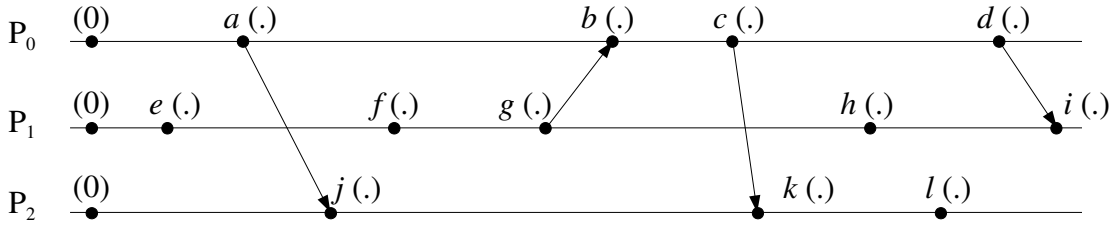
Parte A (12 valores)

1. O que se entende por camada de *middleware*? Faça um diagrama ilustrativo que sinalize a presença da camada para o caso de uma máquina paralela de acoplamento solto, formada por três nós de processamento, que executa uma aplicação distribuída. Explique porque é que a *máquina virtual de Java* (JVM) pode ser considerada um componente do *middleware*.
2. Considere a *troca de mensagens* como o paradigma de comunicação que foi escolhido para se estabelecer um modelo cliente-servidor na variante de *cópia de recursos* entre processos residentes nos diferentes nós de processamento de um sistema distribuído e admita que existem duas cópias. Apresente um diagrama esquemático que descreva a interacção funcional entre os diferentes componentes de um tal sistema no lado do servidor, identificando cada um dos componentes introduzido e elucidando o seu papel.
   Explique que vantagens apresenta esta variante e em que consiste o problema da *consistência* da informação.
3. A figura abaixo descreve a evolução *temporal* de três processos cujos relógios locais são relógios lógicos escalares sincronizados segundo o algoritmo de acerto de Lamport.



   a) Atribua aos diferentes acontecimentos, especificados por letras do alfabeto minúsculo (*a ... l*), que ocorrem nos três processos, o valor da marca temporal (*time stamp*) que lhes está associado.
   b) Indique para os seguintes pares de acontecimentos, *a-f*, *f-h*, *b-i*, *l-i*, *j-i* e *a-l*, se se trata de acontecimentos concorrentes (||) ou de acontecimentos ligados por um nexo de causalidade (->).
4. Considere uma situação no contexto da *comunicação entre pares* em que coexistem três processos, P1, P2 e P3, que pretendem eleger um deles como chefe. Descreva os passos principais de um algoritmo que conduz a essa eleição. Admita que não há perda de mensagens, nem que qualquer dos processos falha. Como alteraria o algoritmo se as suposições anteriores deixassem de ser válidas.

