

Problema	1
----------	---

Productor consumidor con buffer limitado

1.1 Ejercicio propuesto

Obtener una versión de la clase *Buffer*, que se desarrolló en una sección anterior para múltiples productores y consumidores, usando las clases vistas del paquete *monitor*.

1.2 Cambios sobre la clase *Buffer*

Se realizarán los siguientes cambios sobre la clase *Buffer* para que soporte la semántica de monitores estilo *Hoare*.

1. Hacer que la clase *Buffer* extienda a *AbstractMonitor*.
2. Añadir las variables condición, *productores* y *consumidores*.
3. En las cabeceras de los métodos, eliminar la sincronización del método y la excepción.
4. Añadir *enter()* y *leave()* al inicio y al final de los métodos del monitor.
5. Sustituir *wait()* por *await()*, eliminar *notifyAll()* y realizar las esperas y liberaciones sobre las variables condición. Esto es, sustituir *wait()* por *consumidores.await()* ó *productores.await()* según proceda.

1.3 Descripción de los objetos condición usados y su propósito

Se utilizan dos variables condición: *productores* y *consumidores*.

La variable *productores* servirá detendrá a las hebras productoras en el caso de que el buffer esté lleno en el método *depositar* (*productores.await()*), y para liberar a un productor si un consumidor ha liberado una posición en el buffer (*productores.signal()*) en el método *extraer*.

Para la variable *consumidores* será análogo: detendrá a los consumidores en el método *extraer* si el buffer está vacío, y se liberará cuando haya posiciones que leer si un productor ha escrito en el método *depositar*.

1.4 Código fuente

```

import monitor.*;

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

    public Buffer( int p_numSlots )
    {
        enter();
        numSlots = p_numSlots ;
        buffer = new double[numSlots] ;
        leave();
    }
    public void depositar( double valor )
    {
        enter();
        while( cont == numSlots )
            productores.await();
        buffer[cont] = valor ;
        cont++;
        consumidores.signal() ;
        leave();
    }
    public double extraer()
    {
        enter();
        double valor ;
        while( cont == 0 )
            consumidores.await() ;
        cont--;
        valor = buffer[cont] ;
        productores.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("\u001B[32m" +
                thr.getName()+"_produciendo_" + item + "\u001B[0m");
            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("\u001B[33m"+
                    thr.getName()+"_consumiendo_" + item + "\u001B[0m");
            }
        }
        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 ;
        }
    }
}

```

```
}

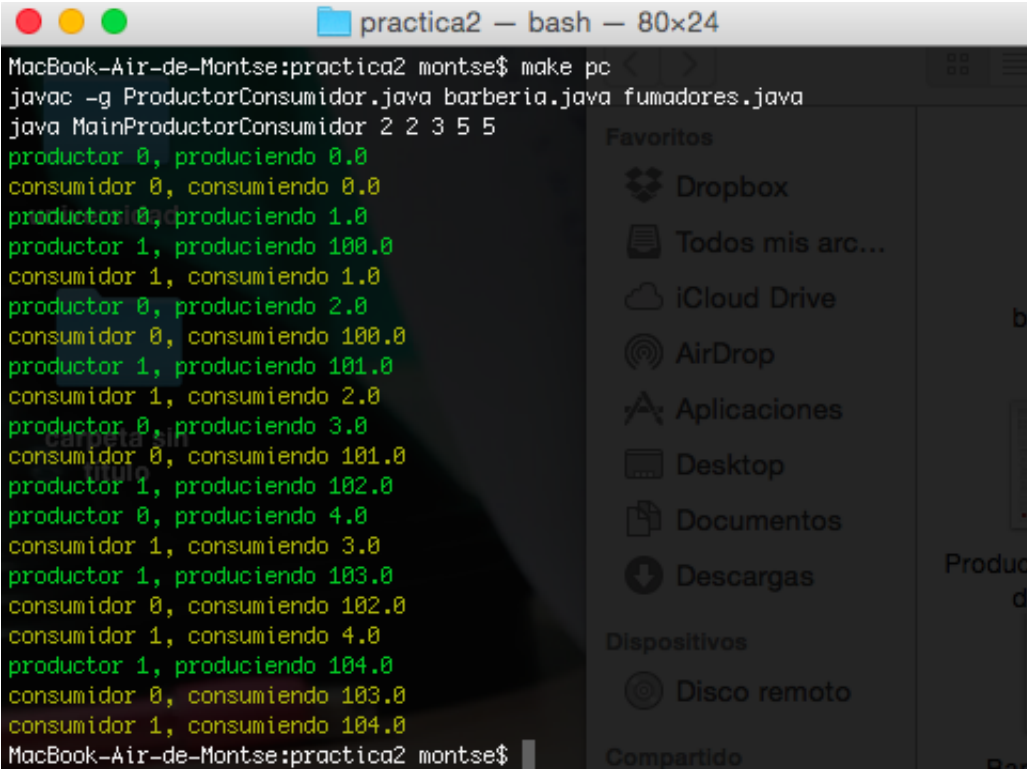
// leer parametros, crear vectores y buffer intermedio
Consumidor[] cons      = new Consumidor[Integer.parseInt(args[0])] ;
    Productor[] prod    = new Productor[Integer.parseInt(args[1])] ;
    Buffer      buffer   = new Buffer(Integer.parseInt(args[2]));
    int         iter_cons = Integer.parseInt(args[3]);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

1.5 Resultado de la ejecución



