Universidad Nacional del Comahue
Facultad de Informática
2023

Programación Concurrente
Trabajo Práctico Obligatorio FINAL
Sistema de Aeropuerto[1]

María Elisa Méndez Cares
Profesorado en Informática
Legajo 61921
maria.mendez@est.fi.uncoma.edu.ar

Docente: Silvia Amaro

[1] https://github.com/merimendez147/Aeropuerto2023/tree/BlockingQueue/src/main/java/com/mycompany/aeropuerto

Introducción

El presente informe detalla la simulación del comportamiento del Aeropuerto "VIAJE BONITO", donde se ha implementado un programa multihilo que representa el proceso de los pasajeros desde su llegada al aeropuerto hasta el momento en que abordan el avión. La simulación incluye operaciones como la generación de reservas aleatorias para cada pasajero, atención en puestos de información, colas de check-in, transporte a las terminales, visitas a FreeShops, esperas y procesos de embarque.

Para lograr una simulación concurrente y sincronizada, se han utilizado diferentes mecanismos de sincronización, como bloques y métodos sincronizados, semáforos, colas bloqueantes, barreras y CountDownLatch. Cada clase involucrada en la simulación cumple un rol específico, como gestionar las reservas, la atención en puestos de check-in, el transporte, las salas de embarque y los FreeShops.

En este informe, se explicarán detalladamente las funcionalidades de cada clase, destacando los mecanismos de sincronización empleados para evitar problemas de concurrencia y garantizar el comportamiento correcto y coordinado de los hilos en el ambiente multithread. Para proporcionar un mejor entendimiento del programa se realizaron diagramas de secuencia UML de las principales operaciones.

Comportamiento del Aeropuerto

El programa simula el funcionamiento del Aeropuerto "VIAJE BONITO”, desde que el pasajero llega al mismo para tomar su vuelo hasta que sube al avión. Algunas características de la simulación son:

- Reservas aleatorias: Se generan reservas aleatorias para cada pasajero.
- Atención en los puestos de informes: Los pasajeros llegan al aeropuerto y solicitan atención en los puestos de información. Allí, se les asigna un número de puesto de check-in.
- Colas de Check-in: Los pasajeros hacen cola en los puestos de check-in y son atendidos secuencialmente.
- Transporte a las Terminales: Los pasajeros utilizan el PeopleMover para ser transportados desde la zona de check-in hasta las terminales de embarque.
- Salas de Embarque y FreeShops: Los pasajeros pueden decidir visitar los FreeShops antes de embarcar. Algunos pasajeros eligen comprar en los FreeShops, mientras que

otros optan por no ingresar o no pueden hacerlo debido a que su vuelo está embarcando.

- Embarque: Los pasajeros esperan el momento de embarcar y se embarcan en sus respectivos vuelos.
- Finalización de Embarques: Los embarques para todas las aerolíneas se cierran después de que todos los pasajeros han embarcado.

Clases

Para simular el funcionamiento del Aeropuerto se implementaron las siguientes clases:

1. La clase Aeropuerto simula la ejecución del aeropuerto para un número específico de pasajeros y aerolíneas, y operaciones relacionadas, como la gestión de vuelos, informes, check-in, transporte y salas de embarque. Crea instancias de las clases necesarias, inicia los procesos de atención en los puestos de check-in y el comportamiento de los pasajeros, y finaliza los embarques para todas las aerolíneas.
2. La clase Pasajero implementa la interfaz Runnable de Java. Cada instancia de la clase Pasajero representa un pasajero que llega a un aeropuerto y sigue una serie de pasos para embarcarse en un vuelo:
    a. Consultar el puesto donde debe hacer el check-in.
    b. Realizar el check-in y obtener la terminal correspondiente al vuelo.
    c. Subir al transporte para ir a la sala de embarque de la terminal.
    d. Visitar los FreeShops antes de embarcar. Algunos pasajeros eligen comprar en los FreeShops, mientras que otros optan por no ingresar o no pueden hacerlo debido a que su vuelo está embarcando.
    e. Esperar el momento de embarcar y embarcarse en sus respectivos vuelos.
3. La clase Vuelos se encarga de coordinar el proceso de reservas y embarque para las tres aerolíneas del aeropuerto. Utiliza mecanismos de sincronización como CountDownLatch y Semaphore, y una estructura de datos pila para las reservas. Además, utiliza un Timer para programar el inicio del embarque de cada vuelo en momentos específicos, simulando así el proceso de embarque de los pasajeros.

