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: Hibernate

Data Storage: MySQL Database

2. Features Implemented –

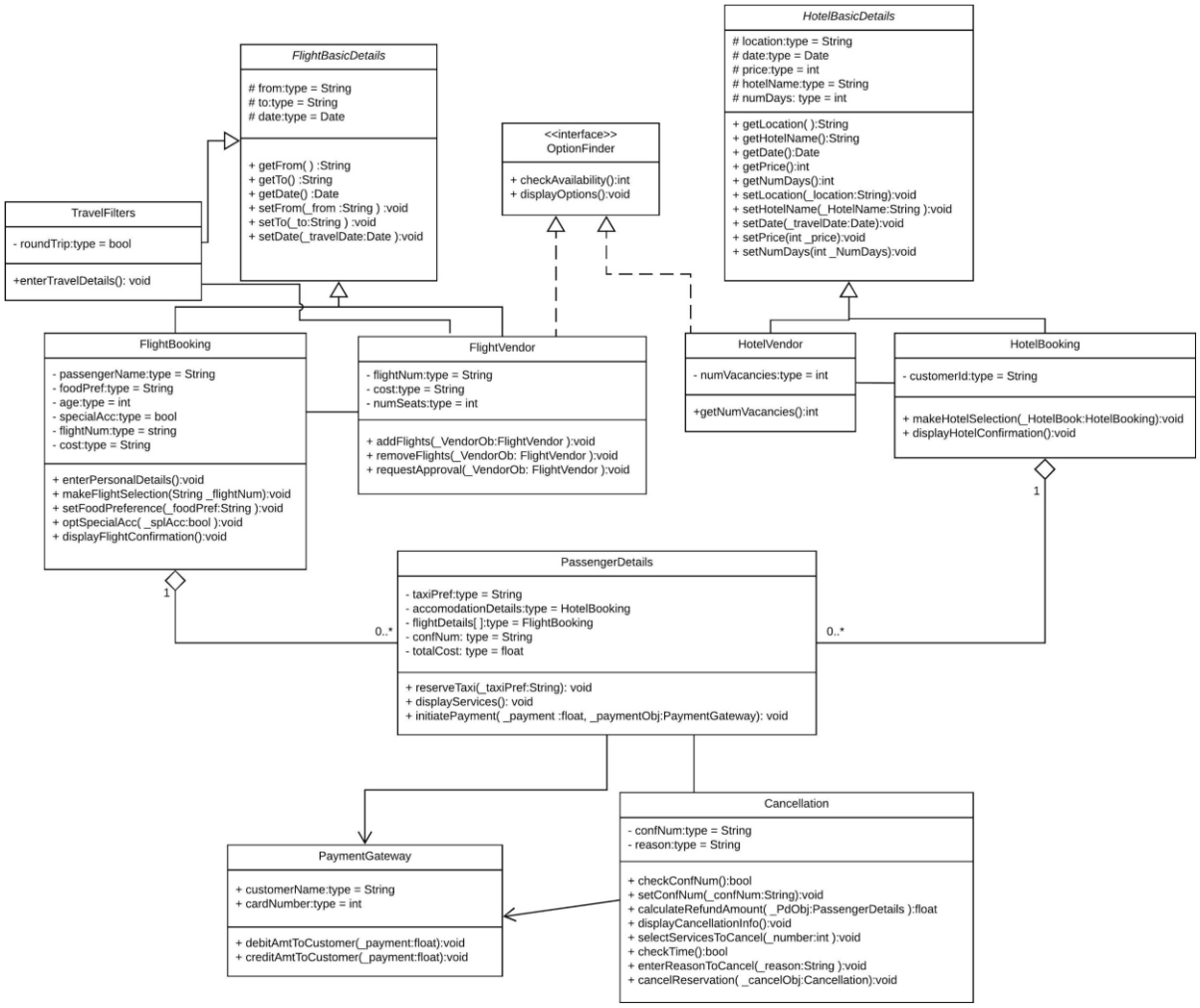
ID	DESCRIPTION
US01	As an Airline vendor, I want to add a new flight to the system by providing the details about the particular flight.
US03	As an Airline vendor, I want to remove any selected route
US05	As a customer, I wish to reserve plane tickets by providing source, destination and dates of travel
US06	As a customer, if I prefer to use the accommodation service, I should be able to select from a list of suggested hotels based on my destination
US07	As a customer, I should be able to pay for the services I have opted for using my credit/debit card
US08	As a customer, I should be able to cancel a reservation by specifying my reservation details
US11	As a customer, I should receive a confirmation message after completing the booking process
US12	As a customer, I should have an option of reserving a taxi from the destination airport