```
MacBook-Air-de-Montse:practica2 montse$ make pc
javac -g ProductorConsumidor.java barberia.java fumadores.java
java MainProductorConsumidor 2 2 3 5 5
productor 0, produciendo 0.0
consumidor 0, consumiendo 0.0
productor 0, produciendo 1.0
productor 1, produciendo 100.0
consumidor 1, consumiendo 1.0
productor 0, produciendo 2.0
consumidor 0, consumiendo 100.0
productor 1, produciendo 101.0
consumidor 1, consumiendo 2.0
productor 0, produciendo 3.0
consumidor 0, consumiendo 101.0
productor 1, produciendo 102.0
productor 0, produciendo 4.0
consumidor 1, consumiendo 3.0
productor 1, produciendo 103.0
consumidor 0, consumiendo 102.0
consumidor 1, consumiendo 4.0
productor 1, produciendo 104.0
consumidor 0, consumiendo 103.0
consumidor 1, consumiendo 104.0
MacBook-Air-de-Montse:practica2 montse$
```

Captura de pantalla 1.1: Ejemplo de ejecución del programa *ProductorConsumidor.java*

Problema 2

El problema de los fumadores

2.1 Enunciado

En este ejercicio consideraremos de nuevo el mismo problema de los fumadores y el estancoero que ya vimos en la práctica 1.

2.2 Descripción de los objetos condición usados y su propósito

Se usarán dos objetos condición: una condición para el estancoero y un array de condición para los fumadores, de manera que habrá una condición por cada fumador.

1. El objeto *condEstancoeros* se usará en los métodos *obtenerIngrediente* y *esperarRecogidaIngrediente*. Una vez que el fumador recoja el ingrediente, se usará el objeto para liberar al estancoero (*condEstancoeros.signal()*).

En el método *esperarRecogidaIngrediente*, simplemente se ejecutará *condEstancoeros.await()* para que el estancoero "duerma" mientras espera a que el fumador correspondiente recoja el ingrediente.

2. El objeto *condFumadores[i]* se usará en los métodos *obtenerIngrediente* y *ponerIngrediente*. En el método *obtenerIngrediente* servirá para que el fumador *i* espere si el ingrediente colocado sobre el mostrador no es el que le corresponde.

En el método *ponerIngrediente* la condición se usará para que el estancoero libere al al fumador *i* cuando coloque el ingrediente que le corresponda sobre el mostrador (*condFumadores[i].signal()*).

2.3 Código fuente

```

import monitor.*;
import java.util.Random;

//monitor estanco

class Estanco extends AbstractMonitor{
    String [] arrayIngredientes = {"cerillas", "tabaco", "papel"};
    int num_fumadores = 3;
    private Condition[] condFumadores = new Condition[3];
    private Condition condEstanqueros;
    private int sobre_el_mostrador;
    public Estanco(){
        condEstanqueros=makeCondition(); //inicializar estanquero
        for(int i=0; i<num_fumadores; i++){ //inicializar fumadores
            condFumadores[i] = makeCondition();
        }
        sobre_el_mostrador=-1; //inicializar variable mostrador
    }
    //invocado por cada fumador, indicando su ingrediente o numero
    public void obtenerIngrediente(int miIngrediente){
        enter();

