# Parte II - Exame Sistemas Operativos 2019/2020

---

## Questão 7 (a) [1.0 valor]

### Pergunta:

Qual será o resultado esperado da execução do código?

### Resposta:

O resultado esperado é que todas as threads incrementem o valor inicial da variável `number` do objeto `Sinal`, de forma sequencial. Cada thread irá imprimir o valor atualizado após o incremento.  
  
Por exemplo, se o número inicial for 0, a saída pode ser:  
```  
Th0: 1  
Th1: 2  
Th2: 3  
Th3: 4  
Th4: 5  
Todas as threads terminaram  
```

## Questão 7 (b) [1.0 valor]

### Pergunta:

Existe a possibilidade de surgir outro resultado diferente?

### Resposta:

Sim, é possível. O código apresenta \*\*condições de corrida (race conditions)\*\*, pois várias threads podem tentar atualizar o valor da variável `number` simultaneamente, sem sincronização adequada. Isso pode resultar em valores incorretos ou duplicados, como:  
```  
Th0: 1  
Th1: 1  
Th2: 2  
Th3: 3  
Th4: 4  
Todas as threads terminaram  
```  
Esse comportamento acontece porque o método `setNumber` não é protegido por sincronização, permitindo que múltiplas threads acessem e modifiquem a variável compartilhada ao mesmo tempo.

## Questão 7 (c) [2.0 valores]

### Requisitos:

1. Eliminar possíveis condições de corrida (race conditions).  
2. Garantir que a linha 40 só é executada quando todas as threads já terminaram.

### Código Corrigido:

import java.util.concurrent.\*;  
  
class Tempo {  
 static void sleep(long millis) {  
 try { Thread.sleep(millis); } catch (InterruptedException ie) {}  
 }  
}  
  
public class EpocaNormal implements Runnable {  
 private Sinal s;  
 private int i;  
 private static Semaphore semaforo = new Semaphore(0);  
  
 public EpocaNormal(Sinal s, int i) {  
 this.s = s;  
 this.i = i;  
 }  
  
 public void run() {  
 int contador = 0;  
 while (contador++ < i) {  
 s.doWait();  
 synchronized (s) {  
 s.setNumber(s.getNumber() + 1);  
 }  
 }  
 Tempo.sleep(100);  
 System.out.println(Thread.currentThread().getName() + ": " + s.getNumber());  
 s.doRun();  
 semaforo.release();  
 }  
  
 public static void main(String[] args) {  
 final Sinal s = new Sinal();  
 int numThreads = 5;  
  
 for (int i = 0; i < numThreads; i++) {  
 new Thread(new EpocaNormal(s, i), "Th" + i).start();  
 }  
  
 Tempo.sleep(100);  
 s.doRun();  
  
 try {  
 semaforo.acquire(numThreads);  
 } catch (InterruptedException e) {}  
  
 System.out.println("Todas as threads terminaram");  
 }  
}  
  
class Sinal {  
 private int number = 0;  
  
 protected synchronized void doRun() {  
 this.notifyAll();  
 }  
  
 protected synchronized void doWait() {  
 try { this.wait(); } catch (InterruptedException ie) {}  
 }  
  
 protected synchronized void setNumber(int number) {  
 this.number = number;  
 }  
  
 protected synchronized int getNumber() {  
 return this.number;  
 }  
}

## Questão 8 [4.0 valores]

### Requisitos:

1. Implementar classes \*\*Printer\*\* e \*\*Desktop\*\* para gerenciar um buffer compartilhado (\*\*Job\*\*).  
2. Garantir sincronização adequada com semáforos.

### Código Resolvido:

import java.util.concurrent.\*;  
import java.util.\*;  
  
class Job {  
 private Queue<String> trabalhos = new LinkedList<>();  
 private int capacidade = 5;  
  
 private Semaphore podeEscrever = new Semaphore(5);  
 private Semaphore podeLer = new Semaphore(0);  
  
 public void escrever(String trabalho) throws InterruptedException {  
 podeEscrever.acquire();  
 synchronized (this) {  
 trabalhos.add(trabalho);  
 System.out.println("Desktop adicionou: " + trabalho);  
 }  
 podeLer.release();  
 }  
  
 public String ler() throws InterruptedException {  
 podeLer.acquire();  
 String trabalho;  
 synchronized (this) {  
 trabalho = trabalhos.poll();  
 System.out.println("Printer imprimiu: " + trabalho);  
 }  
 podeEscrever.release();  
 return trabalho;  
 }  
}  
  
class Printer extends Thread {  
 private Job job;  
  
 Printer(Job job) { this.job = job; }  
  
 public void run() {  
 try {  
 while (true) {  
 job.ler();  
 Thread.sleep(1000);  
 }  
 } catch (InterruptedException e) {}  
 }  
}  
  
class Desktop extends Thread {  
 private Job job;  
 private String trabalho;  
  
 Desktop(Job job, String trabalho) {  
 this.job = job;  
 this.trabalho = trabalho;  
 }  
  
 public void run() {  
 try {  
 job.escrever(trabalho);  
 } catch (InterruptedException e) {}  
 }  
}  
  
public class Impressao {  
 public static void main(String[] args) {  
 Job job = new Job();  
 Printer printer = new Printer(job);  
 printer.start();  
  
 for (int i = 0; i < 10; i++) {  
 new Desktop(job, "Trabalho" + i).start();  
 }  
 }  
}