Programación de Servicios y Procesos. 2021. Primera Evaluación.
Simulación de Casa de Restaurante

Debido a la pandemia del Covid-19 y buscando nuevas oportunidades de negocio, la empresa Food & Drink SL, ha decido montar un restaurantes. Para, realizar la estimación del posible funcionamiento del negocio se nos ha encargado diseñar una aplicación que simule el comportamiento del restaurante Restaurante y la gestión de las comandas.

Así, nuestro objetivo es determinar el tiempo de simulación para atender un conjunto de clientes (**Clase Mesa**) mediante un número limitado de cocineros (en función de una variable `final static int maxCocineros = 8`).

La clase responsable de la simulación (**Clase Restaurante**), que es la que tiene el **main**, debe iniciar un conjunto de instancias de la Clase **Mesa** (dependiendo de un valor prefijado `final int maxMesas = 100`), esperar a que las instancias terminen de simular que son atendidas y mostrar un mensaje con el tiempo en ms de la simulación.

Cuando las instancias de la Clase Mesa, no puedan obtener un cocinero (Semáforo con un valor inicial de maxCocineros), enviarán un mensaje por un pipe (Mesa <id> esperando cocinero) a una instancia de la **Clase Metre.**
Esta clase **(Metre)** que también se ejecuta como hilo y recibe el comunicado debe imprimir "Recibido: Mesa <id> esperando cocinero".

Es necesario realizar una **simulación con [nº de la lista * 10] Mesas; con un máximo de 350, bajad el tiempo del delay para más de 150 mesas]**, debiendo adjuntar **capturas de la simulación** en la entrega del ejercicio, junto con una **copia completa de los directorios del proyecto.**
Por ejemplo, si soy el alumno 10, mis simulaciones serán con 100 Mesas].

Funcionamiento de las clases
Clase Restaurante (main)

- Inicialización de todos las variables
- Inicio de la clase Metre [Metre metre = new Metre(flujoE)]
- Inicio de los hilos que simulan los Mesas
- Imprimir mensaje de "iniciando el servicio de Restaurante"
- Imprimir mensaje con el nombre del alumno y la hora de inicio
- Espera de que todos las Mesas terminen de ser asistidas
- Finalización de la instancia Metre
- Impresión del tiempo de la simulación en ms
- Imprimir mensaje con el nombre del alumno y la hora de fin

Clase Mesa

- Esperar que todos los hilos se inicien concurrentemente (sincronizamos mediante el mecanismo apropiado)
- Mientras no pueda completar su comanda, deben intentar obtener un cocinero (Semáforo).
  - Si lo consigue debe esperar un tiempo aleatorio de 1sg máximo y terminar informando de que ha terminado (sincronizamos mediante el mecanismo apropiado), para que una vez que todos los hilos hayan terminado el padre pueda terminar e imprimir el tiempo de la simulación
  - Si no consigue cocinero, debe esperar un tiempo aleatorio de 1sg máximo e imprimir por pantalla y enviar por el pipe el mensaje Mesa <id> esperando cocinero
- Cuando una mesa imprima por pantalla debe evitar que otra acceda a la pantalla también, por ello es necesario bloquearla.

Clase Metre

- Recibir por el pipe los mensajes de los Mesas e imprimir por pantalla "Recibido: Mesa <id> esperando cocinero"
- Finalizar cuando el main se lo indique
- **Atención**: Es posible que al finalizar los hilos de las Mesas, el Metre continúe su ejecución, y se produzca un **error de brokepipe**, es este caso capturar el error y finalizar la ejecución del Metre.

Nuestra aplicación constará de la siguientes clases (no son necesarias más clases, la incorporación de toda la sincronización una clase **Sincro es opcional)**:

1. **Restaurante**, que es la aplicación principal (tiene el main[]) que inicia todos los hilos y espera que termine todo el proceso, imprimiendo toda la información de la simulación
2. **Metre**, que recibe los mensajes de las Mesas por el pipe y los imprime
3. **Mesa**. Simula el Mesa, esperando a que un cocinero atienda su pedido. Cuando no puede obtener cocinero emite por el pipe un mensaje. Se ejecuta como un hilo
4. **Sincro**. Clase **opcional** que incorpora los mecanismos de sincronización.

Criterios de evaluación:

1. Las clases se ha implementado adecuadamente, realizan la función sugerida y los parámetros de las clases son coherentes con su funcionalidad. (2 puntos)
2. La sincronización de la toda la simulación se ha realizado adecuadamente, utilizando las clases de concurrencia que java proporciona, y ejecutando la sincronización tal y como se ha descrito. (2 puntos)
3. La gestión de los cocineros se ha implementado correctamente, tanto su asignación al Mesa como su no asignación. (2 puntos)
4. La aplicación presenta una lógica coherente y funciona correctamente. (2 puntos)

Entrega del ejercicio

El ejercicio deberá ser entregado en un archivo comprimido con el nombre y apellidos, **conteniendo todos los archivos del proyecto y un documento con las capturas de pantalla de las simulaciones (sin toda esta información no se evaluará el ejercicio)**, donde se aprecie claramente el nombre y hora de inicio y fin de la simulación.

Ejemplo de mensaje de finalización

```
Simulación terminadada
El tiempo de la simulación ha sido de 64730 ms
Antonio F Pele :1:41:41
```

A modo de ejemplo

```
public Metre(BufferedReader flujoE)
```