

GRADO DE INGENIERÍA DE COMPUTADORES



Curso 2023-2024

05953542S - Senespleda Fernández Daniel
06021665N- Noguerales Alburquerque Christian

Tabla de contenido

Análisis de alto nivel.....	3
1. Modelado de Aeropuertos:.....	3
2.- Interfaces Gráficas:.....	3
3. Transporte Terrestre:.....	3
4. Operaciones de Aviones:	3
5. Registro en fichero de texto:	4
6. Gestión de Instalaciones:	4
7. Estrategias de Mantenimiento:.....	4
8. Conclusión:.....	4
Diseño general del sistema:.....	4
Descripción de las clases principales	5
Parte Concurrente	5
Autobús:	5
Avión	6
InterfazServidor:	7
Clase Log:	7
Sistema:.....	8
Aeropuerto:	8
Parte Distribuida.....	15
AeropuertoImpl y AeropuertoRemote:	15
Cliente:	15
InterfazCliente.....	15
Servidor.....	15
Diagramas de clases:	16
Dificultades:	17
Anexo:.....	19

Análisis de alto nivel

La simulación de aeropuertos que hemos diseñado utiliza una serie de elementos interconectados que interactúan para lograr el funcionamiento del intercambio de aviones y pasajeros de dos aeropuertos (Madrid y Barcelona).

1. Modelado de Aeropuertos:

Son los elementos principales de nuestro programa. Ambos están formados por zonas limitadas o no, a través de las cuales pasarán aviones y autobuses para realizar sus correspondientes acciones. Ambos aviones están conectados por unas aerovías, a través de las cuales pasan los aviones para ir de uno a otro.

2.- Interfaces Gráficas:

También tenemos una interfaz gráfica de usuario para visualizar el estado de los aeropuertos y sus zonas y otro para interactuar con el cliente. En este último, el cliente es capaz de visualizar datos como el número de pasajeros en cada aeropuerto, el número de aviones en el Hangar..., además de eso, el cliente será capaz de bloquear y desbloquear las pistas de aterrizaje/despegue.

3. Transporte Terrestre:

Para gestionar el transporte terrestre de pasajeros, hemos modelado autobuses que conectan la ciudad con el aeropuerto y viceversa. Estos autobuses se generan como hilos y tienen un ciclo de vida que incluye la recogida y entrega de pasajeros en ambas direcciones. Generaremos 4000 de estos autobuses de forma escalonada (y a la vez que la generación de aviones), estos autobuses tendrán un ciclo de vida infinito.

4. Operaciones de Aviones:

Se generan 8000 aviones de manera escalonada al inicio del programa para evitar congestiones. Los aviones siguen un ciclo de vida completo, desde el embarque de pasajeros hasta el mantenimiento en taller, con tiempos aleatorios para simular condiciones realistas. Se modelan como hilos para permitir operaciones concurrentes y una simulación precisa del tráfico aéreo, lo que garantiza una representación fiel de las operaciones aeroportuarias. Además, cada cierto número de vuelos, los aviones deben someterse a inspecciones en los talleres de los aeropuertos.

5. Registro en fichero de texto:

Toda la actividad del sistema se registra en un archivo de registro (log) para un análisis posterior. Todo el texto que se introduce en el fichero de texto fue previamente usado para orientarnos en la ejecución del código y comprobar que todo nos funcionaba correctamente, lo cual garantiza la comprensión del fichero de texto a la hora de leerlo.

6. Gestión de Instalaciones:

Zonas Específicas: Cada aeropuerto cuenta con zonas específicas como hangares, talleres, puertas de embarque, pistas, áreas de estacionamiento y áreas de rodaje, diseñadas para realizar funciones específicas y optimizar el flujo de operaciones.

Capacidad Limitada en Algunas Zonas: Algunas áreas, como los talleres, tienen capacidad limitada para el mantenimiento de aviones, lo que garantiza un flujo de trabajo eficiente y evita congestiones.

7. Estrategias de Mantenimiento:

Inspecciones en Taller: Después de cierto número de vuelos, los aviones deben someterse a inspecciones en el taller. Esto asegura el mantenimiento adecuado de las aeronaves y la seguridad de los pasajeros.

Gestión de Colas: Se implementa una estrategia FIFO para gestionar las colas de aviones que esperan inspección en taller, lo que garantiza una distribución justa y equitativa del tiempo de mantenimiento.

8. Conclusión:

En resumen, este proyecto ofrece una simulación detallada y realista del funcionamiento de los aeropuertos de Madrid y Barcelona, así como de la conexión aérea entre ambos. La programación concurrente garantiza un rendimiento eficiente y preciso del sistema.

Diseño general del sistema:

El diseño general del sistema de simulación de los aeropuertos de Barcelona y Madrid consta de dos componentes principales (aparte de los aeropuertos). Estas son los aviones y los autobuses.

Los autobuses transportan pasajeros entre la ciudad y el aeropuerto, mientras que los aviones realizan el embarque, despegue, vuelo, aterrizaje y desembarque de pasajeros (entre otras). Ambos interactúan en las zonas compartidas, algunas son exclusivas de aviones, otras exclusivas de aeropuertos, y en otras intervienen ambos, como en el valor de los pasajeros del aeropuerto, sobre el cual, ambos tipos de hilos realizan modificaciones, por lo que era muy importante asegurar la exclusión mutua de esa zona.

Para asegurar la sincronización en nuestro código, hemos hecho muchos de nuestros métodos `synchronized` para controlar el acceso a estos por parte de los hilos. Por

ejemplo, en los métodos **obtenerPrimeroColaAreaEstacionamiento()**, **actualizarPersonasDentro()**, **entrarHangar()**, **salirHangar()**, ...

Además de esto, para las zonas de nuestros aeropuertos con límite de capacidad como las puertas de embarque, las pistas y el taller, hemos usado los mecanismos de concurrencia aprendidos en clase. Mas concretamente, locks para las puertas de embarque, un Semaphore para el taller (con valor de entrada true, para que fuera un semáforo justo y su cola de espera de adquisición fuera una FIFO) y otro lock para la puerta del taller (para asegurarnos que solo entre 1 avión a la vez). Además de esto, hemos usado la lógica empleada con los monitores de estar comprobando el valor de una variable booleana para ver si el hilo debe seguir con su ejecución o llamar a wait() en el método SiParar(); esto nos sirve para que cuando le demos al botón de parar en la interfaz, todos los hilos de los aeropuertos se paren.

Descripción de las clases principales

Parte Concurrente

Autobús:

Autobus		
Autobus(String, Long, Aeropuerto)		
prefijoID	String	
aeropuerto	Aeropuerto	
pasajerosSubidos	int	
idFormateado	String	
id	long	
run()		
aeropuerto	Aeropuerto	
idFormateado	String	
prefijoID	String	
id	long	
pasajerosSubidos	int	

Atributos:

Id: Sirve para identificar los autobuses como “B-XXXX”, donde XXXX es un número (id) único como, por ejemplo, B-1234

prefijoID: Es la letra de identificación del id, por ejemplo, en B-1234 el prefijoID es B-

idFormateado; Une ambas partes tanto id como su prefijo.

pasajerosSubidos: Número de pasajeros que suben al autobús.

Descripción

La clase autobús describe el ciclo de vida de los autobuses en un sistema de transporte.

Dicha clase llama a la clase aeropuerto y a sus respectivos métodos como paradaCiudad(), viajarAeropuerto(), paradaAeropuerto(), viajarCiudad() que se explicaran más adelante

Avión

Avion	
Avion(Aeropuerto, Aeropuerto, String, long, int, Log)	
idFormateado	String
id	long
capacidadMaxima	int
vuelos	int
prefijoID	String
destino	Aeropuerto
pasajerosDentro	int
copia	Aeropuerto
origen	Aeropuerto
run()	void
capacidadMaxima	int
prefijoID	String
origen	Aeropuerto
idFormateado	String
vuelos	int
destino	Aeropuerto
copia	Aeropuerto
pasajerosDentro	int
id	long

Atributos:

Id: sirve para identificar los autobuses como “B-XXXX”, donde XXXX es un número (id) único como, por ejemplo, B-1234

prefijoID: es la letra de identificación del id, por ejemplo, en B-1234 el prefijoID es B-

idFormateado; une ambas partes tanto id como su prefijo.

capacidadMaxima: capacidad máxima del avión aleatoria entre 100 y 300 pasajeros.

pasajerosDentro: Número de personas dentro del avión.

vuelos: contador de vuelos que ha realizado el avión.

Descripción

La clase avión tendrá el siguiente ciclo de vida, que será **repetido infinitamente**:

El avión, cuando se genera, llama al método entrarHangar() y salirHangar() de aeropuerto para acceder y salir del Hangar. El avión accede al ÁREA DE ESTACIONAMIENTO, donde esperará a que una de las PUERTAS DE EMBARQUE esté disponible y una vez abandonada la PUERTA DE EMBARQUE, accede directamente al ÁREA DE RODAJE, donde esperará a tener una PISTA para despegar. Tras despegar, accede a la AEROVÍA que conecte al aeropuerto de destino, una vez abandonada la PISTA, accede directamente al ÁREA DE RODAJE, donde solicita una PUERTA DE EMBARQUE y esperará hasta tener una asignada para desembarcar a los pasajeros. Una vez completado el desembarque, el avión accede al ÁREA DE ESTACIONAMIENTO y cuando un avión haya realizado 15 vuelos deberá realizar una inspección en profundidad en el TALLER. Si no pudiera acceder (por haber alcanzado la capacidad máxima de aviones dentro del TALLER), se pone en cola hasta que sea su turno (estrategia FIFO). El avión, aleatoriamente, decidirá entre ir al HANGAR a reposar (50% de probabilidad de que esto suceda) o continuar con el ciclo de vida (50% de probabilidad). En caso de ir al HANGAR, permanecerá allí durante un tiempo aleatorio entre 15 y 30 segundos. Si continúa con el ciclo de vida, estará listo para ir al ÁREA DE ESTACIONAMIENTO y continuar con el proceso.

InterfazServidor:

InterfazServidor	
InterfazServidor()	
servidor	Servidor
initComponents()	void
main(String[])	void
windowClosing()	void
pausaActionPerformed(ActionEvent)	void
TransfersCiudadMadridActionPerformed(ActionEvent)	void
TransfersCiudadBarcelonaActionPerformed(ActionEvent)	void
ReanudarActionPerformed(ActionEvent)	void
aeropuertoMadrid	Aeropuerto
aeropuertoBarcelona	Aeropuerto
servidor	Servidor

Atributos:

- **madrid y barcelona:** Representan los aeropuertos de la simulación.
- **sistema:** Se encarga de generar los autobuses y aviones de los aeropuertos madrid y barcelona.
- **log:** Se utiliza para registrar eventos en el servidor.
- **servidor:** Se usa para la conexión de los clientes.
- Los **locks** sirven para la exclusión mutua de la paradaCiudad y de la paradaAeropuerto de cada aeropuerto.

```
Lock lockM = new ReentrantLock();  
Lock lockB = new ReentrantLock();  
Lock lockAM = new ReentrantLock();  
Lock lockAB = new ReentrantLock();
```

- Además, tiene un **constructor** que inicializa los atributos y los hilos **sistema** y **servidor**. También tiene métodos para obtener y establecer los aeropuertos y el **servidor**, y un método **windowClosing** para cerrar el archivo de registro **log**.
- Tiene métodos para manejar los eventos de los botones:
 1. **PausarButton:** Pausa la simulación.
 2. **ReanudarButton:** Reanuda la simulación.

Clase Log:

Tiene como objetivo llevar un registro de los eventos ocurridos en la simulación de los aviones y autobuses y el evento en si del aeropuerto. Esta clase cuenta con atributos como **file**, objeto de tipo **File** que representa el archivo donde se va a guardar el registro de eventos, y **writer**, objeto de tipo **PrintWriter** que permite escribir en el archivo de registro con el método **write(String message)**. El método **close()** cierra el archivo de registro una vez que se han registrado todos los eventos.

Log	
Log()	
writeLog(String)	void
close()	void

Sistema:

Atributos:

- **madrid y barcelona:** Representan los aeropuertos.
- **log:** Se utiliza para registrar eventos en el sistema.

Sistema	
◦ Sistema(Aeropuerto, Aeropuerto)	
run()	void

Descripción

El sistema cuenta con 2 aeropuertos, cada aeropuerto dispone de autobuses que conectan la ciudad con el aeropuerto (y viceversa), para llevar y traer pasajeros.

Los autobuses y los aviones están modelados mediante hilos. El número de aviones que el sistema genera es de 8.000, para evitar congestión los aviones son creados de forma escalonada, en intervalos aleatorios de entre 1 y 3 segundos.

El número de autobuses que el sistema genera es de 4.000, para evitar congestión los autobuses son creados de forma escalonada, en intervalos aleatorios de entre 0,5 y 1 segundo. La generación de aviones y autobuses al comienzo de la ejecución del programa se realiza de forma simultánea.

Aeropuerto:

Atributos:

paradaAeropuerto: es un lock para garantizar la exclusión mutua de los aeropuertos.

paradaCiudad: es un lock para garantizar la exclusión mutua de las ciudades.

personasDentro: número de personas que hay en el avión.

nombre: corresponde con el nombre del aeropuerto que puede ser Madrid o Barcelona.

log: Se utiliza para registrar eventos en el aeropuerto.

parar: es un booleano que se utiliza en el método siParar() que comprueba si se ha pulsado el botón parar de la interfaz para detener el hilo y ponerlo en espera.









hangar, areaRodaje, aerovía, areaEstacionamiento: Son una arrayList de String que contiene los id formateados de los aviones que ocupan dichas zonas que sirven para poder imprimirlos en los jTextField correspondientes de la interfaz. En el caso del areaEstacionamiento es un LinkedList porque es una fifo.

PuertasEmbarque, PuertaDesembarque, pistas: Son arraylist de locks los usamos para comprobar si alguna de ellas esta ocupada y si no es así ocuparla.

pista4	Lock
lock	Lock
avionesTaller	int
puertaLibre4	Lock
areaRodaje	List<String>
PuertasEmbarque	List<Lock>
nombre	String
pista3Cerrada	boolean
puertaLibre2	Lock
algunoOcupado	Condition
puertaLibre3	Lock
pista1Cerrada	boolean
Pistas	List<Lock>
pista4Cerrada	boolean
log	Log
PuertasDesembarque	List<Lock>
taller	Semaphore
pista2	Lock
puertaDesembarque	Lock
personasDentro	int
parado	Condition
avionesHangar	int
avionesAreaEstacionamiento	int
puertaLibre1	Lock
avionesAreaRodaje	int
pistaCerrada	boolean
lockEstacionamiento	Lock
pista2Cerrada	boolean
puertaTaller	Lock
areaEstacionamiento	LinkedList<String>
hangar	List<String>
pista3	Lock
pista1	Lock
puertaEmbarque	Lock
aerovia	List<String>





En el **constructor** se añaden los locks correspondientes en los arraylist mencionados anteriormente.

Métodos que afectan a la clase Autobús:

  <code>paradaAeropuerto(Autobus)</code>	<code>void</code>
  <code>viajarAeropuerto(Autobus)</code>	<code>void</code>
  <code>paradaCiudad(Autobus)</code>	<code>void</code>
  <code>viajarCiudad(Autobus)</code>	<code>void</code>

- **paradaCiudad():** Llegada a la parada del centro de la ciudad, donde se detiene, para esperar que suban los pasajeros, durante un tiempo aleatorio entre 2 y 5 segundos y montarse en el autobús un número aleatorio entre 0 y 50 pasajeros.
- **viajarAeropuerto():** El autobús inicia su marcha en dirección al aeropuerto. Dicho trayecto le llevará un tiempo aleatorio entre 5 y 10 segundos
- **paradaAeropuerto():** Llegada a la parada del aeropuerto, donde los pasajeros entran al aeropuerto, en se momento los pasajeros son incorporados al sistema del aeropuerto y el autobús espera a que suban nuevos pasajeros durante un tiempo aleatorio entre 2 y 5 segundos. Se montan en el autobús un número aleatorio entre 0 y 50 pasajeros y este número de pasajeros son restados del sistema del aeropuerto.
- **viajarCiudad():** El autobús inicia su marcha en dirección al centro de la ciudad. Dicho trayecto le llevará un tiempo aleatorio entre 5 y 10 segundos, llegada a la parada del centro de la ciudad, bajarán los pasajeros, en este momento, los pasajeros dejarán de contar para el sistema.

Métodos que afectan a la clase Avión:

  <code>continuar()</code>	<code>void</code>
  <code>AreaRodaje(Avion, int)</code>	<code>void</code>
  <code>solicitarPistaAterrizaje(Avion)</code>	<code>void</code>
  <code>desembarcarPasajeros(Avion, Lock, int)</code>	<code>void</code>
  <code>controlBloqueoPuertas(Avion, int)</code>	<code>int</code>
  <code>salirAerovia(Avion)</code>	<code>void</code>
  <code>siParar()</code>	<code>void</code>
  <code>quitarAvionHangar(Avion)</code>	<code>void</code>
  <code>annadirAvionAreaRodaje(Avion)</code>	<code>void</code>
  <code>quitarAvionAerovia(Avion)</code>	<code>void</code>
  <code>aterrizar(Avion, Lock, int)</code>	<code>void</code>

- **actualizarPersonasDentro():** Este método sirve exclusivamente para imprimir el numero de personas dentro del aeropuerto en su JTextField y esto lo hicimos para hacer los synchronized y que de esta forma no de problemas cuando muchos hilos quieran modificar este campo a la vez.
- **entrarHangar():** El avión cuando se genera aparece en el Hangar.

- **salirHangar():** El avión sale del Hangar.
- **obtenerPrimeroColaAreaEstacionamiento():** Es un método que usamos para asegurarnos de que el avión que llama a este método sea el primero de la cola fifo área de estacionamiento. En caso de que lo sea intentara acceder a una puerta de embarque y en caso contrario no hará nada.
- **controlBloqueoPuertas():** Comprueba cada una de las puertas de embarque hasta que alguna esté disponible y pueda ocuparla.
- **cerrarBloqueoPuertas():** Sirve para desbloquear las puertas bloqueadas anteriormente.
- **AreaEstacionamiento():** Añade el avión al área de estacionamiento, espera el tiempo indicado por el enunciado, luego llama al método controlbloqueoPuertas y posteriormente embarcarPasajeros que hablaremos más adelante y por último a cerrarbloqueoPuertas para liberar la puerta utilizada por el avión . Este método contempla el caso de que este mismo avión provenga del hangar o de la puerta de desembarque.
- **embarcarPasajeros():**El método embarcarPasajeros se encarga de llevar a cabo el proceso de embarque de pasajeros en un avión. Primero, verifica si la simulación debe pausarse mediante el método siParar(). Luego, itera hasta tres intentos para embarcar pasajeros. Dentro del bucle, se calcula cuántos pasajeros pueden embarcarse en el avión y se realiza el embarque si hay suficientes personas en el aeropuerto, actualizando tanto el avión como el aeropuerto con la cantidad de pasajeros embarcados. Si no hay suficientes personas en el aeropuerto, se embarcan todas las disponibles. Cada iteración del bucle se pausa durante un tiempo aleatorio y se manejan las excepciones de interrupción. El contador de intentos se incrementa al final de cada iteración para realizar un seguimiento de los intentos realizados.

```

③ inspeccion(Avion, int) void
③ despegar(Avion, int) void
③ cerrarBloqueoPuertas(Avion, int, int) void
③ parar() void
③ quitarAvionAreaRodaje(Avion) void
③ actualizarPersonasDentro(int) void
③ AreaEstacionamiento(Avion, int) void
③ mirarSiParar() boolean
③ embarcarPasajeros(Avion, int) void

```

- **AreaRodaje():** Este método maneja las operaciones relacionadas con el área de rodaje de un aeropuerto para los aviones. Dependiendo del caso proporcionado, el método realiza las siguientes acciones:
 1. El avión accede al área de rodaje sin dirigirse a una puerta de embarque. El método registra la llegada del avión al área de rodaje, lo añade al área de rodaje, actualiza el contador de aviones en el área de rodaje y realiza una espera aleatoria para simular las comprobaciones de los pilotos.
 2. El avión accede al área de rodaje para dirigirse a una puerta de embarque. Se registra esta acción en el registro de actividad, se añade el avión al área de rodaje, se actualiza el contador de aviones en el área de rodaje, se realiza una espera aleatoria para comprobaciones de los pilotos y se asegura que la puerta de embarque esté libre antes de dirigirse a ella.

En ambos casos, el método gestiona las excepciones de interrupción de hilos y actualiza el campo de texto correspondiente para reflejar los cambios en el área de rodaje.

- **bloquearPista(), liberarPista():** Son métodos que usa la clase AeropuertoImpl para bloquear y desbloquear las pistas.
- **solicitarPista():** Este método se encarga de solicitar una pista de despegue para un avión en el aeropuerto. Comienza con una pausa en la ejecución utilizando el método `siParar()`. Luego, inicializa un contador y una variable booleana para controlar el bucle.

El bucle while se ejecuta mientras la variable booleana **variable** sea verdadera. Dentro de este bucle, se itera sobre la lista de pistas disponibles en el aeropuerto. Para cada pista, se verifica su disponibilidad utilizando el método `tryLock()` del objeto `Lock`. Si una pista está disponible, se bloquea y se ejecuta el código dentro del bloque if.