4. La clase Reservas es una implementación de una pila que se utiliza para almacenar las reservas de vuelos. Utiliza métodos sincronizados para evitar problemas de concurrencia al insertar y eliminar reservas en la pila.
5. La clase Reserva representa una reserva de vuelo realizada y almacena información como el nombre de la aerolínea, la terminal de embarque y el número de puesto de check-in asociados a dicha reserva.
6. La clase GestorInformes es responsable de gestionar el puesto de informes disponible en el aeropuerto y proporcionar atención a los pasajeros que llegan para obtener información sobre sus vuelos y el check-in. El puesto de informes es un recurso compartido por todos los pasajeros por lo que se utiliza métodos sincronizados para lograr la exclusión mutua y el mecanismo de espera y notificación para lograr la cooperación entre los pasajeros.
7. La clase GestorCheckin se encarga de gestionar los puestos de check-in en el aeropuerto y coordinar el proceso de check-in para los pasajeros. Los pasajeros esperan en colas implementadas con BlockingQueue, y los puestos de check-in los atienden uno por uno en orden de llegada. Se utilizan los métodos suspend() y resume() para suspender y reanudar temporalmente a los pasajeros mientras esperan ser atendidos.
8. La clase AtencionCheckin es una implementación de la interfaz Runnable de Java. Esta clase representa la tarea de atención en un puesto de check-in específico del aeropuerto. Cada puesto de check-in tiene su propia instancia de AtencionCheckin que atiende a los pasajeros.
9. La clase GestorTransporte implementa un mecanismo de transporte (“people mover/transporteATerminal”) para mover a los pasajeros a las diferentes terminales del aeropuerto. Utiliza semáforos y barreras cíclicas (CyclicBarrier) para coordinar el acceso al transporte y asegurar que los pasajeros bajen del tren en el orden correcto. También protege el acceso a las variables compartidas con semáforos para evitar problemas de concurrencia.
10. La clase GestorSalasEmbarque se encarga de coordinar la atención en los free-shops de las diferentes terminales del aeropuerto, permitiendo a los pasajeros ingresar, realizar compras y salir del free-shop de acuerdo con el tiempo disponible antes del embarque. También se encarga de coordinar el proceso de embarque de los pasajeros cuando se inicia el embarque de los vuelos.

11. La clase GestorFreeShop implementa los mecanismos para controlar el acceso de los pasajeros al Free Shop y su salida, así como para gestionar los pagos en las cajas. Utiliza semáforos para coordinar el acceso de los pasajeros al Free Shop y a las cajas de pago.
12. La clase Cajero es una implementación de un hilo (Runnable) que representa el comportamiento de un cajero en el Free Shop del aeropuerto.

Mecanismos de Sincronización

El aeropuerto es un entorno de programación multithread, varios hilos (pasajeros, atención de puestos de Check-in y cajeros del Free-Shop) se ejecutan simultáneamente y compiten por los mismos recursos, lo que puede dar lugar a problemas como condiciones de carrera, bloqueos o inconsistencias en los datos. Algunos de los mecanismos de sincronización utilizados para resolver estos problemas fueron:

- Métodos sincronizados en GestorInformes: Los métodos solicitarAtencion() y consultarPuesto() están declarados como synchronized, lo que garantiza que solo un hilo pasajero pueda acceder a estos métodos a la vez. Esto garantiza que solo un pasajero sea atendido en cada puesto de información, es decir, se logra la exclusión mutua en el puesto de informes.
- Bloque sincronizado en la clase Vuelos: el método embarcar() utiliza un bloque sincronizado con el objeto partida, que es el CountDownLatch asociado a la aerolínea. Esto asegura que solo un hilo a la vez pueda acceder y modificar el contador de partida.
- Bloque sincronizado en GestorCheckin: el método atenderPuestoCheckin() ejecuta las operaciones sacar de la cola de Check-in un hilo pasajero y reanudar su ejecucion asegurando la exclusión mutua y como si fueran atómicas.
- BlockingQueue en GestorCheckin: Se utiliza un arreglo de colas bloqueantes colaCheckin para manejar las colas de check-in en cada puesto. Los pasajeros hacen cola utilizando el método hacerChecking(), y son atendidos utilizando el método atenderPuestoCheckin(). Las colas bloqueantes aseguran que los pasajeros sean atendidos en el orden en que llegan a cada puesto.
- CyclicBarrier en GestorTransporte: Se utilizan las barreras cíclicas inicioTrayecto y finTrayecto para sincronizar la partida y el regreso del transporte PeopleMover

desde la zona de check-in hasta la última terminal. La barrera inicioTrayecto se inicializa con la capacidad del PeopleMover, y los pasajeros esperan en ella hasta que el PeopleMover está lleno y listo para salir. Una vez que el PeopleMover llega a las terminales espera hasta que todos los pasajeros hayan bajado en la terminal correspondiente para volver vacío.

