Travel Management System

Team: Arunsundar Kannan, Ashish Tak, Prajnya Satish (Team #39)

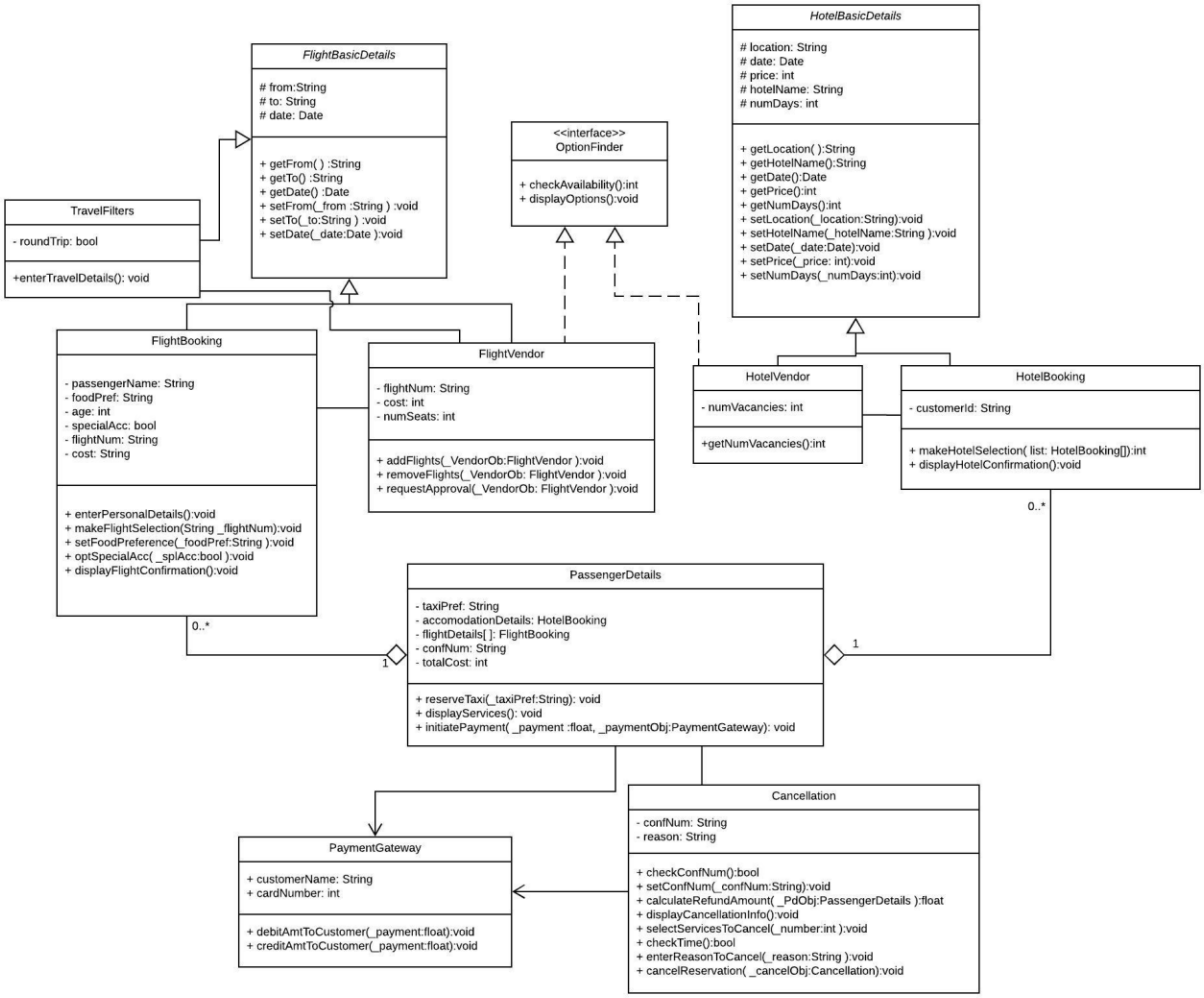
Title: Travel Management System Project

Description: A travel management system to enable a traveller to book flight tickets and manage other aspects of his journey such as food preference, accommodation and commute on the same platform. The customer books for all the services required together. If he so wishes, he could also cancel the booking on the same application.