Dentro del bloque if, se realizan varias acciones:

Se retira el avión del área de rodaje y se actualiza el contador de aviones en el área de rodaje.

Se actualiza la visualización del área de rodaje para reflejar los cambios.

Se registra en el registro de actividad que el avión está accediendo a una pista para despegar.

Se marca la pista como ocupada por el avión y se actualiza la visualización correspondiente.

Se inicia el proceso de despegue del avión llamando al método `despegar()`.

Finalmente, se establece la variable **variable** en falso para salir del bucle. Si ninguna pista está disponible, el contador se incrementa para pasar a la siguiente pista en la siguiente iteración del bucle for. Una vez que se encuentra una pista disponible y se realizan todas las acciones necesarias, el método termina. Este método gestiona las excepciones de interrupción de hilos y actualiza la interfaz de usuario para reflejar los cambios en el estado del aeropuerto.

- **despegar():** Este método se encarga de llevar a cabo el proceso de despegue de un avión en el aeropuerto. Comienza con una pausa en la ejecución utilizando el método `siParar()` para sincronizar la ejecución con otros procesos. A continuación, se realizan verificaciones antes del despegue, lo cual puede incluir procedimientos de seguridad y rutinas de chequeo de la aeronave. Estas acciones son registradas en el registro de actividad mediante el método `writeLog`. Después de las verificaciones, se introduce una pausa aleatoria utilizando `Thread.sleep()` para simular el tiempo necesario para realizar dichas acciones. Una vez completadas las verificaciones, se registra en el registro de actividad que el avión ha despegado y se introduce otra pausa aleatoria para simular el tiempo de vuelo. Posteriormente, se actualiza la interfaz de usuario, específicamente el campo de texto correspondiente a la pista utilizada para el despegue, para indicar que está nuevamente disponible. Si ocurre alguna excepción durante el proceso (como una interrupción de hilo), se maneja adecuadamente utilizando una estructura try-catch y se registra en el registro de actividad. Finalmente, se libera el lock de la pista utilizada para el despegue mediante el método `unlock()`, asegurando que esté disponible para otro avión. En resumen, este método gestiona todo el proceso de despegue de un avión en el aeropuerto, desde las verificaciones previas hasta la actualización de la interfaz de usuario y la liberación de recursos.

- **accederAerovia():** Este método permite que un avión acceda a la aerovía desde su origen para iniciar su trayecto hacia su destino. Comienza con una pausa en la ejecución usando `siParar()`, que puede ser útil para sincronizar con otros procesos. Luego, el avión utiliza el método `entrarAerovia` de su objeto `Origen` para ingresar a la aerovía. Esto actualiza su estado interno para indicar que ha entrado a la aerovía. Se registra en el registro de actividad (log) que el avión ha accedido a la aerovía, especificando si es la aerovía "MADRID-BARCELONA" o "BARCELONA-MADRID", dependiendo de su origen. Después, se introduce una pausa aleatoria utilizando `Thread.sleep()` para simular el tiempo que el avión pasa en la aerovía antes de continuar su trayecto. La duración de esta pausa varía entre 15 y 30 segundos. Una vez que el tiempo en la aerovía ha transcurrido, el avión utiliza el método `solicitarPistaAterrizaje` de su objeto `Destino` para iniciar el proceso de aterrizaje en su destino. Cualquier excepción de interrupción de hilos (`InterruptedException`) es manejada y registrada en el log en caso de que ocurra.
- **entrarAerovia:** Este método permite que un avión entre a la aerovía desde su origen. Comienza con una pausa en la ejecución utilizando `siParar()`, que puede ser útil para sincronizar con otros procesos. Después, el avión actual (`aThis`) se agrega a la lista de aviones en la aerovía utilizando el método `annadirAvionAerovia`. Seguidamente, se actualiza el campo de texto `fieldAerovia` para reflejar la lista actualizada de aviones en la aerovía. Esto se hace utilizando `String.join()` para convertir la lista de aviones en la aerovía en una cadena legible que se muestra en el campo de texto. En resumen, este método sincronizado permite que un avión entre a la aerovía desde su origen, actualizando la lista de aviones en la aerovía y reflejándola en la interfaz de usuario.
- **salirAerovia():** Este método permite que un avión salga de la aerovía en su punto de origen. Comienza con una pausa en la ejecución utilizando `siParar()`, que puede ser útil para sincronizar con otros procesos. Después, el avión actual (`aThis`) se elimina de la lista de aviones en la aerovía utilizando el método `quitarAvionAerovia` del objeto `Origen` del avión. Seguidamente, se actualiza el campo de texto `fieldAerovia` del origen del avión para reflejar la lista actualizada de aviones en la aerovía. Esto se hace utilizando `String.join()` para convertir la lista de aviones en la aerovía en una cadena legible que se muestra en el campo de texto. En resumen, este método sincronizado permite que un avión salga de la aerovía en su punto de origen, actualizando la lista de aviones en la aerovía y reflejándola en la interfaz de usuario.

		<code>accederAerovia(Avion)</code>	<code>void</code>
		<code>entrarAerovia(Avion)</code>	<code>void</code>
		<code>annadirAvionHangar(Avion)</code>	<code>void</code>
		<code>solicitarPista(Avion)</code>	<code>void</code>
		<code>salirHangar(Avion)</code>	<code>void</code>
		<code>pasarPuertaTaller(Avion)</code>	<code>void</code>
		<code>annadirAvionAerovia(Avion)</code>	<code>void</code>
		<code>entrarHangar(Avion)</code>	<code>void</code>
		<code>obtenerPrimeroColaAreaEstacionamiento()</code>	<code>String</code>

- **solicitarPistaAterrizaje():** Este método solicita una pista de aterrizaje para un avión en el aeropuerto. Comienza con una pausa en la ejecución utilizando `siParar()`, lo que puede ser útil para sincronizar con otros procesos. Luego, se inicializa un contador y

una variable booleana que controla un bucle while. Dentro del bucle, se itera sobre la lista de pistas disponibles en el aeropuerto. Para cada pista, se verifica su disponibilidad utilizando el método tryLock() del objeto Lock. Si una pista está disponible, se bloquea y se ejecuta el código dentro del bloque if. Al obtener una pista disponible, el avión actual sale de la aerovía utilizando el método salirAerovia, se registra en el registro de actividad que el avión ha accedido a una pista para aterrizar, y se actualiza la visualización de la pista correspondiente en la interfaz de usuario con información sobre el avión y el tipo de operación (aterrizaje). Luego, se llama al método aterrizar para iniciar el proceso de aterrizaje del avión en la pista, y la variable “variable” se establece en falso para salir del bucle. Si ninguna pista está disponible, se incrementa el contador para pasar a la siguiente pista en la próxima iteración del bucle for. Después de cada intento de solicitud de pista, se introduce una pausa aleatoria para simular el tiempo de espera antes de intentar nuevamente.

- **aterrizar():** Este método simula el proceso de aterrizaje de un avión en una pista del aeropuerto. Comienza con una pausa en la ejecución utilizando siParar(), que puede ser útil para sincronizar con otros procesos. Luego, se registra en el registro de actividad que el avión ha aterrizado en una pista del aeropuerto. A continuación, se introduce una pausa aleatoria utilizando Thread.sleep() para simular el tiempo que el avión tarda en completar el proceso de aterrizaje. La duración de esta pausa puede variar entre 1 y 6 segundos. Después de que el avión ha aterrizado, se actualiza la visualización de la pista correspondiente en la interfaz de usuario para indicar que la pista está nuevamente disponible. Esto se hace eliminando el identificador del avión de la pista. Se manejan las excepciones de interrupción de hilos (InterruptedException) y se registran en el registro de actividad en caso de que ocurran. Finalmente, se libera el lock de la pista utilizando el método unlock() del objeto Lock, asegurando que la pista esté disponible para otro avión.
- **desembarcarPasajeros():** Este método simula el proceso de desembarque de pasajeros de un avión en una puerta de embarque del aeropuerto. Comienza con una pausa en la ejecución utilizando `siParar()`, lo que puede ser útil para sincronizar con otros procesos. Luego, se registra en el registro de actividad (log) la cantidad de pasajeros que están desembarcando del avión actual (`aThis`). Se actualiza el total de personas dentro del aeropuerto sumando la cantidad de pasajeros que desembarcan. Luego, se actualiza la visualización de la cantidad de personas dentro del aeropuerto. Después, se establece el número de pasajeros dentro del avión actual a 0, ya que todos han desembarcado. A continuación, se introduce una pausa aleatoria utilizando Thread.sleep() para simular el tiempo que toma el proceso de desembarque. La duración de esta pausa puede variar entre 1 y 6 segundos. Después de que se completa el desembarque de pasajeros, se actualiza la visualización de la puerta de embarque correspondiente en la interfaz de usuario para indicar que la puerta está nuevamente disponible. Esto se hace eliminando el identificador del avión de la puerta de embarque. Se manejan las excepciones de interrupción de hilos (InterruptedException) y se registran en el registro de actividad en caso de que ocurran. En resumen, este método simula el proceso de desembarque de pasajeros de un avión en una puerta de embarque del aeropuerto, incluyendo una pausa aleatoria para representar el tiempo del proceso. También actualiza la cantidad total de personas dentro del aeropuerto y la visualización de la disponibilidad de la puerta de embarque en la interfaz de usuario.
- **inspección():** Este método simula el proceso de inspección de un avión en el taller del aeropuerto. Comienza con una pausa en la ejecución utilizando siParar(), lo que puede

ser útil para sincronizar con otros procesos. Luego, adquiere un permiso del semáforo taller utilizando `taller.acquire()`, indicando que el avión está ingresando al taller para inspección. Después, se elimina el avión del área de estacionamiento y se actualiza el contador de aviones en el taller y en el área de estacionamiento. Se actualiza la visualización del área de estacionamiento en la interfaz de usuario para reflejar estos cambios. Se introduce una pausa aleatoria para simular el tiempo que lleva la inspección. Se registra en el registro de actividad (log) que el avión ha terminado su inspección y está dejando el taller. Se simula el proceso de que el avión pasa por la puerta del taller llamando al método `pasarPuertaTaller`. Después de eso, se reduce el contador de aviones en el taller y se libera el permiso del semáforo taller utilizando `taller.release()`. En resumen, este método simula el proceso de inspección de un avión en el taller del aeropuerto, incluyendo una pausa aleatoria para representar el tiempo del proceso y la actualización de la interfaz de usuario para reflejar el estado de la inspección.

- **pasarPuertaTaller():** Este método simula el proceso de un avión pasando por la puerta del taller del aeropuerto. Comienza con una pausa en la ejecución utilizando `siParar()`, lo que puede ser útil para sincronizar con otros procesos. Luego, adquiere el lock de la puerta del taller utilizando `puertaTaller.lock()`, asegurando que solo un avión pueda pasar por la puerta a la vez. Después, se registra en el registro de actividad (log) que el avión ha pasado por la puerta del taller. Se introduce una pausa aleatoria utilizando `Thread.sleep()` para simular el tiempo que lleva que el avión pase por la puerta del taller. La duración de esta pausa puede variar entre 0 y 1 segundo. Finalmente, se libera el lock de la puerta del taller utilizando `puertaTaller.unlock()`, permitiendo que otros aviones puedan pasar por la puerta. En resumen, este método simula el proceso de un avión pasando por la puerta del taller del aeropuerto, asegurando que solo un avión pueda pasar por la puerta a la vez y registrando esta actividad en el registro de actividad del aeropuerto.

Parte Distribuida

AeropuertoImpl y AeropuertoRemote:

Descripción:

Estas clases definen el conjunto de métodos remotos invocados en un objeto remoto que implemente esta interfaz, estos métodos se usan para obtener información sobre el número de pasajeros y aviones en diferentes áreas del aeropuerto, así como para cerrar y liberar pistas de aterrizaje en aeropuertos específicos (Madrid y Barcelona). Utiliza una interfaz de servidor ('InterfazServidor') para acceder a las instancias de aeropuerto de Madrid y Barcelona. Los métodos de cierre y liberación de pistas llaman a métodos correspondientes en los aeropuertos de Madrid y Barcelona a través de la interfaz del servidor. La clase maneja excepciones remotas ('RemoteException') que pueden ocurrir durante las operaciones remotas.

Cliente:

Descripción:

Utiliza RMI (Remote Method Invocation) para establecer la conexión con el objeto remoto ubicado en 127.0.0.1 en el puerto predeterminado.

En su método run(), el cliente actualiza periódicamente la interfaz gráfica con la información proporcionada por AeropuertoRemote, como el número de pasajeros y aviones en diferentes áreas de los aeropuertos. Además, el cliente puede enviar solicitudes para cerrar y liberar pistas de aterrizaje en los aeropuertos de Madrid y Barcelona, según las acciones del usuario en la interfaz gráfica.

InterfazCliente

Descripción:

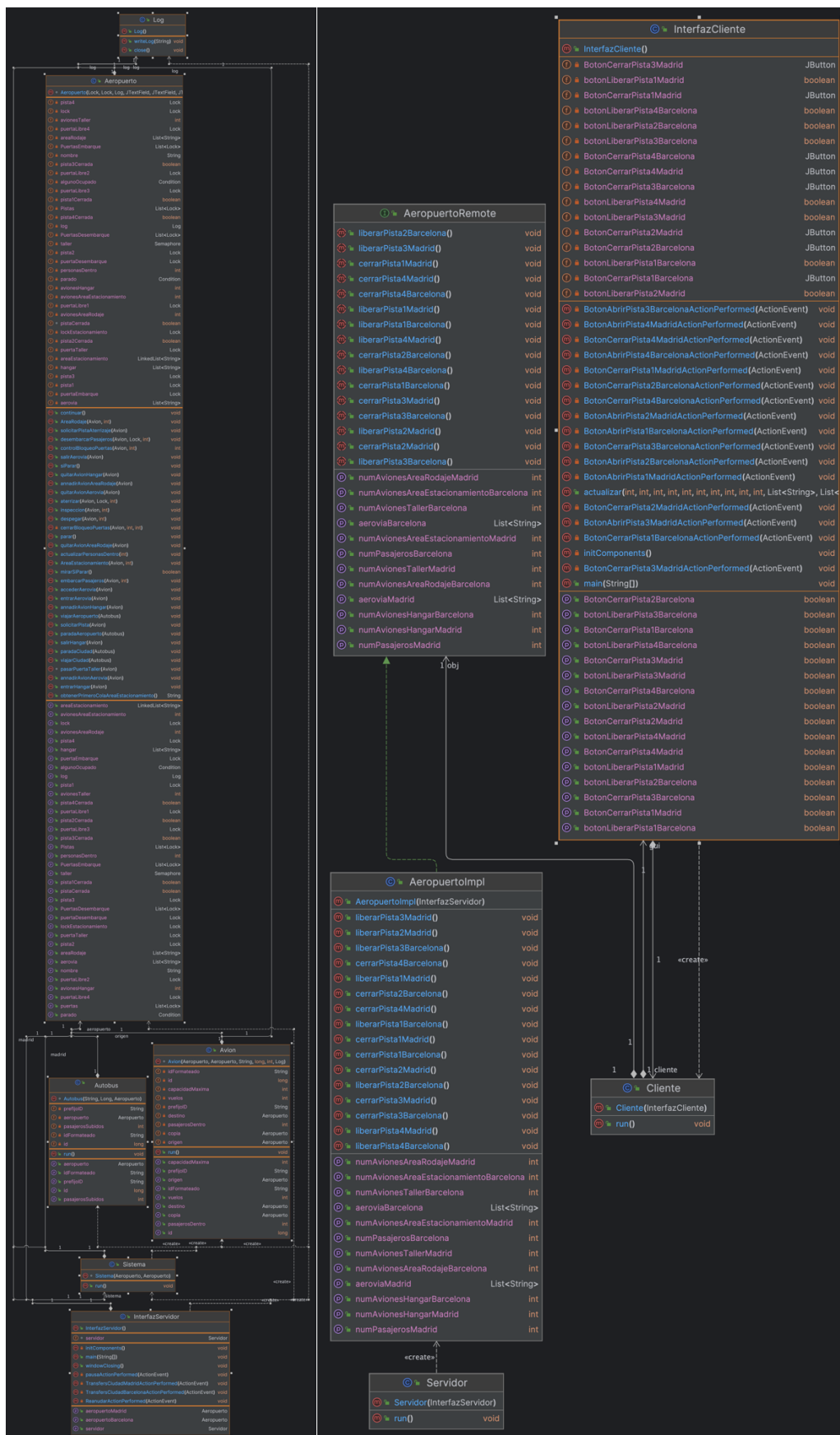
Esta interfaz se utiliza para interactuar con el cliente que se conecta a nuestro servidor para obtener información sobre los aeropuertos y controlar el estado de las pistas de aterrizaje. La interfaz incluye campos de texto para mostrar información como el número de pasajeros, el número de aviones en hangares, talleres, áreas de estacionamiento y áreas de rodaje para los aeropuertos de Madrid y Barcelona. Además, hay campos para mostrar la aerovía de cada aeropuerto. La interfaz también incluye botones para cerrar y abrir pistas de aterrizaje en cada aeropuerto, y estos botones están asociados con variables booleanas que indican si se ha presionado el botón correspondiente. La clase también contiene métodos getter y setter para acceder y modificar estas variables booleanas. La clase 'InterfazCliente' también inicializa un objeto 'Cliente' y un hilo para ejecutarlo en segundo plano.

Servidor

Descripción:

Esta clase utiliza RMI (Remote Method Invocation) para permitir que los clientes accedan a objetos remotos. En su método 'run()', crea un objeto de la clase 'AeropuertoImpl', que implementa la interfaz 'AeropuertoRemote'. Luego, crea un registro RMI local en el puerto 1099 y registra el objeto remoto en este registro utilizando la URL "localhost/ObjetoAeropuerto". Todo esto se realiza dentro de un bloque try-catch para manejar posibles excepciones.

Diagramas de clases:



Dificultades:

En la realización de nuestro código nos hemos encontrado con muchas dificultades como cabría esperar de un trabajo con una importancia como este. Entre ellas se encuentran las siguientes:

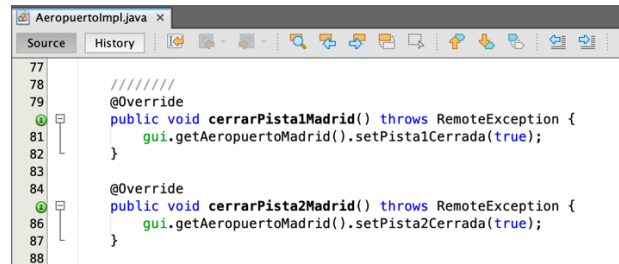
- **Control de las puertas de embarque:** Esta parte del código presentó una especial dificultad para nosotros porque al principio se nos ocurrió la idea de hacer un Array compuesto por los 6 locks que son las puertas de embarque. Intentaríamos recorrer dicho array, en un bucle for, tratando de ocupar cada uno de los locks y pasando al siguiente mediante el método tryLock() y el uso de condicionales "if". Esto y el hecho de que hay mucho código entre el lock y el unlock, generó que dentro de ese código llamáramos a otros como embarcarPasajeros(), para optimizar el código. Además, en vez de hacer lo que acabamos de explicar del Array de Locks, decidimos finalmente comprobar uno a uno con condiciones if, puesto que el array nos estaba dando problemas. Sin embargo, este sistema no lo eliminamos del todo, puesto que sí que lo usamos para el bloqueo y desbloqueo de pistas y junto con una variable int llamada "contador", para decidir sobre que jTextField de la interfaz imprimir el ID del avión correspondiente. Otro aspecto a destacar relacionado con las puertas de embarque, es que en vez de tener un solo Array para todas las puertas de embarque, hemos creado dos Arrays, uno compuesto por las puertas libres y por la puerta de embarque; y otro compuesto por esas mismas puertas libres y la puerta de desembarque; actuando sobre cada una en los métodos correspondientes (embarcarPasajeros() y desembarcarPasajeros() respectivamente)
- **La aerovía y el paso de un avión de un aeropuerto a otro:** Inicialmente se nos ocurrió la idea de usar un exchanger para que los aeropuertos pudieran intercambiar los aviones; pero ante el hecho de que un avión debía poder pasar de un aeropuerto a otro sin la necesidad de que hubiera otro avión en la otra aerovía, se nos ocurrió la idea de crear las variables origen y destino de la clase Avión. Estas variables, permitían llamar dentro del run de cada hilo, al aeropuerto que nos interesara para cada parte del código. Siendo que a mitad de la ejecución del run (en el momento en que el aeropuerto sale de la aerovía), el aeropuerto sobre el que llamamos a los métodos cambia al otro.
- **Bloquear y Desbloquear pistas:** A pesar de que en los métodos de aeropuerto relacionados con la ejecución run() de los hilos de tipo Avión no nos dio ningún problema el bloqueo de los locks de las pistas, sí que nos los dio a la hora de implementar la funcionalidad del Cliente de bloquear las pistas que él quiera a través de los botones de su interfaz. Primero intentamos hacer esta parte del código mediante el uso de los siguientes métodos.

```
public void bloquearPista(int pista) {  
    if (this.getPistas().get(pista).tryLock()) {  
        System.out.println("Pista " + pista + " del aeropuerto de " + this.nombre + " se ha bloqueado por el cliente");  
    } else {  
        System.out.println("Nope");  
    }  
}  
  
public void liberarPista(int pista) {  
    System.out.println("Pista " + pista + " del aeropuerto de " + this.nombre + " se ha desbloqueado por el cliente");  
    this.getPistas().get(pista).unlock();  
}
```

Sin embargo, esto no nos funcionaba, a pesar de que sí que se ejecutaba dichos métodos, el bloqueo de la pista que nos interesase nunca tenía lugar.

