David Criado Ramón

PORTAFOLIOS PRÁCTICA 2 - MONITORES HOARE EN JAVA

Productor/consumidor con buffer limitado

En otro caso más en el que tenemos múltiples hebras que escriben y leen de memoria compartida a partir de un buffer (array) de tamaño estático. Para ello utilizo las siguientes variables y colas condición:

int numSlots → Indica el tamaño del array a utilizar como búffer.
int cont → Indica el número de posiciones del array ocupadas.
double[] buffer → El buffer en sí

Condition lectura→ Condición para la lectura.
Espera: Si el buffer está vacío antes de extraer.
Señal: Cuando se haya depositado en el búffer.

Condition escritura → Condición para la escritura.
Espera: Si el buffer está lleno antes de depositar.
Señal: Cuando se haya estraído un elemento del búffer.

Código fuente:

```
import monitor.*;

class Buffer extends AbstractMonitor
{
  private int       numSlots = 0     ,
                    cont     = 0     ;
  private double[] buffer    = null ;
  private Condition lectura = makeCondition();
  private Condition escritura = makeCondition();

  public Buffer( int p_numSlots )
  {
    numSlots = p_numSlots ;
    buffer = new double[numSlots] ;
  }
  public void depositar( double valor ) throws InterruptedException
  {
    enter(); // Entro
    if (cont==numSlots)
      escritura.await();    // Espero a escritura
    buffer[cont] = valor;
    cont++ ;
    lectura.signal(); // Doy paso a lectura


    leave(); // Salgo
  }
  public double extraer() throws InterruptedException
  {
    double valor ;
    enter(); // Entro a monitor
    if (cont==0)
      lectura.await(); // Espero a lectura
```

```
    cont--;
    valor = buffer[cont]; // Reduzco cont
    escritura.signal(); // Mando señal de escritura
    leave(); // Salgo monitor
    return valor;
  }
}

// ****************************************************************************

class Productor implements Runnable
{
  private Buffer bb     ;
  private int    veces ,
                 numP   ;
  public  Thread thr    ;

  public Productor( Buffer pbb, int pveces, int pnumP )
  {
    bb    = pbb;
    veces = pveces;
    numP  = pnumP ;
    thr   = new Thread(this,"productor "+numP);
  }

  public void run()
  {
    try
    {
      double item = 100*numP ;

      for( int i=0 ; i < veces ; i++ )
      {
        System.out.println(thr.getName()+", produciendo " + item);
        bb.depositar( item++ );
      }
    }
    catch( Exception e )
    {
      System.err.println("Excepcion en main: " + e);
    }
  }
}

// ****************************************************************************

class Consumidor implements Runnable
{
  private Buffer  bb     ;
  private int     veces ,
                  numC   ;
  public  Thread  thr    ;

  public Consumidor( Buffer pbb, int pveces, int pnumC )
  {
```

```
    bb     = pbb;
    veces = pveces;
    numC  = pnumC ;
    thr    = new Thread(this,"consumidor "+numC);
  }
  public void run()
  {
    try
    {
      for( int i=0 ; i<veces ; i++ )
      {
        double item = bb.extraer ();
        System.out.println(thr.getName()+", consumiendo "+item);
      }
    }
    catch( Exception e )
    {
      System.err.println("Excepcion en main: " + e);
    }
  }
}

// ************************************************************************

class MainProductorConsumidor
{
  public static void main( String[] args )
  {
    if ( args.length != 5 )
    {
      System.err.println("Uso: ncons nprod tambuf niterp niterc");
      return ;
    }

    // leer parametros, crear vectores y buffer intermedio
    Consumidor[] cons       = new Consumidor[Integer.parseInt(args[0])] ;
          Productor[]  prod       = new Productor[Integer.parseInt(args[1])] ;
          Buffer       buffer     = new Buffer(Integer.parseInt(args[2]));
          int          iter_cons = Integer.parseInt(args[3]);
          int          iter_prod = Integer.parseInt(args[4]);

    if ( cons.length*iter_cons != prod.length*iter_prod )
          {
      System.err.println("no coinciden número de items a producir con a cosumir");
      return ;
    }

          // crear hebras
          for(int i = 0; i < cons.length; i++)
            cons[i] = new Consumidor(buffer,iter_cons,i) ;
          for(int i = 0; i < prod.length; i++)
            prod[i] = new Productor(buffer,iter_prod,i) ;

          // poner en marcha las hebras
          for(int i = 0; i < prod.length; i++)
```

```
        prod[i].thr.start();
            for(int i = 0; i < cons.length; i++)
        cons[i].thr.start();
    }
}
```

