Exercises (& Solutions) on Code Coverage

Exercise #1

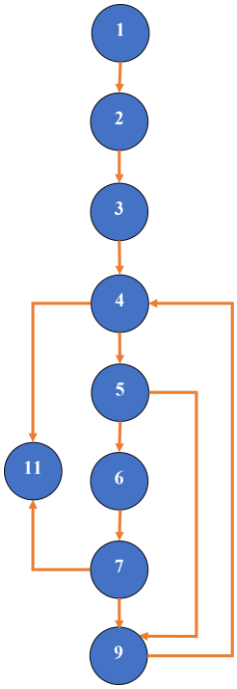
<table>
<tr><td colspan="2">Measure the Code Coverage</td></tr>
<tr><td>Given the following code

1. public int fun_LCM(int n1,int n2) {
2. int lcm=-1;
3. lcm = (n1 > n2) ? n1 : n2;
4. while(true) {
5. if(lcm % n1 == 0 && lcm % n2 == 0) {
6. System.out.printf("LCM(%d, %d)=%d", n1, n2, lcm);
7. return lcm;
8. }
9. ++lcm;
10. }
11.}

And the following two test cases:
@Test
void LCM_test_1() {
LCM lcm=new LCM();
int n1=5,n2=3;
int lcm_value = lcm.fun_LCM(n1, n2);
assertEquals(lcm_value, 15);
}
@Test
void LCM_test_2() {
LCM lcm=new LCM();
int n1=1,n2=1;
int lcm_value = lcm.fun_LCM(n1, n2);
assertEquals(lcm_value, 1);
}

@Test is a label that is used to indicate that the following method is a test case method

The assertEquals(expected, actual, delta) function asserts that the double variable of value expected and the actual one are equal within a positive delta. If they are not, an AssertionError is thrown. If expected and actual are null, they are considered equal.</td><td>(1) Build the Control flow graph

 
</td></tr>
<tr><td>(2) Identify the exercised statements and compute the percentage of code statement coverage for the following test cases:
1. LCM_test_1();
2. LCM_test_2();</td><td>Coverage
1. Percentage: 100%
a. Covered: 1,2,3,4,5,6,7,9,11
b. Out of 9
2. Percentage: 88.9%
a. Covered: 1,2,3,4,5,6,7,11
b. Out of 9</td></tr>
</table>

(3) Identify the exercised decisions and compute the percentage of code decision coverage for the following test cases: • LCM_test_1(); • LCM_test_2();	Coverage 1. Percentage: 80% a. Covered: 4 i. Line 3: T ii. Line 4: T iii. Line 5: T + F b. Out of 5 2. Percentage: 60% a. Covered: 3 i. Line 3: F ii. Line 4: T iii. Line 5: T b. Out of Lines 5
(4) Compute the total suite coverage for the decision coverage criteria by considering LCM_test_1 and LCM_test_2	We start from _test_1 and we add the not covered decisions that are instead covered from _test_2 Total coverage = (4 +1) out of 5 = 5 out of 5
(5) Compute the additional suite coverage for the decision coverage criteria by considering LCM_test_1 and LCM_test_2	I have to consider the test according the their additional coverage degree, then 1. Start from cov(_test_1)= 4 2. Then select the one with the highest difference wrt _test_1, or alternatively (if the difference is the same) the one with the highest coverage → select addCov(_test_2) =1 3. iterate step 2 cov(_test_1)+ addCov(_test_2)= (4+1) out of 5 = 5 out of 5
(6) Identify the exercised conditions and compute the percentage of code condition coverage for the following test cases: • LCM_test_1(); • LCM_test_2();	Coverage 1. Percentage: 85.7% a. Covered: 6 b. Out of 7 2. Percentage: 57.1% a. Covered: 4 b. Out of Lines 7
(7) Identify the exercised branches and compute the percentage of code branch coverage for the following test cases: • LCM_test_1(); • LCM_test_2();	Coverage 1. Percentage: 90% a. Covered: 9 b. Out of 10 2. Percentage: 70% a. Covered: 7 b. Out of Lines 10

