Programación Concurrente y de Tiempo Real
Grado en Ingeniería Informática
Asignación de Prácticas Número 8

En esta asignación aplicará control de exclusión mutua y sincronización, utilizando para ello el API estándar de Java, efecuando la implementación con monitores a soluciones de problemas clásicos de la concurrencia. **Documente todo su código con etiquetas (será sometido a análisis con `javadoc`).**

1. Enunciado

Otro problema clásico de la concurrencia es el de los lectores-escritores, donde hebras lectoras y escritoras acceden a un recurso común para efectuar su tarea. Como es lógico, se permiten lecturas concurrentes, si bien las escrituras concurrentes están prohibidas, y las lecturas y escrituras concurrentes, también. Sigue a continuación una especificación en pseudocódigo de un monitor para resolver el problema, que utiliza variables de condición, etc.:

```
monitor LE;
  var
  lectores: integer; (*numero de lectores leyendo concurrentemente*)
  lector, escritor: condition;
  escribiendo: boolean (*indica si hay un escritor activo*)

  procedure iniciaLectura;
  begin
    if(escribiendo) then wait(lector);
    lectores++;
    send(lector)
  end;

  procedure acabarLectura;
  begin
    lectores--;
    if(lectores=0) then send(escritor)
  end;

  procedure iniciarEscritura;
  begin
    if(lectores!=0) or (escribiendo) then wait(escritor);
```

```
    escribiendo=true;
  end;

  procedure acabarEscritura;
  begin
    escribiendo=false;
    if(empty(lector)) then send(escritor);
    else send(lector);
  end;

begin (*codigo de inicialización del monitor*)
  lectores=0;
  escribiendo=false;
end.
```

A continuación:

- desarrolle una clase `recurso.java` que encapsule una variable `n` de tipo `long` que inicialmente vale cero; la clase dispondrá de métodos para incrementar (`inc()`) y leer (`observer()`) la variable encapsulada.
- escriba ahora utilizando el API estándar de Java (`wait()+notifyAll()`) un monitor a partir del pseudocódigo ilustrado anteriormente, y guárdelo en `lectorEscritor.java`. Asegúrese de proveer condiciones de guarda donde sea necesario.
- ahora escriba un diseño concurrente de varias hebras lectoras y escritoras en `usalectorEscritor.java` que accedan a un objeto de clase `recurso` a través de los métodos del monitor que controlan el acceso al mismo, con unas estructuras de ciclo de vida como las siguientes, para las hebras lectoras y escritoras (nota: `le` es la referencia al monitor de clase `lectorEscritor` que controla los accesos, y `r` es la referencia al recurso común de clase `recurso`):

```
//tarea lectora...
for(long i=0; i<1000000; i++){
    le.iniciarLectura();
    data=r.observer();
    le.acabarLectura();
  }

//tarea escritora...
for(long i=0; i<1000000; i++){
    le.iniciarEscritura();
    r.inc();
    le.acabarEscritura();
  }
```

- finalmente, compruebe que la ejecución de múltiples hebras lectoras y escritoras preserva el valor final esperado sobre la variable `n` encapsulada en el objeto de clase `recurso`.

2. Procedimiento de Entrega

PRODUCTOS A ENTREGAR

- Ejercicio 1: `recurso.java`, `lectorEscritor.java` y `usalectorEscritor.java`.

MÉTODO DE ENTREGA: Tarea de Moodle.