Framework: Spring

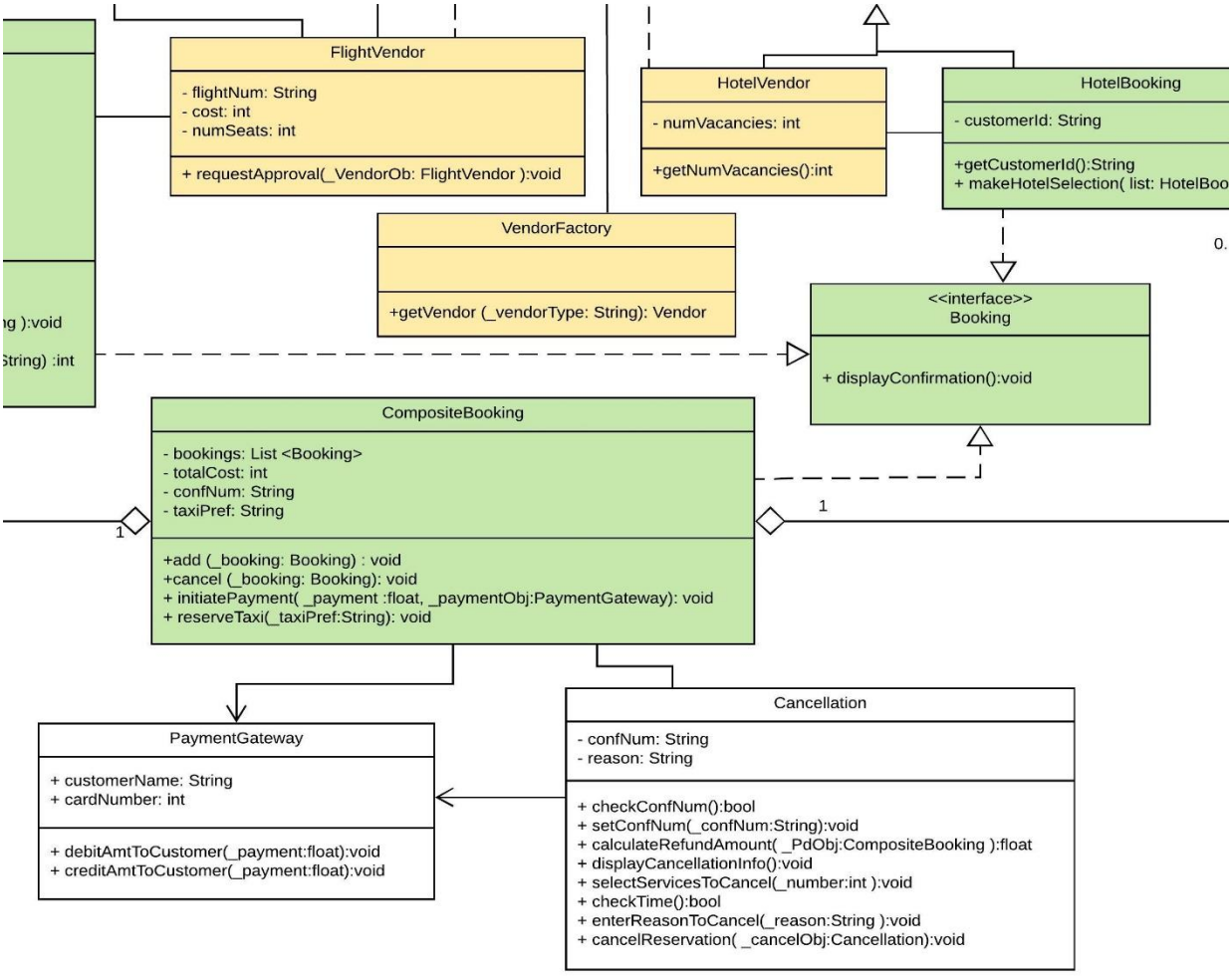
Data Storage: MySQL Database

2. Previous Class Diagram –



3. Completed Class Diagram –

(Class Diagram of only the new classes implemented after project part 3.)



4. Summary –

- We made some changes to the class diagram to support the design pattern choices made.
- Two design patterns were identified and added to the implementation; Factory and Composite Design Patterns
- We made corresponding changes to the code to accommodate these design pattern.
- Completed the unimplemented classes that weren't added in the previous iteration of the progress report. (PaymentGateway, Cancellation and CompositeBooking)
- We are also working on porting the code to Spring Framework.