Exercise #2

Measure the Code Coverage

Given the following code

```
public char Grade_computation(int[] marks) {
 1.float total=0, avg;
 2.char grade='E';
 3.for(int i=0; i< marks.length; i++) {
 4.  total = total + marks[i];
 }
 5.System.out.println("Total="+total);
 6.avg = total/6;
 7.System.out.println("AVG="+avg);
 8.if(avg>=30){
 9.  grade='A';
 }
10.else if(avg>=27 && avg<30){
11.  grade='B';
 }
12.else if(avg>=23 && avg<27){
13.  grade='C';
 }
 14.else if(avg>=19 && avg<23){
 15. grade='D';
 }else{
 16.  grade='E';
 }
 17. return grade;
}
```

... And the following JUnit test cases:

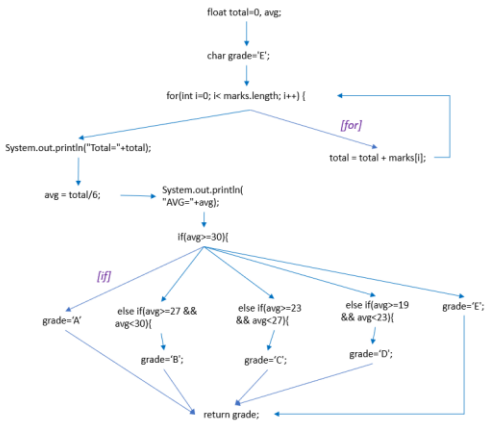
```
@Test
void test1() {
  int marks[] = new int[6];
   marks[0]=28;
   marks[1]=24;
   marks[2]=23;
   marks[3]=25;
   marks[4]=30;
   marks[5]=25;
   Grade G=new Grade();
   char grade=G.Grade_computation(marks);
   assertEquals(grade,'C');
}
@Test
void test2() {
  int marks[] = new int[6];
   marks[0]=31;
   marks[1]=30;
   marks[2]=29;
   marks[3]=30;
   marks[4]=30;
   marks[5]=30;
   Grade G=new Grade();
   char grade=G.Grade_computation(marks);
   assertEquals(grade,'A');
}
@Test
void test3() {
  int marks[] = new int[6];
   marks[0]=28;
   marks[1]=29;
   marks[2]=27;
   marks[3]=25;
   marks[4]=30;
   marks[5]=29;
   Grade G=new Grade();
   char grade=G.Grade_computation(marks);
   assertEquals(grade,'B');
}
@Test
void test4() {
  int marks[] = new int[1];
   marks[0]=21;
   Grade G=new Grade();
   char grade=G.Grade_computation(marks);
   assertEquals(grade,'E');
}
```

@Test is a label that is used to indicate that the following method is a test case method

The assertEquals(expected, actual, delta) function asserts that the double variable of value expected and the actual one are equal within a positive delta. If they are not, an AssertionError is thrown. If expected and actual are null, they are considered equal.

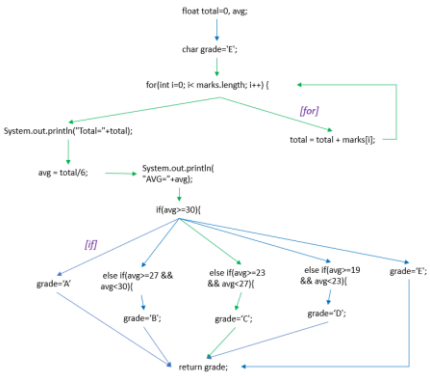
(1) Identify the number of covered statements for the three test cases	Coverage 1. Covered: 12 out of 17 a. Lines: 1,2,3,4,5,6,7,8,10,12,13,17 2. Covered: 10 out of 17 a. Lines: 1,2,3,4,5,6,7,8,9,17 3. Covered: 11 out of 17 a. Lines: 1,2,3,4,5,6,7,8,10,11,17 4. Covered: 13 out of 17 a. Lines: 1,2,3,4,5,6,7,8,10,12,14,16,17
(2) Compute the total suite coverage for the statement coverage criteria by considering the three test cases	Total suite coverage = cov(test1)+ cov(test2)+ cov(test3)+cov(test4)=(12+1+1+2) out of 17 = 16 out of 17
(3) Compute the additional suite coverage for the statement coverage criteria by considering the three test cases	I have to consider the test according the their additional coverage degree, then 4. Start from cov(test4)= 13 5. Then select the one with the highest difference wrt test4, or alternatively (if the difference is the same) the one with the highest coverage → select addCov(test1) = 1 6. iterate step 2 cov(test4)+ addCov(test1)+ cov(test3)+cov(test2)= (13+1+1+1) out of 17 = 16 out of 17
(4) Identify the number of covered decisions for the three test cases and details the decisions and their coverage (in terms of Line of the decision and True/False if it is covered or not covered)	Coverage 1. Covered: 5 out of 10 a. Line 3: T + F b. Line 8: F c. Line 10: F d. Line 12: T 2. Covered: 3 out of 10 a. Line 3: T + F b. Line 8: T 3. Covered: 4 out of 10 a. Line 3: T + F b. Line 8: F c. Line 10: T 4. Covered: 6 out of 10 a. Line 3: T + F b. Line 8: F c. Line 10: F d. Line 12: F e. Line 14: F
(5) Compute the total suite coverage for the decision coverage criteria by considering the three test cases	Total suite coverage = cov(test1)+ cov(test2)+ cov(test3)+cov(test4)=(5+1+1+2) out of 10 = 9 out of 10
(6) Compute the additional suite coverage for the decision coverage	I have to consider the test according the their additional coverage degree, then

