**Concurrent Programming (CSC2044) Assignment**

**Airport Traffic Control (20%)**

**Documentation by:**

**LEON KHO**

**(16080764)**

1. **Description of how you implemented your solution**

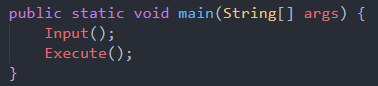
**Overview**

In this assignment, four classes were used to achieve my solution. Each specific class used has a general overview of its role within the system:

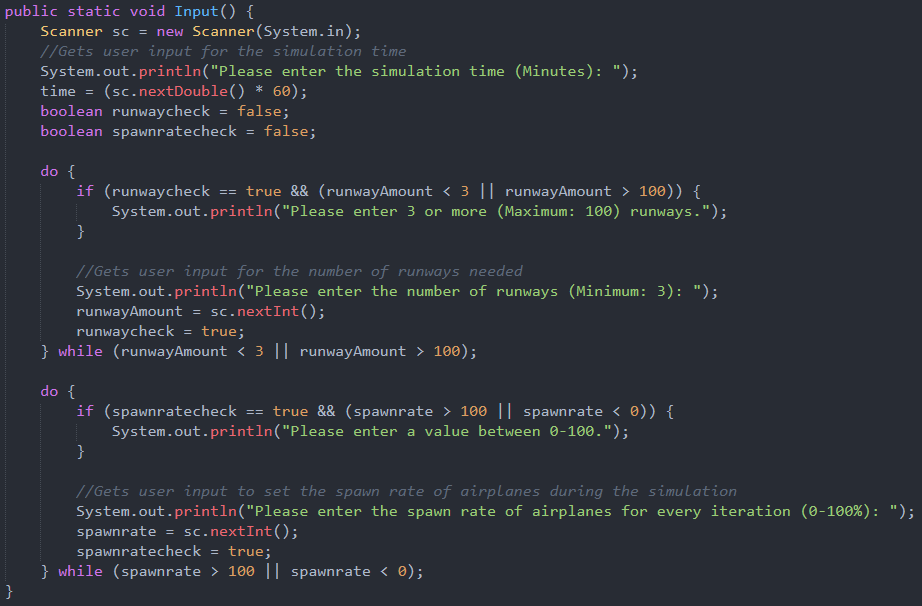
1. **Concurrent\_Assignment -** This class represents the main class of my system. It is primarily used for user inputs to configure the system to adhere with the extra functionalities that have been implemented.
2. **Runway -** The runway class is used to create the runways needed for the system. Each runway has its own unique name, status and properties required to service airplanes that operate on it.
3. **Airplane -** The airplane class is used to create airplanes that will be serviced by any available runway throughout the simulation. Each airplane has its own unique ID, model, duration, status and destination.
4. **AirplaneGenerator -** The AirplaneGenerator is responsible for both the user interface, as well as the creation of new airplanes and a logger mechanism. This is done in accordance to the user inputs provided in the main class.

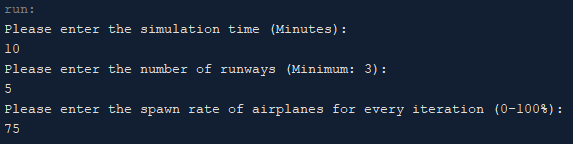
**Implementation**

Concurrent\_Assignment



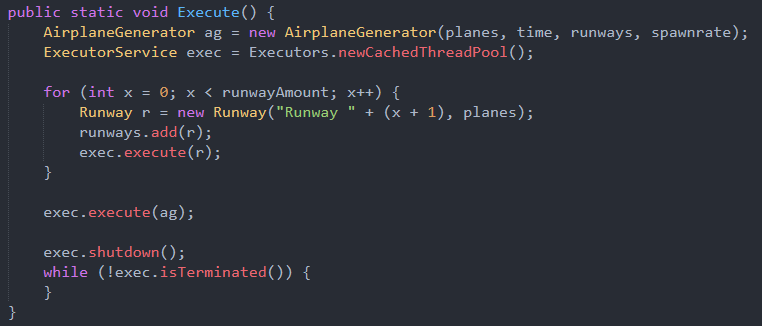
In the main class, two methods are implemented.





