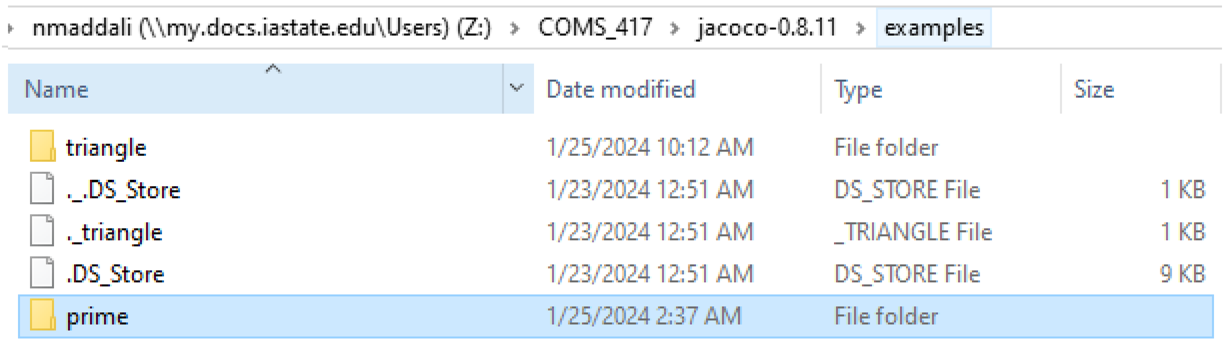
Neha Maddali

ComS417

Assignment 1: Code Coverage

Problem 1

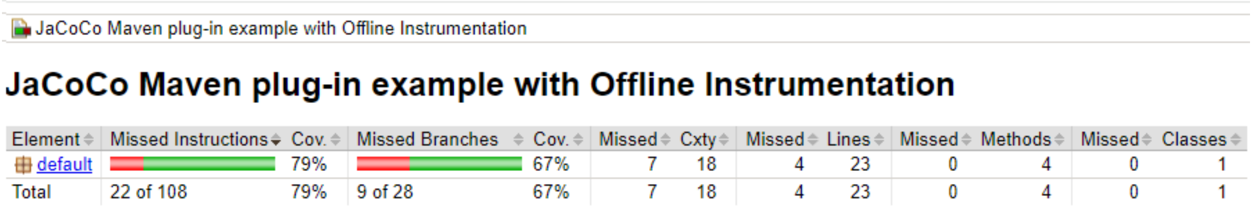


nmaddali (\\my.docs.iastate.edu\Users) (Z:) › COMS_417 › jacoco-0.8.11 › examples

Name	Date modified	Type	Size
triangle	1/25/2024 10:12 AM	File folder	
._.DS_Store	1/23/2024 12:51 AM	DS_STORE File	1 KB
._triangle	1/23/2024 12:51 AM	_TRIANGLE File	1 KB
.DS_Store	1/23/2024 12:51 AM	DS_STORE File	9 KB
prime	1/25/2024 2:37 AM	File folder	

Problem 2
The mvn commands were ran

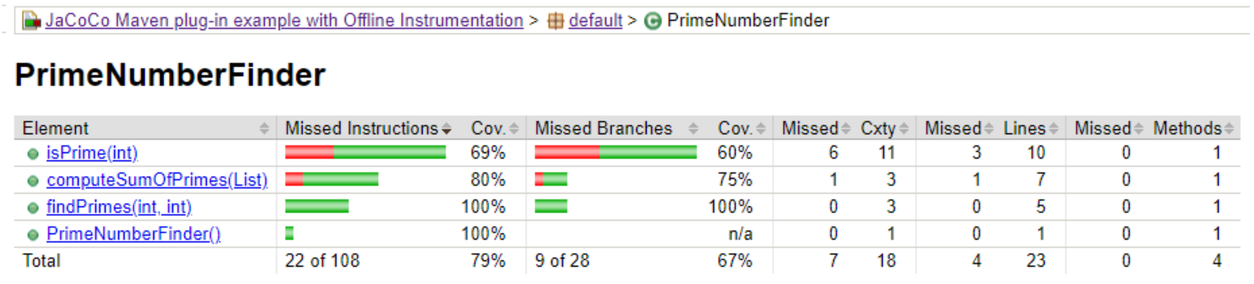
Problem 3
Initial reports: Code Coverage and Junit. The Source Code annotation has the source lines containing executable code color coded. It tells us which lines were fully covered, partly covered and lines that have not been executed at all. Overall summary of these results are found in the sites folder reports.



JaCoCo Maven plug-in example with Offline Instrumentation

JaCoCo Maven plug-in example with Offline Instrumentation

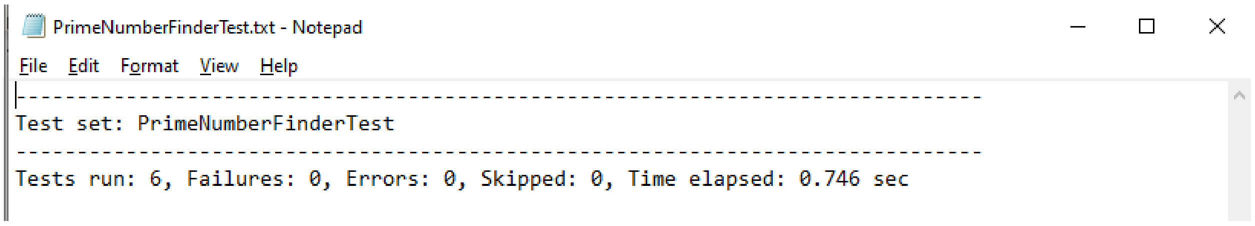
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
default	🟥🟩	79%	🟥🟩	67%	7	18	4	23	0	4	0	1
Total	22 of 108	79%	9 of 28	67%	7	18	4	23	0	4	0	1



JaCoCo Maven plug-in example with Offline Instrumentation > default > PrimeNumberFinder