criteria by considering the three test cases	1. Start from cov(test4)= 6 2. Then select the one with the highest difference wrt test4, or alternatively (if the difference is the same) the one with the highest coverage → select addCov(test1) = 2 3. iterate step 2 cov(test4)+ addCov(test1)+ cov(test3)+cov(test2)= (6+2+1+1) out of 10 = 10 out of 10
(7) Identify the number of covered conditions for the three test cases and details the conditions and their coverage (in terms of Line of the decision and True/False if it is covered or not covered)	Coverage 1. Covered: 6 out of 16 a. Line 3: T + F b. Line 8: F c. Line 10: F d. Line 12: T + T 2. Covered: 3 out of 16 a. Line 3: T + F b. Line 8: T 3. Covered: 5 out of 16 a. Line 3: T + F b. Line 8: F c. Line 10: T + T 4. Covered: 6 out of 16 a. Line 3: T + F b. Line 8: F c. Line 10: F d. Line 12: F e. Line 14: F
(8) Compute the total suite coverage for the condition coverage criteria by considering the three test cases	Total suite coverage = cov(test1)+ cov(test2)+ cov(test3)+cov(test4)=(6+1+2+2) out of 16 = 11 out of 16
(9) Compute the additional suite coverage for the condition coverage criteria by considering the three test cases	I have to consider the test according the their additional coverage degree, then 1. Start from cov(test4)= 6 2. Then select the one with the highest difference wrt test4, or alternatively (if the difference is the same) the one with the highest coverage a. select addCov(test1) = 2 3. iterate step 2 cov(test4)+ addCov(test1)+ cov(test3)+cov(test2)= (6+2+2+1) out of 16 = 11 out of 16
(10) Build the Control Flow Graph of the code and identify the number of branches statements for the three test cases	Control flow graph

