Escuela Politécnica superior
Alcala de henares

Memoria Practica 1
Programación Avanzada

Pedro Barquin Ayuso

23-02-2014

Programa Buffer

En este programa se crea ciertas personas que haran el papel de emisores y un receptor de los mensaje que se van pasando y almacenando en el buffer.

- PruebaBuffer: Aquí es donde se definen todos los emisores con sus características como son nombre numero de cartas y destino, también se crea el buzón y el receptor de los mensaje.

```
package Buffer;
public class PruebaBuffer {
    public static void main(String[] s){
        Buffer bufferX = new Buffer();
        Productor pedro = new Productor("Pedro ",5,bufferX);
        Productor juan = new Productor("Juan ",4,bufferX);
        Productor antonio = new Productor("Antonio ",6,bufferX);
        Productor luis = new Productor("Luis ",7,bufferX);
        Consumidor jose = new Consumidor(21,bufferX);
    }
}
```

- Productor: En este apartado se crean los hilos productores con los datos de PruebaBuffer* también es donde se realiza la llamada que hace el envió del mensaje en función del numero de mensajes se realizan mas llamadas, ha de cumplir restricciones de tiempo.

```
public Productor(String prefijo, int n, Buffer buzon) {
    this.prefijo = prefijo;
    numMensajes = n;
    miBuffer = buzon;
    start();
}

public void run() {
    for (int i = 1; i <= numMensajes; i++) {
        try {
            sleep((int) (500 + 500 * Math.random()));
        } catch (InterruptedException e) {
        }
        try {
            miBuffer.enviaMensaje(prefijo + i);
            System.out.println("Envia" +prefijo + i);
        } catch (InterruptedException ex) {
            Logger.getLogger(Productor.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

- Consumidor: Es prácticamente igual salvo la llamada que realiza es la llamada para leer mensajes que esta en Buffer*.

```
public Consumidor(int numMensajes, Buffer miBuzon){
    this.numMensajes=numMensajes;
    this.miBuffer=miBuzon;
    start();
}

@Override
public void run(){
    for(int i=0; i<numMensajes; i++){
        try{ sleep((int)(500+500*Math.random()));} catch(InterruptedException e){}
        try {
            System.out.println("Recibe"+miBuffer.recibeMensaje());
        } catch (InterruptedException ex) {
            Logger.getLogger(Consumidor.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    System.out.println("Total mensajes leidos: "+numMensajes);
}
```

- Buffer: Aquí se encuentra el arraylist donde se almacenan los mensajes y los métodos que se llaman en Consumidor y en Productores* estos métodos almacenan los parámetros que reviven y los guardan en el arraylist o en el caso del recibeMensaje los borra del arraylist, también hay ciertas condiciones que ha de cumplirse o en caso contrario pone en espera a los Consumidor o Productores* dependiendo del caso.

```
ArrayList<String> mensaje = new ArrayList();

public synchronized void enviaMensaje(String msg) throws InterruptedException {

    while (mensaje.size() == 4) {       //Cuando se almacenan 4 mensajes manda espera
        wait();
    }
    mensaje.add(msg); //añade el mensaje a la cola
    System.out.println(mensaje);
    notifyAll();


}

public synchronized String recibeMensaje() throws InterruptedException {
    String msg;

    while (mensaje.size() ==0) {//cuando hay un mensaje o mas lo envia mientras esper
        wait();
    }
    notifyAll();

    msg = mensaje.get(0);   //almacena el mensaje para evitar que sea borrado antes d
    mensaje.remove(0);      //borra el mensaje de buzon

    return msg; //pasa el menasje al consumidor
```

Programa Bar

En el programa del bar utilizaremos se basa en la creación de 1000 hilos los cuales cumpliendo ciertas restricciones de tamaño y de tiempo se almacenan en un arraylist que es mostrado por pantalla según su posición y los que hay activos en ese momento.

- ElBar: En este apartado se crea la interface gráfica donde va sitiado el botón que mediante la llamada paso activa o desactiva un flag que hace que se detenga el proceso de transferencia, también es donde mediante un bucle for se crean los 1000 hilos persona, se crea el bar e incluye el método que enseña la lista por pantalla.

