

- **Team:**
  - Attaluri Laxmi Naga Santosh
  - Saikrishna Jaliparthi
  - Shruthi Sukumar
- **Title:** Flight Reservation Application
- **Refactoring Analysis:**

On the whole, the refactoring exercise initiated a re-thinking of our original application design. Overall, we changed the structure to reduce coupling between classes in comparison to our previous class diagram. We removed a redundant class **Ticket** and incorporated the methods into the **Reservation** class.

### Design Patterns:

Based on the GoF Design Patterns textbooks as well as Prof. Boese's slides, we incorporated two of the main design patterns in our software design. First, for database access we used the **Singleton** DP. The **Database** class in our class diagram acts as the access point from our code to the database which contains list of flights, customers and reservations. The **Database** class communicates with Hibernate, the object-relational mapping tool for Java. Using the **Singleton** design pattern, the **Database** class by means of the `getDBTable()` method, ensures that no more than one session is created. The following is the code snippet for the **Database** class:

```
class Database
{
    private static SessionFactory factory;

    public static synchronized SessionFactory getDBTable()
    {
        if(factory==null)
            factory=new Configuration().configure().buildSessionFactory();
        return factory;
    }
}
```

Another design pattern that was implemented in our design during refactoring was the **Composite** design pattern. The **Transportation** class is the base class for all possible transportations in an itinerary. Some airports include a chopper and bus in addition to flights. A group of flights to be listed in a search as well as the list of flights in the reservation are compositions of transportations that are available. The application of the **Composite** design patterns enables the groups of transport objects to be treated uniformly as a flight (or bus or chopper). Using this design pattern makes it easy to add new classes for any additional transport medium in the future, without disturbing the base **Transportation** class. The **Composite** DP implementation in our class diagram is evident from the connections from the **Transportation** class to the **Flight** class (which is a leaf node), the **AvailableTransport** class and the **SelectedTransport** class.

