COMPUTACIÓN CONCURRENTE
PRÁCTICA 4

Prof.Manuel Alcántara Juárez Alain Pérez
manuelalcantara52@ciencias.unam.mx

Alejandro Tonatiuh Valderrama Silva
at.valderrama@ciencias.unam.mx

José de Jesús Barajas Figueroa
jebarfig21@ciencias.unam.mx

Luis Fernando Yang Fong Baeza
fernandofong@ciencias.unam.mx

Ricchy Alain Pérez Chevanier
alain.chevanier@ciencias.unam.mx

Fecha Límite de Entrega: 09 de Diciembre de 2020 a las 23:59:59pm.

1. Objetivo

El objetivo de esta práctica es introducir al alumno a la solución de problemas que requieren el uso mecanísmos de sincronización de la API de Java.

2. Introducción

Para resolver esta práctica es necesario que tengas nociones de las propiedades que cumplen las implementaciones de *Lock* y *Condition* de la API de Java.

3. Desarrollo

En esta práctica trabajarás con una base de código construida con Java 8[1] y Maven 3, también proveemos pruebas unitarias escritas con la biblioteca `Junit 5.5.1` que te darán retrospectiva inmediata sobre el correcto funcionamiento de tu implementación[2] .

[1] De nuevo puedes utilizar cualquier versión de java que sea mayor o igual a Java 8 simplemente ajustando el archivo pom.xml

[2] Bajo los casos que vamos a evaluar, mas estas no aseguran que tu es implementación es correcta con rigor formal

Para ejecutar las pruebas unitarias necesitas ejecutar el siguiente comando:

```
$ mvn test
```

Para ejecutar las pruebas unitarias contenidas en una única clase de pruebas, utiliza un comando como el siguiente:

```
$ mvn -Dtest=MyClassTest test
```

En el código que recibirás la clase `App` tiene un método *main* que puedes ejecutar como cualquier programa escrito en *Java*. Para eso primero tienes que empaquetar la aplicación y finalmente ejecutar el jar generado. Utiliza un comando como el que sigue:

```
$ mvn package
...
...
$ java -jar target/practica04.jar
```

4. Problema

4.1. Buffer concurrente

Utilizando únicamente semáforos crea una estructura de datos concurrente que pueda guardar N objetos, y que implemente la siguiente interfaz:

```java
public interface Buffer<T> {
    // Agrega un elemento al buffer;
    // se bloquea si esta lleno
    public void put(T item);
    // Elimina un elemento del buffer;
    // se bloquea si esta vacio.
    public T take();
    // Obtiene el numero de elementos
    // del buffer.
    public int count();
}
```

La interfaz se encuentra en Buffer.java y debes de implementarla en el archivo BufferImpl.java

4.2. Baños Mixtos

En el problema del baño compartido hay dos clases de hilos: hombre y mujer. Y hay un solo baño que es el recurso compartido, el cual se utiliza de la siguiente manera:

- **Exclusión mutua:** Personas del sexo opuesto no pueden ocupar el baño simultáneamente.
- **Libre inanición:** Todos los que necesitan usar el baño eventualmente lo ocupan.

El protocolo se implementa con los siguientes 4 métodos:

```java
// Bloquea el hilo hombre mientras haya mujeres
toilette.enterMale();
// Avisa que un hombre sale
toilette.leaveMale();
// Bloquea el hilo mujer mientras haya hombres
toilette.enterFemale();
// Avisa que una mujer sale
toilette.leaveFemale();
```

Por ejemplo, un hilo del tipo hombre podría invocar:

```
toilette.enterMale();
//seccion critica
male.doStuff();
toilette.leaveMan();
```

1. Utilizando *locks*[3] y *conditions*[4] completa la implementación de la clase `Toilette`. Para verificar que el código trabaja adecuadamente debes de pasar las pruebas unitarias contenidas en la clase `SimulationTest`.

2. Argumenta por qué tu solución cumple con la propiedad de exclusión mutua.

3. Argumenta por qué tu solución cumple con la propiedad de libre de inanición.

4. Modifica tu solución para prevenir que un hilo que ya tenga el candado de los hombres pueda adquirir el de las mujeres o viceversa. Hint: Puedes incluir una estructura map, que guarde la relación de que hilo tiene cada candado. [**Opcional: +1pt**] Implementa la relación utilizando la clase ThreadLocal de Java.

5. Evaluación

Tu entrega debe de incluir el código fuente de tu implementación. En un archivo README.pdf incluye los argumentos de por qué el código cumple con las propiedades requeridas en la descripción problema.

[3] https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/concurrent/locks/package-summary.html

[4] https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/concurrent/locks/Condition.html