5. Breakdown –

Work done by Arunsundar Kannan –

- Made changes to accommodate the factory design pattern.
- Implemented corrections and additions in the FlightVendor and HotelVendor classes.
- Implemented the interactions between these classes to maintain dataflow.
- Modifications in the class diagram to add Factory Design Pattern.

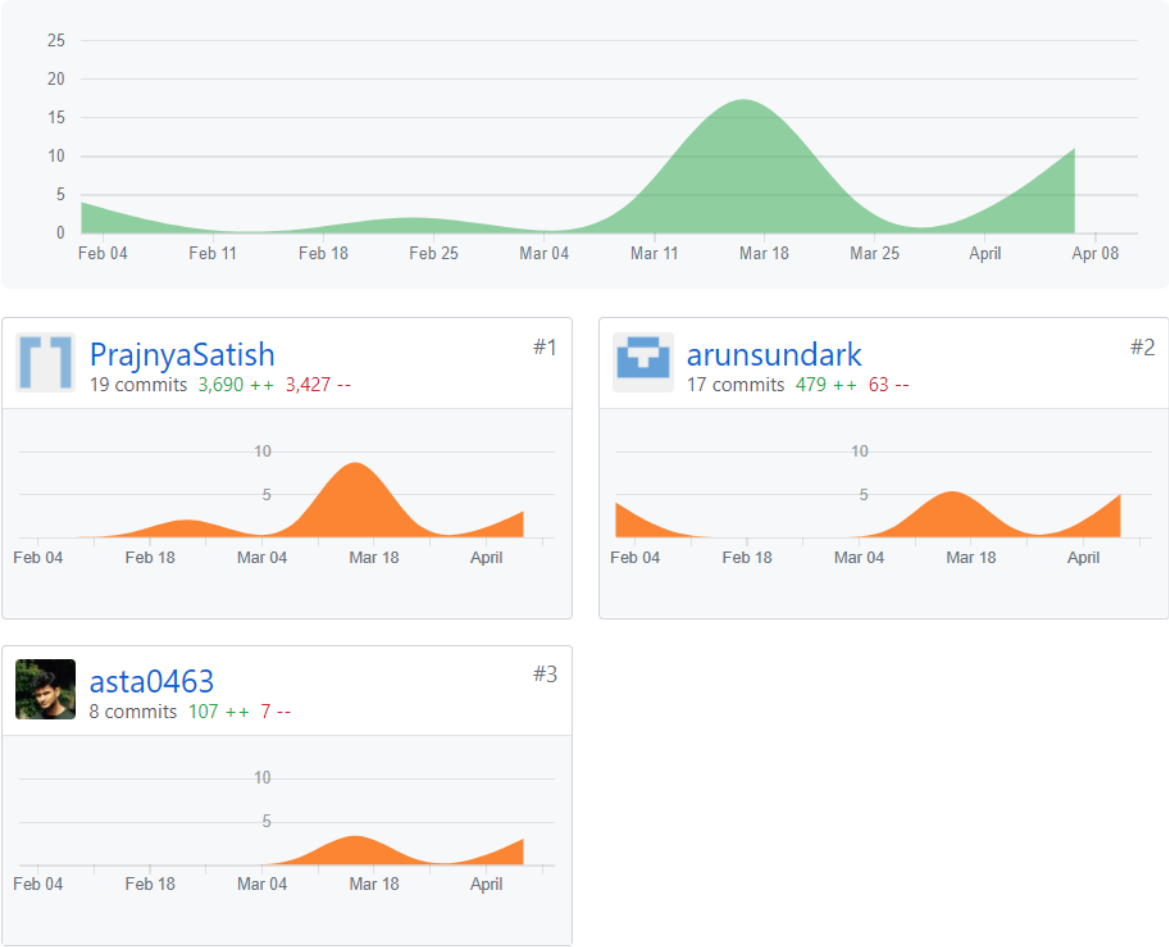
Work done by Ashish Tak –

- Added code to implement the composite design pattern for the Booking class which acts as the interface that interacts with the client.
- Created client interface for the Composite Design Pattern.
- Modifications to HotelBooking and FlightBooking classes to incorporate Composite Design Pattern.
- Corrections in class diagram to implement composite design pattern

Work done by Prajnya Satish –

- Implemented PaymentGateway as an interface between customer and system to debit and credit the customer based on the services chosen.
- Implemented the Cancellation class.
- Created the CompositeBooking class to create the different booking components based on the customer requirements.
- Corrected certain interactions in the other classes based on the changes in the composite sections of the code.