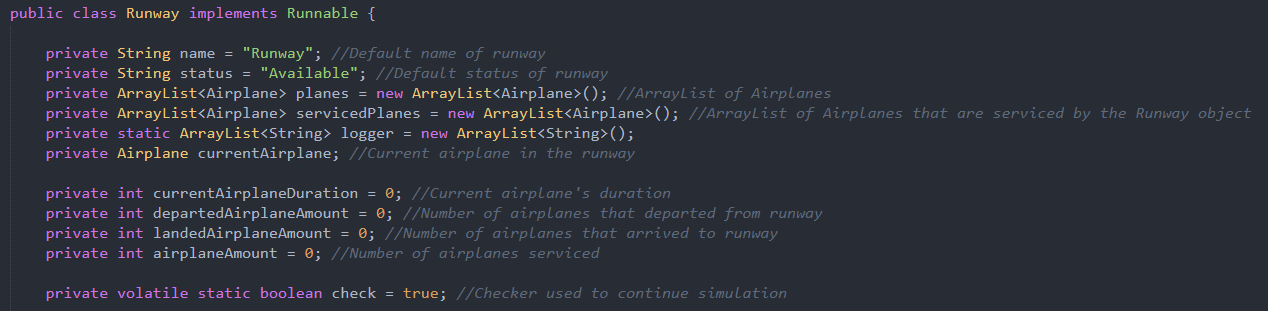
The **Input()** requests for user inputs which are required to configure the simulation system before it starts to run based on the extra functionalities that were implemented. Three inputs are required, namely:

1. Simulation time – The user inputs the simulation time, which reflects on how long the system will simulate the airport traffic control.
2. Number of runways – The user can also input the number of runways needed for the simulation. However, a minimum of three runways are required as stated in the assignment guidelines.
3. Spawn rate of airplanes – The user also has control of the possibility of an airplane being generated for each iteration of the simulation (per second). By inputting 100, airplanes will be generated every second.

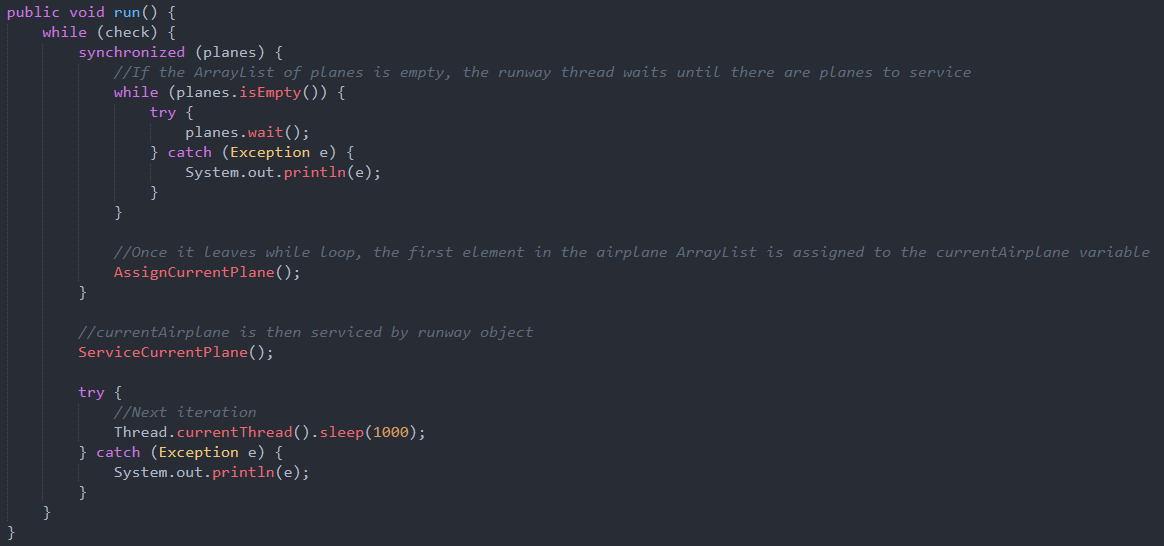


Once the **Input()** receives all of the necessary user inputs, it will then initiate the **Execute()** method. This is where the AirplaneGenerator and Runway objects are instantiated and passed onto threads to execute. A thread pool is used to ensure fairness among the threads.

Runway



The Runway class implements Runnable, and has a number of variables that are used in the simulation process. Each of these variables play a part in the changes that take place within the simulation. The comments beside each of these variables explain their particular roles.