Ante este problema, se nos ocurrió usar variables booleanas que cambiaran de valor a la hora de pulsar los botones de la interfaz del cliente, y que en los métodos en los que se intenta adquirir las pistas, comprobaran también el valor de dichas variables. Así como se ve a continuación:

Ejemplo de cambio del valor del booleano en la clase AeropuertoImpl:



```
77
78
79 //////////////
80 @Override
81 public void cerrarPista1Madrid() throws RemoteException {
82     gui.getAeropuertoMadrid().setPista1Cerrada(true);
83 }
84
85 @Override
86 public void cerrarPista2Madrid() throws RemoteException {
87     gui.getAeropuertoMadrid().setPista2Cerrada(true);
88 }
```

Modificaciones en el código para que se compruebe el valor de las variables antes de adquirirlas

```
public void solicitarPista(Avion aThis) {
    siParar();
    int contador;
    boolean variable = true;
    while (variable) {
        contador = 0;
        for (Lock l : Pistas) {
            switch (contador) {
                case 0 ->
                    pistaCerrada = pista1Cerrada;
                case 1 ->
                    pistaCerrada = pista2Cerrada;
                case 2 ->
                    pistaCerrada = pista3Cerrada;
                case 3 ->
                    pistaCerrada = pista4Cerrada;
                default -> {
            }
        }
    }
    if (l.tryLock() && !pistaCerrada) {
        //Sale del Area de Rodaje
    }
}
```

Anexo:

```
package Concurrida;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.Semaphore;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JTextField;

// Clase para representar un aeropuerto
public class Aeropuerto {

    private boolean pista1Cerrada, pista2Cerrada, pista3Cerrada, pista4Cerrada = false;
    boolean pistaCerrada;

    private final JTextField fieldParadaAeropuerto, fieldParadaCiudad, TFPersonasDentro,
    fieldHangar, fieldTaller, tFAreaEstacionamiento, fieldGate1, fieldGate2, fieldGate3,
    fieldGate4, fieldGate5, fieldGate6, fieldAreaRodaje, fieldPista1, fieldPista2,
    fieldPista3, fieldPista4, fieldAerovia;

    //Variables para Autobus
    private final Lock paradaAeropuerto;
    private final Lock paradaCiudad;

    //Variables para Avion
```

```
private int personasDentro;
```

```
private String nombre;
```

```
private Log log;
```

```
private boolean parar = false;
```

```
//Hangar
```

```
private List<String> hangar = new ArrayList<>();
```

```
private LinkedList<String> areaEstacionamiento = new LinkedList<>();
```

```
private Lock lockEstacionamiento = new ReentrantLock();
```

```
private Condition algunoOcupado = lockEstacionamiento.newCondition();
```

```
private Lock puertaEmbarque = new ReentrantLock();
```

```
private Lock puertaDesembarque = new ReentrantLock();
```

```
private Lock puertaLibre1 = new ReentrantLock();
```

```
private Lock puertaLibre2 = new ReentrantLock();
```

```
private Lock puertaLibre3 = new ReentrantLock();
```

```
private Lock puertaLibre4 = new ReentrantLock();
```

```
private List<Lock> PuertasEmbarque = new ArrayList<>();
```

```
private List<Lock> PuertasDesembarque = new ArrayList<>();
```

```
private List<String> areaRodaje = new ArrayList<>();
```

```
private Lock pista1 = new ReentrantLock();
```

```
private Lock pista2 = new ReentrantLock();
```

```
private Lock pista3 = new ReentrantLock();
```

```
private Lock pista4 = new ReentrantLock();
```

```
private List<Lock> Pistas = new ArrayList<>();
```

```
private Lock lock = new ReentrantLock();
```

```
private Condition parado = lock.newCondition();
```

```
private List<String> aerovia = new ArrayList<>();
```

```
private Semaphore taller = new Semaphore(20, true);
```

```
private Lock puertaTaller = new ReentrantLock();
```

```
private int avionesHangar, avionesTaller, avionesAreaRodaje,  
avionesAreaEstacionamiento;
```

```
Aeropuerto(Lock paradaCiudad, Lock paradaAeropuerto, Log log, JTextField  
TransfersAeropuerto, JTextField TransfersCiudad, JTextField TFPersonasDentro,
```

```
JTextField txtHangar, JTextField txtTaller, JTextField txtAreaEstacion, JTextField  
txtGate1,
```

```
JTextField txtGate2, JTextField txtGate3, JTextField txtGate4, JTextField txtGate5,
```

```
JTextField txtGate6, JTextField txtAreaRodaje, JTextField txtPista1, JTextField  
txtPista2,
```

```
JTextField txtPista3, JTextField txtPista4, JTextField txtAerovia, String nombre) {
```

```
PuertasEmbarque.add(puertaEmbarque);
```

```
PuertasEmbarque.add(puertaLibre1);
```

```
PuertasEmbarque.add(puertaLibre2);
```

```
PuertasEmbarque.add(puertaLibre3);
```

```
PuertasEmbarque.add(puertaLibre4);
```

```
PuertasDesembarque.add(puertaDesembarque);
```

```
PuertasDesembarque.add(puertaLibre1);
```

```
PuertasDesembarque.add(puertaLibre2);
```

```
PuertasDesembarque.add(puertaLibre3);
```

```
PuertasDesembarque.add(puertaLibre4);
```

```
Pistas.add(pista1);
```

```
Pistas.add(pista2);
```

```
Pistas.add(pista3);
```

```
Pistas.add(pista4);
```

```
this.paradaCiudad = paradaCiudad;
```

```
this.paradaAeropuerto = paradaAeropuerto;
```

```
this.log = log;
```

```
this.fieldParadaAeropuerto = TransfersAeropuerto;
```

```
this.fieldParadaCiudad = TransfersCiudad;
```

```
this.TFPersonasDentro = TFPersonasDentro;
```

```
this.fieldHangar = txtHangar;
```

```
this.fieldTaller = txtTaller;
```

```
this.tFAreaEstacionamiento = txtAreaEstacion;
```

```
this.fieldGate1 = txtGate1;
```

```
this.fieldGate2 = txtGate2;
```

```
this.fieldGate3 = txtGate3;
```

```
this.fieldGate4 = txtGate4;
```

```
this.fieldGate5 = txtGate5;
```

```
this.fieldGate6 = txtGate6;
```

```
this.fieldAreaRodaje = txtAreaRodaje;
```

```
this.fieldPista1 = txtPista1;
```

```
this.fieldPista2 = txtPista2;
```

```
this.fieldPista3 = txtPista3;
```

```
this.fieldPista4 = txtPista4;
```

```
this.fieldAerovia = txtAerovia;
```

```
this.nombre = nombre;
```

```
TFPersonasDentro.setText(String.format("%s", personasDentro));
```

```
}
```

```
//MÉTODOS PARA AUTOBUSES
```

```
public void paradaCiudad(Autobus aThis) {  
    siParar();  
    try {  
        paradaCiudad.lock();  
        log.writeLog("El autobús " + aThis.getIdFormateado() + " llegó a " +  
this.getNombre());  
        siParar();  
        fieldParadaCiudad.setText(String.format("%s", aThis.getIdFormateado() + "(" +  
aThis.getPasajerosSubidos() + ")"));  
        Thread.sleep(new Random().nextInt(4) * 1000);  
        // Suben los pasajeros  
        aThis.setPasajerosSubidos(new Random().nextInt(51));  
        log.writeLog("Al autobús " + aThis.getIdFormateado() + " subieron " +  
aThis.getPasajerosSubidos() + " pasajeros.");  
        siParar();  
        fieldParadaCiudad.setText(String.format("%s", aThis.getIdFormateado() + "(" +  
aThis.getPasajerosSubidos() + ")"));  
        Thread.sleep(2000);  
        siParar();  
        fieldParadaCiudad.setText("");  
    } catch (InterruptedException ex) {  
        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);  
    } finally {  
        paradaCiudad.unlock();  
    }  
}
```

```
public void viajarAeropuerto(Autobus aThis) {
```

```
    siParar();
```

```

try{
    log.writeLog("El autobús " + aThis.getIdFormateado() + " viaja hacia el aeropuerto de
" + this.getNombre());

    Thread.sleep((new Random().nextInt(6) * 1000) + 5000);
} catch (InterruptedException ex) {
    Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);
}
}

```

```

public void paradaAeropuerto(Autobus aThis) {
    siParar();

    try{
        paradaAeropuerto.lock();

        siParar();

        fieldParadaAeropuerto.setText(String.format("%s", aThis.getIdFormateado() + "(" +
aThis.getPasajerosSubidos() + ")"));

        Thread.sleep(new Random().nextInt(4) * 1000);

        //Se bajan los pasajeros y se suben otros

        this.setPersonasDentro(this.getPersonasDentro() + aThis.getPasajerosSubidos());

        actualizarPersonasDentro(personasDentro);

        Thread.sleep(2000);

        aThis.setPasajerosSubidos(0);

        siParar();

        fieldParadaAeropuerto.setText(String.format("%s", aThis.getIdFormateado() + "(" +
aThis.getPasajerosSubidos() + ")"));

        aThis.setPasajerosSubidos(new Random().nextInt(51));

        if (aThis.getPasajerosSubidos() >= personasDentro) {
            aThis.setPasajerosSubidos(personasDentro);

            this.setPersonasDentro(0);
        } else {
            this.setPersonasDentro(this.getPersonasDentro() - aThis.getPasajerosSubidos());
        }
    }
}

```



```

        fieldParadaAeropuerto.setText(String.format("%s", aThis.getIdFormateado() + "(" +
aThis.getPasajerosSubidos() + ")"));

        actualizarPersonasDentro(personasDentro);

        log.writeLog("El autobús " + aThis.getIdFormateado() + " llegó al aeropuerto de " +
this.getNombre() + ", se bajaron sus pasajeros y se han subido otros " +
aThis.getPasajerosSubidos() + " nuevos pasajeros.");

        siParar();

        fieldParadaAeropuerto.setText("");

    } catch (InterruptedException ex) {

        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);

    } finally {

        paradaAeropuerto.unlock();

    }

}

```

```

public void viajarCiudad(Autobus aThis) {

    siParar();

    try {

        log.writeLog("El autobús " + aThis.getIdFormateado() + " viaja hacia la ciudad " +
this.getNombre());

        Thread.sleep((new Random().nextInt(6) * 1000) + 5000);

    } catch (InterruptedException ex) {

        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);

    }

}

```

```

public synchronized void siParar() {

    while (parar) {

        try {

            wait();

        } catch (InterruptedException ex) {

            Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);

        }

    }

}

```

```
    }  
    }  
}
```

//MÉTODOS PARA AVIONES

```
public synchronized void actualizarPersonasDentro(int personasDentro) {  
    TFPersonasDentro.setText(String.format("%s", personasDentro));  
}
```

```
public synchronized void entrarHangar(Avion AThis) {  
    siParar();  
    annadirAvionHangar(AThis);  
    avionesHangar++;  
    fieldHangar.setText(String.join(", ", hangar));  
}
```

```
public synchronized void salirHangar(Avion AThis) {  
    siParar();  
    quitarAvionHangar(AThis);  
    avionesHangar--;  
    if (hangar != null && !hangar.isEmpty()) {  
        fieldHangar.setText(String.join(", ", hangar));  
    } else {  
        fieldHangar.setText("");  
    }  
}
```

```
public synchronized String obtenerPrimeroColaAreaEstacionamiento() {  
    siParar();  
    if (this.getAreaEstacionamiento() != null) {  
        return this.getAreaEstacionamiento().get(0);  
    }  
}
```

```
} else {  
    return null;  
}  
}
```

```
public int controlBloqueoPuertas(Avion aThis, int caso) {  
    siParar();  
    int contador = 0;  
    boolean continuar = true;  
    while (continuar) {  
        if (caso == 1) {  
            if (this.getPuertaEmbarque().tryLock()) {  
                siParar();  
                log.writeLog("El avion " + aThis.getIdFormateado() + " ha obtenido la puerta de  
embarque");  
                areaEstacionamiento.remove(aThis.getIdFormateado());  
                avionesAreaEstacionamiento--;  
                if (areaEstacionamiento != null && !areaEstacionamiento.isEmpty()) {  
                    siParar();  
                    tFAreaEstacionamiento.setText(String.join(", ", areaEstacionamiento));  
                } else {  
                    siParar();  
                    tFAreaEstacionamiento.setText("");  
                }  
                siParar();  
                fieldGate1.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro() +  
")" + "EMBARCA");  
                contador = 0;  
                break;  
            }  
        }  
        if (this.getPuertaLibre1().tryLock()) {
```

```

        if (caso == 1) {
            siParar();

            log.writeLog("El avion " + aThis.getIdFormateado() + " ha obtenido la puerta libre
1 para embarcar");

            areaEstacionamiento.remove(aThis.getIdFormateado());

            avionesAreaEstacionamiento--;

            if (areaEstacionamiento != null && !areaEstacionamiento.isEmpty()) {
                siParar();

                tFAreaEstacionamiento.setText(String.join(", ", areaEstacionamiento));
            } else {
                siParar();

                tFAreaEstacionamiento.setText("");
            }

            siParar();

            fieldGate2.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro() +
")" + "EMBARCA");

            contador = 1;

            break;
        } else {
            try {
                Thread.sleep((new Random().nextInt(5) * 1000) + 3000);

                quitarAvionAreaRodaje(aThis);

                avionesAreaRodaje--;

                siParar();

                fieldAreaRodaje.setText(String.join(", ", areaRodaje));

                log.writeLog("El avion " + aThis.getIdFormateado() + " obtiene la puerta libre 1
para desembarcar");

                siParar();

                fieldGate2.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro()
+ ")" + "DESEMBARQUE ");

                contador = 1;

                desembarcarPasajeros(aThis, puertaLibre1, contador);
            }
        }
    }
}

```

```

    } catch (InterruptedException ex) {

        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);

    }

    break;

}

}

if (this.getPuertaLibre2().tryLock()) {

    if (caso == 1) {

        log.writeLog("El avion " + aThis.getIdFormateado() + " ha obtenido la puerta libre
2 para embarcar");

        areaEstacionamiento.remove(aThis.getIdFormateado());

        avionesAreaEstacionamiento--;

        if (areaEstacionamiento != null && !areaEstacionamiento.isEmpty()) {

            siParar();

            tFAreaEstacionamiento.setText(String.join(", ", areaEstacionamiento));

        } else {

            siParar();

            tFAreaEstacionamiento.setText("");

        }

        siParar();

        fieldGate3.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro() +
")" + "EMBARCA");

        contador = 2;

        break;

    } else {

        try {

            Thread.sleep((new Random().nextInt(5) * 1000) + 3000);

            quitarAvionAreaRodaje(aThis);

            avionesAreaRodaje--;

            siParar();

            fieldAreaRodaje.setText(String.join(", ", areaRodaje));

        }
    }
}

```

```

        log.writeLog("El avion " + aThis.getIdFormateado() + " obtiene la puerta libre 2
para desembarcar");

        siParar();

        fieldGate3.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro()
+ ")" + " DESEMBARQUE ");

        contador = 2;

        desembarcarPasajeros(aThis, puertaLibre2, contador);

    } catch (InterruptedException ex) {

        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);

    }

    break;

}

}

if (this.getPuertaLibre3().tryLock()) {

    if (caso == 1) {

        log.writeLog("El avion " + aThis.getIdFormateado() + " ha obtenido la puerta libre
3 para embarcar");

        areaEstacionamiento.remove(aThis.getIdFormateado());

        avionesAreaEstacionamiento--;

        if (areaEstacionamiento != null && !areaEstacionamiento.isEmpty()) {

            siParar();

            tFAreaEstacionamiento.setText(String.join(", ", areaEstacionamiento));

        } else {

            siParar();

            tFAreaEstacionamiento.setText("");

        }

        siParar();

        fieldGate4.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro() +
")" + " EMBARCA");

        contador = 3;

        break;

    } else {

```

```

try {
    Thread.sleep((new Random().nextInt(5) * 1000) + 3000);
    quitarAvionAreaRodaje(aThis);
    avionesAreaRodaje--;
    siParar();
    fieldAreaRodaje.setText(String.join(", ", areaRodaje));
    log.writeLog("El avion " + aThis.getIdFormateado() + " obtiene la puerta libre 3
para desembarcar");
    siParar();
    fieldGate4.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro()
+ ")" + " DESEMBARQUE ");
    contador = 3;
    desembarcarPasajeros(aThis, puertaLibre3, contador);
} catch (InterruptedException ex) {
    Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);
}
break;
}
}

if (this.getPuertaLibre4().tryLock()) {
    if (caso == 1) {
        log.writeLog("El avion " + aThis.getIdFormateado() + " ha obtenido la puerta libre
4 para embarcar");
        areaEstacionamiento.remove(aThis.getIdFormateado());
        avionesAreaEstacionamiento--;
        if (areaEstacionamiento != null && !areaEstacionamiento.isEmpty()) {
            siParar();
            tFAreaEstacionamiento.setText(String.join(", ", areaEstacionamiento));
        } else {
            siParar();
            tFAreaEstacionamiento.setText("");
        }
    }
}

```

```

        fieldGate5.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro() +
        ")" + "EMBARCA");

        contador = 4;

        break;
    } else {
        try {
            Thread.sleep((new Random().nextInt(5) * 1000) + 3000);

            quitarAvionAreaRodaje(aThis);

            avionesAreaRodaje--;

            siParar();

            fieldAreaRodaje.setText(String.join(", ", areaRodaje));

            log.writeLog("El avion " + aThis.getIdFormateado() + " obtiene la puerta libre 4
            para desembarcar");

            siParar();

            fieldGate5.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro()
            + ")" + " DESEMBARQUE ");

            contador = 4;

            desembarcarPasajeros(aThis, puertaLibre4, contador);
        } catch (InterruptedException ex) {

            Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);
        }

        break;
    }
}

if (caso == 2) {
    if (this.getPuertaDesembarque().tryLock()) {
        try {
            Thread.sleep((new Random().nextInt(5) * 1000) + 3000);

            quitarAvionAreaRodaje(aThis);

            avionesAreaRodaje--;

            siParar();

            fieldAreaRodaje.setText(String.join(", ", areaRodaje));

```



```

        log.writeLog("El avion " + aThis.getIdFormateado() + " obtiene la puerta de
desembarque");

        siParar();

        fieldGate6.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro()
+ ")" + " DESEMBARQUE ");

        contador = 0;

        desembarcarPasajeros(aThis, puertaDesembarque, contador);

    } catch (InterruptedException ex) {

        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);

    }

    break;

}

}

}

log.writeLog("El avion: " + aThis.getIdFormateado() + " bloquea la puerta: " + contador
+ " del caso: " + caso);

return contador;

}

```

```

private synchronized void cerrarBloqueoPuertas(Avion aThis, int contador, int caso) {

    siParar();

    try {

        if (caso == 1) {

            switch (contador) {

                case 0 ->

                    this.getPuertaEmbarque().unlock();

                case 1 ->

                    this.getPuertaLibre1().unlock();

                case 2 ->

                    this.getPuertaLibre2().unlock();

                case 3 ->

                    this.getPuertaLibre3().unlock();

```

```

        case 4 ->
            this.getPuertaLibre4().unlock();
        default -> {
        }
    }
} else {
    switch (contador) {
        case 0 ->
            this.getPuertaDesembarque().unlock();
        case 1 ->
            this.getPuertaLibre1().unlock();
        case 2 ->
            this.getPuertaLibre2().unlock();
        case 3 ->
            this.getPuertaLibre3().unlock();
        case 4 ->
            this.getPuertaLibre4().unlock();
        default -> {
        }
    }
}
} catch (Exception ex) {
    Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);
}

log.writeLog("El avion: " + aThis.getIdFormateado() + " desbloquea la puerta: " +
contador + "del caso: " + caso);
}

public void AreaEstacionamiento(Avion aThis, int caso) {
    siParar();

    int contador_puertas = 0;

```

```

if (caso == 1) {
    try {
        int contador = 0;
        areaEstacionamiento.add(aThis.getIdFormateado());
        avionesAreaEstacionamiento++;
        siParar();
        tFAreaEstacionamiento.setText(String.join(", ", areaEstacionamiento));
        Thread.sleep((new Random().nextInt(1) * 1000) + 1000);
        //Hasta que no sea el primero de la FIFO, se queda esperando
        boolean continua = true;
        Thread.sleep(1500);
        String primeroColaEstacionamiento =
this.obtenerPrimeroColaAreaEstacionamiento();
        if (primeroColaEstacionamiento != null) {
            if (primeroColaEstacionamiento.equals(aThis.getIdFormateado())) {
                continua = false;
            }
        }
        //Espera a que una de las puertas de embarque esté libre
        contador = this.controlBloqueoPuertas(aThis, 1);

        //Embarcamos los pasajeros
        embarcarPasajeros(aThis, contador);

        //El avion deja libre la puerta de embarque
        log.writeLog("El avión " + aThis.getIdFormateado() + " cierra sus puertas de
embarque y sale de la puerta de embarque " + contador + 1);
        siParar();
        switch (contador) {
            case 0 ->
                fieldGate1.setText("");
            case 1 ->