```
public class ElBar extends javax.swing.JFrame {

    private Paso paso = new Paso();
    private boolean botonPulsado1 = false;
        .
    /**
     * Creates new form ElBar
     */
    public ElBar() {
        initComponents();

        Bar bar = new Bar();

        //crea las 1000 personas
        for (int i = 1000; i < 1999; i++) {
            Persona persona = new Persona(i, paso, bar);
        }
    }

    //muestra la lista
   public static void lista(String a){
    jTextField1.setText(a);
   }
```

- Bar: En este apartado se encuentra el arraylist donde se almacena toda la información de la cola de personas que hay en el bar, también están los métodos entraBar que que manda guardar en el arrylist la información de la persona que a entrado al bar tiene la restricción de que si hay ya 10 personas no deja entrar mas y pausa el resto de hilos que quieren entrar, el otro método sirve para que una vez se han cumplido las restricciones se puede sacar del arraylist a esa persona con este método

```
public synchronized void entraBar(int id) throws InterruptedException {
    String a = Integer.toString(id);
    while (capacidad.size() == 10) {       //Cuando se almacenan 10 personas esperar
        wait();
    }
    capacidad.add(a); //añade el mensaje a la cola
    b=capacidad.toString(); //pasa el arrylist
    ElBar.lista(b);  //manda el texto
    notifyAll();
}


public synchronized void saleBar (int id) throws InterruptedException {
    String a = Integer.toString(id);

    capacidad.remove(a);        //borra de la lista
     if(capacidad.size()<10 && capacidad.size()>1){
         b=capacidad.toString();
         ElBar.lista(b);        //manda el texto  cuando hay menos de 10
    }
    if(capacidad.size()==0){
        ElBar.lista("Vacio");   //manda el texto al terminar
    }
    notifyAll();
```

- Paso: Este apartado es de vital importancia ya que gracias a al método mirar que es invocado desde persona* que permite ver como se encuentra paso permitiendo de esta forma parar o reanudar el proceso, también esta el proceso abrir que pone paso en continuar para que cuando sea visto deje seguir el programa, mientras que cerrar pone paso a false para que cuando sea visto pause la ejecución.

```
package Bar;
public class Paso {
    private boolean cerrado;
    private Bar bar = new Bar();

    public synchronized void mirar(){
        while(cerrado){
            try{wait();} catch(InterruptedException ie){}
        }
    }
    public synchronized void abrir(){       //abre paso
        cerrado=false;
        notifyAll();
    }
    public synchronized void cerrar(){      //cierra paso
        cerrado=true;
        notifyAll();
    }
}
```

- Persona: Aquí se encuentra el constructor de persona que se a mencionado en ElBar* para que sean creados correctamente todos los hilos, acontinuación esta el run de la persona creada donde se tiene que cumplir ciertos periodos de tiempo como la espera de (0,2seg-0,5seg) para poder entrar antes de hacer las llamadas a los métodos entrar y salir de Bar* también están las llamadas a mirar de Paso*.

```
public Persona(int numPersona, Paso paso, Bar local){
    this.numPersona=numPersona;
    this.paso=paso;
    this.miBar=local;
    start();

}

//proceso que sigue cada persona de las 1000
public void run(){
    //espera entre 0,2 y 0,5 seg
    try {
            sleep((int) (200 + 300 * Math.random()));
        } catch (InterruptedException e) {
        }
    //mira la parada
    paso.mirar();
    //Entra al bar
 try {
        this.miBar.entraBar(numPersona); //llama entrada
    } catch (InterruptedException ex) {
        Logger.getLogger(Persona.class.getName()).log(Level.SEVERE, null, ex);
    }
    //mira la parada
    paso.mirar();
    // Se toma algo en el bar 0.5 1 seg
    try {
            sleep((int) (500 + 1000 * Math.random()));
        } catch (InterruptedException e) {
        }
    //sale del bar
    try {
        this.miBar.saleBar(numPersona);  //llama salida
    } catch (InterruptedException ex) {
        Logger.getLogger(Persona.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```