3. Features not Implemented –

ID	DESCRIPTION
US02	As the system admin, I need to check the details of the newly added route by an airline vendor so that I can decide whether to approve
US04	As the system admin, I need to decide whether to approve the removal of a route based on passenger and business commitments so that inconvenience to passengers is minimized
US09	As the admin, I need to determine the percentage of refund based on the reason given for cancellation by the customer
US10	As a payment gateway operator, I need to process the assigned refund to the customer's credit/debit card

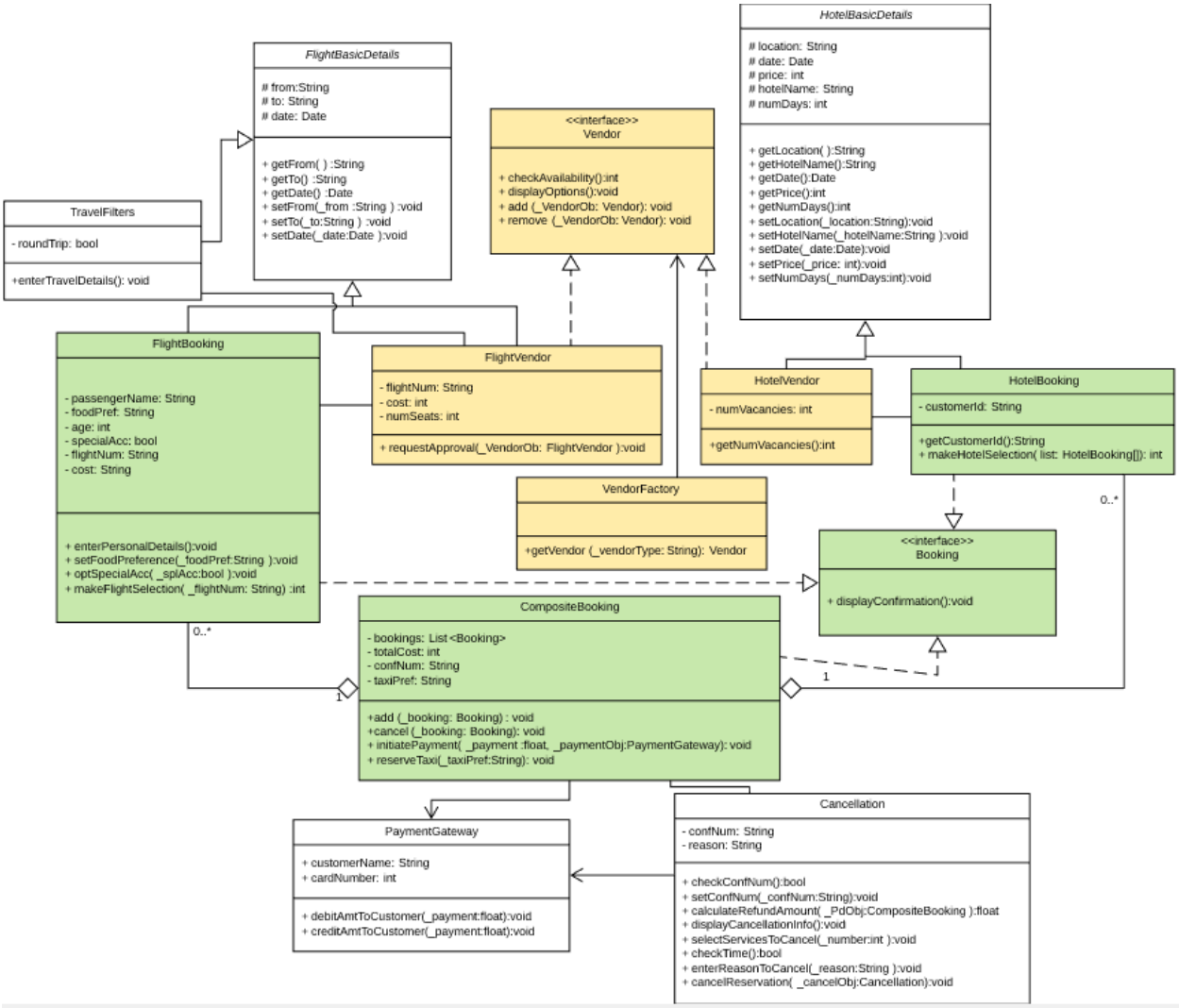
4. Changes in the class diagram through the Design phase –

