

# Programación Concurrente y de Tiempo Real

## Grado en Ingeniería Informática

### Asignación de Prácticas Número 6

Se le plantean a continuación un conjunto de ejercicios sencillos de programación de control de la exclusión mutua y uso de ejecutores, que debe resolver de forma individual como complemento a la sexta sesión práctica. Para cada uno, debe desarrollar un programa independiente que lo resuelva. Documente todo su código con etiquetas (será sometido a análisis con `java-doc`). Si lo desea, puede también agrupar su código en un paquete de clases, aunque no es obligatorio.

## 1. Ejercicios

1. En la carpeta de la práctica se le proporciona el código necesario para implantar una solución cliente-servidor simple multihebrada. El inconveniente de esta solución es la necesidad de crear un *thread* para dar servicio a cada petición de un cliente recibida por el servidor. Una solución mucho más elegante es modelar la tarea de servicio mediante objetos `Runnable`, y delegar su ejecución a un *pool* de *threads*. Se pide:

- Reescriba el código del servidor de forma que cada petición de servicio sea atendida por una hebra que procesará un objeto de clase `ThreadPoolExecutor`, y guárdelo en `ServidorHiloconPool.java`.
- Escriba ahora una versión del cliente que genere un número fijo de peticiones al servidor, y guárdela en `clienteMultiple.java`.

2. Deseamos que varias hebras soportadas mediante herencia de la clase `Thread` escriban datos en un objeto de array de forma segura. Provea una solución llamada `arrSeguro.java` que cumpla con la especificación descrita,

utilizando cerrojos `synchronized`.

3. Escriba una clase en `heterogenea.java` que tendrá dos atributos  $n, m$  a incrementar mediante **métodos diferentes**. Proteja los métodos que gestionan a  $n$  mediante `synchronized`, mientras que los métodos que gestionan a  $m$  no tendrán control alguno. A continuación, escriba en `usaheterogenea.java` un programa donde múltiples hebras accedan a un objeto de clase `heterogenea` concurrentemente, y compruebe, estudiando los valores finales de  $n$  y  $m$ , que en el mismo objeto de Java pueden existir regiones de código bajo exclusión mutua y sin ella.

4. Escriba, utilizando `synchronized`, un código donde tres hebras diferentes (soportadas por herencia de `Thread`) entren en *deadlock*. Guarde el código en `deadlock.java`

5. Reescriba una versión de la integración paralela de *Monte-Carlo* para aproximar la integral definida de la función  $f(x) = \cos(x)$  en  $[0, 1]$  utilizando tareas `Callable`. Guarde el código en `integCallable.java`

6. Rescate ahora todas las condiciones de concurso no controladas que escribió en la asignación número 2, y transfórmelas en versiones concurrentemente seguras utilizando cerrojos `synchronized`. Guarde las nuevas versiones en ficheros con el mismo nombre que ya utilizó para esa asignación, añadiendo a esos nombres la palabra «seguro».

## 2. Procedimiento de Entrega

PRODUCTOS A ENTREGAR:

- Ejercicio 1: `ServidorHiloconPool.java` y `clienteMultiple.java`.
- Ejercicio 2: `arrSeguro.java`.
- Ejercicio 3: `heterogenea.java` y `usaheterogenea.java`.
- Ejercicio 4: `deadlock.java`.
- Ejercicio 5: `integCallable.java`
- Ejercicio 6: ficheros de condición de concurso de la asignación 2, modificados para integrar control de concurrencia.

MÉTODO DE ENTREGA: Tarea de Moodle.