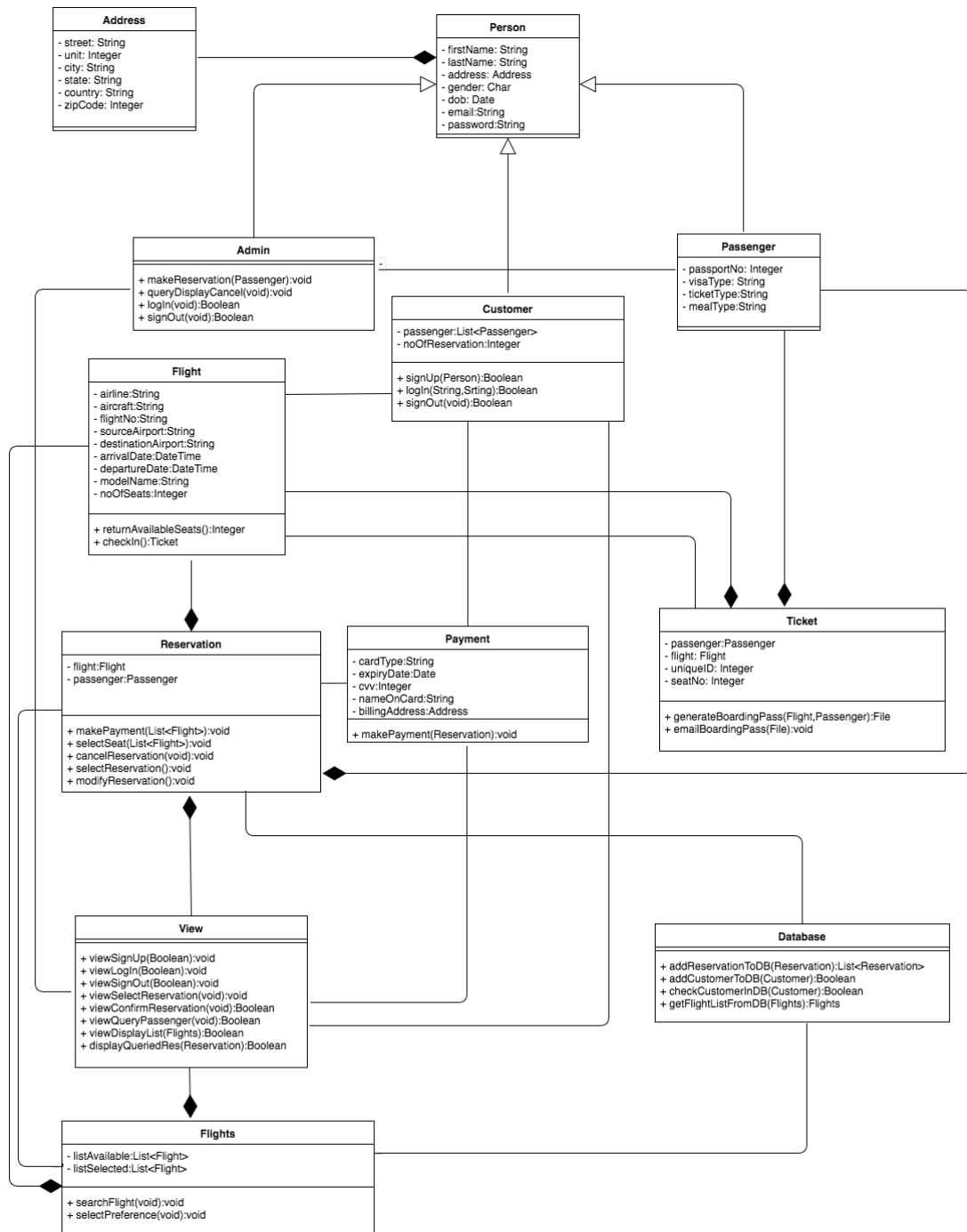


Figure 1: Original Class diagram: Shows compositions, inheritance and dependency relations

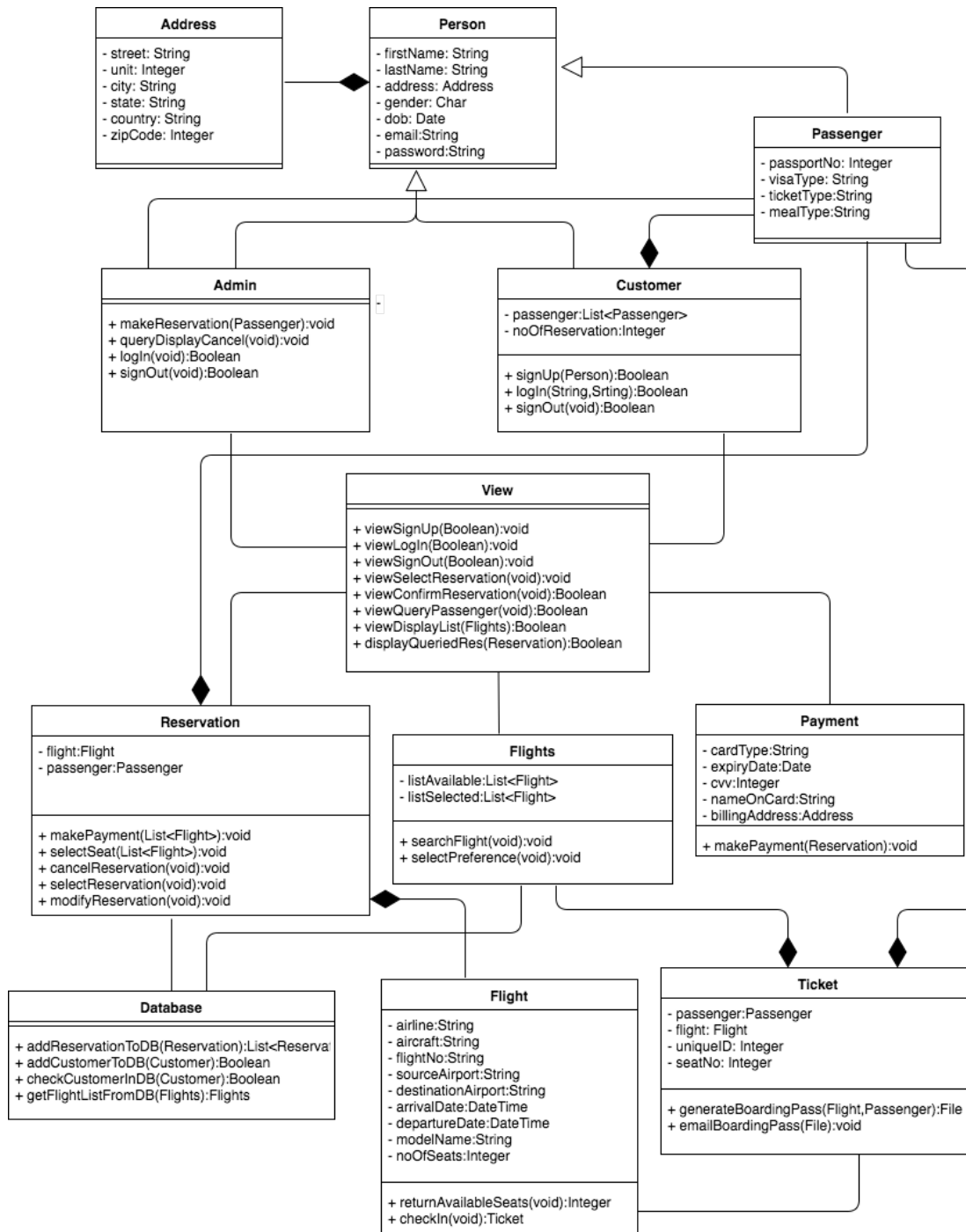


Figure 2: Original Class diagram: Cleaned up version removing some of the inconsistencies noted in the part 2 feedback and in general making it more readable