6. Github graph –



7. Estimation of Remaining Effort –

- Most classes are inter connected now and there is dataflow between them. The Booking class establishes an interface with the client too. We are currently working on porting the project to Spring.
- What remains is to store the data input from the customer into a database and integrate the database with the rest of the system.
- Once this is done, we can run further unit tests to validate the concept based on the use case diagram.

8. Design Patterns –

- **Factory Design Pattern** (Marked Yellow) –The concrete factory class is VendorFactory which serves for dynamic creation of objects implementing the Vendor interface. So, at run-time, depending on the string passed to the getVendor() method of VendorFactory, the factory creates and returns either a FlightVendor or a HotelVendor object.

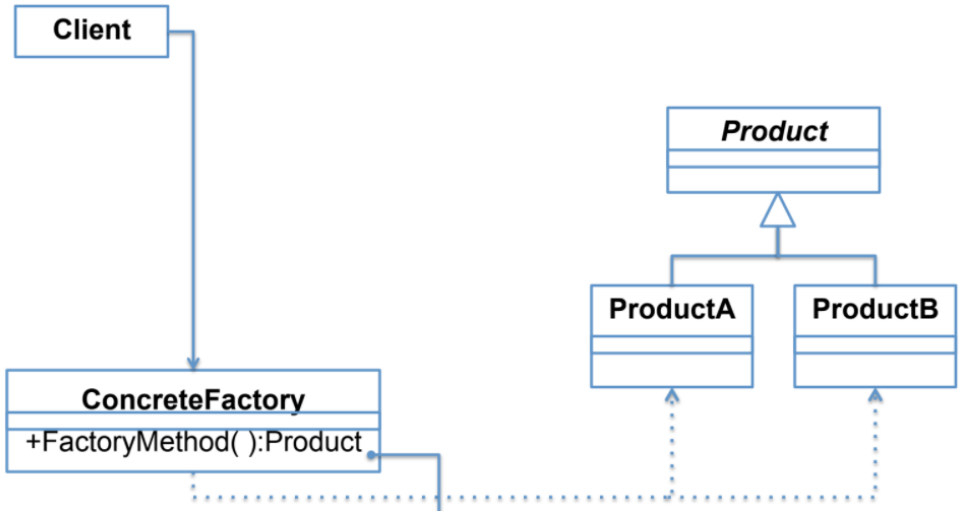
  Implementing this design pattern makes the code open for extension and we are encapsulating what is varying creationally in the VendorFactory class. The client is effectively decoupled from the actual instantiated classes.

- **Composite Design Pattern** (marked Green) – In this case, the Booking interface serves as the component to which the client is coupled, and the leaf nodes are the FlightBooking and HotelBooking classes. CompositeBooking class is the composite containing a list of one or more of these leaf nodes.

