IT313: Software Engineering
Autumn 2024-25
Lab Session: Class Modeling

Q.1 Prepare a class diagram for the following object diagram that shows a portion of Europe.

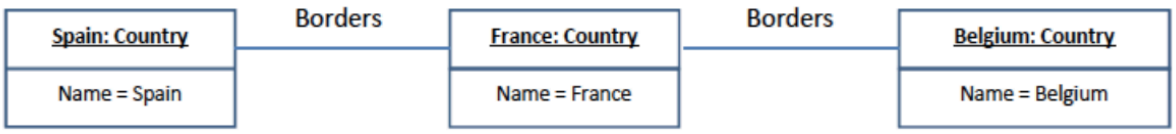
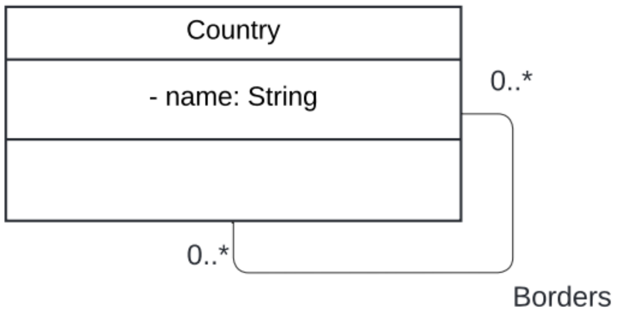


Figure-1



Class Diagram Structure:

- Class Name: Country
- Attributes: name: String
- Associations: Borders : A many-to-many association between the Country class.

Q.2 Prepare a class diagram for object diagram given in Figure -2. Explain your multiplicity decisions. What is the smallest number of points required to construct a polygon? Does it make a difference whether or not point may be shared between polygons? Your answer should address the fact that points are ordered.

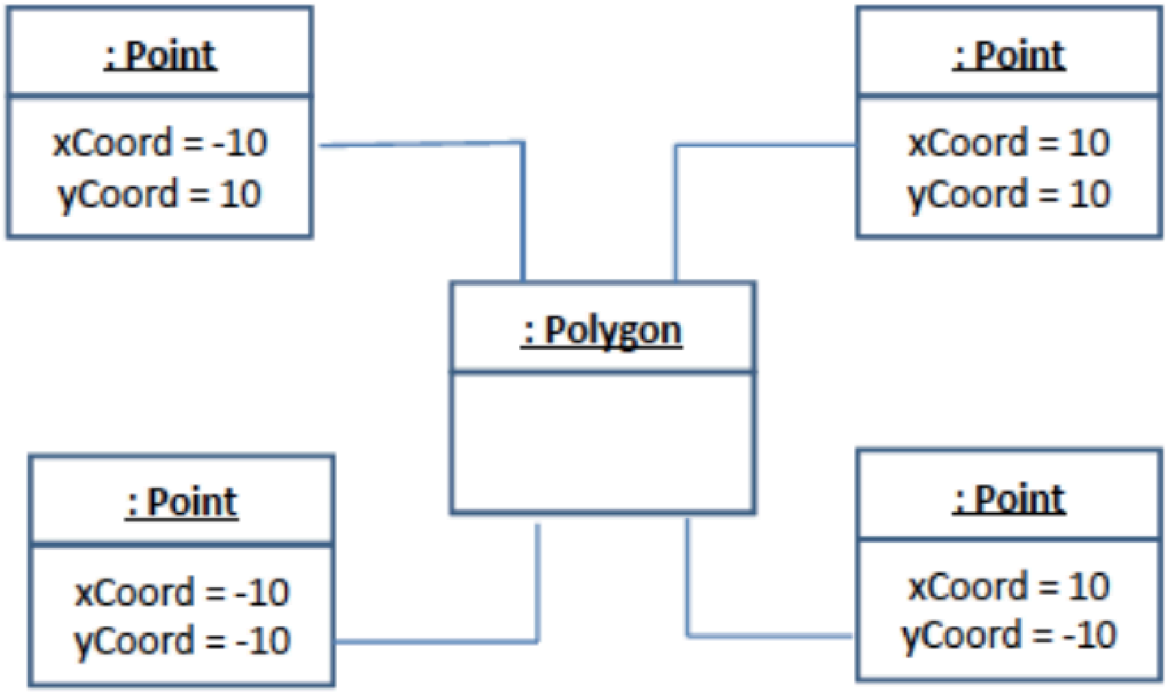
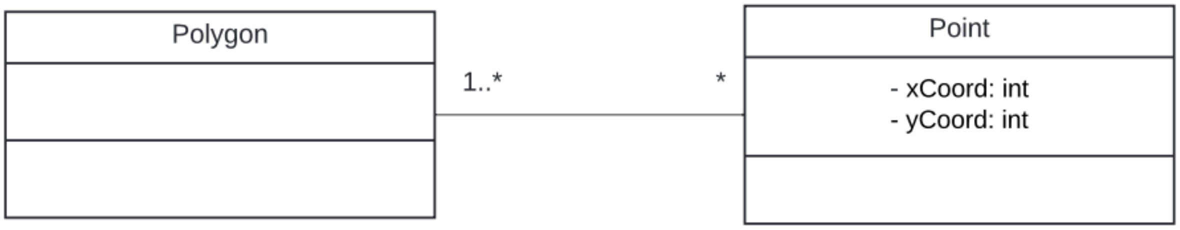


