Memoria Practica (|) Threads

Programación de Servicios y Procesos

Que

Se desea realizar una clase llamada Parking que reciba el número de plazas del parking y el número de coches que tiene el sistema.

Se deben crear tantos threads como coches hayan. El parking dispondrá de una única entrada y una única salida. En la entrada de vehículos habrá un dispositivo de control que permitia o oimpida el acceso de los mismos al parking, dependiendo del estado actual del mismo (plazas de aparcamiento disponibles).

Los tiempos de espera de los vehículos dentro del párking son aleatorios. En el momento en el que un vehículo sale del párking, notifica al dispositivo de control el número de plaza que tenía asignada y se libera la plaza que estuviera ocupando, quedando así estas nuevamente disponibles. Un vehículo que ha salido del párking esperará un tiempo aleatorio para volver a entrar nuevamente en el mismo. Por tanto, los vehículos estarán entrando y saliendo indefinidamente del párking.

Es importante que se diseñar el programa de tal forma que se asegure que, antes o después, un vehículo que permanece esperando a la entrada del párking entrará en el mismo (no se produzca inanición).

Para que

Esta actividad tiene como objetivo proporcionar un aprendizaje básico sobre la utilización de hilos en Java. A través de esta práctica, exploramos de manera sencilla cómo trabajar con threads y comprender su funcionamiento. Esta práctica es útil, ya que nos brinda una introducción básica a cómo los hilos operan y las diferentes formas en que pueden ser utilizados y aplicados.

Pseudocodigo

Clase Parking:

```
    Atributos:
        - plazas (Array de enteros)
        - numeroPlazas (Entero)

    Método Parking(numeroPlazas: Entero):
        this.numeroPlazas = numeroPlazas
        this.plazas = Nuevo Array de enteros de tamaño numeroPlazas
        Poner los plazas a 0

    Método aparcar(cocheID: Entero): Entero
        Mientras numeroPlazas sea igual a 0:
            Intentar:
                Mostrar el Coche cocheID esperando para entrar al parking.
                Esperar
            Capturar InterruptedException

        llamo a buscarPlazaLibre()
        plazas[plaza] = cocheID
        numeroPlazas--

        Mostrar ENTRADA: Coche cocheID aparca en plaza numero de la plaza
        Mostrar Plazas libres: numeroPlazas
        Mostrar Parking: llamo al toString
        Devolver plaza

    Método desaparcar(plaza: Entero, cocheID: Entero)
        plazas[plaza] = 0
        numeroPlazas++

        Mostrar SALIDA: Coche cocheID sale de plaza numero de la plaza
        Mostrar Plazas libres: " + numeroPlazas
        Mostrar Parking: llamo al toString
        Notificar()

    Método buscarPlazaLibre(): Entero
        Para i desde 0 hasta Tamaño de plazas - 1:
            Si plazas[i] == 0:
                Devolver i
        Devolver -1

    Método toString(): Cadena
        resultado = ""
        Para i desde 0 hasta Tamaño de plazas - 1:
            resultado += [cocheID]
        Devolver resultado
```

Clase Coche:

```
    Entrada: cocheID (entero), parking (objeto de la clase Parking)

    Repetir indefinidamente:
        min = 1000
        max = 5000
        Generar un número aleatorio entre min y max y almacenarlo en la variable
random
        Dormir el hilo actual durante random milisegundos
        Capturar y manejar excepción InterruptedException si ocurre

        Llamar al método aparcar del objeto parking, pasando cocheID como argumento y
almacenar el resultado en la variable plaza
        Generar un número aleatorio entre min y max y almacenarlo en la variable
random
        Dormir el hilo actual durante random milisegundos
        Capturar y manejar excepción InterruptedException si ocurre

        Llamar al método desaparcar del objeto parking, pasando plaza y cocheID como
argumentos
Fin de la Clase Coche
```