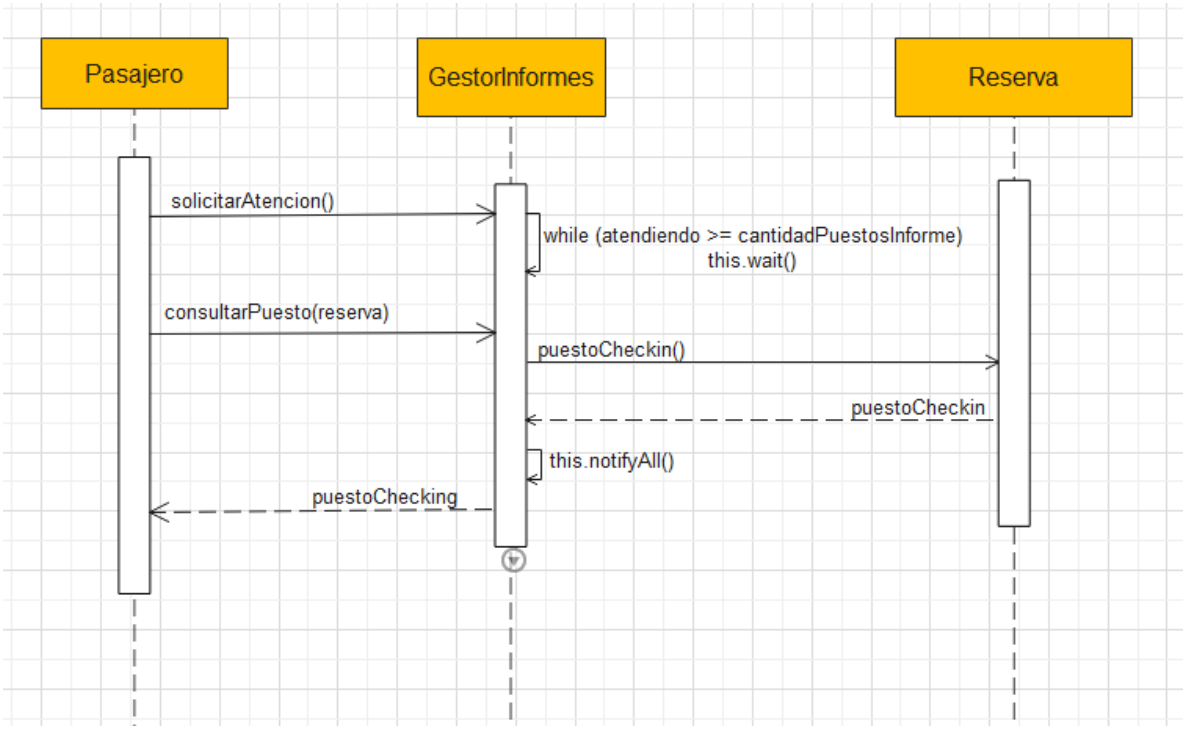
- Semaphore y Mutex en GestorTransporte: Se utilizan los semáforos semTrenParado y mutexContador para controlar el acceso al tren y la manipulación del contador de pasajeros en cada terminal. El semáforo semTrenParado asegura que solo se permita que la capacidad del tren suba a bordo, mientras que el mutex mutexContador protege la modificación de los contadores de pasajeros que bajan en cada terminal.
- Semaphore en GestorFreeShop: Los semáforos ingresarFreeShop, pagarCaja1, esperarCaja1, pagarCaja2 y esperarCaja2 se utilizan para controlar el acceso a los FreeShops y las cajas de pago. El semáforo ingresarFreeShop limita la cantidad de pasajeros que pueden ingresar al FreeShop al mismo tiempo. Los semáforos pagarCaja1 y pagarCaja2 permiten que solo un cajero atienda a un pasajero en cada caja, mientras que los semáforos esperarCaja1 y esperarCaja2 aseguran que los pasajeros esperen a ser atendidos por el cajero antes de realizar el pago.
- Semaphore en GestorSalasEmbarque: El semáforo mutexVueloEmbarcando se utiliza para coordinar el acceso a la información sobre si un vuelo está embarcando o no. Esto evita que los pasajeros intenten embarcar al vuelo antes de tiempo.
- CountDownLatch en Vuelos: Este mecanismo se utiliza para coordinar el inicio del proceso de embarque de los pasajeros en cada aerolínea. Cuando una aerolínea inicia el proceso de embarque, cada pasajero hace un countDown() cuando están listos para embarcar en sus respectivas terminales. Una vez que la cuenta llega a cero, el embarque puede comenzar, y los pasajeros de esa aerolínea específica pueden proceder a embarcar.

