Simulación de una atracción con aforo máximo - HILOS

Vas a programar una **simulación de una atracción de un parque** utilizando **hilos en Java**.

La aplicación tendrá **interfaz no visual**:

- Toda la información se mostrará con **mensajes de texto** (por consola y/o en un fichero de log).
- No es necesario usar ratón, ni elementos gráficos, ni ventanas.

Al final del enunciado hay una **parte opcional** sobre interfaz gráfica, pensada para quien quiera practicar más, pero **la parte obligatoria es totalmente textual**.

1. Clases que debes crear

Deberás crear al menos las siguientes clases:

1. Atraccion
    - Gestiona las plazas de la atracción y el aforo máximo.
2. Plaza
    - Representa una plaza (un coche con un visitante).
3. SubeVisitantes
    - Hilo encargado de subir visitantes a la atracción y poner los coches en marcha.
4. SacaVisitantes
    - Hilo encargado de detener coches en marcha y sacar visitantes.
5. Registro
    - Clase para guardar mensajes de la simulación en un **fichero de texto** y opcionalmente mostrarlos por consola.
6. Inicio
    - Contiene el método main y crea los objetos y los hilos necesarios.

2. Comportamiento de la atracción (Atraccion y Plaza)

Aforo y plazas

- La atracción tiene un **aforo máximo de 10 plazas**.
  Eso significa que como máximo habrá 10 coches con visitante durante toda la simulación.
- Las plazas se numeran de **1 a 10**.
- Cada plaza se representa mediante un objeto de la clase Plaza, que debe almacenar como mínimo:

- Número de plaza (entero).
- Número de vueltas dadas (entero).
- Si el coche está en marcha o no (booleano).
- Si la plaza sigue activa o ya ha terminado definitivamente (booleano).

Reglas de funcionamiento

- En el recorrido de la atracción **solo puede haber 3 coches en marcha a la vez**. Si se quiere poner en marcha un cuarto coche, deberá **esperar** hasta que uno de los tres termine.
- Cada coche tiene un contador de **vueltas**, desde 1 hasta un máximo de 5:
  - Si el coche supera las 5 vueltas (es decir, pasa a 6), se considera que el visitante se marea y el viaje termina de forma brusca.
- El coche empieza con **1 vuelta**.
- Un coche puede **terminar su viaje** de dos formas:
  - Porque llega a más de 5 vueltas (visitante mareado).
  - Porque uno de los hilos SacaVisitantes lo detiene antes.

Mensajes obligatorios

Cada vez que ocurra uno de estos eventos, se debe escribir un mensaje usando la clase Registro:

1. **Asignación de plaza**:
   - Formato:
     PLAZA X ASIGNADA A SVn
     Donde X es el número de plaza y SVn es el nombre del hilo SubeVisitantes, por ejemplo SV3.
2. **Inicio de viaje** (cuando el coche pasa a estar en marcha):
   - Formato orientativo:
     PLAZA X INICIA VIAJE (SVn)
3. **Vuelta completada**:
   - Formato:
     PLAZA X VUELTA V
     Donde V es el número de la vuelta actual.
4. **Fin de viaje por mareo** (se pasa de 5 vueltas):
   - Formato:
     PLAZA X FIN DE VIAJE (MAREADO)
5. **Fin de viaje por salida anticipada** (hilo SacaVisitantes):

- Formato:
  PLAZA X VISITANTE SALIÓ POR SaVn
  Donde SaVn es el nombre del hilo SacaVisitantes, por ejemplo SaV2.

3. Hilos que suben visitantes (SubeVisitantes)

- Deberás crear **5 hilos** de la clase SubeVisitantes.
- Cada uno tendrá un nombre del tipo SV1, SV2, ..., SV5.

Comportamiento

Cada hilo SubeVisitantes debe:

1. Pedir una nueva plaza a la atracción, en orden, mientras queden plazas disponibles.
   - Si ya se han asignado las 10 plazas, dejará de pedir más y terminará.
2. Poner en marcha el coche asociado a esa plaza:
   - Si ya hay 3 coches en marcha, el hilo debe **esperar** hasta que haya hueco.
3. Una vez el coche está en marcha, el hilo irá aumentando el número de vueltas:
   - Cada segundo (aproximadamente) incrementará el número de vueltas en 1
   - Tras cada incremento, llamará a un método de Atracción para que:
     - Actualice el número de vueltas.
     - Escriba el mensaje correspondiente.
     - Indique si el coche puede seguir o debe terminar.
4. Si el coche termina:
   - Por mareo (más de 5 vueltas), o
   - Porque un hilo SacaVisitantes lo ha detenido,
     entonces el hilo SubeVisitantes volverá a pedir otra plaza (si queda) y repetirá el proceso.

4. Hilos que sacan visitantes (SacaVisitantes)

- Deberás crear **5 hilos** de la clase SacaVisitantes.
- Cada uno tendrá un nombre del tipo SaV1, SaV2, ..., SaV5.

Comportamiento

Cada hilo SacaVisitantes debe:

1. Mientras queden plazas activas en la atracción:
   - Esperar un tiempo **aleatorio entre 1 y 10 segundos**.

    - Intentar detener uno de los coches que estén en marcha, llamando a un método de Atracción.

2. Si encuentra un coche en marcha:
    - Lo marcará como finalizado.
    - Reducirá el número de coches en marcha.
    - Escribirá el mensaje correspondiente usando Registro.
3. Si no hay coches en marcha pero todavía quedan plazas que no han empezado o no han terminado, el hilo puede:
    - Volver a intentarlo tras otro tiempo aleatorio.
4. Cuando ya no queden plazas activas (las 10 plazas han terminado de una forma u otra), el hilo SacaVisitantes debe terminar.

