UNIVERSIDAD NACIONAL DE CÓRDOBA
FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES

TRABAJO PRÁCTICO N1 DE PROGRAMACIÓN CONCURRENTE - 2024

Tema: Sistema de reservas de vuelo

Profesor: *Ing. Luis Orlando Ventre ; Ing Mauricio Ludemann*
Fecha: ***7/05/2024***

Integrantes:	**DNI:**
Castillo Darío Alberto	41712130
Mangin Ventura, Sofia Virginia	38150181
Suarez, Joaquin Gabriel	41600609
Condori, Carlos Javier	41712087
Brullo, Maximiliano	41264613

INTRODUCCIÓN:

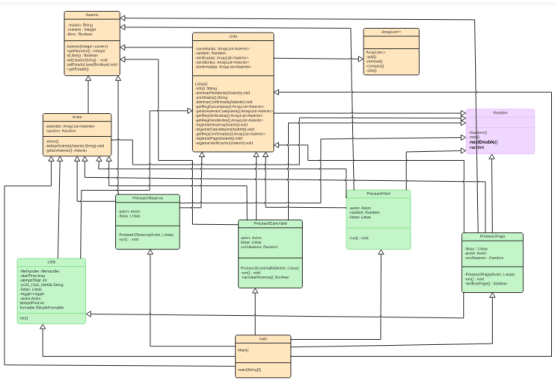
El desarrollo de un sistema para gestionar las reservas de asientos de un vuelo de manera concurrente implica una serie de desafíos relacionados con la coordinación de múltiples procesos en un entorno compartido. En este informe, documentamos nuestro enfoque para diseñar e implementar esta funcionalidad, centrándonos en la representación y manipulación de datos de reservas en un contexto de programación concurrente.

Iniciamos nuestro trabajo mediante la visualización gráfica de las clases y objetos involucrados en el problema, identificando las relaciones y conexiones entre ellos. Posteriormente, nos dedicamos a la creación del diagrama de clases, utilizando inicialmente el software BlueJ para una representación más simple.

Comenzamos el desarrollo del código estableciendo un repositorio en GitHub para colaborar de manera efectiva en el equipo. El primer proyecto tuvo más clases que la versión definitiva, lo que nos llevó a buscar una manera más eficiente de completar el trabajo. Finalmente, utilizamos el software IntelliJ para realizar de manera conjunta la edición del código con todos los integrantes, mejorando así la colaboración y la eficacia en el desarrollo del sistema de reservas de vuelo concurrente.

DIAGRAMA DE CLASES UML

El presente diagrama de clases UML proporciona una representación visual detallada de las entidades y relaciones clave en nuestro sistema de reservas de vuelo. Mediante este diagrama, hemos identificado las clases principales y sus interacciones, lo cual es esencial para comprender la estructura y el diseño del programa.



Avión:
La clase Avión modela un avión con una lista de asientos (Asiento). Al crear una instancia de Avión, se inicializan 186 asientos. Esta clase proporciona métodos para obtener un asiento aleatorio y establecer el estado de un asiento.

Asiento:
La clase Asiento representa un asiento individual en el avión. Cada asiento tiene un número y un estado inicialmente establecido como "disponible". Esta clase proporciona métodos para obtener y establecer el estado del asiento.

Listas:
La clase Listas administra listas de asientos en diferentes estados: pendientes, confirmados, cancelados y verificados. Proporciona métodos para registrar y manipular asientos en estas listas, como registrar reservas, pagos, cancelaciones y verificaciones. También permite obtener un asiento aleatorio de una lista específica.

Las siguientes clases, **Proceso1Reserva**, **Proceso2Pago**, **Proceso3CancValid** y **Proceso4Verif**, implementan la interfaz Runnable y representan procesos concurrentes que interactúan con objetos Lista y un objeto Avión. Cada proceso involucra múltiples hilos y simula operaciones específicas como la reserva, el pago, la cancelación y la verificación de asientos.