Diagramas de Secuencia

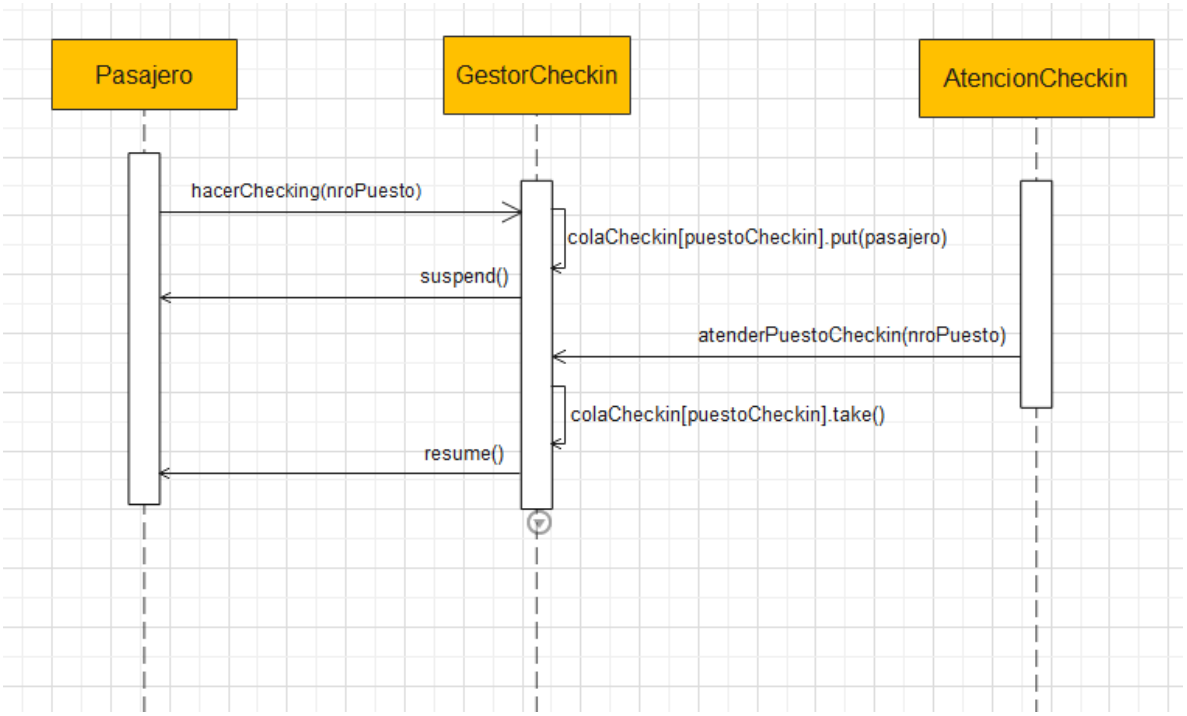
A continuación se presentan los diagramas de secuencias de las operaciones que realiza un pasajero: Consultar el puesto donde debe hacer el check-in; Realizar el check-in y obtener la terminal correspondiente al vuelo; Subir al transporte para ir a la sala

de embarque de la terminal; Visitar los FreeShops antes de embarcar. Comprar en los FreeShops; Esperar el momento de embarcar y embarcarse en su respectivo vuelo.

