Verificação e Validação de Software (VVS)
Departamento de Informática, Faculdade de Ciências da Universidade de Lisboa
Exame de 1ª época — 4 de Junho de 2013
Duration: **2:30 + 30 min. tolerance**

Start each section (A to E) in a new sheet.

Consider the following Java method (noRep) that checks if an array of integers contains no repeated elements.

```
1   public static boolean noRep(int[] v) {
2     if (v == null) {
3       throw new NullPointerException();
4     }
5     for (int i=0; i < v.length - 1; i++) {
6       for (int j=i+1; j < v.length; j++) {
7         if (v[i] == v[j]) {
8           return false;
9         }
10      }
11    }
12    return true;
13  }
```

A. Graph coverage [2.5 valores]

(1) Draw the control flow graph (CFG) of noRep.

(2) Write JUnit tests methods that satisfy the Edge Coverage criteria over the CFG of noRep. Identify the CFG path corresponding to each of the tests.

(3) Identify the test requirements for the Prime Path Coverage criteria over the CFG of noRep and which of those requirements are covered by the JUnit tests you defined in (2).

B. Mutation testing [2.5 valores]

Consider the following mutations over noRep:

m_1 (l. 5)	v.length − 1	⟶	v.length + 1
m_2 (l. 6)	j < v.length	⟶	j >= v.length
m_3 (l. 6)	j < v.length	⟶	j <= v.length
m_4 (l. 7)	v[i] == v[j]	⟶	v[i] != v[j]
m_5 (l. 7)	v[i] == v[j]	⟶	v[i] <= v[j]

(1) Characterize tests that kill the mutants defined by each mutation above, if possible. Explain your choice of tests for the killed mutants and why the remaining mutants can not be killed.

(2) For one of the killed mutants in (1), characterize an additional test that executes the associated mutation but does not infect the state of noRep. Explain why.

(3) For one of the killed mutants in (1), characterize an additional test that infects the state of noRep but does not cause a failure. Explain why.

A triangle is well defined by three side lengths if and only if the sum of each pair of side lengths is greater than the remaining side length. A triangle is considered equilateral if all sides have equal length, isosceles if only two of the sides have equal length, and scalene when all side lengths differ. The following method triType implements this reasoning.

```
1  public static final int INVALID = 0;
2  public static final int SCALENE = 1;
3  public static final int ISOSCELES = 2;
4  public static final int EQUILATERAL = 3;
5
6  public static int triType(int a, int b, int c) {
7    int type ;
8    if (a <= 0 || b <= 0 || c <= 0)
9      type = INVALID;
10   else if (a + b <= c || a + c <= b || b + c <= a)
11     type = INVALID;
12   else if (a == b && a == c)
13     type = EQUILATERAL;
14   else if (a == c || a == b || b == c)
15     type = ISOSCELES;
16   else
17     type = SCALENE;
18   return type;
19 }
```

C. Input space partitioning [2.5 valores]

(1) Define 2 characteristics for partitioning the input space of triType, with at least 3 blocks per characteristic.

(2) Apply the Base Choice Coverage criteria over the defined characteristics to derive a set of test requirements.

(3) Characterize tests that satisfy the previous test requirements. If there are any infeasible requirements, identify them and explain the reason for infeasibility.

D. Logic coverage [2.5 valores]

(1) Identify the predicates and the clauses of each predicate in triType.

(2) Consider the Correlated Active Clause Coverage criteria applied to triType. Identify the predicate determination conditions by clause and the set of test requirements.

(3) Characterize tests that satisfy the previous test requirements. If there are any infeasible requirements, identify them and explain the reason for infeasibility.

E. Complementary aspects [2.5 valores]

Provide brief and clear answers to the following questions.

(1) Indicate two desirable properties for program mutation operators.

(2) What are “mock objects” and what is their purpose?

(3) What is test controllability? Refer to controllability problems in the context of multi-threaded program testing.