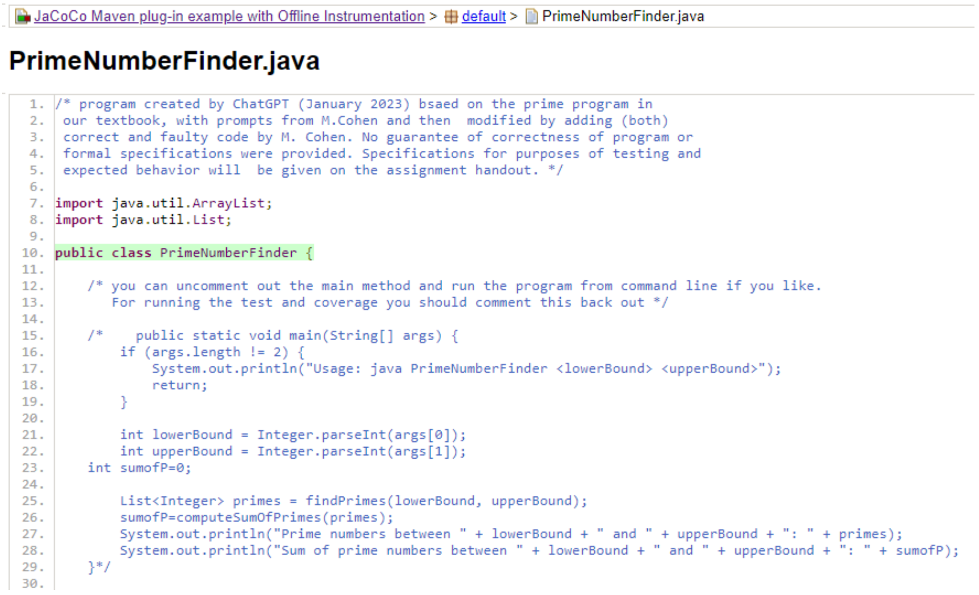
PrimeNumberFinder

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
isPrime(int)	🟥🟩	69%	🟥🟩	60%	6	11	3	10	0	1
computeSumOfPrimes(List)	🟥🟩	80%	🟥🟩	75%	1	3	1	7	0	1
findPrimes(int, int)	🟩	100%	🟩	100%	0	3	0	5	0	1
PrimeNumberFinder()	🟩	100%		n/a	0	1	0	1	0	1
Total	22 of 108	79%	9 of 28	67%	7	18	4	23	0	4



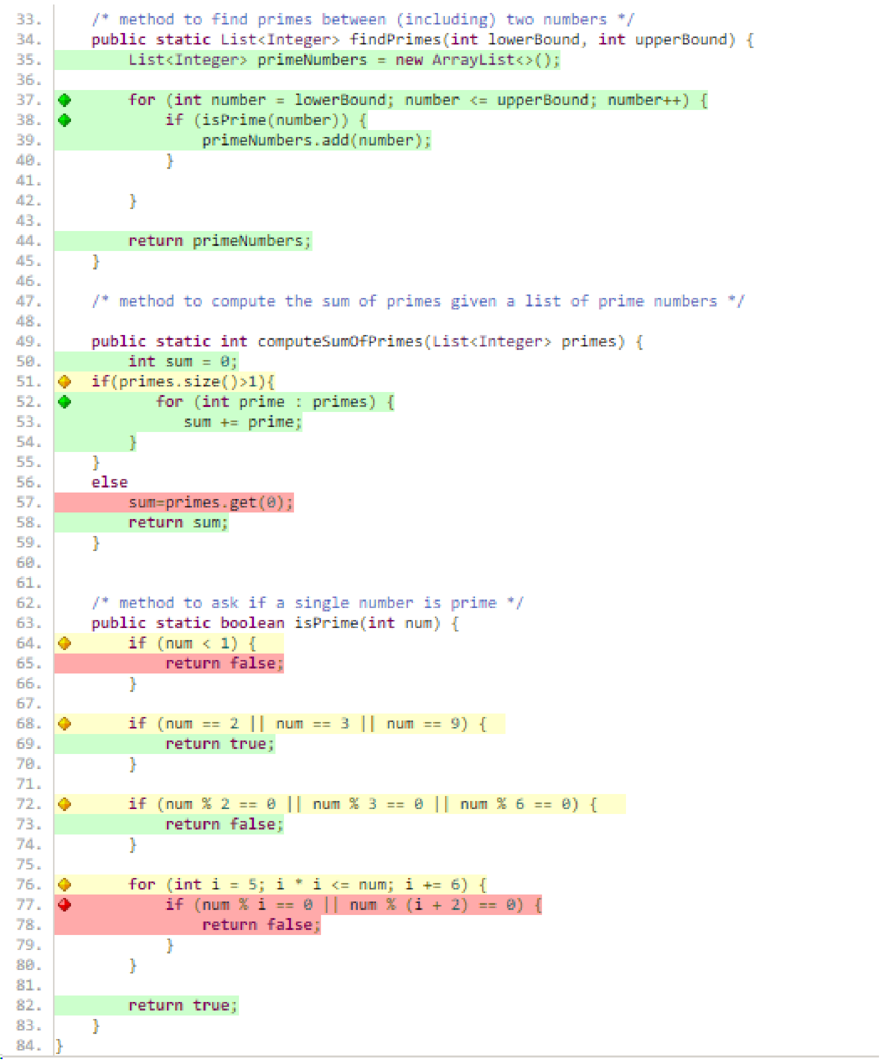
PrimeNumberFinderTest.txt - Notepad
File Edit Format View Help

```
-------------------------------------------------------------------------------
Test set: PrimeNumberFinderTest
-------------------------------------------------------------------------------
Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.746 sec
```


JaCoCo Maven plug-in example with Offline Instrumentation > default > PrimeNumberFinder.java