Salida:

```
productor 0, produciendo 0.0
productor 0, produciendo 1.0
consumidor 0, consumiendo 0.0
productor 1, produciendo 100.0
consumidor 0, consumiendo 100.0
consumidor 1, consumiendo 1.0
productor 0, produciendo 2.0
consumidor 0, consumiendo 2.0
productor 1, produciendo 101.0
consumidor 1, consumiendo 101.0
productor 1, produciendo 102.0
productor 0, produciendo 3.0
consumidor 0, consumiendo 102.0
productor 1, produciendo 103.0
productor 1, produciendo 104.0
consumidor 0, consumiendo 104.0
consumidor 1, consumiendo 103.0
productor 0, produciendo 4.0
consumidor 1, consumiendo 3.0
consumidor 1, consumiendo 4.0
```

El problema de los fumadores

Exactamente tal y cómo se propuso en la práctica anterior pero con semáforos, en esta práctica utilizaremos monitores para resolver el problema en el que tenemos tres hebras fumadoras, a las que a cada una de ellas le falta un ingrediente, y existen una única hebra estanquero, que sólo puede producir un ingrediente de los que falta aleatoriamente y debe esperar hasta que se recoja para producir otro.

Para la resolución del problema utilizo las siguientes variables y colas condición:

int ingrediente → Indica qué ingrediente se ha producido en caso de haberlo.
boolean mostradorLleno → Indica si hay un ingrediente en el mostrador o no

Condition colaFumadores[] → Condición para cada uno de los fumadores.
Espera: si no está si ingrediente (mostrador vacío o ingrediente distinto).
Señal: si el estanquero produce su ingrediente.

Condition colaEstanquero → Condición para el estanquero.
Espera: si hay algo en el mostrador.
Señal: si se retira un ingrediente del mostrador.

Código fuente:

```
import monitor.*;

class Estanco extends AbstractMonitor
{
```

```
    /** No hay nada -> Ingrediente -1
     *  Cerillas -> Ingrediente 0 Fumador 0
     *  Papel -> Ingrediente 1 Fumador 1
     *  Tabaco -> Ingrediente 2 Fumador 2
     */
    private int ingrediente;
    private Condition colaEstanquero;
    private Condition[] colaFumadores=new Condition[3];
    private boolean mostradorLleno;


    public Estanco(){ // Inicializamos las variables condicion y el mostrador vacío
        colaEstanquero=makeCondition();
        for (int i=0; i<colaFumadores.length; ++i)
            colaFumadores[i]=makeCondition();
        mostradorLleno=false;


    }
    public void obtenerIngrediente (int miIngrediente){
        enter();
        if (!mostradorLleno || ingrediente!=miIngrediente)
            colaFumadores[miIngrediente].await();

        switch(miIngrediente){
            case 0:
                System.out.println("Fumador 0: He obtenido cerillas.");
                break;
            case 1:
                System.out.println("Fumador 1: He obtenido papel.");
                break;
            case 2:
                System.out.println("Fumador 2: He obtenido tabaco.");
                break;
        }
        mostradorLleno=false;

        colaEstanquero.signal();

        leave();
    }

    public void ponerIngrediente( int ingrediente ){
        enter();
        switch(ingrediente){
            case 0:
                System.out.println("Estanquero: Produciendo cerillas.");
                break;
            case 1:
                System.out.println("Estanquero: Produciendo papel.");
                break;
            case 2:
                System.out.println("Estanquero: Produciendo tabaco.");
            break;
        }
```

```
            this.ingrediente=ingrediente;
            mostradorLleno=true;
            colaFumadores[ingrediente].signal();
        leave();


    }

    public void esperarRecogidaIngrediente(){
        enter();
            System.out.println("Estanquero: Mostrador lleno (1 objeto).");
            if (mostradorLleno)
                colaEstanquero.await();
            System.out.println("Estanquero: Mostrador vacio (0 objetos).");
        leave();
    }
}

class Estanquero implements Runnable {
    public Thread thr;
    private Estanco estanco;

    public Estanquero(Estanco estanco){
        this.estanco=estanco;
        thr=new Thread(this);
    }

    public void run(){
        int ingrediente;
        while (true){
            ingrediente = (int) (Math.random () * 3.0); // 0, 1 o 2
            estanco.ponerIngrediente(ingrediente);
            estanco.esperarRecogidaIngrediente();
        }
    }
}

class Fumador implements Runnable {
    public Thread thr;
    private Estanco estanco; // Estanco al que acude (monitor)
    private int miIngrediente; // Nº de fumador/ingrediente

    public Fumador(Estanco estanco, int nfumador){
        this.estanco=estanco;
        this.miIngrediente=nfumador;
        thr=new Thread(this,"Fumador "+Integer.toString(nfumador));
    }

    private void Gaste(){
        if (miIngrediente==0)
         System.out.println(thr.getName()+": Me quede sin cerillas.");
        else if (miIngrediente==1)
         System.out.println(thr.getName()+": Me quede sin papel.");
        else
         System.out.println(thr.getName()+": Me quede sin tabaco.");
```

```
    }

    public void run(){
        while (true){
            estanco.obtenerIngrediente(miIngrediente);
            aux.dormir_max(2000);
            Gaste();
        }
    }
}

class Fumadores {
    public static void main( String[] args ) {
        // Creamos variables
        Estanco estanco=new Estanco();
        Estanquero estanquero=new Estanquero(estanco);
        Fumador fumadores[]=new Fumador[3];

        for (int i=0; i<fumadores.length; ++i)
            fumadores[i]=new Fumador(estanco,i);


        // Ponemos en marcha las hebras
        estanquero.thr.start();
        for (int i=0; i<fumadores.length; ++i)
            fumadores[i].thr.start();


    }
}
```

