ASU W. P. CAREY SCHOOL of BUSINESS
ARIZONA STATE UNIVERSITY

CIS 345 – Business Information Systems Development II – Fall 2014

Object-oriented Project: Charms Flight Management System

Due Date: Sunday, Dec 7, 10 PM to Blackboard

SCENARIO

While considering merger talks with the South Pacific Airways, Chandler Air is planning a combined electronic passenger management system: Chandler Air Reporting and Management System (CHARMS). CHARMS will use state of the art technology to manage flight information, searchable passenger manifests and will also be able to interface with the U.S. Department of Homeland Security's division of Transportation Security Administration (TSA) to submit passenger lists for security screening.

Before developing a full system, Chandler Air has decided to commission a prototype from group of talented CIS students studying object-oriented design. Having a prototype would give Chandler Air the ability to see both possibilities and shortcomings, refine its own requirements and expectations, and then go to software consultants. Management also hopes that interfacing with TSA will expose both technological as well as security issues.

Chandler Air tracks flights that are routed across their entire system. Each Flight has a flight number, origin and destination. CHARMS would like to be able to track up to 20 flights in its prototype. For each flight, Chandler Air tracks a list of passengers, known as the Manifest.

Since airlines are subject to TSA regulations regarding carrying passengers, it has chosen to cooperate with TSA in letting them manage the structure of information relating to passengers. Information like first name, last name, loyalty program number, and security flag has already been built into TSA's software systems for tracking passengers. TSA's technology unit provides a

binary file that can be used to create Passenger objects in line with the structure agreed upon by all airlines as well as TSA.

Chandler Air would like the CHARMS prototype to be capable of displaying a list of existing flights, create new flights, search all flights for a particular passenger and be able to manage flights on an individual basis.

It envisions a flight sub-menu, which will let Chandler Air employees edit existing flights (for flight number, origin, or destination), display passenger manifests, add new passengers to an existing flight, and submit the entire passenger manifest for a particular flight to TSA.

TSA provides advanced Windows .NET Dynamic Link Libraries (DLL) that can be used to submit passenger manifests in an agreed-upon structure (something called an “array” of Passenger objects). TSA returns another list of flagged passengers back (an array of FlaggedPassenger objects). CHARMS is supposed to go through the entire returned list and match every name from the returned list to its passenger manifest using first and last names. If the first and last names are a match, CHARMS is supposed to flag the passenger within its manifest. Next time, when an employee views the manifest, those flagged passengers will readily be visible and he or she can take appropriate action according to their security protocols.

The operation above is carried out on an individual flight basis only. When TSA returns a list of FlaggedPasssengers, that list is applicable only to the flight in question. It does not require that CHARMS go through passenger manifests on all flights.

It is possible that TSA returns a few or possibly no passengers as flagged. As meticulous programmers, the CHARMS development team will have to be careful in going through the returned list so that it does not bring down the entire CHARMS system and cause national pandemonium! If the entire system is brought down, it cancels all flights across the country: a very bad scenario for the development team.

Chandler Air hopes that their talented designers will come up with a sophisticated design that would be reusable and extensible in the years to come!

REQUIREMENTS

1) General Requirements

- The project needs to be implemented in Visual C# 2012/13 using .Net Framework 4.5.
- Use object-oriented design: have classes, methods and utilize instance properties and methods.
    - There should be little to no use of static variables and methods.
- Submit the project to Blackboard as a zip file, named in the format: "LastNames-Project.zip." All students must submit the project individually.
- Use variable names that are descriptive and use Pascal and Camel case as appropriate.
- Comment code.

2) Feature Requirements

A sample executable file is provided on Blackboard illustrating the major functionality:

- A main menu that allows employees to:

    1) Display a list of flights

    2) Create new flights

    3) Search all flights for a particular passenger by first and last name.

    4) Select a flight

    5) Cleanly exit the application

- A flight sub-menu that allows employees to:

    1) Display flight information for the selected flight

    2) Display Passenger Manifest

    3) Edit flight information

    4) Add new passengers

    5) Submit the passenger manifest to TSA and update the manifest based on the returned list of flagged passengers.

    6) Exit the sub menu.