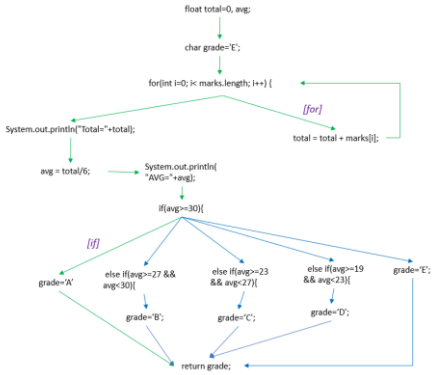


Coverage

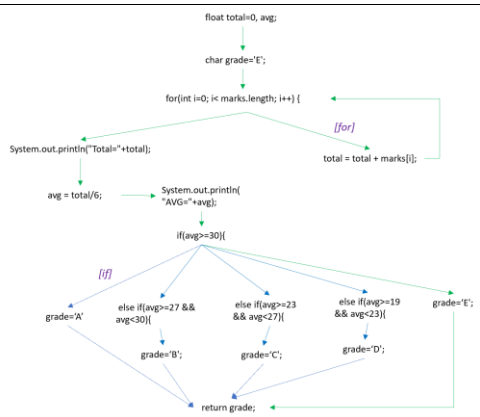
1. Covered: 11 out of 24



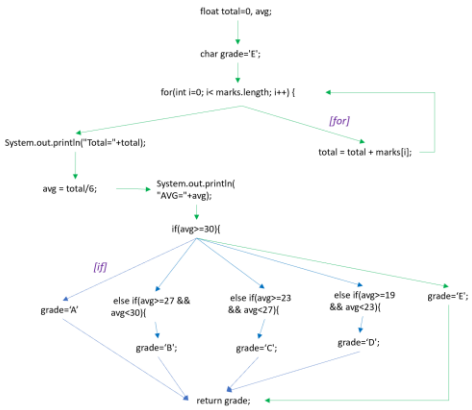
2. Covered: 10 out of 24



3. Covered: 11 out of 24



4. Covered: 10 out of 24



Exercise #3

<table>
<tr><td colspan="2">Measure the Code Coverage</td></tr>
<tr><td>Given the following code

<pre>
public int stringCompare(String str1, String str2)
{
  for (int i = 0; (i < str1.length() && i < str2.length()); i++) {
    if ((int)str1.charAt(i) == (int)str2.charAt(i)) {
      continue;
    }
    else {
      return (int)str1.charAt(i) - (int)str2.charAt(i);
    }
  }

  if (str1.length() < str2.length()) {
    return (str1.length()-str2.length());
  }
  else if (str1.length() > str2.length()) {
    return (str1.length()-str2.length());
  }

  else {
    return 0;
  }
}
</pre></td><td>… And the following JUnit test cases:
<pre>
@Test
void testA() {
  StringCompare stc=new StringCompare();
  String string1 = new String("Geeks");
  String string2 = new String("Practice");
   int ret=stc.stringCompare(string1, string2);
   assertEquals(ret,-9);
}

@Test
void testB() {
        StringCompare stc=new StringCompare();
        String string1 = new String("Geeks");
        String string3 = new String("Geeks");
   int ret=stc.stringCompare(string1, string3);
   assertEquals(ret,0);
}

@Test
void testC() {
        StringCompare stc=new StringCompare();
        String string1 = new String("Geeks");
        String string5 = new String("");
   int ret=stc.stringCompare(string5, string1);
   assertEquals(ret,-5);
}
</pre>
@Test is a label that is used to indicate that the following method is a test case method

The assertEquals(expected, actual, delta) function asserts that the double variable of value expected and the actual one are equal within a positive delta. If they are not, an AssertionError is thrown. If expected and actual are null, they are considered equal.</td></tr>
<tr><td>(11) Identify the number of covered statements for the three test cases</td><td>Coverage
1. Covered: 3 out of 9
2. Covered: 6 out of 9
3. Covered: 4 out of 9</td></tr>
<tr><td>(12) Identify the number of covered decisions for the three test cases</td><td>Coverage
1. Covered: 2 out of 8
2. Covered: 5 out of 8
3. Covered: 3 out of 8</td></tr>
<tr><td>(13) Identify the number of covered conditions for the three test cases</td><td>Coverage
1. Covered: 3 out of 10
2. Covered: 6 out of 10
3. Covered: 3 out of 10</td></tr>
<tr><td>(14) Identify the number of branches statements for the three test cases</td><td>Coverage
1. Covered: 2 out of 8
2. Covered: 5 out of 8
3. Covered: 2 out of 8</td></tr>
</table>

Exercise #4

<table>
<tr><td colspan="2">Measure the Code Coverage</td></tr>
<tr><td>Given the following code

<pre>
public boolean isEven(int number) {
  boolean ret_val=false;
  int i=0;
   if (number < 0) {
     ystem.out.println("Please enter a non-negative integer");
   } else {
      while(i <= number) {
         if (i == number) {
            ret_val=true;
         }
         i += 2;
      }
   }
   return ret_val;
}
</pre>
</td><td>… And the following JUnit test cases:
<pre>
 @Test
  void EvenOddChecker_test_1() {
            EvenOddChecker eoc=new EvenOddChecker();
            int n=1;
            boolean ret_value = eoc.isEven(n);
            assertEquals(ret_value, false);
            }