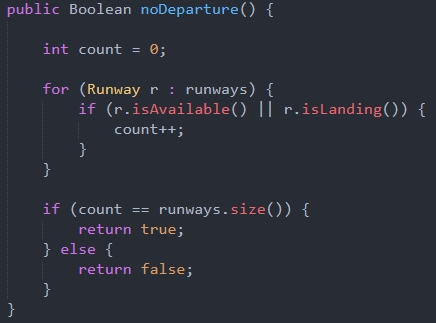


In the **run()** method, a while loop used and is determined by a volatile Boolean named check.

Once it enters the while loop, a synchronize block utilizes with the planes ArrayList is then entered.

If the planes ArrayList is empty, the thread waits until a generated airplane populates the planes ArrayList. When the thread leaves the while loop, it will then call the **AssignCurrentPlane()** method.

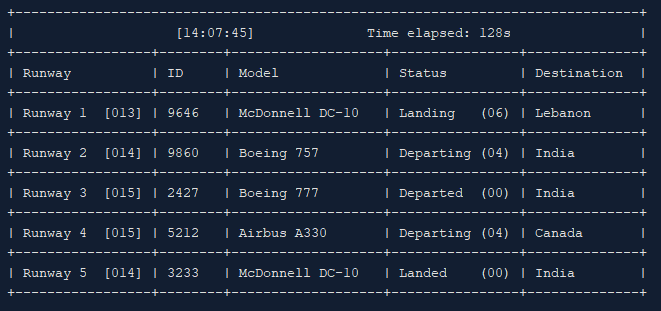




The **AssignCurrentPlane()** assigns the first plane in the ArrayList to the currentAirplane variable along with its duration. Once the runway object has the currentAirplane in its possession, it will either set the status of the currentAirplane to “Landing” or “Departing”, depending on the state of the currentAirplane which would initially be either “Arriving” or “Ready”. However, if the currentAirplane is ready, there is an if statement which checks if all other runways are available. If this condition is met, a random integer between 1 – 5 is generated and added on top of the initial 5 seconds.

Once the details of the first plane in the ArrayList is retrieved, it is then removed from the said ArrayList. A **notifyAll()** method is then used to notify the other threads.

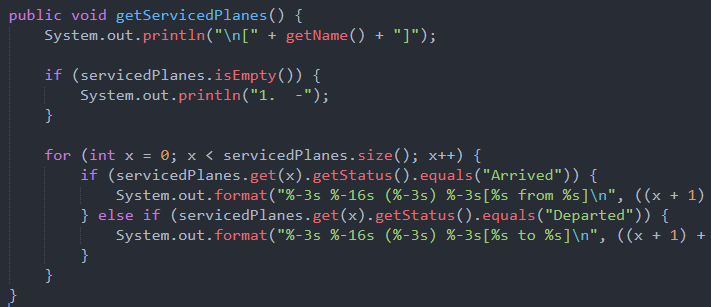




Upon exiting the synchronized block, the **ServiceCurrentPlane()** method is called. Now that the information of the currentAirplane in the particular runway is retrieved, the airplane is then serviced by the runway.

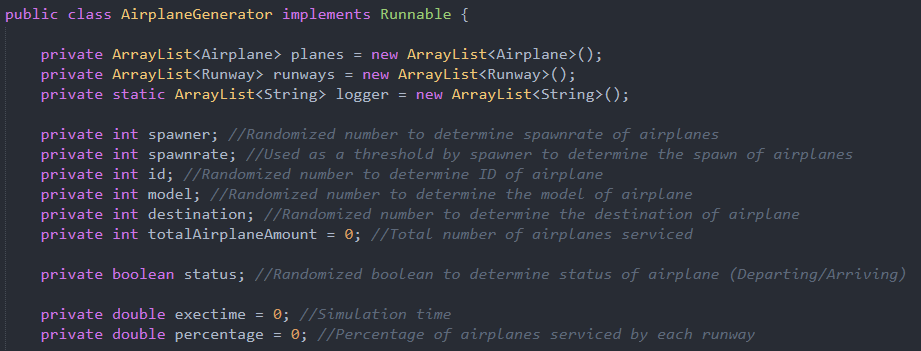
This is done by decrementing the duration of the currentAirplane on the runway using a for loop based on its initial duration. Once it is completed, an if statement is used to determine the status of the currentAirplane and whether the checker Boolean is true (whether the simulation is still running). If the condition is met, the status of the currentAirplane changes from either “Departing” or “Landing” to “Departed” or “Arrived” respectively.