        if(sobre_el_mostrador != miIngrediente){ //si no hay ingrediente o es
                                                    //distinto al suyo
            condFumadores[miIngrediente].await(); //esperar
        }
        System.out.println("\u001B[36m" +
            "Fumador_" + (miIngrediente+1) + "_recoge_" +
            arrayIngredientes[miIngrediente] + "." + "\u001B[0m");
        sobre_el_mostrador=-1; //el mostrador queda vacío
        condEstanqueros.signal(); //liberar al estanquero

        leave();
    }
    //invocado por el estanquero, indicando el ingrediente que pone
    public void ponerIngrediente(int ingrediente){
        enter();

        sobre_el_mostrador = ingrediente; //ponerlo sobre el mostrador
        System.out.println("\u001B[33m" +
            "El_estanquero_coloca_" + arrayIngredientes[ingrediente]
            + "_sobre_el_mostrador" + "\u001B[0m");
        condFumadores[ingrediente].signal(); //liberar al fumador correspondiente

        leave();
    }
    //invocado por el estanquero
    public void esperarRecogidaIngrediente(){
        enter();

        if(sobre_el_mostrador != -1){ //si el mostrador no está vacío, dormir
            condEstanqueros.await();
        }

        leave();
    }

```

```

    }
}

/*class aux{
    static Random genAlea = new Random();
    static void dormir_max(int milisecsMax){
        try{
            Thread.sleep(genAlea.nextInt(milisecsMax));
        } catch (InterruptedException e){
            System.err.println("Sleep interrumpido.");
        }
    }
}*/

//hebra fumador
class Fumador implements Runnable {
    int miIngrediente;
    private Estanco estanco = new Estanco();
    public Thread thr; //objeto hebra encapsulado
    //constructor
    public Fumador(Estanco estanco, int p_miIngrediente){
        miIngrediente = p_miIngrediente;
        this.estanco = estanco;
        thr = new Thread(this, "Fumador_" + (p_miIngrediente+1));
    } //Faltan parámetros
    public void run(){
        System.out.println("\u001B[36m" +
            thr.getName() + "_llega_al_estanco." + "\u001B[0m");
        while(true)
        {
            estanco.obtenerIngrediente(miIngrediente); //coger el ingrediente
            try{
                //aux.dormir_max(2000);
                Thread.sleep(2000);
                System.out.println("\u001B[36m" +
                    thr.getName() + "_termina_de_fumar." + "\u001B[0m");
            }
            catch (InterruptedException e){
                System.err.println("Fumador_interrumpido");
            }
        }
    }
}

//hebra estanquero
class Estanquero implements Runnable {
    public Thread thr; //objeto hebra encapsulado
    private Estanco estanco = new Estanco();
    int ingrediente;
    //constructor
    public Estanquero(Estanco estanco){
        this.estanco=estanco;
        thr = new Thread(this, "Estanquero");
    }
    public void run(){
        System.out.println("\u001B[33m" +
            thr.getName() + "_abre_el_estanco." + "\u001B[0m");
        while (true){
            ingrediente = (int) (Math.random() * 3.0); //0,1,2
            estanco.ponerIngrediente(ingrediente); //poner el ingrediente en el mostrador

```



```
        estanco.esperarRecogidaIngrediente(); //esperar a que el fumador lo recoja
    }
}

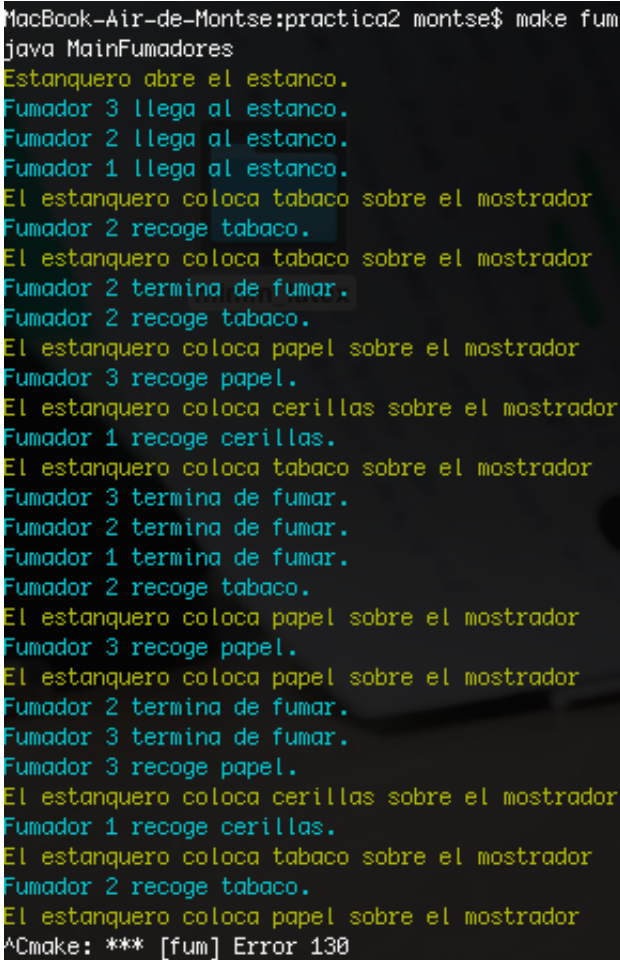
class MainFumadores
{
    public static void main( String [] args )
    {
        int num_fumadores = 3;
        Estanco estanco = new Estanco();
        Fumador[] f = new Fumador[num_fumadores];
            // crear hebras
        Estanquero e = new Estanquero(estanco) ;