5. Registro de eventos en fichero (Registro)

Deberás crear una clase Registro que:

- Abra un **fichero de texto** (por ejemplo: log_atraccion.txt).
- Tenga un método log(String mensaje) que:
    - Añada fecha y hora al mensaje.
    - Escriba la línea en el fichero.
    - Opcionalmente, escriba también el mensaje por consola.

Puedes implementar Registro como **singleton** (una única instancia accesible desde cualquier parte del programa).

6. Clase de inicio (Inicio)

La clase Inicio debe:

1. Crear la instancia de Registro indicando el nombre del fichero de log.
2. Crear una instancia de Atracción.
3. Crear los 5 hilos SubeVisitantes y los 5 hilos SacaVisitantes.
4. Arrancar todos los hilos.
5. Esperar a que terminen (usar join()).
6. Escribir en el log un mensaje final, por ejemplo:
   SIMULACIÓN FINALIZADA.

7. Sincronización y uso de wait() / notifyAll()

En este ejercicio se valorará especialmente:

- El uso correcto de synchronized en los métodos de Atracción.
- El uso correcto de wait() y notifyAll() para:
    - Hacer que los hilos SubeVisitantes esperen cuando ya haya 3 coches en marcha.
    - Despertar a los hilos cuando:
        - Un coche termina (por mareo o salida).
        - Hay hueco para otro coche en marcha.
- Que no haya condiciones de carrera ni inconsistencias en los estados de las plazas.

8. Parte opcional (para ampliar): interfaz visual

Esta parte **no es obligatoria** para superar el ejercicio.
Es solo para quien quiera practicar más.

Opcionalmente, puedes crear una segunda versión que:

- Muestre los mensajes en una pequeña ventana (por ejemplo, usando Swing).
- Siga escribiendo los mensajes en el fichero de log.
- Reutilice la misma lógica de hilos y la misma clase Atracción.

Modelo Productor–Consumidor en la atracción

En este apartado deberás **modificar parte de la solución** para que la atracción funcione explícitamente como un **buffer acotado** siguiendo el modelo **productor–consumidor**.

B.1. Identificación de roles

En el contexto de la atracción:

- Los hilos SubeVisitantes actuarán como **productores**.
    - Producen "coches en marcha" (plazas activas que salen al recorrido).
- Los hilos SacaVisitantes actuarán como **consumidores**.
    - Consumen "coches en marcha" (detienen un coche y sacan al visitante).
- La clase Atraccion actuará como:
    - **Monitor de sincronización** (synchronized, wait(), notifyAll()).
    - **Buffer acotado** con capacidad máxima de **3 coches en marcha**.

B.2. Buffer acotado en la clase Atraccion

Deberás modificar o ampliar la clase Atraccion para que gestione de forma explícita un buffer de coches en marcha.

Se pide:

1. Declarar una estructura interna que represente el buffer de coches en marcha, por ejemplo:
    - Una cola (Queue<Plaza>) o una lista (List<Plaza>).
2. Definir una constante de capacidad (si no existía ya):
    - CAPACIDAD_BUFFER = 3;
      Esta capacidad representa el número máximo de coches que pueden estar en marcha simultáneamente.

B.3. Operación de productor: poner un coche en marcha

Implementar en Atracción un método sincronizado similar a:

public synchronized void ponerEnMarcha(Plaza p, String nombreSV) throws InterruptedException

Este método debe comportarse como una operación **put** en un buffer acotado:

1. Mientras el buffer esté lleno (hay 3 coches en marcha), el hilo productor deberá:
    - Llamar a wait() y quedar bloqueado.
2. Cuando haya hueco:
    - Marcar la plaza como "en marcha".
    - Insertarla en la estructura del buffer.

- Escribir en el log un mensaje del estilo:
  PLAZA X INICIA VIAJE (SVn)
- Llamar a notifyAll() para despertar a posibles consumidores (y otros productores).

Los hilos SubeVisitantes deberán utilizar este método para poner en marcha los coches en lugar de hacerlo directamente.

B.4. Operación de consumidor: detener un coche en marcha

Implementar en Atracción un método sincronizado similar a:

public synchronized Plaza detenerParaSalida(String nombreSaV) throws InterruptedException

Este método debe comportarse como una operación **take** en un buffer acotado:

1. Mientras el buffer esté vacío **y** todavía queden plazas activas en la atracción:
   - El hilo consumidor deberá llamar a wait() y quedar bloqueado.
2. Si el buffer está vacío y ya no quedan plazas activas:
   - Devolver null para indicar que no hay nada más que consumir.
3. Si hay algún coche en marcha:
   - Extraer una plaza del buffer (por ejemplo, con remove() si es una cola).
   - Marcarla como no en marcha y no activa.
   - Actualizar el contador de plazas finalizadas.
   - Escribir en el log un mensaje del estilo:
     PLAZA X VISITANTE SALIÓ POR SaVn
   - Llamar a notifyAll() para despertar a posibles productores.

Los hilos SacaVisitantes deberán utilizar este método para detener coches en marcha con la lógica antes descrita.

B.5. Adaptación de los hilos

1. **Hilos SubeVisitantes (productores)**
   Deberán:
   - Seguir pidiendo plazas con asignarPlaza(...) mientras haya disponibles.
   - Llamar a ponerEnMarcha(plaza, getName()) para introducir la plaza en el buffer.
   - Mantener o adaptar la lógica de vueltas según lo indicado en el Apartado A.
2. **Hilos SacaVisitantes (consumidores)**
   Deberán:
   - Esperar un tiempo aleatorio entre 1 y 10 segundos.

- Llamar a detenerParaSalida(getName()) para consumir una plaza (coche en marcha).
- Terminar cuando el método devuelva null o cuando ya no queden plazas activas.