Figure - 2



- **Classes**

- Point: The diagram shows multiple Point objects with xCoord and yCoord attributes. This suggests we need a class Point with those attributes.
- Polygon: The Polygon class is shown in the center, connected to multiple Point objects, indicating that a polygon is defined by multiple points.

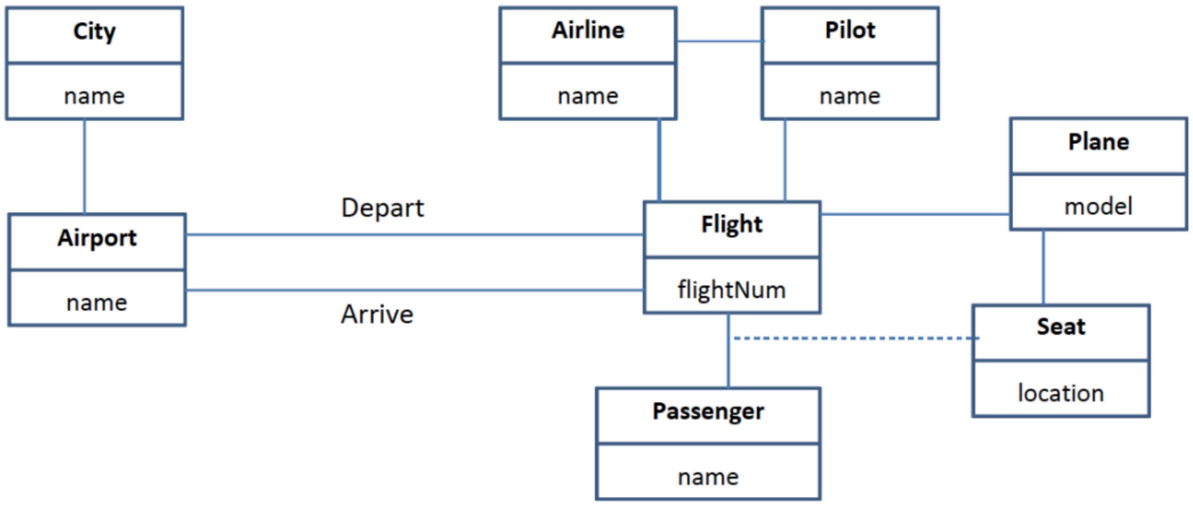
- **Attributes**

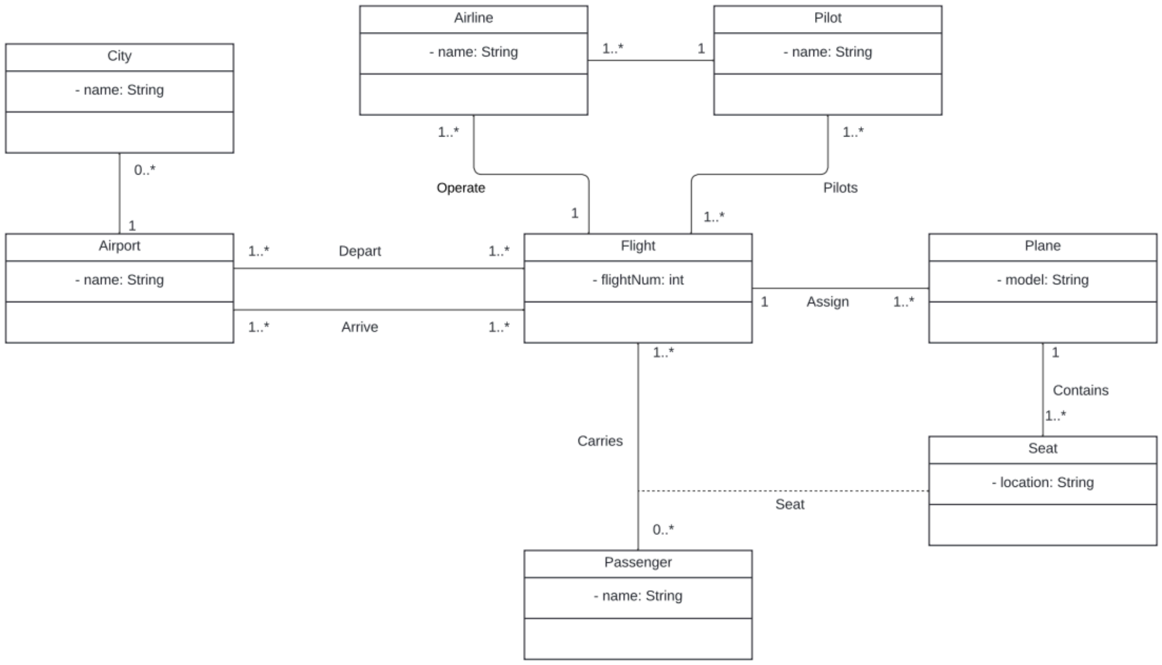
- Point Class: xCoord: int
  yCoord: int
- Polygon Class: The Polygon class itself does not seem to have additional attributes based on the diagram.

- **Explanation of Multiplicity:**

- Polygon to Point: A polygon has at least 3 points (hence 1..* cardinality). A point can belong to multiple polygons (many-to-many relationship).
- Ordered Points: The relationship between Polygon and Point is ordered, meaning the order of the points matters when forming the shape.

Q.3 Figure 3 is a partially completed class diagram of an air transportation system. Add multiplicities in the diagram. Also add association names to unlevelled associations.

