Duncan Di Mauro

SID: 805163177

February 26, 2021

Report 5

**Obstacles:**

The main obstacle that I had with this project was figuring out how to start. The use of header files, the new notation, and understanding the concepts of classes and public/private data members was very challenging. I had to re-watch the lectures/discussions, look into the Extended Banker VS2019 code, and research online to fully understand everything. I also troubleshooted within Visual Studio itself, where Visual Studio prompted me to have the correct notation. For example, if I didn’t ever use PlaneFlight in PlaneFlight.cpp, Visual Studio would suggest adding

PlaneFlight::PlaneFlight(string passengerName, string fromCity, string toCity, double cost, double mileage)

to my PlaneFlight.cpp file. Other than these problems, I did not have any other significant obstacles. I understood how to make all the “if” statements and I understood how to increase/decrease a FrequentFlyerAccount's balance. Other than a small misunderstanding I had within CodeBoard, there were no other main obstacles.

**List of Test Data:**

We can begin the test data by including the tests provided on CCLE:

PlaneFlight shortleg( "Howard", "LAX", "LAS", 49.00, 285 );

PlaneFlight longleg( "Howard", "LAS", "NYC", 399.00, 2800 );

PlaneFlight sample( "Sample", "Sample", "Sample", 0, 1 );

FrequentFlyerAccount account( "Howard" );

assert( shortleg.getFromCity( ) == "LAX" ); //tests getFromCity

assert( shortleg.getToCity( ) == "LAS" ); //tests getToCity

assert( shortleg.getName( ) == "Howard" ); //tests getName

assert( std::to\_string( shortleg.getCost( ) ) == "49.000000" ); //tests getCost

assert( std::to\_string( shortleg.getMileage( ) )=="285.000000" );//tests getMileage

// tests account balance starting at zero

assert( std::to\_string( account.getBalance( ) ) == "0.000000" );

assert( account.getName( ) == "Howard" ); //tests getName

assert( account.canEarnFreeFlight( 3300.00 ) == false ); //tests canEarnFreeFlightt

// checking that flights add to an account balance

assert( account.addFlightToAccount( shortleg ) == true ); // tests if names match

assert( account.addFlightToAccount( longleg ) == true ); // tests if names match

assert( std::to\_string( account.getBalance( ) )=="3085.000000"); //tests getBalance

// checking that free flights reduce an account balance

if (account.canEarnFreeFlight( 285 ))

{

assert( account.freeFlight( "LAS", "LAX", 285, sample ) == true );

// Checking that Howard was able to earn a free flight

assert( sample.getName( ) == "Howard" ); //tests that a FreeFlight was created

with the name on the account

assert( std::to\_string( sample.getCost( ) ) == "0.000000" ); //tests that

the flight is free

assert( sample.getFromCity( ) == "LAS" ); //tests that a FreeFlight was made

with the correct FromCity

assert( sample.getToCity( ) == "LAX" ); //tests that a FreeFlight was made with

the correct ToCity

assert( std::to\_string( sample.getMileage( ) ) == "285.000000" ); //the mileage

of the FreeFlight is correct

// account has been reduced for this free flight...

assert( std::to\_string( account.getBalance( ) ) == "2800.000000" ); //checks

that the milage of the FreeFlight was reduced from the acccount

}

else

{

assert( false ); // there are enough miles in the account...

}

// checks that non-matching names are ignored

PlaneFlight muffin( "Muffin", "LAX", "Doggie Heaven", 500, 500 );

assert( account.addFlightToAccount( muffin ) == false );

assert( std::to\_string( account.getBalance( ) ) == "2800.000000" );

cout << "all tests passed!" << endl;

return( 0 );

}

Next, we can use the following test data to check our code:

Check all flight getters:

PlaneFlight f( "Howard", "LAX", "LAS", 65.00, 285 );

assert( f.getName() == "Howard" );

assert( f.getFromCity() == "LAX" );

assert( f.getToCity() == "LAS" );

assert( f.getCost() == 65 );

assert( f.getMileage() == 285 );

Check all flight setters:

PlaneFlight f( "Howard", "LAX", "LAS", 65.00, 285 );

f.setName( "Pixie" );

f.setFromCity( "CHI" );

f.setToCity( "BOS" );

f.setCost( 100 );

f.setMileage( 100 );

assert( f.getName() == "Pixie" );

assert( f.getFromCity() == "CHI" );

assert( f.getToCity() == "BOS" );

assert( f.getCost() == 100 );

assert( f.getMileage() == 100 );

Check name cannot be empty:

PlaneFlight f( "Howard", "LAX", "LAS", 65.00, 285 );

assert( f.setName( "" );

assert( f.getName() == "Howard" );

Check FromCity cannot match ToCity:

PlaneFlight f( "Howard", "LAX", "LAS", 65.00, 285 );

f.setToCity( "LAX" );

f.setFromCity( "LAS" );

assert( f.getToCity() == "LAS" );

assert( f.getFromCity( ) == "LAX" );

Check that mileage cannot be 0:

PlaneFlight f( "Howard", "LAX", "LAS", 65.00, 285 );

f.setMileage( 0 );

assert( f.getMileage() == -1 );

Check that mileage cannot be less than 0:

PlaneFlight f( "Howard", "LAX", "LAS", 65.00, 285 );

f.setMileage( -50 );

assert( f.getMileage() == -1 );

Check that cost can be equal to 0:

PlaneFlight f( "Howard", "LAX", "LAS", 65.00, 285 );

f.setCost( 0 );

assert( f.getCost() == 0 );

Check that cost cannot be less than 0:

PlaneFlight f( "Howard", "LAX", "LAS", 65.00, 285 );

f.setCost( -50 );

assert( f.getCost() == -1 );

Check that you cannot earn a free flight with a balance of 0:

FrequentFlyerAccount account( "Howard" );

assert( account.canEarnFreeFlight( 100 ) == false );

Check that you cannot earn miles on a flight with a non-matching name:

PlaneFlight f( "Pixie", "LAX", "LAS", 65.00, 285 );

FrequentFlyerAccount account( "Howard" );

account.addFlightToAccount( f );

assert( account.getName() == "Howard" );

assert( std::to\_string( account.getBalance() ) == "0.000000" );

Check that a flight updates the balance:

PlaneFlight f( "Howard", "LAX", "LAS", 65.00, 285 );

FrequentFlyerAccount account( "Howard" );

account.addFlightToAccount( f );

assert( account.getName() == "Howard" );

assert( std::to\_string( account.getBalance() ) == "285.000000" );

Check that a FreeFlight reduces the balance:

PlaneFlight f( "Howard", "LAX", "LAS", 65.00, 285 );

FrequentFlyerAccount account( "Howard" );

account.addFlightToAccount( f );

assert( account.getName() == "Howard" );

assert( std::to\_string( account.getBalance() ) == "285.000000" );

account.addFlightToAccount( f );assert( account.canEarnFreeFlight( 285 ) == true );account.freeFlight( "LAS", "LAX", 285, f );

assert( f.getName() == "Howard" );

assert( f.getFromCity() == "LAS" );

assert( f.getToCity() == "LAX" );

assert( f.getCost() == 0 );

assert( f.getMileage() == 285 );

Check that you cannot earn a free flight without enough miles:

PlaneFlight f( "Howard", "LAX", "LAS", 65.00, 285 );

FrequentFlyerAccount account( "Howard" );

account.addFlightToAccount( f );

assert( account.canEarnFreeFlight( 1000 ) == false );

Test that a free flight doesn’t earn miles in a flyer account:

PlaneFlight f( "Howard", "LAX", "LAS", 65.00, 285 );

FrequentFlyerAccount account( "Howard" );

assert( account.getName() == "Howard" );

assert( std::to\_string( account.getBalance() ) == "285.000000" );

account.addFlightToAccount( f );assert( account.canEarnFreeFlight( 285 ) );account.freeFlight( "LAS", "LAX", 285, f );

assert( f.getName() == "Howard" );

assert( f.getFromCity() == "LAS" );

assert( f.getToCity() == "LAX" );

assert( f.getCost() == 0 );

assert( f.getMileage() == 285 );

account.addFlightToAccount( f );

assert( account.getBalance( ) == 0 );