```

```

        fieldGate2.setText("");
    case 2 ->
        fieldGate3.setText("");
    case 3 ->
        fieldGate4.setText("");
    case 4 ->
        fieldGate5.setText("");
    default -> {
    }
}

contador_puertas = contador;
} catch (InterruptedException ex) {
    Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);
} finally {
    this.cerrarBloqueoPuertas(aThis, contador_puertas, caso);
}
} else if (caso == 2) {
    try {
        lockEstacionamiento.lock();

        areaEstacionamiento.add(aThis.getIdFormateado());

        avionesAreaEstacionamiento++;

        siParar();

        tFAreaEstacionamiento.setText(String.join(", ", areaEstacionamiento));

        Thread.sleep((int) (Math.random() * 5000 + 1000)); //Comprobaciones
    } catch (InterruptedException ex) {
        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        lockEstacionamiento.unlock();
    }
}
}
}

```

```

public void embarcarPasajeros(Avion aThis, int contador) {

    siParar();

    int intentos = 0;

    while (intentos < 3) {

        int numeroPasajerosFaltanEmbarcar = aThis.getCapacidadMaxima() -
aThis.getPasajerosDentro();

        if (numeroPasajerosFaltanEmbarcar <= this.getPersonasDentro()) {

            try{

                aThis.setPasajerosDentro(aThis.getPasajerosDentro() +
numeroPasajerosFaltanEmbarcar);

                log.writeLog("En el avión " + aThis.getIdFormateado() + " han embarcado " +
aThis.getPasajerosDentro() + " pasajeros.");

                this.setPersonasDentro(this.getPersonasDentro() -
numeroPasajerosFaltanEmbarcar);

                actualizarPersonasDentro(this.getPersonasDentro());

                siParar();

                switch (contador) {

                    case 0 ->

                        fieldGate1.setText(aThis.getIdFormateado() + "(" +
aThis.getPasajerosDentro() + ")" + " EMBARQUE ");

                    case 1 ->

                        fieldGate2.setText(aThis.getIdFormateado() + "(" +
aThis.getPasajerosDentro() + ")" + " EMBARQUE ");

                    case 2 ->

                        fieldGate3.setText(aThis.getIdFormateado() + "(" +
aThis.getPasajerosDentro() + ")" + " EMBARQUE ");

                    case 3 ->

                        fieldGate4.setText(aThis.getIdFormateado() + "(" +
aThis.getPasajerosDentro() + ")" + " EMBARQUE ");

                    case 4 ->

                        fieldGate5.setText(aThis.getIdFormateado() + "(" +
aThis.getPasajerosDentro() + ")" + " EMBARQUE ");

                }

            }

        }

    }

}

```

```

        Thread.sleep((new Random()).nextInt(3) * 1000) + 1000);
    } catch (InterruptedException ex) {
        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);
    }
    break;
} else {
    try {
        int numeroPersonasEmbarque = this.getPersonasDentro();

        aThis.setPasajerosDentro(aThis.getPasajerosDentro() +
numeroPersonasEmbarque);

        log.writeLog("En el avión " + aThis.getIdFormateado() + " han embarcado " +
aThis.getPasajerosDentro() + " pasajeros.");

        this.setPersonasDentro(this.getPersonasDentro() - numeroPersonasEmbarque);
        actualizarPersonasDentro(this.getPersonasDentro());

        Thread.sleep((new Random()).nextInt(5) * 1000) + 1000);
    } catch (InterruptedException ex) {
        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);
    }
}
intentos++;
}
}

```

```

public void AreaRodaje(Avion aThis, int caso) {
    siParar();
    if (caso == 1) {
        try {
            for (String a : areaRodaje) {
                if (a == null) {
                    a = "";
                }
            }
        }
    }
}

```

```

        log.writeLog(aThis.getIdFormateado() + " accedió al ÁREA DE RODAJE.");

        annadirAvionAreaRodaje(aThis);

        avionesAreaRodaje++;

        siParar();

        fieldAreaRodaje.setText(String.join(", ", areaRodaje));

        Thread.sleep((new Random().nextInt(5) * 1000) + 1000); // Los pilotos realizan
comprobaciones

    } catch (InterruptedException ex) {

        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);

    }

} else if (caso == 2) {

    try {

        log.writeLog("El avión " + aThis.getIdFormateado() + " accedió al ÁREA DE RODAJE
para dirigirse a la PUERTA DE EMBARQUE.");

        annadirAvionAreaRodaje(aThis);

        avionesAreaRodaje++;

        siParar();

        fieldAreaRodaje.setText(String.join(", ", areaRodaje));

        Thread.sleep((new Random().nextInt(5) * 1000) + 1000);

        int contador = this.controlBloqueoPuertas(aThis, 2);

        this.cerrarBloqueoPuertas(aThis, contador, 2);

    } catch (InterruptedException ex) {

        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);

    }

}

}

```

```

public void solicitarPista(Avion aThis) {

    siParar();

    int contador;

    boolean variable = true;

    while (variable) {

```

```

contador = 0;
for (Lock l : Pistas) {
    /*
    switch (contador) {
        case 0 ->
            pistaCerrada = pista1Cerrada;
        case 1 ->
            pistaCerrada = pista2Cerrada;
        case 2 ->
            pistaCerrada = pista3Cerrada;
        case 3 ->
            pistaCerrada = pista4Cerrada;
        default -> {
            }
        }
    }
*/

    if (l.tryLock() /*&& !pistaCerrada*/) {
        //Sale del Area de Rodaje
        quitarAvionAreaRodaje(aThis);
        avionesAreaRodaje--;
        if (areaRodaje != null && !areaRodaje.isEmpty()) {
            siParar();
            fieldAreaRodaje.setText(String.join(", ", areaRodaje));
        } else {
            siParar();
            fieldAreaRodaje.setText("");
        }
        //Acceder a la Pista
        log.writeLog("El avion " + aThis.getIdFormateado() + " accede a una Pista para
despegar");
        siParar();
    }
}

```



```

switch (contador) {

    case 0 ->

        fieldPista1.setText(aThis.getIdFormateado() + "(" +
aThis.getPasajerosDentro() + ")" + " DESPEGUE ");

        case 1 ->

            fieldPista2.setText(aThis.getIdFormateado() + "(" +
aThis.getPasajerosDentro() + ")" + " DESPEGUE ");

            case 2 ->

                fieldPista3.setText(aThis.getIdFormateado() + "(" +
aThis.getPasajerosDentro() + ")" + " DESPEGUE ");

                case 3 ->

                    fieldPista4.setText(aThis.getIdFormateado() + "(" +
aThis.getPasajerosDentro() + ")" + " DESPEGUE ");

                    default -> {

                        }

                    }

                despegar(aThis, contador);

                variable = false;

                break;

            } else {

                contador++;

            }

        }

    }

}

```

```

public void despegar(Avion aThis, int contador) {

    siParar();

    try{

        // Realizar verificaciones antes del despegue

        log.writeLog("El avión " + aThis.getIdFormateado() + " está realizando verificaciones
antes del despegue.");

        Thread.sleep((new Random().nextInt(3) * 1000) + 1000);
    }
}

```

```

log.writeLog("El avión " + aThis.getIdFormateado() + " despegó.");
Thread.sleep((new Random()).nextInt(5) * 1000 + 1000);
siParar();
switch (contador) {
    case 0 ->
        fieldPista1.setText("");
    case 1 ->
        fieldPista2.setText("");
    case 2 ->
        fieldPista3.setText("");
    case 3 ->
        fieldPista4.setText("");
    default -> {
    }
}
} catch (InterruptedException ex) {
    Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);
} finally {
    //Libera el lock de la pista
    this.getPistas().get(contador).unlock();
}
}

```

```

public void accederAerovia(Avion aThis) {
    siParar();
    try {
        aThis.getOrigen().entrarAerovia(aThis);
        if (aThis.getOrigen().equals("Madrid")) {
            log.writeLog("El avión " + aThis.getIdFormateado() + " accedió a la AEROVÍA MADRID-BARCELONA");
        } else {

```

```

        log.writeLog("El avión " + aThis.getIdFormateado() + " accedió a la AEROVÍA
BARCELONA-MADRID");
    }
    Thread.sleep((new Random().nextInt(16) * 1000) + 15000);
    aThis.getDestino().solicitarPistaAterrizaje(aThis);
} catch (InterruptedException ex) {
    Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);
}
}

```

```

public synchronized void entrarAerovia(Avion aThis) {
    siParar();
    annadirAvionAerovia(aThis);
    fieldAerovia.setText(String.join(", ", aerovia));
}

```

```

public synchronized void salirAerovia(Avion aThis) {
    siParar();
    aThis.getOrigen().quitarAvionAerovia(aThis);
    aThis.getOrigen().fieldAerovia.setText(String.join(", ", aerovia));
}

```

```

public void solicitarPistaAterrizaje(Avion aThis) {
    siParar();
    int contador = 0;
    boolean variable = true;
    while (variable) {
        try {
            contador = 0;
            for (Lock l : Pistas) {

```

```

        if (l.tryLock()) {

            aThis.getOrigen().salirAerovia(aThis);

            log.writeLog("El avion " + aThis.getIdFormateado() + " accede a una Pista para aterrizar");

            siParar();

            switch (contador) {

                case 0 ->

                    fieldPista1.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro() + ")" + " ATERRIZAJE ");

                case 1 ->

                    fieldPista2.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro() + ")" + " ATERRIZAJE ");

                case 2 ->

                    fieldPista3.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro() + ")" + " ATERRIZAJE ");

                case 3 ->

                    fieldPista4.setText(aThis.getIdFormateado() + "(" + aThis.getPasajerosDentro() + ")" + " ATERRIZAJE ");

                default -> {

                    }

            }

            // Aterrizar

            aterrizar(aThis, l, contador);

            variable = false;

            break;

        } else {

            contador++;

        }

    }

    Thread.sleep((int) (Math.random() * 5000 + 1000));

} catch (InterruptedException ex) {

    Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);

}

```

```
}  
}
```

```
public void aterrizar(Avion aThis, Lock l2, int contador) {  
    siParar();  
    try {  
        log.writeLog("El avión " + aThis.getIdFormateado() + " aterrizó en una pista");  
        Thread.sleep((new Random().nextInt(5) * 1000) + 1000);  
        siParar();  
        //Sale de la Pista  
        switch (contador) {  
            case 0 ->  
                fieldPista1.setText("");  
            case 1 ->  
                fieldPista2.setText("");  
            case 2 ->  
                fieldPista3.setText("");  
            case 3 ->  
                fieldPista4.setText("");  
            default -> {  
            }  
        }  
    } catch (InterruptedException ex) {  
        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);  
    } finally {  
        //Libera el lock de la pista  
        this.getPistas().get(contador).unlock();  
    }  
}
```

```
public void desembarcarPasajeros(Avion aThis, Lock l3, int contador) {
```

```

        siParar();

    try {

        // Desembarcar pasajeros

        log.writeLog("Han desembarcado " + aThis.getPasajerosDentro() + " pasajeros del
avión " + aThis.getIdFormateado());

        this.setPersonasDentro(this.getPersonasDentro() + aThis.getPasajerosDentro());

        actualizarPersonasDentro(personasDentro);

        aThis.setPasajerosDentro(0);

        Thread.sleep((int) (Math.random() * 5000 + 1000));

        siParar();

        switch (contador) {

            case 0 ->

                fieldGate6.setText("");

            case 1 ->

                fieldGate2.setText("");

            case 2 ->

                fieldGate3.setText("");

            case 3 ->

                fieldGate4.setText("");

            case 4 ->

                fieldGate5.setText("");

            default -> {

                }

            }

        } catch (InterruptedException ex) {

            Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);

        }

    }

}

public void inspeccion(Avion aThis, int i) {

    siParar();

```

```

try{

    taller.acquire();

    areaEstacionamiento.remove(aThis.getIdFormateado());

    avionesAreaEstacionamiento--;

    avionesTaller++;

    if (areaEstacionamiento != null && !areaEstacionamiento.isEmpty()) {

        siParar();

        tFareaEstacionamiento.setText(String.join(", ", areaEstacionamiento));

    } else {

        siParar();

        tFareaEstacionamiento.setText("");

    }

    if (i == 1) {

        fieldTaller.setText(aThis.getIdFormateado());

        Thread.sleep((int) (Math.random() * 10000 + 5000));

        log.writeLog("El avión " + aThis.getIdFormateado() + " termina su inspeccion en
profundidad y se va del Taller");

        pasarPuertaTaller(aThis);

        siParar();

        fieldTaller.setText("");

    } else {

        fieldTaller.setText(aThis.getIdFormateado());

        Thread.sleep((int) (Math.random() * 10000 + 5000));

        log.writeLog("El avión " + aThis.getIdFormateado() + " termina su inspeccion
rápida y se va del Taller");

        pasarPuertaTaller(aThis);

        siParar();

        fieldTaller.setText("");

    }

    avionesTaller--;

    taller.release();

} catch (InterruptedException ex) {

```

```

        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);
    }
}

void pasarPuertaTaller(Avion aThis) {
    siParar();
    try {
        puertaTaller.lock();

        log.writeLog("El avión " + aThis.getIdFormateado() + " pasa por la puerta del Taller");

        Thread.sleep((int) (Math.random() * 1000));
    } catch (InterruptedException ex) {
        Logger.getLogger(Aeropuerto.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        puertaTaller.unlock();
    }
}

```

///GETTERS Y SETTERS

```

public int getPersonasDentro() {
    return personasDentro;
}

public void setPersonasDentro(int personasDentro) {
    siParar();

    this.personasDentro = personasDentro;

    TFPersonasDentro.setText(String.format("%s", personasDentro));
}

public String getNombre() {
    return nombre;
}

```



```
public void setNombre(String nombre) {  
    this.nombre = nombre;  
}
```

```
public LinkedList<String> getAreaEstacionamiento() {  
    return areaEstacionamiento;  
}
```

```
public void setAreaEstacionamiento(LinkedList<String> areaEstacionamiento) {  
    this.areaEstacionamiento = areaEstacionamiento;  
}
```

```
public Lock getLockEstacionamiento() {  
    return lockEstacionamiento;  
}
```

```
public void setLockEstacionamiento(Lock lockEstacionamiento) {  
    this.lockEstacionamiento = lockEstacionamiento;  
}
```

```
public Condition getAlgunoOcupado() {  
    return algunoOcupado;  
}
```

```
public void setAlgunoOcupado(Condition algunoOcupado) {  
    this.algunoOcupado = algunoOcupado;  
}
```

```
public Semaphore getTaller() {  
    return taller;
```

```
}
```

```
public void setTaller(Semaphore taller) {  
    this.taller = taller;  
}
```

```
public Lock getPuertaTaller() {  
    return puertaTaller;  
}
```

```
public void setPuertaTaller(Lock puertaTaller) {  
    this.puertaTaller = puertaTaller;  
}
```

```
public Lock getPuertaEmbarque() {  
    return puertaEmbarque;  
}
```

```
public void setPuertaEmbarque(Lock puertaEmbarque) {  
    this.puertaEmbarque = puertaEmbarque;  
}
```

```
public Lock getPuertaDesembarque() {  
    return puertaDesembarque;  
}
```

```
public void setPuertaDesembarque(Lock puertaDesembarque) {  
    this.puertaDesembarque = puertaDesembarque;  
}
```

```
public Lock getPuertaLibre1() {
```

```
        return puertaLibre1;
    }
}
```

```
public void setPuertaLibre1(Lock puertaLibre1) {
    this.puertaLibre1 = puertaLibre1;
}
}
```

```
public Lock getPuertaLibre2() {
    return puertaLibre2;
}
}
```

```
public void setPuertaLibre2(Lock puertaLibre2) {
    this.puertaLibre2 = puertaLibre2;
}
}
```

```
public Lock getPuertaLibre3() {
    return puertaLibre3;
}
}
```

```
public void setPuertaLibre3(Lock puertaLibre3) {
    this.puertaLibre3 = puertaLibre3;
}
}
```

```
public Lock getPuertaLibre4() {
    return puertaLibre4;
}
}
```

```
public void setPuertaLibre4(Lock puertaLibre4) {
    this.puertaLibre4 = puertaLibre4;
}
}
```

```
public List<Lock> getPuertas() {  
    return PuertasEmbarque;  
}
```

```
public void setPuertas(List<Lock> Puertas) {  
    this.PuertasEmbarque = Puertas;  
}
```

```
public List<Lock> getPuertasDesembarque() {  
    return PuertasDesembarque;  
}
```

```
public void setPuertasDesembarque(List<Lock> PuertasDesembarque) {  
    this.PuertasDesembarque = PuertasDesembarque;  
}
```

```
public Lock getPista1() {  
    return pista1;  
}
```

```
public void setPista1(Lock pista1) {  
    this.pista1 = pista1;  
}
```

```
public Lock getPista2() {  
    return pista2;  
}
```

```
public void setPista2(Lock pista2) {  
    this.pista2 = pista2;  
}
```

```
public Lock getPista3() {  
    return pista3;  
}
```

```
public void setPista3(Lock pista3) {  
    this.pista3 = pista3;  
}
```

```
public Lock getPista4() {  
    return pista4;  
}
```

```
public void setPista4(Lock pista4) {  
    this.pista4 = pista4;  
}
```

```
public List<Lock> getPistas() {  
    return Pistas;  
}
```

```
public void setPistas(List<Lock> Pistas) {  
    this.Pistas = Pistas;  
}
```

```
public Lock getLock() {  
    return lock;  
}
```

```
public void setLock(Lock lock) {  
    this.lock = lock;
```

```
}
```

```
public Log getLog() {  
    return log;  
}
```

```
public void setLog(Log log) {  
    this.log = log;  
}
```

```
public Condition getParado() {  
    return parado;  
}
```

```
public void setParado(Condition parado) {  
    this.parado = parado;  
}
```

```
public void parar() {  
    this.parar = true;  
}
```

```
public boolean mirarSiParar() {  
    return this.parar;  
}
```

```
public synchronized void continuar() {  
    parar = false;  
    notifyAll();  
}
```

```
public List<Lock> getPuertasEmbarque() {  
    return PuertasEmbarque;  
}
```

```
public List<String> getHangar() {  
    return hangar;  
}
```

```
public void setHangar(List<String> hangar) {  
    this.hangar = hangar;  
}
```

```
public void annadirAvionHangar(Avion avion) {  
    hangar.add(avion.getIdFormateado());  
}
```

```
public void quitarAvionHangar(Avion avion) {  
    hangar.remove(avion.getIdFormateado());  
}
```

```
public List<String> getAreaRodaje() {  
    return areaRodaje;  
}
```

```
public void setAreaRodaje(List<String> areaRodaje) {  
    this.areaRodaje = areaRodaje;  
}
```

```
public void annadirAvionAreaRodaje(Avion avion) {  
    areaRodaje.add(avion.getIdFormateado() + "(" + avion.getPasajerosDentro() + ")");  
}
```

```
public void quitarAvionAreaRodaje(Avion avion) {  
    areaRodaje.remove(avion.getIdFormateado() + "(" + avion.getPasajerosDentro() + ")");  
}
```

```
public List<String> getAerovia() {  
    return aerovia;  
}
```

```
public void setAerovia(List<String> aerovia) {  
    this.aerovia = aerovia;  
}
```

```
public void annadirAvionAerovia(Avion avion) {  
    aerovia.add(avion.getIdFormateado() + "(" + avion.getPasajerosDentro() + ")");  
}
```

```
public void quitarAvionAerovia(Avion avion) {  
    aerovia.remove(avion.getIdFormateado() + "(" + avion.getPasajerosDentro() + ")");  
}
```

```
public int getAvionesHangar() {  
    return avionesHangar;  
}
```

```
public void setAvionesHangar(int avionesHangar) {  
    this.avionesHangar = avionesHangar;  
}
```

```
public int getAvionesTaller() {  
    return avionesTaller;  
}
```



```
}
```

```
public void setAvionesTaller(int avionesTaller) {  
    this.avionesTaller = avionesTaller;  
}
```

```
public int getAvionesAreaRodaje() {  
    return avionesAreaRodaje;  
}
```

```
public void setAvionesAreaRodaje(int avionesAreaRodaje) {  
    this.avionesAreaRodaje = avionesAreaRodaje;  
}
```

```
public int getAvionesAreaEstacionamiento() {  
    return avionesAreaEstacionamiento;  
}
```

```
public void setAvionesAreaEstacionamiento(int avionesAreaEstacionamiento) {  
    this.avionesAreaEstacionamiento = avionesAreaEstacionamiento;  
}
```

```
public boolean isPista1Cerrada() {  
    return pista1Cerrada;  
}
```

```
public void setPista1Cerrada(boolean pista1Cerrada) {  
    this.pista1Cerrada = pista1Cerrada;  
}
```

```
public boolean isPista2Cerrada() {
```

```
    return pista2Cerrada;
}
```

```
public void setPista2Cerrada(boolean pista2Cerrada) {
    this.pista2Cerrada = pista2Cerrada;
}
```

```
public boolean isPista3Cerrada() {
    return pista3Cerrada;
}
```

```
public void setPista3Cerrada(boolean pista3Cerrada) {
    this.pista3Cerrada = pista3Cerrada;
}
```

```
public boolean isPista4Cerrada() {
    return pista4Cerrada;
}
```

```
public void setPista4Cerrada(boolean pista4Cerrada) {
    this.pista4Cerrada = pista4Cerrada;
}
```

```
public boolean isPistaCerrada() {
    return pistaCerrada;
}
```

```
public void setPistaCerrada(boolean pistaCerrada) {
    this.pistaCerrada = pistaCerrada;
}
```

```
}
```

```
package Concurrida;
```

```
import java.util.Random;
```

```
import java.util.concurrent.locks.Lock;
```

```
import javax.swing.JTextField;
```

```
// Clase para representar un autobús
```

```
public class Autobus extends Thread {
```

```
    private long id;
```

```
    private String prefijoID;
```

```
    private String idFormateado;    //
```

```
    private Aeropuerto aeropuerto;
```

```
    private int pasajerosSubidos = 0;
```

```
    Autobus(String prefijoID, Long idAutobus, Aeropuerto aeropuerto) {
```

```
        this.prefijoID = prefijoID;
```

```
        this.id = idAutobus;
```

```
        this.idFormateado = this.getPrefijoID() + String.format("%04d", this.getId());
```

```
        this.aeropuerto = aeropuerto;
```

```
    }
```

```
    // @Override
```

```
    public void run() {
```

```
        while (true) {
```

```
        // Llegada a la parada del centro de la ciudad
        aeropuerto.paradaCiudad(this);
        // Viaje al aeropuerto
        aeropuerto.viajarAeropuerto(this);
        // Llegada a la parada del aeropuerto
        aeropuerto.paradaAeropuerto(this);
        // Viaje a la ciudad
        aeropuerto.viajarCiudad(this);
    }
}
```

```
public Aeropuerto getAeropuerto() {
    return aeropuerto;
}
```

```
public void setAeropuerto(Aeropuerto aeropuerto) {
    this.aeropuerto = aeropuerto;
}
```

```
public long getId() {
    return id;
}
```

```
public void setId(long id) {
    this.id = id;
}
```

```
public String getPrefijoID() {
    return prefijoID;
}
```

```
public void setPrefijoID(String prefijoID) {
    this.prefijoID = prefijoID;
}

public String getIdFormateado() {
    return idFormateado;
}

public void setIdFormateado(String idFormateado) {
    this.idFormateado = idFormateado;
}

public int getPasajerosSubidos() {
    return pasajerosSubidos;
}

public void setPasajerosSubidos(int pasajerosSubidos) {
    this.pasajerosSubidos = pasajerosSubidos;
}

}

package Concurrida;

import java.util.Random;

// Clase para representar un avión
public class Avion extends Thread {

    private long id;
    private String prefijoID;
    private String idFormateado;
```

```
private int capacidadMaxima;
```

```
private int pasajerosDentro;
```

```
private int vuelos = 0;
```

```
private Aeropuerto origen, destino;
```

```
private Aeropuerto copia;
```

```
private Log log;
```

```
Avion(Aeropuerto Origen, Aeropuerto Destino, String prefijoID, long idAvion, int  
capacidad, Log log) {
```

```
    this.id = idAvion;
```

```
    this.prefijoID = prefijoID;
```

```
    this.idFormateado = prefijoID + String.format("%04d", idAvion);
```

```
    this.capacidadMaxima = capacidad;
```

```
    this.origen = Origen;
```

```
    this.destino = Destino;
```

```
    this.log = log;
```

```
}
```

```
//@Override
```

```
public void run() {
```

```
    boolean primeraVez = true;
```

```
    while (true) {
```

```
        try {
```

```
            if(primeravez){
```

```
                primeraVez = false;
```

```
            }else{
```

```
                this.setCopia(this.getDestino());
```

```
                this.setDestino(this.getOrigen());
```

```

        this.setOrigen(this.getCopia());

        this.setCopia(null);
    }

    //Aparece en el Hangar
    origen.entrarHangar(this);

    Thread.sleep(2000);

    origen.salirHangar(this);

    // Acceder al ÁREA DE ESTACIONAMIENTO Y POSTERIORMENTE A LA PUERTA DE
    EMBARQUE
    origen.AreaEstacionamiento(this, 1);

    // Acceder y salir del ÁREA DE RODAJE
    origen.AreaRodaje(this, 1);

    //Solicita pista para el despegue y despega
    origen.solicitarPista(this);

    // Accede a la AEROVÍA correspondiente y aterriza en la pista que le concedan
    origen.accederAerovia(this);

    vuelos ++;

    // Acceder al ÁREA DE RODAJE, solicita PUERTA DESEMBARQUE y DESEMBARCA
    LOS PASAJEROS
    destino.AreaRodaje(this, 2);

    //Accede al Area de Estacionamiento
    destino.AreaEstacionamiento(this, 2);

    // Realizar inspección en el taller después de 15 vuelos
    if (vuelos == 15) {

```

```

        destino.pasarPuertaTaller(this);

        destino.inspeccion(this, 1);

        vuelos = 0;
    } else {

        destino.pasarPuertaTaller(this);

        destino.inspeccion(this, 2);

    }

    // Decidir si ir al HANGAR o continuar con el ciclo de vida
    if (new Random().nextBoolean()) {

        log.writeLog("El avión " + idFormateado + " va al HANGAR a reposar.");
//
        //Thread.sleep((int)(Math.random()*15000+15000));
        Thread.sleep((new Random().nextInt(3)*10000) + 15000);

    }

} catch (InterruptedException e) {

    e.printStackTrace();

}

}

}

////////GETTERS Y SETTERS////////

public int getCapacidadMaxima() {

    return capacidadMaxima;

}

public void setCapacidadMaxima(int capacidadMaxima) {

    this.capacidadMaxima = capacidadMaxima;

}

```



```
public int getPasajerosDentro() {  
    return pasajerosDentro;  
}
```

```
public void setPasajerosDentro(int pasajerosDentro) {  
    this.pasajerosDentro = pasajerosDentro;  
}
```

```
public int getVuelos() {  
    return vuelos;  
}
```

```
public void setVuelos(int vuelos) {  
    this.vuelos = vuelos;  
}
```

```
public Aeropuerto getOrigen() {  
    return origen;  
}
```

```
public void setOrigen(Aeropuerto aeropuerto) {  
    this.origen = aeropuerto;  
}
```

```
public Aeropuerto getDestino() {  
    return destino;  
}
```

```
public void setDestino(Aeropuerto aeropuerto) {  
    this.destino = aeropuerto;  
}
```

```
public long getId() {  
    return id;  
}
```

```
public void setId(long id) {  
    this.id = id;  
}
```

```
public String getPrefijoID() {  
    return prefijoID;  
}
```

```
public void setPrefijoID(String prefijoID) {  
    this.prefijoID = prefijoID;  
}
```

```
public String getIdFormateado() {  
    return idFormateado;  
}
```

```
public void setIdFormateado(String idFormateado) {  
    this.idFormateado = idFormateado;  
}
```

```
public Aeropuerto getCopia() {  
    return copia;  
}
```

```
public void setCopia(Aeropuerto copia) {  
    this.copia = copia;  
}
```

```
}
```

```
}
```

```
package Concurrida;
```

```
import Distribuida.Servidor;
```

```
import static java.lang.constant.ConstantDescs.NULL;
```

```
import java.net.MalformedURLException;
```

```
import java.util.Random;
```

```
import java.util.concurrent.locks.Lock;
```

```
import java.util.concurrent.locks.ReentrantLock;
```

```
import java.util.logging.Level;
```

```
import java.util.logging.Logger;
```

```
public class InterfazServidor extends javax.swing.JFrame {
```

```
    Aeropuerto madrid, barcelona;
```

```
    Log log = new Log();
```

```
    Lock lockM = new ReentrantLock();
```

```
    Lock lockB = new ReentrantLock();
```

```
    Lock lockAM = new ReentrantLock();
```

```
    Lock lockAB = new ReentrantLock();
```

```
    Sistema sistema;
```

```
    Servidor servidor;
```

```
    /**
```

```
     * Creates new form Interfaz
```

```
     */
```

```
    public InterfazServidor() {
```

```
        initComponents();
```

```

        madrid = new Aeropuerto(lockM, lockAM, log, TransfersAeropuertoMadrid,
TransfersCiudadMadrid, N°PasajerosMadrid,

        txtHangarMadrid, txtTallerMadrid, txtAreaEstacionMadrid, txtGate1Madrid,
txtGate2Madrid, txtGate3Madrid, txtGate4Madrid,

        txtGate5Madrid, txtGate6Madrid, txtAreaRodajeMadrid, txtPista1Madrid,
txtPista2Madrid, txtPista3Madrid, txtPista4Madrid,

        AeroviaMadrid_Barcelona, "Madrid");

        barcelona = new Aeropuerto(lockB, lockAB, log, TransfersAeropuertoBarcelona,
TransfersCiudadBarcelona, N°PasajerosBarcelona,

        txtHangarBarcelona, txtTallerBarcelona, txtAreaEstacionBarcelona,
txtGate1Barcelona, txtGate2Barcelona, txtGate3Barcelona,

        txtGate4Barcelona, txtGate5Barcelona, txtGate6Barcelona,
txtAreaRodajeBarcelona, txtPista1Barcelona, txtPista2Barcelona,

        txtPista3Barcelona, txtPista4Barcelona, AeroviaBarcelona_Madrid, "Barcelona");

        sistema = new Sistema(madrid, barcelona);
        sistema.start();

        servidor = new Servidor(this);
        Thread t = new Thread(servidor);
        t.start();
    }

    public Aeropuerto getAeropuertoMadrid() {
        return madrid;
    }

    public void setAeropuertoMadrid(Aeropuerto aeropuerto) {
        this.madrid = aeropuerto;
    }

```

```
public Aeropuerto getAeropuertoBarcelona() {  
    return barcelona;  
}
```

```
public void setAeropuertoBarcelona(Aeropuerto aeropuerto) {  
    this.barcelona = aeropuerto;  
}
```

```
public Servidor getServidor() {  
    return servidor;  
}
```

```
public void setServidor(Servidor servidor) {  
    this.servidor = servidor;  
}
```

```
public void windowClosing() {  
    log.close();  
}
```

```
/**
```

```
 * This method is called from within the constructor to initialize the form.
```

```
 * WARNING: Do NOT modify this code. The content of this method is always
```

```
 * regenerated by the Form Editor.
```

```
 */
```

```
@SuppressWarnings("unchecked")
```

```
// <editor-fold defaultstate="collapsed" desc="Generated Code">
```

```
private void initComponents() {
```

```
    TransfersAeropuertoMadrid = new javax.swing.JTextField();
```

```
    TransfersCiudadMadrid = new javax.swing.JTextField();
```

```
jLabel1 = new javax.swing.JLabel();
jLabel2 = new javax.swing.JLabel();
jLabel3 = new javax.swing.JLabel();
pausa = new javax.swing.JButton();
Reanudar = new javax.swing.JButton();
jLabel4 = new javax.swing.JLabel();
NºPasajerosMadrid = new javax.swing.JTextField();
jLabel5 = new javax.swing.JLabel();
txtHangarMadrid = new javax.swing.JTextField();
jLabel6 = new javax.swing.JLabel();
txtTallerMadrid = new javax.swing.JTextField();
jLabel7 = new javax.swing.JLabel();
txtAreaEstacionMadrid = new javax.swing.JTextField();
jLabel8 = new javax.swing.JLabel();
txtGate1Madrid = new javax.swing.JTextField();
jLabel9 = new javax.swing.JLabel();
txtGate2Madrid = new javax.swing.JTextField();
jLabel10 = new javax.swing.JLabel();
txtAreaRodajeMadrid = new javax.swing.JTextField();
txtGate5Madrid = new javax.swing.JTextField();
jLabel11 = new javax.swing.JLabel();
txtGate6Madrid = new javax.swing.JTextField();
jLabel12 = new javax.swing.JLabel();
txtGate4Madrid = new javax.swing.JTextField();
jLabel13 = new javax.swing.JLabel();
jLabel14 = new javax.swing.JLabel();
txtGate3Madrid = new javax.swing.JTextField();
jLabel15 = new javax.swing.JLabel();
jLabel16 = new javax.swing.JLabel();
txtPista1Madrid = new javax.swing.JTextField();
txtPista2Madrid = new javax.swing.JTextField();
```

```
jLabel17 = new javax.swing.JLabel();
jLabel18 = new javax.swing.JLabel();
txtPista3Madrid = new javax.swing.JTextField();
txtPista4Madrid = new javax.swing.JTextField();
jLabel19 = new javax.swing.JLabel();
AeroviaMadrid_Barcelona = new javax.swing.JTextField();
jLabel20 = new javax.swing.JLabel();
AeroviaBarcelona_Madrid = new javax.swing.JTextField();
txtPista4Barcelona = new javax.swing.JTextField();
txtGate2Barcelona = new javax.swing.JTextField();
jLabel21 = new javax.swing.JLabel();
txtAreaRodajeBarcelona = new javax.swing.JTextField();
txtGate5Barcelona = new javax.swing.JTextField();
jLabel22 = new javax.swing.JLabel();
txtGate6Barcelona = new javax.swing.JTextField();
jLabel23 = new javax.swing.JLabel();
txtGate4Barcelona = new javax.swing.JTextField();
jLabel24 = new javax.swing.JLabel();
jLabel25 = new javax.swing.JLabel();
NºPasajerosBarcelona = new javax.swing.JTextField();
TransfersAeropuertoBarcelona = new javax.swing.JTextField();
TransfersCiudadBarcelona = new javax.swing.JTextField();
jLabel26 = new javax.swing.JLabel();
jLabel27 = new javax.swing.JLabel();
jLabel28 = new javax.swing.JLabel();
txtGate3Barcelona = new javax.swing.JTextField();
jLabel29 = new javax.swing.JLabel();
jLabel30 = new javax.swing.JLabel();
txtHangarBarcelona = new javax.swing.JTextField();
jLabel31 = new javax.swing.JLabel();
jLabel32 = new javax.swing.JLabel();
```

```
jLabel33 = new javax.swing.JLabel();
txtTallerBarcelona = new javax.swing.JTextField();
txtPista1Barcelona = new javax.swing.JTextField();
jLabel34 = new javax.swing.JLabel();
txtPista2Barcelona = new javax.swing.JTextField();
txtAreaEstacionBarcelona = new javax.swing.JTextField();
jLabel35 = new javax.swing.JLabel();
jLabel36 = new javax.swing.JLabel();
jLabel37 = new javax.swing.JLabel();
txtGate1Barcelona = new javax.swing.JTextField();
txtPista3Barcelona = new javax.swing.JTextField();
jLabel38 = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

TransfersCiudadMadrid.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        TransfersCiudadMadridActionPerformed(evt);
    }
});

jLabel1.setText("Transfers Aeropuerto");

jLabel2.setText("Transfers Ciudad");

jLabel3.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
jLabel3.setText("Aeropuerto Madrid");

pausa.setText("Pausa");
pausa.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
```



```
        pausaActionPerformed(evt);  
    }  
});
```

```
Reanudar.setText("Reanudar");  
Reanudar.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        ReanudarActionPerformed(evt);  
    }  
});
```

```
jLabel4.setText("Nº de Pasajeros en Aeropuerto");
```

```
jLabel5.setText("Hangar");
```

```
jLabel6.setText("Taller");
```

```
jLabel7.setText("Area Estacionamiento");
```

```
jLabel8.setText("Gate 1 (Embarque)");
```

```
jLabel9.setText("Gate 2");
```

```
jLabel10.setText("Gate 3");
```

```
jLabel11.setText("Gate 6 (Desembarque)");
```

```
jLabel12.setText("Gate 4");
```

```
jLabel13.setText("Gate 5");
```

```
jLabel14.setText("Área Rodaje");
```

```
jLabel15.setText("Pista 1");
```

```
jLabel16.setText("Pista 2");
```

```
jLabel17.setText("Pista 3");
```

```
jLabel18.setText("Pista 4");
```

```
jLabel19.setText("Aerovía Madrid-Barcelona");
```

```
jLabel20.setText("Aerovía Barcelona-Madrid");
```

```
jLabel21.setText("Gate 3");
```

```
jLabel22.setText("Gate 6 (Desembarque)");
```

```
jLabel23.setText("Gate 4");
```

```
jLabel24.setText("Gate 5");
```

```
jLabel25.setText("Nº de Pasajeros en Aeropuerto");
```

```
TransfersCiudadBarcelona.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        TransfersCiudadBarcelonaActionPerformed(evt);  
    }  
});
```

```
jLabel26.setText("Transfers Aeropuerto");
```

```
jLabel27.setText("Área Rodaje");
```

```
jLabel28.setText("Transfers Ciudad");
```

```
jLabel29.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
```

```
jLabel29.setText("Aeropuerto Barcelona");
```

```
jLabel30.setText("Hangar");
```

```
jLabel31.setText("Pista 1");
```

```
jLabel32.setText("Taller");
```

```
jLabel33.setText("Pista 2");
```

```
jLabel34.setText("Area Estacionamiento");
```

```
jLabel35.setText("Pista 3");
```

```
jLabel36.setText("Gate 1 (Embarque)");
```

```
jLabel37.setText("Pista 4");
```

```
jLabel38.setText("Gate 2");
```

```
javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
```

```
getContentPane().setLayout(layout);
```

```
layout.setHorizontalGroup(
```

```
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
        .addGroup(layout.createSequentialGroup()
```

```

.addGap(28, 28, 28)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addComponent(AeroviaMadrid_Barcelona)
    .addComponent(AeroviaBarcelona_Madrid,
javax.swing.GroupLayout.Alignment.TRAILING))
    .addContainerGap())
    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addGap(190, 190, 190)
        .addComponent(jLabel3))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addComponent(jLabel14)
        .addGap(18, 18, 18)
        .addComponent(txtAreaRodajeMadrid))
    .addGroup(layout.createSequentialGroup()
        .addComponent(jLabel7)
        .addGap(18, 18, 18)
        .addComponent(txtAreaEstacionMadrid))
    .addGroup(layout.createSequentialGroup()
        .addComponent(jLabel6)
        .addGap(27, 27, 27)
        .addComponent(txtTallerMadrid))
    .addGroup(layout.createSequentialGroup()

```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,  
false)
```

```
    .addGroup(layout.createSequentialGroup()
```

```
        .addComponent(jLabel5)
```

```
        .addGap(18, 18, 18)
```

```
        .addComponent(txtHangarMadrid))
```

```
    .addGroup(layout.createSequentialGroup()
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,  
false)
```

```
    .addGroup(layout.createSequentialGroup()
```

```
        .addComponent(jLabel4)
```

```
        .addGap(18, 18, 18)
```

```
        .addComponent(NºPasajerosMadrid))
```

```
    .addGroup(layout.createSequentialGroup()
```

```
        .addComponent(jLabel1)
```

```
        .addGap(18, 18, 18)
```

```
        .addComponent(TransfersAeropuertoMadrid,  
javax.swing.GroupLayout.PREFERRED_SIZE, 99,  
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
        .addGap(29, 29, 29)
```

```
        .addComponent(jLabel2,  
javax.swing.GroupLayout.PREFERRED_SIZE, 96,  
javax.swing.GroupLayout.PREFERRED_SIZE)))
```

```
    .addGap(18, 18, 18)
```

```
    .addComponent(TransfersCiudadMadrid,  
javax.swing.GroupLayout.PREFERRED_SIZE, 100,  
javax.swing.GroupLayout.PREFERRED_SIZE)))
```

```
    .addGap(0, 85, Short.MAX_VALUE))
```

```
    .addGroup(layout.createSequentialGroup()
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
    .addComponent(jLabel8)
```

```
    .addComponent(jLabel9)
```

```

        .addComponent(jLabel10))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addGroup(layout.createSequentialGroup())

            .addComponent(txtGate1Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, 165,
javax.swing.GroupLayout.PREFERRED_SIZE)

                .addGap(0, 0, Short.MAX_VALUE))

            .addComponent(txtGate2Madrid,
javax.swing.GroupLayout.Alignment.TRAILING)

                .addComponent(txtGate3Madrid))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addComponent(jLabel12)

        .addComponent(jLabel13)

        .addComponent(jLabel11))

    .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)

        .addComponent(txtGate4Madrid,
javax.swing.GroupLayout.DEFAULT_SIZE, 125, Short.MAX_VALUE)

        .addComponent(txtGate5Madrid)

        .addComponent(txtGate6Madrid))))

.addGroup(layout.createSequentialGroup())

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)

        .addGroup(layout.createSequentialGroup())

            .addComponent(jLabel16)

            .addGap(18, 18, 18)

```

```

        .addComponent(txtPista2Madrid))

    .addGroup(layout.createSequentialGroup())

        .addComponent(jLabel15)

        .addGap(18, 18, 18)

        .addComponent(txtPista1Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, 155,
javax.swing.GroupLayout.PREFERRED_SIZE)))

    .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)

        .addGroup(layout.createSequentialGroup())

            .addComponent(jLabel18)

            .addGap(18, 18, 18)

            .addComponent(txtPista4Madrid))

        .addGroup(layout.createSequentialGroup())

            .addComponent(jLabel17)

            .addGap(18, 18, 18)

            .addComponent(txtPista3Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, 177,
javax.swing.GroupLayout.PREFERRED_SIZE))))))

        .addGap(11, 11, 11)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addGroup(layout.createSequentialGroup())

            .addGap(190, 190, 190)

            .addComponent(jLabel29))

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addGroup(layout.createSequentialGroup())

            .addComponent(jLabel27)

            .addGap(18, 18, 18)

```

```

        .addComponent(txtAreaRodajeBarcelona))

    .addGroup(layout.createSequentialGroup()

        .addComponent(jLabel34)

        .addGap(18, 18, 18)

        .addComponent(txtAreaEstacionBarcelona))

    .addGroup(layout.createSequentialGroup()

        .addComponent(jLabel32)

        .addGap(27, 27, 27)

        .addComponent(txtTallerBarcelona))

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)

        .addGroup(layout.createSequentialGroup()

            .addComponent(jLabel30)

            .addGap(18, 18, 18)

            .addComponent(txtHangarBarcelona))

        .addGroup(layout.createSequentialGroup()

            .addGroup(layout.createSequentialGroup()

                .addComponent(jLabel25)

                .addGap(18, 18, 18)

                .addComponent(NºPasajerosBarcelona))

            .addGroup(layout.createSequentialGroup()

                .addComponent(jLabel26)

                .addGap(18, 18, 18)

                .addComponent(TransfersAeropuertoBarcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, 99,
javax.swing.GroupLayout.PREFERRED_SIZE)

                .addGap(29, 29, 29)

                .addComponent(jLabel28,
javax.swing.GroupLayout.PREFERRED_SIZE, 96,
javax.swing.GroupLayout.PREFERRED_SIZE)))

```



```

        .addGap(18, 18, 18)

        .addComponent(TransfersCiudadBarcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE)))

    .addGroup(layout.createSequentialGroup())

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addComponent(jLabel36)

        .addComponent(jLabel38)

        .addComponent(jLabel21))

    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addComponent(txtGate2Barcelona,
javax.swing.GroupLayout.Alignment.TRAILING)

        .addComponent(txtGate3Barcelona)

        .addComponent(txtGate1Barcelona))

        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addComponent(jLabel23)

        .addComponent(jLabel24)

        .addComponent(jLabel22))

        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)

        .addComponent(txtGate4Barcelona,
javax.swing.GroupLayout.DEFAULT_SIZE, 110, Short.MAX_VALUE)

        .addComponent(txtGate5Barcelona)

        .addComponent(txtGate6Barcelona))))

    .addGroup(layout.createSequentialGroup())

    .addGap(106, 106, 106)

```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
```

```
        .addGroup(layout.createSequentialGroup())
            .addComponent(jLabel33)
            .addGap(18, 18, 18)
            .addComponent(txtPista2Barcelona))
        .addGroup(layout.createSequentialGroup())
            .addComponent(jLabel31)
            .addGap(18, 18, 18)
            .addComponent(txtPista1Barcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, 155,
javax.swing.GroupLayout.PREFERRED_SIZE)))
        .addGap(18, 18, 18)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
```

```
        .addGroup(layout.createSequentialGroup())
            .addComponent(jLabel37)
            .addGap(18, 18, 18)
            .addComponent(txtPista4Barcelona))
        .addGroup(layout.createSequentialGroup())
            .addComponent(jLabel35)
            .addGap(18, 18, 18)
            .addComponent(txtPista3Barcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, 160,
javax.swing.GroupLayout.PREFERRED_SIZE))))))
        .addGap(88, 88, 88)))
        .addGroup(layout.createSequentialGroup())
            .addGap(446, 446, 446)
            .addComponent(jLabel20)
            .addGap(0, 0, Short.MAX_VALUE))
        .addGroup(layout.createSequentialGroup())
```

```

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addGap(421, 421, 421)
        .addComponent(pausa)
        .addGap(62, 62, 62)
        .addComponent(Reanudar))
    .addGroup(layout.createSequentialGroup()
        .addGap(444, 444, 444)
        .addComponent(jLabel19)))
.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
);
layout.setVerticalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(pausa)
    .addComponent(Reanudar))
    .addGap(29, 29, 29)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
    .addGroup(layout.createSequentialGroup()
        .addComponent(jLabel29)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(jLabel26, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(TransfersAeropuertoBarcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

```

```
.addComponent(jLabel28, javax.swing.GroupLayout.PREFERRED_SIZE, 22,  
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
.addComponent(TransfersCiudadBarcelona,  
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,  
javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
.addGap(18, 18, 18)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```
.addComponent(jLabel25, javax.swing.GroupLayout.PREFERRED_SIZE, 22,  
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
.addComponent(NºPasajerosBarcelona,  
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,  
javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
.addGap(18, 18, 18)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```
.addComponent(jLabel30, javax.swing.GroupLayout.PREFERRED_SIZE, 22,  
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
.addComponent(txtHangarBarcelona,  
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,  
javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
.addGap(18, 18, 18)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```
.addComponent(jLabel32, javax.swing.GroupLayout.PREFERRED_SIZE, 22,  
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
.addComponent(txtTallerBarcelona,  
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,  
javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
.addGap(18, 18, 18)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```
.addComponent(jLabel34, javax.swing.GroupLayout.PREFERRED_SIZE, 22,  
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
.addComponent(txtAreaEstacionBarcelona,  
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,  
javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
.addGap(18, 18, 18)
```

```

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup())

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(jLabel36, javax.swing.GroupLayout.PREFERRED_SIZE,
22, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(txtGate1Barcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
    .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(jLabel38, javax.swing.GroupLayout.PREFERRED_SIZE,
22, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(txtGate2Barcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
    .addGap(18, 18, 18)
    .addComponent(jLabel21, javax.swing.GroupLayout.PREFERRED_SIZE,
22, javax.swing.GroupLayout.PREFERRED_SIZE))
    .addGroup(layout.createSequentialGroup())

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(jLabel23, javax.swing.GroupLayout.PREFERRED_SIZE,
22, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(txtGate4Barcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
    .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(jLabel24, javax.swing.GroupLayout.PREFERRED_SIZE,
22, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(txtGate5Barcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

```

```

        .addGap(18, 18, 18)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel22, javax.swing.GroupLayout.PREFERRED_SIZE,
                22, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(txtGate6Barcelona,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(txtGate3Barcelona,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE)))
        .addGap(18, 18, 18)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel27, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
                javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(txtAreaRodajeBarcelona,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(18, 18, 18)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup())

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel31, javax.swing.GroupLayout.PREFERRED_SIZE,
                22, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(txtPista1Barcelona,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(18, 18, 18)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel33, javax.swing.GroupLayout.PREFERRED_SIZE,
                22, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(txtPista2Barcelona,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE)))

```

```

        .addGroup(layout.createSequentialGroup())

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel35, javax.swing.GroupLayout.PREFERRED_SIZE,
                22, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(txtPista3Barcelona,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel37, javax.swing.GroupLayout.PREFERRED_SIZE,
                22, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(txtPista4Barcelona,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))))
        .addGroup(layout.createSequentialGroup())
        .addComponent(jLabel3)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
                javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(TransfersAeropuertoMadrid,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
                javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(TransfersCiudadMadrid,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
                javax.swing.GroupLayout.PREFERRED_SIZE)

```

```

        .addComponent(NºPasajerosMadrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel5, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE)

        .addComponent(txtHangarMadrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE)

        .addComponent(txtTallerMadrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE)

        .addComponent(txtAreaEstacionMadrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addGroup(layout.createSequentialGroup())

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel8, javax.swing.GroupLayout.PREFERRED_SIZE,
22, javax.swing.GroupLayout.PREFERRED_SIZE)

        .addComponent(txtGate1Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

```



```

        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel9, javax.swing.GroupLayout.PREFERRED_SIZE,
22, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(txtGate2Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(18, 18, 18)
        .addComponent(jLabel10, javax.swing.GroupLayout.PREFERRED_SIZE,
22, javax.swing.GroupLayout.PREFERRED_SIZE))
    .addGroup(layout.createSequentialGroup())

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel12, javax.swing.GroupLayout.PREFERRED_SIZE,
22, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(txtGate4Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel13, javax.swing.GroupLayout.PREFERRED_SIZE,
22, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(txtGate5Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel11, javax.swing.GroupLayout.PREFERRED_SIZE,
22, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(txtGate6Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(txtGate3Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))

```

```

        .addGap(18, 18, 18)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel14, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
                javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(txtAreaRodajeMadrid,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup())

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel15, javax.swing.GroupLayout.PREFERRED_SIZE,
                22, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(txtPista1Madrid,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel16, javax.swing.GroupLayout.PREFERRED_SIZE,
                22, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(txtPista2Madrid,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE)))
            .addGroup(layout.createSequentialGroup())

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel17, javax.swing.GroupLayout.PREFERRED_SIZE,
                22, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(txtPista3Madrid,
                javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)

```

```

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel18, javax.swing.GroupLayout.PREFERRED_SIZE,
22, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(txtPista4Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))))))
        .addGap(18, 18, 18)
        .addComponent(jLabel19, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(AeroviaMadrid_Barcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(jLabel20, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(AeroviaBarcelona_Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addContainerGap(30, Short.MAX_VALUE)
    );

```

```

jLabel1.getAccessibleContext().setAccessibleParent(jLabel1);
jLabel2.getAccessibleContext().setAccessibleParent(jLabel1);

```

```

pack();
} // </editor-fold>

```

```

private void TransfersCiudadMadridActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

```

```

private void pausaActionPerformed(java.awt.event.ActionEvent evt) {

```

```

    madrid.parar();

    barcelona.parar();
}

```

```

private void ReanudarActionPerformed(java.awt.event.ActionEvent evt) {

    madrid.continuar();

    barcelona.continuar();

}

```

```

private void TransfersCiudadBarcelonaActionPerformed(java.awt.event.ActionEvent
evt) {

    // TODO add your handling code here:

}

```

```

/**
 * @param args the command line arguments
 */

public static void main(String args[]) {

    /* Set the Nimbus look and feel */

    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and
feel.

    * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
    */

    try {

        for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {

            if ("Nimbus".equals(info.getName())) {

                javax.swing.UIManager.setLookAndFeel(info.getClassName());

                break;

            }

```

```

    }

    } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(InterfazServidor.class.getName()).log(java.util.logging.
Level.SEVERE, null, ex);

    } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(InterfazServidor.class.getName()).log(java.util.logging.
Level.SEVERE, null, ex);

    } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(InterfazServidor.class.getName()).log(java.util.logging.
Level.SEVERE, null, ex);

    } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(InterfazServidor.class.getName()).log(java.util.logging.
Level.SEVERE, null, ex);

    }
//</editor-fold>
//</editor-fold>

/* Create and display the form */
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new InterfazServidor().setVisible(true);
    }
});
}

// Variables declaration - do not modify
private javax.swing.JTextField AeroviaBarcelona_Madrid;
private javax.swing.JTextField AeroviaMadrid_Barcelona;
private javax.swing.JTextField N°PasajerosBarcelona;
private javax.swing.JTextField N°PasajerosMadrid;

```

```
private javax.swing.JButton Reanudar;
private javax.swing.JTextField TransfersAeropuertoBarcelona;
private javax.swing.JTextField TransfersAeropuertoMadrid;
private javax.swing.JTextField TransfersCiudadBarcelona;
private javax.swing.JTextField TransfersCiudadMadrid;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel11;
private javax.swing.JLabel jLabel12;
private javax.swing.JLabel jLabel13;
private javax.swing.JLabel jLabel14;
private javax.swing.JLabel jLabel15;
private javax.swing.JLabel jLabel16;
private javax.swing.JLabel jLabel17;
private javax.swing.JLabel jLabel18;
private javax.swing.JLabel jLabel19;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel20;
private javax.swing.JLabel jLabel21;
private javax.swing.JLabel jLabel22;
private javax.swing.JLabel jLabel23;
private javax.swing.JLabel jLabel24;
private javax.swing.JLabel jLabel25;
private javax.swing.JLabel jLabel26;
private javax.swing.JLabel jLabel27;
private javax.swing.JLabel jLabel28;
private javax.swing.JLabel jLabel29;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel30;
private javax.swing.JLabel jLabel31;
private javax.swing.JLabel jLabel32;
```

```
private javax.swing.JLabel jLabel33;
private javax.swing.JLabel jLabel34;
private javax.swing.JLabel jLabel35;
private javax.swing.JLabel jLabel36;
private javax.swing.JLabel jLabel37;
private javax.swing.JLabel jLabel38;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
private javax.swing.JButton pausa;
private javax.swing.JTextField txtAreaEstacionBarcelona;
private javax.swing.JTextField txtAreaEstacionMadrid;
private javax.swing.JTextField txtAreaRodajeBarcelona;
private javax.swing.JTextField txtAreaRodajeMadrid;
private javax.swing.JTextField txtGate1Barcelona;
private javax.swing.JTextField txtGate1Madrid;
private javax.swing.JTextField txtGate2Barcelona;
private javax.swing.JTextField txtGate2Madrid;
private javax.swing.JTextField txtGate3Barcelona;
private javax.swing.JTextField txtGate3Madrid;
private javax.swing.JTextField txtGate4Barcelona;
private javax.swing.JTextField txtGate4Madrid;
private javax.swing.JTextField txtGate5Barcelona;
private javax.swing.JTextField txtGate5Madrid;
private javax.swing.JTextField txtGate6Barcelona;
private javax.swing.JTextField txtGate6Madrid;
private javax.swing.JTextField txtHangarBarcelona;
private javax.swing.JTextField txtHangarMadrid;
```

```

private javax.swing.JTextField txtPista1Barcelona;
private javax.swing.JTextField txtPista1Madrid;
private javax.swing.JTextField txtPista2Barcelona;
private javax.swing.JTextField txtPista2Madrid;
private javax.swing.JTextField txtPista3Barcelona;
private javax.swing.JTextField txtPista3Madrid;
private javax.swing.JTextField txtPista4Barcelona;
private javax.swing.JTextField txtPista4Madrid;
private javax.swing.JTextField txtTallerBarcelona;
private javax.swing.JTextField txtTallerMadrid;
// End of variables declaration
}

package Concurrida;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Log {

    private static final String LOG_FILE_NAME = "evolucionAeropuerto.txt";

    private static final DateTimeFormatter DATE_TIME_FORMATTER =
DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");

    private BufferedWriter writer;

    public Log() {
        try {
            writer = new BufferedWriter(new FileWriter(LOG_FILE_NAME, true));

```



```

    } catch (IOException e) {
        System.err.println("Error al abrir el archivo de log: " + e.getMessage());
    }
}

```

```

public synchronized void writeLog(String event) {
    String timeStamp = LocalDateTime.now().format(DATE_TIME_FORMATTER);
    String logLine = timeStamp + " " + event;

    try {
        writer.write(logLine);
        writer.newLine();
        writer.flush();
    } catch (IOException e) {
        System.err.println("Error al escribir en el archivo de log: " + e.getMessage());
    }
}

```

```

public synchronized void close() {
    try {
        writer.close();
    } catch (IOException e) {
        System.err.println("Error al cerrar el archivo de log: " + e.getMessage());
    }
}
}

```

```

package Concurrida;

```

```

import java.util.Random;

```

```

public class Sistema extends Thread{

```

```
Aeropuerto madrid, barcelona;
```

```
Log log = new Log();
```

```
Sistema(Aeropuerto madrid, Aeropuerto barcelona) {
```

```
    this.madrid = madrid;
```

```
    this.barcelona = barcelona;
```

```
}
```

```
public void run(){
```

```
    for (int i = 0; i < 8000; i++) {
```

```
        if (i < 4000) {
```

```
            String prefijoID = "B-";
```

```
            long idAutobus = i + 1;
```

```
            if (idAutobus % 2 == 0) {
```

```
                Autobus autobus = new Autobus(prefijoID, idAutobus, madrid);
```

```
                autobus.start();
```

```
            } else {
```

```
                Autobus autobus = new Autobus(prefijoID, idAutobus, barcelona);
```

```
                autobus.start();
```

```
            }
```

```
        }
```

```
String candidateChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

```
StringBuilder sb = new StringBuilder();
```

```
Random random = new Random();
```

```
for (int n = 0; n < 2; n++) {
```

```
    sb.append(candidateChars.charAt(random.nextInt(candidateChars.length())));
```

```
}
```

```
String prefijoID = String.join("", sb) + "-";
```

```
long idAvion = i + 1;
```

```
int capacidad = (int) (Math.random() * 200 + 100);
```

```
if (idAvion % 2 == 0) {
```

```

        Avion avion = new Avion(madrid, barcelona, prefijoID, idAvion, capacidad, log);
        avion.start();
    } else {
        Avion avion = new Avion(barcelona, madrid, prefijoID, idAvion, capacidad, log);
        avion.start();
    }
    try {
        Thread.sleep((new Random().nextInt(3) * 1000) + 1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

}

package Distribuida;

import Concurrida.InterfazServidor;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.List;

public class AeropuertoImpl extends UnicastRemoteObject implements
AeropuertoRemote {

    private InterfazServidor gui;

    public AeropuertoImpl(InterfazServidor gui) throws RemoteException {
        super();
        this.gui = gui;
    }
}

```

@Override

```
public int getNumPasajerosMadrid() throws RemoteException {  
    return gui.getAeropuertoMadrid().getPersonasDentro();  
}
```

@Override

```
public int getNumPasajerosBarcelona() throws RemoteException {  
    return gui.getAeropuertoBarcelona().getPersonasDentro();  
  
}
```

@Override

```
public int getNumAvionesHangarMadrid() throws RemoteException {  
    return gui.getAeropuertoMadrid().getAvionesHangar();  
}
```

@Override

```
public int getNumAvionesHangarBarcelona() throws RemoteException {  
    return gui.getAeropuertoBarcelona().getAvionesHangar();  
}
```

@Override

```
public int getNumAvionesTallerMadrid() throws RemoteException {  
    return gui.getAeropuertoMadrid().getAvionesTaller();  
}
```

@Override

```
public int getNumAvionesTallerBarcelona() throws RemoteException {  
    return gui.getAeropuertoBarcelona().getAvionesTaller();  
}
```

@Override

```
public int getNumAvionesAreaEstacionamientoMadrid() throws RemoteException {  
    return gui.getAeropuertoMadrid().getAvionesAreaEstacionamiento();  
}
```

@Override

```
public int getNumAvionesAreaEstacionamientoBarcelona() throws RemoteException {  
    return gui.getAeropuertoBarcelona().getAvionesAreaEstacionamiento();  
}
```

@Override

```
public int getNumAvionesAreaRodajeMadrid() throws RemoteException {  
    return gui.getAeropuertoMadrid().getAvionesAreaRodaje();  
}
```

@Override

```
public int getNumAvionesAreaRodajeBarcelona() throws RemoteException {  
    return gui.getAeropuertoBarcelona().getAvionesAreaRodaje();  
}
```

@Override

```
public List<String> getAeroviaMadrid() throws RemoteException {  
    return gui.getAeropuertoMadrid().getAerovia();  
}
```

@Override

```
public List<String> getAeroviaBarcelona() throws RemoteException {  
    return gui.getAeropuertoBarcelona().getAerovia();  
}
```

////////

@Override

```
public void cerrarPista1Madrid() throws RemoteException {  
    gui.getAeropuertoMadrid().setPista1Cerrada(true);  
}
```

@Override

```
public void cerrarPista2Madrid() throws RemoteException {  
    gui.getAeropuertoMadrid().setPista2Cerrada(true);  
}
```

@Override

```
public void cerrarPista3Madrid() throws RemoteException {  
    gui.getAeropuertoMadrid().setPista3Cerrada(true);  
}
```

@Override

```
public void cerrarPista4Madrid() throws RemoteException {  
    gui.getAeropuertoMadrid().setPista4Cerrada(true);  
}
```

@Override

```
public void cerrarPista1Barcelona() throws RemoteException {  
    gui.getAeropuertoBarcelona().setPista1Cerrada(true);  
}
```

@Override

```
public void cerrarPista2Barcelona() throws RemoteException {  
    gui.getAeropuertoBarcelona().setPista2Cerrada(true);  
}
```

@Override

```
public void cerrarPista3Barcelona() throws RemoteException {  
    gui.getAeropuertoBarcelona().setPista3Cerrada(true);  
}
```

@Override

```
public void cerrarPista4Barcelona() throws RemoteException {  
    gui.getAeropuertoBarcelona().setPista4Cerrada(true);  
}
```

////////

@Override

```
public void liberarPista1Madrid() throws RemoteException {  
    gui.getAeropuertoMadrid().setPista1Cerrada(false);  
}
```

@Override

```
public void liberarPista2Madrid() throws RemoteException {  
    gui.getAeropuertoMadrid().setPista2Cerrada(false);  
}
```

@Override

```
public void liberarPista3Madrid() throws RemoteException {  
    gui.getAeropuertoMadrid().setPista3Cerrada(false);  
}
```

@Override

```
public void liberarPista4Madrid() throws RemoteException {  
    gui.getAeropuertoMadrid().setPista4Cerrada(false);  
}
```

@Override

```
public void liberarPista1Barcelona() throws RemoteException {  
    gui.getAeropuertoBarcelona().setPista1Cerrada(false);  
}
```

@Override

```
public void liberarPista2Barcelona() throws RemoteException {  
    gui.getAeropuertoBarcelona().setPista2Cerrada(false);  
}
```

@Override

```
public void liberarPista3Barcelona() throws RemoteException {  
    gui.getAeropuertoBarcelona().setPista3Cerrada(false);  
}
```

@Override

```
public void liberarPista4Barcelona() throws RemoteException {  
    gui.getAeropuertoBarcelona().setPista4Cerrada(false);  
}  
}
```

package Distribuida;

import java.rmi.Remote;

import java.rmi.RemoteException;

import java.util.List;

public interface AeropuertoRemote extends Remote {

public int getNumPasajerosMadrid() throws RemoteException;

public int getNumPasajerosBarcelona() throws RemoteException;

public int getNumAvionesHangarMadrid() throws RemoteException;

public int getNumAvionesHangarBarcelona() throws RemoteException;


