Team Echo: Testing Document

Preface

This document includes Team Echo's Plan, Implementation and Results for Testing for the fourth phase of our CS325 project. This document was not intended to be a guide for testing other projects, but rather this document was meant to show our team's approach on testing our own project all together. The reason for this document is to be a reference for our team as well as a reference for Quality Assurance Testing. This will assist us in identifying our previous issues and decrease the work of re-testing something that has previously been tested.

Our approach for Integration Testing is using the bottom-up method by testing the individual functions first, then the class as a whole, and finally the whole software system as a whole. In this manner, we plan on using J-Unit testing for Implementation Testing as well as our Black-Box Testing.

Black-Box Testing & Border Testing

From the beginning of this project we knew our testing would first involve testing the individual units first rather than testing the product as a whole. With this in mind, we determined the bottom-up approach method was the best fit for our team during the testing process. Because of that, in this document we will start with Black-Box Testing and Border Testing (since these were also the first two testing methods our team started with).

Black-Box Testing

Step 1:

Since we started testing from the individual unit up, we started our testing with Black-Box Testing via J-Unit Tests. The first individual testing involved debugging the file input of CRATD. We tested the file input when doing the update to make sure it functioned properly. After we were sure the CRATD was able to update the Express Flights system, we then set up individual test cases for individual entry files.

Step 2:

Another section of our Black-Box testing involved testing the AgentControl methods. A few test cases include:

- Creating a new Customer [CreateCustomerAccount()]. We wanted to make sure that a new customer could be created and saved to the system.
- Managing a new/returning Customer [UpdateCustomer()]. We wanted to make sure an agent could make changes to a customer profile as needed, as well as delete/remove an existing customer if needed.
- Search for an existing Customer [CustomerLookUp()]. We wanted to make sure previously created customers were being saved to the system as well as make sure the could be found through a search.
- Creating a new Itinerary Case [CreateNewItinerary()]. We wanted to see if an agent would be able to create a flight reservation for a customer.
- Modify a Reservation [ModifyReservation()]. We wanted to make sure a flight reservation for a customer could be updated or changed.
- Cancel or Delete a Reservation [CancelReservation()]. We wanted to make sure an agent could cancel a reservation for a customer if deemed necessary.
- Search for Itinerary Cases.

Step 3

Yet another section that involved Black-Box Testing, we tested management responsibilities via the ManagerControl methods. A few examples include:

- Creating a new agent [CreateNewAgent()]. The purpose of our test of this method was to see if a new agent could be created in the system.
- Make Changes to an Agent [ModifyAgentAccount()]. We tested to see if a agent profile could be updated and to see if an agent could also be deleted from the system.
- Management Checks at Log-In to Disable Manager Usage By Agents.
- Load The Flights into the GUI. We wanted to make sure the system was being updated and that it would output for agents to see the new updates.
- The Manager is able to modify Fees [ManageFeeStructure()]. In this method, we tested to see if a fee could be modified by the manager (and only the manager).

Border Testing

Step 1:

We first started our border testing process by testing individual units (via J-Unit testing) for classes and methods that had border values. Such examples include computing cost and computing flight time. In the case of computing cost, we wanted to make sure the mathematical algorithm Cougar Path Travel wanted would guarantee an accurate output for cost return. We wanted to guarantee a low-cost return for the traveling customer! In this particular part of border testing, a few of our test cases included:

- Compute Basic Cost of Entire Flight
- Compute Airport Fees Into Flight
- Compute Included Airline Discount on Continuing Legs of Flight (with the same Airline)
- Returning Customer Discount (with the Agency)
- Return of Cheapest Flight Available (based on start-end locations)

This was mostly done through a J-Unit Test for ComputeCost().

Our team did not want to mess up this mathematical formula as it dealt with finances. Such mistakes could lead to a financial lawsuit, and for the security/protection of both our team and our customer we were rather thorough in making sure all mathematical algorithms worked as flawlessly as possible.

Step 2:

In the case of computing flight time, we wanted to be able to calculate the shortest flight time possible. A few cases for this border test included:

- Subtracting/Adding Hours for Time Zone
- Using the Distance Formula to Calculate Miles Between Airports
- Calculate Leg Flight Time and Overall Flight Time
- Calculate Flight Delay

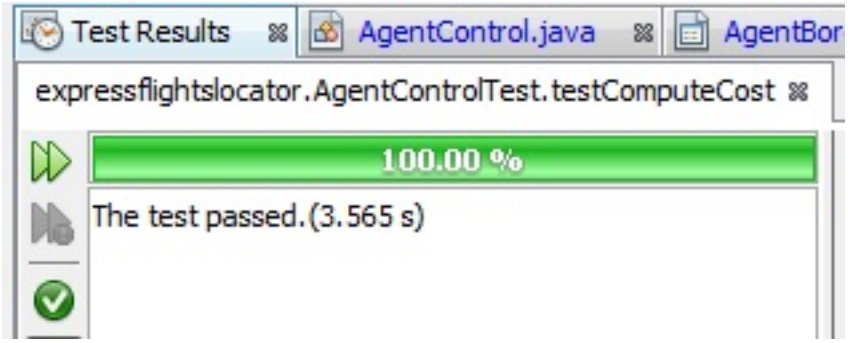
Flight-Time testing was done via J-Unit Test for the method ComputeFlightTime().

