***Por:***

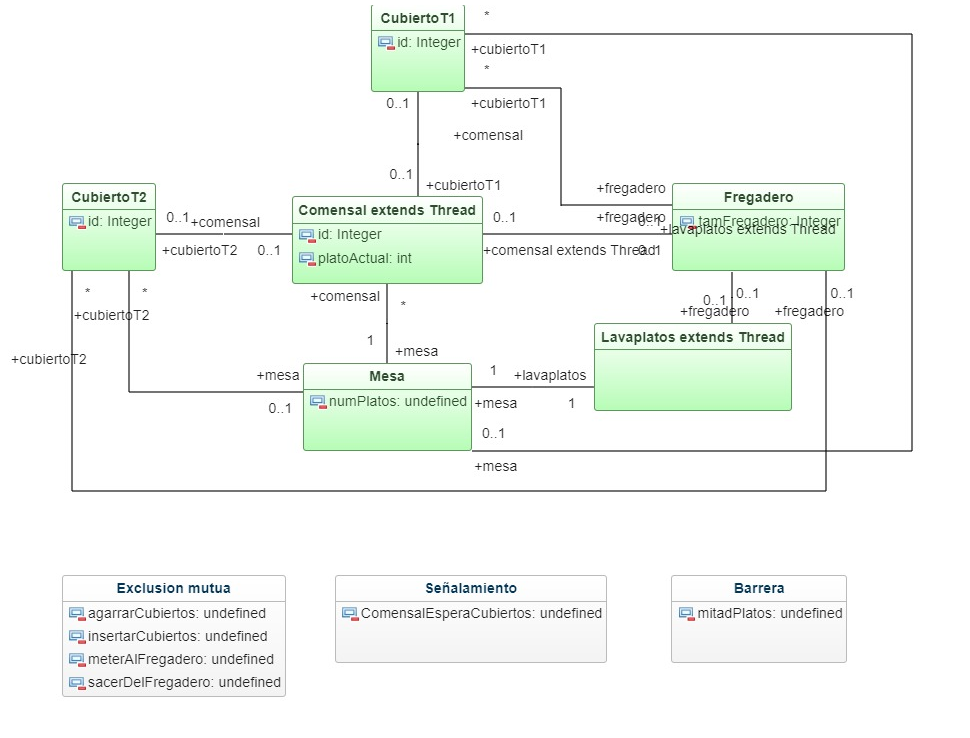
*Juan Sebastián Ramírez 201923800*

*Andrés Santiago Triana 201923265*

**Explicación solución Caso 1**

**Diagrama**

En primer lugar, hicimos un diagrama para guiarnos en el proceso de crear el código. Como fue una idea preliminar a la implementación, no es exactamente fidedigno al resultado final, sin embargo, la estructura es muy similar. (en este diagrama no pusimos el main pero en el código esta clase se conecta con todas las demás y las inicializa).



**Explicación de las clases**

**Cubierto**

Cubierto es la clase padre de CubiertoT1 y cubiertoT2. Los cubiertos tienen solo un id para identificarlos y una descripción que se llena directamente en el constructor. Es decir, la descripción dice de que tipo es el cubierto.

**Mesa**

La mesa tiene varias funciones. En primer lugar, los cubiertos van a crearse al inicio y van a ser puestos en la mesa. En segundo lugar, va a guardar un número ilimitado de cubiertos, pero va a ser capaz de distinguir si se le solicita un cubierto tipo 1 o un cubierto tipo 2. En tercer lugar, guarda la cantidad de platos que va a comer cada uno de los invitados.

**Comensal**

En primer lugar, el comensal extiende Thread porque cada comensal se va a ejecutar como un Thread propio. Los comensales van a guardar el número de platos que han comido para poder hacer la verificación cuando lleguen a la mitad y deban esperar a los demás. Tienen un id para distinguirlos, una relación con la mesa, una relación con el fregadero y tienen variables para guardar un cubierto de cada tipo. Finalmente, tienen una barrera de tipo CyclicBarrier (explicada más adelante).

**Fregadero**

En el fregadero se van a guardar un número limitado de cubiertos de cualquier tipo, este número viene dado por parámetro desde la clase main. Además de guardar su tamaño, guarda la lista de los cubiertos que están dentro de este, en una lista encadenada tipo cola (tipo fifo).

**Lavaplatos**

Lavaplatos es un Thread que se va a encargar de tomar un cubierto del fregadero cada vez que haya uno disponible. Una vez acaba de lavar el cubierto lo envía a la mesa. Tiene una relación con fregadero y con mesa para poder hacer estas acciones. Su implementación consta de un ciclo infinito que se da por terminado cuando el main le informa la terminación de los comensales y cuando no existan más cubiertos en el fregadero para lavar.