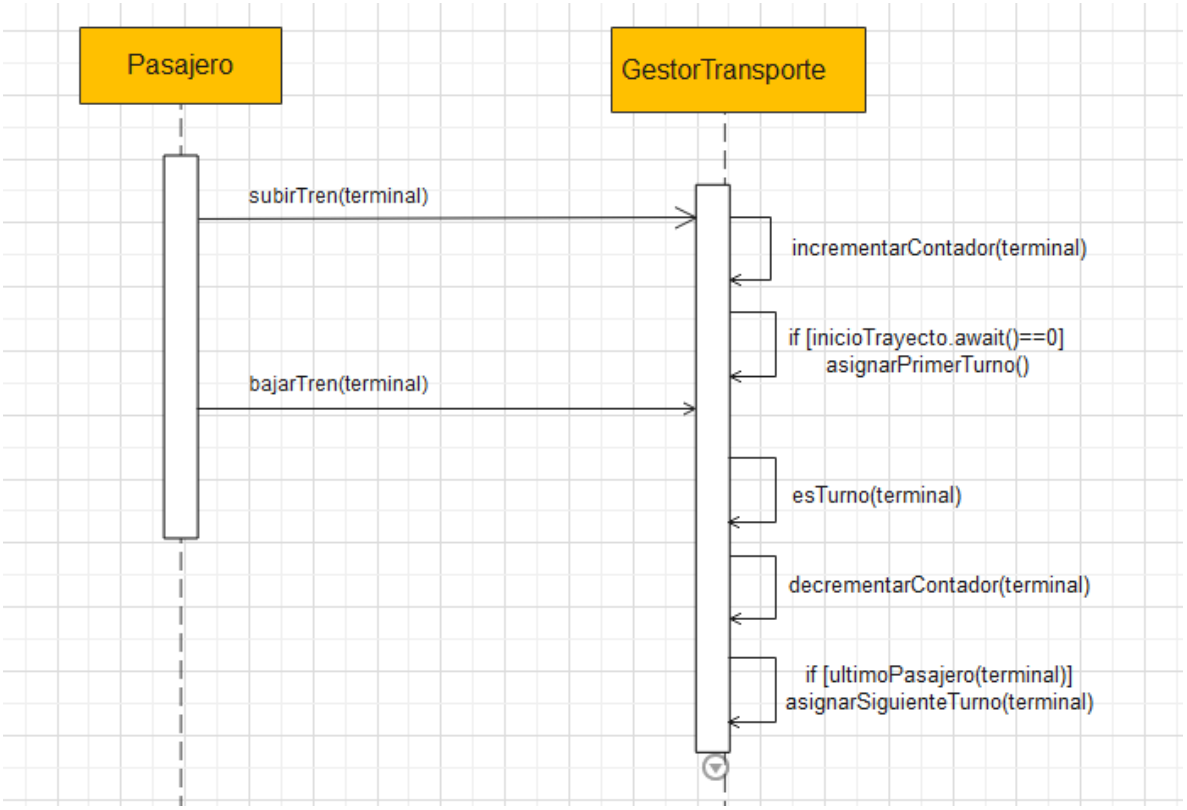
- Consultar el puesto donde debe hacer el check-in: El pasajero utiliza la clase GestorInformes para realizar esta operación. La clase GestorInformes es un Monitor, es decir, es una estructura de datos con todos los métodos sincronizados y los atributos privados. Utiliza la combinación de métodos sincronizados, solicitarAtencion() y consultarPuesto(reserva), para lograr la exclusión mutua y el mecanismo de espera (wait) y notificación (notifyAll) para lograr la cooperación entre los hilos.