Parte B (8 valores)

```
public class Semaphore
{
   private int val = 0;
   private int numbBlockThreads = 0;
   public synchronized void down ()
   {
      if (val == 0)
         { numbBlockThreads += 1;
           try
           { wait ();
           }
           catch (InterruptedException e) {}
         }
         else val -= 1;
   }
   public synchronized void up ()
   {
      if (numbBlockThreads != 0)
         { numbBlockThreads -= 1;
           notify ();
         }
         else val += 1;
   }
}

public class GenRegion
{
   private int n;
   private int [] valSet = {8, 2, 3, 7, 4, 9, 1, 5, 11, 6, 10, 12};
   private Semaphore access;
   public GenRegion ()
   {
      n = 0;
      access = new Semaphore ();
      access.up ();
   }
   public int produceVal ()
   {
      int val = 0;
      access.down ();
      if (n < valSet.length)
         { val = valSet[n];
           n += 1;
         }
      access.up ();
      return val;
   }
   public int getSize ()
   {
      return valSet.length;
   }
}
public class StoreRegion
{
   private int mem = 0;
   private int mult;
   private int [][] tmp;
   private int [] ind;
   private Semaphore access = new Semaphore ();
   private Semaphore [] stat;
```

```
   public StoreRegion (int mult, int size)
   {
      this.mult = mult;
      tmp = new int[mult][size];
      ind = new int[mult];
      access.up ();
      stat = new Semaphore[mult+2];
      for (int i = 0; i < mult + 2; i++)
      { if (i < mult) ind[i] = 0;
        stat[i] = new Semaphore ();
      }
      stat[mult+1].up ();
   }
   public void putVal (int val)
   {
      int m = (val % 100) % mult;
      stat[mult+1].down ();
      access.down ();
      if (ind[(m + 1) % mult] != 0)
         { ind[(m + 1) % mult] -= 1;
           mem = 10000 * (tmp[(m + 1) % mult][ind[(m + 1) % mult]] / 100) +
                 tmp[(m + 1) % mult][ind[(m + 1) % mult]] % 100 + 100 * (val / 100) +
                 val % 100;
           for (int i = 0; i <= mult; i++)
             stat[i].up ();
         }
         else { tmp[m][ind[m]] = val;
                ind[m] += 1;
                stat[mult+1].up ();
              }
      access.up ();
      stat[m].down ();
   }
   public int getVal ()
   {
      int val;
      stat[mult].down ();
      access.down ();
      val = mem;
      mem = 0;
      stat[mult+1].up ();
      access.up ();
      return val;
   }
}

public class Resource
{
   private int n;
   private Semaphore access = new Semaphore ();
   public Resource (int n)
   {
      this.n = n;
      access.up ();
   }
   public boolean printVal (int val)
   {
      access.down ();
      if ((n > 0) && (val != 0))
         { System.out.println ("O valor processado por " + (val/10000) + " e por " +
                   ((val-10000*(val/10000))/100) + " foi " + (val%100) + ".");
           n -= 1;
         }
      access.up ();
      return (n == 0);
   }
}
```

```
public class ThreadType1 extends Thread
{
   private int id;
   private GenRegion gen;
   private StoreRegion store;
   public ThreadType1 (int id, GenRegion gen, StoreRegion store)
   {
      this.id = id;
      this.gen = gen;
      this.store = store;
   }
   public void run ()
   {
      int val;
      do
      { try
        { Thread.sleep ((long) (1 + 10*Math.random ()));
        }
        catch (InterruptedException e) {};
        val = gen.produceVal ();
        if (val != 0)
           { try
             { Thread.sleep ((long) (1 + 10*Math.random ()));
             }
             catch (InterruptedException e) {};
             val += 100*id;
             store.putVal (val);
           }
      } while (val != 0);
   }
}

public class ThreadType2 extends Thread
{
   private StoreRegion store;
   private Resource writer;
   public ThreadType2 (StoreRegion store, Resource writer)
   {
      this.store = store;
      this.writer = writer;
   }
   public void run ()
   {
      int val;
      boolean end = false;
      while (!end)
      { val = store.getVal ();
        end = writer.printVal (val);
      }
   }
}

public class SimulSituation
{
   public static void main (String [] args)
   {
      GenRegion gen = new GenRegion ();
      StoreRegion store = new StoreRegion (2, gen.getSize () / 2);
      Resource writer = new Resource (gen.getSize () / 2);
      ThreadType1 [] thr1 = new ThreadType1[gen.getSize () / 2 + 1];
      for (int i = 0; i < gen.getSize () / 2 + 1; i++)
        thr1[i] = new ThreadType1 (i + 1, gen, store);
      ThreadType2 thr2 = new ThreadType2 (store, writer);
      thr2.start ();
      for (int i = 0; i < gen.getSize () / 2 + 1; i++)
        thr1[i].start ();
   }
}
```

1. Representando as entidades activas por círculos e as entidades passivas por rectângulos, faça um diagrama ilustrativo da interacção em presença e indique por palavras simples qual é o papel desempenhado pelos *threads* de cada tipo (não mais do que uma frase).
2. Explique detalhadamente qual é o papel desempenhado por cada um dos elementos do *array* `stat` do tipo de dados `StoreRegion`.
3. Forneça o texto impresso por uma execução do programa. Tenha em atenção que, face à aleatoridade introduzida, os valores impressos não são únicos.
4. Altere o programa de modo a garantir o processamento de *ternos* de valores. Comece por indicar o que significa o processamento de *ternos* de valores e aponte, face a isso, que tipo de modificações vai introduzir. Só depois apresente as alterações.