Clase principal Practica_01:

```
    Crear un objeto ReadClient llamado rc

    Solicitar al usuario el número de coches
    Leer y almacenar el número en la variable numeroCoches

    Solicitar al usuario el número de plazas
    Leer y almacenar el número en la variable numeroPlazas

    Crear un objeto Parking llamado parking, con el número de plazas igual a
numeroPlazas

    Para i desde 1 hasta numeroCoches:
        Crear un objeto Coche llamado coche, pasando i y parking como argumentos
        Iniciar el hilo del coche
Fin de la Clase principal
```

Como

Este programa consta de tres clases que simulan la gestión de un estacionamiento con múltiples coches. A continuación, se detalla la funcionalidad de cada una de las clases:

Parking

Comenzando por mi clase "Parking", representa el estacionamiento y gestiona la disponibilidad de plazas para los coches. Esta tiene dos variables generales: el número de plazas del parking y un array de enteros que me servirá para saber qué plazas están ocupadas por los coches.

El constructor de "Parking" recibe como parámetro el número de plazas. Luego, asigna al array de plazas la longitud correspondiente al número de plazas y establece cada una de las posiciones del array en 0.

Tengo dos funciones synchronized:

"aparcar": Esta función es pública y devuelve la plaza donde ha aparcado el coche. Recibe como parámetro el ID de un coche, comprueba si el coche tiene una plaza donde aparcar y, si la hay, aparca. Si no hay plazas disponibles, el hilo espera hasta que pueda aparcar. Para aparcar, asigna el ID a una posición del array que esté disponible utilizando la función "buscarPlazaLibre()" e imprime que el coche ha podido aparcar. Finalmente, devuelve la plaza donde ha aparcado.

"salir()": Esta función no devuelve nada y tiene como parámetros de entrada la plaza donde está aparcado el coche y su ID. Guarda en la posición donde estaba aparcado el coche el valor 0, informa por pantalla y, además, llama a la función "notify()" que despierta a otros coches en espera.

La función "buscarPlazaLibre()" comprueba si hay alguna posición igual a 0 en el array y devuelve su posiciónm sino devuelve -1.

Finalmente, la función "toString()" imprime el array.

Coche

Mi clase "Coche" tiene dos variables generales: "CocheID" (la identificación del vehículo) y "Parking" al que está asignado, con un constructor que recibe ambas variables. Además, esta clase extiende "Thread" para poder utilizar su única función "run()".

En la función "run" heredada de "Thread", declaro dos variables que son simplemente valores límite para la función "Math.random()". Luego, creo un bucle while infinito. Dentro de este bucle, primero hago que el hilo espere un tiempo aleatorio definido anteriormente utilizando la función "sleep()", y gestiono cualquier posible error que pueda ocurrir durante la espera, para que en caso de error, el hilo pueda detenerse adecuadamente.

Después de la espera, llamo a la función "aparcar" del objeto "Parking", pasándole el ID del coche, y guardo su valor de retorno en una variable. Esto me permite luego llamar a la función "salir". Antes de llamar a la función "salir", espero nuevamente un tiempo aleatorio y, en esta ocasión, llamo a la función "salir()", pasando como parámetros la plaza y el ID del vehículo.

Main

En mi clase principal "Practica_01", que contiene la función principal, solicito al usuario el número de coches y el número de plazas utilizando las funciones del objeto de tipo "ReadClient", como se explica en las prácticas de Acceso a Datos.

Luego, creo un objeto "Parking" con un número máximo de plazas asignadas por el usuario.

Finalmente, creo un bucle "for" con un máximo igual al número de coches, donde en cada iteración creo un objeto de tipo "Coche" y llamo a su función "start". De esta manera, estoy creando y lanzando un hilo por cada coche.

Conclusión

Esta práctica ha sido útil para aprender a programar con hilos en Java. También he explorado distintas formas de hacerlo en Internet, como el uso de semáforos y la interfaz "Runnable". Sin embargo, al final me decidí por esta forma porque la encontré la más sencilla y fácil de usar.