Escuela Politécnica superior
Alcala de henares

Memoria Practica 2
Programación Avanzada

Pedro Barquin Ayuso

09-03-2014

Objetivo:

Con estos ejercicios se basa en aprender a usar y manejar programas concurrentes y aprender que son las regiones criticas , abarcando las distintas las posibilidades que nos ofrece java para su programación usando: Monitores Semáforos o Lock.

Bases de los ejercicios:

Los 3 tipos de ejercicios tienen la misma estructura con unos ligeros cambios entre un tipo y otro, el programa crea 10 hilos que serán los escritores de el libro y 15 lectores, cada grupo cumple unas restricciones de tiempo distintas, en el caso de los escritores tienen prioridad a la hora de pedir permiso y solo puede haber uno activo, en el caso de los lectores puede haber varios a la vez en ejecución. La finalidad es que los 10 escritores completen el libro poniendo 5 veces su nombre mientras que los lectores van leyendo el contenido siempre que se lo permitan hasta que se complete el libro.

El apartado Paso Escritor Lector y Biblioteca descritos a continuación son iguales en los tres modelos de ejercicios.

- Paso: Este apartado es de vital importancia ya que gracias a al método mirar que es invocado desde (escritor o lector)* 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.

```
public class Paso {
    private boolean cerrado;


    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();
    }
}
```

- Biblioteca: Aquí es donde se encuentra la interfaces gráfica del programa y también es la parte mas importante de todas ya que es donde se crean los hilos del programa los 15 lectores y los 10 escritores(pasando les su numero para que realicen la operaciones posteriores). También incluye los métodos que son usados por los escritores/lectores para pasar los datos a la interface y que sean mostrados por pantalla, otra parte es donde se define el comportamiento que tienen los botones con el programa en nuestro caso para que paren la ejecución con ayuda de paso*.

```
public class Biblioteca extends javax.swing.JFrame {
    private Paso paso = new Paso();
    private boolean botonPulsado1 = false; // cuando se incluya boton

    /**
     * Creates new form Biblioteca
     */
    public Biblioteca() {
        initComponents();
        Libro libro=new Libro();
        for (int i = 0; i <= 9; i++) {   //crea escritores
            Escritor escritor = new Escritor(i, paso, libro);
        }       //crear los 10 escritores
        for (int a = 20; a <= 34; a++) { //crear los 15 lectores
            Lector lector = new Lector(a, paso, libro);
        }
    }
```

```
}
public static void listaLectores(String a){
jTextField1.setText(a); //lista lectores
}
public static void listaEscritores(String b)
jTextField2.setText(b); //lista escritor
}
public static void listaLibro(String c){
jTextField3.setText(c); //lista libro
}
```

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
if (!botonPulsado1) {                //Si no se ha pulsado
        botonPulsado1 = true;
        jButton1.setText("Reanudar");//lo cambiamos a pulsado
        paso.cerrar();//Cerramos el paso para que los pintores se detengan
    } else {                         //Si ya se había pulsado
        botonPulsado1 = false;
        jButton1.setText("Detener");//lo cambiamos
        paso.abrir();    //Abrimos el paso para que los pintores sigan traba
    }       // TODO add your handling code here:
}
```

- Escritores: Los los 10 escritores son creados desde biblioteca* recibiendo el numero que usaran para pasar al libro en el que escriben, este paso se va a repetir el proceso 5 veces ya que así es como se especifica en el ejercicio, el proceso es esperan durante 1 u 2 segundos, llaman a escribir en el libro, esperan otra vez y llaman para terminar de escribir en el libro, este paso se realiza con unas condiciones que se describirán en el apartado libro* quedando el escritor a la espera cuando no se cumplan dichas condiciones de programación como son que no puede haber mas de un escritor activo y que el libro tiene que estas sin usar por los lectores teniendo prioridad el escritor en todo momento. Las 2 partes están estructuradas de diferente forma pero el funcionamiento es el mismo. En caso de lector se encuentra de una forma mas limpia y ordenada.

```
public Escritor(int numEscritor, Paso paso, Libro libro){      //Constructor
    this.numEscritor=numEscritor;
    this.paso=paso;
    this.libro=libro;
    start();
}