3) Technical Requirements:

- The project should use the HomelandSecurity DLL file that has been provided and ***use the Passenger class available within it.***
    - Copy the provided DLL and the XML file to your project debug\bin folder.
    - In order to utilize the DLL, the file needs to be added as a reference within the Visual Studio solution.
- The project should create a TSA object and call its RunSecurityCheck method. It should supply the method with an array of Passenger objects and the count of how many passengers are in the array. It should then store and process the returned array of FlaggedPassenger objects.
    - It should update the boolean flag within the Passenger objects in the manifest based on the FlaggedPassenger objects (see hints section).
- The method to display the passenger manifest should indicate that a passenger is flagged if the boolean flag in the passenger object is set to true.
- It should have at least one class to manage a Flight and one class to manage the overall CHARMS system. You are free to have more classes.
- It should have a method that loads sample flight and passenger data at startup.

DELIVERABLES

- A working Visual Studio 2012/13 Project (submitted to Blackboard as a zip file).
    - The submission is expected on-time and on Blackboard. If there is no file submitted <u>on time on Blackboard</u>, or if your file cannot be opened or is corrupt in any way, <u>it will receive a score of zero</u>. *Check your files three times AND run your project on Citrix before submission. If you do not check your files, be prepared to do that at the risk of receiving a zero.*
- A mandatory one-on-one Question & Answer session on Final Exam Day.
    - You will be asked conceptual questions as well as questions about the code and possibly be asked to modify code. You should bring your completed project on the final exam day on your own computer or have it accessible and running on the school computer.

OPTIONAL ENHANCEMENTS AND CHALLENGES

- *Logic and Program Features:* Add the ability to delete a passenger from the manifest. If you delete a passenger in the middle of the array, move the rest of the passengers up the list.
- *C# .NET Library:* Learn how to use the StreamWriter class to write to a text file. Write the contents of the flight arrays and manifest arrays to a text file. Use the StreamReader class to load data from the text file. Thus, implement data persistence by providing save and load features.
- *Implement a Computer Science algorithm*: Sort the passengers by name by looking up and implementing the bubble sort algorithm.
- *Logic and Program Features:* Add a waitlist to the manifest and allow the users to adjust the waitlist so that passengers can "move up the list" and onto the manifest.
- *Additional Classes:* Implement another FrequentFlier class that bumps people off or inserts them into the manifest based on airline miles. Implement a miles/frequent flier status property for passengers by utilizing inheritance.

GUIs

You *may* add a Graphical User Interface (GUI) with the following caution in mind: ***it is considerably difficult to manage multiple forms and objects within multiple forms. You should not attempt it unless you are somewhere in the high 'A' or 'A+' range for your grade and the basic application itself would be relatively easy for you i.e. you can afford to spend time on the GUI since you do not need to spend extensive time on implementing object and array logic.*** Exchanging data between multiple forms is sophisticated. Static variable usage is not an option since that circumvents encapsulation but more importantly, you need instance properties for passengers since each flight has its own copy of passengers.

I can provide you with a basic structure (as well as sample C# application!) of how data can be exchanged properly between forms using event handling (by implementing custom event handlers, delegates, derived classes for event arguments, and polymorphism). If you are interested, come by my office since I will need to show you how it all works.

HINTS AND RECOMMENDATIONS

- A sample executable is provided to you; however, you are not restricted to having your program limited to that sample either in terms of appearance or functionality. The sample implements the minimum functionality required. Enhance the program; have fun!
- You have a fair degree of flexibility in modeling your project design, implementing methods, etc.
- Use In-class9 and Assignment 6 as your template in getting started.
- The arrays for objects should always be outside the object. You will not be able to have a static array inside the Flight class, because each flight has a different list of passengers. Your arrays will be instance properties.
- Have a DisplayMenu method in your main class. Implement your menu calls using a do-while sentinel-controlled loop. Do not call your DisplayMenu method over and over from other methods.
    - Similarly, have a menu method within your Flight class. Again, use do-while and don't call the method over and over from other methods.
- Implement the SecurityCheck feature at the very end after all other functionality is working: it is the most sophisticated application feature and nothing in the rest of the project depends on it.
    - For updating the manifest after the security check, you will need to go through all FlaggedPassengers and the manifest for a particular flight. Use nested loops – outer loop for FlaggedPassengers and inner loop for the manifest. If the flagged passenger names are equal to the manifest names, set the boolean in your Manifest to true.
- If you find yourself repeating code, try to have it structured such that it is in one method and you should call that method again rather than type redundant code.
- You might be interested in having a Utilities class, in which you could store common utility methods that read numbers, handle exceptions, insert pauses/"Press any key" statements, etc.
    - Create static methods, ReadInteger, ReadDouble, Pause, etc. Do try-catch handling within these methods, so that the vast majority of your code remains clean and easier to implement as well as read. *Re-use the class/method you wrote for In-class10.*
    - Create one method to write the header of the application and reuse it on every screen.
    - Place here the WriteCenteredLine method you were asked to write in Assignment 3. If you didn't write it, make your life easier: write it now!
- Come for help!

