Mekai Johnson
Professor Johnson
CSCD 212
4 December 2024

# CSCD212 Airlines Project

## Builder Pattern

The Builder Pattern is implemented in the ReservationBuilder class which builds the reservation for the user. This allows us to set details such as source and destination airport codes (lines 53-73), date and time (lines 81-87), and tickets to be added to the reservation (lines 120-123). The build method at the end of the class takes all of this information together and makes it into one reservation class. It does this by looping through lists and adding the tickets and layovers set by the user. The benefits of using this design pattern are having improved readability by chaining methods together; prevention of errors because all fields must be filled before construction of a Reservation object; and extensibility because it is easy to add new components like a loyalty program without modifying the base Reservation class. A drawback to using this design pattern is that it can increase code complexity.

## Decorator Pattern

The Decorator Pattern is implemented in the CostDecorator and SeatClassDecorator classes to allow for a Ticket to get features added to it without modifying its base class. CostDecorator is an abstract class, and SeatClassDecorator adds different costs to the ticket depending on what the user picks for their seat class (lines 23-37). The benefits of this design are Open/close principle allowing for new cost modifiers; Flexibility because decorators can be stacked; and reusability because logic is decoupled. Drawbacks are increased complexity and it requires the Ticket base class to be highly extensible.

## Strategy Pattern

The Strategy Pattern is implemented with the abstract Aircraft class and its concrete subclasses CommercialAircraft and PrivateAircraft. The abstract Aircraft class outlines the general information every subclass has such as name (line 8), the number of rows (line 13), and the number of seats per row (line 18). Benefits of this design pattern include code rues through sharing common fields and methods; polymorphism by treating all aircraft types uniformly in other parts of the code; and extensibility because there can be many more types of aircraft added later on. Drawbacks include Rigidity because changes to the abstract class affect all subclasses, and this pattern also may enforce unnecessary fields such as a helicopter that has no rows.

## Considerations After Applying These Patterns

Combining these design patterns together into one program requires a lot of thought and care. The interaction between the ReservationBuilder class and Ticket class decorators must be good in order to prevent inconsistencies in data. Debugging is also very challenging.