**Main**

El main se va a encargar de inicializar los elementos anteriormente descritos y por ende, simular toda la cena. Así mismo, es esta clase la que carga todos los parámetros del archivo con la información dada por el usuario (localizado en ./data/data1.txt)

**Implementaciones**

**Inicialización**

En primer lugar, hicimos un método para poder cargar todos los parámetros que llegan del txt y con estos datos poder inicializar correctamente la aplicación. Una vez se hace la carga, se corre un método de iniciar cena, este crea e inicializa todos los objetos (los cubiertos, la mesa, el fregadero, el lavaplatos y los comensales) y también hace el **start()** de los Threads (comensales y lavaplatos).

**Modelamiento de los comensales comiendo**

Un comensal es un thread. El run de este Thread funciona de la siguiente manera: En primer lugar, toma de la mesa los cubiertos que va a usar, este método requiere **exclusión mutua** puesto que si dos comensales toman un cubierto al mismo tiempo está la probabilidad de que este sea duplicado. Si un comensal toma un tipo de cubierto y no del otro, entonces devuelve el cubierto que logró coger de la mesa y hace un **wait()** hasta que la mesa le haga un **notify()** para que intente nuevamente tomar cubiertos.

Si el comensal logre coger los dos cubiertos, va a empezar a comer, lo cual se modela como un random entre 3 a 5 segundos. Una vez acaba de comer, envía los cubiertos al fregadero y espera 1 ó 2 segundos para empezar con el siguiente plato.

Si el comensal intenta enviar los cubiertos al fregadero y este está lleno, hace una espera pasiva (con un **wait()**) hasta que el fregadero le notifique que ya tiene espacio (**notify()**).

Finalmente, cada vez que el comensal va a empezar un plato se verifica en qué plato va y en caso de que este se encuentre en la mitad de todos los platos, hace un **await()** con la **CyclicBarrier** pasada por parámetro en su declaración (creada por el main con la cantidad de comensales). De esta manera se asegura que se añade una barrera a los comensales, para que estos se esperen entre si antes de seguir con la otra mitad de los platos.

**Modelamiento del lavado de los cubiertos**

El fregadero va a guardar los cubiertos que van dejando los comensales y no se modela con exclusión mutua puesto que no existiría problema con que varios comensales dejen sus cubiertos a la vez (no existe posibilidad de que se duplique nada).

El lavaplatos está constantemente verificando que haya cubiertos para lavar en el fregadero, esto se hace por medio de un método dentro de fregadero que indica el tamaño de la cola de cubiertos, si esta está en 0, entonces se entiende que no hay cubiertos disponibles para lavar.

Tan pronto encuentra un plato disponible, lo lava, el cual es un modelamiento en el que espera 1 o 2 segundos simulando lavar el cubierto. Una vez finaliza, envía el cubierto a la mesa donde queda nuevamente disponible para que los comensales lo utilicen.

Una vez se desocupa espacio en el fregadero (después de estar lleno) se hace un **notify()** para avisar a los comensales que pueden estar esperando a dejar sus cubiertos en el fregadero y que cuando lo intentaron este estaba lleno.

**Modelamiento de la Mesa**

La mesa recibe (de comensales y del lavaplatos) cubiertos y aprovisiona a los comensales. Estos dos métodos (dar y añadir) están denotados con **synchronized** para evitar que dos threads cojan o dejen cubiertos a la vez (dando espacio para la duplicación de cubiertos). Esto significa que tienen exclusión mutua.

Cuando llega un tipo de cubierto del cual la mesa no tenía ninguno, se llama a un **notify()** de manera que se activa un comensal que puede estar esperando por un cubierto.

**Conclusiones**

Pudimos evidenciar los 3 tipos de sincronización durante la resolución del caso:

* Exclusión mutua: En la mesa es necesario que los métodos dar y añadir cubiertos estén sincronizados y no se ejecuten al tiempo, ni tampoco que sean llamados por varios Threads a la vez.
* Señalamiento: Cuando no hay cubiertos limpios en la mesa los comensales pasan a esperas pasivas, que después son interrumpidas por la mesa cuando llegan cubiertos limpios para ser usados. De igual forma, los comensales también hacen esperas pasivas cuando el fregadero está lleno. El fregadero les notifica cuando ya hay espacio para que depositen sus cubiertos sucios.
* Encuentro: Se usó encuentro en dos fragmentos específicos; A la mitad de la comida, los comensales se esperan entre si con una barrera declarada en el main. Esta solamente continua la ejecución de estos threads cuando se asegura que todos los comensales han llegado a la mitad de su cena (Usando **CyclicBarrier**). Además, se hace un encuentro en el main para verificar la terminación de los Threads de los comensales y avisarle al lavaplatos que la cena a finalizado (Usando **join()**).