Adrián de la Torre Rodríguez 2ºA

PORTAFOLIOS PRACTICA 2

BUFFER

En este programa la clase Buffer se transforma en monitor de estilo Hoare, y cada método que tiene se mete entre un enter() y un leave() para asegurar la exclusión mutua. También cambia la manera en que las hebras se bloquean y se reanudan: En el programa con monitor nativo en java se comprobaba periódicamente si el vector estaba en uno de sus extremos y se hacía un wait()

```
while( cont == numSlots )
          wait();
```

Y después de consumir o producir se notificaba a todas las hebras con un notifyAll()

```
buffer[cont] = valor ;
cont++ ;
notifyAll() ;
```

El resto del programa, tanto funciones como el main se mantienen igual.

CODIGO FUENTE DEL PROGRAMA

```
import monitor.*;

class Buffer extends AbstractMonitor{

  private int      numSlots = 0    ,
                   cont     = 0    ;
  private double[] buffer   = null ;

  //Condiciones que uso para evitar que consuma o produzca cuando el
//vector esta en sus límites
  private Condition producir = makeCondition();
  private Condition consumir = makeCondition();

  //Constructor
  public Buffer( int p_numSlots )
  {
    numSlots = p_numSlots ;
    buffer = new double[numSlots] ;
  }

  //Función que deposita un valor en la posición correspondiente del vector
  public  void depositar( double valor )
  {

    enter();

    if(cont == numSlots)    //Si el vector esta completo...
        producir.await();   //Se espera a que otra hebra saque un dato

    buffer[cont] = valor ;  //Se mete el dato y se actualiza el contador
    cont++ ;
    consumir.signal();      //Se indica que se puede consumir

    leave();
  }

  //Función que extrae un valor del vector en la posición correspondiente
  public double extraer()
  {


    enter();

    if( cont == 0)             //Si el vector esta vacio...
        consumir.await();      //Se espera a que se meta un dato en el vector

    cont--;                          //Se actualiza contador
    double valor = buffer[cont] ;   //Se almacena en "valor" el dato del vector
```

```
    producir.signal();              //Se indica que se puede producir

    leave();
    return valor;


  }
}

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

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


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

  public void run()
  {

      double item = 100*numP ;

      for( int i=0 ; i < veces ; i++ )
      {
        System.out.println(thr.getName()+", produciendo " + item);
        bb.depositar( item++ );
      }


  }
}

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

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

```
  //Constructor
  public Consumidor( Buffer pbb, int pveces, int pnumC )
  {
    bb    = pbb;
    veces = pveces;
    numC  = pnumC ;
    thr   = new Thread(this,"consumidor "+numC);
  }
  public void run()
  {

      for( int i=0 ; i<veces ; i++ )
      {
        double item = bb.extraer ();
        System.out.println(thr.getName()+", consumiendo "+item);
      }


  }
}

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

class MainProductorConsumidorMonitor
{
  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 ;
    }

        // creación 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) ;
```

```
        // puesta 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();

  }
}
```

LISTADO DE SALIDA

Para 2 productores, 2 consumidores, un buffer de tamaño 3 y 5 iteraciones por hebra uso:

"java MainProductorConsumidorMonitor 2 2 3 5 5"

Y obtengo el siguiente resultado:

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

FUMADORES

Las condiciones que he puesto en este programa son las siguientes:

```
 private Condition puede_recoger_0 = makeCondition();
 private Condition puede_recoger_1 = makeCondition();
 private Condition puede_recoger_2 = makeCondition();
 private Condition puede_fabricar = makeCondition();
```

Las condiciones “puede_recoger_x” se activan por el estanquero, que cuando produce un ingrediente, hace un signal() a la condición correspondiente, es decir, que indica a la hebra fumadora que esta bloqueada que su ingrediente se ha producido y que puede recogerlo.

Una vez recogido el ingrediente, el estanquero hace un wait() a la condicion “puede_fabricar”, es decir, espera a que alguien recoja su ingrediente y haga un signal a “puede_fabricar”.

CODIGO DEL PROGRAMA

```
import monitor.* ;

class Estanco extends AbstractMonitor{


    //Condiciones que indican si cada fumador puede recoger su ingrediente
    // y si el estanquero puede fabricar o no
    private Condition puede_recoger_0 = makeCondition();
    private Condition puede_recoger_1 = makeCondition();
    private Condition puede_recoger_2 = makeCondition();
    private Condition puede_fabricar = makeCondition();

    // Invocado por cada fumador, indicando su ingrediente o numero
    public void obtenerIngrediente( int miIngrediente ){

        enter();


