Actividad Actividad UF1-2. Multitarea. Centro de exámenes

El ejercicio parte de un supuesto donde una serie de hilos acceden a un recurso para consumirlo, mientras que otros hilos estarán produciendo dicho recurso.

Esta situación puede dar lugar a los siguientes casos de conflicto:

- Puede que los "examinados" intenten consumir un recurso que no existe.
- Puede que se produzcan más exámenes de los necesarios.

Con respecto a esta segunda situación, no hay ningún problema, ya que solamente hay 3 hilos produciendo y cada uno de estos hilos va a producir solamente una vez. Aunque en caso de que estuvieran produciendo de forma recurrente, sería necesario añadir alguna lógica adicional para evitar un exceso de stock -por ejemplo, eliminando recursivamente el inicio de la cola o bloqueando la producción hasta que la longitud de la cola disminuya.

En cualquier caso, para este ejercicio vamos a centrarnos en la lógica que se refiere al Examinado y su relación con el buffer (ya que esta es la clase que tiene que bloquear el proceso y a la que se le notificará el fin del bloqueo).

Partimos de 3 clases:

1. Clase "Examinado" (consumidor).
2. Clase "ProductorExamenes".
3. Buffer de Exámenes, donde se van a ir almacenando.
4. Clase "main" donde se van a ejecutar todos los hilos de forma simultanea.

Lógica de BufferExamenes

Todo parte del método "consumirExamen". Creamos un método public synchronyzed donde el Examinado intentará consumir un examen de la cola que hemos creado para tal propósito (recordamos que hemos instanciado una LinkedList sobre la que se va a consumir siempre el item que esté en cabeza).

```
public synchronized String consumirExamen(){
        // Forzamos un tiempo de espera para que no se adelante
        int espera = 0;
        while (colaExamenes.isEmpty() && espera < 20){
            System.out.println("No hay exámenes que consumir");
            espera ++;
            try{
                wait(1000);
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
            }
        }
        if (!colaExamenes.isEmpty()){
            return colaExamenes.remove();
```

```
            } else {
                return null;
            }
        }
```

Nuestra lógica va a crear un tiempo de espera de 20 iteraciones multiplicado por 1 segundo de espera mientras que la cola esté vacía. Después de 20 iteraciones, el buffer se libera bien sea con un resultado exitoso -se consume el examen- o bien sin ningún resultado -return null.

El método wait() pondrá en pausa la instancia que lo haya llamado y quedará a la espera de un .notify(), que se llamará desde alguna de las instancias de la clase ProductorExamenes y que "despertará" cualquier hilo que esté en pausa -aunque igualmente, esta lógica no es estrictamente necesaria ya que después de 20 iteraciones la instancia del Examinado resolverá su situación.

```
public synchronized void fabricarNuevoExamen(String codigo) {
        colaExamenes.add(codigo);
        notify();
    }
```

Resultados

Tras varias pruebas, el resultado por consola acaba siendo siempre algo similar a esto, pudiendo concluir con éxito el ejercicio:

No hay exámenes que consumir No hay exámenes que consumir No hay exámenes que consumir E3 Se ha generado examen con código = E3-2024 E3-2024 - Rosa Pregunta: 1B E1 Se ha generado examen con código = E1-2024 E2-2024 - Miguel Pregunta: 1E E2 Se ha generado examen con código = E2-2024 E2-2024 - Miguel Pregunta: 2A E2-2024 - Miguel Pregunta: 3B E2-2024 - Miguel Pregunta: 4C E2-2024 - Miguel Pregunta: 5A E2-2024 - Miguel Pregunta: 6E E2-2024 - Miguel Pregunta: 7B E2-2024 - Miguel Pregunta: 8E E2-2024 - Miguel Pregunta: 9E E2-2024 - Miguel Pregunta: 10D E2-2024 - Miguel- Examen terminado. E1-2024 - Carlos Pregunta: 1E E3-2024 - Rosa Pregunta: 2A E3-2024 - Rosa Pregunta: 3B E3-2024 - Rosa Pregunta: 4D E3-2024 - Rosa Pregunta: 5E E3-2024 - Rosa Pregunta: 6B E3-2024 - Rosa Pregunta: 7B E3-2024 - Rosa Pregunta: 8E E3-2024 - Rosa Pregunta: 9D E3-2024 - Rosa Pregunta: 10A E3-2024 - Rosa- Examen terminado. E1-2024 - Carlos Pregunta: 2A E1-2024 - Carlos Pregunta: 3E E1-2024 - Carlos Pregunta: 4D E1-2024 - Carlos Pregunta: 5B E1-2024 - Carlos Pregunta: 6A E1-2024 - Carlos Pregunta: 7A E1-2024 - Carlos Pregunta: 8C E1-2024 - Carlos Pregunta: 9C E1-2024 - Carlos Pregunta: 10D E1-2024 - Carlos- Examen terminado. Process finished with exit code 0

Métodos wait y notify + synchronized

- NOTA: a continuación una breve introducción teórica que me servirá de apuntes en el futuro, aunque no venga exactamente al caso del ejercicio. *

-Cuando un hilo llama a wait(), se libera el bloqueo (lock) que tiene sobre ese objeto, permitiendo que otros hilos lo adquieran. El hilo que llamó a wait() se detiene y queda en espera hasta que se le notifique que puede

continuar.

```
synchronized void nombreDelMetodo(Params parametros) {
    while (!condition) {
        wait();
    }
    // Código para ejecutar cuando condition es verdadera
}
```

-Se llama a notify() después de realizar una acción que permite a otros hilos avanzar. El hilo que ha sido despertado no continúa de inmediato; primero tiene que esperar hasta que pueda adquirir el bloqueo del objeto.

```
synchronized void nombreDelMetodoQueDespierta() {
    notify();  // Despierta un hilo que está esperando
}
```

Syncronized

El wait y el notify no se pueden meter a pelo, necesitan un método con el decorador "Syncronized", que es lo que permite bloquear el proceso.

Todo el código que esté dentro de un bloque Syncronized bloqueará lo que suceda dentro para que nadie más acceda al recurso hasta que se ejecute dicho bloqueo. Por lo general, se va a bloquear a sí mismo.