Salida:

Estanquero: Produciendo cerillas.
Estanquero: Mostrador lleno (1 objeto).
Fumador 0: He obtenido cerillas.
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo tabaco.
Fumador 2: He obtenido tabaco.
Estanquero: Mostrador lleno (1 objeto).
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo cerillas.
Estanquero: Mostrador lleno (1 objeto).
Fumador 0: Me quede sin cerillas.
Fumador 0: He obtenido cerillas.
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo papel.
Fumador 1: He obtenido papel.
Estanquero: Mostrador lleno (1 objeto).
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo cerillas.
Estanquero: Mostrador lleno (1 objeto).
Fumador 2: Me quede sin tabaco.
Fumador 1: Me quede sin papel.
Fumador 0: Me quede sin cerillas.

David Criado Ramón

Fumador 0: He obtenido cerillas.
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo cerillas.
Estanquero: Mostrador lleno (1 objeto).
Fumador 0: Me quede sin cerillas.
Fumador 0: He obtenido cerillas.
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo tabaco.
Fumador 2: He obtenido tabaco.
Estanquero: Mostrador lleno (1 objeto).
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo cerillas.
Estanquero: Mostrador lleno (1 objeto).
Fumador 2: Me quede sin tabaco.
Fumador 0: Me quede sin cerillas.
Fumador 0: He obtenido cerillas.
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo cerillas.
Estanquero: Mostrador lleno (1 objeto).
Fumador 0: Me quede sin cerillas.
Fumador 0: He obtenido cerillas.
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo tabaco.
Fumador 2: He obtenido tabaco.
Estanquero: Mostrador lleno (1 objeto).
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo tabaco.
Estanquero: Mostrador lleno (1 objeto).
Fumador 2: Me quede sin tabaco.
Fumador 2: He obtenido tabaco.
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo tabaco.
Estanquero: Mostrador lleno (1 objeto).
Fumador 2: Me quede sin tabaco.
Fumador 2: He obtenido tabaco.
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo papel.
Fumador 1: He obtenido papel.
Estanquero: Mostrador lleno (1 objeto).
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo tabaco.
Estanquero: Mostrador lleno (1 objeto).
Fumador 0: Me quede sin cerillas.
Fumador 2: Me quede sin tabaco.
Fumador 2: He obtenido tabaco.
Estanquero: Mostrador vacio (0 objetos).
Estanquero: Produciendo papel.
Estanquero: Mostrador lleno (1 objeto).

