Programación Concurrente y de Tiempo Real
Grado en Ingeniería Informática
Asignación de Prácticas Número 7

En esta asignación aplicará control de exclusión mutua y sincronización, utilizando para ello el API estándar de Java, a diferentes situaciones que se le plantean, mediante el uso de monitores. Documente todo su código con etiquetas (será sometido a análisis con `javadoc`). Si lo desea, puede también agrupar su código en un paquete de clases, aunque no es obligatorio.

1. Enunciados

1. En el Tema 3 del curso teórico se ha analizado el concepto de monitor, y se ha el problema del productor-consumidor; a partir de aquí:

- repase el funcionamiento del monitor productor-consumidor desde un punto de vista teórico, utilizando la descripción del texto de Ben-Ari, disponible en la carpeta de la práctica.
- la carpeta de la práctica contiene también la propuesta de Ben-Ari para un monitor (API estándar) que da solución en Java al productor-consumidor; preste especial atención a las diferencias entre el modelo teórico del monitor productor-consumidor, que utiliza dos variables de condición para sincronizar, y su implementación en Java, que debe lograr la sincronización con un único *wait-set*; descargue y compile.
- escriba un diseño de hebras productoras y consumidoras que utilicen este monitor, y guárdelo en `usaProdCon.java`.
- reduzca el tamaño del buffer a uno y vea qué ocurre.
- ejecute un productor y varios consumidores y vea qué ocurre.
- ejecute varios productores y un consumidor y vea qué ocurre.
- redacte un documento `analisis.pdf` con sus impresiones acerca de las diferentes pruebas de ejecución propuestas.

2. La tripulación de un *drakkar* vikingo comparte una marmita con un almuerzo a base de m anguilas. Cuando un vikingo quiere comer, se sirve una anguila. Si ya no quedan, avisa al vikingo cocinero para que proceda a llenar la marmita de nuevo, utilizando las inagotables provisiones de anguilas disponibles a bordo del navío. Ahora:

- Escriba un monitor teórico en pseudocódigo, con tantas variables de condición como necesite que modele el problema, y guárdelo en `drakkarTeorico.pdf`.
- Desarrolle ahora un monitor en Java que modele esta curiosa situación a partir del *wait-set*, y provea la sincronización necesaria utilizando métodos `synchronized` y notificación `wait()-notifyAll()`. Guarde el monitor en `drakkarVikingo.java`.
- Finalmente, escriba un programa que crea las tareas (vikingos) y usa el monitor en `usaDrakkarVikingo.java`

3. El problema de los lectores-escritores es otro problema clásico de la concurrencia. En él, hebras lectoras y escritoras tratan de acceder a un recurso común bajo los siguientes criterios de control:

- Los lectores pueden acceder al recurso siempre que no haya un escritor escribiendo en él.
- Los escritores deben esperar a que no haya ni otro escritor, ni lectores, accediendo al recurso.
- En la carpeta de la práctica dispone de una descripción del monitor teórico para resolver el problema, tomado del texto de Ben-Ari, en su segunda edición. Se le pide que programe una solución equivalente en Java utilizando un monitor soportado por el API estándar de control de la concurrencia. Guarde el monitor en `lectorEscritor.java`. Escriba también un diseño de hebras lectoras y escritoras en `usalectorEscritor.java` que emplee el monitor.

2. Procedimiento de Entrega

PRODUCTOS A ENTREGAR

- Ejercicio 1: `usaProdCon.java` y `analisis.pdf`.
- Ejercicio 2: `drakkarTeorico.pdf`, `drakkarVikingo.java` y `usaDrakkarVikingo.java`,
- Ejercicio 3: `lectorEscritor.java` y `usalectorEscritor.java`.

MÉTODO DE ENTREGA: Tarea de Moodle.