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

        // poner en marcha las hebras
        e.thr.start();
        for(int i = 0; i < f.length; i++)
            f[i].thr.start();
    }
}
```

2.4 Resultado de la ejecución

En la figura (??) se puede ver la salida en terminal en el final de la ejecución del programa.



```
MacBook-Air-de-Montse:practica2 montse$ make fum
java MainFumadores
Estanquero abre el estanco.
Fumador 3 llega al estanco.
Fumador 2 llega al estanco.
Fumador 1 llega al estanco.
El estanquero coloca tabaco sobre el mostrador
Fumador 2 recoge tabaco.
El estanquero coloca tabaco sobre el mostrador
Fumador 2 termina de fumar.
Fumador 2 recoge tabaco.
El estanquero coloca papel sobre el mostrador
Fumador 3 recoge papel.
El estanquero coloca cerillas sobre el mostrador
Fumador 1 recoge cerillas.
El estanquero coloca tabaco sobre el mostrador
Fumador 3 termina de fumar.
Fumador 2 termina de fumar.
Fumador 1 termina de fumar.
Fumador 2 recoge tabaco.
El estanquero coloca papel sobre el mostrador
Fumador 3 recoge papel.
El estanquero coloca papel sobre el mostrador
Fumador 2 termina de fumar.
Fumador 3 termina de fumar.
Fumador 3 recoge papel.
El estanquero coloca cerillas sobre el mostrador
Fumador 1 recoge cerillas.
El estanquero coloca tabaco sobre el mostrador
Fumador 2 recoge tabaco.
El estanquero coloca papel sobre el mostrador
^Cmake: *** [fum] Error 130
```

Captura de pantalla 2.1: Ejemplo de ejecución del programa *fumadores.java*

Problema 3

El problema del barbero durmiente

3.1 Enunciado

El problema del barbero durmiente trata sobre una barbería en la que hay dos tipos de actores: un barbero y varios clientes.

La barbería consta de una sala para cortar el pelo y de una sala de espera. Si no hay clientes en la sala de espera, el barbero espera dormido a que llegue un cliente a la barbería. Si hay clientes en la sala de espera, entonces llama a uno de ellos.

El barbero pela al cliente durante un intervalo de tiempo, cuya duración la determina el barbero, avisa al cliente de que ha terminado de pelarlo y éste sale de la barbería durante un tiempo. Si hay clientes en la sala de espera, el barbero avisa al siguiente y si no hay nadie, duerme.

Este proceso se repite indefinidamente.

3.2 Descripción de los objetos condición usados y su propósito

Se utilizan tres objetos condición: *salaEspera*, *barbero* y *silla*:

1. *salaEspera*: Se utilizará en los métodos *cortarPelo* y *siguienteCliente*. En el primer método, cuando el cliente entra en la barbería, si encuentra la sala de espera (llama en la condición del *if* a *salaEspera.isEmpty()*) o la silla del barbero ocupadas, se sentará a esperar (*salaEspera.await()*). En otro caso, despertará al barbero si está dormido y se sentará en la silla para que le corten el pelo.

En el método *siguienteCliente* se usará para que el barbero permita pasar al siguiente cliente que esté esperando si la sala de espera está ocupada y para que duerma si no hay nadie ni en la silla ni en la sala de espera (*salaEspera.signal()*).

2. *barbero*: Se utilizará en los métodos *cortarPelo* y *siguienteCliente*. En el primer método se usará en la sentencia *barbero.signal()* para que el cliente que ha entrado en la sala de espera despierte al barbero si éste está dormido (comprobamos *barbero.isEmpty()*) y para despertarlo *barbero.signal()* y en el segundo para que el barbero duerma si no hay nadie en la silla ni en la sala de espera (*barbero.await()*).

3. *silla*: Se utilizará en los tres métodos. En *cortarPelo* servirá para comprobar si la silla está vacía en la condición de entrada en la sala para cortar el pelo (*silla.isEmpty()*), y para sentarse si es el turno de dicho cliente (*silla.await()*).

En *siguienteCliente* servirá para que el barbero controle si ya se ha ido el cliente que estaba ocupando la silla (*silla.isEmpty()*).

Por último, en el método *finCliente* esta condición liberará la silla cuando el cliente que se estaba cortando el pelo se levante para salir de la barbería un tiempo (*silla.signal()*).

3.3 Código fuente