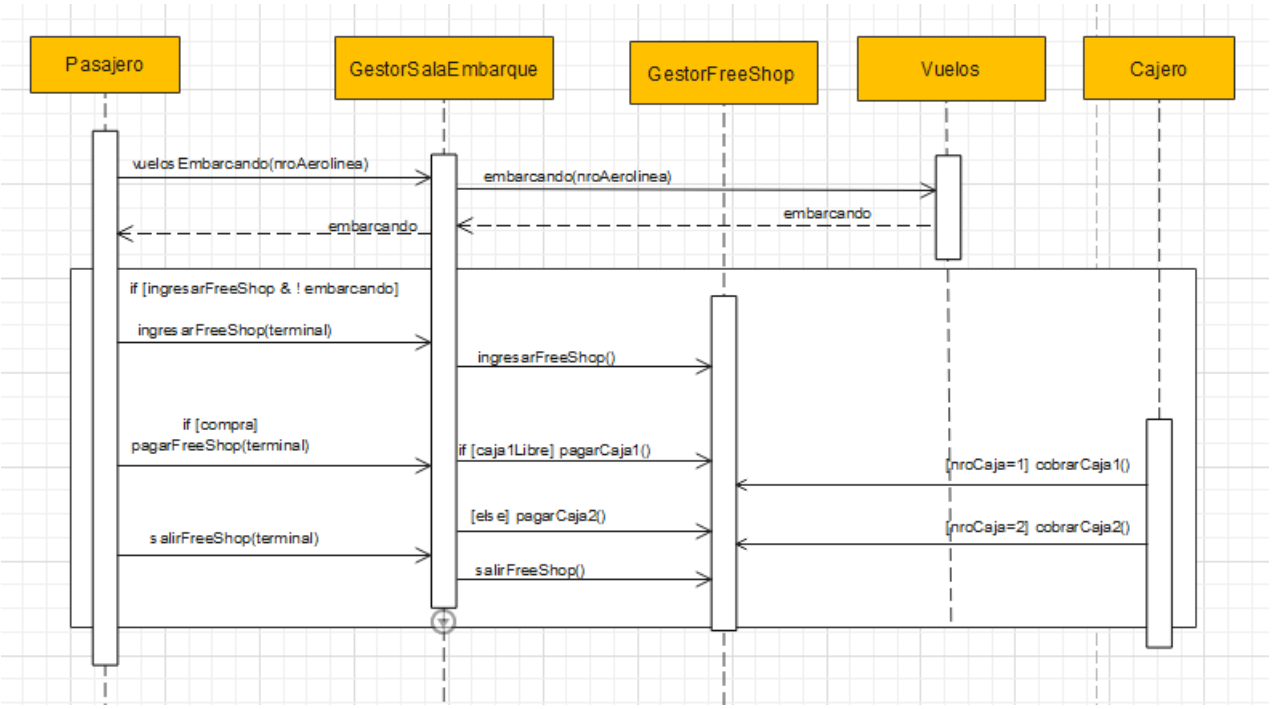
- Realizar el check-in y obtener la terminal correspondiente al vuelo: el pasajero usa la clase GestorCheckin para hacer el check-in, esta clase también es usada por los hilos encargados de la atención de cada puesto de check-in. El pasajero espera en la colaCheckin implementada con BlockingQueue y AtencionCheckin del puesto correspondiente los atienden uno por uno. Además se utilizan los métodos suspend() y resume() para suspender y reanudar temporalmente a los pasajeros mientras esperan ser atendidos.



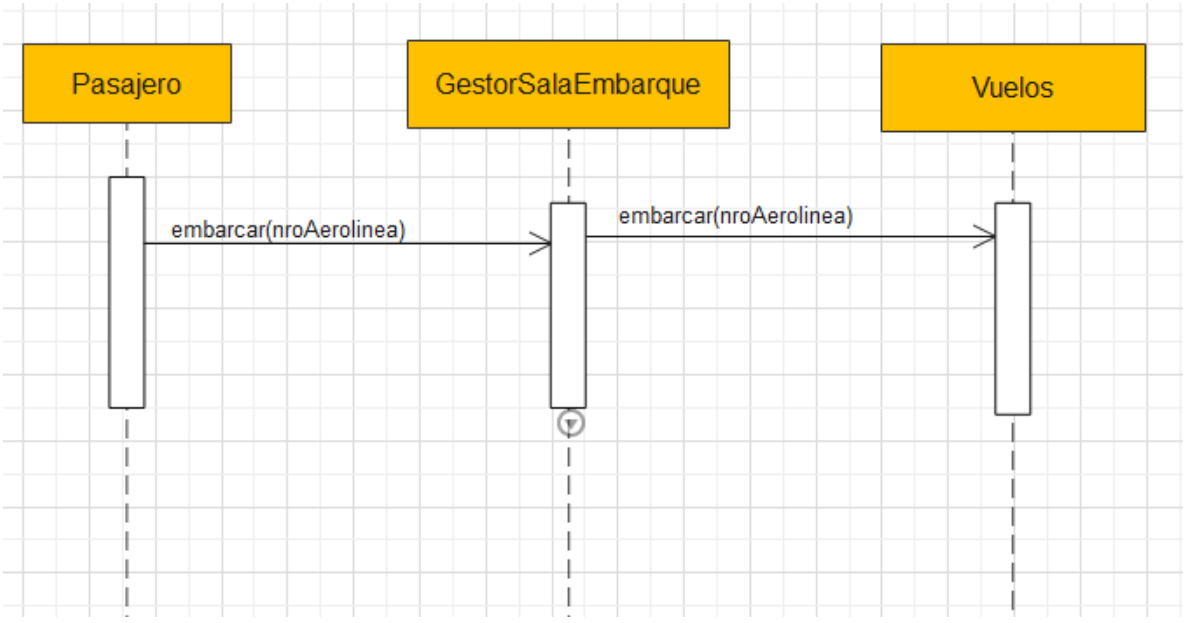
- Subir al transporte para ir a la sala de embarque de la terminal: El pasajero usa el método subirTren(terminal) para subir al tren indicando la terminal de bajada. Este método de la clase GestorTransporte utiliza un semáforo para asegurarse de que el tren esté detenido antes de abordar. Luego, incrementa el contador de pasajeros que bajan en cada terminal. Para saber cuándo el tren está lleno y listo para partir utiliza la barrera cíclica inicioTrayecto. El método bajarTren(terminal) se usa para bajar del tren en una terminal específica. El método contiene un semáforo (turnoBajarTren) para asegurarse de que los pasajeros bajen del tren en el orden correcto y no haya problemas de concurrencia. Cuando el tren está vacío, utiliza la barrera cíclica finTrayecto para coordinar el regreso del tren a su estado inicial.



