GMU

CS-332 Assignment 1

Prof Ammann

## Included in this document

Project Description

Individual class summary

Source Code

## Team

**VG =** Valeria Green

**JM =** Jorge L Martinez

## Collaboration Summary

* **(VG)** Designed and implemented Factor Class
* **(JM)** Designed and Implemented the Factor Class tests
* **(VG)** Designed and implemented the Prime Class
* **(JM)** Designed and implemented the Prime Class tests
* **(VG)(JM)** Worked on integrating all methods in the FindPrimeFactor class
* **(VG)** Individual class Description
* **(JM)** Project Description repo and Docs setup
* **(VG)** Added comprehensive tests

## Project GitHub Repository Link:

<https://github.com/cubangeorge/cs332group/tree/master/a1>

**Project Description :**

**Goal:** Basic familiarity with Java.

Consider the following specification:

public static int findPrimeFactor (List<Integer> a, List<Integer> b)

// Requires: a not null; b not null;

// there is some index i where b[i] is

// both prime and a factor of a[i]

//

// Effects: return the least index

// at which b[i] is a prime factor of a[i]

// E.g. findPrimeFactor ([12, 25, 18, 8], [6, 2, 3, 2]) = 2

// (Note: 6 is a factor of 12, but is not prime,

// and 2 is prime, but is not a factor of 25. However,

// 3 is a prime factor of 18. Hence, index "2" is the correct

// answer. index "3" is not a possible answer, because the

// third index is not the least index with the desired property.)

// Also note that a[] and b[] need not be of the same length.

Implement this method in a class called FindPrimeFactor. Although you should (obviously) test your implementation yourself, you do not need to include any test code. The GTA will test your code. Your solution should be minimal in the sense that it doesn't include any unnecessary code. In particular, error checking in findPrimeFactor() for parameters that do not satisfy the precondtions is not appropriate for this assignment. (It will be appropriate for assignment 2.) We will discuss this point in detail in class.

**Grading Criteria:**

* Adherence to instructions. Do what I ask for above, not something else.
* Syntax: Java compiles and runs.
* You can earn a bonus point for supplying a decent set of JUnit tests.

**Class Summary:**

**The FindPrimeFactor.java**: Works by making calls to smaller methods defined in their own static class. The overall idea is to determine whether a number is prime and then also determine if it is a factor. This is individually determined by Prime.java and Factor.java. Once a number checks out to be both of prime and a factor number, the class will return the smaller index in which this condition became true. The List<Integer> is traversed from smaller to larger index, until either a match is found or either List is fully traversed and the operation can no longer be continued. In the event that no prime factor is found the FindPrimeFactorMethod will return a negative number to indicate this search resulted in no matches.

This will return where the first instance of a prime factor was found. It examines both lists at the same indexes at the same time to see if a number in the second list is both a prime number and a factor of the number at the same index of the first list.

**Prime.java:** Was implemented using a recursive call which starts at the smallest prime and increments a counter until it becomes the same value passed as an argument. During the process, it increments it to check to see if the counter is a divisor of the target number. If a divisor is found other than 1 and the number itself, the function returns immediately to indicating the number was not prime. If no other divisors are found the function returns a 1 indicating the number is prime indeed.

**Factor.java:** Works by using the mod operator, this why when a number is a factor and mod is zero the function returns a zero. This is why it has a reversed logic and returns a zero on success. There are several edge cases that are validated within the function but this is the general idea.

Source Code :

public class Prime {

static int firstprime = 2; // used globally in this class

public static int check\_if\_prime(int num, int i) {

if (num < firstprime)

return 0; // stop condition: return a zero since is less than smallest prime

else if (i > num - 1)

return 1; // stop condition: didn't find divisors other than self and one

else if (num % i == 0)

return 0; // stop condition: return since found a divisor

return check\_if\_prime(num, i + 1); /\* Continues recursion \*/

}

public static int is\_prime(int num) {

/\*this return 1 if number is prime zero otherwise\*/

return check\_if\_prime(num, firstprime); /\* Initializes recursion \*/

}

}

import java.util.List;

public class FindPrimeFactors {

public static int findPrimeFactor (List<Integer> a, List<Integer> b) {

// Requires: a not null; b not null;

// there is some index i where b[i] is

// both prime and a factor of a[i]

//

// Effects: return the least index

// at which b[i] is a prime factor of a[i]

// E.g. findPrimeFactor ([12, 25, 18, 8], [6, 2, 3, 2]) = 2

// (Note: 6 is a factor of 12, but is not prime,

// and 2 is prime, but is not a factor of 25. However,

// 3 is a prime factor of 18. Hence, index "2" is the correct

// answer. index "3" is not a possible answer, because the

// third index is not the least index with the desired property.)