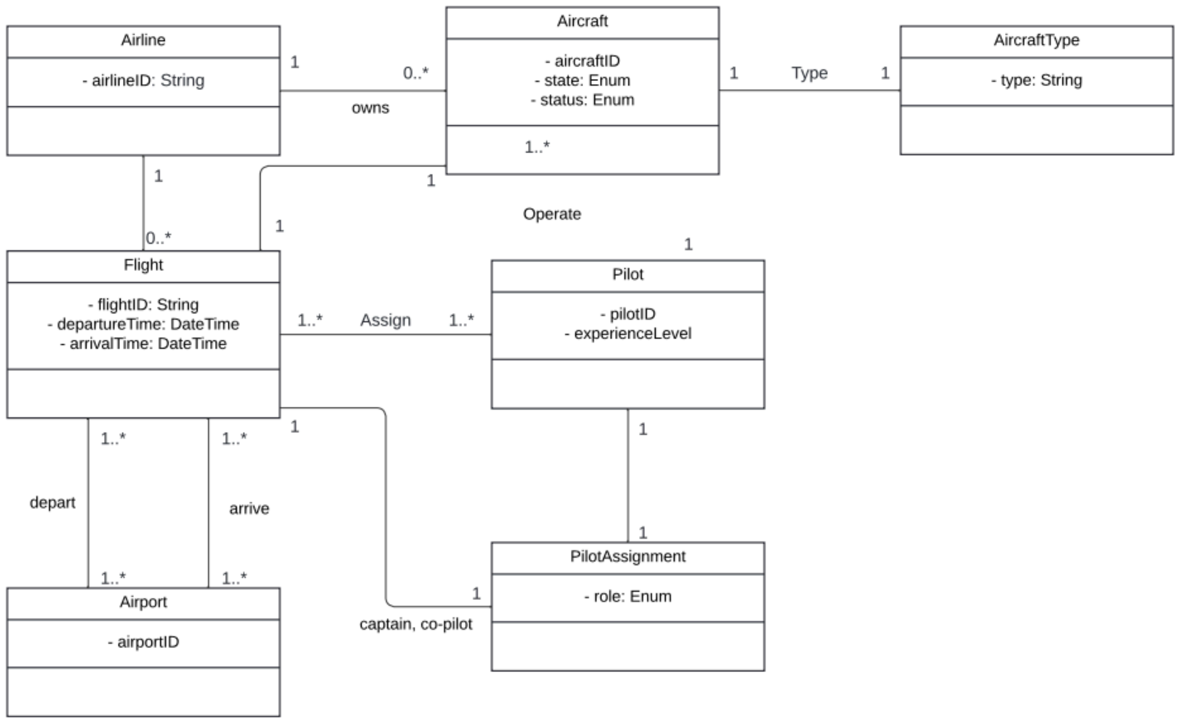




- **Multiplicities**

1. City ↔ Airport: A city can have multiple airports, but each airport is in exactly one city. Multiplicity: 0..* (City) ↔ 1 (Airport)
2. Airport ↔ Flight: A flight departs from and arrives at exactly one airport, but an airport can handle many flights. Multiplicity: 1..* (Airport) ↔ 1..* (Flight) for both the "Depart" and "Arrive" relationships.
3. Flight ↔ Airline: A flight is operated by exactly one airline, but an airline can operate many flights. Multiplicity: 1..* (Airline) ↔ 1 (Flight)
4. Flight ↔ Pilot: A flight may have multiple pilots, but each pilot can be assigned to multiple flights. Multiplicity: 1..* (Flight) ↔ 1..* (Pilot)
5. Flight ↔ Plane: A flight uses exactly one plane, but a plane can be used for multiple flights (although not at the same time). Multiplicity: 1 (Flight) ↔ 1..* (Plane)
6. Plane↔ Seat: A plane contains many seats, but each seat is associated with exactly one flight. Multiplicity: 1 (Plane) ↔ 0..* (Seat)
7. Flight ↔ Passenger: A flight carries many passengers, and each passenger can fly on multiple flights. Multiplicity: 1..* (Flight) ↔ 0..* (Passenger)

Q.4 We want to model a system for management of flights and pilots. An airline operates flights. Each airline has an ID. Each flight has an ID a departure airport and an arrival airport: an airport as a unique identifier. Each flight has a pilot and a co-pilot, and it uses an aircraft of a certain type; a flight has also a departure time and an arrival time. An airline owns a set of aircrafts of different types. An aircraft can be in a working state or it can be under repair. In a particular moment an aircraft can be landed or airborne. A company has a set of pilots: each pilot has an experience level: 1 is minimum, 3 is maximum. A type of aeroplane may need a particular number of pilots, with a different role (e.g.: captain, co-pilot, navigator): there must be at least one captain and one co-pilot, and a captain must have a level 3.