- Visitar los FreeShops antes de embarcar. Comprar en los FreeShops: Si el pasajero decide ingresar al FreeShop y el vuelo aún no está embarcando utiliza la clase GestorSalaEmbarque para ingresar al FreeShop, si decide comprar en el FreeShop, realizar la compra y pagar. Luego sale del FreeShop. La clase GestorSalaEmbarque controla el acceso a los gestores de FreeShop asociados a cada terminal. La clase GestorFreeShop utiliza semáforos para gestionar la entrada y salida de los pasajeros a la zona de Free Shop de una terminal específica, así como la gestión de los pagos en las cajas. Primero, si hay espacio disponible (permisos en el semáforo ingresarFreeShop), el pasajero ingresa; de lo contrario, esperará hasta que haya espacio disponible. Al salir, libera un permiso en el semáforo ingresarFreeShop para indicar que ha salido y deja lugar disponible para que ingrese otro pasajero. Si decide comprar, los semáforos pagarCaja1 y esperarCaja1 se utilizan como sincronización tipo “rendezvous”, en la cual el hilo Pasajero y el hilo Cajero se esperan para poder continuar. Ídem con los semáforos pagarCaja2 y esperarCaja2.



- Esperar el momento de embarcar y embarcarse en su respectivo vuelo: el método embarcar(nroAerolinea) de la clase Vuelos se encarga de permitir que un pasajero embarque en el vuelo de la aerolínea especificada. Para ello, se utiliza un Semaphore para habilitar el embarque y un CountDownLatch para que cuando todos los pasajeros embarquen el vuelo cierre el proceso de embarque. La clase Vuelos tiene declarado un arreglo de CountDownLatch, donde cada elemento representa un vuelo y se utiliza para sincronizar el inicio del embarque de cada vuelo cuando se alcanza el número de pasajeros necesarios para ese vuelo.



Conclusión

Esperamos que en la simulación del comportamiento del Aeropuerto "VIAJE BONITO" hayamos implementado de manera adecuada los mecanismos de sincronización y la coordinación de las clases involucradas. Cada una de las clases desempeña un papel específico en el proceso de los pasajeros, desde su llegada al aeropuerto hasta el embarque en sus respectivos vuelos.

Creemos que los mecanismos de sincronización, como bloques y métodos sincronizados, semáforos, colas bloqueantes, barreras y CountDownLatch, han permitido resolver los desafíos de concurrencia que se presentan en un ambiente multithread, evitando condiciones de carrera, bloqueos y asegurando la cooperación adecuada entre los hilos.

Con esta simulación, creemos haber logrado ofrecer una representación realista del funcionamiento de un aeropuerto, considerando aspectos importantes como la atención en puestos de información, colas de check-in, transporte a las terminales y visitas a FreeShops, proporcionando una experiencia fluida y coordinada para los pasajeros.