Teste Modelo de Programação Concorrente

Versão 1

Nome completo:

Turma:

Durante a prova, é obrigatório ligar a câmara do computador ou do smartphone para realizar esta prova. Quaisquer suspeitas de auxílios externos para a realização desta prova causam a sua anulação.

As respostas devem ser dadas numa folha em branco, indicando claramente para cada pergunta a sua resposta. Respostas não legíveis serão consideradas como zero. No final da prova, o formando tem 5 minutos para tirar uma fotografia à sua resolução e enviar por email (upskill.java@gmail.com e em CC: secretariado_upskills@iscte-iul.pt).

Poderão ser tiradas dúvidas durante os primeiros **15 minutos** do início da prova. A duração da prova é de **3h**.

Boa Sorte!

Grupo I

Pergunta 1 [cotação prevista: X valores]

Qual a diferença entre uma situação `Deadlock` e uma situação de `Livelock` ?

Pergunta 2 [cotação prevista: X valores]

Descreva o que faz cada um dos seguintes métodos: `wait()`, `notify()` e `notifyAll()`. Como podem ser usados para coordenar processos ligeiros? Dê um exemplo.

Pergunta 3 [cotação prevista: X valores]

Explique o que é a sincronização de `threads`. Explique onde e porquê a sincronização é necessária, com recurso a um exemplo.

Pergunta 4 [cotação prevista: X valores]

Indique o que será escrito na consola decorrente da execução do seguinte código:

```
public class Barreira {
    private static final int MAX = 3;
    public int countIn = 0;
    public int countOut = 0;

    public synchronized void u1() {
        try {
            countIn++;
            while (countIn < MAX) {
                wait();
            }
            notifyAll();
        } catch (InterruptedException i) {
        }
        countOut++;
        if (countOut == MAX) {
            countIn = 0;
            countOut = 0;
        }
    }


public class SleepMessages extends Thread {
    private Barreira u;
    private long startTime;

    public SleepMessages(String name, Barreira u, long startTime) {
        setName(name);
        this.startTime = startTime;
        this.u = u;
    }

    public void run() {
        String importantInfo[] = {"Cold", "Hot"};
        for (int i = 0; i < importantInfo.length; i++) {
            u.u1();
            try {
                sleep(3000);
                System.out.println(getName() + "-" + importantInfo[i] + "-" +
                        (System.currentTimeMillis() - startTime) / 1000 + "s");
            } catch (InterruptedException e) {
            }
        }
    }

    public static void main(String args[]) throws InterruptedException {
        long sTime = System.currentTimeMillis();
        Barreira u = new Barreira();
        Thread t1 = new SleepMessages("t1", u, sTime);
        Thread t2 = new SleepMessages("t2", u, sTime);
        System.out.println("Here they go!...");
        t1.start();
        t2.start();
        sleep(1000);
        t1.interrupt();
        sleep(1000);
        u.u1();
        sleep(3000);
        u.u1();
        sleep(1000);
        t2.interrupt();
        t1.join();
        t2.join();
        System.out.println("Main - Main done - " +
                (System.currentTimeMillis() - sTime) / 1000 + "s");
    }
}
```

Grupo II

Pergunta 1 [cotação prevista: X valores]

Pretende-se desenvolver uma aplicação que calcula todos os números primos até um valor definido (`VALOR_MAX`). Para calcular a lista dos números primos a aplicação deve verificar todos os números de 2 a `VALOR_MAX`.

Por questões de eficiência, o cálculo dos números primos deve ser implementado através de thread pools, recorrendo ao uso da classe de fábrica `Executors`, disponível na API do JAVA. A pool deve ter 4 threads. Cada tarefa pode tratar até um máximo de 1/4 do `VALOR_MAX`, isto é se o `VALOR_MAX = 100` então cada tarefa poderá tratar 25 valores.

Os números que forem detectados como primos devem ser impressos na consola por ordem crescente quando todos as tarefas estiverem terminadas. Tente criar uma solução onde garante a máxima concorrência possível.

Ajuda: Um número é primo se não for divisível por nenhum número maior que 1 e inferior a ele. Para verificar se um número é divisível por outro pode usar o operador do resto da divisão inteira representado em Java por %. Por exemplo 9 % 3 = 0.

Pergunta 2 [cotação prevista: X valores]

Um lingote típico de ouro é construído com 12,5 kg de ouro. Para fazer um lingote, são precisos diversos pedaços de ouro extraídos da natureza.

Uma escavadora recolhe pequenos pedaços de ouro (cada pedaço pesa até 1 kg) e vai colocando o ouro recolhido numa balança. Quando a balança tem 12,5 kg de ouro ou mais, o ourives recolhe o ouro para fazer o lingote.

