COMPUTACIÓN CONCURRENTE
PRÁCTICA 2, PRIMITIVAS DE SINCRONIZACION

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

Leonardo Hernández Cano
leonardohernandezcano@ciencias.unam.mx

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

Fecha Límite de Entrega: 13 de Septiembre de 2019 a las 23:59:59pm.

1. Objetivo

El objetivo de esta práctica es implementar algunos algoritmos de sincronización vistos en clase y una aplicación del uso de las primitivas de sincronización que construiste mediante estos algoritmos.

2. 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] .
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:

[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

```
$ mvn -Dtest=PetersonLockTest 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/practica02.jar
```

3. Evaluación

Además de que tu código pase las pruebas unitarias sin que estas hayan sido modificadas, también debes de escribir una justificación breve de por qué tu solución cumple con las propiedades requeridas en la descripción del problema, añade estas justificaciones al inicio de cada clase o como pequeños comentarios intermedio dentro de tu implementación.

4. Problemas

4.1. Algoritmo de Peterson

Para esta actividad tienes que implementar el algoritmo de Peterson visto en clase para generar un *candado* que solucione el problema de la exclusión mutua para dos hilos. En el código fuente que acompaña a este documento encontrarás la clase `PetersonLock` que implementa la interfaz `Lock`, tu tarea es completar la implementación de esta clase. Para validar que tu implementación es correcta tienes que pasar todas las pruebas unitarias que se encuentran en la clase `PetersonLockTest`.

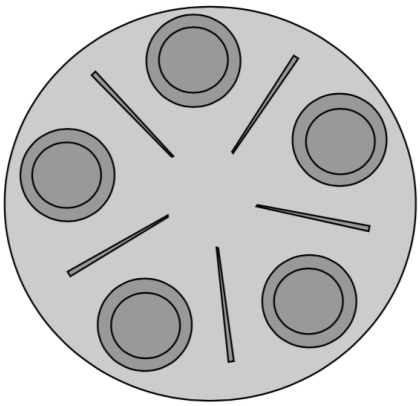
4.2. Algoritmo del Filtro

Utilizando la idea del `Algoritmo del Filtro` visto en clase, crea una implementación que permita pasar a lo más a L hilos a su sección crítica con $L < N$. Completa la implementación de la clase `FilterSemaphoreImpl`, para verificar que tu imple-

mentación funciona tienes que pasar todas las pruebas unitarias que se encuentran en la clase `FilterSemaphoreTest`.

4.3. El Problema de los Filósofos

El problema de los filósofos fue inventado por E. W. Dijkstra, un pionero de la concurrencia, para clarificar la noción de algoritmos libres de ***abrazos mortales*** y de ***hambruna***. Imagina cinco filósofos que sólo pasan su vida pensando y comiendo. Ellos se sientan alrededor de una mesa redonda con cinco sillas. La mesa tiene cinco platos grandes de arroz, sin embargo, solamente hay cinco palillos disponibles (un poco insalubre), como se muestra en la figura. Cada filósofo piensa, luego le da hambre, por lo que se sienta y toma los dos palillos que se encuentran a su lado. Si el filósofo pueda tomar ambos palillos, entonces procede a comer por un rato. Una vez que el filósofo termina de comer, éste regresa los palillos al lugar de donde los tomó y de nuevo se pone a pensar.



Para la solución a las siguientes actividades solo puedes utilizar primitivas de sincronización que tú mismo implementes, no puedes utilizar las que proporciona el *API* de `Java`. Realiza las siguientes actividades:

1. Termina de implementar la clase `ChopstickImpl`, la idea es que puedas asegurar que este objeto no puede ser tomado por más de un hilo simultáneamente. Para verificar que tu solución es correcta necesitas pasar todas las pruebas unitarias que se encuentran en la clase `ChopstickTest`.

2. Tu siguiente tarea es completar la implementación del problema de los filósofos, dicha implementacion tiene que cumplir las siguientes 3 características:

   a) Los cinco filósofos interactúan concurrentemente en la mesa, compitiendo por tomar los palillos para comer.

b) La simulación es libre de *abrazos mortales*.

c) La implementación es libre de *hambruna*.

En el código fuente encontrarás dos clases `Philosopher` y `PhilosopherWIthFilterAlgorithm`, estas son las que tendrás que completar para poder pasar las pruebas unitarias que se encuentran en la clase `PhilosopherTest`.
Para solucionar este problema puedes utilizar las siguientes estrategias:

a) Modifica la clase `PhilosopherWIthFIlterAlgorithm` para que a lo mas 4 filósofos puedan comer al mismo tiempo en ella por medio de un semáforo, utiliza la implementación de semáforos que creaste en el ejercicio 3.1.