The number of departed or landed airplanes, and the amount of airplanes serviced by that particular runway is then incremented. This is also followed by currentAirplane being added into an ArrayList that stores all the airplanes that have been serviced by the particular runway, named servicedPlanes.



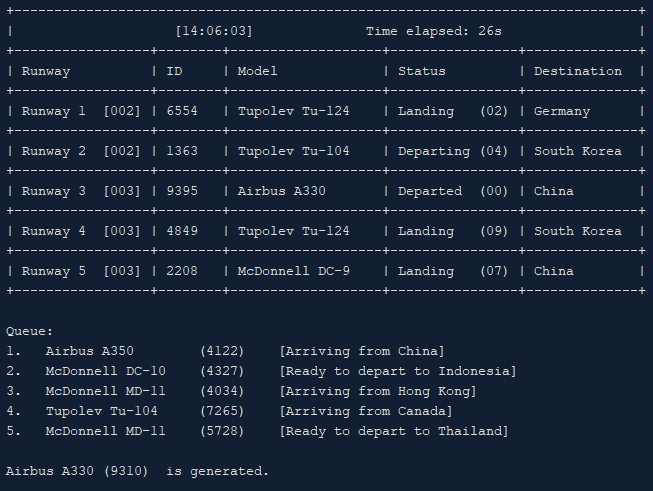
Another method used for extra functionalities that is found in the Runway class is the **getServicedPlanes()** method. This method is used at the end of the simulation to generate a list of serviced planes by that particular runway. The results will be shown later on in the documentation.

AirplaneGenerator



The AirplaneGenerator class also implements Runnable, and has a number of variables used to store the values needed to run the system as well. Each variable in the AirplaneGenerator class plays a role in creating unique aircrafts during the simulation.





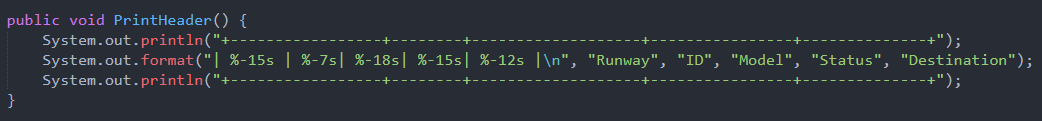
In the **run()** method of AirplaneGenerator, the user interface is generated depending on the simulation time (exectime). For each iteration, the current time and time elapsed is printed above the header of the UI, and is followed by the **PrintHeader()**. This is followed by the **PrintRunways()** method which displays the number of runways that are present in the system.





Each runway has its own counter which increments when an airplane is serviced by that particular runway.

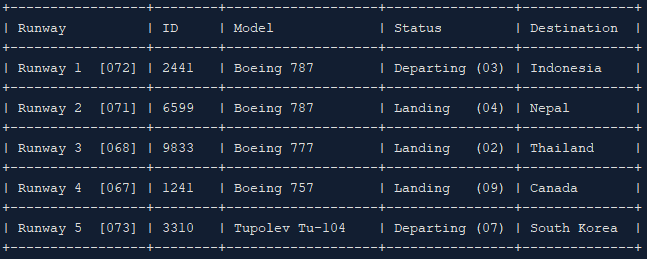
The system then enters a synchronized block that utilizes the planes ArrayList. The **Queue()** method then displays any airplanes that might be waiting when all runways are occupied. Next, is the **GenerateAirplane()** method which will be called, which is used to spawn unique aircrafts based on the spawn rate that the user has input in the beginning of the system. Once this is done, **notifyAll()** is called so that the rest of the threads will gain the lock back. After the simulation has ended, a for loop is used to terminate the threads that are used by the all the runways.





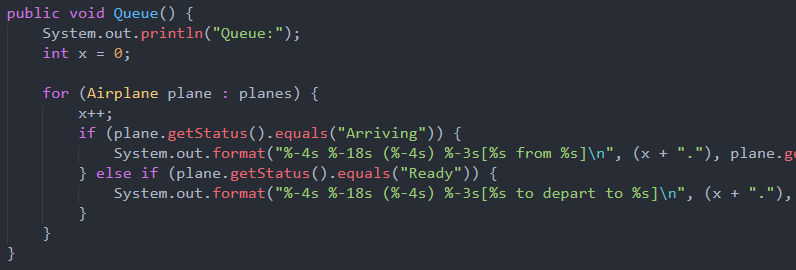
The **PrintHeader()** method prints out the headers which indicates Runway, ID, Model, Status and Destination respectively.