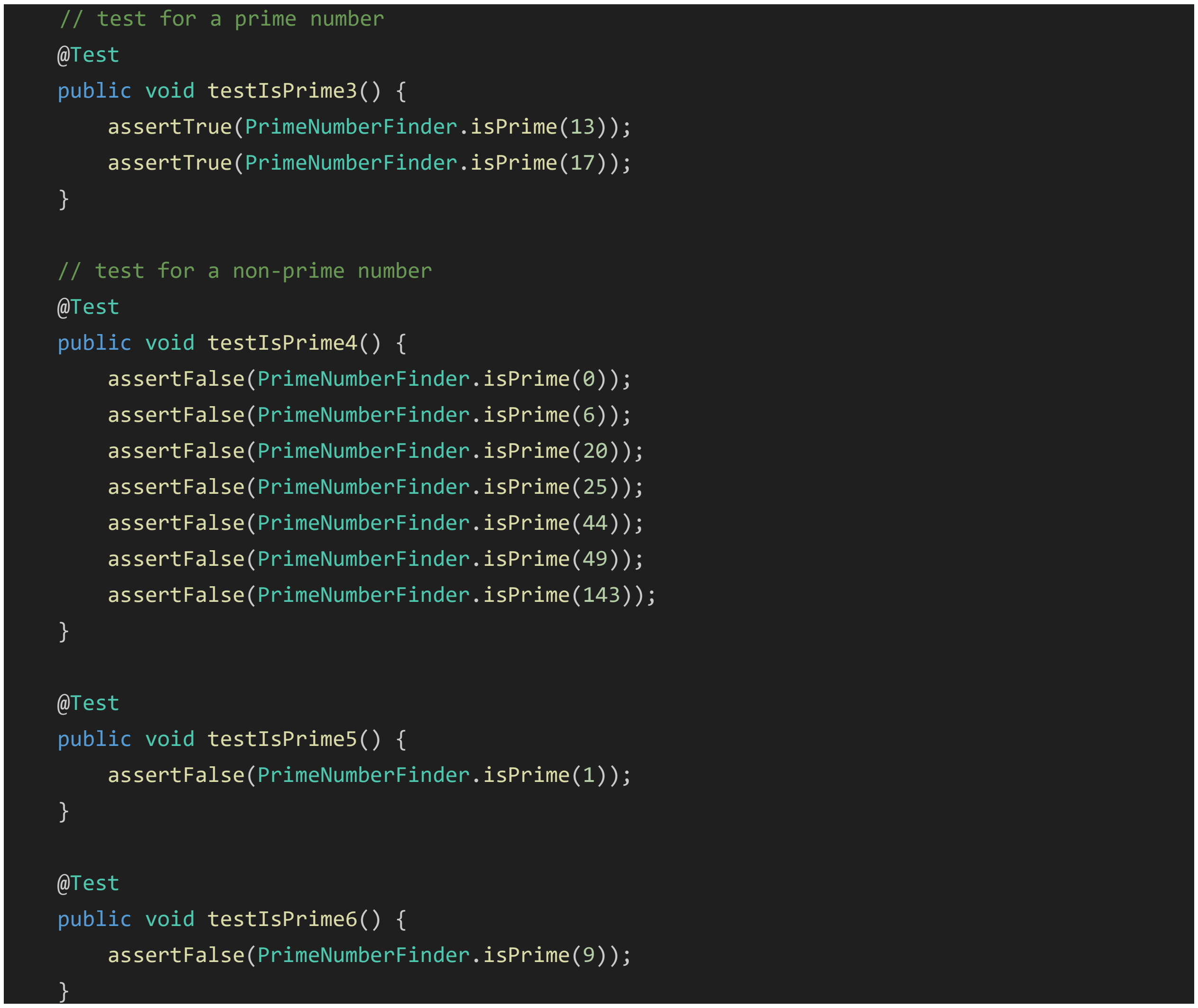
PrimeNumberFinder.java

```java
/* program created by ChatGPT (January 2023) bsaed on the prime program in
 our textbook, with prompts from M.Cohen and then  modified by adding (both)
 correct and faulty code by M. Cohen. No guarantee of correctness of program or
 formal specifications were provided. Specifications for purposes of testing and
 expected behavior will  be given on the assignment handout. */

import java.util.ArrayList;
import java.util.List;

public class PrimeNumberFinder {

    /* you can uncomment out the main method and run the program from command line if you like.
       For running the test and coverage you should comment this back out */

    /*    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java PrimeNumberFinder <lowerBound> <upperBound>");
            return;
        }

        int lowerBound = Integer.parseInt(args[0]);
        int upperBound = Integer.parseInt(args[1]);
    int sumofP=0;

        List<Integer> primes = findPrimes(lowerBound, upperBound);
        sumofP=computeSumOfPrimes(primes);
        System.out.println("Prime numbers between " + lowerBound + " and " + upperBound + ": " + primes);
        System.out.println("Sum of prime numbers between " + lowerBound + " and " + upperBound + ": " + sumofP);
    }*/

```

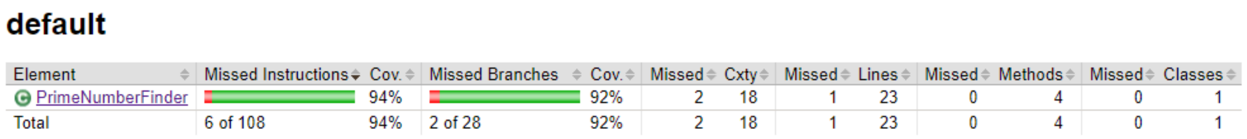


```java
    /* method to find primes between (including) two numbers */
    public static List<Integer> findPrimes(int lowerBound, int upperBound) {
        List<Integer> primeNumbers = new ArrayList<>();

        for (int number = lowerBound; number <= upperBound; number++) {
            if (isPrime(number)) {
                primeNumbers.add(number);
            }

        }

        return primeNumbers;
    }

    /* method to compute the sum of primes given a list of prime numbers */

    public static int computeSumOfPrimes(List<Integer> primes) {
        int sum = 0;
    if(primes.size()>1){
            for (int prime : primes) {
                sum += prime;
            }
    }
    else
        sum=primes.get(0);
        return sum;
    }


    /* method to ask if a single number is prime */
    public static boolean isPrime(int num) {
        if (num < 1) {
            return false;
        }

        if (num == 2 || num == 3 || num == 9) {
            return true;
        }

        if (num % 2 == 0 || num % 3 == 0 || num % 6 == 0) {
            return false;
        }

        for (int i = 5; i * i <= num; i += 6) {
            if (num % i == 0 || num % (i + 2) == 0) {
                return false;
            }
        }

        return true;
    }
}
```

Problem 4
Below are the test cases I wrote. One of the challenges I faced was that the original implementation of *isPrime* doesn't handle numbers less than 1 correctly. The implementation of checking divisibility by 2, 3, and 6 for even numbers is redundant as well. The method is also classifying 9 as a prime number. I got 100% coverage with my tests.

