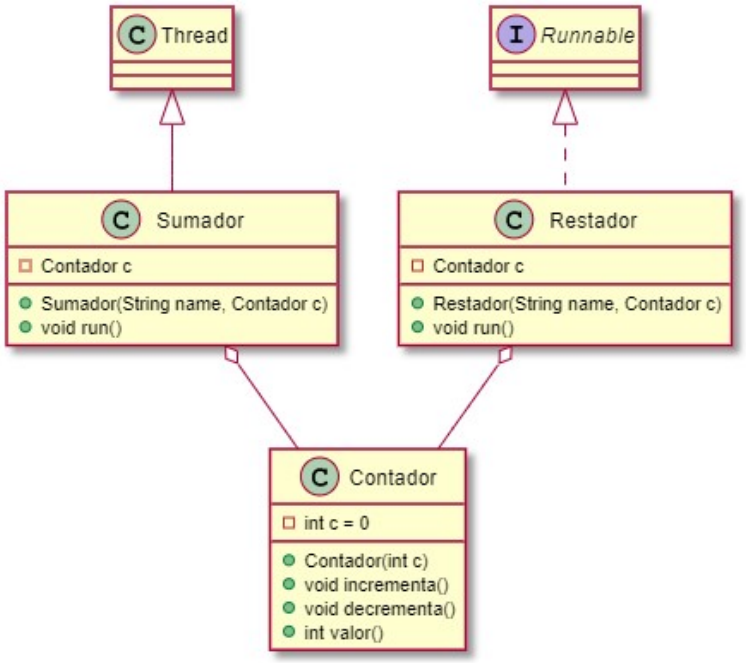
Programación de Hilos en Java

Ejercicios de Sincronización

Ejercicio 1: Contador Síncrono

Implementa un sistema que maneje correctamente el acceso concurrente a un contador compartido por varios hilos. En este sistema, algunos hilos se encargarán de sumar al contador, mientras que otros hilos lo restarán. Es necesario que el acceso al contador esté sincronizado para evitar condiciones de carrera y garantizar que las operaciones de suma y resta se realicen correctamente.



Ejercicio 2: Carrera de Galgos

Implementar un programa en Java que simule una carrera de caballos utilizando hilos. Cada hilo representará un caballo, y el progreso de cada caballo se debe simular utilizando `Thread.sleep()` con tiempos aleatorios. El número de caballos y la distancia de la carrera será especificado por el usuario.

Requisitos:

1. Debes implementar un método `public static void startRace(int numHorses, int distance)` que reciba como parámetros el número de caballos y la distancia de la carrera.
2. Cada hilo debe representar un caballo y debe avanzar en la carrera en intervalos de tiempo aleatorios.
3. Se debe implementar una forma de asegurar que solo un galgo a la vez pueda anunciar su llegada y se debe determinar el orden de llegada de los galgos.

El programa debe imprimir el progreso de cada caballo y notificar cuándo llegue a la meta, indicando el tiempo total que ha tardado.

Ejercicio 3: Urgencias

Implementa un programa en Java que simule el funcionamiento de las urgencias de un hospital, donde los pacientes llegan con diferentes niveles de urgencia (LEVE, MODERADO y GRAVE) y deben ser atendidos por un único médico a la vez. Los pacientes más urgentes tienen prioridad sobre los demás. Para simular esta prioridad, se usará el mecanismo de prioridades de los hilos en Java, donde un hilo con mayor prioridad representa un paciente con mayor urgencia.

Ampliación Ejercicio 3: Urgencias

Mide el tiempo medio de espera de cada tipo de paciente (LEVE, MODERADO, GRAVE)

Ejercicio 4: La cena de los Filósofos

Cinco filósofos están sentados alrededor de una mesa circular para cenar. Cada filósofo pasa su vida pensando y comiendo. En el centro de la mesa hay un plato de espaguetis, y entre cada dos filósofos hay un único tenedor. Para comer, un filósofo necesita tener los dos tenedores a su disposición: el que está a su izquierda y el que está a su derecha. Después de comer, el filósofo deja los tenedores y vuelve a pensar.

Reglas:

1. Solo un filósofo puede comer en un momento dado si tiene ambos tenedores.
2. Dos filósofos no pueden usar el mismo tenedor simultaneamente.
3. Los filósofos deben alternar entre pensar y comer.

Implementa el problema de la cena de los filósofos en Java usando concurrencia. Cada filósofo debe ser modelado como un hilo independiente. Los tenedores deben representarse como recursos compartidos (objetos sincronizados) entre los filósofos vecinos. La solución debe evitar el bloqueo mutuo (deadlock) y, de ser posible, minimizar la espera activa (busy waiting).

Ejercicio 5: Productor-Consumidor de Factoriales

Implementa un programa en Java que simule el problema del **Productor-Consumidor** utilizando hilos. En este escenario, hay varios **productores** que generan números enteros aleatorios y los depositan en una **cola compartida de tamaño fijo**. Simultáneamente, hay varios **consumidores** que toman los números de la cola, calculan su factorial e imprimen el resultado en la consola.

Reglas del problema:

1. **Productores**:
    - Cada productor debe generar un número entero aleatorio entre 1 y 10.
    - El número generado debe colocarse en una cola compartida.
    - Si la cola está llena, el productor debe esperar hasta que haya espacio disponible.
2. **Consumidores**:
    - Los consumidores deben tomar los números de la cola.
    - Por cada número que tomen, deben calcular el factorial de dicho número.
    - Una vez calculado, deben imprimir en la consola el número original y su factorial.
    - Si la cola está vacía, el consumidor debe esperar hasta que haya números disponibles.
3. **Cola compartida de tamaño fijo**:
    - La cola utilizada para compartir los números entre los productores y consumidores debe tener un **tamaño máximo predefinido**.
    - Si la cola está llena, los productores deben esperar hasta que haya espacio disponible para producir.
    - Si la cola está vacía, los consumidores deben esperar hasta que haya números disponibles.
    - El acceso a la cola debe estar protegido para evitar condiciones de carrera mediante el uso de técnicas de sincronización (`synchronized`, `wait()`, `notifyAll()`).
4. El programa debe funcionar con **múltiples productores y consumidores** ejecutándose en paralelo.

Ejercicio 6: FastFood Service

Se desea simular el funcionamiento de un restaurante utilizando hilos en Java. El sistema debe incluir **camareros**, **cocineros** y **food runners** (repartidores de comida), que interactúan mediante colas compartidas para gestionar los pedidos de manera concurrente.

Descripción del escenario:

1. **Camareros**:
    - Los camareros recibirán los pedidos de los clientes y los colocarán en una cola de pedidos.
    - Si la cola de pedidos está llena, el camarero deberá esperar hasta que haya espacio disponible.
    - No es necesario incluir detalles del pedido, pero es necesario simular el tiempo que tarda el camarero en tomar nota del pedido.
2. **Cocineros**:
    - Los cocineros leerán los pedidos de la cola de pedidos.
    - Una vez que un pedido es leído, el cocinero lo preparará (simulado con un tiempo de espera).
    - Cuando el pedido esté preparado, lo colocarán en una segunda cola de pedidos preparados.
    - Si la cola de pedidos está vacía, el cocinero deberá esperar hasta que haya nuevos pedidos disponibles.
3. **Food Runners**:
    - Los food runners tomarán los pedidos de la cola de pedidos preparados y los llevarán a los clientes.
    - Una vez que un pedido es entregado, el food runner lo quitará de la cola de preparados.
    - Si la cola de pedidos preparados está vacía, el food runner deberá esperar hasta que haya nuevos pedidos listos para ser entregados.