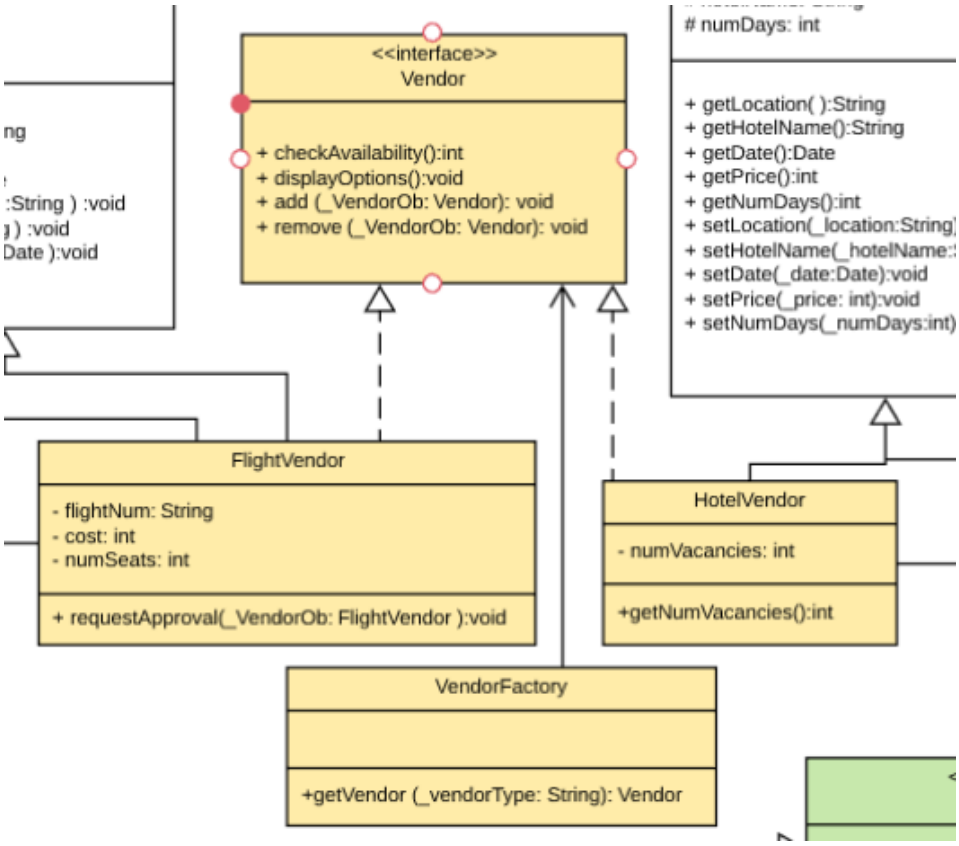
  Implementing this design pattern will help us access every type of booking in a uniform way. A customer may choose to book only a flight, or both flight and hotel making the hotel a choice. Having a composite class maintaining a list of all the bookings which a customer makes is much more efficient than the earlier approach which we had used where there were separate objects created for each type of booking. Extensibility is again achieved well through this pattern since we can now have multiple flight and hotel bookings for a customer contained inside one data member as a list and treat every booking in the same way.

