Verificação e Validação de Software (VVS)
Departamento de Informática, Faculdade de Ciências da Universidade de Lisboa
Exame de época especial — 18 de Julho de 2013
Duração: **2:30 + 30 min. tolerância**

Comece cada parte (A a E) numa página separada. Para cada conjunto de testes pedido no exame, não se esqueça de identificar para cada teste os correspondentes requisitos cobertos. Não se esqueça também de assinalar requisitos de teste inviáveis que encontre.

Considere o seguinte método Java (amplitude), que calcula a amplitude de valores num vector de inteiros, isto é, a diferença entre os valores máximo e mínimo no vector.

```
1  public static int amplitude(int[] v) {
2    if (v == null)
3      throw new NullPointerException();
4    if (v.length == 0)
5      return 0;
6    int min = v[0];
7    int max = v[0];
8    for (int i = 1; i < v.length; i++) {
9      if (v[i] > max)
10       max = v[i];
11     else if (v[i] < min)
12       min = v[i];
13   }
14   return max - min;
15 }
```

A. Cobertura por grafos [3.0 valores]

(1) Desenhe o gráfico de controlo de fluxo (CFG – "control flow graph") de amplitude.

(2) Identifique todas as definições, usos e caminhos-du no CFG.

(3) Identifique um conjunto de caminhos-du que satisfaça a cobertura todas-as-definições (ADC – "All-definitions coverage"), mas não cobertura todos-os-usos (AUC – "All-uses coverage"). Justifique a sua escolha.

(4) Caracterize testes que cubram os caminhos-du da alínea anterior.

B. Testes de mutação [3.0 valores]

Considere as seguintes mutações sobre amplitude:

m_1 (l. 4)	v.length == 0	$\longrightarrow$	v.length != 0
m_2 (l. 8)	i < v.length	$\longrightarrow$	i <= v.length
m_3 (l. 9)	v[i] > max	$\longrightarrow$	v[i] >= max
m_4 (l. 11)	v[i] < min	$\longrightarrow$	v[i] >= min
m_5 (l. 14)	max − min	$\longrightarrow$	max + min

(1) Caracterize testes que matam os mutantes associados a cada mutação, quando tal for possível. Justifique as suas escolhas de testes para os mutantes mortos e porque é que os restantes mutantes não podem ser mortos.

(2) Para um dos mutantes mortos em **(1)**, caracterize um teste adicional que executa a mutação associada mas não leva a uma infecção do estado. Justifique a escolha.

(3) Para um dos mutantes mortos em **(1)**, caracterize um teste adicional que leva a infecção do estado mas não a uma falha. Justifique a escolha.

(código repetido de amplitude para evitar virar a folha)

```
1  public static int amplitude(int[] v) {
2    if (v == null)
3      throw new NullPointerException();
4    if (v.length == 0)
5      return 0;
6    int min = v[0];
7    int max = v[0];
8    for (int i = 1; i < v.length; i++) {
9      if (v[i] > max)
10       max = v[i];
11     else if (v[i] < min)
12       min = v[i];
13   }
14   return max - min;
15 }
```

C. Partição do espaço de entrada [2.5 valores]

(1) Proponha **duas** características para partição do espaço de entrada do método amplitude, contendo cada característica pelo menos **três** blocos.

(2) Para as características dadas, defina um conjunto de requisitos de teste segundo o critério BCC ("Base Choice Coverage").

(3) Caracterize testes que satisfaçam os requisitos definidos na alínea anterior.

D. Cobertura lógica [2.0 valores]

(1) Identifique os predicados lógicos no código de amplitude.

(2) Identifique os requisitos de teste usando cobertura de predicados (PC – "Predicate Coverage") sobre amplitude.

(3) Escreva métodos de teste JUnit que satisfaçam os requisitos da alínea anterior.

E. Aspectos complementares [2.0 valores]

(1) O que entende por injecção de dependências ("dependency injection")? Explique a sua utilidade e relação com o uso de "mock objects".

(2) O que entende por observabilidade em testes de software? Refira problemas inerentes a observabilidade no contexto de programas de tempo real.