- **Original Class Diagram**



Link to image

- **Final Class Diagram**



As can be seen from the highlighted parts in the new class diagram, it's the design patterns that have driven most of the changes in the class diagram through the semester.

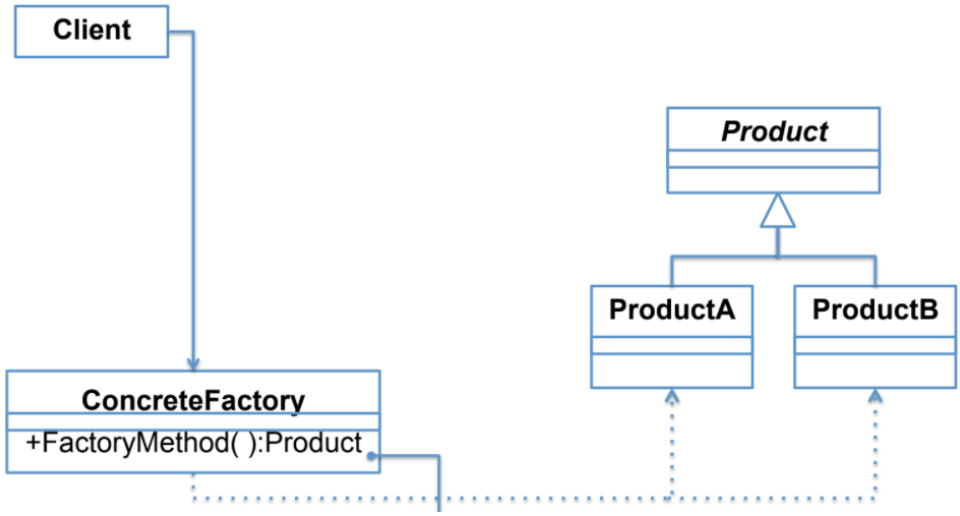
The first major change was when the VendorFactory class was introduced in the system to serve for dynamic creation of objects for all types of vendors. Implementing such a modification made the system open for extension, and closed for modification for the different types of vendor actors

which might get added to the system dynamically. Any type of vendor client is effectively decoupled from the concrete instantiated classes. We had chosen this design pattern in particular because of the different types of vendor actors that we have in our system. Implementation of this kind of a creational design pattern will make it easier to incorporate even newer types of transportation vendors to the system.

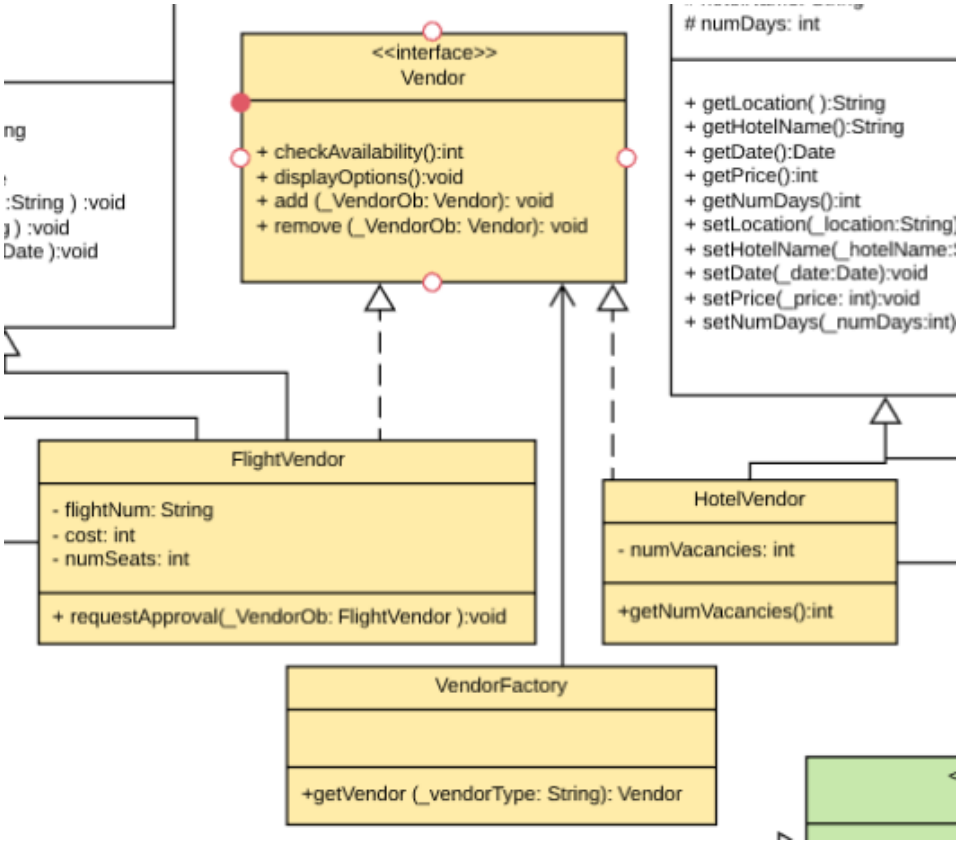
The second major change was through the CompositeBooking class and the Booking interface. Since our system is providing a customer the flexibility to choose to book only a flight, a hotel or a combination of any number of these, it is important that every booking made is accessed in a uniform way. This is indeed the reason behind such a modification in the design which helps to access multiple types of bookings in a uniform way. Extensibility is thus again well achieved through this design pattern to add any other type of ticket bookings to this list. In our system, we are thus coding to an interface, not to an implementation.

5. Design Patterns –

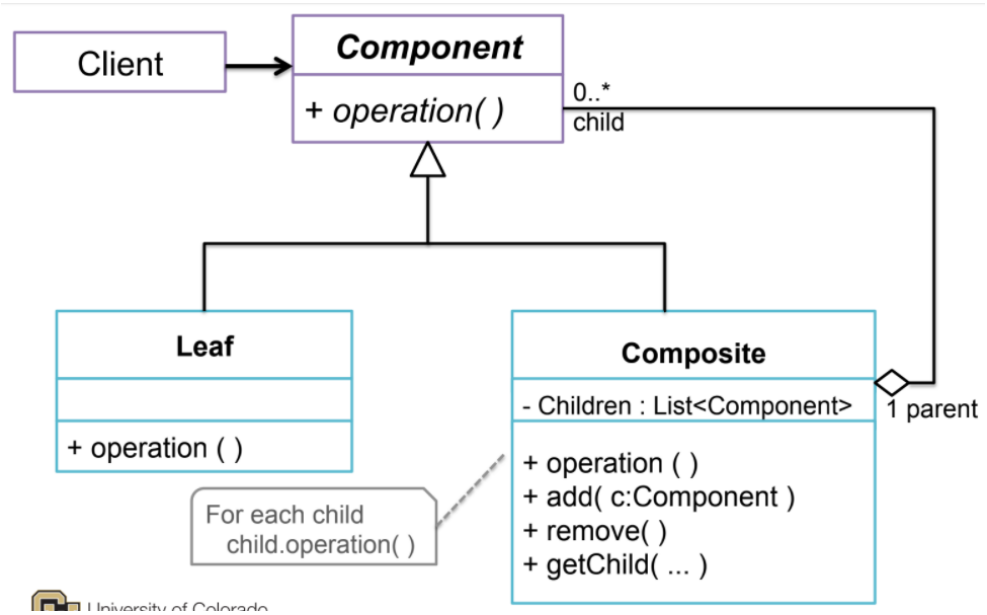
- **Factory Design Pattern:**
  - Class diagram for the actual design pattern



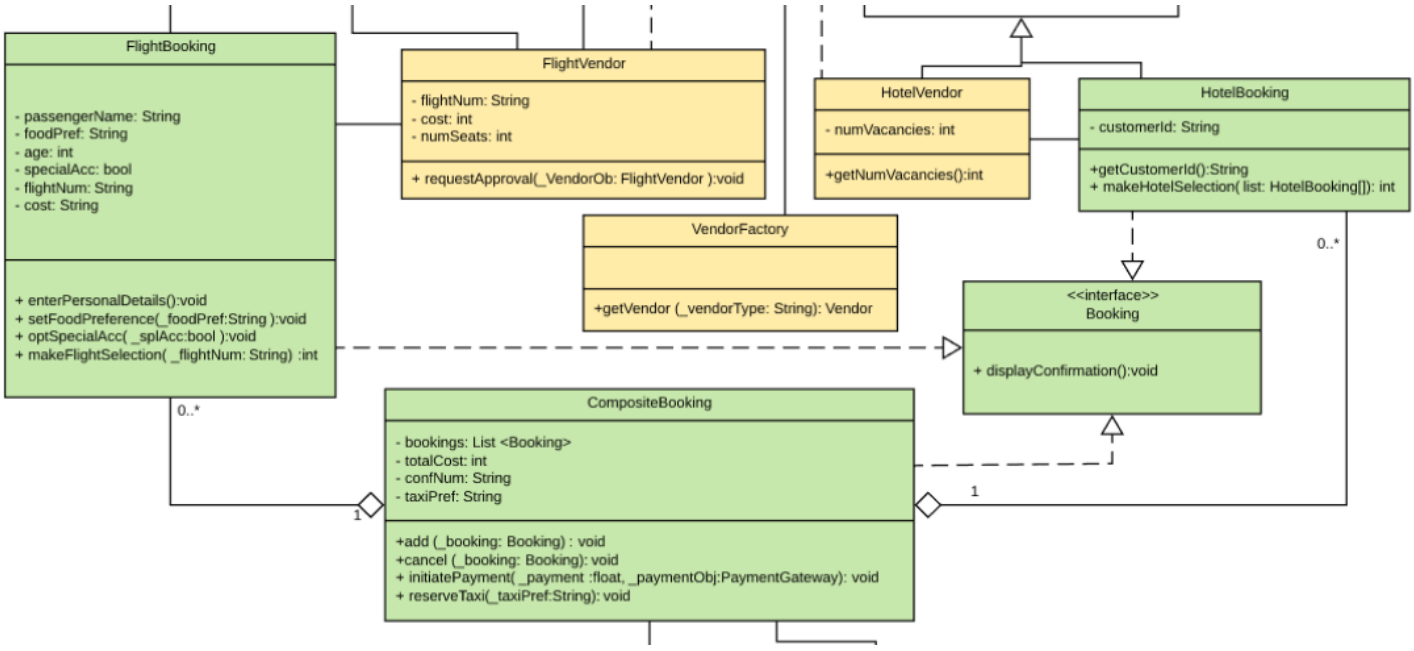
  - Portion of the class diagram implementing the design pattern



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



    - Portion of the class diagram implementing the design pattern



6. Learnings from the process of Analysis and Design –

Through this process of analysis and design in the project, we as a team have learnt a number of things, a few of which are summarized below:

- The most important learning was that of SOLID Design principles. Keeping in mind these principles during the design phase helped us realize at every stage, how our system is being more robust and extensible in its structure.
- An extension of this has been through the implementation of the Design Patterns learnt in class. After learning about the design patterns, not only did we develop a slightly different perspective towards our entire project's class diagram, but it also helped us to get the necessary hands-on experience of how few of the design patterns would be implemented for a real-world system.
- Then during the implementation phase, what we learnt was the importance of avoiding redundancy of attributes and/or methods in the classes, by inculcating Abstract classes and interfaces. Not only did it make the implementation less tedious, but it also helped us provide the necessary extensibility to add newer features on the fly.
- This project also helped us in realizing why it is important to design the classes as per the overall vision of the system and not as per the structure of the underlying database alone. We also learnt on how to avoid anti-patterns like blob classes (Although some of our classes like the FlightBooking class was comparatively large, we ensured that we are adhering to Single Responsibility while doing so)
- Finally, what we learnt from the overall system design and development was the necessity of trade-offs. At multiple stages, there was a choice of more than one design patterns to be implemented, or more than one design principles to be adopted. But we did realize steadily that this choice should be affected by all the possible scenarios making use of that part of the system, and not by a single use-case alone.