Discussion

PlaneCreator – A Factory design pattern was used to create plane objects. This was set up so that the levels of indirection made it easy for Program.cs to add planes to a Brand, simply by calling the AddPlanes method of the relevant brand, with a plane model number and a number of planes to add as parameters.

In this case, the use of a Factory design pattern seemed appropriate because planes are, in fact, a kind of manufactured good. A specific model of plane will have unique characteristics, which one can specify in the FactoryMethod of PlaneCreator. Different planes would have certain characteristics that would have to be completed in detail, and the use of the IPlane interface can enforce this.

In the case of the Airline, establishing it as a Singleton made sense because they are unique; you would not have to “Air Canada” companies at the same time.

For tickets, the different time limits were set up using the Strategy pattern. Testing this did raise some challenges when testing the code, because it was desirable to make sure the output changed when the allotted time had passed after which a ticket could no longer be cancelled. This was accomplished by hard-coding some dates temporarily for testing purposes. The use of the strategy pattern in this cases was selected because the strategy pattern lends itself to this type of situations, when you have several items that are similar but have slightly different details in how you execute some operations.

Flights use the Composite pattern because the composite pattern allows one to have composite objects and associated primitive objects (sometimes referred to as ‘leaf’ objects). In the case of selecting flights, this pattern was therefore desirable because one might want to consider a list of alternative flights and present it to the customer or an employee helping a customer. If the customer wants to go from one starting point to an end point, the system may have to find flights that start on the desired date, then search for connections and show all possible connecting flights for the customer to choose from that would complete their journey. So the first flight in a list would be fixed, followed by alternative connections. On a long journey, this ‘tree’ of flights could become complex with many branches and sub-branches, here a simple case is used to illustrate.

\