Programación multihilo

Estaciones eléctricas

Índice:

Introducción

Para este ejercicio se usarán **semáforos**.

El concepto de **Semáforo** (`java.util.concurrent.Semaphore`) es fundamental en la programación concurrente y es excelente para un ejercicio de dificultad media-alta, ya que permite controlar el acceso a un número limitado de recursos (no solo uno, como en la exclusión mutua simple).

Aquí tenemos un ejercicio detallado que **simula un escenario de recursos limitados**.

Gestión de estaciones eléctricas

Se quiere desarrollar un software para simular un punto de carga de vehículos eléctricos donde el **número de estaciones de carga es limitado**. La intención es usar los mecanismos necesarios para controlar la concurrencia y asegurar que sólo un número fijo de vehículos puedan cargar al mismo tiempo.

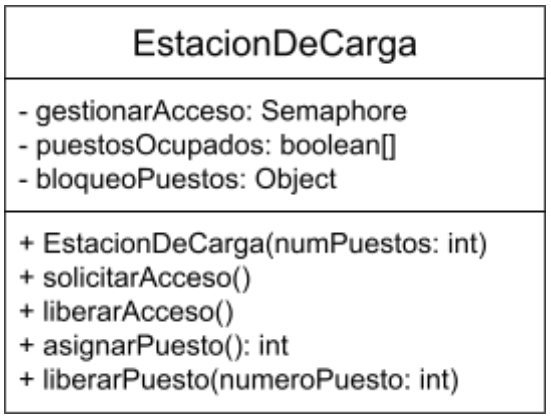
Un parking dispone de una zona con **5 estaciones de carga eléctrica** disponibles para vehículos.

- El tiempo de carga es variable.
- Los vehículos llegan de forma asíncrona (como hilos) y deben esperar si no hay estaciones libres.

Se debe tener en cuenta lo siguiente.

EstacionDeCarga

Esta clase se define de la forma siguiente:



Donde el constructor inicializará los valores, rellenando todos los puestos ocupados con

`false` (todos están libres).

`solicitarAcceso()`
Eleveará una `InterruptedException`.

Solicita permiso para conectarse mostrando el mensaje:

```
Vehículo LLEGA. Esperando permiso del Semáforo...
```

Cuando consiga acceder a un punto de carga mostrará lo siguiente:

```
Permiso CONCEDIDO. Buscando puesto…
```

`liberarAcceso()`
Libera el punto de carga y muestra el siguiente mensaje:

```
Permiso LIBERADO. Puestos disponibles: <nº puntos disponibles>"
```

`asignarPuesto(): int`
Asigna un puesto libre.

⚠️ Hay que implementar este método con cuidado para evitar que dos hilos lean y asignen el mismo puesto simultáneamente. Para ello puede usarse el objeto `bloqueoPuestos`.

Se recorrerá la lista de puntos de carga hasta encontrar uno libre (`false`) y se ocupará (se pondrá a `true`).

⚠️ Los puestos se enumeran **del 1 al 5**, no del 0 al 4, por lo que deben devolver un nº válido. Si no se encuentra ninguno, devolver **`-1`**.

`liberarPuesto(numeroPuesto: int)`
Libera un punto de carga.

⚠️ Hay que implementar este método con cuidado para evitar que dos hilos lean y asignen el mismo puesto simultáneamente. Para ello puede usarse el objeto `bloqueoPuestos`.

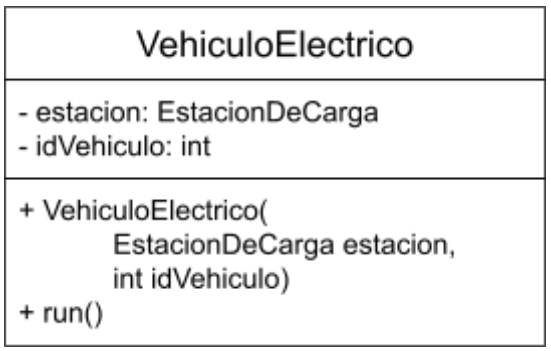
⚠️ Recuerda que se introducirá un nº entre 1 y 5, no entre 0 y 4.

⚠️ Valida el número de entrada por si es una posición incorrecta.

VehiculoElectrico

Esta clase representa a cualquier vehículo eléctrico que quiera hacer uso de un punto de carga. Pueden ser muchos, por lo que se implementará como un hilo.

Se define de la forma siguiente:



Al ejecutar cada hilo deben realizarse los siguientes pasos:

1. Solicitar acceso a un punto de carga.
2. Obtener el puesto asignado.
    a. Si se ha obtenido algún punto, mostrar el siguiente mensaje:

    ```
    <Nombre vehículo> - ASIGNADO al Puesto <nº puesto>.
    ```

    b. Si no, si obtenemos un -1 es que algo salió mal con la lógica del puesto por lo que liberamos el semáforo y terminamos.
3. Simular que el vehículo está cargando. Para ello, esperamos **entre 1 a 3 segundos**. Primero mostramos un mensaje:

    ```
    <Nombre vehículo> - INICIA Carga. Duración: XX sg.
    ```

    Y al transcurrir el tiempo indicado se mostrará:

    ```
    <Nombre vehículo> - Carga FINALIZADA
    ```
4. Por último se debe **garantizar que se liberan el puesto y el acceso**.

Main

Crea un Main.java donde se puedan realizar pruebas para comprobar que nuestro programa funciona correctamente.