PROJECT TEAMS

- This project can be completed alone or in teams of two.
- If you work in a team, you need to be paired with somebody of roughly the same expertise-level, so that there is no free-rider problem and so that one person does not end up doing all the work.
- Teams will need to be approved. Teams will not be approved if the two students vary substantially in class performance and grades, e.g., students averaging an 'A' will likely not be approved to team up with a student averaging a 'C'.
    - This is to maximize the chance of work being distributed equitably and to ensure that both team members have a good chance of learning by doing as well as supporting each other.
- There will be a peer evaluation process for those working in teams. A team member who did not contribute equally might not receive the same level of points.
- Either or both team members can be asked to explain the code in detail. Do not rely on your team member to do all the work. If you do so, and are unable to properly answer questions and/or reproduce the code when asked, you will not get the same level of credit as the other member.

GETTING HELP

Skill levels vary and while some people may get done much quicker, you should expect to spend at least 20-30 hours on this project. If you start early, you can get help early. You are welcome to come to me for help in approaching the project, solving issues, implementing specific portions, or debugging code. I am here to help you in all aspects of implementing the project! You only need to make sure that you start early enough so that you can get help. Help does need to be provided in person. Very little help can be provided over e-mail since it needs to be done interactively.

You should also to go the Tutor for help. The tutor also helps CIS 340 students, so if you wait till the end, getting assistance will be very tough due to capacity constraints.

Do NOT wait till the end of semester to get started. Do not struggle alone! ***Get started early, get stuck early, get help early!***

PROJECT ORIGINALITY

Any work submitted must be original and represent your own effort. You must not submit code that has not been written by you or your team members or that you cannot together explain conceptually and cannot replicate again in C#. As per the syllabus, you can be asked to explain any portion of the code you submit for credit.

You will be asked to explain code briefly during the final exam session and may be asked to come in later and explain/rewrite/fix C# code in detail in the presence of a W. P. Carey staff or faculty member. If you are not available to come in, you will receive a grade of Incomplete (I), which will be changed only after you come in.

Project originality will also be checked using Artificial Intelligence software pattern detection software. This software is able to detect duplicate code despite name changes, variable changes, dead code, blank lines, writelines, or any other device that makes the code look different but still behave in the same manner. If your project is flagged, you may be called in to explain/rewrite/fix C# code in the presence of a W. P. Carey staff or faculty member as detailed above. Your project will also be forwarded to the W. P. Carey Academic Affairs office.

If you are not able to sufficiently explain or replicate the code that you submitted as your own work, a grade of zero can be given for the project and academic dishonesty proceedings may be initiated resulting in harsher penalties, such as a grade of E or XE, as stated in the syllabus and ASU policy documents. XE grades are assigned for failure due to academic dishonesty and remain on student transcripts permanently, even after the course has been repeated successfully.

All duplicate projects will be considered as instances of cheating – both “original” and “copied” projects will receive the same penalty.

Academic Integrity violations will be documented with W. P. Carey’s Academic Affairs Office. The College will pursue additional actions and penalties above and beyond class penalties such as removal from the major and from the W. P. Carey School of Business. After you finish this class, your project will still be run against projects submitted in the future. If a match is found, it will be reported and penalties will still apply.

Summary: Don’t submit code you haven’t written on your own. Don’t give your code to others.