@Override
public void run(){ //mirar paso antes de escribir  cuando pide escribir para el
    /*try {
         paso.mirar();
         sleep((int) (1000 + 1000 * Math.random()));
         paso.mirar();
         this.libro.escribir(numEscritor); //llama escribir
         sleep((int) (1000 + 1000 * Math.random()));
         paso.mirar();
         this.libro.deja_escribir(); //llama deja-escribir
    } catch (Exception e) {
    }*/
     for (int i = 0; i < 5; i++) { //escribe durante 5 veces
        paso.mirar();
        try {   //espera de 1 a 2 seg
            sleep((int) (1000 + 1000 * Math.random()));
        } catch (InterruptedException e) {
        }
        paso.mirar();
        try {
        this.libro.escribir(numEscritor); //llama escribir
    } catch (InterruptedException ex) {
        Logger.getLogger(Escritor.class.getName()).log(Level.SEVERE, null, ex);
    }
        paso.mirar();
        try {   //espera de 1 a 2 seg
            sleep((int) (1000 + 1000 * Math.random()));
        } catch (InterruptedException e) {
        }
        paso.mirar();
        try {
        this.libro.deja_escribir(); //llama deja-escribir
    } catch (InterruptedException ex) {
        Logger.getLogger(Escritor.class.getName()).log(Level.SEVERE, null, ex);
    }
```

- Lectores: Al igual que los escritores son creados de biblioteca* y reciben el numero con el cual operan en la clase libro*, en este caso el proceso se repite hasta que se cumpla la condición de que el libro esta completo (50 datos), siguiendo este esquema. Espera durante 1 u 2 segundos, acto seguido manda leer el libro y espera durante 0,5 o 1,5 seg para después dejar de leer. Este paso se realiza con unas condiciones que se describirán en el apartado libro* quedando el lector a la espera cuando no se cumplan dichas condiciones de programación como son en este caso que puede haber varios lectores pero la prioridad la tienen los escritores.

```
public Lector(int numescritor, Paso paso, Libro libro){
    this.numescritor=numescritor;
    this.paso=paso;
    this.libro=libro;
    start();
}


@Override
    public void run(){ //mirar paso antes de escribir  cu
    try {
        do{
        paso.mirar();
        sleep((int) (1000 + 2000 * Math.random()));
        paso.mirar();
        this.libro.toma_leer(numescritor);
        paso.mirar();
        sleep((int) (500 + 1500 * Math.random()));
        paso.mirar();
        this.libro.deja_leer(numescritor);
    } while(libro.libro.size()<=49);
    } catch (Exception e) {
    }
```

Monitor:

En esta parte utilizaremos Wait y notifyAll para despertar o dormir los hilos, usamos dos arraylist en los cuales se almacenan los lectores que hay y lo que hay escrito en el libro hasta el momento, también contamos con una variable leyendo para saber en todo momento cuantos lectores hay en ejecución , la parte mas importante es el booleano escribiendo ya que con el se gestiona que no puedan entrar los lectores cuando quiere escribir alguien.

En la parte que entra el escritor hay dos bloqueos para que en caso de que no sea el primer escritor que hay se bloquee y el otro caso es que pare si aun hay gente leyendo el libro, si pasa el primer bloque cambia escribiendo para que no puedan entrar mas escritores, luego ya realiza los paso descritos como son añadirse al libro poner que esta escribiendo y cuando termina despertar a los dormidos en la sección dejar escribir.

En el caso de toma leer no hay restricciones de cantidad en ejecución solo que cuando escribiendo esta activado no pueden entrar mas lectores hasta que termine, por lo demás manda la información de la lista de lectores al array para mostrarlo y aumenta el contador de lectores.
En la parte de deja leer extrae su nombre de la lista de lectores mandándola actualizada y resta en 1 el contador de lectores activos para después despertar a al resto de dormidos.

```
public class Libro {
    private int leyendo=0;
    private boolean escribiendo=false;
    private String b;
    protected ArrayList<String> libro = new ArrayList();
    protected ArrayList<String> lectores = new ArrayList();

    public synchronized void escribir(int id) throws InterruptedException {
        while (escribiendo==true) {       //Si hay un escritor activo bloquea al
            wait();
        }
        escribiendo=true;           //activa el flag de escribiendo
        while (leyendo>=1) {       //Cuando esten leyendo alguien espera hasta q
            wait();
        }
        String a = Integer.toString(id);
        libro.add(a); //añade el mensaje al libro
        Biblioteca.listaEscritores(a);  //manda el escritor para que se muestre
        b=libro.toString();  //pasa datos a otro tipo
        Biblioteca.listaLibro(b);//envia el contenido del libro para mostrarlo
    }

    public synchronized void deja_escribir() throws InterruptedException {
        Biblioteca.listaEscritores("Vacio");    //muestra vacio
        escribiendo=false; //cuando termina pone el escritor a falso
        notifyAll();    //abre los hilos

    }

    public synchronized void toma_leer(int id) throws InterruptedException {
        while (escribiendo==true) {       //Cuando esten escribiendo manda esper
            wait();
        }
        leyendo++;      //aumenta el contador de los lectores
        String a = Integer.toString(id);
        lectores.add(a);    //añade lector a la lista
        b=lectores.toString();  //pasa datos
        Biblioteca.listaLectores(b);    //muestra lectores por pantalla
    }