        /*
           Lógicamente se debería de poner este signal después de las siguientes
           ya que se debe indicar, que se puede fabricar una vez recogidos los
           ingredientes. Pero si lo hago así las hebras se interbloquean, así que
           he optado por ponerla antes de las condiciones. No hay problema ya que
           este trozo de código se ejecuta en exclusión mutua y no importa si se
           pone antes o después
        */
        puede_fabricar.signal();
```

```
        //Condiciones
        //Según el ingrediente que se pase como argumento se esperará a que el
        // estanquero indique que se puede recoger
        if( miIngrediente == 0 ){
            puede_recoger_0.await();
        }
        else if( miIngrediente == 1 ){
            puede_recoger_1.await();
        }
        else if( miIngrediente == 2 ){
            puede_recoger_2.await();
        }


        leave();

    }



    // Invocado por el estanquero, indicando el ingrediente que pone
    public void ponerIngrediente( int ingrediente ){

        enter();

        //Condiciones
        //Se indica que se puede recoger el ingrediente que se pasa como argumento
        if( ingrediente == 0 ){
            puede_recoger_0.signal();
        }
        else if( ingrediente == 1 ){
            puede_recoger_1.signal();
        }
        else if( ingrediente == 2 ){
            puede_recoger_2.signal();
        }

        leave();

    }



    // Invocado por el estanquero
    public void esperarRecogidaIngrediente(){

        enter();

        //Se espera a que una hebra fumadora recoja su ingrediente
        // y indique que se puede fabricar
        puede_fabricar.await();

        leave();
    }

}
```

```
/*
    Al fumador 0 le falta tabaco,
    al fumador 1 le falta papel y
    al fumador 2 le faltan cerillas
*/

class Fumador implements Runnable{

    private int miIngrediente;
    private String ingre;
    public Thread thr ;
    public Estanco estanco;

    //Constructor
    public Fumador( int p_miIngrediente, String nombre, Estanco es  ){

        miIngrediente = p_miIngrediente;
        thr = new Thread(this, nombre);
        estanco = es;

        //Almaceno el nombre del ingreiente en un String simplemente
        // por comodidad a la hora de imprimir por pantalla
        if( p_miIngrediente == 0)
            ingre = "tabaco";
        else if( p_miIngrediente == 1)
            ingre = "papel";
        else if( p_miIngrediente == 2)
            ingre = "cerillas";

    }

    public void run(){
        while ( true ){

             /*
                Se imprime por pantalla que el fumador esta esperando, y se indica al estanco
              que se quiere recoger el ingrediente indicado. Cuando se obtiene se imprime
              por pantalla que se ha obtenido el ingrediente.
             */
             System.out.println("El fumador "+ miIngrediente +" esta esperando " + ingre
);
             estanco.obtenerIngrediente( miIngrediente );
             System.out.println("El fumador "+ miIngrediente +" ha recibido " + ingre );

             /*
                Una vez obtenido el ingrediente, se imprime que se ha comenzado a fumar
                y se expera un tiempo entre 0 y 2 segundos. Cuando se acaba ese tiempo,
                se imprime que se ha terminado de fumar y se vuelve a la parte inicial
                del bucle donde se indica que se quiere el ingrediente.
             */
             System.out.println("El fumador "+ miIngrediente +" ha comenzado a fumar.");
             aux.dormir_max( 2000 );
             System.out.println("El fumador "+ miIngrediente +" ha terminado de
fumar.");

         }
    }

}
```

```
class Estanquero implements Runnable{

    public Thread thr ;
    public Estanco estanco;
    String ingre;

    //Constructor
    public Estanquero( Estanco es ){

        thr = new Thread(this);
        estanco = es;

    }

    public void run(){

        int ingrediente ;

        while (true){

            //Se genera de forma aleatorio un ingrediente
            ingrediente = (int) (Math.random () * 3.0); // 0,1 o 2

            //Se almacena el nombre del ingrediente en un String
            if( ingrediente == 0)
                ingre = "tabaco";
            else if( ingrediente == 1)
                ingre = "papel";
            else if( ingrediente == 2)
                ingre = "cerillas";

            //Se imprime que se ha producido el ingrediente
            System.out.println("El estanquero ha producido " + ingre);

            //Se llama a esta función que se encarga de indicar a la hebra fumadora
            // que se ha puesto su ingrediente
            estanco.ponerIngrediente( ingrediente );

            //Se espera a que la hebra fumadora recoja el ingrediente
            estanco.esperarRecogidaIngrediente() ;
        }
    }

}

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

class MainFumadores{

    public static void main( String[] args ) {


        final int N_FUMADORES = 3;

        Fumador[] fumadores = new Fumador[N_FUMADORES];
        Estanco estanco = new Estanco();
        Estanquero estanquero = new Estanquero( estanco );

