

**06661503J - Rodríguez Sánchez, Nicolás - InfoAde**

**A10397664 - Luby, Paean - Computer Science**



**INDEX**

[**I) High-level analysis: 4**](#_hlsnqsu63hdg)

[How to run the programs: 4](#_poiq24mntc4r)

[Description of the problem: 4](#_pozh2pwmbn3b)

[**II) General design of the system and discussion of synchronization tools: 5**](#_2j4i2dxqccry)

[1. Locks: 5](#_choj0n2jy4lg)

[2. Conditions: 5](#_u8ab5docm8ml)

[3. BlockingQueues: 5](#_cpntrvvntucn)

[4. ConcurrentLinkedQueues: 6](#_jjadaexavw7t)

[5. Synchronized: 6](#_pdzmaiean3g)

[6. AtomicInteger: 6](#_fpp8nhj3otek)

[**III) Class descriptions: 6**](#_p9rl3a4wsgo4)

[1. AirplaneCreator extends Thread: 6](#_ggvnlzvc8hbh)

[2. Airplane extends Thread: 7](#_fe4rnrwww92p)

[3. Airport extends UnicastRemoteObject implements RemoteInterface: 8](#_qdvbswxgusd9)

[4. Airway: 9](#_opnycjwtzjqz)

[5. BoardingGates: 9](#_94jccxfd8o6f)

[6. Bus: 10](#_tv18pnqg1yaa)

[7. Bus Creator: 11](#_c4j5f66cyogb)

[8. Client (Main): 12](#_vzzj3g93li0t)

[9. DistributedProgrammingGUI extends javax.swing.JFrame: 12](#_mjw242j5e2r1)

[10. Gateway](#_tzt6j6g19feq):[. 12](#_n7rbxisxlhl)

[11. GraphicalInterface: 12](#_ds6n2k5vek3q)

[12. Log: 13](#_2bl3t0ct6q3j)

[13. Maintenance Hall: 13](#_qmwt85i3pm0b)

[14. Parking: 14](#_y5y00boyuod1)

[15. Runways: 15](#_ef3nuk2nnkbw)

[16. Remote Interface: 15](#_k3wbeac7g4fc)

[17. Server (main): 15](#_r1qcn6c54w1e)

[18. Taxi Area: 16](#_typ7aocs705r)

[**IV) Class diagram: 17**](#_krqu56sr1fzd)

[**V) Source code: 17**](#_imong6ysex95)

[Airplane: 17](#_xf0ltfew4502)

[AirplaneCreator: 35](#_7a6mynzaxtzf)

[Airport: 37](#_smb5sq4jv58u)

[Airway: 40](#_i2ioicxubq5c)

[BoardingGates: 42](#_gffjwuhti4ic)

[Bus: 44](#_2rkqtpx8ojt5)

[BusCreator: 48](#_83yetw4p34rz)

[Client: 49](#_c3wjy7vdi6bh)

[DistributedProgrammingGUI: 50](#_p7uh8wt97hh4)

[Gateway: 77](#_l4v0vlqcjtrf)

[GraphicalInterface: 78](#_94l08w5jy3cf)

[Hangar: 114](#_6d1kd3lnl05r)

[Log: 116](#_fxy57ra2oird)

[MaintenanceHall: 118](#_dwzmu3gb0h12)

[Parking: 120](#_xvmc1d3319t0)

[RemoteInterface: 122](#_dsbqmpavck9f)

[Runways: 123](#_u41zp75x8gp7)

[Server: 125](#_qgfmemyf8z98)

[TaxiArea: 126](#_x92bxfrjg51v)

# I) High-level analysis:

## 

## How to run the programs:

First, launch the Server by right-clicking on its file and selecting “Run”. Repeat this process for the Client. It is crucial that the Server is started before the Client to ensure proper connectivity and functionality.

## 

## Description of the Problem:

The problem centers on modeling zones of activity in the Madrid and Barcelona airports, along with the connections between them. The threads to simulate these activities are buses and airplanes that are generated simultaneously throughout the start of the program and continue their operations indefinitely. The buses’ purpose is to transport passengers between airports, while the airplanes have to access each of the zones. Some of these zones have a limited capacity or have sections that can only be entered or exited by one airplane at a time, requiring mutual exclusion mechanisms and data structures for synchronized communication to model realistic interactions within the airport ecosystem. Other than the airplane and bus threads, the program’s primary actors and their quantities are as follows:

1. **AirplaneCreator (1):** Generates the 8000 airplanes with their respective properties and unique identifiers in a staggered manner.
2. **BusCreator (1):** Generates the 4000 buses with their respective properties and unique identifiers in a staggered manner.
3. **Airports (2):** Comprise zones—hangars, maintenance halls, boarding gates, runways, parking areas, and taxi areas—accessed during the airplane and bus cycles.
   1. *Hangar:* Unlimited capacity
   2. *Maintenance Hall:* Maximum capacity of 20. FIFO entrance from parking.
   3. *Boarding Gates:* Maximum capacity of 6. FIFO entrance from parking.
   4. *Runways:* Maximum capacity of 4.
   5. *Parking area:* Unlimited capacity.
   6. *Taxi area:* Unlimited capacity.
4. **Passengers:** While not modeled as threads, the passengers’ movement is tracked in the simulation during transfer between buses, airplanes, and airports.
5. **Airways (2):** Connect the airports to simulate airplane flights.
6. **Downtown bus stops (2):** Interaction points for buses to pick up and drop off passengers.
7. **Log (1):** Records all activities in the simulation in real-time.
8. **Graphical User Interface (1):** Interface displaying real-time information about airport and connection operations, including the number of passengers, airplane statuses, and the ability to pause/resume the system.
9. **Distributed programming:** To address this requirement, Java RMI was used due to its straightforward integration within Java environments. This approach allowed the implementation of a client capable of interacting with remote objects. Through a defined remote interface, the client executes methods remotely, facilitating the real-time display of airport information on the interface.

# II) System design and synchronization tools:

## Locks:

Locks, specifically reentrant locks, were employed in methods that required mutual exclusion. These methods included recording data in the log and in the graphical interface. To avoid that more than one thread writes to the log simultaneously, the thread that wants to write something has to check if the lock is available, lock it, write and unlock it. We didn’t use write/read locks because the written data is only read by the user (through a UI, logs, files, etc.) and not read back into the program for further processing or decision-making, making the complexity of managing read/write locks unnecessary. Locks were set to be unfair because any thread could attempt to enter from the parking area to the maintenance hall or boarding gate, though only the first could enter.

## Conditions:

Locks and conditions were employed for managing states other than full and empty that concurrent data structures manage by default, including whether the airplane calling the method was at the front of the queue. Conditions were also employed to check if all runways or boarding gates were full so that airplane threads could be signaled when a boarding gate was available, as both of these structures were represented with arrays. These threads would be woken up with signalAll() in both classes due to the aforementioned possibility of any thread acquiring the lock to check if its first. Conditions were included in the gateway class as well for managing the pausing and resuming of the server GUI simulation.

## BlockingQueues:

Two types of blocking queues were used in the program, specifically the LinkedBlockingQueue and ArrayBlockingQueue. The LinkedBlockingQueue was used in the Parking and Taxi classes to avoid having to check if the queue was empty prior to releasing airplanes into boarding or maintenance. This blocking was particularly necessary because of the FIFO structure of the boarding gate entrance, as the thread would check prior to the transfer if it was at the head of the queue. Likewise, the maintenance hall class used an ArrayBlockingQueue to facilitate management of the 20 slots, as it would block any additional threads from accessing the maintenance hall until a spot is available.

## ConcurrentLinkedQueues:

ConcurrentLinkedQueues were leveraged in the Hangar and Airway classes to avoid ConcurrentModificationExceptions arising from Iterator in printing to the GUI because of their thread safety. The FIFO properties of this collection class (i.e.poll()) were not used so that each airplane could manage their own removal from the list regardless of their position.

## Synchronized:

We used synchronized to ensure mutual exclusion in shared methods between the server and the client in part 2, specifically openClose in the airport class. Therefore, if there is more than one client, they would be able access the shared method with mutual exclusion.

## AtomicInteger:

The passengers variable in the airport class is modeled as an AtomicInteger for lock-free thread-safe management of the variable’s modification by the bus and airplane threads.

# III) Class descriptions:

## AirplaneCreator extends Thread:

This class is intended to simultaneously generate airplane threads alongside bus threads. It has fields representing the variables that each plane will need to access in simulating flight operations, including the two airports, a log, and a graphical interface.

* 1. private String randomLetters(): This method generates a unique string identifier for each airplane, selecting a random letter from a list of letters in the alphabet and concatenating them.
  2. public void run():Uses a for loop to generate 8000 airplanes. Within the for loop, a 4-digit identifier is created by adding zeros to the loop counter. A capacity between 100 and 300 inclusive is generated using Math.random(). The airplane is assigned to have the Madrid airport as the starting airport and the Barcelona airport as the destination airport, in the case of an even identifier, or vice-versa in the case of an odd identifier. The airway is also assigned to be Mad\_Bar or Bar\_Mad via a boolean variable, with airplanes originating from the Madrid Airport originally using the Mad\_Bar airway (true) and the Barcelona airport using Bar\_Mad (false). After initializing the airplane with these values, the AirplaneCreator starts an Airplane thread and sleeps for 1 to 3 seconds before generating the next airplane.

## Airplane extends Thread:

The Airplane class models the operations and lifecycle of an airplane, simulating real-time interactions and behaviors in an airport environment. Its attributes comprise a constant CAPACITY, the current number of passengers onboard, the plane’s identifier, a log for recording events and operations, the starting airport for the airplane, the destination airport for the airplane, a GraphicalInterface to output visual updates of the airplane's status and location, a boolean indicating whether the airplane's current route is from Madrid to Barcelona, a boolean indicating if the airplane is currently landing, and an integer tracking the number of flights the airplane has completed.

* 1. public void run(): Defines the primary behavior for the airplane as a thread, repeatedly executing its lifecycle between two airports. Handles exceptions related to thread interruptions and remote operations.
  2. public void GraphicalTaxi, GraphicalParking, GraphicalBoardingGate, GraphicalRunway, GraphicalAirway (Airport airport): Update the graphical interface to reflect the airplane's current status and location at each point in the airport.
  3. public void lifeCycle(Airport airport1, Airport airport2): Orchestrates the sequence of events an airplane undergoes during a single round-trip between two airports, including boarding, taxiing, taking off, flying, landing, and potentially entering maintenance.
  4. public void switchAirway(): Toggles the route status (Madrid-Barcelona or Barcelona-Madrid) for determining the correct airway during travel or graphical updates.
  5. public Airway getAirway(Airport airport): Retrieves the appropriate airway based on the current route status.
  6. public void attemptBoarding(int passengersToTake, Airport airport): Manages the boarding process at a gate, including updating passenger counts and simulating the time taken for passengers to board.

## Airport extends UnicastRemoteObject implements RemoteInterface:

The Airport class models the functionalities and properties of an airport within a simulation system. It manages the facilities and operations concerning airplanes, such as hangars, taxi areas, runways, and boarding gates. It also implements a RemoteInterface, allowing for remote method invocation. Its attributes include an AtomicInteger currentPassengers to track the number of passengers currently at the airport with thread-safe updates, the String name of the airport, along with the component parts of the airport and airways between them.

* 1. public void addPassengers(int passengers): Adds a specified number of passengers to the airport, ensuring atomicity in the operation via the AtomicInteger.
  2. public int releasePassengers(int passengers): Reduces the number of passengers at the airport by a specified amount, ensuring the total never goes below zero. This method uses a loop with compareAndSet() for thread-safe and accurate updates.
  3. public AtomicInteger getPassengers(), public void setPassengers(AtomicInteger passengers): Getters and setters for managing the atomic passenger count.
  4. getHangar(), getMad\_Bar(), getBar\_Mad(), getTaxiArea(), getParking(), getMaintenanceHall(), getBoardingGates(), getRunways(): Accessor methods for the various facilities within the airport. Setters are not provided since these components are not modified during the simulation.
  5. public String to String(): Returns the name of the airport.
  6. public synchronized int numHangar(), numMaintenance(), numParking(), numTaxiArea(): Methods that return the count of airplanes in various parts of the airport.
  7. public synchronized int showMadBarAirway(), showBarMadAirway(): Methods that return string representations of the respective airways, showcasing the data encapsulated by the airport regarding its air routes.
  8. openClose(int runway, boolean opCl): Calls the openClose method of the airway. The method was designed to just have a remote object as the airplane, even if the method works on the runways, and is synchronized to prevent concurrent modifications that could lead to race conditions in the hypothetical case of multiple clients.

## Airway:

While the airway isn’t managed in a FIFO manner, the airplanes entering and exiting the airway are placed into a ConcurrentLinkedQueue airplanes without a capacity to facilitate thread management, including exceptions like ConcurrentModificationExceptions. The airway also has a String field name, “Mad-Bar” or “Bar\_Mad”, for recording in the log.

* 1. public void enterAirway(Airplane airplane): Adds the airplane in the argument to the end of the queue.
  2. public Airplane releaseAirplane(Airplane airplane, Log log): Removes the airplane in the argument from the queue (non-FIFO), logging if the airplane was successfully or not successfully removed.
  3. public Queue<Airplane> getAirplanes(): Returns the airplanes queue.
  4. getAirwayName(): Returns name of airway.
  5. public String toString(): Returns a list of airplanes in the airway along with the current number of passengers each holds.

## BoardingGates:

Each airport’s boarding gates are modeled as an array of 6 airplanes. A ReentrantLock gateLock is initialized with two conditions, full to indicate when there is no space in the boarding gate and first to wake up the airplane attempting to access a boarding gate when they are at the front of the parking queue, which allows for enforcement of FIFO.

* 1. public int enterGateFromParking(Airplane airplane, Airport airport) throws InterruptedException: This method controls the FIFO transition from the taxi area to the boarding gates. The excludedGate is initialized as 0, indicating that boarding planes cannot enter that gate. While the airplane that called the method is not the first in the queue, it will wait until a gate is available. This checking is done in a while loop that calls the private method isGatePresent(excludedGate), returning an available gate or -1 if no gate is currently available for boarding. It will then assign the open gate to the airplane directly removed from the parking area, returning the gate. The FIFO transition is maintained by the first condition signaling after the first airplane has been removed from parking and placed into the gate, along with a gateLock that surrounds the shared variable airplanes.
  2. public int enterGateFromTaxiArea(Airplane airplane, Airport airport) throws InterruptedException: This method controls the transition from the taxi area to the landing gates. The excludedGate is initialized as 1, indicating that landing planes cannot enter that gate. A local int gate variable is initialized to the result of the method isGatePresent(excludedGate), which returns an available gate or -1 if no gate is currently available for landing. Inside a while loop, the airplane thread will wait until a gate is available. It will then assign the open gate to the airplane directly removed from the taxi area, returning the gate. A lock gateLock is likewise employed to ensure mutual exclusion of the shared resource airplanes. The method returns the airplane supplied as an argument.
  3. public Airplane releaseGate(Airplane airplane) throws InterruptedException: The boarding gate of the airplane is determined with the indexOf method, converting the array to an ArrayList via the Arrays package. This index is then set to null to indicate an opening, which all threads are alerted to via the signalAll() method. The method returns the airplane supplied as an argument. If the airplane isn’t found in the boarding gate, an error message is printed.
  4. private int isGatePresent(int unusableGate): Uses a for loop to look for an unoccupied space in the boardingGate, ignoring the unusable gate. The method returns -1 if all of the boarding gates are full.

## Bus:

The Bus class models the behavior of a bus within a transportation system, handling the movements and interactions between passengers moving between downtown areas and an airport. It maintains internal state and interacts with graphical interfaces and logging systems to display user movement. It encapsulates a String identifier, an integer number of passengers currently on the bus, an instance of Log used to record operations and events concerning the bus, an airport with which it interacts, and a graphical interface to update and monitor the state of the bus visually.

* 1. private String getCity(): Returns Madrid or Barcelona depending on if the bus has an even or odd identifier, respectively.
  2. public void run(): Executes private methods in the following order, checking the pause and resume buttons, along with updating the graphical interface, between method calls.
     1. private void arriveDowntown(): Simulates entrance of passengers from the downtown bus stop for random time between 2 and 5 seconds with Thread.sleep().
     2. private void boardDowntownPassengers(): Adds a random number of passengers between 0 and 50 inclusive from the downtown bus stop to the bus.
     3. private void travelToAirport(): Simulates 5 to 10-second drive from the downtown bus stop to the airport with Thread.sleep().
     4. private void arriveAtAirport(): Simulates entrance of passengers to the airport for random time between 2 and 5 seconds with Thread.sleep().
     5. private void boardPassengersAtAirport(): Adds a random number of passengers between 0 and 50 inclusive from the airport bus stop to the bus.
     6. private void travelDowntown(): Simulates 5 to 10-second route to downtown with Thread.sleep().
     7. public void setPassengers(int passengers): Sets passengers field equal to argument.
     8. public int getPassengers(): Returns number of passengers currently on the bus.

## Bus Creator:

This class is intended to simultaneously generate bus threads alongside airplane threads. It has as fields the variables that each bus will need to access in simulating transport operations, including the two airports, a log, and a graphical interface.

* 1. public void run():Uses a for loop to generate 4000 airplanes. Within the for loop, a “B-” is concatenated with a 4-digit identifier that is created by adding zeros to the loop counter. Depending on whether the identifier is even or odd, the bus is assigned either to have the Madrid airport as the headquarter airport or the Barcelona airport in the case of an odd identifier. After initializing the bus with these values, the BusCreator starts the Bus thread and sleeps for 0.5 to 1 seconds before generating the next bus. The pause and resume buttons are also checked periodically throughout the loop.

## Client (Main):

This is the main class of the second part where the DistributedProgrammingGUI is created and set visible. It is also where the set method that inputs new information from the airports starts.

## DistributedProgrammingGUI extends javax.swing.JFrame:

This is the graphical interface of the second part. It has a set method for each text field so that all of the airport components can be displayed. It also has access to both airports remotely with the remoteInterface, along with buttons to open and close the runways via the shared method openClose().

1. public void set() throws RemoteException: Infinitely uploads new information from the airport’s shared methods to the text fields, accessing each field via a respective private method.
2. public static void main(String args[]) throws NotBoundException, MalformedURLException, RemoteException: Places the Runnable object on the Event Dispatch Thread, which is the thread that handles all GUI operations in Java. Main also initiates an instance of the DistributedProgrammingGUI class and makes it visible.
3. **Gateway:** The Gateway class is intended to provide a mechanism to stop and restart the program by pressing the pause or resume buttons. To do so, the class encapsulates a boolean that checks if the gateway is already closed or not, a ReentrantLock, and a Condition called stop.
   1. public void open() and public void close(): Locks the lock and changes the boolean to true to close it and to false to open it. The open() method signals all threads when the gateway is opened (i.e. the resume button is pressed) and the close() method puts all threads to sleep when the gateway is closed (i.e. the pause button is pressed).
   2. public void look(): Checks if the gateway has been opened or closed. The method first locks, and, after the gateway closes, it makes the calling thread wait. Inside the loop, the thread calls await() on the stop condition variable, and the thread will remain in this state until another thread calls open() or signalAll() on the same condition variable stop in the open() method to indicate that the simulation has resumed.

## GraphicalInterface extends javax.swing.JFrame : The graphical interface of part 1 that displays the progress of the airport simulation. The class contains JTextField, JLabel, JButton, and JPanel to display and interact with various aspects of airport operations, including gates, runways, maintenance areas, passenger counts, and transportation between the airport and town. To display the airport’s components, the class has different setter methods to set text fields equivalent to the simulation’s current properties, ensuring mutual exclusion via locks. It locks, it writes, and it unlocks. The gateways and the runways do not need to ensure mutual exclusion in the interface because only one airplane can be on a runway or in a boarding gate at a time.

## Log: This Log class records events in the airport simulation, ensuring mutual exclusion, in a log (airportEvolution.txt). For that, in the constructor of the class, a FileWriter is created with a relative path that corresponds to the text file and a BufferedWriter is created with that FileWriter.

## public void write(String text): Locks the write method via a ReentrantLock the events composing the airport simulation, including the local time.

## public void close(): Closes the FileWriter and the BufferedWriter, first flushing any remaining data in the buffer to the file to ensure all data is written. It then closes each to free up resources.

## Maintenance Hall:

The `MaintenanceHall` class is designed to manage the entry and release of airplanes into a maintenance facility from the parking area. It uses an ArrayBlockingQueue with a capacity of 20 to ensure FIFO entrance after all positions are occupied. A ReentrantLock is leveraged to simulate the entrance of a tight door, along with a Condition called first to ensure a FIFO transition from the parking area to the maintenance hall.

* 1. public void enterHallDoor(Airplane airplane, Airport airport): Handles the process of an airplane trying to enter the maintenance hall. It locks the method to ensure exclusive access to the tight door, checks if the airplane is the next expected to enter by peeking the parking queue, waiting if not via a Condition first, and logs the entry when it happens. After the airplane enters the hall, sleeping for 1 second to simulate the amount of time to enter, it signals all waiting airplanes to check if they are first in queue and releases the lock.
  2. public Airplane releaseHall(Airplane airplane): Handles the process of an airplane trying to exit the maintenance hall. It locks the method to ensure exclusive access to the tight door, and attempts to remove the airplane from the hall, returning the airplane if the removal is successful and false otherwise. After the airplane enters the hall, the thread sleeps for 1 second to simulate the amount of time to exit and releases the lock.
  3. public String toString(): Provides a string representation of all airplanes currently in the maintenance hall. It iterates through the airplanes queue and concatenates the identifiers of the airplanes.

## Parking:

The Parking class manages the storage and release of airplanes in an airport setting, specifically handling two different private queues: one for airplanes, airplanesForBoarding, waiting to board passengers and another, airplanesForMaintenance, for airplanes needing maintenance. Both are LinkedBlockingQueues for lock-free thread safety, along with to avoid having to check if the queue is empty prior to releasing airplanes into boarding or maintenance.

* 1. public void addAirplane(Airplane airplane): Adds an airplane to the appropriate queue based on whether it is landing for maintenance or preparing to board passengers. The method utilizes the offer method of BlockingQueue, which is a non-blocking operation, adding to the end of the queue.
  2. public Airplane releaseAirplaneForBoarding(Airplane airplane): Attempts to remove an airplane from the airplanesForBoarding queue for boarding purposes. This method uses poll, which removes the head of the queue, ensuring FIFO order is maintained. It outputs logs indicating the action and the current state of the queue.
  3. public Airplane releaseAirplaneForMaintenance(Airplane airplane): Similar to the boarding queue, this method removes an airplane from the airplanesForMaintenance queue for maintenance. It also logs details about the operation and updates on the queue’s state, ensuring FIFO strategy.
  4. public String toString():Provides a string representation of all airplanes currently in both parking queues, detailing their identifiers and, for the boarding queue, the number of passengers on each airplane.
  5. public Queue<Airplane> getAirplanesForBoarding() and public Queue<Airplane> getAirplanesForMaintenance(): These getter methods provide access to the queues, allowing their destination locations, boarding gates and maintenance hall, to view the airplanes currently waiting for boarding or maintenance.

## Runways:

The Runways class manages the operation of runways at an airport, specifically handling the allocation of runways to airplanes for takeoff or landing, and maintaining the status of each runway as open or closed. It encapsulates runways, an array of Airplane objects representing the planes currently occupying each of the four runways, openCloseList, an array of booleans that tracks whether each runway is open (true) or closed (false), along with a Lock and full Condition to prevent active waiting when a runway is not available.

* 1. public int enterRunway(Airplane airplane): Tries to place an airplane onto a free and open runway. If all runways are occupied, the method waits until a runway becomes available. This method uses a lock to ensure that two airplanes are not assigned the same runway simultaneously and returns the index of the runway the airplane enters, or -1 if unable to place the airplane.
  2. public Airplane releaseRunway(Airplane airplane): Removes an airplane from its assigned runway, making it available for other airplanes. The method checks if the given airplane is indeed on a runway and, if so, clears that spot and signals to other threads that a runway has become available. Returns the airplane if removed successfully, or null if the airplane was not found.
  3. public void openClose(int runway, boolean opCl): Opens or closes a specified runway based on the boolean value provided (true opens, false closes) and the gateway number provided by the graphical interface of the part 2 open and close buttons.

## Remote Interface: This is an interface where the shared methods between the server and the client are visible.

## Server (main):

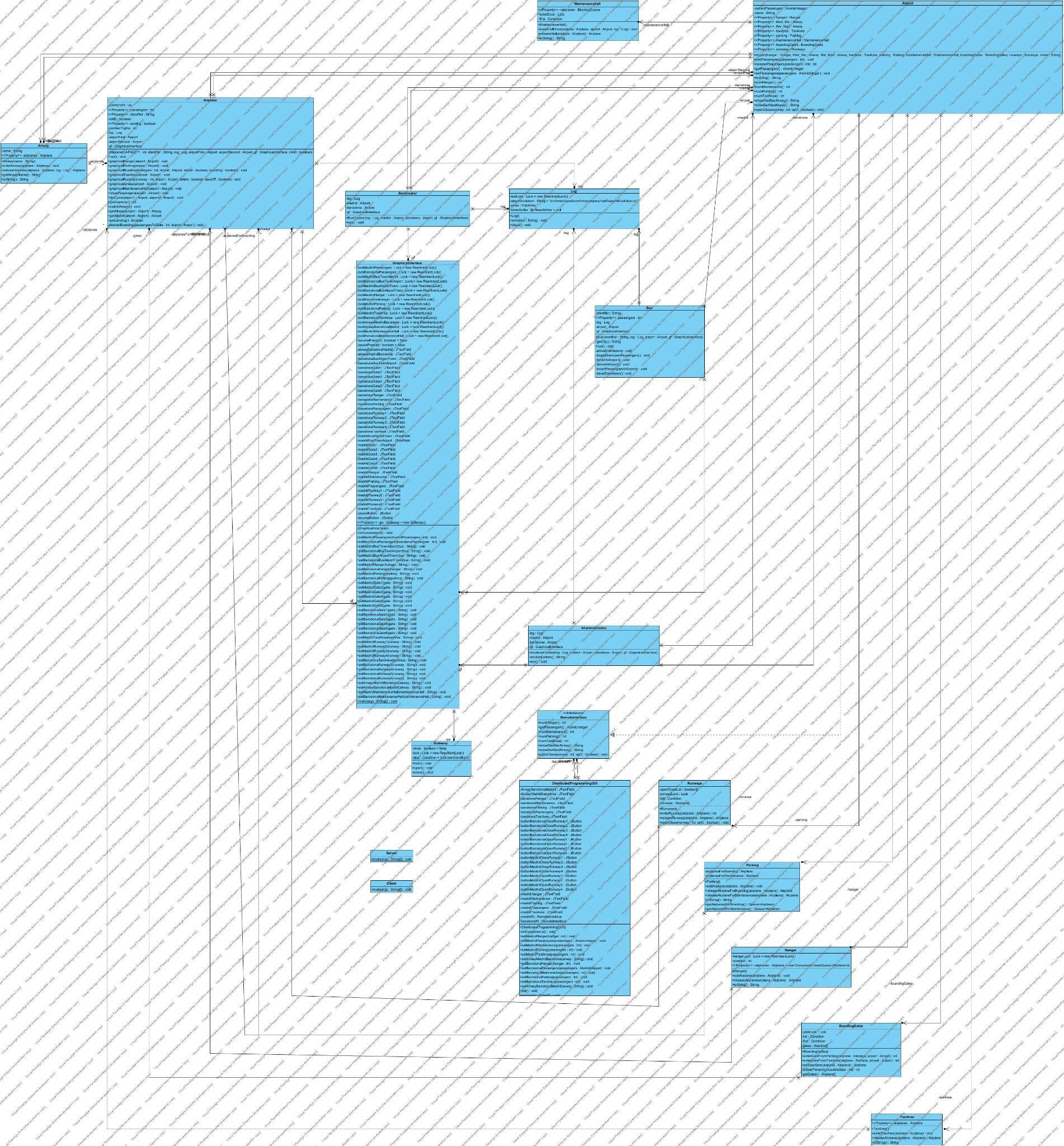
This is the main class of the first part where all the attributes of each airport are created, and where the airports are created. It is also where the GraphicalInterface of the first part is displayed, along with where the creator threads start their work. In this class, a connection to the remote server is started in port 1099, and Naming.rebind is likewise called for the remote objects (the airports) so that the client can access them. Finally, it ensures that if the program is finished or interrupted, the log is automatically closed to avoid the loss of information.

## Taxi Area:

The TaxiArea class simulates the management of an area in an airport where airplanes access a taxi area after landing or before takeoff, handling the queuing of airplanes as they enter and exit the taxi area with a thread-safe LinkedBlockingQueue. This queue manages the entry and exit of airplanes in FIFO order.

* 1. public void enterTaxiArea(Airplane airplane): Adds an airplane to the end of the queue.
  2. public BlockingQueue<Airplane> getAirplanes(): Getter method for the airplanes queue to allow for the retrieval of the queue in the connecting destination and origin areas.
  3. public Airplane releaseAirplane(Airplane airplane): Removes a specified airplane from the queue, returning the airplane after removing it from the queue.
  4. public String toString(): Provides a string representation of all airplanes currently in the taxi area. It iterates over the queue, appending details of each airplane, including its identifier and the number of passengers.

# IV) Class diagram:



The jLabel and automatically generated methods both graphical interfaces have been erased from this class diagram to make it more visible. It can be zoomed on the image contained in the submission folder called “SummaryClassDiagram”. The whole class diagram, including the jLabel and automatically generated JavaSwing components, can be seen and zoomed in the “ClassDiagram” image in the submission folder. The .vpp file can also be viewed in VisualParadigm.

# V) Source code\*:

\*Classes ordered alphabetically

## Airplane:

package com.mycompany.catl;

import java.rmi.RemoteException;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class Airplane extends Thread {

private final int CAPACITY;

private int passengers;

private String identifier;

private final Log log;

private final Airport airportFirst; //Starting airport

private final Airport airportSecond; //Destination airport

private final GraphicalInterface gf;

private boolean mAD; //Whether the airway is Mad\_Bar

private boolean landing; //Whether the plane is landing

private int numberFlights;

public Airplane(int CAPACITY, String identifier, Log log, Airport airportFirst, Airport airportSecond, GraphicalInterface gf, boolean mAD) {

this.CAPACITY = CAPACITY;

this.passengers = 0;

this.identifier = identifier;

this.log = log;

this.airportFirst = airportFirst;

this.airportSecond = airportSecond;

this.gf = gf;

this.numberFlights = 0;

this.mAD = mAD;

this.landing = false; //All airplanes start as boarding, so landing is false

this.log.write("Airplane " + this.getIdentifier() + " with capacity " + CAPACITY + " has been created.");

}

@Override

public void run() {

while (true) {

try {

lifeCycle(airportFirst, airportSecond);

lifeCycle(airportSecond, airportFirst);

} catch (InterruptedException | RemoteException ex) {

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

}

}

}

/\*\*

\* Graphical interface output of the hangar

\*

\* @param airport it is the current airport

\*/

public void graphicalHangar(Airport airport) {

if (airport.toString().equals("MAD")) {

gf.setMadridHangar(airport.getHangar().toString());

} else {

gf.setBarcelonaHangar(airport.getHangar().toString());

}

}

/\*\*

\* Graphical interface output of the parking

\*

\* @param airport it is the current airport

\*/

public void graphicalParking(Airport airport) {

if (airport.toString().equals("MAD")) {

gf.setMadridParking(airport.getParking().toString());

} else {

gf.setBarcelonaParking(airport.getParking().toString());

}

}

/\*\*

\* Graphical interface output of the different boarding gates

\*

\* @param gate it is the gate where the airplane has entered

\* @param airport it is the current airport

\* @param delete if it is true, it cleans the runway because the airplane

\* @param boarding is true if plane is boarding

\*/

public void graphicalBoardingGate(int gate, Airport airport, boolean delete, boolean boarding) {

if (delete) {

if (airport.toString().equals("MAD")) {

switch (gate) {

case 0 ->

gf.setMadridGate1("");

case 1 ->

gf.setMadridGate2("");

case 2 ->

gf.setMadridGate3("");

case 3 ->

gf.setMadridGate4("");

case 4 ->

gf.setMadridGate5("");

case 5 ->

gf.setMadridGate6("");

default ->

System.out.println("ERROR in boarding gates graphicalBoardingGate");

}

} else {

switch (gate) {

case 0 ->

gf.setBarcelonaGate1("");

case 1 ->

gf.setBarcelonaGate2("");

case 2 ->

gf.setBarcelonaGate3("");

case 3 ->

gf.setBarcelonaGate4("");

case 4 ->

gf.setBarcelonaGate5("");

case 5 ->

gf.setBarcelonaGate6("");

default ->

System.out.println("ERROR in boarding gates graphicalBoardingGate");

}

}

} else {

if (boarding) {

if (airport.toString().equals("MAD")) {

switch (gate) {

case 0 ->

gf.setMadridGate1("Boarding: " + this.getIdentifier());

case 1 ->

gf.setMadridGate2("Boarding: " + this.getIdentifier());

case 2 ->

gf.setMadridGate3("Boarding: " + this.getIdentifier());

case 3 ->

gf.setMadridGate4("Boarding: " + this.getIdentifier());

case 4 ->

gf.setMadridGate5("Boarding: " + this.getIdentifier());

case 5 ->

gf.setMadridGate6("Boarding: " + this.getIdentifier());

default ->

System.out.println("ERROR in boarding gates graphicalBoardingGate");

}

} else {

switch (gate) {

case 0 ->

gf.setBarcelonaGate1("Boarding: " + this.getIdentifier());

case 1 ->

gf.setBarcelonaGate2("Boarding: " + this.getIdentifier());

case 2 ->

gf.setBarcelonaGate3("Boarding: " + this.getIdentifier());

case 3 ->

gf.setBarcelonaGate4("Boarding: " + this.getIdentifier());

case 4 ->

gf.setBarcelonaGate5("Boarding: " + this.getIdentifier());

case 5 ->

gf.setBarcelonaGate6("Boarding: " + this.getIdentifier());

default ->

System.out.println("ERROR in boarding gates graphicalBoardingGate");

}

}

} else {

if (airport.toString().equals("MAD")) {

switch (gate) {

case 0 ->

gf.setMadridGate1("Disembark: " + this.getIdentifier());

case 1 ->

gf.setMadridGate2("Disembark: " + this.getIdentifier());

case 2 ->

gf.setMadridGate3("Disembark: " + this.getIdentifier());

case 3 ->

gf.setMadridGate4("Disembark: " + this.getIdentifier());

case 4 ->

gf.setMadridGate5("Disembark: " + this.getIdentifier());

case 5 ->

gf.setMadridGate6("Disembark: " + this.getIdentifier());

default ->

System.out.println("ERROR in boarding gates graphicalBoardingGate");

}

} else {

switch (gate) {

case 0 ->

gf.setBarcelonaGate1("Disembark: " + this.getIdentifier());

case 1 ->

gf.setBarcelonaGate2("Disembark: " + this.getIdentifier());

case 2 ->

gf.setBarcelonaGate3("Disembark: " + this.getIdentifier());

case 3 ->

gf.setBarcelonaGate4("Disembark: " + this.getIdentifier());

case 4 ->

gf.setBarcelonaGate5("Disembark: " + this.getIdentifier());

case 5 ->

gf.setBarcelonaGate6("Disembark: " + this.getIdentifier());

default ->

System.out.println("ERROR in boarding gates graphicalBoardingGate");

}

}

}

}

}

/\*\*

\* Graphical interface output of the taxiArea

\*

\* @param airport it is the current airport

\*/

public void graphicalTaxiArea(Airport airport) {

if (airport.toString().equals("MAD")) {

if (!airport.getTaxiArea().getAirplanes().isEmpty()) {

gf.setMadridTaxiArea(airport.getTaxiArea().toString());

} else {

gf.setMadridTaxiArea("");

}

} else {

if (!airport.getTaxiArea().getAirplanes().isEmpty()) {

gf.setBarcelonaTaxiArea(airport.getTaxiArea().toString());

} else {

gf.setBarcelonaTaxiArea("");

}

}

}

/\*\*

\* Graphical interface output of the different runways

\*

\* @param runway it is the gate where the airplane has entered

\* @param airport it is the current airport

\* @param delete if it is true, it cleans the runway because the airplane

\* has left; if it is false, it introduces the airplane into the boarding gate

\* @param takeOff true when the airplane takes off, false when it is landing

\*/

public void graphicalRunway(int runway, Airport airport, boolean delete, boolean takeOff) {

if (delete) {

if (airport.toString().equals("MAD")) {

switch (runway) {

case 0 ->

gf.setMadridRunway1("");

case 1 ->

gf.setMadridRunway2("");

case 2 ->

gf.setMadridRunway3("");

case 3 ->

gf.setMadridRunway4("");

default ->

System.out.println("ERROR in runway graphicalRunway");

}

} else {

switch (runway) {

case 0 ->

gf.setBarcelonaRunway1("");

case 1 ->

gf.setBarcelonaRunway2("");

case 2 ->

gf.setBarcelonaRunway3("");

case 3 ->

gf.setBarcelonaRunway4("");

default ->

System.out.println("ERROR in runway graphicalRunway");

}

}

} else {

if (airport.toString().equals("MAD")) {

switch (runway) {

case 0 ->

gf.setMadridRunway1(this.getIdentifier());

case 1 ->

gf.setMadridRunway2(this.getIdentifier());

case 2 ->

gf.setMadridRunway3(this.getIdentifier());

case 3 ->

gf.setMadridRunway4(this.getIdentifier());

default ->

System.out.println("ERROR in runway graphicalRunway");

}

} else {

switch (runway) {

case 0 ->

gf.setBarcelonaRunway1(this.getIdentifier());

case 1 ->

gf.setBarcelonaRunway2(this.getIdentifier());

case 2 ->

gf.setBarcelonaRunway3(this.getIdentifier());

case 3 ->

gf.setBarcelonaRunway4(this.getIdentifier());

default ->

System.out.println("ERROR in runway graphicalRunway");

}

}

if (takeOff) {

if (airport.toString().equals("MAD")) {

switch (runway) {

case 0 ->

gf.setMadridRunway1("Take-off: " + this.getIdentifier() + "(" + this.passengers + ")");

case 1 ->

gf.setMadridRunway2("Take-off: " + this.getIdentifier() + "(" + this.passengers + ")");

case 2 ->

gf.setMadridRunway3("Take-off: " + this.getIdentifier() + "(" + this.passengers + ")");

case 3 ->

gf.setMadridRunway4("Take-off: " + this.getIdentifier() + "(" + this.passengers + ")");

default ->

System.out.println("ERROR in runway graphicalRunway");

}

} else {

switch (runway) {

case 0 ->

gf.setBarcelonaRunway1("Take-off: " + this.getIdentifier() + "(" + this.passengers + ")");

case 1 ->

gf.setBarcelonaRunway2("Take-off: " + this.getIdentifier() + "(" + this.passengers + ")");

case 2 ->

gf.setBarcelonaRunway3("Take-off: " + this.getIdentifier() + "(" + this.passengers + ")");

case 3 ->

gf.setBarcelonaRunway4("Take-off: " + this.getIdentifier() + "(" + this.passengers + ")");

default ->

System.out.println("ERROR in runway graphicalRunway");

}

}

} else {

if (airport.toString().equals("MAD")) {

switch (runway) {

case 0 ->

gf.setMadridRunway1("Landing: " + this.getIdentifier() + "(" + this.passengers + ")");

case 1 ->

gf.setMadridRunway2("Landing: " + this.getIdentifier() + "(" + this.passengers + ")");

case 2 ->

gf.setMadridRunway3("Landing: " + this.getIdentifier() + "(" + this.passengers + ")");

case 3 ->

gf.setMadridRunway4("Landing: " + this.getIdentifier() + "(" + this.passengers + ")");

default ->

System.out.println("ERROR in runway graphicalRunway");

}

} else {

switch (runway) {

case 0 ->

gf.setBarcelonaRunway1("Landing: " + this.getIdentifier() + "(" + this.passengers + ")");

case 1 ->

gf.setBarcelonaRunway2("Landing: " + this.getIdentifier() + "(" + this.passengers + ")");

case 2 ->

gf.setBarcelonaRunway3("Landing: " + this.getIdentifier() + "(" + this.passengers + ")");

case 3 ->

gf.setBarcelonaRunway4("Landing: " + this.getIdentifier() + "(" + this.passengers + ")");

default ->

System.out.println("ERROR in runway graphicalRunway");

}

}

}

}

}

/\*\*

\* Graphical interface output of the airways

\*

\* @param airport it is the current airport

\*/

public void graphicalAirway(Airport airport) {

if (airport.toString().equals("MAD")) {

if (!airport.getMad\_Bar().getAirplanes().isEmpty()) {

gf.setAirwayMadridBarcelona(airport.getMad\_Bar().toString());

} else {

gf.setAirwayMadridBarcelona("");

}

} else {

if (!airport.getBar\_Mad().getAirplanes().isEmpty()) {

gf.setAirwayBarcelonaMadrid(airport.getBar\_Mad().toString());

} else {

gf.setAirwayBarcelonaMadrid("");

}

}

}

/\*\*

\* Graphical interface output of the maintenance hall

\*

\* @param airport it is the current airport

\*/

public void graphicalMaintenanceHall(Airport airport) {

if (airport.toString().equals("MAD")) {

gf.setMadridMaintenanceHall(airport.getMaintenanceHall().toString());

} else {

gf.setBarcelonaMaintenanceHall(airport.getMaintenanceHall().toString());

}

}

/\*\*

\* Graphical interface output of passengers in the client

\*

\* @param airport it is the current airport

\*/

public void showPassengers(Airport airport) throws RemoteException {

if (airport.toString().equals("MAD")) {

gf.setMadridPassengers(airport.getPassengers().get());

} else {

gf.setBarcelonaPassengers(airport.getPassengers().get());

}

}

/\*\*

\* Simulates one cycle of flights from the starting to the destination airport

\*

\* @param airport1 it is the starting airport

\* @param airport2 it is the destination airport

\*/

public void lifeCycle(Airport airport1, Airport airport2) throws InterruptedException, RemoteException {

numberFlights++; //Increment number of flights

gf.getGw().look(); //Check the pause/resume bottons

/\*

\* ================ ENTER HANGAR OF STARTING AIRPORT ========================

`\*/

if (numberFlights == 1) { //If flight is first flight

//Add airplane to hangar

airport1.getHangar().addAirplane(this);

this.graphicalHangar(airport1); //Update hangar GUI. Airplane should display in hangar

this.log.write("The airplane " + this.getIdentifier() + " has been created in the hangar of the airport of: " + airport1);

gf.getGw().look(); //Check the pause/resume bottons

}

/\*

\* ================ ENTER PARKING OF STARTING AIRPORT =======================

\*/

airport1.getParking().addAirplane(airport1.getHangar().releaseAirplane(this)); //Take the airplane from the hangar and put in it in parking

this.graphicalHangar(airport1);

this.graphicalParking(airport1);

this.log.write("The airplane " + this.getIdentifier() + " leaves the hangar and enters the parking of airport " + airport1 + ".");

gf.getGw().look(); //Check the pause/resume bottons

/\*

\* ================ ENTER BOARDING GATE OF STARTING AIRPORT =================

\*/

int gate;

gate = airport1.getBoardingGates().enterGateFromParking(this, airport1); //Enter into free boarding gate from parking

this.log.write("Airplane " + this.getIdentifier() + " has entered into boarding gate " + gate + " of airport " + airport1 + ".");

this.graphicalParking(airport1);

this.graphicalBoardingGate(gate, airport1, false, true);

gf.getGw().look(); //Check the pause/resume bottons

/\*

\* ================ ATTEMPT TO BOARD AT STARTING AIRPORT ====================

\*/

int remainingAttempts = 2;

this.attemptBoarding(Math.min(CAPACITY, airport1.getPassengers().get()), airport1); //Attempt 1. Takes maximum number of passengers or current number at airport

while (CAPACITY > this.getPassengers() && remainingAttempts > 0) { //While plane hasn't reached capacity from the airport passengers, keep picking them up

remainingAttempts--;

this.attemptBoarding(Math.min(CAPACITY-this.getPassengers(), airport1.getPassengers().get()), airport1); //If too many passengers take the number possible

this.graphicalBoardingGate(gate, airport1, false, true);

Thread.sleep((long) (Math.random() \* 4000 + 1000)); //Sleep for random time between 1 and 5 seconds if there aren't enough passengers

this.log.write("Airplane " + identifier + " is finishing boarding attempt " + (3 - remainingAttempts) + " with " + this.getPassengers() + " in gate " + gate + " of airport " + airport1 + ".");

gf.getGw().look(); //Check the pause/resume bottons

}

showPassengers(airport1);

this.graphicalBoardingGate(gate, airport1, false, true);

this.log.write("Airplane " + identifier + " is finishing boarding on attempt " + (3 - remainingAttempts) + ". Preparing for takeoff with " + this.getPassengers() + " from gate " + gate + " of airport " + airport1 + ".");

/\*

\* ================ ENTER TAXI AREA OF STARTING AIRPORT =================

\*/

airport1.getTaxiArea().enterTaxiArea(airport1.getBoardingGates().releaseGate(this)); //Enter taxi area

this.log.write("Airplane " + this.getIdentifier() + " has left the boarding gate and entered the taxi area of airport " + airport1 + ".");

this.graphicalBoardingGate(gate, airport1, true, false);

this.graphicalTaxiArea(airport1);

this.log.write("Airplane " + this.getIdentifier() + " completing checks in taxi area before requesting runway in " + airport1 + ".");

Thread.sleep((long) (1000 + Math.random() \* 4000)); //Check for period between 1 and 5 seconds

gf.getGw().look(); //Check the pause/resume bottons

/\*

\* ================ ENTER RUNWAY OF STARTING AIRPORT =================

\*/

int rw = airport1.getRunways().enterRunway(this);

while(rw==-1){

rw = airport1.getRunways().enterRunway(this);

}

airport1.getTaxiArea().releaseAirplane(this);

this.graphicalTaxiArea(airport1);

this.graphicalRunway(rw, airport1, false, true);

this.log.write("Airplane " + this.getIdentifier() + " completing final checks in runway of airport " + airport1);

Thread.sleep((long) Math.random() \* 2000 + 1000); //Final checks between 1 and 3 seconds

gf.getGw().look(); //Check the pause/resume bottons

this.log.write("Airplane " + this.getIdentifier() + " is taking off from runway of airport " + airport1);

Thread.sleep((long) Math.random() \* 4000 + 1000); //Take off between 1 and 5 seconds

gf.getGw().look(); //Check the pause/resume bottons

/\*

\* ================ ENTER AIRWAY OF STARTING AIRPORT =================

\*/

getAirway(airport1).enterAirway(airport1.getRunways().releaseRunway(this)); //Enter airway and remove it from the runway

this.log.write("Airplane " + this.getIdentifier() + " is entering airway " + getAirway(airport1).getAirwayName());

this.graphicalRunway(rw, airport1, true, true);

this.graphicalAirway(airport1);

Thread.sleep((long) (Math.random() \* 1500 + 1500)); //Flight between 15 and 30 seconds

gf.getGw().look(); //Check the pause/resume bottons

/\*

\* ================ REQUEST RUNWAY OF DESTINATION AIRPORT =================

\*/

//Attempt to access runway of the other airport

this.log.write("Airplane " + this.getIdentifier() + " requested runway for landing at airport " + airport2);

/\*

\* ================ ENTER RUNWAY OF DESTINATION AIRPORT =================

\*/

this.setLanding(true); //Set landing to true

int runway = airport2.getRunways().enterRunway(this);

while (runway == -1) {

this.log.write("Airplane " + this.getIdentifier() + " taking a detour.");

Thread.sleep((long) (Math.random() \* 4000 + 1000)); //Detour random time between 1 and 5 seconds

gf.getGw().look(); //Check the pause/resume bottons

runway = airport2.getRunways().enterRunway(this);

}

this.graphicalRunway(runway, airport2, false, false);

getAirway(airport1).releaseAirplane(this, log);

this.log.write("Airplane " + this.getIdentifier() + " entered runway " + runway + "at airport " + airport2);

this.graphicalAirway(airport1);

this.graphicalRunway(runway, airport2, false, false);

switchAirway(); //Switch airway to opposite

/\*

\* ================ LAND AT DESTINATION AIRPORT =================

\*/

Thread.sleep((long) (Math.random() \* 4000 + 1000)); //Land for a random time between 1 and 5 seconds

gf.getGw().look(); //Check the pause/resume bottons

/\*

\* ================ ENTER TAXI AREA OF DESTINATION AIRPORT =================

\*/

airport2.getTaxiArea().enterTaxiArea(airport2.getRunways().releaseRunway(this)); //Leave runway and directly access taxi area

this.log.write("Airplane " + this.getIdentifier() + " entered the taxi area of airport " + airport2);

gf.getGw().look(); //Check the pause/resume bottons

this.graphicalRunway(runway, airport2, true, false);

this.graphicalTaxiArea(airport2);

/\*

\* ================ ENTER BOARDING GATE OF DESTINATION AIRPORT =================

\*/

gate = airport2.getBoardingGates().enterGateFromTaxiArea(this, airport2);

gf.getGw().look(); //Check the pause/resume bottons

this.log.write("Airplane " + this.getIdentifier() + " flying between taxi area and boarding gate at airport " + airport2);

this.graphicalTaxiArea(airport2);

this.graphicalBoardingGate(gate, airport2, false, false);

Thread.sleep((long) (Math.random() \* 2000 + 3000)); //Flight time between 2 and 5 seconds

gf.getGw().look(); //Check the pause/resume bottons

/\*

\* ================ BEGIN DISEMBARKING =================

\*/

this.log.write("Airplane " + this.getIdentifier() + " disembarking " + this.getPassengers() + " at boarding gate of " + airport2);

Thread.sleep((long) (Math.random() \* 4000 + 1000)); //All passengers' transference from the airplane between 1 and 5 seconds

airport2.addPassengers(this.getPassengers());

this.setPassengers(0); //Reset passengers

gf.getGw().look(); //Check the pause/resume bottons

showPassengers(airport2);

/\*

\* ================ COMPLETE CHECKS IN PARKING AREA =================

\*/

airport2.getParking().addAirplane(airport2.getBoardingGates().releaseGate(this));

this.log.write("Airplane " + this.getIdentifier() + " disembarked . Accessing parking area of " + airport2);

gf.getGw().look(); //Check the pause/resume bottons

this.graphicalBoardingGate(gate, airport2, true, false);

this.graphicalParking(airport2);

this.log.write("Airplane " + this.getIdentifier() + " completing pilot checks in parking area of " + airport2);

Thread.sleep((long) (1000 + Math.random() \* 4000)); //Check for period between 1 and 5 seconds

gf.getGw().look(); //Check the pause/resume bottons

this.graphicalBoardingGate(gate, airport2, true, false);

this.graphicalParking(airport2);

/\*

\* ================ GO TO THE MAINTENANCE HALL FOR INSPECTION =================

\*/

this.log.write("Airplane " + this.getIdentifier() + " will attempt to enter maintenance hall door of airport " + airport2);

this.getAirport(airport2).getMaintenanceHall().enterHallDoor(this, airport2, log); //Directly pulls airplanes from parking area

gf.getGw().look(); //Check the pause/resume bottons

this.graphicalParking(airport2);

this.graphicalMaintenanceHall(airport2);

//Check if airplane needs to be sent to the maintenance hall for deep or light inspection

if (numberFlights % 15 == 0) { //if it's been 15 flight since last tune up

this.log.write("Airplane " + this.getIdentifier() + " going in for a deep inspection in maintenance hall of " + airport2);

Thread.sleep((long) (Math.random() \* 5000 + 5000)); //Inspection takes random time between 5 and 10 seconds

gf.getGw().look(); //Check the pause/resume bottons

} else {

this.log.write("Airplane " + this.getIdentifier() + " going in for a quick inspection in maintenance hall of " + airport2);

Thread.sleep((long) (Math.random() \* 1000 + 4000)); //Inspection takes random time between 1 and 5 seconds

gf.getGw().look(); //Check the pause/resume bottons

}

this.graphicalMaintenanceHall(airport2);

this.log.write("Airplane " + this.getIdentifier() + " finished inspection in airport " + airport2);

//Airplane decides to rest in hangar or continue life cycle

int choice = 1 + (int) (Math.random() \* 2); //50% chance

if (choice == 1) { //if choice 1, rest in hangar for 15 to 30 seconds

this.log.write("Airplane " + this.getIdentifier() + " resting in hangar of " + airport2);

airport2.getHangar().addAirplane(airport2.getMaintenanceHall().releaseHall(this));

this.graphicalMaintenanceHall(airport2);

this.graphicalHangar(airport2);

Thread.sleep((long) (Math.random() \* 15000 + 15000));

gf.getGw().look(); //Check the pause/resume bottons

} else {

this.log.write("Airplane " + this.getIdentifier() + " immediately restarting it's life cycle.");

airport2.getHangar().addAirplane(airport2.getMaintenanceHall().releaseHall(this));

this.graphicalMaintenanceHall(airport2);

this.graphicalHangar(airport2);

}

this.setLanding(false);

this.log.write("Airplane " + this.getIdentifier() + " has finished lap " + numberFlights + " from " + airport1 + " to " + airport2);

gf.getGw().look(); //Check the pause/resume bottons

}

public int getCapacity() {

return CAPACITY;

}

public int getPassengers() {

return passengers;

}

public void setPassengers(int passengers) {

this.passengers = passengers;

}

public String getIdentifier() {

return identifier;

}

public void setIdentifier(String identifier) {

this.identifier = identifier;

}

public void switchAirway() {

if (mAD) {

mAD = false;

}

}

public Airway getAirway(Airport airport) {

if (mAD) {

return airport.getMad\_Bar();

} else {

return airport.getBar\_Mad();

}

}

public Airport getAirport(Airport airport) {

return airport;

}

public boolean getLanding() {

return landing;

}

public void setLanding(boolean landing) {

this.landing = landing;

}

private void attemptBoarding(int passengersToTake, Airport airport) {

this.setPassengers(passengersToTake); //Add new passengers

this.getAirport(airport).releasePassengers(passengersToTake); //Subtract those passengers from the airport

try {

showPassengers(airport);

for (int i = 0; i < passengersToTake; i++) {

Thread.sleep((long) (Math.random() \* 2000 + 1000)); //Each passanger's transference to the airplane between 1 and 3 seconds

gf.getGw().look(); //Check the pause/resume bottons

}

} catch (RemoteException | InterruptedException ex) {

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

}

}

}

## AirplaneCreator:

package com.mycompany.catl;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class AirplaneCreator extends Thread {

private final Log log;

private final Airport madrid;

private final Airport barcelona;

private final GraphicalInterface gf;

public AirplaneCreator(Log log, Airport madrid, Airport barcelona, GraphicalInterface gf) {

this.log = log;

this.madrid = madrid;

this.barcelona = barcelona;

this.gf = gf;

}

/\*\*

\* Create a random identifier

\*

\* @return two random capital letters and a "-" Ex:XX-

\*/

private String randomLetters() {

List<String> alphabet = new ArrayList<>(Arrays.asList(

"A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M",

"N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"

));

int firstLetter = (int) (Math.random() \* 26); //Indices 0-25

int secondtLetter = (int) (Math.random() \* 26);

String randomLetters = alphabet.get(firstLetter) + alphabet.get(secondtLetter) + "-";

return randomLetters;

}

public void run() {

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

gf.getGw().look(); //Check the pause/resume bottons

Airplane airplane;

String identifier = String.valueOf(i);

while (identifier.length() != 4) { //If the identifier doesn't have 4 digits

identifier = "0" + identifier; //We add 0 until it has 4 digits

}

identifier = randomLetters() + identifier; //We add two random letters at the beginning

int capacity = (int) (Math.random() \* 201) + 100; //Capacity between 100 and 300

if (i % 2 == 0) { //Even identifier for Madrid

airplane = new Airplane(capacity, identifier, log, madrid, barcelona, gf, true);

} else { //Odd identifier for Barcelona

airplane = new Airplane(capacity, identifier, log, barcelona, madrid, gf, false);

}

airplane.start(); //Starts airplane thread

try {

Thread.sleep((long) (Math.random() \* 2000 + 1000)); //Random stagger time between 1 and 3 seconds for next airplane

} catch (InterruptedException ex) {

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

}

gf.getGw().look(); //Check the pause/resume bottons

}

}

}

## Airport:

package com.mycompany.catl;

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

import java.util.concurrent.atomic.AtomicInteger;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class Airport extends UnicastRemoteObject implements RemoteInterface {

private AtomicInteger currentPassengers;

private final Hangar hangar;

private final Airway Mad\_Bar;

private final Airway Bar\_Mad;

private final TaxiArea taxiArea;

private final Parking parking;

private final MaintenanceHall maintenanceHall;

private final BoardingGates boardingGates;

private final Runways runways;

private final String name;

public Airport(Hangar hangar, Airway Mad\_Bar, Airway Bar\_Mad, TaxiArea taxiArea, Parking parking, MaintenanceHall maintenanceHall, BoardingGates boardingGates, Runways runways, String name) throws RemoteException {

this.hangar = hangar;

this.Mad\_Bar = Mad\_Bar;

this.Bar\_Mad = Bar\_Mad;

this.taxiArea = taxiArea;

this.parking = parking;

this.maintenanceHall = maintenanceHall;

this.boardingGates = boardingGates;

this.runways = runways;

this.name = name;

this.currentPassengers = new AtomicInteger(0);

}

// Method to add passengers.

public void addPassengers(int passengers) {

currentPassengers.addAndGet(passengers);

}

// Method to offload passengers, ensuring total passengers never go below 0.

public int releasePassengers(int passengers) {

while (true) {

int nowPassengers = this.currentPassengers.get();

int passengersToOffload = Math.min(passengers, nowPassengers);

int newPassengers = Math.max(0, nowPassengers - passengers);

if (this.currentPassengers.compareAndSet(nowPassengers, newPassengers)) {

return passengersToOffload;

}

}

}

public AtomicInteger getPassengers() throws RemoteException {

return currentPassengers;

}

public void setPassengers(AtomicInteger passengers) {

this.currentPassengers = passengers;

}

public Hangar getHangar() {

return hangar;

}

public Airway getMad\_Bar() {

return Mad\_Bar;

}

public Airway getBar\_Mad() {

return Bar\_Mad;

}

public TaxiArea getTaxiArea() {

return taxiArea;

}

public Parking getParking() {

return parking;

}

public MaintenanceHall getMaintenanceHall() {

return maintenanceHall;

}

public BoardingGates getBoardingGates() {

return boardingGates;

}

public Runways getRunways() {

return runways;

}

@Override

public String toString() {

return this.name;

}

public int numHangar() throws RemoteException {

int num = this.hangar.getAirplanes().size();

return num;

}

public int numMaintenance() throws RemoteException {

int num = this.maintenanceHall.getAirplanes().size();

return num;

}

public int numParking() throws RemoteException {

int num = this.parking.getAirplanesForBoarding().size() + this.parking.getAirplanesForMaintenance().size();

return num;

}

public int numTaxiArea() throws RemoteException {

int num = this.taxiArea.getAirplanes().size();

return num;

}

public String showMadBarAirway() throws RemoteException {

String airway = this.Mad\_Bar.toString();

return airway;

}

public String showBarMadAirway() throws RemoteException {

String airway = this.Bar\_Mad.toString();

return airway;

}

/\* It opens or closes a runway. It is synchronized in case there is more than one client

\*

\*@param runway its the runway that is going to be open or closed

\*@param opCl If it is true it opens the runway, if it is false it closes the runway

\*/

public synchronized void openClose(int runway, boolean opCl) throws RemoteException {

this.runways.openClose(runway, opCl);

}

}

## Airway:

package com.mycompany.catl;

import java.util.Iterator;

import java.util.Queue;

import java.util.concurrent.ConcurrentLinkedQueue;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class Airway {

private Queue<Airplane> airplanes;

private final String name;

public Airway(String name) {

airplanes = new ConcurrentLinkedQueue<Airplane>();

this.name = name;

}

public void enterAirway(Airplane airplane) {

this.airplanes.add(airplane); //Add the airplane at the end of the list

}

public Airplane releaseAirplane(Airplane airplane, Log log) {

if (airplanes.remove(airplane)) { //Remove the airplane from whatever position in the list

log.write("Current airplanes in airway are: " + toString());

log.write("Airplane " + airplane.getIdentifier() + " was removed from airway " + name);

return airplane;

} else {

log.write("Error removing airplane " + airplane.getIdentifier() + " from airway " + name);

return null;

}

}

public Queue<Airplane> getAirplanes() {

return airplanes;

}

public String getAirwayName() {

return name;

}

@Override

public String toString() {

StringBuilder allPlanes = new StringBuilder();

Iterator<Airplane> airwayIterator = airplanes.iterator(); // Create a new iterator

while (airwayIterator.hasNext()) {

Airplane currPlane = airwayIterator.next(); // Assuming the object type is Airplane

String identifier = currPlane.getIdentifier();

int passengers = currPlane.getPassengers(); // Get the number of passengers

allPlanes.append(identifier + "(" + passengers + ")" + ", ");

}

return allPlanes.toString();

}

}

## BoardingGates:

package com.mycompany.catl;

import java.util.Arrays;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class BoardingGates {

private Airplane[] gates;

private Lock gateLock;

private Condition full;

private Condition first;

public BoardingGates() {

this.gates = new Airplane[6];

this.gateLock = new ReentrantLock();

this.full = gateLock.newCondition();

this.first = gateLock.newCondition();

}

public int enterGateFromParking(Airplane airplane, Airport airport) throws InterruptedException {

gateLock.lock();

int excludedGate = 0;

try {

int gate = isGatePresent(excludedGate);

while (!airplane.getAirport(airport).getParking().getAirplanesForBoarding().peek().equals(airplane)) {

first.await(); //Wait until the airplane wanting to enter is first in the list

}

while (gate == -1) {

full.await(); //While there is no available gate, wait until one becomes available for boarding

gate = isGatePresent(excludedGate);

}

gates[gate] = airplane.getAirport(airport).getParking().releaseAirplaneForBoarding(airplane); //Take airplane from airport and put in boarding gate when there's an opening

first.signalAll(); //Signal to all threads that a different thread is at the front of the queue

return gate;

} catch (ArrayIndexOutOfBoundsException e) {

System.err.println("Airplane " + airplane.getIdentifier() + " couldn't be transferred to the boarding gate from parking.");

return -1;

} finally {

gateLock.unlock();

}

}

public int enterGateFromTaxiArea(Airplane airplane, Airport airport) throws InterruptedException {

gateLock.lock();

int excludedGate = 1;

try {

int gate = isGatePresent(excludedGate);

while (isGatePresent(excludedGate) == -1) {

full.await(); //While there is no available gate, wait until one becomes available for boarding

gate = isGatePresent(excludedGate);

}

gates[gate] = airplane.getAirport(airport).getTaxiArea().releaseAirplane(airplane); //Take airplane from taxi area and put in boarding gate when there's an opening

return gate;

} finally {

gateLock.unlock();

}

}

public Airplane releaseGate(Airplane airplane) throws InterruptedException {

gateLock.lock();

try {

int planeIndex = Arrays.asList(gates).indexOf(airplane); //Find airplane in boarding gate list

gates[planeIndex] = null; //Remove airplane from boarding gates

full.signalAll(); //Signal to all threads that an opening is available

return airplane;

} finally {

gateLock.unlock();

}

}

private int isGatePresent(int unusableGate) {

for (int i = 0; i < gates.length; i++) {

if (i != unusableGate && gates[i] == null) { //Skip index that is not usable for current operation

return i;

}

}

return -1;

}

private Airplane[] getGates() {

return gates;

}

}

## Bus:

package com.mycompany.catl;

import java.util.logging.Level;

import java.util.logging.Logger;

import java.rmi.RemoteException;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class Bus extends Thread {

private final String identifier;

private int passengers;

private final Log log;

private final Airport airport;

private final GraphicalInterface gf;

public Bus(String identifier, Log log, Airport airport, GraphicalInterface gf) {

this.identifier = identifier;

this.log = log;

this.airport = airport;

this.gf = gf;

this.passengers = 0;

}

/\*\*

\* Determine if the bus is in Madrid or Barcelona

\*

\* @return Madrid if the bus has an even identifier, Barcelona if it has an

\* odd identifier

\*/

private String getCity() {

if (Character.getNumericValue(this.identifier.charAt(5)) % 2 == 0) {

return "Madrid";

} else {

return "Barcelona";

}

}

@Override

public void run() {

while (true) {

try {

//Arrival to downtown

gf.getGw().look(); //Check the pause/resume bottons

arriveDowntown();

//Passengers board

gf.getGw().look();//Check the pause/resume bottons

boardDowntownPassengers();

gf.getGw().look(); //Check the pause/resume bottons

//Update GUI

if (this.getCity().equals("Madrid")) {

gf.setMadridBusTownAirport(identifier + " (" + this.getPassengers() + ")");

} else {

gf.setBarcelonaBusTownAirport(identifier + " (" + this.getPassengers() + ")");

}

travelToAirport();

gf.getGw().look(); //Check the pause/resume bottons

this.log.write("The bus " + this.identifier + " has arrived to the airport of " + this.getCity() + " with " + this.getPassengers() + " passengers.");

arriveAtAirport();

gf.getGw().look(); //Check the pause/resume bottons

//Update GUI

if (this.getCity().equals("Madrid")) {

gf.setMadridPassengers(airport.getPassengers().get());

} else {

gf.setBarcelonaPassengers(airport.getPassengers().get());

}

gf.getGw().look(); //Check the pause/resume bottons

boardPassengersAtAirport();

//Update GUI

if (this.getCity().equals("Madrid")) {

gf.setMadridPassengers(airport.getPassengers().get());

} else {

gf.setBarcelonaPassengers(airport.getPassengers().get());

}

gf.getGw().look(); //Check the pause/resume bottons

//Update GUI

if (this.getCity().equals("Madrid")) {

gf.setMadridBusAirportTown(identifier + " (" + this.getPassengers() + ")");

} else {

gf.setBarcelonaBusAirportTown(identifier + " (" + this.getPassengers() + ")");

}

travelDowntown();

gf.getGw().look(); //Check the pause/resume bottons

this.log.write("The bus " + this.identifier + " has arrived to the downtown of " + this.getCity() + " with " + passengers + " passengers.");

} catch (RemoteException ex) {

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

}

}

}

private void arriveDowntown() {

this.log.write("The bus " + this.identifier + " has arrived to the city of " + this.getCity());

try {

Thread.sleep((long) (Math.random() \* 3000 + 2000));

gf.getGw().look();

} catch (InterruptedException ex) {

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

}

}

private void boardDowntownPassengers() {

int jumpIn = (int) (Math.random() \* 51);

this.setPassengers(jumpIn);

this.log.write(jumpIn + " passengers have accessed to the bus " + this.identifier + " that initiates its route toward the airport of " + this.getCity());

}

private void travelToAirport() {

try {

Thread.sleep((long) (Math.random() \* 5000 + 5000)); //Drive from downtown to the airport takes between 5 and 10 seconds

} catch (InterruptedException ex) {

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

}

}

private void arriveAtAirport() {

try {

//Bus waits for 2-5 seconds for new passengers

Thread.sleep((long) (Math.random() \* 3000 + 2000));

airport.addPassengers(this.getPassengers());

} catch (InterruptedException ex) {

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

}

}

private void boardPassengersAtAirport() {

int jumpIn = (int) (Math.random() \* 51);

this.setPassengers(this.airport.releasePassengers(jumpIn));

this.log.write(jumpIn + " passengers have accessed to the bus " + this.identifier + " that initiates its route towards the downtown of " + this.getCity());

}

private void travelDowntown() {

try {

Thread.sleep((long) (Math.random() \* 5000 + 5000));

} catch (InterruptedException ex) {

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

}

}

public void setPassengers(int passengers) {

this.passengers = passengers;

}

public int getPassengers() {

return passengers;

}

}

## BusCreator:

package com.mycompany.catl;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class BusCreator extends Thread {

private final Log log;

private final Airport madrid;

private final Airport barcelona;

private final GraphicalInterface gf;

public BusCreator(Log log, Airport madrid, Airport barcelona, GraphicalInterface gf) {

this.log = log;

this.madrid = madrid;

this.barcelona = barcelona;

this.gf = gf;

}

public void run() {

for (int i = 0; i < 4000; i++) {

gf.getGw().look(); //Check the pause/resume bottons

Bus bus;

String identifier = String.valueOf(i);

while (identifier.length() != 4) { //If the identifier doesn't have 4 digits

identifier = "0" + identifier; //We add 0 until it has 4 digits

}

identifier = "B-" + identifier; //We add B-

if (i % 2 == 0) { //Even identifier for Madrid

bus = new Bus(identifier, log, madrid, gf);

} else { //Odd identifier for Barcelona

bus = new Bus(identifier, log, barcelona, gf);

}

bus.start();

long sleepTime = (long) (Math.random() \* 500 + 500); //Random between 0,5 and 1

try {

Thread.sleep(sleepTime); //Sleeps between 0,5 and 1 second between each bus

} catch (InterruptedException ex) {

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

}

gf.getGw().look(); //Check the pause/resume bottons

}

}

}

## Client:

package com.mycompany.catl;

import java.rmi.RemoteException;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class Client {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

DistributedProgrammingGUI gui = new DistributedProgrammingGUI();

gui.setVisible(true);

try {

gui.set();

} catch (RemoteException ex) {

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

}

}

}

## DistributedProgrammingGUI:

package com.mycompany.catl;

import java.net.MalformedURLException;

import java.rmi.Naming;

import java.rmi.NotBoundException;

import java.rmi.RemoteException;

import java.util.concurrent.atomic.AtomicInteger;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class DistributedProgrammingGUI extends javax.swing.JFrame {

private RemoteInterface madridRi;

private RemoteInterface barcelonaRi;

/\*\*

\* Creates new form DistributedProgrammingGUI

\*/

public DistributedProgrammingGUI() {

initComponents();

try {

madridRi = (RemoteInterface) Naming.lookup("//localhost/madrid");

barcelonaRi = (RemoteInterface) Naming.lookup("//localhost/barcelona");

} catch (NotBoundException | MalformedURLException | RemoteException ex) {

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

}

}

/\*\*

\* 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() {

jLabel1 = new javax.swing.JLabel();

jLabel2 = new javax.swing.JLabel();

jLabel3 = new javax.swing.JLabel();

jLabel4 = new javax.swing.JLabel();

jLabel5 = new javax.swing.JLabel();

jLabel6 = new javax.swing.JLabel();

jLabel7 = new javax.swing.JLabel();

jLabel8 = new javax.swing.JLabel();

buttonMadridCloseRunway1 = new javax.swing.JButton();

buttonMadridOpenRunway1 = new javax.swing.JButton();

jLabel9 = new javax.swing.JLabel();

buttonMadridCloseRunway2 = new javax.swing.JButton();

buttonMadridOpenRunway2 = new javax.swing.JButton();

jLabel10 = new javax.swing.JLabel();

jLabel11 = new javax.swing.JLabel();

buttonMadridCloseRunway3 = new javax.swing.JButton();

buttonMadridCloseRunway4 = new javax.swing.JButton();

buttonMadridOpenRunway4 = new javax.swing.JButton();

buttonMadridOpenRunway3 = new javax.swing.JButton();

madridHangar = new javax.swing.JTextField();

madridParking = new javax.swing.JTextField();

madridPassengers = new javax.swing.JTextField();

madridTaxiArea = new javax.swing.JTextField();

madridMaintenance = new javax.swing.JTextField();

jLabel12 = new javax.swing.JLabel();

barcelonaPassengers = new javax.swing.JTextField();

jLabel13 = new javax.swing.JLabel();

barcelonaHangar = new javax.swing.JTextField();

jLabel14 = new javax.swing.JLabel();

barcelonaMaintenance = new javax.swing.JTextField();

jLabel15 = new javax.swing.JLabel();

barcelonaParking = new javax.swing.JTextField();

jLabel16 = new javax.swing.JLabel();

barcelonaTaxiArea = new javax.swing.JTextField();

jLabel17 = new javax.swing.JLabel();

buttonBarcelonaCloseRunway1 = new javax.swing.JButton();

buttonBarcelonaOpenRunway1 = new javax.swing.JButton();

jLabel18 = new javax.swing.JLabel();

buttonBarcelonaCloseRunway3 = new javax.swing.JButton();

buttonBarcelonaOpenRunway3 = new javax.swing.JButton();

jLabel19 = new javax.swing.JLabel();

buttonBarcelonaCloseRunway2 = new javax.swing.JButton();

buttonBarcelonaOpenRunway2 = new javax.swing.JButton();

jLabel20 = new javax.swing.JLabel();

buttonBarcelonaCloseRunway4 = new javax.swing.JButton();

buttonBarcelonaOpenRunway4 = new javax.swing.JButton();

jLabel21 = new javax.swing.JLabel();

AirwayMadridBarcelona = new javax.swing.JTextField();

jLabel22 = new javax.swing.JLabel();

AirwayBarcelonaMadrid = new javax.swing.JTextField();

jSeparator1 = new javax.swing.JSeparator();

jSeparator2 = new javax.swing.JSeparator();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

jLabel1.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N

jLabel1.setText("Madrid Airport");

jLabel2.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N

jLabel2.setText("Barcelona Airport");

jLabel3.setText("Number of passengers at the Airport:");

jLabel4.setText("Number of airplanes in Hangar:");

jLabel5.setText("Number of airplanes in Maintenance:");

jLabel6.setText("Number of airplanes in Parking:");

jLabel7.setText("Number of airplanes in Taxi Area:");

jLabel8.setText("Runway 1");

buttonMadridCloseRunway1.setText("Close");

buttonMadridCloseRunway1.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonMadridCloseRunway1ActionPerformed(evt);

}

});

buttonMadridOpenRunway1.setText("Open");

buttonMadridOpenRunway1.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonMadridOpenRunway1ActionPerformed(evt);

}

});

jLabel9.setText("Runway 2");

buttonMadridCloseRunway2.setText("Close");

buttonMadridCloseRunway2.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonMadridCloseRunway2ActionPerformed(evt);

}

});

buttonMadridOpenRunway2.setText("Open");

buttonMadridOpenRunway2.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonMadridOpenRunway2ActionPerformed(evt);

}

});

jLabel10.setText("Runway 4");

jLabel11.setText("Runway 3");

buttonMadridCloseRunway3.setText("Close");

buttonMadridCloseRunway3.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonMadridCloseRunway3ActionPerformed(evt);

}

});

buttonMadridCloseRunway4.setText("Close");

buttonMadridCloseRunway4.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonMadridCloseRunway4ActionPerformed(evt);

}

});

buttonMadridOpenRunway4.setText("Open");

buttonMadridOpenRunway4.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonMadridOpenRunway4ActionPerformed(evt);

}

});

buttonMadridOpenRunway3.setText("Open");

buttonMadridOpenRunway3.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonMadridOpenRunway3ActionPerformed(evt);

}

});

madridHangar.setEditable(false);

madridHangar.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

madridHangarActionPerformed(evt);

}

});

madridParking.setEditable(false);

madridParking.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

madridParkingActionPerformed(evt);

}

});

madridPassengers.setEditable(false);

madridPassengers.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

madridPassengersActionPerformed(evt);

}

});

madridTaxiArea.setEditable(false);

madridTaxiArea.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

madridTaxiAreaActionPerformed(evt);

}

});

madridMaintenance.setEditable(false);

madridMaintenance.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

madridMaintenanceActionPerformed(evt);

}

});

jLabel12.setText("Number of passengers at the Airport:");

barcelonaPassengers.setEditable(false);

barcelonaPassengers.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

barcelonaPassengersActionPerformed(evt);

}

});

jLabel13.setText("Number of airplanes in Hangar:");

barcelonaHangar.setEditable(false);

barcelonaHangar.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

barcelonaHangarActionPerformed(evt);

}

});

jLabel14.setText("Number of airplanes in Maintenance:");

barcelonaMaintenance.setEditable(false);

barcelonaMaintenance.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

barcelonaMaintenanceActionPerformed(evt);

}

});

jLabel15.setText("Number of airplanes in Parking:");

barcelonaParking.setEditable(false);

barcelonaParking.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

barcelonaParkingActionPerformed(evt);

}

});

jLabel16.setText("Number of airplanes in Taxi Area:");

barcelonaTaxiArea.setEditable(false);

barcelonaTaxiArea.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

barcelonaTaxiAreaActionPerformed(evt);

}

});

jLabel17.setText("Runway 1");

buttonBarcelonaCloseRunway1.setText("Close");

buttonBarcelonaCloseRunway1.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonBarcelonaCloseRunway1ActionPerformed(evt);

}

});

buttonBarcelonaOpenRunway1.setText("Open");

buttonBarcelonaOpenRunway1.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonBarcelonaOpenRunway1ActionPerformed(evt);

}

});

jLabel18.setText("Runway 3");

buttonBarcelonaCloseRunway3.setText("Close");

buttonBarcelonaCloseRunway3.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonBarcelonaCloseRunway3ActionPerformed(evt);

}

});

buttonBarcelonaOpenRunway3.setText("Open");

buttonBarcelonaOpenRunway3.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonBarcelonaOpenRunway3ActionPerformed(evt);

}

});

jLabel19.setText("Runway 2");

buttonBarcelonaCloseRunway2.setText("Close");

buttonBarcelonaCloseRunway2.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonBarcelonaCloseRunway2ActionPerformed(evt);

}

});

buttonBarcelonaOpenRunway2.setText("Open");

buttonBarcelonaOpenRunway2.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonBarcelonaOpenRunway2ActionPerformed(evt);

}

});

jLabel20.setText("Runway 4");

buttonBarcelonaCloseRunway4.setText("Close");

buttonBarcelonaCloseRunway4.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonBarcelonaCloseRunway4ActionPerformed(evt);

}

});

buttonBarcelonaOpenRunway4.setText("Open");

buttonBarcelonaOpenRunway4.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

buttonBarcelonaOpenRunway4ActionPerformed(evt);

}

});

jLabel21.setText("Airway Madrid-Barcelona");

AirwayMadridBarcelona.setEditable(false);

AirwayMadridBarcelona.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

AirwayMadridBarcelonaActionPerformed(evt);

}

});

jLabel22.setText("Airway Barcelona-Madrid");

AirwayBarcelonaMadrid.setEditable(false);

AirwayBarcelonaMadrid.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

AirwayBarcelonaMadridActionPerformed(evt);

}

});

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

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

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

.addGroup(layout.createSequentialGroup()

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

.addGroup(layout.createSequentialGroup()

.addContainerGap()

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

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel7)

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

.addComponent(madridTaxiArea))

.addGroup(layout.createSequentialGroup()

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

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel8)

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

.addComponent(buttonMadridCloseRunway1, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE)

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

.addComponent(buttonMadridOpenRunway1, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel9)

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

.addComponent(buttonMadridCloseRunway2, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE)

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

.addComponent(buttonMadridOpenRunway2, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE)))

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

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

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel11)

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

.addComponent(buttonMadridCloseRunway3, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE)

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

.addComponent(buttonMadridOpenRunway3, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel10)

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

.addComponent(buttonMadridCloseRunway4, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE)

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

.addComponent(buttonMadridOpenRunway4, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE))))

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel6)

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

.addComponent(madridParking))

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel3)

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

.addComponent(madridPassengers))

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel5)

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

.addComponent(madridMaintenance))

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel4)

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

.addComponent(madridHangar, javax.swing.GroupLayout.PREFERRED\_SIZE, 192, javax.swing.GroupLayout.PREFERRED\_SIZE))))

.addGroup(layout.createSequentialGroup()

.addGap(105, 105, 105)

.addComponent(jLabel1)))

.addGap(4, 4, 4)

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

.addGroup(layout.createSequentialGroup()

.addGap(0, 0, Short.MAX\_VALUE)

.addComponent(jLabel2)

.addGap(153, 153, 153))

.addGroup(layout.createSequentialGroup()

.addComponent(jSeparator2, javax.swing.GroupLayout.PREFERRED\_SIZE, 7, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(7, 7, 7)

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

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel16)

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

.addComponent(barcelonaTaxiArea))

.addGroup(layout.createSequentialGroup()

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

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel17)

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

.addComponent(buttonBarcelonaCloseRunway1, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE)

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

.addComponent(buttonBarcelonaOpenRunway1, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel19)

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

.addComponent(buttonBarcelonaCloseRunway2, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE)

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

.addComponent(buttonBarcelonaOpenRunway2, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE)))

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

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

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel18)

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

.addComponent(buttonBarcelonaCloseRunway3, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE)

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

.addComponent(buttonBarcelonaOpenRunway3, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel20)

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

.addComponent(buttonBarcelonaCloseRunway4, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE)

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

.addComponent(buttonBarcelonaOpenRunway4, javax.swing.GroupLayout.PREFERRED\_SIZE, 61, javax.swing.GroupLayout.PREFERRED\_SIZE))))

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel15)

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

.addComponent(barcelonaParking))

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel12)

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

.addComponent(barcelonaPassengers))

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel14)

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

.addComponent(barcelonaMaintenance))

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel13)

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

.addComponent(barcelonaHangar, javax.swing.GroupLayout.PREFERRED\_SIZE, 192, javax.swing.GroupLayout.PREFERRED\_SIZE)))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))))

.addGroup(layout.createSequentialGroup()

.addContainerGap()

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

.addComponent(AirwayMadridBarcelona)

.addComponent(AirwayBarcelonaMadrid)

.addComponent(jSeparator1))

.addContainerGap())

.addGroup(layout.createSequentialGroup()

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

.addGroup(layout.createSequentialGroup()

.addGap(313, 313, 313)

.addComponent(jLabel21))

.addGroup(layout.createSequentialGroup()

.addGap(312, 312, 312)

.addComponent(jLabel22)))

.addGap(0, 0, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

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

.addGroup(layout.createSequentialGroup()

.addContainerGap()

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

.addComponent(jLabel1)

.addComponent(jLabel2))

.addGap(18, 18, 18)

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

.addGroup(layout.createSequentialGroup()

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

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

.addComponent(jLabel3)

.addComponent(madridPassengers, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addComponent(jSeparator2))

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

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

.addComponent(jLabel4)

.addComponent(madridHangar, 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(madridMaintenance, 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(jLabel12)

.addComponent(barcelonaPassengers, 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)

.addComponent(barcelonaHangar, 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(barcelonaMaintenance, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 18, Short.MAX\_VALUE)

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

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

.addGroup(layout.createSequentialGroup()

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

.addComponent(jLabel6)

.addComponent(madridParking, 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(madridTaxiArea, 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(jLabel8)

.addComponent(buttonMadridCloseRunway1, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(buttonMadridOpenRunway1, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE))

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

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

.addComponent(jLabel9)

.addComponent(buttonMadridCloseRunway2, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(buttonMadridOpenRunway2, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE)))

.addGroup(layout.createSequentialGroup()

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

.addComponent(jLabel11)

.addComponent(buttonMadridCloseRunway3, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(buttonMadridOpenRunway3, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE))

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

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

.addComponent(jLabel10)

.addComponent(buttonMadridCloseRunway4, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(buttonMadridOpenRunway4, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE))))

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

.addGroup(layout.createSequentialGroup()

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

.addComponent(jLabel15)

.addComponent(barcelonaParking, 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)

.addComponent(barcelonaTaxiArea, 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(jLabel17)

.addComponent(buttonBarcelonaCloseRunway1, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(buttonBarcelonaOpenRunway1, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE))

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

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

.addComponent(jLabel19)

.addComponent(buttonBarcelonaCloseRunway2, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(buttonBarcelonaOpenRunway2, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE)))

.addGroup(layout.createSequentialGroup()

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

.addComponent(jLabel18)

.addComponent(buttonBarcelonaCloseRunway3, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(buttonBarcelonaOpenRunway3, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE))

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

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

.addComponent(jLabel20)

.addComponent(buttonBarcelonaCloseRunway4, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(buttonBarcelonaOpenRunway4, javax.swing.GroupLayout.PREFERRED\_SIZE, 16, javax.swing.GroupLayout.PREFERRED\_SIZE)))))

.addGap(11, 11, 11)

.addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED\_SIZE, 10, javax.swing.GroupLayout.PREFERRED\_SIZE)

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

.addComponent(jLabel21)

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

.addComponent(AirwayMadridBarcelona, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(18, 18, 18)

.addComponent(jLabel22)

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

.addComponent(AirwayBarcelonaMadrid, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(10, 10, 10))

);

pack();

}// </editor-fold>

private void buttonMadridCloseRunway1ActionPerformed(java.awt.event.ActionEvent evt) {

try {

madridRi.openClose(0, false);

} catch (RemoteException ex) {

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

}

}

private void buttonMadridOpenRunway1ActionPerformed(java.awt.event.ActionEvent evt) {

try {

madridRi.openClose(0, true);

} catch (RemoteException ex) {

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

}

}

private void buttonMadridCloseRunway2ActionPerformed(java.awt.event.ActionEvent evt) {

try {

madridRi.openClose(1, false);

} catch (RemoteException ex) {

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

}

}

private void buttonMadridOpenRunway2ActionPerformed(java.awt.event.ActionEvent evt) {

try {

madridRi.openClose(1, true);

} catch (RemoteException ex) {

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

}

}

private void buttonMadridCloseRunway3ActionPerformed(java.awt.event.ActionEvent evt) {

try {

madridRi.openClose(2, false);

} catch (RemoteException ex) {

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

}

}

private void buttonMadridCloseRunway4ActionPerformed(java.awt.event.ActionEvent evt) {

try {

madridRi.openClose(3, false);

} catch (RemoteException ex) {

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

}

}

private void buttonMadridOpenRunway4ActionPerformed(java.awt.event.ActionEvent evt) {

try {

madridRi.openClose(3, true);

} catch (RemoteException ex) {

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

}

}

private void buttonMadridOpenRunway3ActionPerformed(java.awt.event.ActionEvent evt) {

try {

madridRi.openClose(2, true);

} catch (RemoteException ex) {

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

}

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

private void buttonBarcelonaCloseRunway1ActionPerformed(java.awt.event.ActionEvent evt) {

try {

barcelonaRi.openClose(0, false);

} catch (RemoteException ex) {

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

}

}

private void buttonBarcelonaOpenRunway1ActionPerformed(java.awt.event.ActionEvent evt) {

try {

barcelonaRi.openClose(0, true);

} catch (RemoteException ex) {

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

}

}

private void buttonBarcelonaCloseRunway3ActionPerformed(java.awt.event.ActionEvent evt) {

try {

barcelonaRi.openClose(2, false);

} catch (RemoteException ex) {

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

}

}

private void buttonBarcelonaOpenRunway3ActionPerformed(java.awt.event.ActionEvent evt) {

try {

barcelonaRi.openClose(2, true);

} catch (RemoteException ex) {

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

}

}

private void buttonBarcelonaCloseRunway2ActionPerformed(java.awt.event.ActionEvent evt) {

try {

barcelonaRi.openClose(1, false);

} catch (RemoteException ex) {

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

}

}

private void buttonBarcelonaOpenRunway2ActionPerformed(java.awt.event.ActionEvent evt) {

try {

barcelonaRi.openClose(1, true);

} catch (RemoteException ex) {

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

}

}

private void buttonBarcelonaCloseRunway4ActionPerformed(java.awt.event.ActionEvent evt) {

try {

barcelonaRi.openClose(3, false);

} catch (RemoteException ex) {

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

}

}

private void buttonBarcelonaOpenRunway4ActionPerformed(java.awt.event.ActionEvent evt) {

try {

barcelonaRi.openClose(3, true);

} catch (RemoteException ex) {

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

}

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

public void setMadridHangar(int hangar) {

this.madridHangar.setText(Integer.toString(hangar));

}

public void setMadridPassengers(AtomicInteger passengers) {

this.madridPassengers.setText(passengers.toString());

}

public void setMadridMaintenance(int passengers) {

this.madridMaintenance.setText(Integer.toString(passengers));

}

public void setMadridParking(int passengers) {

this.madridParking.setText(Integer.toString(passengers));

}

public void setMadridTaxiArea(int passengers) {

this.madridTaxiArea.setText(Integer.toString(passengers));

}

public void setAirwayMadridBarcelona(String arway) {

this.AirwayMadridBarcelona.setText(arway);

}

public void setBarcelonaHangar(int hangar) {

this.barcelonaHangar.setText(Integer.toString(hangar));

}

public void setBarcelonaPassengers(AtomicInteger passengers) {

this.barcelonaPassengers.setText(passengers.toString());

}

public void setBarcelonaMaintenance(int passengers) {

this.barcelonaMaintenance.setText(Integer.toString(passengers));

}

public void setBarcelonaParking(int passengers) {

this.barcelonaParking.setText(Integer.toString(passengers));

}

public void setBarcelonaTaxiArea(int passengers) {

this.barcelonaTaxiArea.setText(Integer.toString(passengers));

}

public void setAirwayBarcelonaMadrid(String arway) {

this.AirwayBarcelonaMadrid.setText(arway);

}

public void set() throws RemoteException {

while (true) {

setMadridHangar(madridRi.numHangar());

setMadridPassengers(madridRi.getPassengers());

setMadridMaintenance(madridRi.numMaintenance());

setMadridParking(madridRi.numParking());

setMadridTaxiArea(madridRi.numTaxiArea());

setBarcelonaHangar(barcelonaRi.numHangar());

setBarcelonaPassengers(barcelonaRi.getPassengers());

setBarcelonaMaintenance(barcelonaRi.numMaintenance());

setBarcelonaParking(barcelonaRi.numParking());

setBarcelonaTaxiArea(barcelonaRi.numTaxiArea());

setAirwayMadridBarcelona(madridRi.showMadBarAirway());

setAirwayBarcelonaMadrid(madridRi.showBarMadAirway());

}

}

public static void main(String args[]) throws NotBoundException, MalformedURLException, RemoteException {

/\* Create and display the form \*/

java.awt.EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

new DistributedProgrammingGUI().setVisible(true);

}

});

}

// Variables declaration - do not modify

private javax.swing.JTextField AirwayBarcelonaMadrid;

private javax.swing.JTextField AirwayMadridBarcelona;

private javax.swing.JTextField barcelonaHangar;

private javax.swing.JTextField barcelonaMaintenance;

private javax.swing.JTextField barcelonaParking;

private javax.swing.JTextField barcelonaPassengers;

private javax.swing.JTextField barcelonaTaxiArea;

private javax.swing.JButton buttonBarcelonaCloseRunway1;

private javax.swing.JButton buttonBarcelonaCloseRunway2;

private javax.swing.JButton buttonBarcelonaCloseRunway3;

private javax.swing.JButton buttonBarcelonaCloseRunway4;

private javax.swing.JButton buttonBarcelonaOpenRunway1;

private javax.swing.JButton buttonBarcelonaOpenRunway2;

private javax.swing.JButton buttonBarcelonaOpenRunway3;

private javax.swing.JButton buttonBarcelonaOpenRunway4;

private javax.swing.JButton buttonMadridCloseRunway1;

private javax.swing.JButton buttonMadridCloseRunway2;

private javax.swing.JButton buttonMadridCloseRunway3;

private javax.swing.JButton buttonMadridCloseRunway4;

private javax.swing.JButton buttonMadridOpenRunway1;

private javax.swing.JButton buttonMadridOpenRunway2;

private javax.swing.JButton buttonMadridOpenRunway3;

private javax.swing.JButton buttonMadridOpenRunway4;

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;

private javax.swing.JSeparator jSeparator1;

private javax.swing.JSeparator jSeparator2;

private javax.swing.JTextField madridHangar;

private javax.swing.JTextField madridMaintenance;

private javax.swing.JTextField madridParking;

private javax.swing.JTextField madridPassengers;

private javax.swing.JTextField madridTaxiArea;

// End of variables declaration

}

## Gateway:

package com.mycompany.catl;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class Gateway {

private boolean close = false;

private Lock lock = new ReentrantLock();

private Condition stop = lock.newCondition();

public void look() {

try {

lock.lock();

while (close) {

try {

stop.await();

} catch (InterruptedException ie) {

}

}

} finally {

lock.unlock();

}

}

public void open() {

try {

lock.lock();

close = false;

stop.signalAll();

} finally {

lock.unlock();

}

}

public void close() {

try {

lock.lock();

close = true;

} finally {

lock.unlock();

}

}

}

## GraphicalInterface:

package com.mycompany.catl;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class GraphicalInterface extends javax.swing.JFrame {

/\*\*

\* Creates new form GraphicalInterface

\*/

public GraphicalInterface() {

initComponents();

}

/\*\*

\* 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() {

jPanel1 = new javax.swing.JPanel();

jLabel1 = new javax.swing.JLabel();

pauseButton = new javax.swing.JButton();

resumeButton = new javax.swing.JButton();

jLabel3 = new javax.swing.JLabel();

jLabel4 = new javax.swing.JLabel();

jLabel5 = new javax.swing.JLabel();

jLabel6 = 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();

jLabel12 = new javax.swing.JLabel();

jLabel13 = new javax.swing.JLabel();

jLabel14 = new javax.swing.JLabel();

jLabel15 = new javax.swing.JLabel();

jLabel16 = new javax.swing.JLabel();

jLabel31 = new javax.swing.JLabel();

jLabel32 = new javax.swing.JLabel();

jLabel33 = new javax.swing.JLabel();

madridMaintenance = new javax.swing.JTextField();

madridHangar = new javax.swing.JTextField();

madridParking = new javax.swing.JTextField();

madridGate1 = new javax.swing.JTextField();

madridGate4 = new javax.swing.JTextField();

madridGate2 = new javax.swing.JTextField();

madridGate5 = new javax.swing.JTextField();

madridGate3 = new javax.swing.JTextField();

madridGate6 = new javax.swing.JTextField();

madridTaxiArea = new javax.swing.JTextField();

madridRunway1 = new javax.swing.JTextField();

madridRunway2 = new javax.swing.JTextField();

madridRunway3 = new javax.swing.JTextField();

madridRunway4 = new javax.swing.JTextField();

jSeparator1 = new javax.swing.JSeparator();

madridBusTownAirport = new javax.swing.JTextField();

madridBusAirportTown = new javax.swing.JTextField();

madridPassengers = new javax.swing.JTextField();

jLabel34 = new javax.swing.JLabel();

jLabel35 = new javax.swing.JLabel();

barcelonaBusTownAirport = new javax.swing.JTextField();

jLabel36 = new javax.swing.JLabel();

barcelonaBusAirportTown = new javax.swing.JTextField();

jLabel37 = new javax.swing.JLabel();

barcelonaPassengers = new javax.swing.JTextField();

jLabel38 = new javax.swing.JLabel();

barcelonaHangar = new javax.swing.JTextField();

jLabel39 = new javax.swing.JLabel();

barcelonaMaintenance = new javax.swing.JTextField();

jLabel40 = new javax.swing.JLabel();

barcelonaParking = new javax.swing.JTextField();

jLabel41 = new javax.swing.JLabel();

barcelonaGate1 = new javax.swing.JTextField();

jLabel42 = new javax.swing.JLabel();

barcelonaGate2 = new javax.swing.JTextField();

barcelonaGate4 = new javax.swing.JTextField();

jLabel43 = new javax.swing.JLabel();

jLabel44 = new javax.swing.JLabel();

barcelonaGate5 = new javax.swing.JTextField();

jLabel45 = new javax.swing.JLabel();

barcelonaGate6 = new javax.swing.JTextField();

barcelonaGate3 = new javax.swing.JTextField();

jLabel46 = new javax.swing.JLabel();

jLabel47 = new javax.swing.JLabel();

barcelonaTaxiArea = new javax.swing.JTextField();

barcelonaRunway1 = new javax.swing.JTextField();

jLabel48 = new javax.swing.JLabel();

jLabel49 = new javax.swing.JLabel();

barcelonaRunway2 = new javax.swing.JTextField();

barcelonaRunway4 = new javax.swing.JTextField();

jLabel50 = new javax.swing.JLabel();

jLabel51 = new javax.swing.JLabel();

barcelonaRunway3 = new javax.swing.JTextField();

jLabel52 = new javax.swing.JLabel();

airwayMadridBarcelona = new javax.swing.JTextField();

jLabel53 = new javax.swing.JLabel();

airwayBarcelonaMadrid = new javax.swing.JTextField();

filler1 = new javax.swing.Box.Filler(new java.awt.Dimension(0, 0), new java.awt.Dimension(0, 0), new java.awt.Dimension(0, 32767));

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

setBackground(new java.awt.Color(255, 255, 255));

jPanel1.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

jLabel1.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N

jLabel1.setText("Madrid Airport");

jPanel1.add(jLabel1, new org.netbeans.lib.awtextra.AbsoluteConstraints(370, 40, -1, -1));

pauseButton.setText("Pause");

pauseButton.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

pauseButtonActionPerformed(evt);

}

});

jPanel1.add(pauseButton, new org.netbeans.lib.awtextra.AbsoluteConstraints(680, 20, 76, 17));

resumeButton.setText("Resume");

resumeButton.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

resumeButtonActionPerformed(evt);

}

});

jPanel1.add(resumeButton, new org.netbeans.lib.awtextra.AbsoluteConstraints(770, 20, -1, 17));

jLabel3.setText("Hangar:");

jPanel1.add(jLabel3, new org.netbeans.lib.awtextra.AbsoluteConstraints(170, 160, -1, -1));

jLabel4.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);

jLabel4.setText("Maintenance: ");

jPanel1.add(jLabel4, new org.netbeans.lib.awtextra.AbsoluteConstraints(140, 190, -1, -1));

jLabel5.setText("Parking Area:");

jPanel1.add(jLabel5, new org.netbeans.lib.awtextra.AbsoluteConstraints(140, 220, -1, -1));

jLabel6.setText("Gate 1:");

jPanel1.add(jLabel6, new org.netbeans.lib.awtextra.AbsoluteConstraints(170, 260, -1, -1));

jLabel7.setText("Gate 2:");

jPanel1.add(jLabel7, new org.netbeans.lib.awtextra.AbsoluteConstraints(170, 290, -1, -1));

jLabel8.setText("Gate 3:");

jPanel1.add(jLabel8, new org.netbeans.lib.awtextra.AbsoluteConstraints(170, 330, -1, -1));

jLabel9.setText("Gate 5:");

jPanel1.add(jLabel9, new org.netbeans.lib.awtextra.AbsoluteConstraints(480, 290, -1, -1));

jLabel10.setText("Gate 4:");

jPanel1.add(jLabel10, new org.netbeans.lib.awtextra.AbsoluteConstraints(480, 260, -1, -1));

jLabel11.setText("Gate 6:");

jPanel1.add(jLabel11, new org.netbeans.lib.awtextra.AbsoluteConstraints(480, 330, 44, -1));

jLabel12.setText("Taxi area:");

jPanel1.add(jLabel12, new org.netbeans.lib.awtextra.AbsoluteConstraints(160, 360, -1, -1));

jLabel13.setText("Runway 2:");

jPanel1.add(jLabel13, new org.netbeans.lib.awtextra.AbsoluteConstraints(160, 420, -1, -1));

jLabel14.setText("Runway 1:");

jPanel1.add(jLabel14, new org.netbeans.lib.awtextra.AbsoluteConstraints(160, 390, -1, -1));

jLabel15.setText("Runway 3:");

jPanel1.add(jLabel15, new org.netbeans.lib.awtextra.AbsoluteConstraints(470, 390, -1, -1));

jLabel16.setText("Runway 4:");

jPanel1.add(jLabel16, new org.netbeans.lib.awtextra.AbsoluteConstraints(470, 420, -1, -1));

jLabel31.setText("Number airport passengers:");

jPanel1.add(jLabel31, new org.netbeans.lib.awtextra.AbsoluteConstraints(140, 130, -1, -1));

jLabel32.setText("Transfers Downtown:");

jPanel1.add(jLabel32, new org.netbeans.lib.awtextra.AbsoluteConstraints(430, 90, -1, -1));

jLabel33.setText("Transfers Airport:");

jPanel1.add(jLabel33, new org.netbeans.lib.awtextra.AbsoluteConstraints(140, 90, -1, -1));

madridMaintenance.setEditable(false);

madridMaintenance.setBackground(new java.awt.Color(255, 255, 255));

madridMaintenance.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

madridMaintenanceActionPerformed(evt);

}

});

jPanel1.add(madridMaintenance, new org.netbeans.lib.awtextra.AbsoluteConstraints(220, 190, 494, -1));

madridHangar.setEditable(false);

madridHangar.setBackground(new java.awt.Color(255, 255, 255));

madridHangar.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

madridHangarActionPerformed(evt);

}

});

jPanel1.add(madridHangar, new org.netbeans.lib.awtextra.AbsoluteConstraints(220, 160, 494, -1));

madridParking.setEditable(false);

madridParking.setBackground(new java.awt.Color(255, 255, 255));

madridParking.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

madridParkingActionPerformed(evt);

}

});

jPanel1.add(madridParking, new org.netbeans.lib.awtextra.AbsoluteConstraints(220, 220, 496, -1));

madridGate1.setEditable(false);

madridGate1.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(madridGate1, new org.netbeans.lib.awtextra.AbsoluteConstraints(220, 260, 183, -1));

madridGate4.setEditable(false);

madridGate4.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(madridGate4, new org.netbeans.lib.awtextra.AbsoluteConstraints(534, 260, 183, -1));

madridGate2.setEditable(false);

madridGate2.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(madridGate2, new org.netbeans.lib.awtextra.AbsoluteConstraints(220, 290, 183, -1));

madridGate5.setEditable(false);

madridGate5.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(madridGate5, new org.netbeans.lib.awtextra.AbsoluteConstraints(534, 290, 183, -1));

madridGate3.setEditable(false);

madridGate3.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(madridGate3, new org.netbeans.lib.awtextra.AbsoluteConstraints(220, 330, 183, -1));

madridGate6.setEditable(false);

madridGate6.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(madridGate6, new org.netbeans.lib.awtextra.AbsoluteConstraints(534, 330, 183, -1));

madridTaxiArea.setEditable(false);

madridTaxiArea.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(madridTaxiArea, new org.netbeans.lib.awtextra.AbsoluteConstraints(220, 360, 496, -1));

madridRunway1.setEditable(false);

madridRunway1.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(madridRunway1, new org.netbeans.lib.awtextra.AbsoluteConstraints(220, 390, 183, -1));

madridRunway2.setEditable(false);

madridRunway2.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(madridRunway2, new org.netbeans.lib.awtextra.AbsoluteConstraints(220, 420, 183, -1));

madridRunway3.setEditable(false);

madridRunway3.setBackground(new java.awt.Color(255, 255, 255));

madridRunway3.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

madridRunway3ActionPerformed(evt);

}

});

jPanel1.add(madridRunway3, new org.netbeans.lib.awtextra.AbsoluteConstraints(534, 390, 183, -1));

madridRunway4.setEditable(false);

madridRunway4.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(madridRunway4, new org.netbeans.lib.awtextra.AbsoluteConstraints(534, 420, 183, -1));

jSeparator1.setForeground(new java.awt.Color(255, 255, 255));

jPanel1.add(jSeparator1, new org.netbeans.lib.awtextra.AbsoluteConstraints(140, 40, 795, -1));

madridBusTownAirport.setEditable(false);

madridBusTownAirport.setBackground(new java.awt.Color(255, 255, 255));

madridBusTownAirport.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

madridBusTownAirportActionPerformed(evt);

}

});

jPanel1.add(madridBusTownAirport, new org.netbeans.lib.awtextra.AbsoluteConstraints(260, 90, 160, -1));

madridBusAirportTown.setEditable(false);

madridBusAirportTown.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(madridBusAirportTown, new org.netbeans.lib.awtextra.AbsoluteConstraints(560, 90, 160, -1));

madridPassengers.setEditable(false);

madridPassengers.setBackground(new java.awt.Color(255, 255, 255));

madridPassengers.setText("0");

madridPassengers.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

madridPassengersActionPerformed(evt);

}

});

jPanel1.add(madridPassengers, new org.netbeans.lib.awtextra.AbsoluteConstraints(340, 130, 379, -1));

jLabel34.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N

jLabel34.setText("Barcelona Airport");

jPanel1.add(jLabel34, new org.netbeans.lib.awtextra.AbsoluteConstraints(1030, 40, -1, -1));

jLabel35.setText("Transfers Airport:");

jPanel1.add(jLabel35, new org.netbeans.lib.awtextra.AbsoluteConstraints(760, 90, -1, -1));

barcelonaBusTownAirport.setEditable(false);

barcelonaBusTownAirport.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(barcelonaBusTownAirport, new org.netbeans.lib.awtextra.AbsoluteConstraints(870, 90, 160, -1));

jLabel36.setText("Transfers Downtown:");

jPanel1.add(jLabel36, new org.netbeans.lib.awtextra.AbsoluteConstraints(1070, 90, -1, -1));

barcelonaBusAirportTown.setEditable(false);

barcelonaBusAirportTown.setBackground(new java.awt.Color(255, 255, 255));

barcelonaBusAirportTown.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

barcelonaBusAirportTownActionPerformed(evt);

}

});

jPanel1.add(barcelonaBusAirportTown, new org.netbeans.lib.awtextra.AbsoluteConstraints(1200, 90, 160, -1));

jLabel37.setText("Number airport passengers:");

jPanel1.add(jLabel37, new org.netbeans.lib.awtextra.AbsoluteConstraints(760, 130, -1, -1));

barcelonaPassengers.setEditable(false);

barcelonaPassengers.setBackground(new java.awt.Color(255, 255, 255));

barcelonaPassengers.setText("0");

jPanel1.add(barcelonaPassengers, new org.netbeans.lib.awtextra.AbsoluteConstraints(960, 130, 401, -1));

jLabel38.setText("Hangar:");

jPanel1.add(jLabel38, new org.netbeans.lib.awtextra.AbsoluteConstraints(818, 160, -1, -1));

barcelonaHangar.setEditable(false);

barcelonaHangar.setBackground(new java.awt.Color(255, 255, 255));

barcelonaHangar.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

barcelonaHangarActionPerformed(evt);

}

});

jPanel1.add(barcelonaHangar, new org.netbeans.lib.awtextra.AbsoluteConstraints(870, 160, 494, -1));

jLabel39.setText("Maintenance: ");

jPanel1.add(jLabel39, new org.netbeans.lib.awtextra.AbsoluteConstraints(790, 180, -1, 40));

barcelonaMaintenance.setEditable(false);

barcelonaMaintenance.setBackground(new java.awt.Color(255, 255, 255));

barcelonaMaintenance.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

barcelonaMaintenanceActionPerformed(evt);

}

});

jPanel1.add(barcelonaMaintenance, new org.netbeans.lib.awtextra.AbsoluteConstraints(870, 190, 494, -1));

jLabel40.setText("Parking Area:");

jPanel1.add(jLabel40, new org.netbeans.lib.awtextra.AbsoluteConstraints(790, 230, -1, -1));

barcelonaParking.setEditable(false);

barcelonaParking.setBackground(new java.awt.Color(255, 255, 255));

barcelonaParking.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

barcelonaParkingActionPerformed(evt);

}

});

jPanel1.add(barcelonaParking, new org.netbeans.lib.awtextra.AbsoluteConstraints(870, 230, 494, -1));

jLabel41.setText("Gate 1:");

jPanel1.add(jLabel41, new org.netbeans.lib.awtextra.AbsoluteConstraints(820, 270, -1, -1));

barcelonaGate1.setEditable(false);

barcelonaGate1.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(barcelonaGate1, new org.netbeans.lib.awtextra.AbsoluteConstraints(870, 270, 183, -1));

jLabel42.setText("Gate 2:");

jPanel1.add(jLabel42, new org.netbeans.lib.awtextra.AbsoluteConstraints(820, 300, -1, -1));

barcelonaGate2.setEditable(false);

barcelonaGate2.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(barcelonaGate2, new org.netbeans.lib.awtextra.AbsoluteConstraints(870, 300, 183, -1));

barcelonaGate4.setEditable(false);

barcelonaGate4.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(barcelonaGate4, new org.netbeans.lib.awtextra.AbsoluteConstraints(1180, 270, 183, -1));

jLabel43.setText("Gate 4:");

jPanel1.add(jLabel43, new org.netbeans.lib.awtextra.AbsoluteConstraints(1130, 270, -1, -1));

jLabel44.setText("Gate 5:");

jPanel1.add(jLabel44, new org.netbeans.lib.awtextra.AbsoluteConstraints(1130, 300, -1, -1));

barcelonaGate5.setEditable(false);

barcelonaGate5.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(barcelonaGate5, new org.netbeans.lib.awtextra.AbsoluteConstraints(1180, 300, 183, -1));

jLabel45.setText("Gate 6:");

jPanel1.add(jLabel45, new org.netbeans.lib.awtextra.AbsoluteConstraints(1130, 330, -1, -1));

barcelonaGate6.setEditable(false);

barcelonaGate6.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(barcelonaGate6, new org.netbeans.lib.awtextra.AbsoluteConstraints(1180, 330, 183, -1));

barcelonaGate3.setEditable(false);

barcelonaGate3.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(barcelonaGate3, new org.netbeans.lib.awtextra.AbsoluteConstraints(870, 330, 183, -1));

jLabel46.setText("Gate 3:");

jPanel1.add(jLabel46, new org.netbeans.lib.awtextra.AbsoluteConstraints(820, 330, -1, -1));

jLabel47.setText("Taxi area:");

jPanel1.add(jLabel47, new org.netbeans.lib.awtextra.AbsoluteConstraints(810, 360, -1, -1));

barcelonaTaxiArea.setEditable(false);

barcelonaTaxiArea.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(barcelonaTaxiArea, new org.netbeans.lib.awtextra.AbsoluteConstraints(870, 360, 494, -1));

barcelonaRunway1.setEditable(false);

barcelonaRunway1.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(barcelonaRunway1, new org.netbeans.lib.awtextra.AbsoluteConstraints(870, 390, 183, -1));

jLabel48.setText("Runway 1: ");

jPanel1.add(jLabel48, new org.netbeans.lib.awtextra.AbsoluteConstraints(810, 390, -1, -1));

jLabel49.setText("Runway 2:");

jPanel1.add(jLabel49, new org.netbeans.lib.awtextra.AbsoluteConstraints(810, 420, -1, -1));

barcelonaRunway2.setEditable(false);

barcelonaRunway2.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(barcelonaRunway2, new org.netbeans.lib.awtextra.AbsoluteConstraints(870, 420, 183, -1));

barcelonaRunway4.setEditable(false);

barcelonaRunway4.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(barcelonaRunway4, new org.netbeans.lib.awtextra.AbsoluteConstraints(1180, 420, 183, -1));

jLabel50.setText("Runway 4:");

jPanel1.add(jLabel50, new org.netbeans.lib.awtextra.AbsoluteConstraints(1120, 420, -1, -1));

jLabel51.setText("Runway 3:");

jPanel1.add(jLabel51, new org.netbeans.lib.awtextra.AbsoluteConstraints(1120, 390, -1, -1));

barcelonaRunway3.setEditable(false);

barcelonaRunway3.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(barcelonaRunway3, new org.netbeans.lib.awtextra.AbsoluteConstraints(1180, 390, 183, -1));

jLabel52.setText("Airway Madrid-Barcelona:");

jPanel1.add(jLabel52, new org.netbeans.lib.awtextra.AbsoluteConstraints(160, 460, -1, -1));

airwayMadridBarcelona.setEditable(false);

airwayMadridBarcelona.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(airwayMadridBarcelona, new org.netbeans.lib.awtextra.AbsoluteConstraints(160, 480, 1206, -1));

jLabel53.setText("Airway Barcelona-Madrid:");

jPanel1.add(jLabel53, new org.netbeans.lib.awtextra.AbsoluteConstraints(160, 520, -1, -1));

airwayBarcelonaMadrid.setEditable(false);

airwayBarcelonaMadrid.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.add(airwayBarcelonaMadrid, new org.netbeans.lib.awtextra.AbsoluteConstraints(160, 540, 1206, -1));

jPanel1.add(filler1, new org.netbeans.lib.awtextra.AbsoluteConstraints(1282, 64, -1, -1));

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

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

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

.addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT\_SIZE, 1528, Short.MAX\_VALUE)

);

layout.setVerticalGroup(

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

.addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT\_SIZE, 604, Short.MAX\_VALUE)

);

pack();

}// </editor-fold>

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

if (!resumePressed) //if not pressed

{

resumePressed = true; //we change it to pressed

pausePressed = false;

gw.open(); //Open the gateway

}

}

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

if (!pausePressed) //if not pressed

{

resumePressed = false; //we change it to pressed

pausePressed = true;

gw.close(); //Close the gateway

}

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

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

// TODO add your handling code here:

}

private void madridRunway3ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

}

//All set methods for the graphical interface avoiding mutual exclusion when necessary

private Lock lockMadridPassengers = new ReentrantLock();

private Lock lockBarcelonaPassengers = new ReentrantLock();

private Lock lockMadridBusTownAirport = new ReentrantLock();

private Lock lockBarcelonaBusTownAirport = new ReentrantLock();

private Lock lockMadridBusAirportTown = new ReentrantLock();

private Lock lockBarcelonaBusAirportTown = new ReentrantLock();

private Lock lockMadridHangar = new ReentrantLock();

private Lock lockBarcelonaHangar = new ReentrantLock();

private Lock lockMadridParking = new ReentrantLock();

private Lock lockBarcelonaParking = new ReentrantLock();

private Lock lockMadridTaxiArea = new ReentrantLock();

private Lock lockBarcelonaTaxiArea = new ReentrantLock();

private Lock lockAirwayMadridBarcelona = new ReentrantLock();

private Lock lockAirwayBarcelonaMadrid = new ReentrantLock();

private Lock lockMadridMaintenanceHall = new ReentrantLock();

private Lock lockBarcelonaMaintenanceHall = new ReentrantLock();

private Gateway gw = new Gateway();

private boolean resumePressed = false;

private boolean pausePressed = false;

public Gateway getGw() {

return gw;

}

public void setMadridPassengers(int madridPassengers) {

lockMadridPassengers.lock();

try {

this.madridPassengers.setText(String.valueOf(madridPassengers));

} catch (Exception e) {

} finally {

lockMadridPassengers.unlock();

}

}

public void setBarcelonaPassengers(int barcelonaPassengers) {

lockBarcelonaPassengers.lock();

try {

this.barcelonaPassengers.setText(String.valueOf(barcelonaPassengers));

} catch (Exception e) {

} finally {

lockBarcelonaPassengers.unlock();

}

}

public void setMadridBusTownAirport(String bus) {

lockMadridBusTownAirport.lock();

try {

this.madridBusTownAirport.setText(bus);

} catch (Exception e) {

} finally {

lockMadridBusTownAirport.unlock();

}

}

public void setBarcelonaBusTownAirport(String bus) {

lockBarcelonaBusTownAirport.lock();

try {

this.barcelonaBusTownAirport.setText(bus);

} catch (Exception e) {

} finally {

lockBarcelonaBusTownAirport.unlock();

}

}

public void setMadridBusAirportTown(String bus) {

lockMadridBusAirportTown.lock();

try {

this.madridBusAirportTown.setText(bus);

} catch (Exception e) {

} finally {

lockMadridBusAirportTown.unlock();

}

}

public void setBarcelonaBusAirportTown(String bus) {

lockBarcelonaBusAirportTown.lock();

try {

this.barcelonaBusAirportTown.setText(bus);

} catch (Exception e) {

} finally {

lockBarcelonaBusAirportTown.unlock();

}

}

public void setMadridHangar(String hangar) {

lockMadridHangar.lock();

try {

this.madridHangar.setText(hangar);

} catch (Exception e) {

} finally {

lockMadridHangar.unlock();

}

}

public void setBarcelonaHangar(String hangar) {

lockBarcelonaHangar.lock();

try {

this.barcelonaHangar.setText(hangar);

} catch (Exception e) {

// Aquí podrías manejar la excepción si lo deseas

} finally {

lockBarcelonaHangar.unlock();

}

}

public void setMadridParking(String parking) {

lockMadridParking.lock();

try {

this.madridParking.setText(parking);

} catch (Exception e) {

// Aquí podrías manejar la excepción si lo deseas

} finally {

lockMadridParking.unlock();

}

}

public void setBarcelonaParking(String parking) {

lockBarcelonaParking.lock();

try {

this.barcelonaParking.setText(parking);

} catch (Exception e) {

// Aquí podrías manejar la excepción si lo deseas

} finally {

lockBarcelonaParking.unlock();

}

}

public void setMadridGate1(String gate) {

this.madridGate1.setText(gate);

}

public void setMadridGate2(String gate) {

this.madridGate2.setText(gate);

}

public void setMadridGate3(String gate) {

this.madridGate3.setText(gate);

}

public void setMadridGate4(String gate) {

this.madridGate4.setText(gate);

}

public void setMadridGate5(String gate) {

this.madridGate5.setText(gate);

}

public void setMadridGate6(String gate) {

this.madridGate6.setText(gate);

}

public void setBarcelonaGate1(String gate) {

this.barcelonaGate1.setText(gate);

}

public void setBarcelonaGate2(String gate) {

this.barcelonaGate2.setText(gate);

}

public void setBarcelonaGate3(String gate) {

this.barcelonaGate3.setText(gate);

}

public void setBarcelonaGate4(String gate) {

this.barcelonaGate4.setText(gate);

}

public void setBarcelonaGate5(String gate) {

this.barcelonaGate5.setText(gate);

}

public void setBarcelonaGate6(String gate) {

this.barcelonaGate6.setText(gate);

}

public void setMadridTaxiArea(String taxiArea) {

lockMadridTaxiArea.lock();

try {

this.madridTaxiArea.setText(taxiArea);

} catch (Exception e) {

} finally {

lockMadridTaxiArea.unlock();

}

}

public void setMadridRunway1(String runway) {

this.madridRunway1.setText(runway);

}

public void setMadridRunway2(String runway) {

this.madridRunway2.setText(runway);

}

public void setMadridRunway3(String runway) {

this.madridRunway3.setText(runway);

}

public void setMadridRunway4(String runway) {

this.madridRunway4.setText(runway);

}

public void setBarcelonaTaxiArea(String taxiArea) {

lockBarcelonaTaxiArea.lock();

try {

this.barcelonaTaxiArea.setText(taxiArea);

} catch (Exception e) {

} finally {

lockBarcelonaTaxiArea.unlock();

}

}

public void setBarcelonaRunway1(String runway) {

this.barcelonaRunway1.setText(runway);

}

public void setBarcelonaRunway2(String runway) {

this.barcelonaRunway2.setText(runway);

}

public void setBarcelonaRunway3(String runway) {

this.barcelonaRunway3.setText(runway);

}

public void setBarcelonaRunway4(String runway) {

this.barcelonaRunway4.setText(runway);

}

public void setAirwayMadridBarcelona(String airway) {

lockAirwayMadridBarcelona.lock();

try {

this.airwayMadridBarcelona.setText(airway);

} catch (Exception e) {

} finally {

lockAirwayMadridBarcelona.unlock();

}

}

public void setAirwayBarcelonaMadrid(String airway) {

lockAirwayBarcelonaMadrid.lock();

try {

this.airwayBarcelonaMadrid.setText(airway);

} catch (Exception e) {

} finally {

lockAirwayBarcelonaMadrid.unlock();

}

}

public void setMadridMaintenanceHall(String maintenanceHall) {

lockMadridMaintenanceHall.lock();

try {

this.madridMaintenance.setText(maintenanceHall);

} catch (Exception e) {

} finally {

lockMadridMaintenanceHall.unlock();

}

}

public void setBarcelonaMaintenanceHall(String maintenanceHall) {

lockBarcelonaMaintenanceHall.lock();

try {

this.barcelonaMaintenance.setText(maintenanceHall);

} catch (Exception e) {

} finally {

lockBarcelonaMaintenanceHall.unlock();

}

}

/\*\*

\* @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(GraphicalInterface.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

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

} catch (IllegalAccessException ex) {

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

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

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

}

//</editor-fold>

/\* Create and display the form \*/

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new GraphicalInterface().setVisible(true);

}

});

}

// Variables declaration - do not modify

private javax.swing.JTextField airwayBarcelonaMadrid;

private javax.swing.JTextField airwayMadridBarcelona;

private javax.swing.JTextField barcelonaBusAirportTown;

private javax.swing.JTextField barcelonaBusTownAirport;

private javax.swing.JTextField barcelonaGate1;

private javax.swing.JTextField barcelonaGate2;

private javax.swing.JTextField barcelonaGate3;

private javax.swing.JTextField barcelonaGate4;

private javax.swing.JTextField barcelonaGate5;

private javax.swing.JTextField barcelonaGate6;

private javax.swing.JTextField barcelonaHangar;

private javax.swing.JTextField barcelonaMaintenance;

private javax.swing.JTextField barcelonaParking;

private javax.swing.JTextField barcelonaPassengers;

private javax.swing.JTextField barcelonaRunway1;

private javax.swing.JTextField barcelonaRunway2;

private javax.swing.JTextField barcelonaRunway3;

private javax.swing.JTextField barcelonaRunway4;

private javax.swing.JTextField barcelonaTaxiArea;

private javax.swing.Box.Filler filler1;

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 jLabel3;

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 jLabel39;

private javax.swing.JLabel jLabel4;

private javax.swing.JLabel jLabel40;

private javax.swing.JLabel jLabel41;

private javax.swing.JLabel jLabel42;

private javax.swing.JLabel jLabel43;

private javax.swing.JLabel jLabel44;

private javax.swing.JLabel jLabel45;

private javax.swing.JLabel jLabel46;

private javax.swing.JLabel jLabel47;

private javax.swing.JLabel jLabel48;

private javax.swing.JLabel jLabel49;

private javax.swing.JLabel jLabel5;

private javax.swing.JLabel jLabel50;

private javax.swing.JLabel jLabel51;

private javax.swing.JLabel jLabel52;

private javax.swing.JLabel jLabel53;

private javax.swing.JLabel jLabel6;

private javax.swing.JLabel jLabel7;

private javax.swing.JLabel jLabel8;

private javax.swing.JLabel jLabel9;

private javax.swing.JPanel jPanel1;

private javax.swing.JSeparator jSeparator1;

private javax.swing.JTextField madridBusAirportTown;

private javax.swing.JTextField madridBusTownAirport;

private javax.swing.JTextField madridGate1;

private javax.swing.JTextField madridGate2;

private javax.swing.JTextField madridGate3;

private javax.swing.JTextField madridGate4;

private javax.swing.JTextField madridGate5;

private javax.swing.JTextField madridGate6;

private javax.swing.JTextField madridHangar;

private javax.swing.JTextField madridMaintenance;

private javax.swing.JTextField madridParking;

private javax.swing.JTextField madridPassengers;

private javax.swing.JTextField madridRunway1;

private javax.swing.JTextField madridRunway2;

private javax.swing.JTextField madridRunway3;

private javax.swing.JTextField madridRunway4;

private javax.swing.JTextField madridTaxiArea;

private javax.swing.JButton pauseButton;

private javax.swing.JButton resumeButton;

// End of variables declaration

}

## Hangar:

package com.mycompany.catl;

import java.util.Iterator;

import java.util.Queue;

import java.util.concurrent.ConcurrentLinkedQueue;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class Hangar {

Queue<Airplane> airplanes = new ConcurrentLinkedQueue<Airplane>();

private Lock hangarLock = new ReentrantLock();

int position;

//Constructor

public Hangar() {

position = -1;

}

public Queue<Airplane> getAirplanes() {

return airplanes;

}

public void setAirplanes(Queue<Airplane> airplanes) {

this.airplanes = airplanes;

}

/\*\*

\* It adds an airplane into the hangar and returns its position in the list

\*

\* @param airplane the new airplane

\*/

public void addAirplane(Airplane airplane) {

this.hangarLock.lock();

try {

this.airplanes.offer(airplane); //Adds the airplane to the airplanes list of hangar

} finally {

this.hangarLock.unlock();

}

}

/\* It takes an airplane from the hangar

\*

\* @param puesto the position of the airplane ib the list

\* @return the airplane in the position from the list

\*/

public Airplane releaseAirplane(Airplane airplane) {

hangarLock.lock();

try {

this.airplanes.remove(airplane);

} finally {

hangarLock.unlock();

}

return airplane;

}

/\*\*

\* It transform the hangar array into a String

\*

\* @return a string of the airplanes in the hangar

\*/

public String toString() {

StringBuilder allPlanes = new StringBuilder();

Iterator<Airplane> newIterator = airplanes.iterator(); // Create a new iterator

while (newIterator.hasNext()) {

String currPlane = newIterator.next().getIdentifier();

allPlanes.append(currPlane.concat(", "));

}

return allPlanes.toString();

}

}

## Log:

package com.mycompany.catl;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.time.LocalDateTime;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class Log {

private Lock textLock = new ReentrantLock();

private String airportEvolution = "src\\main\\java\\com\\mycompany\\catl\\airportEvolution.txt";

private FileWriter writer;

private BufferedWriter writerBuffer = null;

public Log() {

try {

// Create a FileWriter with the file name, using true to allow appending to the end of the file

writer = new FileWriter(airportEvolution, true);

// Create a BufferedWriter to write in the file

writerBuffer = new BufferedWriter(writer);

} catch (IOException e) {

System.err.println("Error upon opening the file " + e.getMessage());

}

}

/\*\*

\* Writes update in the log file, maintaining mutual exclusion

\*

\* @param text the text that you want to write

\*/

public void write(String text) {

textLock.lock(); //Lock the log for writing

try {

LocalDateTime date = LocalDateTime.now(); //Take the date to the second

writerBuffer.write(date + ": " + text); //Write date: text

writerBuffer.newLine();

} catch (Exception e) {

} finally {

textLock.unlock(); //Unlock the log

}

}

/\*\*

\* Flushes and closes the buffer and the FileWriter

\*/

public void close() {

try {

if (writerBuffer != null) {

writerBuffer.flush();

writerBuffer.close();

}

if (writer != null) {

writer.close();

}

} catch (IOException e) {

System.err.println("Error upon opening the file " + e.getMessage());

}

}

}

## MaintenanceHall:

package com.mycompany.catl;

import java.util.Iterator;

import java.util.concurrent.ArrayBlockingQueue;

import java.util.concurrent.BlockingQueue;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class MaintenanceHall {

BlockingQueue airplanes;

Lock enterDoor;

Condition first;

public MaintenanceHall() {

airplanes = new ArrayBlockingQueue<Airplane>(20); //Capacity of 20

enterDoor = new ReentrantLock();

first = enterDoor.newCondition();

}

public void enterHallDoor(Airplane airplane, Airport airport, Log log) throws InterruptedException {

enterDoor.lock(); //Only one plane can enter through the door at a time

try {

while (!airplane.getAirport(airport).getParking().getAirplanesForMaintenance().peek().equals(airplane)) {

log.write("Airplane " + airplane.getIdentifier() + " waiting to enter hall of airport " + airport);

first.await(); //Calling airplane waits until it is first in the list

}

airplanes.put(airplane.getAirport(airport).getParking().releaseAirplaneForMaintenance(airplane)); //When first, add plane to maintenance hall from parking

log.write("Airplane " + airplane.getIdentifier() + " entered hall of airport " + airport);

first.signalAll(); //Signal to all threads after removing calling thread to check if they're first in the list

Thread.sleep(1000);

} finally {

enterDoor.unlock();

}

}

public Airplane releaseHall(Airplane airplane) throws InterruptedException {

enterDoor.lock(); //Only one plane can exit through the door at a time

try {

if (airplanes.remove(airplane)) {

System.out.println("Airplane " + airplane.getIdentifier() + " successfully exiting the maintenance hall.");

Thread.sleep(1000);

return airplane;

} else {

System.err.println("Error removing airplane " + airplane.getIdentifier() + " from the maintenance hall.");

return null;

}

} finally {

enterDoor.unlock();

}

}

@Override

public String toString() {

StringBuilder allPlanes = new StringBuilder();

Iterator<Airplane> newIterator = airplanes.iterator(); // Create a new iterator

while (newIterator.hasNext()) {

String currPlane = newIterator.next().getIdentifier();

allPlanes.append(currPlane.concat(" "));

}

return allPlanes.toString();

}

public BlockingQueue getAirplanes() {

return airplanes;

}

}

## Parking:

package com.mycompany.catl;

import java.util.Queue;

import java.util.Iterator;

import java.util.concurrent.BlockingQueue;

import java.util.concurrent.LinkedBlockingQueue;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class Parking {

private BlockingQueue<Airplane> airplanesForBoarding;

private BlockingQueue<Airplane> airplanesForMaintenance;

public Parking() {

airplanesForBoarding = new LinkedBlockingQueue<Airplane>();

airplanesForMaintenance = new LinkedBlockingQueue<Airplane>();

}

/\*\*

\* It adds an airplane into the parking

\*

\* @param airplane the new airplane

\* @throws java.lang.InterruptedException

\*/

public void addAirplane(Airplane airplane) throws InterruptedException {

if (!airplane.getLanding()) { //If airplane boarding, send to the boarding queue

this.airplanesForBoarding.offer(airplane); //Add the airplane at the end of the boarding list

} else {

this.airplanesForMaintenance.offer(airplane); //Add the airplane at the end of the maintenance list

}

System.out.println("Airplane " + airplane.getIdentifier() + " was added to parking.");

}

/\*\*

\* Takes last airplane from the parking area

\*

\* @param airplane the airplane that we want to take out from the parking

\* @return the airplane that we take out

\* @throws java.lang.InterruptedException

\*/

public Airplane releaseAirplaneForBoarding(Airplane airplane) throws InterruptedException {

Airplane removedAirplane;

// Airplane is at the front of the queue and can proceed

removedAirplane = airplanesForBoarding.poll(); // Remove the airplane from the queue

System.out.println("Airplane " + removedAirplane.getIdentifier() + " was removed from parking.");

System.out.println("Current airplanes in parking are: " + toString());

return removedAirplane;

}

public Airplane releaseAirplaneForMaintenance(Airplane airplane) throws InterruptedException {

System.out.println("Airplane " + airplane.getIdentifier() + " waiting. First airplane waiting for maintenance is " + airplanesForMaintenance.peek().getIdentifier());

Airplane removedAirplane;

// Airplane is at the front of the queue and can proceed

removedAirplane = airplanesForMaintenance.poll(); // Remove the airplane from the queue

System.out.println("Airplane " + removedAirplane.getIdentifier() + " was removed from parking.");

System.out.println("Current airplanes in parking are: " + toString());

return removedAirplane;

}

@Override

public String toString() {

StringBuilder allPlanes = new StringBuilder();

Iterator<Airplane> boardingIterator = airplanesForBoarding.iterator(); // Create a new iterator

Iterator<Airplane> maintenanceIterator = airplanesForMaintenance.iterator(); // Create a new iterator

while (boardingIterator.hasNext()) {

Airplane currPlane = boardingIterator.next(); // Assuming the object type is Airplane

String identifier = currPlane.getIdentifier();

int passengers = currPlane.getPassengers(); // Get the number of passengersString currPlane = boardingIterator.next().getIdentifier();

allPlanes.append(identifier + "(" + passengers + ")" + ", ");

}

while (maintenanceIterator.hasNext()) {

String currPlane = maintenanceIterator.next().getIdentifier();

allPlanes.append(currPlane.concat(" "));

}

return allPlanes.toString();

}

public Queue<Airplane> getAirplanesForBoarding() {

return airplanesForBoarding;

}

public Queue<Airplane> getAirplanesForMaintenance() {

return airplanesForMaintenance;

}

}

## RemoteInterface:

package com.mycompany.catl;

import java.rmi.Remote;

import java.rmi.RemoteException;

import java.util.concurrent.atomic.AtomicInteger;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public interface RemoteInterface extends Remote {

public int numHangar() throws RemoteException;

public AtomicInteger getPassengers() throws RemoteException;

public int numMaintenance() throws RemoteException;

public int numParking() throws RemoteException;

public int numTaxiArea() throws RemoteException;

public String showMadBarAirway() throws RemoteException;

public String showBarMadAirway() throws RemoteException;

public void openClose(int runway, boolean opCl) throws RemoteException;

}

## Runways:

package com.mycompany.catl;

import java.util.Arrays;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class Runways {

private Airplane[] runways;

private boolean[] openCloseList;

private Lock runwayLock;

private Condition full;

public Runways() {

runways = new Airplane[4]; //Runway can fit 4 airplanes

runwayLock = new ReentrantLock();

full = runwayLock.newCondition();

openCloseList = new boolean[]{true, true, true, true};

}

public int enterRunway(Airplane airplane) throws InterruptedException {

runwayLock.lock();

try {

int runway = -1;

while (!Arrays.asList(runways).contains(null)) {

full.await(); //Thread waits until an opening

}

for (int i = 0; i < 4; i++) {

if (runways[i] == null && openCloseList[i]) {

runways[i] = airplane; //Calling airplane fills opening when signaled

runway = i;

break;

}

}

return runway;

} finally {

runwayLock.unlock();

}

}

public Airplane releaseRunway(Airplane airplane) throws InterruptedException {

runwayLock.lock();

try {

int index = -1;

index = Arrays.asList(runways).indexOf(airplane);

if (index != -1) {

runways[index] = null;

full.signalAll(); //Signal to all threads that there is an opening

return airplane;

} else {

return null;

}

} finally {

runwayLock.unlock();

}

}

/\* It opens or closes an runway

\*

\*@param runway is the runway that will be open or closed

\*@param opCl if it is true it opens the runway, if it is false it closes the runway

\*/

public void openClose(int runway, boolean opCl) {

this.openCloseList[runway] = opCl;

}

}

## Server:

package com.mycompany.catl;

import java.rmi.Naming;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class Server {

public static void main(String[] args) throws RemoteException {

Log log = new Log();

//Creation of Madrid and Barcelona airports

Airway Mad\_Bar = new Airway("Mad\_Bar");

Airway Bar\_Mad = new Airway("Bar\_Mad");

BoardingGates boardingGatesMadrid = new BoardingGates();

Runways runwaysMadrid = new Runways();

BoardingGates boardingGatesBarcelona = new BoardingGates();

Runways runwaysBarcelona = new Runways(); //

//Madrid class

Hangar hangarMadrid = new Hangar();

MaintenanceHall maintenanceHallMadrid = new MaintenanceHall();

Parking parkingMadrid = new Parking();

TaxiArea taxiMadrid = new TaxiArea();

Airport madrid = new Airport(hangarMadrid, Mad\_Bar, Bar\_Mad, taxiMadrid, parkingMadrid, maintenanceHallMadrid, boardingGatesMadrid, runwaysMadrid, "MAD");

//Barcelona class

Hangar hangarBarcelona = new Hangar();

MaintenanceHall maintenanceHallBarcelona = new MaintenanceHall();

Parking parkingBarcelona = new Parking();

TaxiArea taxiBarcelona = new TaxiArea();

Airport barcelona = new Airport(hangarBarcelona, Mad\_Bar, Bar\_Mad, taxiBarcelona, parkingBarcelona, maintenanceHallBarcelona, boardingGatesBarcelona, runwaysBarcelona, "BAC");

GraphicalInterface gf = new GraphicalInterface();

gf.setVisible(true);

try {

Registry reg = LocateRegistry.createRegistry(1099);

Naming.rebind("//localhost/madrid", madrid);

Naming.rebind("//localhost/barcelona", barcelona);

} catch (Exception e) {

e.printStackTrace();

}

AirplaneCreator airplaneCreator = new AirplaneCreator(log, madrid, barcelona, gf);

BusCreator busCreator = new BusCreator(log, madrid, barcelona, gf);

//If the program is finished or interrupted, the log is automatically closed to avoid the loss of information

Runtime.getRuntime().addShutdownHook(new Thread(() -> {

airplaneCreator.interrupt();

busCreator.interrupt();

log.close();

}));

airplaneCreator.start();

busCreator.start();

}

}

## TaxiArea:

package com.mycompany.catl;

import java.util.concurrent.BlockingQueue;

import java.util.concurrent.LinkedBlockingQueue;

/\*\*

\*

\* @author Paean Luby

\* @author Nicolás Rodríguez Sánchez

\*/

public class TaxiArea {

private BlockingQueue<Airplane> airplanes;

public TaxiArea() {

airplanes = new LinkedBlockingQueue<Airplane>();

}

public void enterTaxiArea(Airplane airplane) {

airplanes.offer(airplane); //Add an airplane to the front of the queue

}

public BlockingQueue<Airplane> getAirplanes() {

return airplanes;

}

public Airplane releaseAirplane(Airplane airplane) { //Remove the calling airplane

this.airplanes.remove(airplane);

return airplane;

}

@Override

public String toString() {

StringBuilder allPlanes = new StringBuilder();

for (Airplane currPlane : airplanes) {

String identifier = currPlane.getIdentifier();

int passengers = currPlane.getPassengers();

allPlanes.append(identifier + "(" + passengers + ")" + ", ");

}

return allPlanes.toString();

}

}