    public synchronized void deja_leer(int id) throws InterruptedException {
        String a = Integer.toString(id);
        lectores.remove(a);     //borra el lector
        b=lectores.toString();  //pasa datos
        Biblioteca.listaLectores(b);    //muestra lectores por pantalla
        leyendo--;
        if(leyendo==0){
          notifyAll();
```

Semáforos:

En esta parte utilizaremos semaforo.acquire y semaforo1.release para gestionar los accesos la gran parte del código es similar al descrito en el apartado monitor salvo lo descrito acontinuacion. En este caso usamos los semáforos para controlar los hilos que aceden a los recursos cumpliendo a si con los requisitos de la practica, para ello es necesaria la construcción de un objeto semáforo que tiene 15 permisos para entregar a los usuarios según el tipo, en el caso de escribir el escritor adquiere 15 permisos de golpe (semaforo1.acquire(15)) para que no puedan entrar los lectores ni ningún otro escritor, en el apartado de dejar de escribir da los permisos (semaforo1.release(15)) para continuar con el programa. En la parte del lector los permisos son cogidos de 1 en uno (semaforo1.release()) por los lectores para mas tarde devolverse de uno en uno también el el método dejar de leer.

```
private Semaphore semaforo2;
public Libro(Semaphore semaforo, Semaphore semaforoo) {
    semaforo1 = semaforo;
    semaforo2 = semaforoo;
}
//probar con un semaforo para cada uno
public  void escribir(int id) throws InterruptedException {

    try {
        semaforo1.acquire(15);
    } catch (InterruptedException ex) {}
    String a = Integer.toString(id);
    libro.add(a); //añade el mensaje al libro
    Biblioteca.listaEscritores(a);  //manda el escritor para que se mues
    b=libro.toString();  //pasa datos a otro tipo
    Biblioteca.listaLibro(b);//envia el contenido del libro para mostrar
}

public  void deja_escribir() throws InterruptedException {
    Biblioteca.listaEscritores("Vacio");    //muestra vacio
    semaforo1.release(15);    //abre los hilos
}

public void toma_leer(int id) throws InterruptedException {

    try {
        semaforo1.acquire();
    } catch (InterruptedException ex) {}
    leyendo++;      //aumenta el contador de los lectores
    String a = Integer.toString(id);
    lectores.add(a);    //añade lector a la lista
    b=lectores.toString();  //pasa datos
    Biblioteca.listaLectores(b);    //muestra lectores por pantalla
}

public  void deja_leer(int id) throws InterruptedException {
    try {
        String a = Integer.toString(id);
        lectores.remove(a);     //borra el lector
        b=lectores.toString();  //pasa datos
        Biblioteca.listaLectores(b);    //muestra lectores por pantalla
        leyendo--;
        if(leyendo==0){
                Biblioteca.listaLectores("Vacio");
            }
    } finally {
        semaforo1.release();
    }
```

Lock:

En esta parte utilizaremos lock y unlock para gestionar los accesos, la base de lo que hacen los métodos es la misma que en el caso de los monitores salvo lo mencionado acontinuacion. En este método hay que definir los lock que vamos a usar para parar o liberar los procesos, en este caso creamos (lock r) para leer y (lock w) para escribir, en los métodos escribir y toma leer se activan los lock de leer y escribir respetando las restricciones que nos impone el programa como son que no haya mas de un lector y que no pueden estar a la vez los dos teniendo prioridad el escritor. En los métodos deja de leer y deja de escribir se activa el unlock que deja libres los permisos para que continué la ejecución.

```
protected ArrayList<String> lectores = new ArrayList();
private ReadWriteLock lock = new ReentrantReadWriteLock();
private Lock r = lock.readLock();
private Lock w = lock.writeLock();

public void escribir(int id) throws InterruptedException{
    w.lock();
    String a = Integer.toString(id);
    libro.add(a); //añade el mensaje al libro
    System.out.println(".........................");
    Biblioteca.listaEscritores(a);  //manda el escritor para que se mues
    b=libro.toString();  //pasa datos a otro tipo
    Biblioteca.listaLibro(b);//envia el contenido del libro para mostrar
}

public void deja_escribir()  throws InterruptedException {
    try{
        Biblioteca.listaEscritores("Vacio");    //muestra vacio
    escribiendo=false; //cuando termina pone el escritor a falso
    } finally {
        w.unlock();
    }
}

public void toma_leer(int id){
    r.lock();
    leyendo++;      //aumenta el contador de los lectores
    String a = Integer.toString(id);
    lectores.add(a);    //añade lector a la lista
    System.out.println("Se añade "+a);
    b=lectores.toString();  //pasa datos
    Biblioteca.listaLectores(b);    //muestra lectores por pantalla
}

public void deja_leer(int id) {
    try{
        String a = Integer.toString(id);
        System.out.println("Se borra "+a);
        lectores.remove(a);     //borra el lector
        b=lectores.toString();  //pasa datos
        Biblioteca.listaLectores(b);    //muestra lectores por pantalla
        leyendo--;
        if(leyendo<=1){
            Biblioteca.listaLectores("Vacio");
            r.unlock();
        }
    } finally {r.unlock();}
```

Conclusiones:

La conclusión de este ejercicio es aprender a usar los tipos diferentes que hay en java para la programación con ocurrente como son semáforo lock y monitores.

- Ejemplo de como queda el programa final: