



Software Engineering
(CSE334s) Group Assignment 2

Spring 2024

Airlines System



Presented To
Dr. Islam Al Maddah

Contents

THE TEAM'S DETAILS.....	2
Team Members	2
Introduction	2
Class Diagram	2
Code Test	4
Test 1. List Aircraft Types at Airport	4
Test 2. Find Airports Supporting Boeing 757.....	4
Test 3. List Airports in France	5
Test 4. Track Aircraft and Flights.....	5
Appendix.....	6
Table Of Figures.....	6

THE TEAM'S DETAILS

Team Members

Team Member	ID
Mahmoud Abdelraouf Mahmoud	2001436
Eslam Mohamed Marzouk	2000252
Ahmed Khaled Abdelmaksod Ebrahim	2000218
Adham Khaled Abdelmaqsoud	2000066
Karim Ibrahim Saad Abd-Elrazek	2001118
Ahmed Mohammed Bakr Ahmed	2000037
Marwan Wael Mahmoud abbas	2001244
Somaya Ayman El-Sayed Ahmed	2000423
Mennatallah Amr Ali Abdel-Motaleb	2002111
Maya Ahmed Abdullah Ali	2002124

INTRODUCTION

This project represents an exploration into software engineering concepts through the design and implementation of an airline system using Java. As part of our software engineering coursework, we delved into the intricacies of object-oriented programming and system design to develop a robust and functional application that models various aspects of the airline industry. Leveraging Java's versatility and object-oriented paradigm, we constructed a system that encompasses entities such as airlines, countries, airports, fleets of aircraft, and aircraft types, each with their own attributes and relationships.

The implementation code for this project can be accessed at the following link: [GitHub Repository](#)

CLASS DIAGRAM

The class diagram (shown in *Figure 1*) for the airline system consists of several key entities, including Aircraft, AircraftType, Airline, Airport, City, Country, and Fleet. Each class encapsulates specific attributes and behaviors relevant to the airline domain. Aircraft and AircraftType represent

the several types of aircraft used by airlines, while Airline encapsulates information about individual airlines, including their fleets and landing arrangements. Airport and City represent geographical locations, with Airport containing information about supported aircraft types. Country acts as a container for cities and airlines operating within its territory. Finally, Fleet represents a specific flight route taken by an aircraft within a given week. The relationships between these classes reflect the associations and dependencies defined in the provided system requirements.

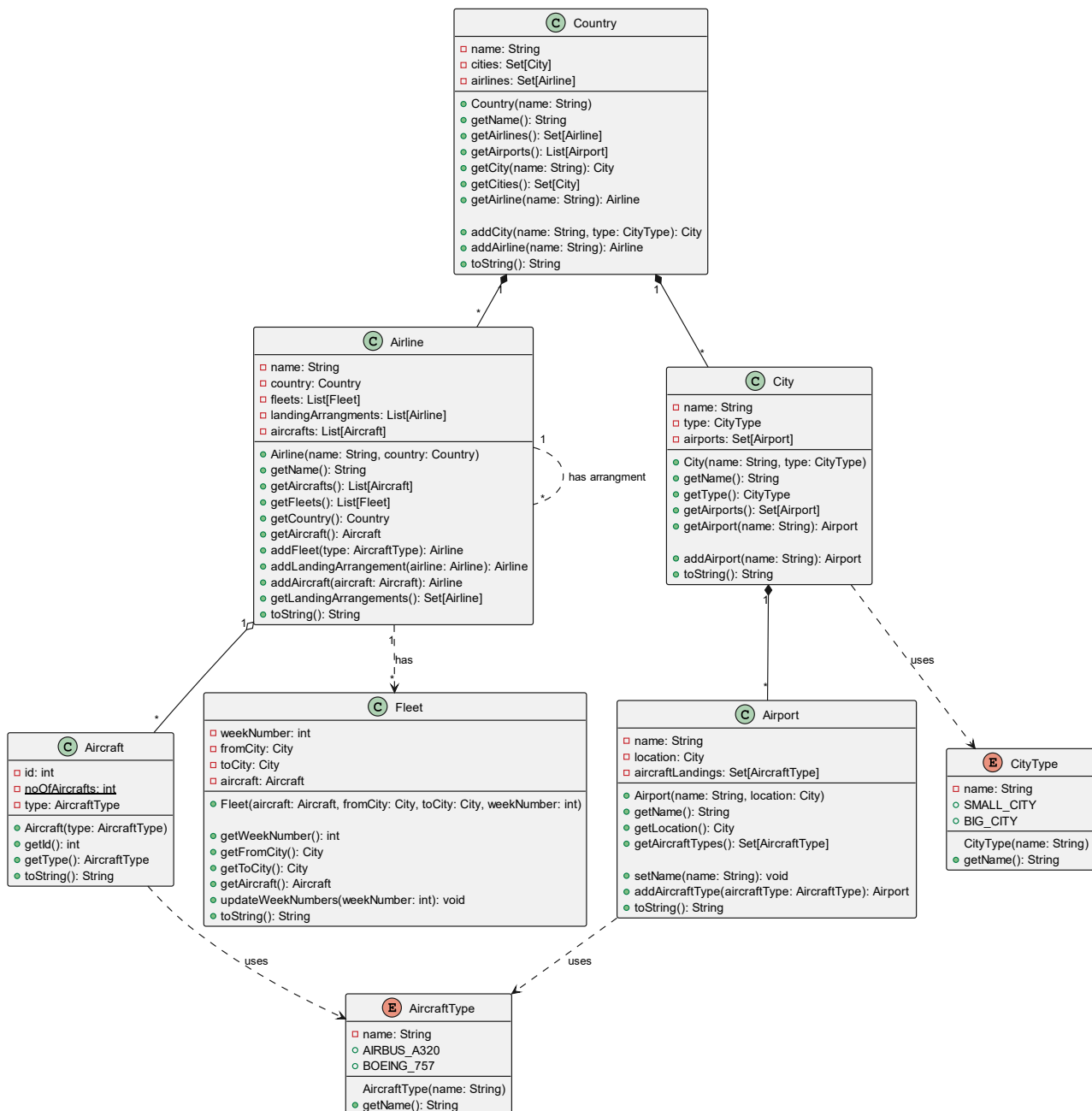


Figure 1: Class Diagram: Modelling an Airline System

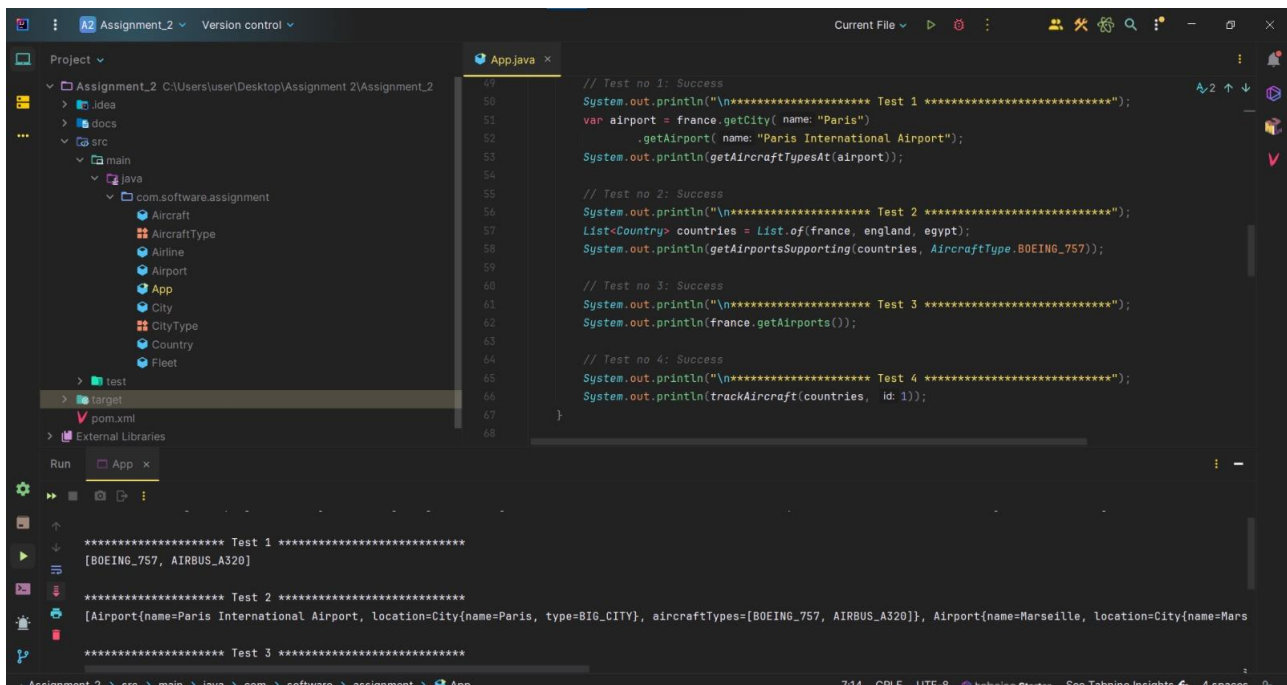
CODE TEST

Test 1. List Aircraft Types at Airport

This functionality retrieves an airport object based on its name and location, then accesses its associated set of supported aircraft types. By querying the airport's attributes, specifically the collection of aircraft types it can accommodate, this functionality provides a straightforward approach to listing all aircraft types that are permitted to land at a particular airport. This is achieved by invoking the **getAircraftTypes()** method of the Airport class, which returns a set containing the supported aircraft types. The result is a comprehensive list of aircraft types, facilitating informed decisions for airline operations and airport management. Refer to *Figure 2* for the test result.

Test 2. Find Airports Supporting Boeing 757

This functionality traverses through the list of countries, cities, and airports to identify all airports across multiple regions that can accommodate a Boeing 757 aircraft type. By iterating over the data structure representing geographical entities, it systematically checks each airport's list of supported aircraft types to determine if the Boeing 757 is present. This search is executed through the **getAirportsSupporting()** method, which takes a list of countries and the target aircraft type as parameters, returning a list of airports that meet the specified criteria. The outcome is a comprehensive compilation of airports capable of accommodating the Boeing 757, facilitating strategic flight planning and logistics for airlines. Refer to *Figure 2* for the test result.



```
// Test no 1: Success
System.out.println("\n***** Test 1 *****");
var airport = france.getCity( name: "Paris")
    .getAirport( name: "Paris International Airport");
System.out.println(getAircraftTypesAt(airport));

// Test no 2: Success
System.out.println("\n***** Test 2 *****");
List<Country> countries = List.of(france, england, egypt);
System.out.println(getAirportsSupporting(countries, AircraftType.BOEING_757));

// Test no 3: Success
System.out.println("\n***** Test 3 *****");
System.out.println(france.getAirports());

// Test no 4: Success
System.out.println("\n***** Test 4 *****");
System.out.println(trackAircraft(countries, id: 1));
}
```

***** Test 1 *****
[BOEING_757, AIRBUS_A320]

***** Test 2 *****
[Airport{name=Paris International Airport, location=City{name=Paris, type=BIG_CITY}, aircraftTypes={BOEING_757, AIRBUS_A320}}, Airport{name=Marseille, location=City{name=Mars

***** Test 3 *****

***** Test 4 *****

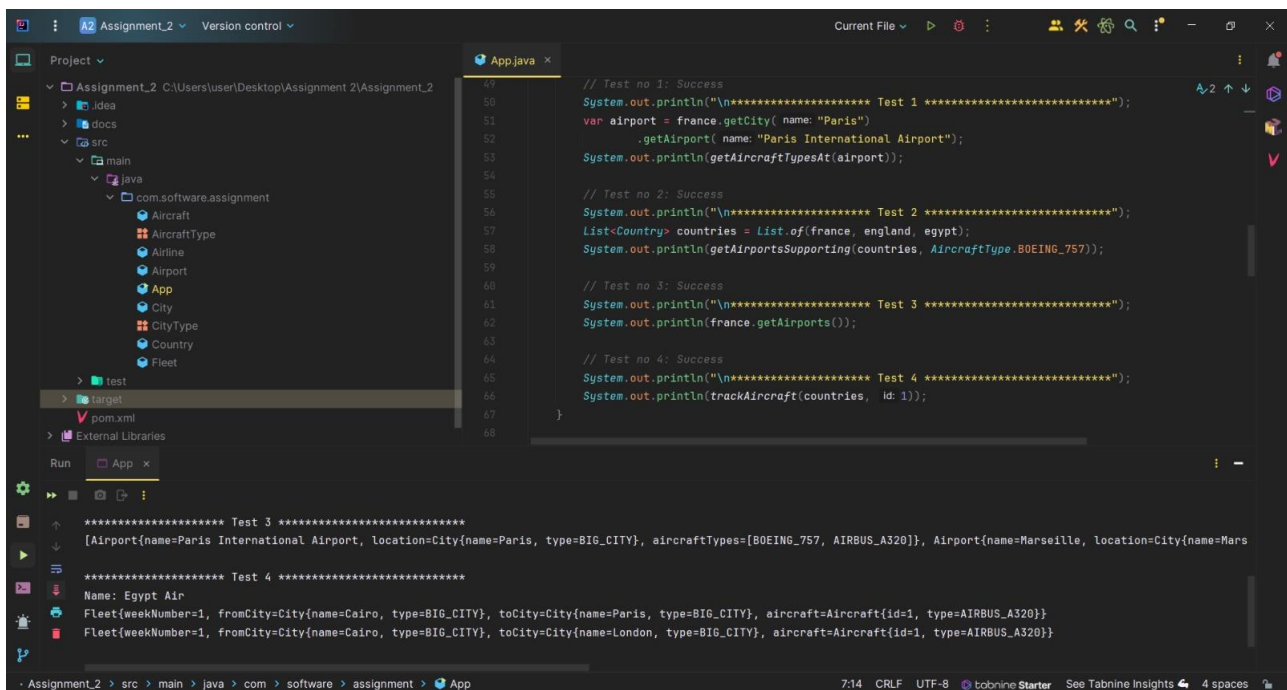
Figure 2: Result of Test Cases 1 and 2

Test 3. List Airports in France

This functionality leverages the hierarchical structure of countries, cities, and airports to retrieve a list of all airports located within a specific country, in this case, France. By accessing the list of cities within the target country object and subsequently retrieving the airports associated with each city, this functionality systematically assembles a comprehensive catalogue of airports situated within France's borders. The implementation is achieved through the **getAirports()** method of the Country class, which aggregates airports from all cities within the country. The result is a concise inventory of airports in France, essential for facilitating efficient air travel and logistical planning within the country's airspace. Refer to *Figure 3* for the test result.

Test 4. Track Aircraft and Flights

This functionality facilitates the tracking of a specific aircraft's ownership and its flight history within the last week based on its unique identifier. By iterating through the list of countries and airlines, it searches for the aircraft with the specified ID and retrieves the associated airline. Subsequently, it queries the airline's fleet to identify all flights made by the aircraft within the last week. This process is facilitated by the **trackAircraft()** method, which takes a list of countries and the aircraft ID as parameters, returning detailed information about the aircraft's ownership and recent flight activity. The result is a comprehensive overview of the aircraft's operational history, enabling effective maintenance scheduling and regulatory compliance for airline operators. Refer to *Figure 3* for the test result.



The screenshot shows an IDE with a project named 'Assignment_2'. The left sidebar shows the project structure with a 'test' directory. The main editor displays Java code for four tests. The output console at the bottom shows the results of these tests.

```
// Test no 1: Success
System.out.println("\n***** Test 1 *****");
var airport = france.getCity( name: "Paris")
    .getAirport( name: "Paris International Airport");
System.out.println(getAircraftTypesAt(airport));

// Test no 2: Success
System.out.println("\n***** Test 2 *****");
List<Country> countries = List.of(france, england, egypt);
System.out.println(getAirportsSupporting(countries, AircraftType.BOEING_757));

// Test no 3: Success
System.out.println("\n***** Test 3 *****");
System.out.println(france.getAirports());

// Test no 4: Success
System.out.println("\n***** Test 4 *****");
System.out.println(trackAircraft(countries, id: 1));
```

***** Test 3 *****
[Airport{name=Paris International Airport, location=City{name=Paris, type=BIG_CITY}, aircraftTypes=[BOEING_757, AIRBUS_A320]}, Airport{name=Marseille, location=City{name=Mars
***** Test 4 *****
Name: Egypt Air
Fleet{weekNumber=1, fromCity=City{name=Cairo, type=BIG_CITY}, toCity=City{name=Paris, type=BIG_CITY}, aircraft=Aircraft{id=1, type=AIRBUS_A320}}
Fleet{weekNumber=1, fromCity=City{name=Cairo, type=BIG_CITY}, toCity=City{name=London, type=BIG_CITY}, aircraft=Aircraft{id=1, type=AIRBUS_A320}}

Figure 3: Result of Test Cases 3 and 4

APPENDIX

Table Of Figures

Figure 1: Class Diagram: Modelling an Airline System3

Figure 2: Result of Test Cases 1 and 24

Figure 3: Result of Test Cases 3 and 45