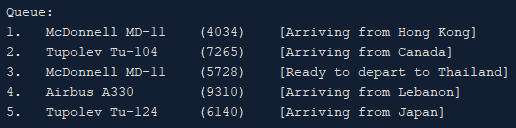




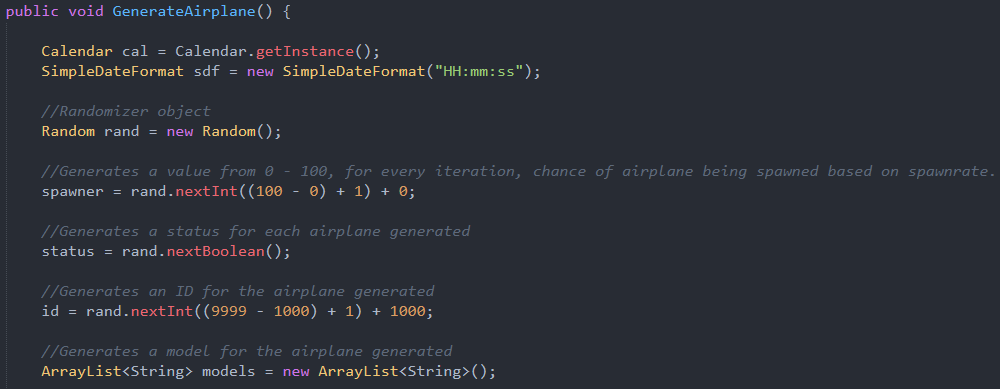
In the **PrintRunways()** method, runways are printed based on their statuses. There are five states a runway can be in, namely “Available”, “Landing”, “Departing”, “Landed”, “Departed”.

One implementation in the **PrintRunways()** worth noting is that if the runway has landed or departed an airplane, a logger ArrayList is used to store the timestamp, along with a statement when an airplane is landed/departed as needed by the assignment requirements.

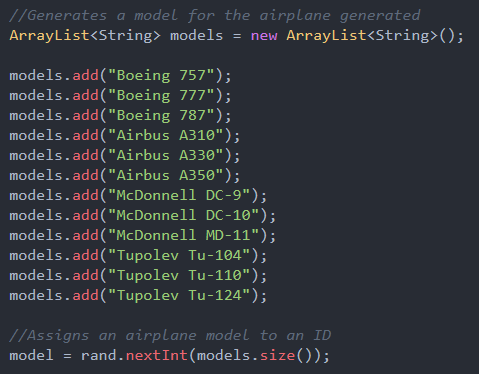




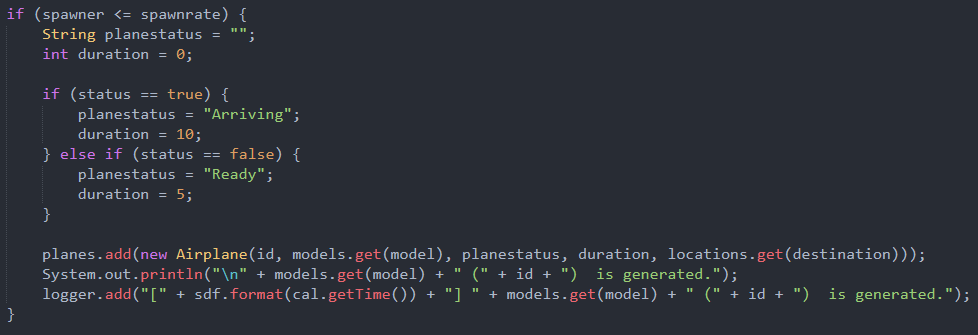
In the **Queue()** method, a for each loop is used to show the planes that are waiting when all runways are occupied. Each airplane prints a different statement based on the statuses of that particular airplane (“Arriving”/”Ready”).



In **GenerateAirplane()**, a Random object is used to generate the unique attributes of each airplane generated.