        //Creación de hebras
        for( int i=0; i < N_FUMADORES; i++)
```

```
            fumadores[i] = new Fumador( i, "El fumador "+i, estanco);

        //Puesta en marcha de hebras
        for( int i=0; i < N_FUMADORES; i++)
            fumadores[i].thr.start();

        estanquero.thr.start();

    }



}
```

LISTADO DE SALIDA

```
El fumador 0 esta esperando tabaco
El estanquero ha producido cerillas
El fumador 1 esta esperando papel
El fumador 2 esta esperando cerillas
El fumador 2 ha recibido cerillas
El fumador 2 ha comenzado a fumar.
El fumador 2 ha terminado de fumar.
El fumador 2 esta esperando cerillas
El estanquero ha producido papel
El fumador 1 ha recibido papel
El fumador 1 ha comenzado a fumar.
El fumador 1 ha terminado de fumar.
El fumador 1 esta esperando papel
El estanquero ha producido cerillas
El fumador 2 ha recibido cerillas
El fumador 2 ha comenzado a fumar.
El fumador 2 ha terminado de fumar.
El fumador 2 esta esperando cerillas
El estanquero ha producido cerillas
El fumador 2 ha recibido cerillas
El fumador 2 ha comenzado a fumar.
El fumador 2 ha terminado de fumar.
El fumador 2 esta esperando cerillas
El estanquero ha producido tabaco
El fumador 0 ha recibido tabaco
El fumador 0 ha comenzado a fumar.

< Interrupción del programa con teclado ( Ctrl + D ) >
```

BARBERIA

Las condiciones que he usado son estas:

```
   private Condition durmiendo = makeCondition();
   private Condition puede_pelar = makeCondition();
```

La condición durmiendo se activa cuando no hay clientes. El barbero cuando llama al siguiente cliente y si ve que no hay (atributo del monitor Barbería “n_clientes == 0”) entonces hace durmiendo.await().

La condición puede_pelar se activa cuando la hebra barbero termina de pelar un cliente. La hebras cliente cuando entran a la barbería hacen un puede_pelar.await() y se bloquean hasta que el barbero finaliza de pelar a un cliente.

CODIGO DEL PROGRAMA

```
import monitor.*;


class Barberia extends AbstractMonitor{


    private Condition durmiendo = makeCondition();
    private Condition puede_pelar = makeCondition();

    private int n_clientes = 0;

    // Invcado por los clientes para cortarse el pelo
    public void cortarPelo (){

        enter();

        //Si no hay clientes es que el barbero esta dormido y hay que despertarlo
        if( n_clientes == 0 )
            durmiendo.signal();
        else
            puede_pelar.await();

        //Si se llama a cortarPelo() quiere decir que ha entrado un cliente nuevo
        //por lo tanto se aumenta el numero de clientes
        n_clientes++;


        leave();


    }

    // Invocado por el barbero para esperar (si procede) a un nuevo cliente y
    //sentarlo para el corte
    public void siguienteCliente (){
```

```
        enter();

        //Esta función básicamente sirve para comprabar si no hay clientes en
        //la barbería y que se duerma el barbero
        if(n_clientes == 0){
            System.out.println("El barbero se ha
dormido.****************************************\n");    //Pongo asteriscos para facilitar ver
            durmiendo.await();                            //Cuando el barbero se duerme
            System.out.println("El barbero se ha
despertado.****************************************\n");
        }

        leave();

    }

    // Invocado por el barbero para indicar que ha terminado de cortar el pelo
    public void finCliente (){

        enter();

        //Se indica que ya puede pelar y se decrementa el número de clientes
        puede_pelar.signal();
        n_clientes--;

        leave();

    }

}


class Cliente implements Runnable{

    public Thread thr;

    private Barberia bar;
    private int n_cliente;

    //Constructor
    public Cliente( Barberia nbar, int cli){

        bar = nbar;
        thr   = new Thread(this,"cliente " + cli);
        n_cliente = cli;

    }

    public void run(){

        while( true ){
            System.out.println("El cliente " + n_cliente + " esta esperando a ser pelado.\n");
            bar.cortarPelo(); // el cliente espera (si procede) y se corta el pelo
            System.out.println("El cliente " + n_cliente + " se ha pelado y se sale fuera.\n");
            aux.dormir_max(2000);  // el cliente está fuera de la barberia un tiempo
            System.out.println("El cliente " + n_cliente + " ha vuelto a entrar a la barberia.\n");

        }
    }
}

class Barbero implements Runnable{

    public Thread thr;

    private Barberia bar;
```

```
    //Constructor
    public Barbero( Barberia nbar){

        bar = nbar;
        thr   = new Thread(this,"barbero ");