David Criado Ramón

El problema del barbero durmiente

Para la resolución del problema utilizo las siguientes variables y colas condición:

boolean ocupado → Indica si el barbero está atendiendo a algún cliente

boolean durmiendo → Indica si el barbero está durmiendo

Condition despertador → Controla el tiempo que duerme el barbero.
Espera: si no hay clientes cuando el barbero los solicita.
Señal: si el barbero está durmiendo cuando entra un cliente.

Condition finCorte → Controla el tiempo que tarda el barbero en cortar el pelo.
Espera: Cuando el cliente entra a cortar el pelo.
Señal: Cuando el barbero termina de cortar el pelo.

Condition colaClientes → Controla la cola de cliente.
Espera: Si el barbero está ocupado al entrar un cliente.
Señal: Cuando el barbero solicita un nuevo cliente.

Código fuente:

```
import monitor.*;


class Barberia extends AbstractMonitor {
    private Condition despertador=makeCondition();
    private Condition finCorte=makeCondition();
    private Condition colaClientes=makeCondition();

    private boolean ocupado=false;
    private boolean durmiendo=false;
    // Invocado por los clientes para cortarse el pelo
    public void cortarPelo(){
      enter();
      if (ocupado)
          colaClientes.await();
      if (durmiendo) {
          System.out.println("Despierte, barbero");
          durmiendo=false;
          despertador.signal();
      }

      ocupado=true;
      System.out.println("Me estan cortando el pelo");
      finCorte.await();
      System.out.println("Terminaron de cortarme el pelo");
      leave();
    }

    // Invocado por el barbero para esperar(si procede) a un nuevo cliente y sentarlo para el corte
    public void siguienteCliente(){
        enter();
        if (colaClientes.isEmpty()){
            durmiendo=true;
            System.out.println("Barbero durmiendo");
            despertador.await();
```

```
        }
        else
            colaClientes.signal();
        leave();
    }

    // Invocado por el barbero para indicar que ha terminado de cortar el pelo
    public void finCliente(){
        enter();
        if (ocupado) {
            finCorte.signal();
            ocupado=false;
        }
        leave();


    }
}

class Cliente implements Runnable {
    public Thread thr;
    private int ncliente;
    private Barberia barberia;
    public Cliente(Barberia barba, int ncliente){
        barberia=barba;
        this.ncliente=ncliente;
        thr=new Thread(this,"Cliente " + ncliente);
    }
    public void run(){
        while(true) {
            barberia.cortarPelo(); // El cliente espera (si procede) y se corta el pelo
            aux.dormir_max(2000); // El cliente está fuera de la barbería un tiempo
        }
    }
}

class Barbero implements Runnable {
    public Thread thr;
    private Barberia barberia;
    public Barbero(Barberia barba){
        barberia=barba;
        thr=new Thread(this);
    }

    public void run(){
        while(true){
            barberia.siguienteCliente(); // Barbero sienta al proximo cliente
            aux.dormir_max(2500); // Barbero cortando pelo al cliente
            barberia.finCliente(); // Barbero ha terminado de cortar el pelo al cliente
        }
    }
}
```

David Criado Ramón

```
class BarberoDurmiente {
    public static void main( String[] args ) {

        // Creamos variables
        Barberia barberia=new Barberia();
        Barbero barbero=new Barbero(barberia);
        Cliente clientes[]=new Cliente[3];

        for (int i=0; i<clientes.length; ++i)
            clientes[i]=new Cliente(barberia,i);


        // Ponemos en marcha las hebras
        barbero.thr.start();
        for (int i=0; i<clientes.length; ++i)
            clientes[i].thr.start();
    }
}
```

Salida:

```
Barbero durmiendo
Despierte, barbero
Me estan cortando el pelo
Me estan cortando el pelo
Terminaron de cortarme el pelo
Me estan cortando el pelo
Terminaron de cortarme el pelo
Me estan cortando el pelo
Terminaron de cortarme el pelo
Me estan cortando el pelo
Terminaron de cortarme el pelo
Me estan cortando el pelo
Terminaron de cortarme el pelo
Barbero durmiendo
Despierte, barbero
Me estan cortando el pelo
Terminaron de cortarme el pelo
Me estan cortando el pelo
Terminaron de cortarme el pelo
Me estan cortando el pelo
Terminaron de cortarme el pelo
Me estan cortando el pelo
Terminaron de cortarme el pelo
Barbero durmiendo
Despierte, barbero
Me estan cortando el pelo
```