A escavadora e o ourives devem ser threads. A escavadora pode acrescentar ouro na balança apenas se houver menos do que 12,5 kg de ouro na balança. Por outro lado, o ourives pode apenas recolher o ouro quando existem 12,5 kg ou mais na balança. Depois de o ourives recolher o ouro da balança, leva 3 segundos a transformar o ouro num lingote.

Utilize um `double` para representar a quantidade atual de ouro na balança.

Implemente a classe `Balança` e os seus métodos para colocar e retirar o ouro. Tenha atenção à necessidade de coordenação entre a escavadora e o ourives.

Pergunta 3 [cotação prevista: X valores]

O DNA consiste numa sequência constituída por açúcar, grupos de fosfato e quatro bases A, T, C e G. Estas 4 bases constituem a informação genética do DNA. Realize um programa que conta a ocorrência de cada uma das letras A, T, C e G, respectivamente, em `Strings`. Considere que tem na

sua classe principal um método `getSequencedData()` que devolve uma `String` (assuma que este método já está implementado). Este método devolve novas strings sempre que for invocado, até um ponto em que não haverá mais strings e devolverá `null`.

Por questões de eficiência, as strings devem ser processadas em paralelo, mas não deve haver em nenhum momento mais de 10 strings a serem processadas. Use um semáforo (veja sumário da API abaixo) para garantir que apenas existem 10 threads. Uma vez que todas as strings tenham sido processadas, os número totais de A, T, C e Gs, respectivamente, devem ser mostradas na consola (utilizando o `System.out.println()`).

Recorde que o método `charAt(int pos)` retorna uma letra da string numa dada posição. Por exemplo, numa dada `String s`, `s.charAt(0)` retorna a primeira letra e `s.charAt(5)` retorna a letra na posição 5, ou seja a sexta letra.

A seguinte classe deverá ser a sua classe principal. Implemente todas as funcionalidades necessárias com excepção do método `getSequencedData()`.

Considere o seguinte exemplo da execução do programa para 3 cadeias de DNA:

"ATTTTAAAAGGGCGCGCGCCCCA"

"AAAC"

"CCCCCCCCCC"

Resultado:

A: 9 vezes

T: 4 vezes

C: 18 vezes

G: 6 vezes

```
public class DNACounter {

    private String getSequencedData() { // do not implement}

}
```

Anexos

Cabeçalhos e pedaços de código comuns:

```
public static void main (String[] args) {...}
try {...} catch(FileNotFoundException e) {...}
Collections.sort(...);
Integer.parseInt(...);
```

Semaphore

Constructor and Description
`Semaphore(int permits)` Creates a Semaphore with the given number of permits and nonfair fairness setting.
`Semaphore(int permits, boolean fair)` Creates a Semaphore with the given number of permits and the given fairness setting.

Modifier and Type	Method and Description
void	`acquire()` Acquires a permit from this semaphore, blocking until one is available, or the thread is `interrupted`.
void	`acquire(int permits)` Acquires the given number of permits from this semaphore, blocking until all are available, or the thread is `interrupted`.
int	`availablePermits()` Returns the current number of permits available in this semaphore.
void	`release()` Releases a permit, returning it to the semaphore.
void	`release(int permits)` Releases the given number of permits, returning them to the semaphore.

CyclicBarrier

Constructor and Description
CyclicBarrier(int parties) Creates a new CyclicBarrier that will trip when the given number of parties (threads) are waiting upon it, and does not perform a predefined action when the barrier is tripped.
CyclicBarrier(int parties, Runnable barrierAction) Creates a new CyclicBarrier that will trip when the given number of parties (threads) are waiting upon it, and which will execute the given barrier action when the barrier is tripped, performed by the last thread entering the barrier.

Modifier and Type	Method and Description
int	await(), throws InterruptedException, BrokenBarrierException Waits until all parties have invoked await on this barrier.
int	await(long timeout, TimeUnit unit), throws InterruptedException, BrokenBarrierException, TimeoutException Waits until all parties have invoked await on this barrier, or the specified waiting time elapses.
int	getNumberWaiting() Returns the number of parties currently waiting at the barrier.
int	getParties() Returns the number of parties required to trip this barrier.
boolean	isBroken()() Queries if this barrier is in a broken state.
void	reset()() Resets the barrier to its initial state.

Scanner

Constructor and Description
Scanner(File source) Creates a new Scanner that produces values scanned form the specified file.

Modifier and Type	Method and Description
void	close() Closes this scanner.
boolean	hasNext() Returns true if this scanner has another token in its input.
boolean	hasNext(String pattern) Returns true if the next token matches the pattern constructed from the specified string.
boolean	hasNextInt() Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the nextInt() method.
boolean	hasNextLine() Returns true if there is another line in the input of this scanner.
String	next() Finds and returns the next complete token from this scanner.
int	nextInt() Scans the next token of the input as an int.
String	nextLine() Advances this scanner past the current line and returns the input that was skipped.