Capital On Tap Code Test

Introduction

We are passionate about creating high quality software to aid our business in its everyday needs. Our software is used extensively, both internally and externally, and it must stand up to on-going heavy use and constantly evolving requirements. We also need people who can explain to colleagues what they have done and why.

The kind of things we value highly:

- A clear focus on quality, maintainable code
- Appropriate consideration to securing both existing and new functionality via automated testing
- A passion to learn the right ways of doing things
- Appropriately commented code
- An acceptance that keeping things simple does not necessarily mean sticking with the first solution that first presents itself
- The ability to keep code extensible without violating the YAGNI principle (You aren't going to need it)
- An awareness of SOLID and DRY principles
- Where appropriate adhering to best practices driven by the wider software community
- A disciplined approach to working with software honed through practice
- Knowing when the code is good enough to move on with further functionality
- An ability and willingness to seek help when needed

As part of the assessment process, we provide technical tests such as this to help us gauge if you see things as we do.

The details of the test are set out below. Following the details of the test is a section explaining where to find the source code to start your solution from, and an explanation of how to submit that solution.

Test Details

A small airline wants you to help improve their flight booking service. They currently enter passenger details and then get a summary report on whether a scheduled flight is allowed to proceed or not.

The programmer who originally developed the system has since left and a consultant they recently had look at it has recommended that in order to add more features the system should be improved first.

Your job is to add the following features, making improvements to the existing code (refactoring and commenting) where you can, while keeping the code extensible to allow for future enhancements.

System description

There are three types of passengers currently catered for:

General – normal fare paying passengers
Loyalty Members – repeat customers who get benefits for choosing to fly with the airline
Airline Employee – employees of the airline who fly with the airline on a free basis as a perk

For each flight the airline charges a base price for a specific route however the loyalty members can choose to pay with their loyalty points instead. Airline employees always fly free.

Loyalty points are given to loyalty members for certain routes. The amount depends on the route.
Baggage allowance is allocated on a basis of 1 bag allowed for everyone and 1 extra bag allowed for a loyalty member.

Below is an example of sample input passed to the system:

```
add general Steve 30
add general Mark 12
add general James 36
add general Jane 32
add loyalty John 29 1000 true
add loyalty Sarah 45 1250 false
add loyalty Jack 60 50 false
add airline Trevor 47
add general Alan 34
add general Suzy 21
print summary
```

```
Flight summary for London to Paris
Total passengers: 10
General sales: 6
Loyalty member sales: 3
Airline employee comps: 1
Total expected baggage: 13
Total revenue from flight: 800
Total costs from flight: 500
Flight generating profit of: 300
Total loyalty points given away: 10
Total loyalty points redeemed: 100
THIS FLIGHT MAY PROCEED
```

Required features

- The airline would like to add a new type of passenger – **Discounted**. This passenger type does not accrue loyalty points, is not allowed a bag, but is only charged half the normal base price for the route. They would like this type of passenger reflected on the summary report in the same way the other passenger types are reported.

- Currently the system has some business rules for deciding whether the flight can proceed. These are:

    - The revenue generated from the flight must exceed the cost of the flight
    - The number of passengers cannot exceed the amount of seats on the plane
    - The aircraft must have a minimum percentage of passengers booked for that route

  The business would like to keep these default rules but also have an option to choose a different set where they relax the requirement for the revenue generated exceeding the cost only if the amount of airline employees aboard is greater than the minimum percentage of passengers required. They want the option to switch between these rules sets. They have indicated they might want more rule sets in the future.

- Often the airline finds that they overbook flights and are then stuck. As they have a range of aircraft they would like the summary report to list any aircraft which would be able to fly instead. For example:

```
THIS FLIGHT MAY NOT PROCEED.
Other more suitable aircraft are:
Bombardier Q400 could handle this flight.
ATR 640 could handle this flight.
```

  If there are no other planes that could handle the flight then you don't need to list anything extra. You can decide how you make this information available to the summary report.

Please note:

It is very important to the airline that you do not break any of the existing functionality! In other words given the same passenger data is entered they would like to see the same data present on the summary report (along with any additional information).

The airline has let us know that the `FlightRoute` and `Plane` classes cannot change as other systems depend on them (but they can be refactored and commented).

There is a sample console application which demonstrates how a user might interact with the flight booking system. Feel free to update this but note that a separate UI is not required.

Submission Guidelines

The starting point for your solution will be the source code enclosed in the zip file. In particular, the primary code is location in the `FlightBooking.Core` project and in the `ScheduledFlight` class.

We make the assumption that the problem will be completed using a version of Visual Studio. We realise that not everyone has access to a full version of Visual Studio and our tests can just as easily be completed using the Community Edition of Visual Studio.

The current version of Visual Studio we are using is Visual Studio 2017 but we also accept submissions created with earlier versions of Visual Studio.

When submitting completed practical code tests please compress the source files and any other supporting files and then forward them on to us.

Please make sure you remove all binary files before compressing (so there are no .exe files etc). If you do not do this the file may get blocked by our firewall.

We believe a good enough solution can be created within 2-3 hours of starting and believe well-motivated individuals will be able to create the time needed within the turn-around window.