Miguel Ángel Ramírez Pérez mramper0604@g.educaand.es

Explicación - Ejercicio 03 - Control de tráfico

El ejercicio es un sistema de control de tráfico para un carril utilizando ***wait()*** y ***notifyAll()***.
Los vehículos que van en la misma dirección pueden cruzar juntos, pero los que van en dirección contraria deben esperar hasta que el carril quede completamente libre.

El programa Java está compuesto por las siguientes clases:

1. Enum **Direccion**: Define las posibles direcciones (IZDA, DCHA, NINGUNA).
2. Interfaz **CarrilUnico**: Contrato que define los métodos **entrar()** y **salir().**
3. Clase **ControlTrafico**: Monitor que sincroniza el acceso al carril.
4. Clase **Vehiculo**: Representa cada vehículo como un hilo independiente.
5. Clase **Main**: Contiene toda la simulación.

1. Direccion.java

Es un enumerado que define las siguientes 3 constantes:

- **IZDA:** Para vehículos que van hacia la izquierda
- **DCHA:** Para vehículos que van hacia la derecha
- **NINGUNA:** Estado inicial cuando el carril está vacío

2. CarrilUnico.java

Es la interfaz que define el comportamiento que debe tener cualquier implementación de control de tráfico.

- **entrar(Direccion direccion):** Permite que un vehículo intente acceder al carril. Puede bloquearse si hay tráfico.
- **salir(Direccion direccion):** Registra la salida de un vehículo del carril.

Ambos métodos lanzan InterruptedException porque pueden usar wait() y ser interrumpidos.

3. ControlTrafico.java

Esta clase implementa CarrilUnico y actúa como monitor por todos los vehículos.

Variables:

- **direccionActual:** Indica hacia dónde están circulando los vehículos actualmente (IZDA, DCHA O NINGUNA)
- **cochesEnCarril:** Contador de cuántos vehículos hay dentro del carril
- **viaOcupada:** boolean que indica si hay algún vehículo cruzando

Método entrar():
Este método es ***synchronized***, lo que significa que solo un hilo puede ejecutarlo a la vez. Cuando un vehículo quiere entrar:

1. **Comprueba la condición** con un bucle **while.**
2. Si la condición **es verdadera**, llama a ***wait()***. Esto hace tres cosas:

   a. El hilo suelta el candado del monitor
   b. Se queda “dormido” en estado WAITING
   c. Espera hasta que el otro hilo haga ***notifyAll()***

3. **Uso de while en lugar de if:** Después de despertar, el hilo vuelve a comprobar la condición.

4. **Si puede entrar (sale del while):**
   a. Si es el primer vehículo (contador en 0), establece la dirección del carril y marca la vía como ocupada.
   b. Incrementa el contador de vehículos
   c. Imprime mensajes informativos

Método salir():
También es ***synchronized*****.** Cuando un vehículo sale:

1. **Decrementa el contador** de vehículos porque uno acaba de salir.
2. **Comprueba si es el últim:** Si el contador llega a 0, entonces ya no hay vehículos en el carril. En ese momento:
   a. La dirección se pone a NINGUNA
   b. Marca la vía como no ocupada
   c. Llama a ***notifyAll()***: Esto despierta a todos los hilos que estás esperando en el ***wait()***

4. Vehiculo.java

Representa cada vehículo como un hilo que implementa ***Runnable***

Constructor:
Recibe tres parámetros:

- El monitor **ControlTrafico** compartido por todos
- Su dirección (IZDA o DCHA)
- Su nombre identificativo

Método run()
Define el comportamiento dentro del hilo.

1. **Anuncia su llegada** con un mensaje que se muestra por pantalla
2. **Llama a carril.entrar(direccion):** Esta línea es crítica. Si hay vehículos en dirección contraria, el hilo se bloqueará aquí dentro del ***wait()*** hasta que sea su turno
3. **Simula el cruce:** Una vez dentro, usa ***Thread.sleep()*** con un tiempo aleatorio entre 1 y 3 segundos para simular que está cruzando el carril.
4. **Llamar a carril.salir(direccion):** Registra su salida. Si es el último vehículo, esto provocará que el monitor despierte a los hilos de la dirección contraria.
5. **Manejo de excepciones:** Todo está dentro de un bloque try-catch. Si el hilo es interrumpido mientras duerme o espera, captura la **InterruptedException**, imprime un mensaje de error y vuelve a marcar el hilo como interrumpido.

5. Main.java

Proceso de ejecución:

1. **Crea el monitor:** Instancia un solo objeto ***ControlTrafico*** que será compartido por todos los vehículos. Esto es esencial para la sincronización.
2. **Crea la lista de hilos:** Usa un ***ArrayList*** para almacenar todos los hilos que representan los vehículos
3. **Genera los vehículos:** En un bucle crea los 10 vehículos los impares van a la IZDA y los pares hacia la DCHA. Cada vehículo se crea como un objeto **Vehiculo (Runnable)** que luego se pasa al constructor de Thread.
4. **Pausa inicial:** Espera 1 segundo antes de empezar para que la salida sea más clara.
5. **Incia todos los hilos:** Con un bucle llama a start() en cada hilo. Cada vehículo empieza a ejecutar su método **run()**.
6. **Espera la finalización:** Usa ***join()*** a cada hilo. Esto hace que el hilo main se bloquee esperando a que cada vehículo termine el recorrido. Solo cuando el último ***join()*** finaliza, main continúa.
7. **Mensaje final:** Imprime que todos los vehículos se han cruzado