Devin Harrison
Dr. Chris Cain
March 5, 2024

Airplane Ticket Reservation Writeup

Strategy

I implemented the strategy pattern in airplane fly behavior. The abstract plane superclass has a fly behavior interface within it that can be instantiated at the time of concrete plane class construction and swapped out at any time. There are 3 behaviors currently: Propellor, Jet, and Warp for the Millenium Falcon. The FlyBehavior interface holds the method fly(), and the 3 classes implement this interface, with each printing out something unique to the terminal. The 3 planes have default flyBehaviors but these can be swapped at any time with Plane.setFlyBehavior (line 67 in Plane). Adding a new flyBehavior is very simple, all it needs to do is implement the interface and have a void fly() method of some sort.

Builder

Builder was fairly straight-forward to implement. There is a TicketBuilder class in ticketing.classes that will construct everything on a ticket, including the complex description and decorator system, then build a final ticket using all of this information. I elected not to use a director because I didn't think there would be many different kinds of tickets we'd need to build. The builder has a particularly cool method: getFlightPath() (line 65). This method takes the coordinates of all airports on the ticket and uses the Pythagorean theorem to find the total flight distance between all airports. This is eventually passed to the ticket description when it is built.

Decorator

This is the doozy. Implementing decorator in the way I wanted to was challenging. From a coding perspective, it is fairly simple, but using OOP structure to my advantage here was challenging to think about. I wanted to have access to the outermost decorator object from within the ticketbuilder, while the ticketbuilder was itself the centermost object within the decorator pattern. I implemented this by adding a decorate method to the decorator interface (line 24 in TicketDecorator) and then adding an internal TicketDecoratorItem to the ticketbuilder class.

When decorator is called with a decorator object as an argument, the ticketbuilder first assigns its current decoratoritem to be the new decoratoritems internal item, wrapping itself in the passed in item. After this, it assigns the whole wrapped decorator item as it’s own internal decorator item, essentially making a giant loop of decorator items (line 137 in TicketBuilder). In order to make this work from instantiation, the constructor assigns the ticketbuilder to be it’s own decoratoritem initially, so it passes itself to any decorator the first time. To avoid stack overflows, the getCost() and getDescription() methods are not recursive, so they will both stop at the ticketbuilder, running through all decorator objects once. I’m very happy with this implementation.