Ciências ULisboa | Informática

VVS 2017/2018
Exam 2 – July 18, 2019
Duration: 2 hours

Consider the method that finds and prints the first n prime integers:

```
1  private static void printPrimes(int n){
2      int curPrime;      // value currently considered for primeness.
3      int numPrimes;     // number of primes found so far.
4      boolean isPrime;   // is curPrime prime?
5      int[] primes = new int [MAXPRIMES]; // list of prime numbers.
6
7      // initialize 2 into the list of primes.
8      primes[0] = 2;
9      numPrimes = 1;
10     curPrime = 2;
11     while(numPrimes < n){
12         curPrime++;
13         isPrime = true;
14         for (int i = 0; i <= numPrimes-1; i++)
15             if(isDivisible(primes[i], curPrime)){
16                 // found a divisor, curPrime is not prime.
17                 isPrime = false;
18                 break;
19             }
20         if(isPrime){ // save it
21             primes[numPrimes] = curPrime;
22             numPrimes++;
23         }
24     } //while
25
26     // print all the primes out.
27     for (int i = 0; i <= numPrimes-1; i++){
28         System.out.println("Prime: " + primes[i]);
29     }
30 }
```

Group 1

1. Draw the control flow graph (CFG) of method `printPrimes()`. Assume the call of `isDivisible()` as a simple command.
2. List the requirements produced by edge-pair coverage of `printPrimes()`. Do not need to include node and edge requirements.
3. Write a set of JUnit tests that satisfy the previous requirements.
4. Identify 15 prime paths of the CFG that are cycles.

Group 2

1. Distinguish *defect*, *error* and *failure*.

2. Create the def-use table of method `printPrimes()`.
3. List the DU-pairs requirements of all variables except for variable i.

Group 3

Consider the following code:

```
1   /**  Check if a given string is a palindrome.
2    *   A palindrome is a string that is the same
3    *   when read right-to-left.
4    */
5   public static boolean isPalindrome(String s) {
6     if (s == null)
7       throw new NullPointerException();
8     int left = 0;
9     int right = s.length() - 1;
10    boolean result = true;
11    while (left < right && result == true) {
12      if (s.charAt(left) != s.charAt(right))
13        result = false;
14      left++;
15      right--;
16    }
17    return result;
18  }
```

1. Using *Combinatorial Coverage*, list its requirements.
2. Write JUnit tests to test these requirements. If there are infeasible requirements, justify why is it so.