LOG:
La clase LOG extiende Thread y se encarga de registrar información del sistema en un archivo de registro (información.log). Este registro incluye información periódica sobre el estado de las listas y la ocupación del avión al finalizar la ejecución del programa.

Main

En la clase Main, se inicia el proceso principal. En el método main, se realiza la inicialización de componentes clave y se inician múltiples hilos para ejecutar los procesos concurrentemente.

Se comienza con la creación de un objeto Avion y Listas, que representan el avión y las listas de asientos respectivamente, necesarios para el funcionamiento del sistema.
Se establece un tiempo de inicio y se define la duración total del proceso en segundos (tiempoTotal) y el intervalo de tiempo para definir cada cuánto tiempo se debe imprimir información (tiempoPrint) en milisegundos.

Se instancia un objeto LOG que se encargará de manejar el registro de información sobre las operaciones en curso, pasando como parámetros las listas de asientos y el avión, el tiempo de inicio, la duración total y el intervalo de impresión.
Se inicia un hilo para el objeto LOG mediante el método start().

Se inician varios hilos para ejecutar los diferentes procesos de reserva, pago, cancelación y verificación de asientos, cada uno representado por las clases Proceso1Reserva, Proceso2Pago, Proceso3CancValid y Proceso4Verif. Cada hilo se inicia con su respectivo proceso y se muestra un mensaje indicando el inicio de cada hilo y el proceso que representa.

Esta estructura permite que los procesos se ejecuten de manera simultánea, gestionando las reservas de asientos de forma concurrente y eficiente en el sistema.

Proceso1Reserva
La clase Proceso1Reserva implementa la interfaz Runnable. Este proceso se encarga de gestionar la reserva de los asientos del avión

Los atributos son “avión”, en el que se realizarán las reservas, y “listas” que representa las listas de asientos y reservas del sistema. Estos atributos son necesarios para acceder a la información del avión y las listas de reservas, lo que permite llevar a cabo las operaciones de reserva de manera adecuada.
En el constructor, recibe el avión y las listas como parámetros para inicializar los atributos de la clase. Al recibir estos objetos como parámetros, la clase Proceso1Reserva puede interactuar con el avión y las listas de manera directa, sin depender de instancias específicas dentro de run().
El método run() utiliza un bucle while() que toma como condición que el último proceso no haya terminado, con una clave creada para dicho propósito. Lo mismo se repite para los demás procesos. Obtiene un asiento disponible del avión y verifica su estado antes de realizar la reserva.
Si el asiento está disponible, se marca como “ocupado” en el avión y se registra la reserva en las listas. La verificación del estado del asiento evita reservar un asiento que ya está ocupado o reservado por otro proceso. La importancia de implementar la interfaz recientemente mencionada, es vital al utilizar hilos separados para cada proceso de

reserva. Con esto, se evitan problemas de concurrencia, ya que que múltiples reservas puedan ser gestionadas simultáneamente sin interferir unas con otras.

Proceso2Pago

Esta clase, representa el pago de las reservas en estado pendiente.
Tiene como atributos un objeto Avion, un objeto Listas y un objeto Random, que se utiliza más adelante para generar números aleatorios.
El constructor, recibe como parámetros los primeros dos objetos, e inicializa los tres.
En el método run() se crea una variable local Asiento para guardar una reserva cualquiera de las pendientes. Luego dicho asiento pasa a ser pagado con un 90% de probabilidad de éxito.Este método simula de manera realista la verificación de pago con una probabilidad de éxito del 90%, reflejando el comportamiento de un sistema de pago real. Si hay éxito, se registra el pago, y en caso de que no, se registra su cancelación.