 @Test
  void EvenOddChecker_test_2() {
            EvenOddChecker eoc=new EvenOddChecker();
            int n=2;
            boolean ret_value = eoc.isEven(n);
            assertEquals(ret_value, true);
            }
   @Test
  void EvenOddChecker_3() {
            EvenOddChecker eoc=new EvenOddChecker();
            int n=3;
            boolean ret_value = eoc.isEven(n);
            assertEquals(ret_value, false);
            }
</pre>
@Test is a label that is used to indicate that the following method is a test case method

The assertEquals(expected, actual, delta) function asserts that the double variable of value expected and the actual one are equal within a positive delta. If they are not, an AssertionError is thrown. If expected and actual are null, they are considered equal.</td></tr>
<tr><td>(1) Identify the number of covered statements for the three test cases</td><td>Coverage
1. Covered: 7 out of 9
2. Covered: 8 out of 9
3. Covered: 7 out of 9</td></tr>
<tr><td>(2) Identify the number of covered decisions for the three test cases</td><td>Coverage
1. Covered: 4 out of 6
2. Covered: 5 out of 6
3. Covered: 4 out of 6</td></tr>
<tr><td>(3) Identify the number of covered conditions for the three test cases</td><td>Coverage
1. Covered: 4 out of 6
2. Covered: 5 out of 6
3. Covered: 4 out of 6</td></tr>
<tr><td>(4) Identify the number of branches statements for the three test cases</td><td>Coverage
1. Covered: 8 out of 12
2. Covered: 10 out of 12
3. Covered: 10 out of 12</td></tr>
</table>

Exercise #5

<table>
<tr><td colspan="2">Measure the Code Coverage</td></tr>
<tr><td>Given the following code

<pre>
public char isLeapYear(int year) {
 char ret_value='0';
 if (year<=0 || year>=3000) {
    return ret_value; }
 boolean x = false;
 if ((year % 4)==0)
   x = true;
 boolean y = false;
 if ((year % 100)!=0)
    y = true;
 boolean z = false;
 if ((year % 100 == 0) && (year % 400 == 0))
    z = true;
 System.out.println("x="+x+" y="+y+" z="+z);
 if (x && (y || z)){
    ret_value='y'; }
 else {
   ret_value='n'; }
 return ret_value;
}
</pre>
</td><td>… And the following JUnit test cases:
<pre>
@Test
void LeapYear_test_1() {
        LeapYear ly=new LeapYear();
        int y=2016;
        char ret_value = ly.isLeapYear(y);
        assertEquals(ret_value, 'y');
}
@Test
void LeapYear_test_6() {
        LeapYear ly=new LeapYear();
        int y=1700;
        char ret_value = ly.isLeapYear(y);
        assertEquals(ret_value, 'n');
}
@Test
void LeapYear_test_9() {
        LeapYear ly=new LeapYear();
        int y=000;
        char ret_value = ly.isLeapYear(y);
        assertEquals(ret_value, '0');
}
</pre>
@Test is a label that is used to indicate that the following method is a test case method

The assertEquals(expected, actual, delta) function asserts that the double variable of value expected and the actual one are equal within a positive delta. If they are not, an AssertionError is thrown. If expected and actual are null, they are considered equal.</td></tr>
<tr><td>(1) Identify the number of covered statements for the three test cases</td><td>Coverage
1. Covered: 14 out of 17
2. Covered: 13 out of 17
3. Covered: 3 out of 17</td></tr>
<tr><td>(2) Identify the number of covered decisions for the three test cases</td><td>Coverage
1. Covered: 5 out of 10
2. Covered: 5 out of 10
3. Covered: 1 out of 10</td></tr>
<tr><td>(3) Identify the number of covered conditions for the three test cases</td><td>Coverage
1. Covered: 8 out of 18
2. Covered: 9 out of 18
3. Covered: 1 out of 18</td></tr>
<tr><td>(4) Identify the number of branches statements for the three test cases</td><td>Coverage
1. Covered: 13 out of 20
2. Covered: 12 out of 20
3. Covered: 2 out of 20</td></tr>
</table>

Exercise #6

<table>
<tr><td colspan="2">Measure the Code Coverage</td></tr>
<tr><td>Given the following code

<pre>
public double fun_largest(double n1, double n2,
double n3) {
 double ret=-1.0;
 if(n1 >= n2) {
   if(n1 >= n3)
     ret=n1;
   else
     ret=n3;
 } else {
   if(n2 >= n3)
     ret=n2;
 else
   ret=n3;
 }
 return ret;
}
</pre>
</td><td>… And the following JUnit test cases:
<pre>
@Test
void Nested_test_1() {
    Nested n=new Nested();
    double n1 = -4.5, n2 = 3.9, n3 = 5.5;
    double ret = n.fun_largest(n1, n2, n3);
    assertEquals(5.5, ret, 0.001);
}
 T2.