```

import monitor.*;

class Barberia extends AbstractMonitor{
    private Condition salaEspera;
    private Condition barbero;
    private Condition silla;
    public Barberia(){
        salaEspera = makeCondition();
        barbero = makeCondition();
        silla = makeCondition();
    }
    //invocado por los clientes para cortarse el pelo
    public void cortarPelo(int i){
        enter();
        if(!silla.isEmpty() || !salaEspera.isEmpty()){ //si el barbero esta ocupado
            System.out.println("\u001B[32m" + "El cliente_" + i
                + " se sienta a esperar en la sala de espera." + "\u001B[0m");
            salaEspera.await(); //esperar en la sala de espera
        }
        //cliente liberado
        if(!barbero.isEmpty()){ //si el barbero esta durmiendo
            barbero.signal(); //liberar al barbero
        }
        System.out.println("\u001B[32m" + "Cliente_"
            + i + " entra a cortarse el pelo." + "\u001B[0m");
        silla.await(); //ocupar la silla

        leave();
    }
    //invocado por el barbero para esperar(si procede) a un nuevo cliente
    //y sentarlo para el cortar
    public void siguienteCliente(){
        enter();
        //si no hay nadie en la sala de espera ni en la silla, dormir
        if(salaEspera.isEmpty() && silla.isEmpty()){
            System.out.println("\u001B[33m" + "Parece que no hay nadie..." + "\u001B[0m");
            System.out.println("\u001B[33m" + "zZzzZzZZzzzZzZZ" + "\u001B[0m");
            barbero.await(); //zZzzZ
        }
        else if(!salaEspera.isEmpty()){
            System.out.println("\u001B[36m"
                + "¡Que pase el siguiente cliente!" + "\u001B[0m");
            salaEspera.signal(); //dejamos pasar al siguiente
        }

        leave();
    }
    //invocado por el barbero para indicar que ha terminado de cortar el pelo
    public void finCliente(){
        enter();
        System.out.println("\u001B[36m"
            + "Ya he terminado de afeitarse al cliente." + "\u001B[0m");
        silla.signal(); //liberamos la silla

        leave();
    }
}

```

```

class Cliente implements Runnable{
    public Thread thr;
    private int num_cliente;
    private Barberia barberia;
    public Cliente(Barberia barberia ,int num){
        num_cliente = num + 1;
        this.barberia = barberia;
        thr = new Thread(this,"cliente_" + num_cliente);
    }
    public void run(){
        while(true){
            System.out.println("\u001B[35m" + "El_" + thr.getName()
                + "_ha_entrado_en_la_barbería." + "\u001B[0m");
            barberia.cortarPelo(num_cliente); //el cliente espera(si procede) y se corta el pelo
            try{
                Thread.sleep(25000); //el cliente está fuera de la barbería un tiempo.
            }
            catch(InterruptedException e){
                System.err.println("Cliente_interrumpido");
            }
        }
    }
}

class Barbero implements Runnable{
    public Thread thr;
    private Barberia barberia;
    public Barbero(Barberia barberia){
        this.barberia = barberia;
        thr = new Thread(this, "barbero");
    }
    public void run(){
        while(true){
            try{
                Thread.sleep(2500); //el barbero esta cortando el pelo
                barberia.finCliente();
                barberia.siguienteCliente();
            }
            catch(InterruptedException e){
                System.err.println("Barbero_interrumpido");
            }
        }
    }
}

class MainBarberia
{
    public static void main( String[] args )
    {
        final int num_clientes = 5;
        Barberia barberia = new Barberia();
        Barbero b = new Barbero(barberia);
        Cliente[] c = new Cliente[num_clientes];
        for(int i= 0; i < num_clientes; i++){
            c[i] = new Cliente(barberia ,i);
        }
    }
}

```