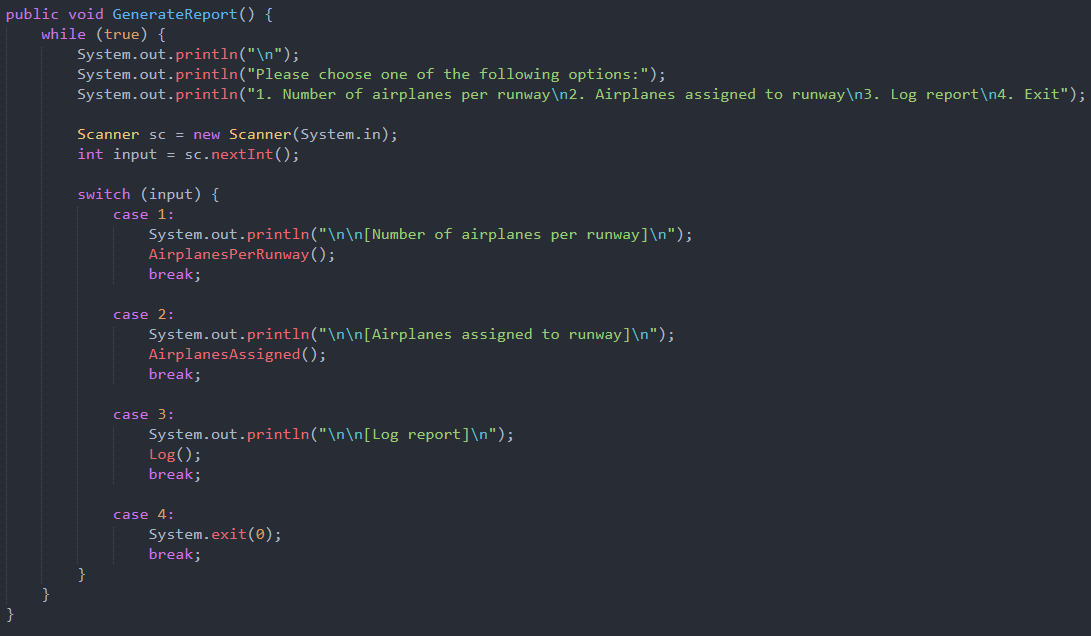
The randomized number is also used to determine the model that will be assigned to the new Airplane object.

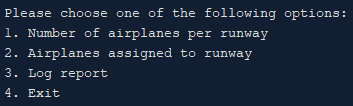


Based on the spawn rate which is entered by the user in the beginning of the system, it is used to determine whether an airplane can spawn based on the randomized spawner integer.

If the spawner value is less than or equals to the spawn rate, an airplane object is created. The status of the airplane can be either “Arriving” or “Ready”, determined by the status Boolean.

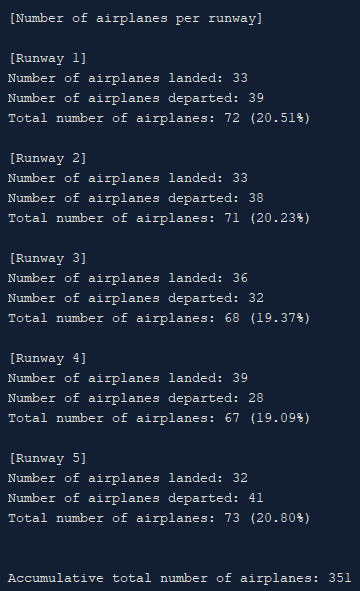
When an airplane is generated, the logger is also used to store the timestamp along with a statement indicating that the airplane has been generated.



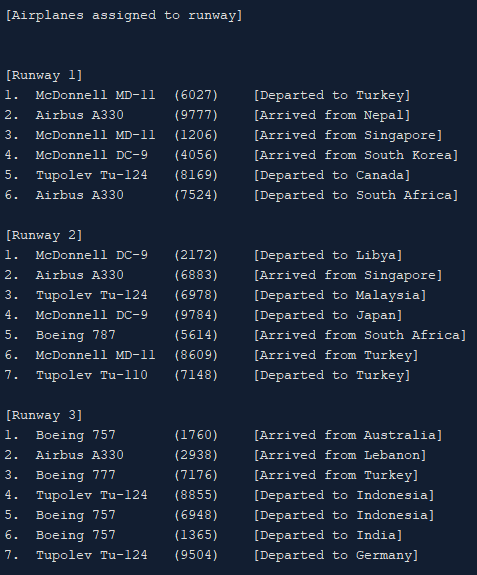


An extra functionality implemented is the **GenerateReport()** method, which is called at the end of the simulation. Essentially, it provides a menu which allows users to navigate to more extra implementations of the system, including number of airplanes per runway, airplanes assigned to runway, log report and an exit function.