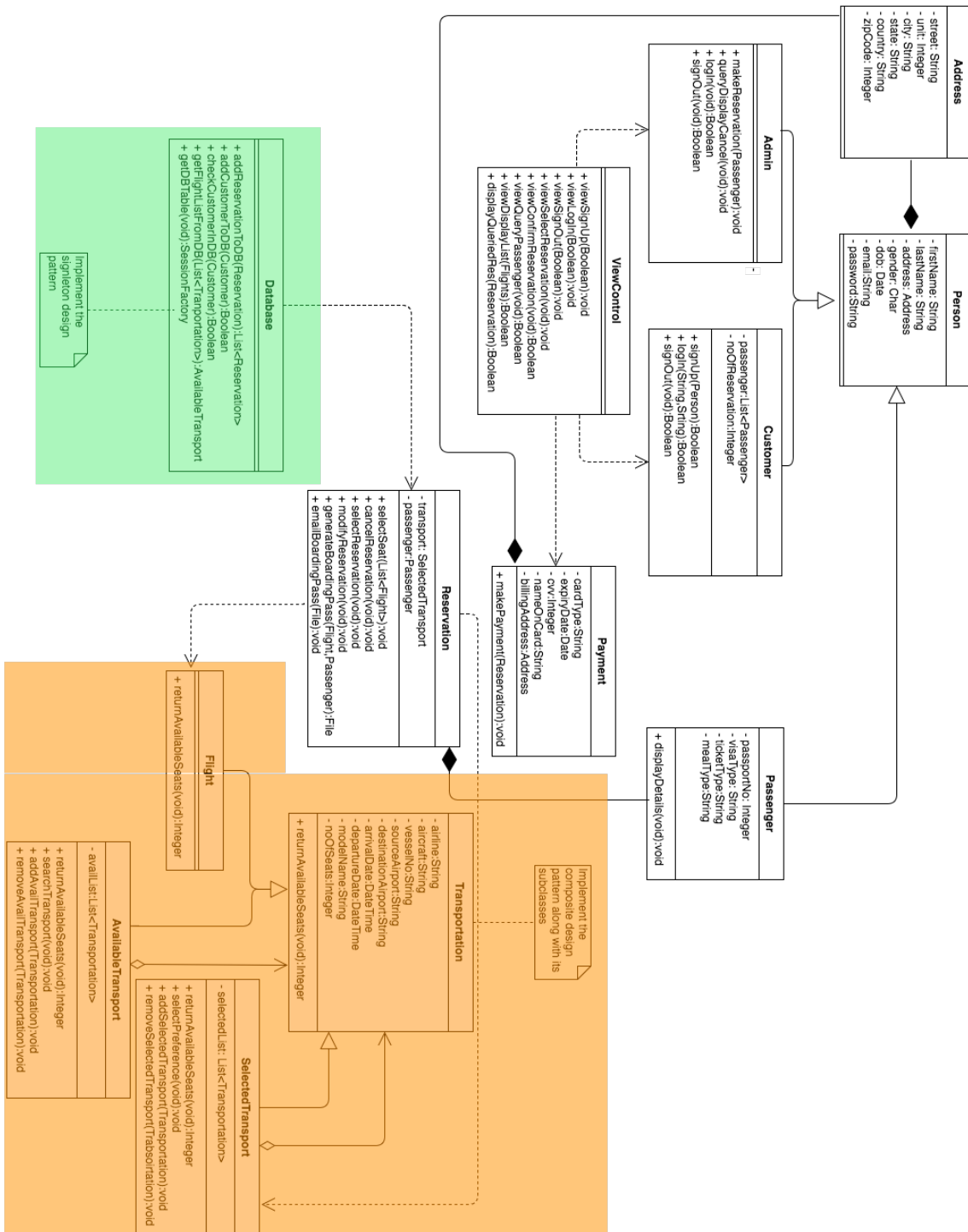


Figure 3: Refactored Class diagram: Shows the singleton (green shading) and composite (orange shading) design patterns implemented by means of notes in the class diagram