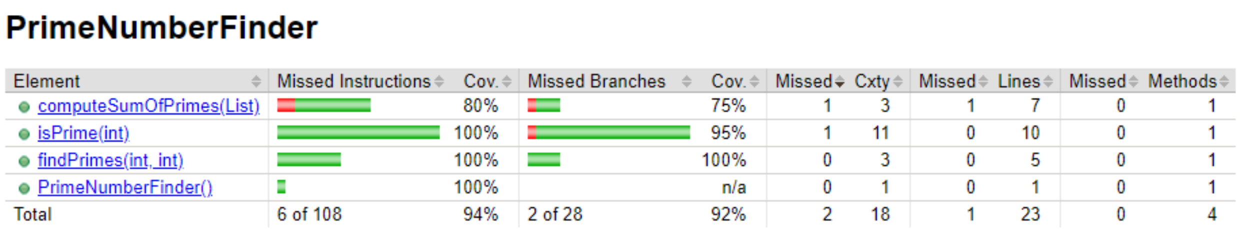


```
    // test for a prime number
    @Test
    public void testIsPrime3() {
        assertTrue(PrimeNumberFinder.isPrime(13));
        assertTrue(PrimeNumberFinder.isPrime(17));
    }

    // test for a non-prime number
    @Test
    public void testIsPrime4() {
        assertFalse(PrimeNumberFinder.isPrime(0));
        assertFalse(PrimeNumberFinder.isPrime(6));
        assertFalse(PrimeNumberFinder.isPrime(20));
        assertFalse(PrimeNumberFinder.isPrime(25));
        assertFalse(PrimeNumberFinder.isPrime(44));
        assertFalse(PrimeNumberFinder.isPrime(49));
        assertFalse(PrimeNumberFinder.isPrime(143));
    }

    @Test
    public void testIsPrime5() {
        assertFalse(PrimeNumberFinder.isPrime(1));
    }

    @Test
    public void testIsPrime6() {
        assertFalse(PrimeNumberFinder.isPrime(9));
    }
```



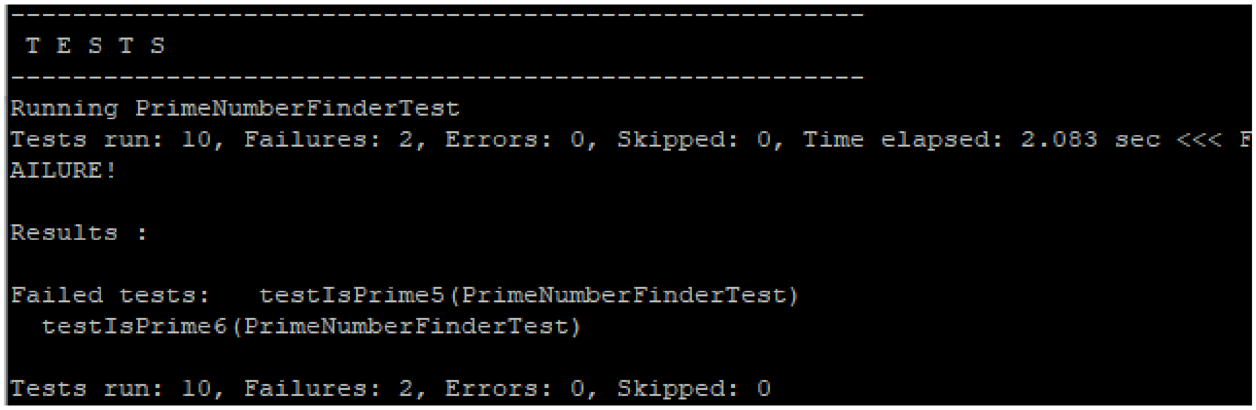
default

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
PrimeNumberFinder	🟥🟩	94%	🟥🟩	92%	2	18	1	23	0	4	0	1
Total	6 of 108	94%	2 of 28	92%	2	18	1	23	0	4	0	1