// Also note that a[] and b[] need not be of the same length.

int i =0;

int size\_a = a.size();

int size\_b = b.size();

/\*validate empty list here

\*

\*

\*/

while ( (i < size\_a && i < size\_b) ) {

if ( !integerToBoolean(

Factor.is\_factor( a.get(i),b.get(i) )

) &&

integerToBoolean(

Prime.is\_prime(b.get(i))

)

) {

return i;}

i++;

} //end while loop

//if it gets here no prime factors were found

return -1;

}

public static boolean integerToBoolean(int i) {

//returns true when i is anything except zero

//if i==0 returns false

if(i==0)

return false;

return true;

}

}

public class Factor {

public static int is\_factor(int a, int b){

//this method HAS REVERSE LOGIC!!!!!

//it will return 0 if b is a factor of a, or return 1 otherwise

if (a<0) //flip a's sign if negative

a=a\*-1;

if (b==0) {// this covers my zero edge cases since 0\*0 = 0

if (a==0) return 0;

else return 1;

}

if (b>a) return 1 ; // edge case where b is a greater number

if (b<0 && (a\*a<b\*b)) return 1; //case (4, -12) b is neg and abs(b)>abs(a)

return a%b;

}//end is\_factor

}

import static org.junit.jupiter.api.Assertions.assertEquals;

import java.util.Arrays;

import java.util.HashMap;

import java.util.List;

import java.util.Random;

import org.junit.jupiter.api.Test;

class Test\_Suite {

//test data arrays for more comprehensive testing

public static int[] nonPrimes = new int[] {0, 1, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 25, 26, 27, 28, 30, 32, 33, 34, 35, 36, 38, 39, 40, 42, 44, 45, 46, 48, 49, 50, 51, 52, 54, 55, 56, 57, 58, 60, 62, 63 };

public static int[] primes = new int[] {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97};

public static int[] compositeNumbers = new int[] {4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 25, 26, 27, 28, 30, 32, 33, 34, 35, 36, 38, 39, 40, 42, 44, 45, 46, 48, 49, 50, 51, 52, 54, 55, 56, 57, 58, 60, 62, 63, 64, 65, 66, 68, 69, 70, 72, 74, 75, 76, 77, 78, 80, 81, 82, 84, 85, 86, 87, 88, 90, 91, 92, 93, 94, 95, 96, 98, 99, 100};

Random numbersToTest = new Random();

@Test

void test\_is\_prime() {

assertEquals(0,Prime.is\_prime(1));//check not primes less than 1st prime

assertEquals(0,Prime.is\_prime(0));

assertEquals(1,Prime.is\_prime(2));//checking primes

assertEquals(1,Prime.is\_prime(3));

assertEquals(1,Prime.is\_prime(5));

assertEquals(1,Prime.is\_prime(7));

assertEquals(1,Prime.is\_prime(11));

assertEquals(0,Prime.is\_prime(4));//check not primes more than first prime

assertEquals(0,Prime.is\_prime(6));

assertEquals(0,Prime.is\_prime(10));

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

assertEquals(0, Prime.is\_prime(nonPrimes[i]));

}

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

assertEquals(1, Prime.is\_prime(primes[i]));

}

}

@Test

void test\_is\_factor() {

assertEquals(0,Factor.is\_factor(0, 0)); //0\*0 =0

assertEquals(1,Factor.is\_factor(4, 0)); //0\*anything = 0

assertEquals(0,Factor.is\_factor(4, 1)); //1\*4 = 4

assertEquals(0,Factor.is\_factor(4, 4)); //4\*1 = 4

assertEquals(0,Factor.is\_factor(4, 2));

assertEquals(0,Factor.is\_factor(4, -2));

assertEquals(0,Factor.is\_factor(4, -4));

assertEquals(0,Factor.is\_factor(4, -1));

assertEquals(0,Factor.is\_factor(4, 2));

assertEquals(1,Factor.is\_factor(4, 3));

assertEquals(1,Factor.is\_factor(4, 6));

assertEquals(1,Factor.is\_factor(4, 8));

assertEquals(1,Factor.is\_factor(3, -12));

assertEquals(0,Factor.is\_factor(3, -3));

assertEquals(1,Factor.is\_factor(3, -5));