    }



    public void run(){

        while( true ){

            //Primero siempre se llama al siguiente cliente y si no hay se duerme
            bar.siguienteCliente ();

            //Cuando llega un cliente se imprime que se esta pelando
            System.out.println("El barbero esta pelando a un cliente\n");
            aux.dormir_max( 2500 ); // el barbero esta cortando el pelo
            System.out.println("El barbero ha pelado a un cliente\n");

            //Se llama a finCliente que se encarga de indicar que se puede
            //pelar y de reducir el numero de clientes
            bar.finCliente ();

        }
    }
}


class MainBarbero{

  public static void main( String[] args ){

    if ( args.length != 1 ){

      System.err.println("Uso: <n_clientes>");
      return ;
    }

    int n_clientes = Integer.parseInt(args[0]);

    Cliente[] clientes = new Cliente[ Integer.parseInt(args[0]) ];
    Barberia barberia = new Barberia();

    //Creacion de hebras
    for( int i=0; i < n_clientes; i++)
        clientes[i] = new Cliente( barberia, i );

    Barbero barbero = new Barbero( barberia );

    //Puesta en marcha de las hebras
    barbero.thr.start();

    for( int i=0; i < n_clientes; i++)
        clientes[i].thr.start();


    }
}
```

LISTADO DE SALIDA DEL PROGRAMA

Primero pongo un listado con dos clientes para ver cuando se duerme el barbero, ya que si pongo más es muy difícil que no haya hebras esperando a ser peladas.

```
                              Nº clientes = 2

El cliente 1 esta esperando a ser pelado.

El barbero se ha dormido.*****************************************

El cliente 0 esta esperando a ser pelado.

El barbero se ha despertado.**************************************************

El barbero esta pelando a un cliente

El cliente 1 se ha pelado y se sale fuera.

El cliente 1 ha vuelto a entrar a la barberia.

El cliente 1 esta esperando a ser pelado.

El barbero ha pelado a un cliente

El cliente 0 se ha pelado y se sale fuera.

El barbero esta pelando a un cliente

El cliente 0 ha vuelto a entrar a la barberia.

El cliente 0 esta esperando a ser pelado.

El barbero ha pelado a un cliente

El cliente 1 se ha pelado y se sale fuera.

El barbero esta pelando a un cliente

El barbero ha pelado a un cliente

El cliente 0 se ha pelado y se sale fuera.

El barbero esta pelando a un cliente

El barbero ha pelado a un cliente
```

```
El barbero se ha dormido.****************************************

El cliente 1 ha vuelto a entrar a la barberia.

El cliente 1 esta esperando a ser pelado.

El barbero se ha despertado.*****************************************

El barbero esta pelando a un cliente

El cliente 1 se ha pelado y se sale fuera.

El cliente 0 ha vuelto a entrar a la barberia.

El cliente 0 esta esperando a ser pelado.

El barbero ha pelado a un cliente

El cliente 0 se ha pelado y se sale fuera.

El barbero esta pelando a un cliente

El barbero ha pelado a un cliente

El barbero se ha dormido.****************************************

El cliente 1 ha vuelto a entrar a la barberia.

El cliente 1 esta esperando a ser pelado.

El barbero se ha despertado.*****************************************

El barbero esta pelando a un cliente

El cliente 1 se ha pelado y se sale fuera.

El cliente 1 ha vuelto a entrar a la barberia.

El cliente 1 esta esperando a ser pelado.

El cliente 0 ha vuelto a entrar a la barberia.

El cliente 0 esta esperando a ser pelado.
```

Ahora un listado con 5 clientes:

Nº clientes = 5

```
El cliente 0 esta esperando a ser pelado.

El cliente 4 esta esperando a ser pelado.

El cliente 3 esta esperando a ser pelado.

El cliente 2 esta esperando a ser pelado.

El cliente 1 esta esperando a ser pelado.

El barbero se ha dormido.****************************************

El barbero se ha despertado.****************************************

El barbero esta pelando a un cliente

El cliente 0 se ha pelado y se sale fuera.

El cliente 4 se ha pelado y se sale fuera.

El cliente 0 ha vuelto a entrar a la barberia.

El cliente 0 esta esperando a ser pelado.

El cliente 4 ha vuelto a entrar a la barberia.

El cliente 4 esta esperando a ser pelado.

El barbero ha pelado a un cliente

El cliente 3 se ha pelado y se sale fuera.

El barbero esta pelando a un cliente

El barbero ha pelado a un cliente

El cliente 2 se ha pelado y se sale fuera.

El barbero esta pelando a un cliente

El cliente 3 ha vuelto a entrar a la barberia.

El cliente 3 esta esperando a ser pelado.
```