```

public int getNumAvionesTallerMadrid() throws RemoteException;
public int getNumAvionesTallerBarcelona() throws RemoteException;
public int getNumAvionesAreaEstacionamientoMadrid() throws RemoteException;
public int getNumAvionesAreaEstacionamientoBarcelona() throws RemoteException;
public int getNumAvionesAreaRodajeMadrid() throws RemoteException;
public int getNumAvionesAreaRodajeBarcelona() throws RemoteException;
public List<String> getAeroviaMadrid() throws RemoteException;
public List<String> getAeroviaBarcelona() throws RemoteException;
public void cerrarPista1Madrid() throws RemoteException;
public void cerrarPista2Madrid() throws RemoteException;
public void cerrarPista3Madrid() throws RemoteException;
public void cerrarPista4Madrid() throws RemoteException;
public void cerrarPista1Barcelona() throws RemoteException;
public void cerrarPista2Barcelona() throws RemoteException;
public void cerrarPista3Barcelona() throws RemoteException;
public void cerrarPista4Barcelona() throws RemoteException;
public void liberarPista1Madrid() throws RemoteException;
public void liberarPista2Madrid() throws RemoteException;
public void liberarPista3Madrid() throws RemoteException;
public void liberarPista4Madrid() throws RemoteException;
public void liberarPista1Barcelona() throws RemoteException;
public void liberarPista2Barcelona() throws RemoteException;
public void liberarPista3Barcelona() throws RemoteException;
public void liberarPista4Barcelona() throws RemoteException;

```

```

}

```

```

package Distribuida;

```

```

import java.net.MalformedURLException;

```

```

import java.rmi.Naming;

```

```

import java.rmi.NotBoundException;

```

```

import java.rmi.RemoteException;

```

```

public class Cliente extends Thread {

    private final InterfazCliente gui;

    private AeropuertoRemote obj;

    private int vecesPista1Madrid, vecesPista2Madrid, vecesPista3Madrid,
    vecesPista4Madrid, vecesPista1Barcelona,

        vecesPista2Barcelona, vecesPista3Barcelona, vecesPista4Barcelona = 0;

    private int vLiberaPista1Madrid, vLiberaPista2Madrid, vLiberaPista3Madrid,
    vLiberaPista4Madrid, vLiberaPista1Barcelona,

        vLiberaPista2Barcelona, vLiberaPista3Barcelona, vLiberaPista4Barcelona = 0;

    public Cliente(InterfazCliente gui) {

        this.gui = gui;
    }

    @Override

    public void run() {

        try {

            obj = (AeropuertoRemote) Naming.lookup("//127.0.0.1/ObjetoAeropuerto");
            //Localiza el objeto distribuido

            System.out.println("Cliente conectado");

            while (true) {

                gui.actualizar(obj.getNumPasajerosMadrid(), obj.getNumPasajerosBarcelona(),
                obj.getNumAvionesHangarMadrid(),

                    obj.getNumAvionesHangarBarcelona(), obj.getNumAvionesTallerMadrid(),
                obj.getNumAvionesTallerBarcelona(),

                    obj.getNumAvionesAreaEstacionamientoMadrid(),
                obj.getNumAvionesAreaEstacionamientoBarcelona(),

                    obj.getNumAvionesAreaRodajeMadrid(),
                obj.getNumAvionesAreaRodajeBarcelona(), obj.getAeroviaMadrid(),

                    obj.getAeroviaBarcelona());
            }
        }
    }
}

```

```
//BLOQUEAR LAS PISTAS
```

```
if (gui.isBotonCerrarPista1Madrid()) {
```

```
    vLiberaPista1Madrid = 0;
```

```
    vecesPista1Madrid++;
```

```
    if (vecesPista1Madrid == 1) {
```

```
        obj.cerrarPista1Madrid();
```

```
    }
```

```
}
```

```
if (gui.isBotonCerrarPista2Madrid()) {
```

```
    vLiberaPista2Madrid = 0;
```

```
    vecesPista2Madrid++;
```

```
    if (vecesPista2Madrid == 1) {
```

```
        obj.cerrarPista2Madrid();
```

```
    }
```

```
}
```

```
if (gui.isBotonCerrarPista3Madrid()) {
```

```
    vLiberaPista3Madrid = 0;
```

```
    vecesPista3Madrid++;
```

```
    if (vecesPista3Madrid == 1) {
```

```
        obj.cerrarPista3Madrid();
```

```
    }
```

```
}
```

```
if (gui.isBotonCerrarPista4Madrid()) {
```

```
    vLiberaPista4Madrid = 0;
```

```
    vecesPista4Madrid++;
```

```
    if (vecesPista4Madrid == 1) {
```

```
        obj.cerrarPista4Madrid();
```

```
    }
```

```
}
```

```
if (gui.isBotonCerrarPista1Barcelona()) {  
    vLiberaPista1Barcelona = 0;  
    vecesPista1Barcelona++;  
    if (vecesPista1Barcelona == 1) {  
        obj.cerrarPista1Barcelona();  
    }  
}  
  
if (gui.isBotonCerrarPista2Barcelona()) {  
    vLiberaPista2Barcelona = 0;  
    vecesPista2Barcelona++;  
    if (vecesPista2Barcelona == 1) {  
        obj.cerrarPista2Barcelona();  
    }  
}  
  
if (gui.isBotonCerrarPista3Barcelona()) {  
    vLiberaPista3Barcelona = 0;  
    vecesPista3Barcelona++;  
    if (vecesPista3Barcelona == 1) {  
        obj.cerrarPista3Barcelona();  
    }  
}  
  
if (gui.isBotonCerrarPista4Barcelona()) {  
    vLiberaPista4Barcelona = 0;  
    vecesPista4Barcelona++;  
    if (vecesPista4Barcelona == 1) {  
        obj.cerrarPista4Barcelona();  
    }  
}  
  
//LIBERAR LAS PISTAS  
  
if (gui.isBotonLiberarPista1Madrid()) {
```

```
        vecesPista1Madrid = 0;

        vLiberaPista1Madrid++;

        if (vLiberaPista1Madrid == 1) {
            obj.liberarPista1Madrid();
        }
    }

    if (gui.isBotonLiberarPista2Madrid()) {

        vecesPista2Madrid = 0;

        vLiberaPista2Madrid++;

        if (vLiberaPista2Madrid == 1) {
            obj.liberarPista2Madrid();
        }
    }

    if (gui.isBotonLiberarPista3Madrid()) {

        vecesPista3Madrid = 0;

        vLiberaPista3Madrid++;

        if (vLiberaPista3Madrid == 1) {
            obj.liberarPista3Madrid();
        }
    }

    if (gui.isBotonLiberarPista4Madrid()) {

        vecesPista4Madrid = 0;

        vLiberaPista4Madrid++;

        if (vLiberaPista4Madrid == 1) {
            obj.liberarPista4Madrid();
        }
    }

    if (gui.isBotonLiberarPista1Barcelona()) {

        vecesPista1Barcelona = 0;
```

```

        vLiberaPista1Barcelona++;

        if (vLiberaPista1Barcelona == 1) {
            obj.liberarPista1Barcelona();
        }
    }

    if (gui.isBotonLiberarPista2Barcelona()) {
        vecesPista2Barcelona = 0;
        vLiberaPista2Barcelona++;
        if (vLiberaPista2Barcelona == 1) {
            obj.liberarPista2Barcelona();
        }
    }

    if (gui.isBotonLiberarPista3Barcelona()) {
        vecesPista3Barcelona = 0;
        vLiberaPista3Barcelona++;
        if (vLiberaPista3Barcelona == 1) {
            obj.liberarPista3Barcelona();
        }
    }

    if (gui.isBotonLiberarPista4Barcelona()) {
        vecesPista4Barcelona = 0;
        vLiberaPista4Barcelona++;
        if (vLiberaPista4Barcelona == 1) {
            obj.liberarPista4Barcelona();
        }
    }

}

} catch (MalformedURLException | NotBoundException | RemoteException e) {
    System.out.println("Excepción : " + e.getMessage());
}

```

```

    }

}

}

package Distribuida;

import java.io.IOException;

import java.util.List;

/**
 *
 * @author christian
 */
public class InterfazCliente extends javax.swing.JFrame {

    private final Cliente cliente;

    private boolean botonCerrarPista1Madrid, botonCerrarPista2Madrid,
    botonCerrarPista3Madrid, botonCerrarPista4Madrid,

        botonCerrarPista1Barcelona, botonCerrarPista2Barcelona,
    botonCerrarPista3Barcelona, botonCerrarPista4Barcelona = false;

    private boolean botonLiberarPista1Madrid, botonLiberarPista2Madrid,
    botonLiberarPista3Madrid, botonLiberarPista4Madrid,

        botonLiberarPista1Barcelona, botonLiberarPista2Barcelona,
    botonLiberarPista3Barcelona, botonLiberarPista4Barcelona = false;

    public InterfazCliente() throws IOException {

        initComponents();

        BotonAbrirPista1Madrid.setEnabled(false);

        BotonAbrirPista2Madrid.setEnabled(false);

```

```
BotonAbrirPista3Madrid.setEnabled(false);  
BotonAbrirPista4Madrid.setEnabled(false);  
BotonAbrirPista1Barcelona.setEnabled(false);  
BotonAbrirPista2Barcelona.setEnabled(false);  
BotonAbrirPista3Barcelona.setEnabled(false);  
BotonAbrirPista4Barcelona.setEnabled(false);
```

```
cliente = new Cliente(this);  
Thread t = new Thread(cliente);  
t.start();  
}
```

```
public void actualizar(int numPasajerosMadrid, int numPasajerosBarcelona, int  
numAvionesHangarMadrid, int numAvionesHangarBarcelona, int  
numAvionesTallerMadrid, int numAvionesTallerBarcelona, int  
numAvionesAreaEstacionamientoMadrid,  
int numAvionesAreaEstacionamientoBarcelona, int numAvionesAreaRodajeMadrid,  
int numAvionesAreaRodajeBarcelona, List<String> aeroviaMadrid, List<String>  
aeroviaBarcelona) {
```

```
TxtPasajerosMadrid.setText(Integer.toString(numPasajerosMadrid));  
TxtPasajerosBarcelona.setText(Integer.toString(numPasajerosBarcelona));
```

```
TxtAvionesHangarMadrid.setText(Integer.toString(numAvionesHangarMadrid));  
TxtAvionesHangarBarcelona.setText(Integer.toString(numAvionesHangarBarcelona));
```

```
TxtAvionesTallarMadrid.setText(Integer.toString(numAvionesTallerMadrid));  
TxtAvionesTallarBarcelona.setText(Integer.toString(numAvionesTallerBarcelona));
```

```
TxtAvionesAreaEstacionamientoMadrid.setText(Integer.toString(numAvionesAreaEstacio  
namientoMadrid));
```



```
TxtAvionesAreaEstacionamientoBarcelona.setText(Integer.toString(numAvionesAreaEstacionamientoBarcelona));
```

```
TxtAvionesAreaRodajeMadrid.setText(String.format("%s", numAvionesAreaRodajeMadrid));
```

```
TxtAvionesAreaRodajeBarcelona.setText(String.format("%s", numAvionesAreaRodajeBarcelona));
```

```
TxtAeroviaMadrid.setText(String.join(" ", aeroviaMadrid));
```

```
TxtAeroviaBarcelona.setText(String.join(" ", aeroviaBarcelona));
```

```
}
```

```
@SuppressWarnings("unchecked")
```

```
// <editor-fold defaultstate="collapsed" desc="Generated Code">
```

```
private void initComponents() {
```

```
jLabel18 = new javax.swing.JLabel();
```

```
BotonCerrarPista1Barcelona = new javax.swing.JButton();
```

```
BotonCerrarPista2Barcelona = new javax.swing.JButton();
```

```
BotonCerrarPista3Barcelona = new javax.swing.JButton();
```

```
jLabel2 = new javax.swing.JLabel();
```

```
BotonCerrarPista4Barcelona = new javax.swing.JButton();
```

```
jLabel3 = new javax.swing.JLabel();
```

```
jLabel19 = new javax.swing.JLabel();
```

```
jLabel4 = new javax.swing.JLabel();
```

```
jLabel20 = new javax.swing.JLabel();
```

```
jLabel5 = new javax.swing.JLabel();
```

```
jLabel21 = new javax.swing.JLabel();
```

```
jLabel6 = new javax.swing.JLabel();
```

```
jLabel22 = new javax.swing.JLabel();
```

```
jLabel7 = new javax.swing.JLabel();
jLabel8 = new javax.swing.JLabel();
jLabel9 = new javax.swing.JLabel();
jLabel10 = new javax.swing.JLabel();
jLabel11 = new javax.swing.JLabel();
BotonCerrarPista1Madrid = new javax.swing.JButton();
BotonCerrarPista2Madrid = new javax.swing.JButton();
BotonCerrarPista3Madrid = new javax.swing.JButton();
BotonCerrarPista4Madrid = new javax.swing.JButton();
jLabel12 = new javax.swing.JLabel();
jLabel13 = new javax.swing.JLabel();
TxtAeroviaMadrid = new javax.swing.JTextField();
TxtAeroviaBarcelona = new javax.swing.JTextField();
BotonAbrirPista1Madrid = new javax.swing.JButton();
BotonAbrirPista2Madrid = new javax.swing.JButton();
BotonAbrirPista3Madrid = new javax.swing.JButton();
BotonAbrirPista4Madrid = new javax.swing.JButton();
TxtPasajerosMadrid = new javax.swing.JTextField();
TxtAvionesTallarMadrid = new javax.swing.JTextField();
TxtAvionesAreaEstacionamientoMadrid = new javax.swing.JTextField();
TxtAvionesAreaRodajeMadrid = new javax.swing.JTextField();
TxtAvionesHangarMadrid = new javax.swing.JTextField();
BotonAbrirPista1Barcelona = new javax.swing.JButton();
BotonAbrirPista2Barcelona = new javax.swing.JButton();
BotonAbrirPista3Barcelona = new javax.swing.JButton();
BotonAbrirPista4Barcelona = new javax.swing.JButton();
TxtPasajerosBarcelona = new javax.swing.JTextField();
TxtAvionesTallarBarcelona = new javax.swing.JTextField();
TxtAvionesAreaEstacionamientoBarcelona = new javax.swing.JTextField();
TxtAvionesAreaRodajeBarcelona = new javax.swing.JTextField();
TxtAvionesHangarBarcelona = new javax.swing.JTextField();
```

```
jLabel14 = new javax.swing.JLabel();
jLabel15 = new javax.swing.JLabel();
jLabel16 = new javax.swing.JLabel();
jLabel17 = new javax.swing.JLabel();
jLabel1 = new javax.swing.JLabel();
```

```
setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
```

```
jLabel18.setText("Pista 4");
```

```
BotonCerrarPista1Barcelona.setText("Cerrar");
```

```
BotonCerrarPista1Barcelona.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        BotonCerrarPista1BarcelonaActionPerformed(evt);
    }
});
```

```
BotonCerrarPista2Barcelona.setText("Cerrar");
```

```
BotonCerrarPista2Barcelona.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        BotonCerrarPista2BarcelonaActionPerformed(evt);
    }
});
```

```
BotonCerrarPista3Barcelona.setText("Cerrar");
```

```
BotonCerrarPista3Barcelona.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        BotonCerrarPista3BarcelonaActionPerformed(evt);
    }
});
```

```
jLabel2.setText("Aeropuerto Barcelona");
```

```
BotonCerrarPista4Barcelona.setText("Cerrar");
```

```
BotonCerrarPista4Barcelona.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        BotonCerrarPista4BarcelonaActionPerformed(evt);  
    }  
});
```

```
jLabel3.setText("Nº de Pasajeros en Aeropuerto:");
```

```
jLabel19.setText("Nº de Pasajeros en Aeropuerto:");
```

```
jLabel4.setText("Nº Aviones en Hangar:");
```

```
jLabel20.setText("Nº Aviones en Hangar:");
```

```
jLabel5.setText("Nº Aviones en Taller:");
```

```
jLabel21.setText("Nº Aviones en Taller:");
```

```
jLabel6.setText("Nº Aviones en Área Estacionamiento:");
```

```
jLabel22.setText("Nº Aviones en Área Estacionamiento:");
```

```
jLabel7.setText("Nº Aviones en Área Rodaje:");
```

```
jLabel8.setText("Pista 3");
```

```
jLabel9.setText("Pista 2");
```

```
jLabel10.setText("Pista 1");
```

```
jLabel11.setText("Pista 4");
```

```
BotonCerrarPista1Madrid.setText("Cerrar");
```

```
BotonCerrarPista1Madrid.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        BotonCerrarPista1MadridActionPerformed(evt);  
    }  
});
```

```
BotonCerrarPista2Madrid.setText("Cerrar");
```

```
BotonCerrarPista2Madrid.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        BotonCerrarPista2MadridActionPerformed(evt);  
    }  
});
```

```
BotonCerrarPista3Madrid.setText("Cerrar");
```

```
BotonCerrarPista3Madrid.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        BotonCerrarPista3MadridActionPerformed(evt);  
    }  
});
```

```
BotonCerrarPista4Madrid.setText("Cerrar");
```

```
BotonCerrarPista4Madrid.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        BotonCerrarPista4MadridActionPerformed(evt);  
    }  
});
```

```
jLabel12.setText("Aerovía Madrid-Barcelona");
```

```
jLabel13.setText("Aerovía Barcelona-Madrid");
```

```
BotonAbrirPista1Madrid.setText("Abrir");
```

```
BotonAbrirPista1Madrid.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        BotonAbrirPista1MadridActionPerformed(evt);  
    }  
});
```

```
BotonAbrirPista2Madrid.setText("Abrir");
```

```
BotonAbrirPista2Madrid.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        BotonAbrirPista2MadridActionPerformed(evt);  
    }  
});
```

```
BotonAbrirPista3Madrid.setText("Abrir");
```

```
BotonAbrirPista3Madrid.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        BotonAbrirPista3MadridActionPerformed(evt);  
    }  
});
```

```
BotonAbrirPista4Madrid.setText("Abrir");
```

```
BotonAbrirPista4Madrid.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        BotonAbrirPista4MadridActionPerformed(evt);  
    }  
}
```

```
});
```

```
BotonAbrirPista1Barcelona.setText("Abrir");
```

```
BotonAbrirPista1Barcelona.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        BotonAbrirPista1BarcelonaActionPerformed(evt);  
    }  
});
```

```
BotonAbrirPista2Barcelona.setText("Abrir");
```

```
BotonAbrirPista2Barcelona.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        BotonAbrirPista2BarcelonaActionPerformed(evt);  
    }  
});
```

```
BotonAbrirPista3Barcelona.setText("Abrir");
```

```
BotonAbrirPista3Barcelona.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        BotonAbrirPista3BarcelonaActionPerformed(evt);  
    }  
});
```

```
BotonAbrirPista4Barcelona.setText("Abrir");
```

```
BotonAbrirPista4Barcelona.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent evt) {  
        BotonAbrirPista4BarcelonaActionPerformed(evt);  
    }  
});
```

```
jLabel14.setText("Nº Aviones en Área Rodaje:");
```

```
jLabel15.setText("Pista 3");
```

```
jLabel16.setText("Pista 2");
```

```
jLabel17.setText("Pista 1");
```

```
jLabel1.setText("Aeropuerto Madrid");
```

```
javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(188, 188, 188)
            .addComponent(jLabel1)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
                javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(jLabel2)
            .addGap(157, 157, 157))
        .addGroup(layout.createSequentialGroup()
            .addGap(31, 31, 31)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGap(9, 9, 9)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
                        false)
                    )
                )
            )
        )
    );
```



```

        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup())

        .addComponent(jLabel3)

        .addGap(18, 18, 18)

        .addComponent(TxtPasajerosMadrid))

        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup())

        .addComponent(jLabel4)

        .addGap(18, 18, 18)

        .addComponent(TxtAvionesHangarMadrid))

        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup())

        .addComponent(jLabel5)

        .addGap(18, 18, 18)

        .addComponent(TxtAvionesTallarMadrid))

        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup())

        .addComponent(jLabel6)

        .addGap(18, 18, 18)

        .addComponent(TxtAvionesAreaEstacionamientoMadrid))

        .addGroup(layout.createSequentialGroup())

        .addComponent(jLabel7)

        .addGap(18, 18, 18)

        .addComponent(TxtAvionesAreaRodajeMadrid,
javax.swing.GroupLayout.PREFERRED_SIZE, 241,
javax.swing.GroupLayout.PREFERRED_SIZE)))

        .addGroup(layout.createSequentialGroup())

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
false)

            .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup())

                .addComponent(jLabel10)

            )

        )

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

```

```

        .addComponent(BotonCerrarPista1Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, 1, Short.MAX_VALUE))

        .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup())

        .addComponent(jLabel9)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

        .addComponent(BotonCerrarPista2Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, 64,
javax.swing.GroupLayout.PREFERRED_SIZE)))

        .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)

        .addComponent(BotonAbrirPista1Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, 64,
javax.swing.GroupLayout.PREFERRED_SIZE)

        .addComponent(BotonAbrirPista2Madrid,
javax.swing.GroupLayout.PREFERRED_SIZE, 1, Short.MAX_VALUE))

        .addGap(26, 26, 26)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addGroup(layout.createSequentialGroup()

            .addComponent(jLabel8)

            .addGap(18, 18, 18)