Examples of J-Unit Testing

Here are a few snapshots of how our team used J-Unit Testing for Calculating the Flight Time as well as Computing the Cost:

```
public ArrayList<ArrayList<FlightEntity>> fabricateDummyFlights(){
    ArrayList<ArrayList<FlightEntity>> flights=new ArrayList<>();
    ArrayList<FlightEntity> flight = new ArrayList<>();;
    FlightEntity leg = new FlightEntity();

    GregorianCalendar arrivalTime = new GregorianCalendar(2012,11,4,22,47);
    GregorianCalendar departureTime = new GregorianCalendar(2012,11,4,14,57);
    leg.setAirlineAbbreviation("AA");
    leg.setArrivalTime(arrivalTime);
    leg.setDepartureTime(departureTime);
    leg.setOriginAirport("ABQ");
    leg.setDestinationAirport("ATL");
    leg.setStopsDuringFlight(1);
    flight.add(leg);

    arrivalTime = new GregorianCalendar(2012,11,4,18,45);
    leg.setArrivalTime(arrivalTime);
    leg.setDestinationAirport("ORD");
    leg.setStopsDuringFlight(0);
    flight.add(leg);

    arrivalTime = new GregorianCalendar(2012,11,4,22,47);
    departureTime = new GregorianCalendar(2012,11,4,19,55);
    leg.setArrivalTime(arrivalTime);
    leg.setDepartureTime(departureTime);
    leg.setOriginAirport("ORD");
    leg.setDestinationAirport("ATL");
    flight.add(leg);

    flights.add(flight);



  public void testComputeCost() {
      System.out.println("ComputeCost");
      ArrayList<ArrayList<FlightEntity>> flights = fabricateDummyFlights();
      AgentControl instance = new ManagerControl();
      instance.getAgentBorder().setVisible(false);

      double expectedResult = 0.0;
      double result = instance.ComputeCost(null);
      assertEquals(expectedResult, result, 0.0);

      expectedResult = 367.80;
      result = instance.ComputeCost(flights.get(0));
      assertEquals(expectedResult, round(result,2), 367.80);

      expectedResult = 365.80;
      result = instance.ComputeCost(flights.get(1));
      assertEquals(expectedResult, round(result,2), 365.80);

      expectedResult = 245.67;
      result = instance.ComputeCost(flights.get(2));
      assertEquals(expectedResult, round(result,2), 245.67);

      expectedResult = 486.91;
      result = instance.ComputeCost(flights.get(3));
      assertEquals(expectedResult, round(result,2), 486.91);

      expectedResult = 174.85;
      result = instance.ComputeCost(flights.get(4));
      assertEquals(expectedResult, round(result,2), 174.85);
```

If the J-Unit Tests Passed, our end result would have an output like this:



Integration Testing

After a few individual units, mainly those in Agent Control, were functional we began to integrate the units. For the integration testing process we applied top-down method rather as our main approach. Our first step was to test the integration of the customer and the itinerary. We noticed many times that the addition of a new module would cause more unseen bugs to appear, and thus we discovered it is almost impossible to make a software program entirely bug proof. Examples of our Integration Testing include the following:

- Creating a Price Watch. We wanted to make sure the system would be able to watch the flight system prices (for up to 30 days).
- Provide Met Watches. We tested to see if the system would report if there were any met price watches. The Manager updated the system daily (which included the new Flight Prices). The system would then check existing customers on the watch list to see if their desired Flight Price was met.
- Cancel Watches. We tested to see if the system could cancel a watch and remove the customer itinerary from the system.
- Print Daily Reports. We tested to see if the system would take all the current data on the system and print it into a report. This test was to see if the data 1) was being saved on the system, 2) being updated on the system, and 3) show the manager any changes in the system as a whole.

- Print Receipts. We tested to see if the system was able to generate a receipt with both customer data and flight data. We wanted to know if the data we entered could be used with data we obtained from an external source.

White-Box Testing

For our White-Box Testing, we checked to see if our GUIs connected together as a whole. We wanted to know if the paths and connections within our system interacted as they should. Such examples might include syncing flight data from the CRATD (that the manager updated to the system) to a screen the agent could see said data. Another example of white-box testing was fault injection via alpha testing. We introduced errors for input to see if our program would accept them or not. We accomplished this via try-catch statements for such things as a value in PriceWatch. The value we desired was a digit, and if the user entered data that was not a digit then the try-catch block would handle the error (it would not be accepted).