```
// crear hebras

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

// poner en marcha las hebras
for(int i = 0; i < c.length; i++){
    c[i].thr.start();
}

b.thr.start();
}
```

3.4 Resultado de la ejecución

En la figura (??) se puede ver la salida en terminal en el final de la ejecución del programa. La figura (??) muestra una modificación del código para poder ver cómo el barbero duerme si no encuentra clientes.

```
MacBook-Air-de-Montse:practica2 montse$ make barb
javac -g ProductorConsumidor.java barberia.java fumadores.java
java MainBarberia
El cliente 2 ha entrado en la barbería.
El cliente 7 ha entrado en la barbería.
El cliente 6 ha entrado en la barbería.
El cliente 5 ha entrado en la barbería.
Cliente 2 entra a cortarse el pelo.
El cliente 4 ha entrado en la barbería.
El cliente 1 ha entrado en la barbería.
El cliente 3 ha entrado en la barbería.
El cliente 8 ha entrado en la barbería.
El cliente 9 ha entrado en la barbería.
El cliente 10 ha entrado en la barbería.
El cliente 5 se sienta a esperar en la sala de espera.
El cliente 9 se sienta a esperar en la sala de espera.
El cliente 8 se sienta a esperar en la sala de espera.
El cliente 3 se sienta a esperar en la sala de espera.
El cliente 1 se sienta a esperar en la sala de espera.
El cliente 4 se sienta a esperar en la sala de espera.
El cliente 7 se sienta a esperar en la sala de espera.
El cliente 6 se sienta a esperar en la sala de espera.
El cliente 10 se sienta a esperar en la sala de espera.
Ya he terminado de afeitarse al cliente.
¿Que pase el siguiente cliente!
Cliente 5 entra a cortarse el pelo.
Ya he terminado de afeitarse al cliente.
¿Que pase el siguiente cliente!
Cliente 9 entra a cortarse el pelo.
Ya he terminado de afeitarse al cliente.
¿Que pase el siguiente cliente!
Cliente 8 entra a cortarse el pelo.
Ya he terminado de afeitarse al cliente.
¿Que pase el siguiente cliente!
Cliente 3 entra a cortarse el pelo.
Ya he terminado de afeitarse al cliente.
¿Que pase el siguiente cliente!
Cliente 1 entra a cortarse el pelo.
Ya he terminado de afeitarse al cliente.
¿Que pase el siguiente cliente!
Cliente 4 entra a cortarse el pelo.
Ya he terminado de afeitarse al cliente.
¿Que pase el siguiente cliente!
Cliente 7 entra a cortarse el pelo.
^Cmake: *** [barb] Error 130
```

Captura de pantalla 3.1: Ejemplo de ejecución del programa *barberia.java* con 10 hebras


```

MacBook-Air-de-Montse:practica2 montse$ make barb
javac -g ProductorConsumidor.java barberia.java fumadores.java
java MainBarberia
El cliente 1 ha entrado en la barbería.
El cliente 2 ha entrado en la barbería.
Cliente 1 entra a cortarse el pelo.
El cliente 2 se sienta a esperar en la sala de espera.
Ya he terminado de afeitarse al cliente.
¡Que pase el siguiente cliente!
Cliente 2 entra a cortarse el pelo.
Ya he terminado de afeitarse al cliente.
Parece que no hay nadie...
zzzzzzzzzzzzzzzz
El cliente 1 ha entrado en la barbería.
Cliente 1 entra a cortarse el pelo.
El cliente 2 ha entrado en la barbería.
El cliente 2 se sienta a esperar en la sala de espera.
Ya he terminado de afeitarse al cliente.
¡Que pase el siguiente cliente!
Cliente 2 entra a cortarse el pelo.
Ya he terminado de afeitarse al cliente.
Parece que no hay nadie...
zzzzzzzzzzzzzzzz

```

Captura de pantalla 3.2: Ejemplo de ejecución del programa *barberia.java* con 2 hebras