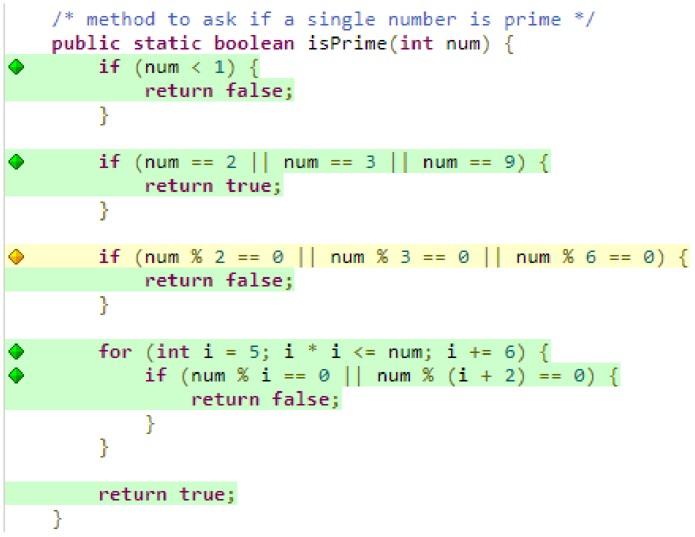


PrimeNumberFinder

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
computeSumOfPrimes(List)	🟥🟩	80%	🟥🟩	75%	1	3	1	7	0	1
isPrime(int)	🟩	100%	🟥🟩	95%	1	11	0	10	0	1
findPrimes(int, int)	🟩	100%	🟩	100%	0	3	0	5	0	1
PrimeNumberFinder()	🟩	100%		n/a	0	1	0	1	0	1
Total	6 of 108	94%	2 of 28	92%	2	18	1	23	0	4



```
------------------------------------------------------
 T E S T S
------------------------------------------------------
Running PrimeNumberFinderTest
Tests run: 10, Failures: 2, Errors: 0, Skipped: 0, Time elapsed: 2.083 sec <<< F
AILURE!

Results :

Failed tests:   testIsPrime5(PrimeNumberFinderTest)
  testIsPrime6(PrimeNumberFinderTest)

Tests run: 10, Failures: 2, Errors: 0, Skipped: 0
```

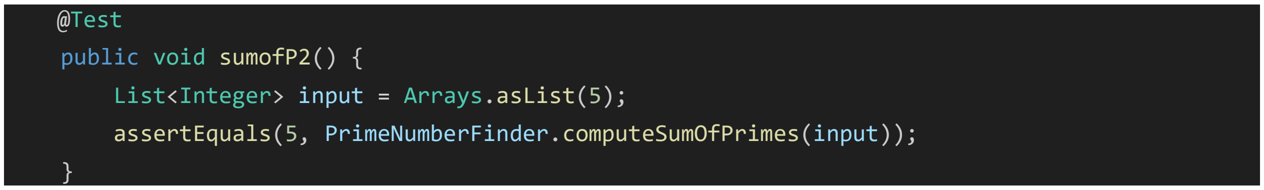


```java
    /* method to ask if a single number is prime */
    public static boolean isPrime(int num) {
        if (num < 1) {
            return false;
        }

        if (num == 2 || num == 3 || num == 9) {
            return true;
        }

        if (num % 2 == 0 || num % 3 == 0 || num % 6 == 0) {
            return false;
        }

        for (int i = 5; i * i <= num; i += 6) {
            if (num % i == 0 || num % (i + 2) == 0) {
                return false;
            }
        }

        return true;
    }
```

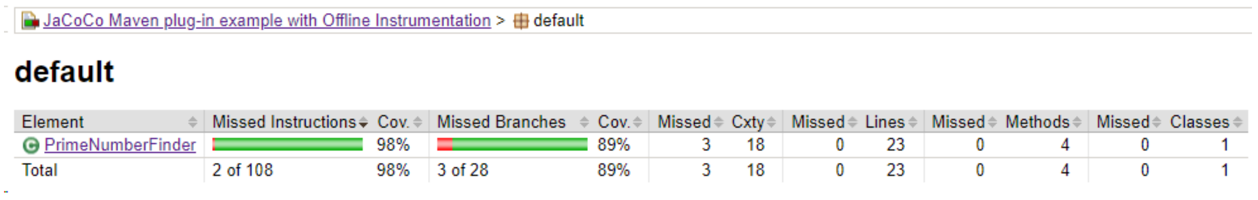
Problem 5

Below is a test case for the *computeSumOfPrimes*. Testing the *findPrimes* method might be easier, because it involves iterating over a range and checking for primality. Testing the *computeSumOfPrimes* method is straightforward, mainly focusing on the sum calculation. *isPrime* has some more complexity. More complex methods may require more diverse test cases to achieve thorough coverage.

The *computeSumOfPrimes* method currently works based on the sum of the provided list of numbers, regardless of whether they are prime or not. This design allows flexibility, as the method can be used for any list of numbers. But it might be beneficial to have additional specifications or checks if there are specific requirements for its use with prime numbers only. If the method is explicitly designed for prime numbers, additional checks could enhance clarity and prevent misuse.



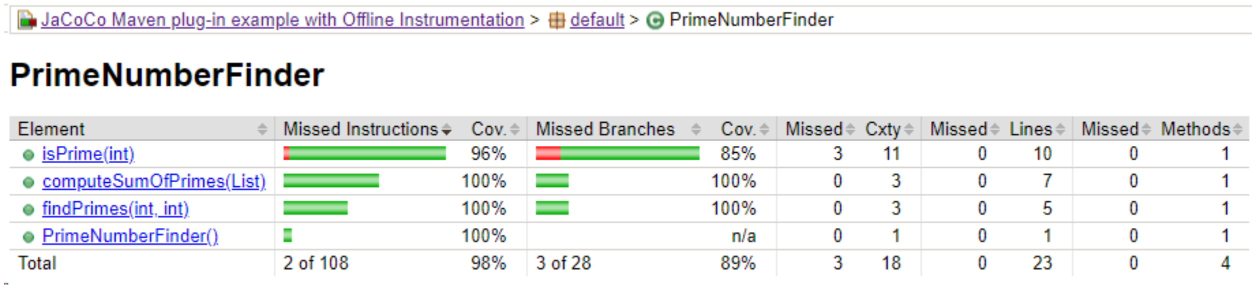
```java
   @Test
    public void sumofP2() {
        List<Integer> input = Arrays.asList(5);
        assertEquals(5, PrimeNumberFinder.computeSumOfPrimes(input));
    }
```