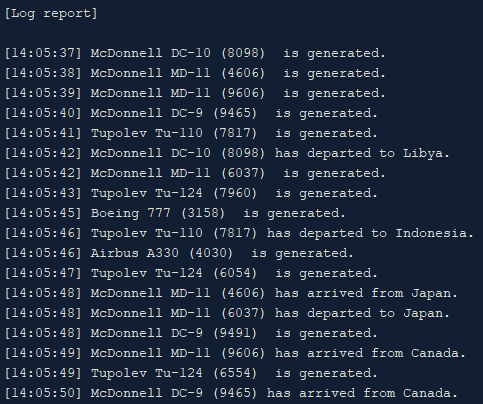




In the **AirplanesPerRunway()** method, the accumulative total of all airplanes serviced by all runways is calculated. The percentage of airplanes serviced by each runway is also calculated based on the number of airplanes serviced by that particular runway. The method then displays all information necessary, including the number of airplanes that have landed/departed in the runway, and its total. Once the for loop is run, the accumulative total of all airplanes is then displayed.

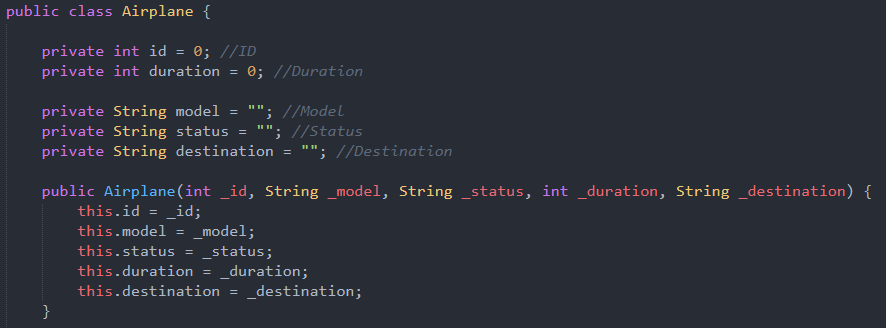


In the **AirplanesAssigned()** method, a for loop is run to call the **getServicedPlanes()** method by each runway, which in turn prints out all of the serviced runways.



The **Log()** method is similar to the previous method, where a for loop is run to get the size of the logger ArrayList, and prints out all logs generated.

Airplanes



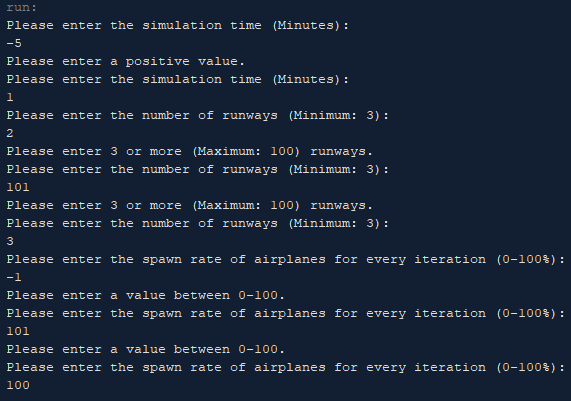
The Airplane class is used as instances to be serviced by the runways. It contains variables that store the values needed by the airplanes.

1. **Elements of the assignment that you were not able to complete.**

Based on the requirements stated in the assignment guidelines, all criteria have been met. Each of the solutions required have been covered in the first part of the documentation.

1. **List and description of any bugs in your program.**

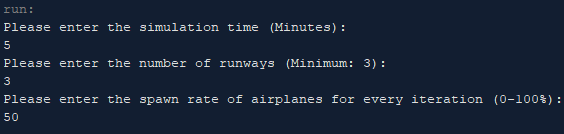
Extensive testing has been carried out with various inputs that may cause the system to malfunction. However, the system runs fine due to range of inputs that have been implemented in the **Input()** method of Concurrent\_Assignment class.



No bugs have been found throughout the testing phase of the system.

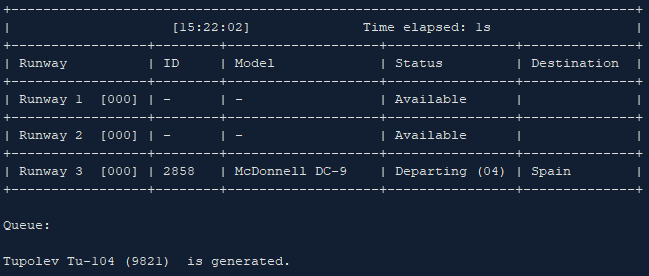
1. **Description of how you tested your program, along with sample output from your program.**

The system has been tested using multiple values that are within the range of input.