9. Class Diagram with each Design Pattern implemented –

- **Factory Design Pattern:**
  - Class diagram for the actual design pattern

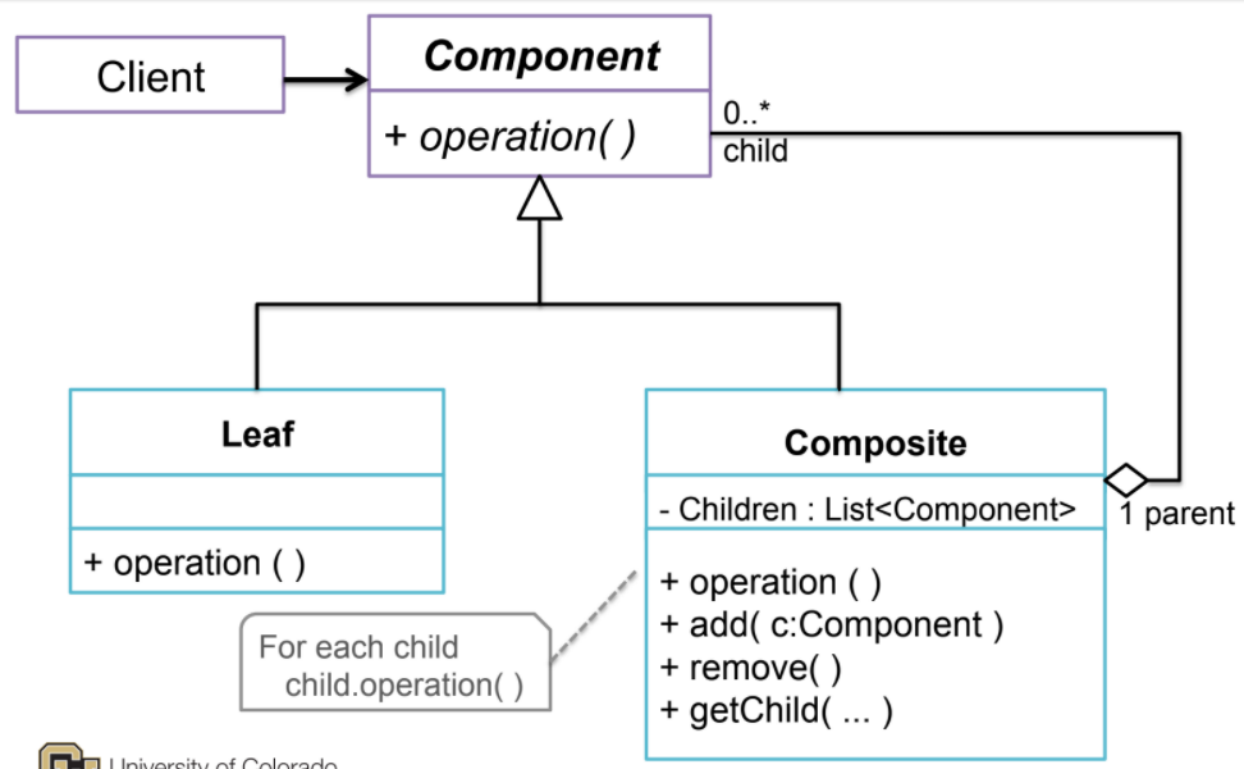


  - Portion of the class diagram implementing the design pattern

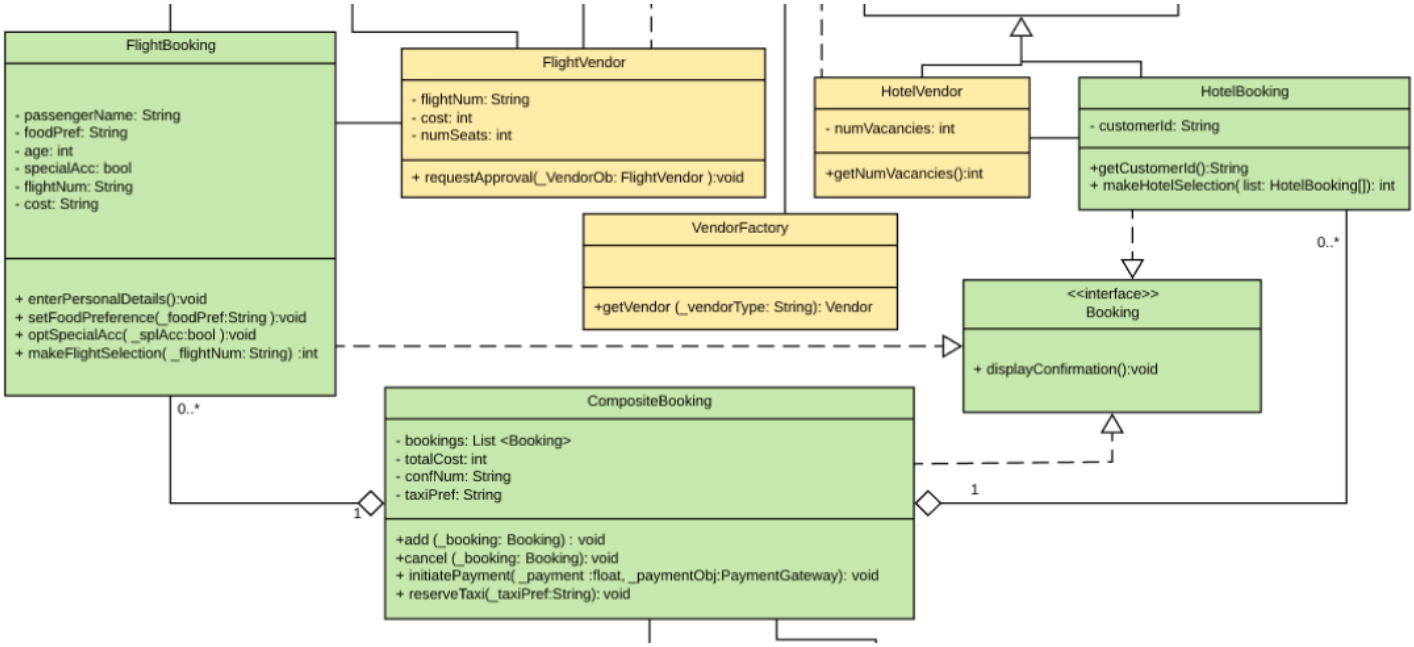


- Participant Mapping
  The 1:1 mapping of the participants is as follows:
  - ConcreteFactory: VendorFactory
  - Product: Vendor
  - ProductA: FlightVendor
  - ProductB: HotelVendor

- **Composite Design Pattern:**
  - Class diagram for the actual design pattern



- Portion of the class diagram implementing the design pattern



- Participant Mapping
  - Component: Booking
  - Leaf: FlightBooking, HotelBooking
  - Composite: CompositeBooking

10. **Final Iteration** –

- In the final iteration, we plan to port the code to spring framework in order to implement the MVC architecture.
- We also need to store and retrieve the data from the MySQL database.
- The final step would be to test the project based on the use case document and verify that all the requirements have been satisfied.