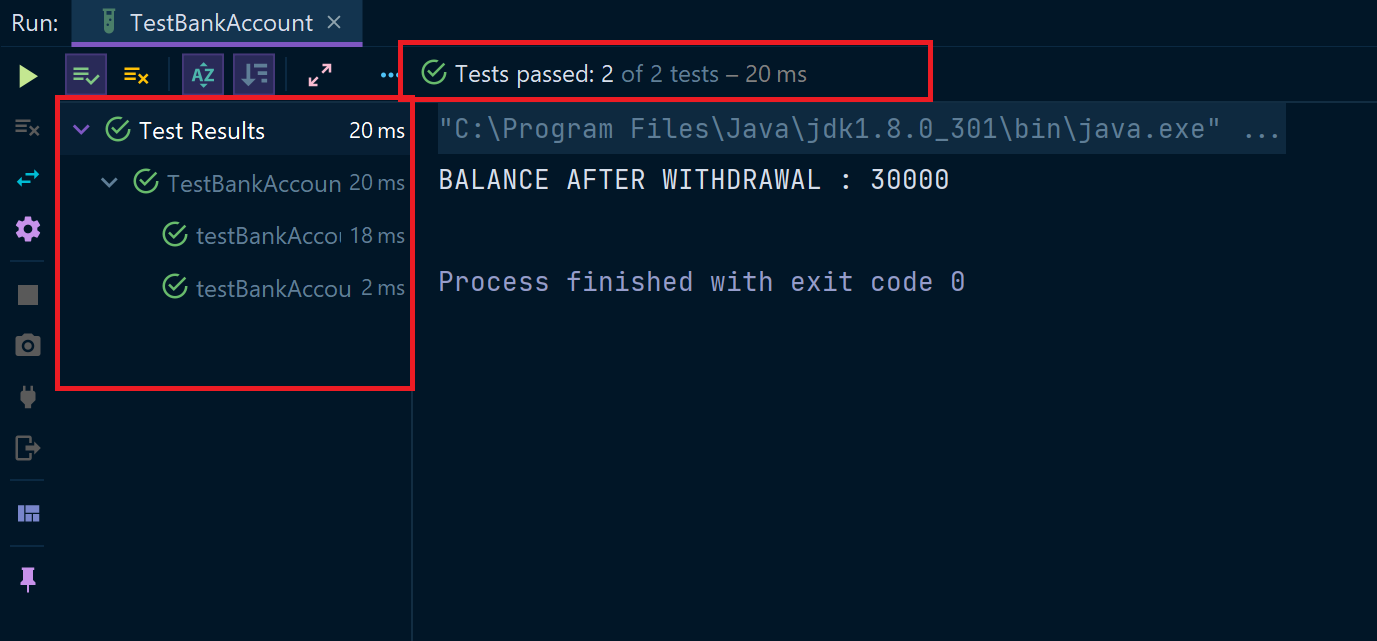
**Code for Testing:**

import org.junit.jupiter.api.Assertions;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
public class TestBankAccount {  
 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));  
 });  
 }  
}



**Testing Output:**

The below is showing that there is no error or failures while testing our code and all the defined tests passed successfully. (shows build success).



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 Assignment4 {  
  
 @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");  
 }  
}

