Benjamin Joncas, 002299246

CS321 – Final Project

April 25th, 2024

Air Traffic Control Simulation

Overview

For my final project in CS321, I decided to create a simulation of an air traffic control tower at an airport. The goal of my project was to implement design patterns and techniques learned throughout this course to coordinate a control tower, multiple airplanes, and multiple runways. The following is a brief description of the main components I created:

- **Aircraft:** Represents an airplane with 4 states (ground, takeoff, air, landing), has a perform method with appropriate procedures that simulate the time spent on each task, and the requests for a runway when needed. Aircraft is observable by the control tower.
- **Control Tower:** There is only one control tower, it holds the list of aircrafts and runways and knows the state of every aircraft.
- **Runway:** There are 3 runways available for use at the airport by the 5 aircrafts, aircrafts must request to use them. If the runway is not currently in use, access is given to the aircraft.
- **GUI:** An attempt at a visual representation of the simulation.

Design/implementation decisions

- **Singleton:** There is only one control tower, the singleton pattern ensures that there remains only one instance of this class and is not instantiated more than once throughout the simulation.
- **Template:** This pattern is implemented through the subclasses of aircraft, private aircraft and commercial aircraft, this gives the possibility of having different procedures or wait times for each type. My implementation simply states the type of flight.
- **Semaphore/Concurrency:** Given that there a finite number of runways and no two aircrafts are permitted to use the same runway at once, this pattern is used to

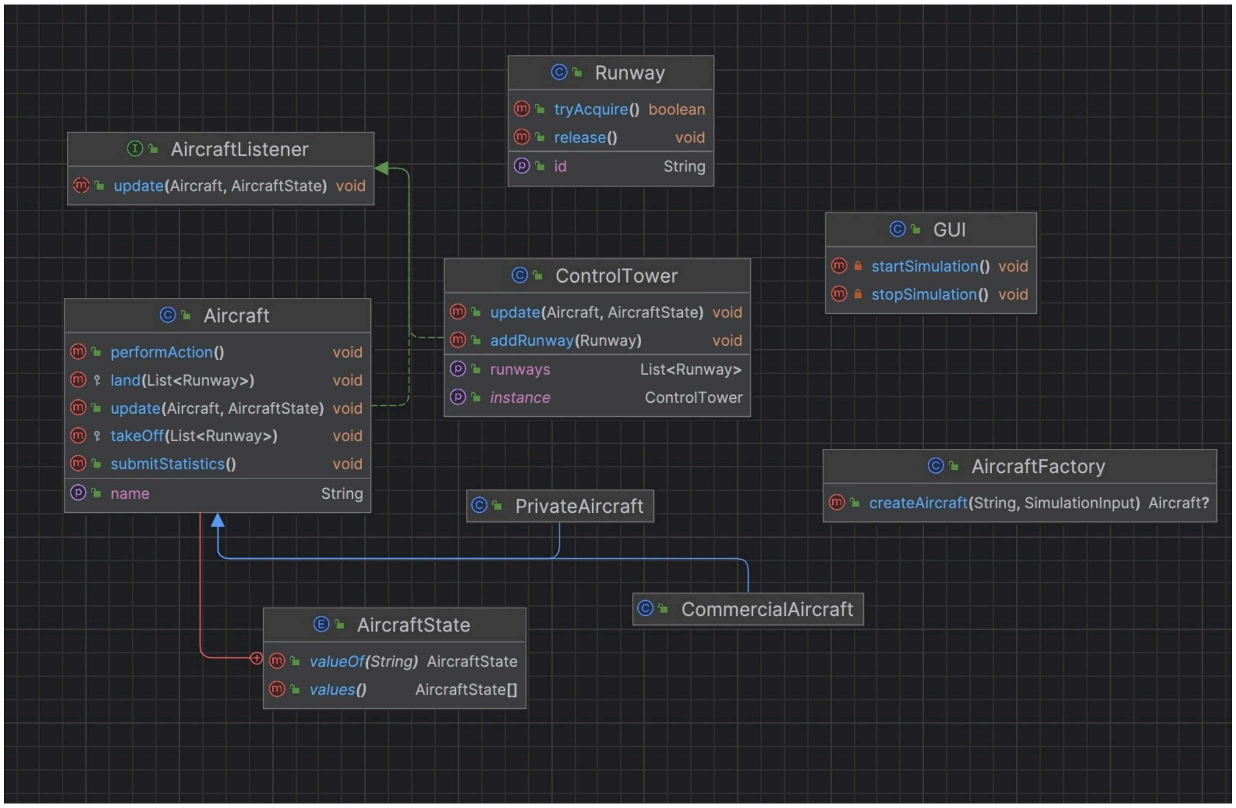
ensure that permission is only granted when appropriate. The runway class uses semaphores for each of its runways.

- **Observer:** I wanted the control tower to act as one does and to be an observer of the aircrafts it manages. Every time there is a change in the state of the aircraft, the tower is notified so that it can react appropriately. The interface aircraft listener is used to implement the pattern.
- **Factory:** The aircraft factory is implemented to have the flexibility of creating different types of aircrafts through the input and encapsulating object creation.

Logic Choices

The main choices I made in my implementation were centered around the use of semaphores to manage the shared resource that is the runways to ensure no conflicts between aircrafts and the state management using switch statements to handle each state of the aircrafts. Semaphores ensuring that only one aircraft can use a runway at a time is crucial for preventing race conditions in a multithreaded environment. Lists are also used to handle collections of runways for iterating over available runways.

UML


Runway
tryAcquire() boolean
release() void
id String
AircraftListener
update(Aircraft, AircraftState) void
GUI
startSimulation() void
stopSimulation() void
ControlTower
update(Aircraft, AircraftState) void
addRunway(Runway) void
runways List<Runway>
instance ControlTower
Aircraft
performAction() void
land(List<Runway>) void
update(Aircraft, AircraftState) void
takeOff(List<Runway>) void
submitStatistics() void
name String
AircraftFactory
createAircraft(String, SimulationInput) Aircraft?
PrivateAircraft
CommercialAircraft
AircraftState
valueOf(String) AircraftState
values() AircraftState[]