Consider the following code

```
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 ("The value processed by " + (val/100) + " and by " +
                               id + " was " + (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 ();
   }
}
```

=> mudar 12 para 6 (em vez de 12 valores serão 6 pares de valores)

Active Entities => Threads

Passive Entities => Regions

Seguir a simulation

1. Representing the active entities by circles and the passive entities by squares, sketch a diagram which illustrates the interaction that is taking place and describe in simple words the role played by the threads of each type (do not use more than one or two sentences in your explanation).
2. Assume that, when the program is run, one gets the following printing

```
The value processed by 1 and by 2 was 24.
The value processed by 3 and by 1 was 3.
The value processed by 1 and by 2 was 12.
The value processed by 2 and by 4 was 5.
The value processed by 4 and by 3 was 4.
The value processed by 4 and by 4 was 11.
The value processed by 3 and by 2 was 36.
The value processed by 1 and by 3 was 2.
The value processed by 4 and by 4 was 7.
The value processed by 2 and by 1 was 30.
The value processed by 4 and by 4 was 16.
The value processed by 1 and by 1 was 18.
```

"The value processed by " + (val/100) + " and by " + id + " was " + (val%100) + "."

   Bear in mind that, because of the randomness which was introduced, this is not the only possible result. In fact, it is not even correct. There are three errors in the printing at lines 2, 6 and 9, respectively. Identify them and justify your claims carefully.
3. Explain how the termination of the program is always ensured.
4. Change the code so that it becomes possible to process pairs of values (the values of each pair should be added). Start your answer by pointing out the changes that have to be made. They should be minimal.

3. Como o seu ciclo de vida foi projetado, as threads do tipo 2 não controlam diretamente sua terminação. Quando o último valor é impresso, a thread que o fez termina imediatamente. No entanto, as outras threads estão bloqueadas ou logo estarão, nas variáveis de condição implícitas dos monitores store[0] e store[1]. Assim, a operação noWait é então chamada por essa thread nos dois monitores para acordar as threads restantes e permitir sua terminação.