            .addComponent(BotonCerrarPista3Madrid)

            .addGap(18, 18, 18)

            .addComponent(BotonAbrirPista3Madrid))

        .addGroup(layout.createSequentialGroup()

            .addComponent(jLabel11)

            .addGap(18, 18, 18)

            .addComponent(BotonCerrarPista4Madrid)

            .addGap(18, 18, 18)

            .addComponent(BotonAbrirPista4Madrid))))))

```

```
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,  
24, Short.MAX_VALUE)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
.addGroup(layout.createSequentialGroup())
```

```
.addGap(9, 9, 9)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,  
false)
```

```
.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,  
layout.createSequentialGroup())
```

```
.addComponent(jLabel19)
```

```
.addGap(18, 18, 18)
```

```
.addComponent(TxtPasajerosBarcelona))
```

```
.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,  
layout.createSequentialGroup())
```

```
.addComponent(jLabel20)
```

```
.addGap(18, 18, 18)
```

```
.addComponent(TxtAvionesHangarBarcelona))
```

```
.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,  
layout.createSequentialGroup())
```

```
.addComponent(jLabel21)
```

```
.addGap(18, 18, 18)
```

```
.addComponent(TxtAvionesTallarBarcelona))
```

```
.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,  
layout.createSequentialGroup())
```

```
.addComponent(jLabel22)
```

```
.addGap(18, 18, 18)
```

```
.addComponent(TxtAvionesAreaEstacionamientoBarcelona))
```

```
.addGroup(layout.createSequentialGroup())
```

```
.addComponent(jLabel14)
```

```
.addGap(18, 18, 18)
```

```
.addComponent(TxtAvionesAreaRodajeBarcelona,  
javax.swing.GroupLayout.PREFERRED_SIZE, 241,  
javax.swing.GroupLayout.PREFERRED_SIZE)))
```

```

        .addGroup(layout.createSequentialGroup())

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
false)

        .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()

            .addComponent(jLabel17)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

            .addComponent(BotonCerrarPista1Barcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, 1, Short.MAX_VALUE))

        .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()

            .addComponent(jLabel16)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

            .addComponent(BotonCerrarPista2Barcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, 64,
javax.swing.GroupLayout.PREFERRED_SIZE)))

        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)

        .addComponent(BotonAbrirPista1Barcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, 64,
javax.swing.GroupLayout.PREFERRED_SIZE)

        .addComponent(BotonAbrirPista2Barcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, 64,
javax.swing.GroupLayout.PREFERRED_SIZE))

        .addGap(26, 26, 26)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addGroup(layout.createSequentialGroup()

            .addComponent(jLabel15)

            .addGap(18, 18, 18)

            .addComponent(BotonCerrarPista3Barcelona)

            .addGap(18, 18, 18)

```

```

        .addComponent(BotonAbrirPista3Barcelona))
    .addGroup(layout.createSequentialGroup()
        .addComponent(jLabel18)
        .addGap(18, 18, 18)
        .addComponent(BotonCerrarPista4Barcelona)
        .addGap(18, 18, 18)
        .addComponent(BotonAbrirPista4Barcelona))))))
    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
        .addComponent(TxtAeroviaMadrid,
javax.swing.GroupLayout.DEFAULT_SIZE, 888, Short.MAX_VALUE)
        .addComponent(TxtAeroviaBarcelona))
        .addGap(0, 0, Short.MAX_VALUE)))
    .addContainerGap())
    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(405, 405, 405)
            .addComponent(jLabel13))
        .addGroup(layout.createSequentialGroup()
            .addGap(407, 407, 407)
            .addComponent(jLabel12)))
        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(37, 37, 37)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)

```

```
.addGroup(layout.createSequentialGroup())

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

    .addComponent(jLabel1)

    .addComponent(jLabel2))

.addGap(34, 34, 34)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

    .addComponent(jLabel3)

    .addComponent(TxtPasajerosMadrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

    .addComponent(jLabel4)

    .addComponent(TxtAvionesHangarMadrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

    .addComponent(jLabel5)

    .addComponent(TxtAvionesTallarMadrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

    .addComponent(jLabel6)

    .addComponent(TxtAvionesAreaEstacionamientoMadrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```

        .addComponent(jLabel7)

        .addComponent(TxtAvionesAreaRodajeMadrid,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

        .addGap(82, 82, 82)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(BotonCerrarPista3Madrid)
        .addComponent(BotonAbrirPista3Madrid))
        .addComponent(jLabel8)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(BotonAbrirPista1Madrid)
        .addComponent(BotonCerrarPista1Madrid)
        .addComponent(jLabel10)))
        .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
        .addComponent(jLabel9)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(BotonCerrarPista2Madrid)
        .addComponent(BotonAbrirPista2Madrid)
        .addComponent(jLabel11)
        .addComponent(BotonCerrarPista4Madrid)
        .addComponent(BotonAbrirPista4Madrid))))
        .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel19)

        .addComponent(TxtPasajerosBarcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

```

```

        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel20)

        .addComponent(TxtAvionesHangarBarcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel21)

        .addComponent(TxtAvionesTallarBarcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel22)

        .addComponent(TxtAvionesAreaEstacionamientoBarcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

        .addGap(18, 18, 18)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel14)

        .addComponent(TxtAvionesAreaRodajeBarcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

        .addGap(82, 82, 82)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(BotonCerrarPista3Barcelona)

        .addComponent(BotonAbrirPista3Barcelona))

        .addComponent(jLabel15)

```



```

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(BotonAbrirPista1Barcelona)
    .addComponent(BotonCerrarPista1Barcelona)
    .addComponent(jLabel17)))
.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
    .addComponent(jLabel16)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(BotonCerrarPista2Barcelona)
    .addComponent(BotonAbrirPista2Barcelona)
    .addComponent(jLabel18)
    .addComponent(BotonCerrarPista4Barcelona)
    .addComponent(BotonAbrirPista4Barcelona))))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
.addComponent(jLabel12)
.addGap(18, 18, 18)

.addComponent(TxtAeroviaMadrid, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
.addGap(24, 24, 24)
.addComponent(jLabel13)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(TxtAeroviaBarcelona,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addGap(27, 27, 27))

);

pack();
} // </editor-fold>

```

```
private void BotonCerrarPista1MadridActionPerformed(java.awt.event.ActionEvent evt)
{
    botonCerrarPista1Madrid = true;
    botonLiberarPista1Madrid = false;
    BotonAbrirPista1Madrid.setEnabled(true);
    BotonCerrarPista1Madrid.setEnabled(false);
}
```

```
private void BotonCerrarPista2MadridActionPerformed(java.awt.event.ActionEvent evt)
{
    botonCerrarPista2Madrid = true;
    botonLiberarPista2Madrid = false;
    BotonAbrirPista2Madrid.setEnabled(true);
    BotonCerrarPista2Madrid.setEnabled(false);
}
```

```
private void BotonCerrarPista3MadridActionPerformed(java.awt.event.ActionEvent evt)
{
    botonCerrarPista3Madrid = true;
    botonLiberarPista3Madrid = false;
    BotonAbrirPista3Madrid.setEnabled(true);
    BotonCerrarPista3Madrid.setEnabled(false);
}
```

```
private void BotonCerrarPista4MadridActionPerformed(java.awt.event.ActionEvent evt)
{
    botonCerrarPista4Madrid = true;
    botonLiberarPista4Madrid = false;
    BotonAbrirPista4Madrid.setEnabled(true);
    BotonCerrarPista4Madrid.setEnabled(false);
}
```

```
private void BotonCerrarPista1BarcelonaActionPerformed(java.awt.event.ActionEvent
evt) {
    botonCerrarPista1Barcelona = true;
    botonLiberarPista1Barcelona = false;
    BotonAbrirPista1Barcelona.setEnabled(true);
    BotonCerrarPista1Barcelona.setEnabled(false);
}
```

```
private void BotonCerrarPista2BarcelonaActionPerformed(java.awt.event.ActionEvent
evt) {
    botonCerrarPista2Barcelona = true;
    botonLiberarPista2Barcelona = false;
    BotonAbrirPista2Barcelona.setEnabled(true);
    BotonCerrarPista2Barcelona.setEnabled(false);
}
```

```
private void BotonCerrarPista3BarcelonaActionPerformed(java.awt.event.ActionEvent
evt) {
    botonCerrarPista3Barcelona = true;
    botonLiberarPista3Barcelona = false;
    BotonAbrirPista3Barcelona.setEnabled(true);
    BotonCerrarPista3Barcelona.setEnabled(false);
}
```

```
private void BotonCerrarPista4BarcelonaActionPerformed(java.awt.event.ActionEvent
evt) {
    botonCerrarPista4Barcelona = true;
    botonLiberarPista4Barcelona = false;
    BotonAbrirPista4Barcelona.setEnabled(true);
    BotonCerrarPista4Barcelona.setEnabled(false);
}
```

```
private void BotonAbrirPista1MadridActionPerformed(java.awt.event.ActionEvent evt) {  
    botonLiberarPista1Madrid = true;  
    botonCerrarPista1Madrid = false;  
    BotonCerrarPista1Madrid.setEnabled(true);  
    BotonAbrirPista1Madrid.setEnabled(false);  
}
```

```
private void BotonAbrirPista2MadridActionPerformed(java.awt.event.ActionEvent evt) {  
    botonLiberarPista2Madrid = true;  
    botonCerrarPista2Madrid = false;  
    BotonCerrarPista2Madrid.setEnabled(true);  
    BotonAbrirPista2Madrid.setEnabled(false);  
}
```

```
private void BotonAbrirPista3MadridActionPerformed(java.awt.event.ActionEvent evt) {  
    botonLiberarPista3Madrid = true;  
    botonCerrarPista3Madrid = false;  
    BotonCerrarPista3Madrid.setEnabled(true);  
    BotonAbrirPista3Madrid.setEnabled(false);  
}
```

```
private void BotonAbrirPista4MadridActionPerformed(java.awt.event.ActionEvent evt) {  
    botonLiberarPista4Madrid = true;  
    botonCerrarPista4Madrid = false;  
    BotonCerrarPista4Madrid.setEnabled(true);  
    BotonAbrirPista4Madrid.setEnabled(false);  
}
```

```
private void BotonAbrirPista1BarcelonaActionPerformed(java.awt.event.ActionEvent  
evt) {
```

```
    botonLiberarPista1Barcelona = true;

    botonCerrarPista1Barcelona = false;

    BotonCerrarPista1Barcelona.setEnabled(true);

    BotonAbrirPista1Barcelona.setEnabled(false);
}
```

```
private void BotonAbrirPista2BarcelonaActionPerformed(java.awt.event.ActionEvent
evt) {

    botonLiberarPista2Barcelona = true;

    botonCerrarPista2Barcelona = false;

    BotonCerrarPista2Barcelona.setEnabled(true);

    BotonAbrirPista2Barcelona.setEnabled(false);
}
```

```
private void BotonAbrirPista3BarcelonaActionPerformed(java.awt.event.ActionEvent
evt) {

    botonLiberarPista3Barcelona = true;

    botonCerrarPista3Barcelona = false;

    BotonCerrarPista3Barcelona.setEnabled(true);

    BotonAbrirPista3Barcelona.setEnabled(false);
}
```

```
private void BotonAbrirPista4BarcelonaActionPerformed(java.awt.event.ActionEvent
evt) {

    botonLiberarPista4Barcelona = true;

    botonCerrarPista4Barcelona = false;

    BotonCerrarPista4Barcelona.setEnabled(true);

    BotonAbrirPista4Barcelona.setEnabled(false);
}
```

```
public boolean isBotonCerrarPista1Madrid() {

    return botonCerrarPista1Madrid;
}
```

```
}
```

```
public void setBotonCerrarPista1Madrid(boolean botonCerrarPista1Madrid) {  
    this.botonCerrarPista1Madrid = botonCerrarPista1Madrid;  
}
```

```
public boolean isBotonCerrarPista2Madrid() {  
    return botonCerrarPista2Madrid;  
}
```

```
public void setBotonCerrarPista2Madrid(boolean botonCerrarPista2Madrid) {  
    this.botonCerrarPista2Madrid = botonCerrarPista2Madrid;  
}
```

```
public boolean isBotonCerrarPista3Madrid() {  
    return botonCerrarPista3Madrid;  
}
```

```
public void setBotonCerrarPista3Madrid(boolean botonCerrarPista3Madrid) {  
    this.botonCerrarPista3Madrid = botonCerrarPista3Madrid;  
}
```

```
public boolean isBotonCerrarPista4Madrid() {  
    return botonCerrarPista4Madrid;  
}
```

```
public void setBotonCerrarPista4Madrid(boolean botonCerrarPista4Madrid) {  
    this.botonCerrarPista4Madrid = botonCerrarPista4Madrid;  
}
```

```
public boolean isBotonCerrarPista1Barcelona() {
```

```
        return botonCerrarPista1Barcelona;  
    }  
}
```

```
public void setBotonCerrarPista1Barcelona(boolean botonCerrarPista1Barcelona) {  
    this.botonCerrarPista1Barcelona = botonCerrarPista1Barcelona;  
}  
}
```

```
public boolean isBotonCerrarPista2Barcelona() {  
    return botonCerrarPista2Barcelona;  
}  
}
```

```
public void setBotonCerrarPista2Barcelona(boolean botonCerrarPista2Barcelona) {  
    this.botonCerrarPista2Barcelona = botonCerrarPista2Barcelona;  
}  
}
```

```
public boolean isBotonCerrarPista3Barcelona() {  
    return botonCerrarPista3Barcelona;  
}  
}
```

```
public void setBotonCerrarPista3Barcelona(boolean botonCerrarPista3Barcelona) {  
    this.botonCerrarPista3Barcelona = botonCerrarPista3Barcelona;  
}  
}
```

```
public boolean isBotonCerrarPista4Barcelona() {  
    return botonCerrarPista4Barcelona;  
}  
}
```

```
public void setBotonCerrarPista4Barcelona(boolean botonCerrarPista4Barcelona) {  
    this.botonCerrarPista4Barcelona = botonCerrarPista4Barcelona;  
}  
}
```

```
public boolean isBotonLiberarPista1Madrid() {  
    return botonLiberarPista1Madrid;  
}
```

```
public void setBotonLiberarPista1Madrid(boolean botonLiberarPista1Madrid) {  
    this.botonLiberarPista1Madrid = botonLiberarPista1Madrid;  
}
```

```
public boolean isBotonLiberarPista2Madrid() {  
    return botonLiberarPista2Madrid;  
}
```

```
public void setBotonLiberarPista2Madrid(boolean botonLiberarPista2Madrid) {  
    this.botonLiberarPista2Madrid = botonLiberarPista2Madrid;  
}
```

```
public boolean isBotonLiberarPista3Madrid() {  
    return botonLiberarPista3Madrid;  
}
```

```
public void setBotonLiberarPista3Madrid(boolean botonLiberarPista3Madrid) {  
    this.botonLiberarPista3Madrid = botonLiberarPista3Madrid;  
}
```

```
public boolean isBotonLiberarPista4Madrid() {  
    return botonLiberarPista4Madrid;  
}
```

```
public void setBotonLiberarPista4Madrid(boolean botonLiberarPista4Madrid) {  
    this.botonLiberarPista4Madrid = botonLiberarPista4Madrid;  
}
```



```
public boolean isBotonLiberarPista1Barcelona() {  
    return botonLiberarPista1Barcelona;  
}
```

```
public void setBotonLiberarPista1Barcelona(boolean botonLiberarPista1Barcelona) {  
    this.botonLiberarPista1Barcelona = botonLiberarPista1Barcelona;  
}
```

```
public boolean isBotonLiberarPista2Barcelona() {  
    return botonLiberarPista2Barcelona;  
}
```

```
public void setBotonLiberarPista2Barcelona(boolean botonLiberarPista2Barcelona) {  
    this.botonLiberarPista2Barcelona = botonLiberarPista2Barcelona;  
}
```

```
public boolean isBotonLiberarPista3Barcelona() {  
    return botonLiberarPista3Barcelona;  
}
```

```
public void setBotonLiberarPista3Barcelona(boolean botonLiberarPista3Barcelona) {  
    this.botonLiberarPista3Barcelona = botonLiberarPista3Barcelona;  
}
```

```
public boolean isBotonLiberarPista4Barcelona() {  
    return botonLiberarPista4Barcelona;  
}
```

```
public void setBotonLiberarPista4Barcelona(boolean botonLiberarPista4Barcelona) {  
    this.botonLiberarPista4Barcelona = botonLiberarPista4Barcelona;  
}
```

```

}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {

    /* Set the Nimbus look and feel */

    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and
    feel.
    * For details see
    http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
    */

    try {

        for (javax.swing.UIManager.LookAndFeelInfo info :
        javax.swing.UIManager.getInstalledLookAndFeels()) {

            if ("Nimbus".equals(info.getName())) {

                javax.swing.UIManager.setLookAndFeel(info.getClassName());

                break;

            }

        }

    } catch (ClassNotFoundException ex) {

        java.util.logging.Logger.getLogger(InterfazCliente.class.getName()).log(java.util.logging.L
        evel.SEVERE, null, ex);

    } catch (InstantiationException ex) {

        java.util.logging.Logger.getLogger(InterfazCliente.class.getName()).log(java.util.logging.L
        evel.SEVERE, null, ex);

    } catch (IllegalAccessException ex) {

        java.util.logging.Logger.getLogger(InterfazCliente.class.getName()).log(java.util.logging.L
        evel.SEVERE, null, ex);

```

```
} catch (javax.swing.UnsupportedLookAndFeelException ex) {
```

```
java.util.logging.Logger.getLogger(InterfazCliente.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
```

```
}
```

```
//</editor-fold>
```

```
//</editor-fold>
```

```
//</editor-fold>
```

```
//</editor-fold>
```

```
/* Create and display the form */
```

```
java.awt.EventQueue.invokeLater(new Runnable() {
```

```
    public void run() {
```

```
        try {
```

```
            new InterfazCliente().setVisible(true);
```

```
        } catch (IOException ex) {
```

```
        }
```

```
    }
```

```
});
```

```
}
```

```
// Variables declaration - do not modify
```

```
private javax.swing.JButton BotonAbrirPista1Barcelona;
```

```
private javax.swing.JButton BotonAbrirPista1Madrid;
```

```
private javax.swing.JButton BotonAbrirPista2Barcelona;
```

```
private javax.swing.JButton BotonAbrirPista2Madrid;
```

```
private javax.swing.JButton BotonAbrirPista3Barcelona;
```

```
private javax.swing.JButton BotonAbrirPista3Madrid;
```

```
private javax.swing.JButton BotonAbrirPista4Barcelona;
```

```
private javax.swing.JButton BotonAbrirPista4Madrid;
```

```
private javax.swing.JButton BotonCerrarPista1Barcelona;
```

```
private javax.swing.JButton BotonCerrarPista1Madrid;
private javax.swing.JButton BotonCerrarPista2Barcelona;
private javax.swing.JButton BotonCerrarPista2Madrid;
private javax.swing.JButton BotonCerrarPista3Barcelona;
private javax.swing.JButton BotonCerrarPista3Madrid;
private javax.swing.JButton BotonCerrarPista4Barcelona;
private javax.swing.JButton BotonCerrarPista4Madrid;
private javax.swing.JTextField TxtAeroviaBarcelona;
private javax.swing.JTextField TxtAeroviaMadrid;
private javax.swing.JTextField TxtAvionesAreaEstacionamientoBarcelona;
private javax.swing.JTextField TxtAvionesAreaEstacionamientoMadrid;
private javax.swing.JTextField TxtAvionesAreaRodajeBarcelona;
private javax.swing.JTextField TxtAvionesAreaRodajeMadrid;
private javax.swing.JTextField TxtAvionesHangarBarcelona;
private javax.swing.JTextField TxtAvionesHangarMadrid;
private javax.swing.JTextField TxtAvionesTallarBarcelona;
private javax.swing.JTextField TxtAvionesTallarMadrid;
private javax.swing.JTextField TxtPasajerosBarcelona;
private javax.swing.JTextField TxtPasajerosMadrid;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel11;
private javax.swing.JLabel jLabel12;
private javax.swing.JLabel jLabel13;
private javax.swing.JLabel jLabel14;
private javax.swing.JLabel jLabel15;
private javax.swing.JLabel jLabel16;
private javax.swing.JLabel jLabel17;
private javax.swing.JLabel jLabel18;
private javax.swing.JLabel jLabel19;
private javax.swing.JLabel jLabel2;
```

```

private javax.swing.JLabel jLabel20;
private javax.swing.JLabel jLabel21;
private javax.swing.JLabel jLabel22;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
// End of variables declaration
}
package Distribuida;

import Concurrída.InterfazServidor;

import java.rmi.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Servidor extends Thread {

    private InterfazServidor gui;

    public Servidor(InterfazServidor gui) {
        this.gui = gui;
    }

    @Override
    public void run() {
        try {

```

AeropuertoImpl obj = new AeropuertoImpl(gui); //Crea una instancia del objeto que implementa la interfaz, como objeto a registrar

Registry registry = LocateRegistry.createRegistry(1099); //Arranca rmiregistry local en el puerto 1099

Naming.rebind("//localhost/ObjetoAeropuerto", obj); //rebind sólo funciona sobre una url del equipo local

System.out.println("El ObjetoAeropuerto ha quedado registrado");

} catch (Exception e) {

System.out.println(" Error: " + e.getMessage());

e.printStackTrace();

}

}

}