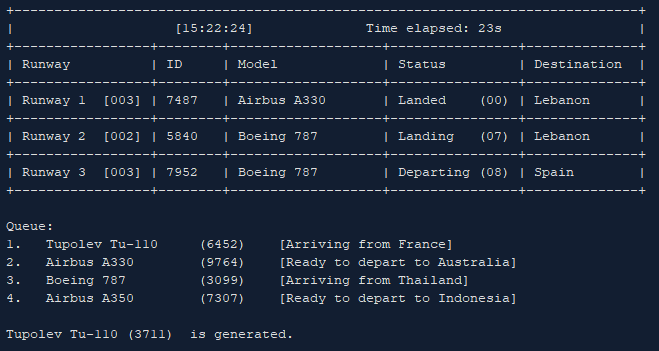


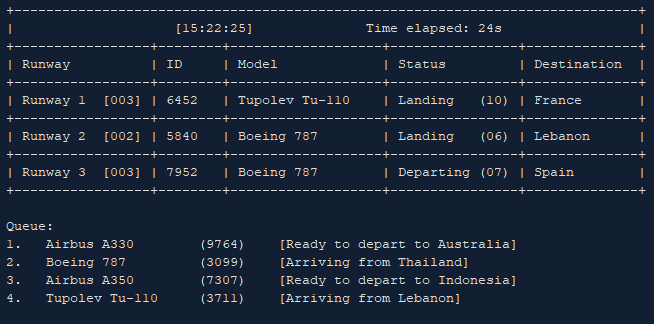
Sample outputs used are:

* Simulation time: 5
* Number of runways: 3
* Spawn rate: 50

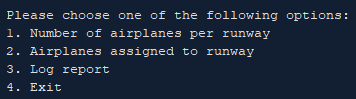


The system then begins to run until the end of the simulation time. Runways will then service generated airplanes.



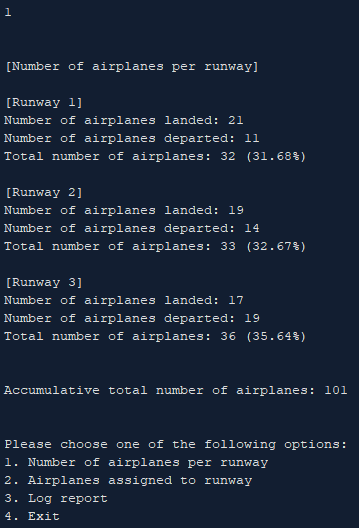


When all available runways are occupied, a queue list displays the number of awaiting airplanes. As seen in the two images above, Tupolev Tu-110 (6452) enters Runway 1 after Airbus 330 (7487) has landed.

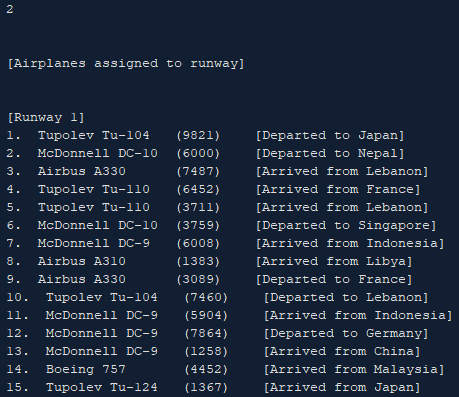


Once the simulation time has passed, an option menu is displayed to let users get an overview of statistics generated from the simulation.

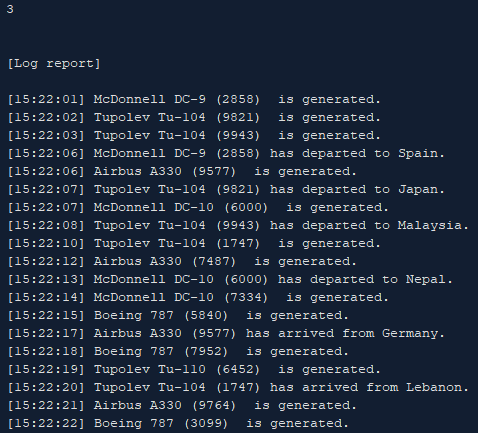
Input: 1 (Number of airplanes per runway)



Input: 2 (Airplanes assigned to runway)



Input: 3 (Log report)



Input: 4 (Exit)