Proceso3CancValid

La presente clase, representa la cancelación de las reservas previamente pagadas.
Cuenta con los mismos atributos y constructor que la clase anteriormente detallada.
En su método run(), se verifica que no se vaya a tratar con una lista de reservas confirmadas vacía, y posteriormente se utiliza una variable local que alberga un objeto de tipo Asiento de dicha lista.
Luego se verifica que la cancelación sea exitosa. En caso de que sí, tal reserva se quita de la lista actual, y se registra en la lista de canceladas. Tal asiento queda descartado. Pero en caso contrario, la reserva se “reafirma” y el asiento queda chequeado.

Proceso4Verif

La última etapa. Esta clase representa la verificación de las reservas en estado “chequeado”. Mismos atributos, mismo constructor. En su método run(), se encarga de pedir un asiento a la lista de confirmadas, y si su estado es “chequeado”, pasa a la lista de verificadas. Pasando a ser la persona en cuestión un ocupante del avión. Es de suma importancia comprobar que dicho asiento tenga estado “chequeado”, ya que la lista que las contiene(confirmadas) también alberga asientos con estado “ocupado”, lo cual resultaría en un cruzamiento de datos no deseado.

CONFIGURACIÓN DE LOS TIEMPOS DE PROCESO:

Para simular condiciones realistas, se configuraron los tiempos de espera y las demoras utilizando métodos como **synchronized** para garantizar la exclusión mutua y **random** para generar números pseudoaleatorios que simulan diferentes situaciones.

tiempo para proceso 1 reservar:70ms
tiempo para proceso 2 pagar :50 ms
tiempo para proceso 3 cancelar/validar:70ms
tiempo para proceso 4 verificar:60ms

Los tiempos de ejecución para cada proceso fueron elegidos a criterio del grupo.

ANÁLISIS DE TIEMPO DE DEMORA DEL PROGRAMA:

Durante la ejecución del programa, registramos algunos números importantes relacionados con el tiempo de demora y las operaciones realizadas. Estos datos se recopilan en un registro de programa que nos permite analizar el rendimiento y la eficiencia del sistema.

Se puede observar que la cantidad de asientos ocupados al final del programa tiene un patrón bastante regular en su distribución, lo cual puede ser un indicador de que el programa está funcionando de manera estable y predecible en términos de cómo se asignan y liberan los asientos a lo largo del tiempo.

Otro punto en cuestión, es que cabe señalar que mientras más reservas se cancelen, más rápido termina el programa, debido a que menos reservas pasan por el último proceso. Esto se vió en algunas de las ejecuciones donde la ocupación final del avión resultó ser menor al promedio. Mientras más reservas pasen todos los procesos, más tiempo tardará en finalizar el programa.

CONCLUSIÓN:

En conclusión, el desarrollo de este sistema de reservas de vuelo concurrente ha sido . Hemos aprendido a coordinar eficazmente múltiples procesos en un entorno compartido, utilizando técnicas como exclusión mutua y tiempos de espera seteados con criterio para garantizar la coherencia y la eficiencia del sistema

El registro de actividades muestra un progreso significativo en el proceso de verificación en comparación con el proceso de cancelación. A lo largo del período de tiempo analizado, se puede observar un aumento constante en la cantidad de elementos verificados, mientras que la cantidad de elementos cancelados también ha aumentado, pero a un ritmo más lento.

La implementación de múltiples hilos para realizar operaciones concurrentes (como reserva, pago, cancelación y verificación) demuestra una coordinación efectiva entre procesos para gestionar las reservas de manera simultánea a diferencia de que si hubiéramos tenido menos hilos.

El uso de técnicas como synchronized garantiza que cada asiento sea accesible por un solo hilo a la vez, evitando conflictos de reserva simultánea y asegurando la coherencia de los datos.

Los hilos comparten recursos como la matriz de asientos y las listas de reservas. El presente diseño del programa permite que cada hilo manipule estos recursos de manera segura y coherente, pero también reflejando condiciones realistas.

Bibliografía

Material del Aula Virtual de la FCEFyN
Programación Concurrente - José Tomás Palma Mendez
Java Concurrency Cookbook - Javier Fernández González