

Considere o código seguinte:

```
public class GenRegion
{
    private int n = 0;
    private int [] valSet = {5, 12, 3, 6, 4, 2, 1, 18, 15, 16, 9, 7};
    public synchronized int produceVal ()
    {
        if (n == valSet.length)
            return 0;
        else { int val = valSet[n];
              n += 1;
              return val;
            }
    }
}

public class StoreRegion
{
    private int mem = 0;
    private int stat = 0;
    public synchronized void putVal (int val)
    {
        while (stat != 0)
        { try
          { wait ();
            }
          catch (InterruptedException e) {}
        }
        stat = 1;
        mem = val;
        notifyAll ();
        while (stat == 1)
        { try
          { wait ();
            }
          catch (InterruptedException e) {}
        }
    }
    public synchronized int getVal ()
    {
        while ((stat != 1) && (stat != 2))
        { try
          { wait ();
            }
          catch (InterruptedException e) {}
        }
        if (stat == 1) stat = 0;
        int val = mem;
        mem = 0;
        notifyAll ();
        return val;
    }
    public synchronized void noWait ()
    {
        stat = 2;
        notifyAll ();
    }
}

public class Resource
{
    private int n;
    private StoreRegion [] store = null;
    public Resource (int n, StoreRegion [] store)
    {
        this.n = n;
        this.store = store;
    }
}
```

```

    public synchronized boolean printVal (int id, int val)
    {
        if (n >= 1)
        { System.out.println ("0 valor processado por " + (val/100) + " e por " +
                             id + " foi " + (val%100) + ".");
          n -= 1;
          if (n == 0)
              for (int i = 0; i < 2; i++)
                  store[i].noWait ();
        }
        return (n == 0);
    }
}

public class ThreadType1 extends Thread
{
    private int id;
    private GenRegion gen = null;
    private StoreRegion [] store = null;
    public ThreadType1 (int id, GenRegion gen, StoreRegion [] store)
    {
        this.id = id;
        this.gen = gen;
        this.store = store;
    }
    public void run ()
    {
        int val;
        do
        { try
          { sleep ((int) (1 + 10*Math.random ()));
            }
          catch (InterruptedException e) {};
          val = gen.produceVal ();
          try
          { sleep ((int) (1 + 10*Math.random ()));
            }
          catch (InterruptedException e) {};
          if (val != 0)
              switch (val % 3)
              { case 0: store[0].putVal (100*id+2*val);
                break;
                case 1:
                case 2: store[1].putVal (100*id+val);
              }
        } while (val != 0);
    }
}

public class ThreadType2 extends Thread
{
    private int id;
    private StoreRegion [] store = null;
    private Resource writer = null;
    public ThreadType2 (int id, StoreRegion [] store, Resource writer)
    {
        this.id = id;
        this.store = store;
        this.writer = writer;
    }
    public void run ()
    {
        int val;
        boolean end = false;
        while (!end)
        { val = store[(id-1)/2].getVal ();
          try
          { sleep ((int) (1 + 10*Math.random ()));
            }
          catch (InterruptedException e) {};
          end = writer.printVal (id, val);
        }
    }
}

```

```

public class SimulSituation
{
    public static void main (String [] args)
    {
        StoreRegion [] store = new StoreRegion [2];
        for (int i = 0; i < 2; i++)
            store[i] = new StoreRegion ();
        GenRegion gen = new GenRegion ();
        Resource writer = new Resource (12, store);
        ThreadType1 [] thr1 = new ThreadType1[4];
        for (int i = 0; i < 4; i++)
            thr1[i] = new ThreadType1 (i+1, gen, store);
        ThreadType2 [] thr2 = new ThreadType2[4];
        for (int i = 0; i < 4; i++)
            thr2[i] = new ThreadType2 (i+1, store, writer);
        for (int i = 0; i < 4; i++)
            thr2[i].start ();
        for (int i = 0; i < 4; i++)
            thr1[i].start ();
    }
}

```

- 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).
- Assuma que, quando o programa é executado, se obtém o resultado seguinte
 - 0 valor processado por 1 e por 2 foi 24.
 - 0 valor processado por 3 e por 1 foi 3.
 - 0 valor processado por 1 e por 2 foi 12.
 - 0 valor processado por 2 e por 4 foi 5.
 - 0 valor processado por 4 e por 3 foi 4.
 - 0 valor processado por 4 e por 4 foi 11.
 - 0 valor processado por 3 e por 2 foi 36.
 - 0 valor processado por 1 e por 3 foi 2.
 - 0 valor processado por 4 e por 4 foi 7.
 - 0 valor processado por 2 e por 1 foi 30.
 - 0 valor processado por 4 e por 4 foi 16.
 - 0 valor processado por 1 e por 1 foi 18.

Tenha em atenção que, face à aleatoriedade introduzida, este não é o único resultado possível. De facto, nem sequer está correcto. Existem três erros, respectivamente, nas linhas 2, 6 e 9. Identifique-os. Justifique cuidadosamente a sua resposta.

- Explique como é sempre garantida a terminação do programa.
- Altere o programa de modo a ser possível processar pares (somas) de valores. Comece por explicar que tipo de alterações tem que fazer face à organização do programa que lhe é apresentado.