JaCoCo Maven plug-in example with Offline Instrumentation > default

default

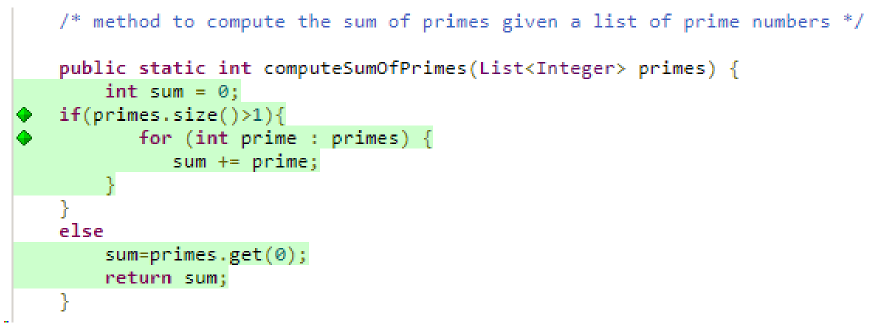
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
PrimeNumberFinder		98%		89%	3	18	0	23	0	4	0	1
Total	2 of 108	98%	3 of 28	89%	3	18	0	23	0	4	0	1



JaCoCo Maven plug-in example with Offline Instrumentation > default > PrimeNumberFinder

PrimeNumberFinder

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
isPrime(int)		96%		85%	3	11	0	10	0	1
computeSumOfPrimes(List)		100%		100%	0	3	0	7	0	1
findPrimes(int, int)		100%		100%	0	3	0	5	0	1
PrimeNumberFinder()		100%		n/a	0	1	0	1	0	1
Total	2 of 108	98%	3 of 28	89%	3	18	0	23	0	4



```
/* method to compute the sum of primes given a list of prime numbers */

public static int computeSumOfPrimes(List<Integer> primes) {
      int sum = 0;
   if(primes.size()>1){
         for (int prime : primes) {
            sum += prime;
      }
   }
   else
      sum=primes.get(0);
      return sum;
   }
```

Problem 6

Fault 1: The *isPrime* has a problem with input 9. The expected outcome is *false* but the actual outcome was *true* which is incorrect. The test case is below:

```
    @Test
    public void testIsPrime7() {
        assertFalse(PrimeNumberFinder.isPrime(9));
        // Oracle: Expecting false as 9 is not a prime number.
    }
```

Fault 2: The *isPrime* has a problem in the following: *if(num % 2 == 0 || num % 3 == 0 ||* ***num % 6 == 0****) { return false;}*. We are not able to ever hit *num % 6* because *% 2* and *% 3* encompass that. A test is not able to catch this because it is unreachable.

Fault 3: The *isPrime* has a problem with input 1. The expected outcome is *false* but the actual outcome was *true* which is incorrect. The test case is below:

```
    @Test
    public void testIsPrime8() {
        assertFalse(PrimeNumberFinder.isPrime(1));
    }
```

Exception: The *computeSumOfPrimes* doesn't handle the case where the input list is empty. It will throw an *IndexOutOfBoundsException* when trying to access the first element of the empty list. The test case is below:

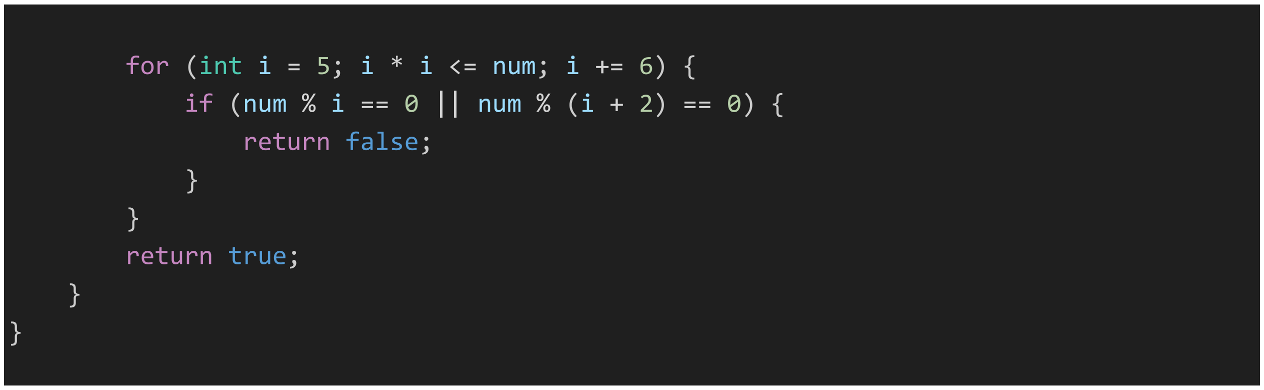
```
  @Test

   public void testComputeSumOfPrimesWithEmptyList() {
       List<Integer> input = Arrays.asList();
       int result = PrimeNumberFinder.computeSumOfPrimes(input);
       assertEquals(0, result);
   }
```

Problem 7

The three faults I found in *isPrime* were fixed. Here is the updated PrimeNumberFinder.java:

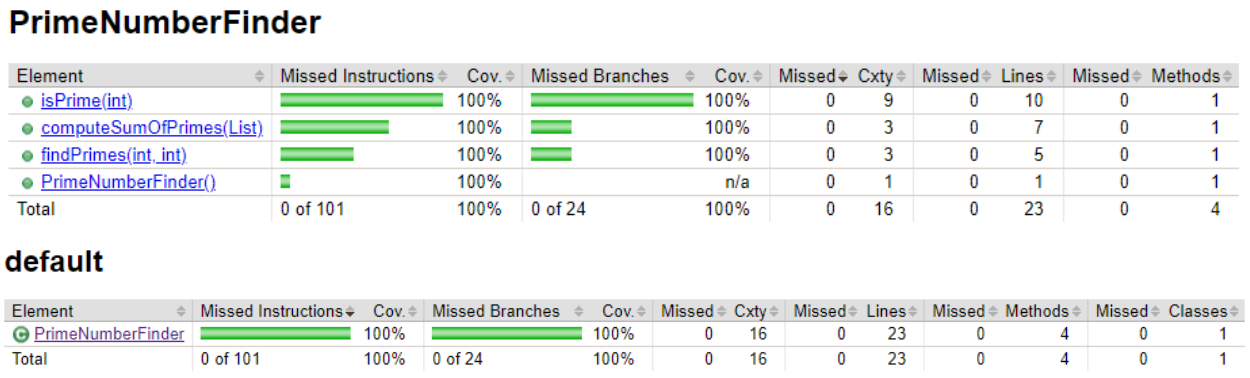
```
/* program created by ChatGPT (January 2023) bsaed on the prime program in
 our textbook, with prompts from M.Cohen and then  modified by adding (both)
 correct and faulty code by M. Cohen. No guarantee of correctness of program or
 formal specifications were provided. Specifications for purposes of testing and
 expected behavior will  be given on the assignment handout. */

import java.util.ArrayList;
import java.util.List;

public class PrimeNumberFinder {

    /*
     * you can uncomment out the main method and run the program from command line
     * if you like.
     * For running the test and coverage you should comment this back out
     */

    /*
     * public static void main(String[] args) {
     * if (args.length != 2) {
     * System.out.println("Usage: java PrimeNumberFinder <lowerBound> <upperBound>"
     * );
     * return;
     * }
     *
     * int lowerBound = Integer.parseInt(args[0]);
     * int upperBound = Integer.parseInt(args[1]);
     * int sumofP=0;
     *
     * List<Integer> primes = findPrimes(lowerBound, upperBound);
     * sumofP=computeSumOfPrimes(primes);
     * System.out.println("Prime numbers between " + lowerBound + " and " +
```

```java
 * upperBound + ": " + primes);
 * System.out.println("Sum of prime numbers between " + lowerBound + " and " +
 * upperBound + ": " + sumofP);
 * }
 */

/* method to find primes between (including) two numbers */
public static List<Integer> findPrimes(int lowerBound, int upperBound) {
    List<Integer> primeNumbers = new ArrayList<>();

    for (int number = lowerBound; number <= upperBound; number++) {
        if (isPrime(number)) {
            primeNumbers.add(number);
        }
    }
    return primeNumbers;
}

/* method to compute the sum of primes given a list of prime numbers */

public static int computeSumOfPrimes(List<Integer> primes) {
    int sum = 0;
    if (primes.size() > 1) {
    //if(!primes.isEmpty()){
        for (int prime : primes) {
            sum += prime;
        }
    } else
        sum = primes.get(0);
        return sum;
}

/* method to ask if a single number is prime */
public static boolean isPrime(int num) {
    if (num <= 1) {
        return false;
    }

    if (num == 2 || num == 3) {
        return true;
    }

    if (num % 2 == 0 || num % 3 == 0) {
        return false;
    }
```



```
        for (int i = 5; i * i <= num; i += 6) {
            if (num % i == 0 || num % (i + 2) == 0) {
                return false;
            }
        }
        return true;
    }
}
```

Here is the 100% code coverage report:



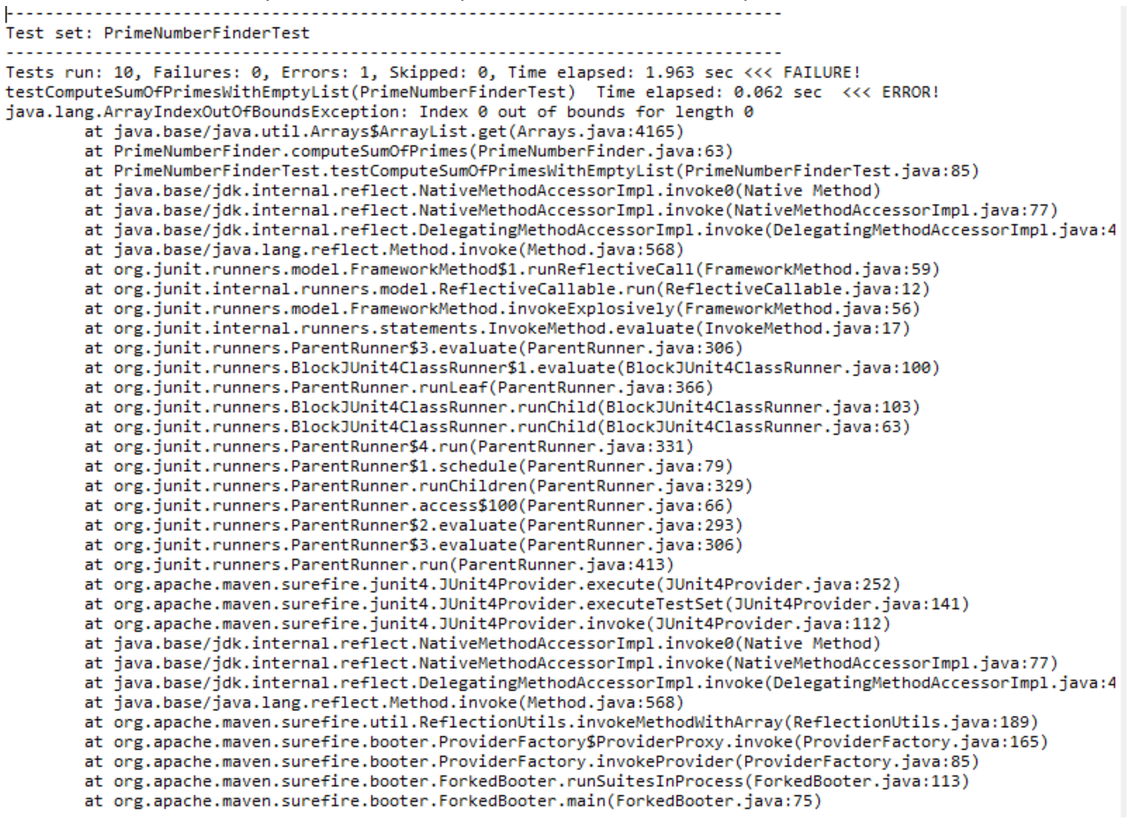
PrimeNumberFinder

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● isPrime(int)	🟩	100%	🟩	100%	0	9	0	10	0	1
● computeSumOfPrimes(List)	🟩	100%	🟩	100%	0	3	0	7	0	1
● findPrimes(int, int)	🟩	100%	🟩	100%	0	3	0	5	0	1
● PrimeNumberFinder()	🟩	100%		n/a	0	1	0	1	0	1
Total	0 of 101	100%	0 of 24	100%	0	16	0	23	0	4

default

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
PrimeNumberFinder	🟩	100%	🟩	100%	0	16	0	23	0	4	0	1
Total	0 of 101	100%	0 of 24	100%	0	16	0	23	0	4	0	1