@Test
void Nested_test_2() {
   Nested n=new Nested();
   double n1 = 10, n2 = 2, n3 = 1;
   double ret = n.fun_largest(n1, n2, n3);
   assertEquals(10, ret, 0.001);
}
</pre>
@Test is a label that is used to indicate that the following method is a test case method

The assertEquals(expected, actual, delta) function asserts that the double variable of value expected and the actual one are equal within a positive delta. If they are not, an AssertionError is thrown. If expected and actual are null, they are considered equal.</td></tr>
<tr><td>(1) Identify the number of covered statements for the three test cases</td><td>Coverage
1. Covered: 8 out of 13 (1,2,3,8,9,11,12,14)
2. Covered: 6 out of 13 (1,2,3,4,5,14)</td></tr>
<tr><td>(2) Compute the total suite coverage for the decision coverage criteria by considering _test_1 and _test_2</td><td>We start from _test_1 and we add the not covered decisions that are instead covered from _test_2
Total coverage = (8 +2) out of 13 = 10 out of 13</td></tr>
<tr><td>(3) Compute the additional suite coverage for the decision coverage criteria by considering _test_1 and _test_2</td><td>I have to consider the test according the their additional coverage degree, then
7. Start from cov(_test_1)= 8
8. Then select the one with the highest difference wrt _test_1, or alternatively (if the difference is the same) the one with the highest coverage
→ select addCov(_test_2) =2
9. iterate step 2

cov(_test_1)+ addCov(_test_2)=
(8+2) out of 13 = 10 out of 13</td></tr>
<tr><td>(4) Identify the number of covered decisions for the three test cases</td><td>Coverage
1. Covered: 2 out of 6
2. Covered: 2 out of 6</td></tr>
<tr><td>(5) Identify the number of covered conditions for the three test cases</td><td>Coverage
1. Covered: 2 out of 6
2. Covered: 2 out of 6</td></tr>
</table>

(6) Identify the number of branches statements for the three test cases	Coverage 1. Covered: 5 out of 12 2. Covered: 5 out of 12

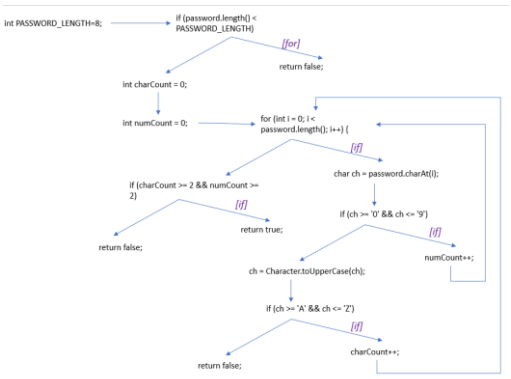
Exercise #7

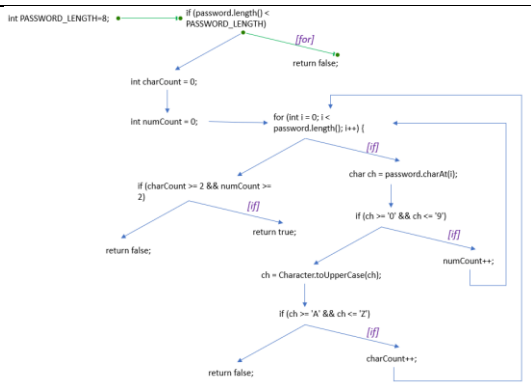
<table>
<tr><td colspan="2">Measure the Code Coverage</td></tr>
<tr><td>Given the following code

