Verificação e Validação de Software (VVS), 2014/15
Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Exame de 2ª época — 1 de Julho de 2015
Duração: **2:30** — Cotação: **14 valores**

Comece cada parte (A a D) numa página separada da folha de exame. Para cada caso de teste que caracterizar, identifique os requisitos cobertos pelo teste. Sempre que detectar um requisito de teste inviável, assinale-o e explique a razão para a inviabilidade.

O método Java isPrime (abaixo) verifica se um número dado é primo. Um número primo n é um inteiro positivo maior que 1 que não tem divisores positivos além de 1 e dele próprio, por exemplo 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, ...

```
1   public static boolean isPrime(int n) {
2     if (n <= 1)
3       throw new IllegalArgumentException();
4     boolean isPrime = true;
5     if (n > 3) {
6       if (n % 2 == 0) {
7         isPrime = false;
8       } else {
9         int d = 3;
10        while (d * d <= n && isPrime == true) {
11          if (n % d == 0)
12            isPrime = false;
13          else
14            d = d + 2;
15        }
16      }
17    }
18    return isPrime;
19  }
```

A. Cobertura por grafos / Testes de mutação [6 valores]

(1) Desenhe o grafo de controlo de fluxo de isPrime.

(2) Identifique os requisitos de teste para uma cobertura por caminhos primos (PPC: "Prime Path Coverage")

(3) Caracterize testes para uma cobertura PPC.

(4) Considere os mutantes definidos a seguir sobre o código de isPrime:

$M1$ (l. 2)	n <= 1	$\longrightarrow$	n < 1
$M2$ (l. 5)	n > 3	$\longrightarrow$	n >= 3
$M3$ (l. 10)	isPrime == **true**	$\longrightarrow$	isPrime == **false**
$M4$ (l. 12)	isPrime = **false**	$\longrightarrow$	isPrime = **true**
$M5$ (l. 14)	d + 2	$\longrightarrow$	d + 1

Para **cada um** dos mutantes caracterize se possível: **(a)** um teste que mata o mutante; **(b)** um teste que leva um estado de erro mas não a uma falha; **(c)** um teste que atinge a mutação mas não leva a um estado de erro. Justifique as suas respostas, explicando a execução do teste ou porque não é possível encontrar um teste nas condições estipuladas.

(5) Os testes que definiu em **(4.a)** satisfazem uma cobertura de nós (NC: "Node Coverage") sobre o grafo de controlo de fluxo? Justifique se é ou não o caso. Se não for, proponha mutantes adicionais e testes que os matem por forma a que o conjunto de testes, os de **(4.a)** mais os adicionais, satisfaçam NC.

B. Testes baseados em lógica [3 valores]

O código do método daysInMonth (abaixo) devolve o número de dias em um mês (parâmetro m) de determinado ano (parâmetro y).

```
1   public static final int JANUARY = 1;
2   public static final int FEBRUARY = 2;
3   ...
4   public static final int JULY = 7;
5   public static final int AUGUST = 8;
6   ...
7   public static final int DECEMBER = 12;
8
9   public static int daysInMonth(int m, int y) {
10    int d;
11    if (m == FEBRUARY) {
12      d = y % 4 == 0 && (y % 100 != 0 || y % 400 == 0) ? 29 : 28;
13    } else if (m >= JANUARY && m <= JULY) {
14      d = m % 2 == 1 ? 31 : 30;
15    } else if (m >= AUGUST && m <= DECEMBER){
16      d = m % 2 == 0 ? 31 : 30;
17    } else {
18      throw new IllegalArgumentException("Invalid month: " + m);
19    }
20    return d;
21  }
```

(1) Identifique os predicados e cláusulas em daysInMonth e condições de alcance para cada um dos predicados. **Nota**: para expressões do tipo e1 ? e2 : e3 considere que e1 define um predicado.

(2) Identifique os requisitos de teste para uma cobertura por cláusulas activas geral (GACC: "General Active Clause Coverage").

(3) Caracterize testes para os requisitos GACC.

(4) Os testes que propôs satisfazem uma cobertura por cláusulas activas correlacionadas (CACC: "Correlated Active Clause Coverage") ? Justifique.

C. Metodologias de teste [2 valores]

(1) Que problemas devemos evitar ao programar testes para aplicações que usam bases de dados? Refira-se a pelo menos dois padrões/técnicas de programação de testes que sejam relevantes para resolver esses problemas.

(2) O que são testes de sistema, integração e de módulo? Para uma estratégia disciplinada de testes, será aconselhável uma escolha entre realizar alguns destes tipo de testes e não outros? Refira-se a pelo menos duas metodologias de programação de testes quando um sistema de software com dependências entre os seus componentes.

D. Partição do espaço de entrada [3 valores]

Considere o seguinte esqueleto para a API de uma fila ("queue") com a tradicional disciplina FIFO ("first-in, first-out") e uma capacidade máxima especificada em tempo de construção. O estado de uma fila pode ser observado usando os métodos capacity(), size(), front(), back() e toArray(). O estado de uma fila pode ser alterado usando os métodos add() e remove().

```
public class MyQueue {
  /** Construtor.
   *  A fila é inicialisada vazia e com a capacidade especificada.
   * @param capacity Capacidade > 0 */
  public MyQueue(int capacity) { ... }

  /** @return Capacidade da fila. */
  public int capacity() { ... }

  /** @return Número de elementos na fila. */
  public int size() { ... }

  /** @return Devolve elemento na frente da fila. */
  public Object front() { ... }

  /** @return Devolve elemento na traseira da fila. */
  public Object back() { ... }

  /** Devolve vector com elementos armazenados na fila. */
  public Object[] toArray() { ... }

  /** Adiciona elemento à traseira da fila se esta não está cheia.
   * @param elem Elemento a adicionar.
   * @throws IllegalStateException Se a fila está cheia. */
  public void add(Object elem) { ... }

  /** Remove elemento da frente da fila.
   * @return Elemento removido se a fila não está vazia.
   * @throws IllegalStateException Se a fila está vazia. */
  public Object remove() { ... }
}
```

(1) Proponha **duas** características com pelo menos **três** blocos cada para modelar o estado da fila. Tenha em conta a capacidade da fila e a sua utilização (número de elementos face à capacidade de fila).

(2) Tendo em conta as características que definiu, identifique requisitos de teste aplicando o critério de uma escolha base (BCC: "Base Choice Coverage").

(3) Caracterize testes sobre o método remove() para os requisitos BCC, na forma de sequências de chamadas a métodos de MyQueue.

(4) Escreva **dois** dos testes anteriores na forma de métodos Java/JUnit. **Nota**: relembra-se que o método assertArrayEquals da API do JUnit permite validar se dois vectores ("arrays") têm conteúdo equivalente.

(5) O critério BCC subsume uma cobertura pelo critério de pares-de-escolhas (PWC: "Pair-Wise Coverage")? E uma cobertura de cada-escolha (ECC: "Each Choice Coverage")? Justifique as suas respostas, ilustrando como os requisitos BCC definidos para a questão (**2**) se relacionam com os requisitos de teste para PWC e ECC.