Portafolio Productor_Consumidor

Los cambio sobre la clase Buffer son:

-Añadir en la cabecera de clase Bufer extends AbstractMonitor.
-Eliminar la palabra "synchronized".
-Añadir al principio de los metodos del monitor la llamada enter() y al final leave().
-Usar await() y signal() en lugar de wait() y notifyAll().
-Ahora el bucle while es un if.

Condiciones:

Produciendo: para indicarle al consumidor que hay escritor produciendo datos.
Consumiendo: para indicar al productor que hay consumidor consumiendo datos.

Codigo fuente:

```
// ****************************************************************************
import monitor.*;

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

  public Buffer( int p_numSlots )
  {
    numSlots = p_numSlots ;
    buffer = new double[numSlots] ;
  }
  public void depositar( double valor ) throws InterruptedException
  {
        enter();
    if( cont == numSlots )
      produciendo.await();
    buffer[cont] = valor ;
    cont++ ;
    consumiendo.signal();
        leave();
}
  public double extraer() throws InterruptedException
  {
        enter();
    double valor ;
    if( cont == 0 )
      consumiendo.await() ;
```

```
    cont--;
    valor = buffer[cont] ;
    produciendo.signal();
         leave();
    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 ProductorConsumidor
{
  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 del programa para 2 productores, 2 consumidores, un buffer de tamaño 3 y 5 iteraciones por hebra:

```
yang@yang-VirtualBox:~/Escritorio/p2$ java ProductorConsumidor 2 2 3 5 5
productor 0, produciendo 0.0
productor 0, produciendo 1.0
productor 0, produciendo 2.0
productor 0, produciendo 3.0
productor 1, produciendo 100.0
productor 0, produciendo 4.0
consumidor 0, consumiendo 2.0
productor 1, produciendo 101.0
consumidor 0, consumiendo 3.0
consumidor 0, consumiendo 100.0
productor 1, produciendo 102.0
consumidor 0, consumiendo 4.0
productor 1, produciendo 103.0
consumidor 0, consumiendo 101.0
productor 1, produciendo 104.0
consumidor 1, consumiendo 102.0
consumidor 1, consumiendo 103.0
consumidor 1, consumiendo 104.0
consumidor 1, consumiendo 1.0
consumidor 1, consumiendo 0.0
```