Exclusión mutua en Java

Índice de contenido

1. Requisitos software

- Eclipse http://www.eclipse.org/downloads/
- VisualVM incluida en jdk versión 6 o superiores

2. Objetivos específicos

- Aplicar mecanismos dinámicos de sincronización para garantizar la exclusión mutua
- Conocer mecanismos de preservación de invariantes de los objetos compartidos en programación concurrente para evitar estados inconsistentes
- Monitorizar la ejecución de programas concurrentes en Java analizando los estados de los hilos (estado *sleeping, monitor, running*)
- Uso de colecciones y sus recorridos en entornos concurrentes
- Aplicación del patrón Singleton en colecciones concurrentes

3. Enunciado

En la práctica se presentan dos casos de estudio de programación concurrente en Java que describen situaciones de exclusión mutua.

(a) Caso de estudio 1: Simulación de entradas a un parque de dos entradas

Desarrollar una aplicación Java, en un paquete llamado `p03.c01`, que simule el siguiente comportamiento funcional. La Ilustración 1 muestra, de una manera esquemática, un parque con dos entradas (que se identifican con A y B). Por estas entradas, los visitantes pueden acceder al mismo y los responsables quieren saber en todo momento cuántas personas han entrado (no importan las salidas, o se puede suponer que nadie de los que entra sale posteriormente).

Se trata de escribir un programa que simule el comportamiento del sistema durante un periodo determinado de ejecución hasta que se alcancen 20 entradas por cada una de las dos puertas. El programa debe indicar, para cada entrada, el instante de tiempo en que se produce dicha entrada en el parque y la puerta por la que entra. Por otro lado debe informar del tiempo medio transcurrido desde que cada visitante entró en el parque. Para llevar a cabo la simulación, deben lanzarse dos tareas, una por cada entrada.

Ilustración 1: Descripción gráfica del problema

Debe tenerse en cuenta:

- La exclusión mutua se implementará mediante mecanismos dinámicos basados en sincronizaciones que garantizan que sólo un hilo pueda acceder al estado de un objeto (`synchronized`).
- Para manejo del tiempo se puede usar la funcionalidad `System.currentTimeMillis()`.
- Define explícitamente los invariantes utilizando los `assert` de Java
- Para simular la generación de un tiempo aleatorio de dos entradas consecutivas se puede utilizar la clase `java.util.Random;`

En la Tabla 1se muestra el resultado de una posible salida de ejecución.

```
Entrada al parque por puerta A
--> Personas en el parque 1 tiempo medio de estancia: 0.797
----> Por puerta A 1
----> Por puerta B 0

Entrada al parque por puerta B
--> Personas en el parque 2 tiempo medio de estancia: 1.715
----> Por puerta A 1
----> Por puerta B 1

Entrada al parque por puerta A
--> Personas en el parque 3 tiempo medio de estancia: 2.867
----> Por puerta A 2
----> Por puerta B 1

…
Finalizada entrada por la puerta A
Entrada al parque por puerta B
--> Personas en el parque 40 tiempo medio de estancia: 55.807791788701245
----> Por puerta A 20
----> Por puerta B 20
```

Tabla 1: Resultado de una posible salida de ejecución

- **Preguntas:** Ejecuta el programa varias veces y analiza el resultado de cada ejecución. ¿Son las trazas de ejecución siempre iguales? ¿Cuál es invariante del parque que se intenta preservar (usar `assert` de Java)? ¿Si se elimina la sincronización se cumple el invariante (ejecutar con las aserciones activadas -ea)?

(b) Caso de estudio 2: Simulación de entradas a un parque de múltiples entradas

Desarrollar una aplicación Java, en un paquete llamado `p03.c02,` que simule el siguiente comportamiento funcional. Se plantea como ejercicio generalizar el programa de simulación del caso de estudio anterior de forma que el número de puertas sea un entero N mayor que 2, identificadas por letras mayúsculas consecutivas a partir de la A. La Ilustración 2 muestra, de una manera esquemática, dos parques, uno con cuatro entradas y otro con cinco (que se identifican con A, B, C, D y E).

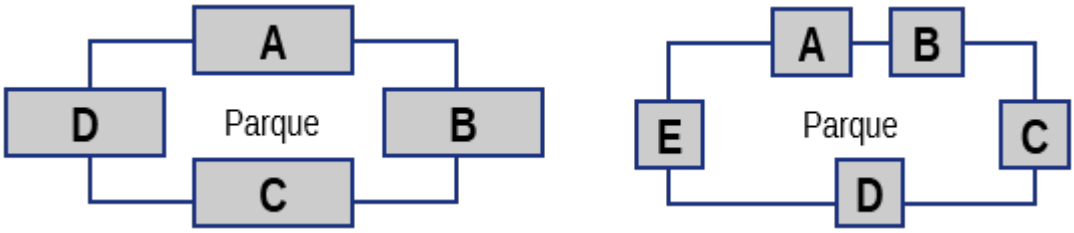

Ilustración 2: Descripción gráfica del problema del parque genérico

Para llevar a cabo la simulación, deben lanzarse N + 1 `Thread`, uno por cada entrada y otro adicional para gestionar la simulación. El valor del número N de entradas se declarará como una constante en el código Java.

En la Tabla 2se muestra el resultado de una posible salida de ejecución.

```
Entrada al parque por puerta A
--> Personas en el parque 1 tiempo medio de estancia: 0.6745
----> Por puerta D 0
----> Por puerta A 1

Entrada al parque por puerta D
--> Personas en el parque 2 tiempo medio de estancia: 2.21925
----> Por puerta C 0
----> Por puerta D 1
----> Por puerta A 1

Entrada al parque por puerta C
--> Personas en el parque 3 tiempo medio de estancia: 3.080125
----> Por puerta C 1
----> Por puerta D 1
----> Por puerta A 1
----> Por puerta B 0

Entrada al parque por puerta B
--> Personas en el parque 4 tiempo medio de estancia: 4.7040625
----> Por puerta C 1
----> Por puerta D 1
----> Por puerta A 1
----> Por puerta B 1
...
```

Tabla 2: Resultado de una posible salida de ejecución con N = 4

- **Preguntas:** Ejecuta el programa varias veces el programa, con diferentes valores de N, y analiza el resultado de cada ejecución. ¿Son las trazas de ejecución siempre iguales? ¿Cuál es invariante del parque que se intenta preservar? ¿Si se elimina la sincronización se cumple el invariante?

4. Productos a entregar

En esta práctica no es obligatorio entregar nada al profesor, pero es interesante documentar la experiencia, con la siguiente información de los casos de estudio

- Descripción del caso
- Código fuente en Java

- Conjunto de salidas de ejecución del programa Java en distintos instantes de tiempo
- Conjunto de *snapshots* de monitorización de la ejecución obtenidos con VisualVM
- Respuesta a las preguntas de reflexión sobre el caso de estudio.

Puedes aplicar múltiples diseños concurrentes basados en los conceptos de teoría: java.util.concurrent.locks.ReentrantLock , aplicación patrón de diseño singleton sobre el recurso compartido por los hilos, uso de colecciones concurrentes, confinamiento de objetos basados en adaptadores concurrentes.
La práctica será evaluada mediante un cuestionario presencial que se realizará en el laboratorio durante horas de prácticas.

5. Bibliografía

- Oracle. «Lesson: Concurrency (The Java™ Tutorials > Essential Classes)». Accedido febrero 27, 2025. http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html .
- Oracle. «VisualVM». Accedido febrero 27, 2025. http://docs.oracle.com/javase/6/docs/technotes/guides/visualvm/index.html .