//cases with a negative "a" variable

assertEquals(1,Factor.is\_factor(-1, 0));//zero times anything is always zero

assertEquals(0,Factor.is\_factor(-0, 0));//zero times anything is always zero

assertEquals(0,Factor.is\_factor(-4, 2)); //2\*-2 =-4

assertEquals(0,Factor.is\_factor(-4, -2)); //2\*-2 =-4

assertEquals(1,Factor.is\_factor(-4, -6));

assertEquals(1,Factor.is\_factor(-4, 6));

assertEquals(1,Factor.is\_factor(-4, -8)); //8\*-.5 =-4

assertEquals(1,Factor.is\_factor(-4, 8)); //8\*.5 =-4

//insert tests to determine if negative or not

for (int j = 0; j < compositeNumbers.length; j++) {

for (int i = 1; i < compositeNumbers.length; i++) {

if (compositeNumbers[j] % i == 0) {

assertEquals(0,Factor.is\_factor(compositeNumbers[j], i));

}

if ((-1 \* compositeNumbers[j]) % (-1\* i) == 0) {

assertEquals(0,Factor.is\_factor(compositeNumbers[j], i));

}

}

}

}

@Test

void test\_integerToBoolean(){

assertEquals(true, FindPrimeFactors.integerToBoolean(1));

assertEquals(false, FindPrimeFactors.integerToBoolean(0));

assertEquals(true, FindPrimeFactors.integerToBoolean(4));

assertEquals(true, FindPrimeFactors.integerToBoolean(-1));

for (int i = 0; i < 100; i++) {

if (i != 0) {

assertEquals(true, FindPrimeFactors.integerToBoolean(i));

} else {

assertEquals(false, FindPrimeFactors.integerToBoolean(i));

}

}

}//end test

@Test

void test\_fPF\_1() {

//here the first number is a prime factor

List<Integer> a = Arrays.asList(12, 24, 3, 4, 5);

List<Integer> b = Arrays.asList(3, 2, 16, 1, 5); //list with pf @ 0

assertEquals(0,FindPrimeFactors.findPrimeFactor(a, b));

List<Integer> c = Arrays.asList(30, 79, 6, 20, 83);

List<Integer> d = Arrays.asList(5, 34, 3, 4, 5);

assertEquals(0, FindPrimeFactors.findPrimeFactor(c, d));

List<Integer> e = Arrays.asList(10, 50, 4, 0, 83);

List<Integer> f = Arrays.asList(2, 2, 3, 4, 5);

assertEquals(0, FindPrimeFactors.findPrimeFactor(e, f));

List<Integer> g = Arrays.asList(33, 24, 35, 107, 32);

List<Integer> h = Arrays.asList(3, 3, 5, 4, 4);

assertEquals(0, FindPrimeFactors.findPrimeFactor(g, h));

}//end test

@Test

void test\_fPF\_2() {

//here the last number is a prime factor

List<Integer> a = Arrays.asList(5, 23, 3, 34, 25);

List<Integer> b = Arrays.asList(3, 2, 16, 12, 5); //list with pf @ 4

assertEquals(4,FindPrimeFactors.findPrimeFactor(a, b));

List<Integer> c = Arrays.asList(103, 67, 100, 34, 35);

List<Integer> d = Arrays.asList(3, 10, 25, 98, 7); //list with pf @ 4

assertEquals(4,FindPrimeFactors.findPrimeFactor(c, d));

List<Integer> e = Arrays.asList(67, 60, 151, 92, 24);

List<Integer> f = Arrays.asList(9, 28, 30, 14, 3); //list with pf @ 4

assertEquals(4,FindPrimeFactors.findPrimeFactor(e, f));

}//end test

@Test

void test\_fPF\_3() {

//here the middle index number is a prime factor

List<Integer> a = Arrays.asList(5 ,23,25,34 ,24);

List<Integer> b = Arrays.asList(3 ,2 ,5 ,12 ,5); //list with pf @ 2

assertEquals(2,FindPrimeFactors.findPrimeFactor(a, b));

}//end test

@Test

void test\_fPF\_4() {

//here the middle index number and last are a prime factor

List<Integer> a = Arrays.asList(5 ,23,25,34 ,25);

List<Integer> b = Arrays.asList(3 ,2 ,5 ,12 ,5); //list with pf @ 2

assertEquals(2,FindPrimeFactors.findPrimeFactor(a, b));

}//end test

@Test

void test\_fPF\_5() {

//here no prime factor

List<Integer> a = Arrays.asList(5 ,23,66,34 ,23);

List<Integer> b = Arrays.asList(3 ,2 ,5 ,12 ,5); //list with pf @ -1

assertEquals(-1,FindPrimeFactors.findPrimeFactor(a, b));

}//end test

}