COMPUTACIÓN CONCURRENTE

PRÁCTICA 3

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

Karla Vargas Godoy
karla.vargas@ciencias.unam.mx

Omar Arroyo Munguía
omar.am@ciencias.unam.mx

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

Fecha de Entrega: 24 de Marzo de 2019 a las 23:59:59pm.

1. Introducción

En esta práctica trabajarás con una base de código construida con Java 8 y Maven 3.

En el código que recibirás junto con este documento, prodrás encontrar la clase `App` que tiene un método *main* que puedes ejecutar como cualquier programa escrito en *Java*. Para eso primero tienes que empaquetar la aplicación y después ejecutar el *jar* generado. Utiliza comandos como los siguientes:

```
$ mvn package
...
...
$ java -jar target/practica03-1.0.jar
```

2. Pantalla de vuelos

El objetivo de este ejercicio es estudiar una situación en donde los hilos concurrentes interfieren con el uso de estructuras de datos compartidas. Para ello, vamos a considerar una pantalla como las que se encuentran en los aeropuertos y que sirven para desplegar información correspondiente a los vuelos.

En la clase `JScreen` es una interfaz gráfica que implementa la interfaz `ScreenHW`, la cual simula la programación del hardware de una pantalla que puede mostrar texto.

```
public interface ScreenHW {
    /***
     * Return the rows that this screen has
     */
    int getRows();
    /**
     * Return the columns that this screen has
     */
    int getColumns();
    /**
     * Writes <i>character</i> at the position
     * <i>(row, column)</i>
     */
    void write(char character, int row, int column);
}
```

No es necesario modificar la clase `JScreen` en esta práctica, pero si te recomiendamos que veas la clase `App`, la cual contiene un pequeño método *main* que al recibir el argumento ***JScreen***, escribe una letra A y B en distintas posiciones, después se espera 3 segundos y finalmente elimina la letra A. Prueba ejecutando el siguiente comando:

```
$ java -jar target/practica03-1.0.jar JScreen
```

Como pudes observar la interfaz `ScreenHW` es de muy bajo nivel para programar el dispositivo, así que el fabricante provee una interfaz de alto nivel llamado `ScreenHL` que es más simple consumir:

```
public interface ScreenHL {
    /**
     * Clears the whole screen
     */
    void clear();
    /**
     * Writes <i>str</i> at the end of the screen.
     */
    void addRow(String str);
    /**
     * Erases the content of the screen shown at row
     * <i>row</i>, moving all following rows one
     * position above
     */
    void deleteRow(int row);
}
```

De hecho, la intención es que sea posible agregar renglones incluso cuando la pantalla esté completamente llena; este renglón no será visible en la pantalla hasta que algunos renglones hayan sido eliminados.

La clase `JScreen2` implementa la interfaz `ScreenHL`, es importante que te familiarizes con esta clase. Además de visitar el código de la misma, para interactuar con ella puedes ejecutar el siguiente comando:

```
$ java -jar target/practica03-1.0.jar JScreen2
```

Nota que es posible acceder a la pantalla desde un programa *multi-hilos*, en donde varias aerolíneas (*hilos*) de manera concurrente pueden actualizar el contenido de la

pantalla.

3. Actividad

Tu trabajo es escribir un programa concurrente para probar la clase `JScreen2` bajo ciertas circusntancias. Un esqueleto puede encontrarse en método *exercise* de la clase `App`, el cual muestra una estructura de un programa que crea una pantalla ***S*** y crea dos hilos, uno ejecutando el método *addProc(d)* y el otro el método *deleteProc(d)*. Debes de completar el cuerpo de estos dos métodos. Llena *addProc* con una secuencia de llamadas al método *addRow* intercalados con siestas aleatorias. De manera similar completa *deleteProc* con llamadas a *deleteRow(0)*. Para no hacer una simulación aburrida procura que las siestas estén en el orden de segundos o fracciones de segundos. No empieces a borrar inmediatamente, da tiempo para que haya algunas líneas de texto en la pantalla, también procura que el borrado sea un poco más lento que la escritura y alterna los textos para que se pueda ver una clara diferencia en los errores.

Una vez que termines de implementarlos, ejecuta tu programa, probablemente verás un comportamiento incorrecto. Asegurate de que entiendes por qué estos problemas ocurren. De hecho la clase `JScreen2` no es *thread-safe*; por lo que no se garantiza su correcto comportamiento cuando sus métodos son utilizados de manera concurrente por varios hilos.

Vamos ahora a arreglar el problema de dos manera diferentes:

- El departamento de tecnología del aeropuerto ha desarrollado la aplicación `App` implementando el método *exercise*, pero no tiene acceso a la clase `JScreen2`, sólo a la interfaz que implementa. Por lo que se tiene que resolver el problema sin modificar la clase `JScreen2`. Una manera de hacerlo es indentificando las secciones críticas en `App.exercise` y protegerlas utilizando un semáforo. Puedes utilizar una instancia de `java.util.concurrent.Semaphore`. Nota que el método *acquiere()* puede lanzar la excepción `InterruptedException`. Prueba tu programa de nuevo para ver si parace que tiene un comportamiento adecuado. Mencionamos la palabra *parece*, ya que a estas alturas debes de tener una idea de las dificultades que pueden llegar a presentar las pruebas en un programa concurrente.

- El departamento oficial del aeropuerto, le pidió al fabricante una implementación *thread safe* de la clase `JScree2`. Programa este requerimiento, es bastante sencillo.