Here is the surefire-report on the Exception that was found in problem 6:



```
-------------------------------------------------------------------------------
Test set: PrimeNumberFinderTest
-------------------------------------------------------------------------------
Tests run: 10, Failures: 0, Errors: 1, Skipped: 0, Time elapsed: 1.963 sec <<< FAILURE!
testComputeSumOfPrimesWithEmptyList(PrimeNumberFinderTest)  Time elapsed: 0.062 sec  <<< ERROR!
java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0
	at java.base/java.util.Arrays$ArrayList.get(Arrays.java:4165)
	at PrimeNumberFinder.computeSumOfPrimes(PrimeNumberFinder.java:63)
	at PrimeNumberFinderTest.testComputeSumOfPrimesWithEmptyList(PrimeNumberFinderTest.java:85)
	at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
	at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
	at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:4
	at java.base/java.lang.reflect.Method.invoke(Method.java:568)
	at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:59)
	at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
	at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:56)
	at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
	at org.junit.runners.ParentRunner$3.evaluate(ParentRunner.java:306)
	at org.junit.runners.BlockJUnit4ClassRunner$1.evaluate(BlockJUnit4ClassRunner.java:100)
	at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:366)
	at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:103)
	at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:63)
	at org.junit.runners.ParentRunner$4.run(ParentRunner.java:331)
	at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:79)
	at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:329)
	at org.junit.runners.ParentRunner.access$100(ParentRunner.java:66)
	at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:293)
	at org.junit.runners.ParentRunner$3.evaluate(ParentRunner.java:306)
	at org.junit.runners.ParentRunner.run(ParentRunner.java:413)
	at org.apache.maven.surefire.junit4.JUnit4Provider.execute(JUnit4Provider.java:252)
	at org.apache.maven.surefire.junit4.JUnit4Provider.executeTestSet(JUnit4Provider.java:141)
	at org.apache.maven.surefire.junit4.JUnit4Provider.invoke(JUnit4Provider.java:112)
	at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
	at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
	at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:4
	at java.base/java.lang.reflect.Method.invoke(Method.java:568)
	at org.apache.maven.surefire.util.ReflectionUtils.invokeMethodWithArray(ReflectionUtils.java:189)
	at org.apache.maven.surefire.booter.ProviderFactory$ProviderProxy.invoke(ProviderFactory.java:165)
	at org.apache.maven.surefire.booter.ProviderFactory.invokeProvider(ProviderFactory.java:85)
	at org.apache.maven.surefire.booter.ForkedBooter.runSuitesInProcess(ForkedBooter.java:113)
	at org.apache.maven.surefire.booter.ForkedBooter.main(ForkedBooter.java:75)
```

Here is the PrimeNumberFinderTest.java:

```java
import org.junit.Test;
import static org.junit.Assert.*;
import java.util.List;
import java.util.ArrayList;
import java.util.Arrays;

/*sample tests for homework. you will need to add to these */

public class PrimeNumberFinderTest {

    // Instantiate class - this will cover the constructor of the class
    @Test
    public void instantiateClass() {
        PrimeNumberFinder myPrime = new PrimeNumberFinder();
    }

    // Tests for the findPrimes method (you can add to these)
    @Test
    public void testFindPrimes1() {
        List<Integer> primes = PrimeNumberFinder.findPrimes(2, 8);
        List<Integer> expected = Arrays.asList(2, 3, 5, 7);
        assertArrayEquals(expected.toArray(), primes.toArray());
    }

    @Test
    public void testFindPrimes2() {
        List<Integer> primes = PrimeNumberFinder.findPrimes(10, 21);
        List<Integer> expected = Arrays.asList(11, 13, 17, 19);
        assertArrayEquals(expected.toArray(), primes.toArray());
    }

    // tests for the isPrime method

    // test for a prime number
    @Test
    public void testIsPrime1() {
        assertTrue(PrimeNumberFinder.isPrime(23));
    }

    // test for a non-prime number
    @Test
    public void testIsPrime2() {
        assertFalse(PrimeNumberFinder.isPrime(10));
    }
    // test for a prime number
    @Test
    public void testIsPrime3() {
        assertTrue(PrimeNumberFinder.isPrime(13));
        assertTrue(PrimeNumberFinder.isPrime(17));
    }

    // test for a non-prime number
    @Test
    public void testIsPrime4() {
        assertFalse(PrimeNumberFinder.isPrime(0));
        assertFalse(PrimeNumberFinder.isPrime(1));
        assertFalse(PrimeNumberFinder.isPrime(6));
        assertFalse(PrimeNumberFinder.isPrime(9));
        assertFalse(PrimeNumberFinder.isPrime(20));
        assertFalse(PrimeNumberFinder.isPrime(25));
        assertFalse(PrimeNumberFinder.isPrime(44));
        assertFalse(PrimeNumberFinder.isPrime(49));
        assertFalse(PrimeNumberFinder.isPrime(143));
    }

    // tests for the sumofP method - note the list provided is the list of primes
    // to be summed - not the lower and upper bound

    @Test
    public void sumofP1() {
        List<Integer> input = Arrays.asList(5, 7);
        assertEquals(12, PrimeNumberFinder.computeSumOfPrimes(input));
    }

    @Test
    public void sumofP2() {
        List<Integer> input = Arrays.asList(5);
        assertEquals(5, PrimeNumberFinder.computeSumOfPrimes(input));
    }

    @Test
    public void testComputeSumOfPrimesWithEmptyList() {
        List<Integer> input = Arrays.asList();
        int result = PrimeNumberFinder.computeSumOfPrimes(input);
        assertEquals(0, result);
    }
}
```