<pre>
public boolean is_Valid(String password) {
 int PASSWORD_LENGTH=8;
 if (password.length() < PASSWORD_LENGTH)
  return false;
 int charCount = 0;
 int numCount = 0;
 for (int i = 0; i < password.length(); i++) {
  char ch = password.charAt(i);
  if (ch >= '0' && ch <= '9')
   numCount++;
  else {
   ch = Character.toUpperCase(ch);
   if (ch >= 'A' && ch <= 'Z')
    charCount++;
   else
    return false;
   }
 }
 if (numCount>= 2 && charCount>= 2)
  return true;
 else
  return false;
}
</pre></td><td>… And the following JUnit test cases:
<pre>
@Test
void testA() {
        Psw2 psw2=new Psw2();
        String p="";
        boolean b=psw2.is_Valid(p);
        assertFalse("False.", b);
}
@Test
void testB() {
        Psw2 psw2=new Psw2();
        String p="testtest";
        boolean b=psw2.is_Valid(p);
        assertFalse("False.", b);
}
@Test
void testC() {
        Psw2 psw2=new Psw2();
        String p="testtest12";
        boolean b=psw2.is_Valid(p);
        assertTrue("True", b);
}
</pre>
@Test is a label that is used to indicate that the following method is a test case method

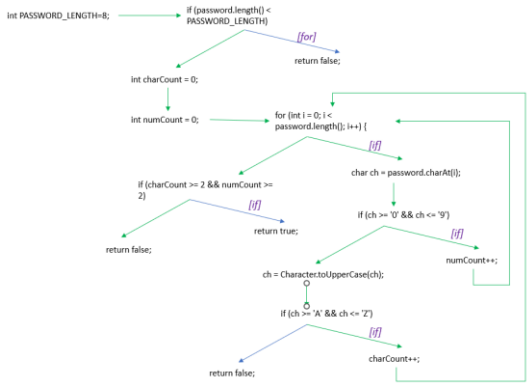
The assertTrue(“message”, boolean)| assertFalse(“message”, boolean) function asserts that the boolean is true|false. If it is not, an AssertionError is thrown and the “message” is printed.</td></tr>
<tr><td>(1) Identify the number of covered statements for the three test cases</td><td>Coverage
1. Covered: 3 out of 16
a. Lines: 1,2,3
2. Covered: 12 out of 16
a. Lines: 1,2,4,5,6,7,8,10,11,12,14,16
3. Covered: 12 out of 16
a. Lines: 1,2,4,5,6,7,8,10,11,12,14,15</td></tr>
<tr><td>(2) Compute the total suite coverage for the decision coverage criteria by considering the three tests</td><td>We start from testA and we add the not covered decisions that are instead covered from testB, and then testC
Total coverage = (3+10+1) out of 16 = 14 out of 16</td></tr>
<tr><td>(3) Compute the additional suite coverage for the decision coverage criteria by considering the three tests</td><td>I have to consider the test according the their additional coverage degree, then
1. Start from cov(testB)= 12
2. Then select the one with the highest difference wrt testB, or alternatively (if the difference is the same) the one with the highest coverage
a. select addCov(testC) =1
3. iterate step 2

cov(testB)+ addCov(testC) + addCov(testA)= (12+1+1) out of 16 = 14 out of 16</td></tr>
</table>

(4) Identify the number of covered decisions for the three test cases	Coverage 1. Covered: 1 out of 10 a. Line 2: T 2. Covered: 6 out of 10 a. Line 2: F b. Line 6: F + T c. Line 8: F d. Line 11: T e. Line 14: F 3. Covered: 7 out of 10 a. Line 2: F b. Line 6: F + T c. Line 8: T + F d. Line 11: F e. Line 14: T
(5) Identify the number of covered conditions for the three test cases	Coverage 1. Covered: 1 out of 16 a. Line 2: T 2. Covered: 7 out of 16 a. Line 2: F b. Line 6: F + T c. Line 8: T + F d. Line 11: T e. Line 14: F 3. Covered: 11 out of 16 a. Line 2: F b. Line 6: F + T c. Line 8: T + F + T d. Line 11: T + F +T e. Line 14: T + T
(4) Build the Control Flow Graph of the code and identify the number of branches statements for the three test cases	Control Flow Graph  Coverage 1. Covered: 2 out of 16



2. Covered: 14 out of 16



3. Covered: 14 out of 16