Key Classes and Their Attributes
1. Airline: Attributes: airlineID: String Associations: Owns a set of aircraft, operates multiple flights.
2. Flight: Attributes: flightID: String departureTime: DateTime arrivalTime: DateTime Associations: Departs from and arrives at an Airport . Has a Pilot and Co-pilot . Operates with an Aircraft .
3. Airport: Attributes: airportID: String (unique identifier)
4. Pilot: Attributes: pilotID: String experienceLevel: Integer (1-3) Associations: Assigned to multiple flights in different roles (captain, co-pilot, navigator).
5. Aircraft: Attributes: aircraftID: String state: Enum (working, underRepair) status: Enum (landed, airborne) Associations: Owned by an Airline , used in a Flight .
6. AircraftType: Attributes: type: String Associations: Associated with multiple Aircraft . Specifies the required number of pilots (roles: captain, co-pilot, navigator, etc.).

Relationships (Associations)
1. Airline ↔ Aircraft: An airline owns multiple aircraft, but an aircraft belongs to exactly one airline. Multiplicity: 1 (Airline) ↔ 0..* (Aircraft)
2. Flight ↔ Airport: A flight departs from one airport and arrives at another. Multiplicity: 1 (Flight) ↔ 1 (Airport) for both departure and arrival airports.
3. Flight ↔ Pilot: A flight has a minimum of one captain and one co-pilot, and potentially other roles like navigator. A captain must have an experience level of 3. Multiplicity: 1..* (Flight) ↔ 1..* (Pilot) (depending on the roles defined by the aircraft type).
4. Flight ↔ Aircraft: A flight uses one aircraft, and an aircraft can be used for multiple flights over time. Multiplicity: 1 (Flight) ↔ 1 (Aircraft)
5. Aircraft ↔ AircraftType: Each aircraft is of a certain type, and an aircraft type can apply to multiple aircraft. Multiplicity: 1 (Aircraft) ↔ 1 (AircraftType)
6. AircraftType ↔ Pilot: An aircraft type determines the number of pilots required and their roles (captain, co-pilot, navigator). Multiplicity: 1 (AircraftType) ↔ 1..* (Pilot)