**C# OOP Advanced Retake Exam – Travel**

After the biggest music festival in Bulgaria ended, Bay Kiro and his chalga associates found themselves out of a job, so Kiro founded the *United**Runways**–* *an* ***airline*** *with a twist™*.

The twist is that **every flight is free**! What’s the business model then, you ask? Well, **baggage** **confiscation**. The thing is, the terms of service state that **any baggage with a combined value of over $3000** gets confiscated by the airline – but nobody reads those, right?! Exactly! It’s practically free money!

Well, that’s only **half** of the business model… the other half is a classic **overbooking** scheme. A flight allows an unlimited number of people to board it, but when the time for **taking off** comes, **random passengers** get **deplaned** (and their baggage confiscated, of course).

**Overview**

Your task is to build a software project, which manages an **airline**. This airline will have **trips**, which have **source** and **destination cities**, andtheir own **airplane**. Each passenger has **checked** bags and **carry-on** bags, which they explicitly declare at check-in.

**Task I: Structure**

Unfortunately, the previous maintainer of the airline code was doing a terrible job. All he managed to do correctly was **create all the interfaces** and the **FlightController**. **Do not modify the interfaces or their namespaces!**

The main structure of the program should include the following elements:

* **Engine** – **Processes input commands** and **sends them** to the relevant **controllers** to handle
* **AirportController** – **Handles** all the **commands** listed in the **I/O** section
* **FlightController** – Responsible for **performing the trip** (loading luggage into the airplane, deboarding passengers in case the flight is overbooked, etc.)
* **Airport** – **Repository**, which **holds** all the **trips**, **passengers**, **confiscated luggage** and **checked** **luggage**, and provides **methods** for **retrieving** and **storing** them.

**Guidelines**

* Make sure your Visual Studio is up-to-date. If it isn’t, you risk a bug where **the .NET Core skeleton doesn’t build** (and has red squiggles everywhere).
* Upload **only the Travel** project in every problem **except** **03. Unit Tests**. For **03. Unit Tests**, upload **only the Travel.Tests project** with **all using statements pointing to FestivalManager** **removed (leave any System using statements alone)**.
* **Do not modify any of the interfaces or their namespaces!**
* You will have to refactor everything else as you see fit. Use **strong cohesion** and **loose coupling**.
* **Use inheritance and the provided interfaces wherever possible**. This includes **constructors**, **method parameters** and **return types**!
* **Do not** violate your **interface** **implementations** by adding **more public methods** or **properties** in the concrete class than the interface has defined!
* **Do not modify any constructors**.
* Make sure you have **no public fields** anywhere.

Below, you will find a detailed description of all entities and their methods.

**Airplanes**

An abstract **airplane** has the following characteristics:

* **int** **Seats**
* **int** **BaggageCompartments**
* **IReadOnlyCollection** **BaggageCompartment** – the **carry-on bags** in the plane
* **IReadOnlyCollection** **Passengers**
* **bool** **IsOverbooked** – **true** if the passengers count is **greater than** the **Seats** (calculated property).

Airplanes have several **methods**, most of which are self-explanatory.

* **AddPassenger** – **adds** the provided passenger into the plane.
* **RemovePassenger** – **removes** the passenger at **the provided seat index**.
* **EjectPassengerBags** – Finds all the baggage, with the provided owner, **removes it from the plane** and **returns a list** of all the **removed baggage**.
* **LoadBag** – Adds the provided bag into the **BaggageCompartment**. If the number of bags inside the plane is **larger than** the number of **baggage compartments**, throw an **InvalidOperationException** with the message "**No more bag room in {planeName}!**"

Create **two** **classes** for each of the three **types** of airplanes we have:

* **LightAirplane** – **5** seats, **8** baggage compartments
* **MediumAirplane** – **10** seats, **14** baggage compartments

**Items**

An abstract **item** has the following characteristics:

* **int** **Value** – the **monetary** **value** of the item in **USD**

Create **six** **classes** for each of the three **types** of items we have:

* **Toothbrush** – **$3 value**
* **TravelKit** – **$30 value**
* **Jewelery** – **$300 value**
* **CellPhone** – **$700 value**
* **Laptop** – **$3000 value**
* **Colombian** – **$50000 value**

**Bag**

A bag is a container of **items**. It has the following characteristics:

* **IPassenger** **Owner**
* **IReadOnlyCollection<IItem>** **Items**

The bag receives its **owner** and **items** from the constructor, so they **cannot be altered**.

**Passenger**

A passenger has the following characteristics:

* **string** **Username**
* **IList<IBag>** **Bags**

**Trip**

A trip has the following characteristics:

* **string** **Id** – The **combination** of the **source** and **destination** cities, plus an ever-increasing **integer** (starting at **1**)
* **string** **Source**
* **string** **Destination**
* **bool** **IsCompleted**
* **IAirplane** **Airplane**

Trips only have **one method**: **Complete()**. I’ll let you figure out what that does by yourself…

**Airport**

The **airport** class is a **repository** where all the current **passengers**, **trips**, **checked baggage** and **confiscated** **baggage** are **stored**. It has:

* Methods for **Adding passengers**, **trips** and **checked/confiscated bags**
* Methods for retrieving **passengers** by **name** and trips by **id**

**Note**: The **Airport** is **shared** between the **airport controller** and the **flight controller**.

**Task II: Business Logic**

Your code should only catch exceptions on the **engine level**.

**Commands**

The software needs to be able to process several commands in the form of **methods**:

**RegisterPassenger {username}**

If the airport already has a passenger with that username, throw an **InvalidOperationException** with the message "**Passenger {username} already registered!**".

The command adds a new passenger into the **airport** and returns "**Registered {passenger.Username}**"

**RegisterTrip {source} {destination} {planeType}**

Creates a trip with that **source** and **destination** and **adds it** to the **airport**. The Id is auto-generated from the **Trip** class itself.

The command returns "**Registered trip {tripId}**".

**RegisterBag {username} {bagItem1} {bagItem2} {bagItemN}**

Gets a passenger with the provided **username** from the airport. Then, creates a **bag** with all the provided **items** and adds it to the **passenger’s bags**.

The command returns "**Registered bag with item1, item2, itemN for {username}**"

**CheckIn {username} {tripId} {bagIndex1}, {bagIndex2}, {bagIndexN}**

Gets a **passenger** with the provided **username** and a **trip** with the provided **id**.

If the passenger has **already checked in** (is already in any trips’ airplanes), throw an **InvalidOperationException** with the message **"{username} is already checked in!"**.

Then, the command **checks in all the passenger bags** with that **index**.

Checking in works like this:

The bag with that index gets removed from the passenger’s bags. Then, depending on whether the bag should be **confiscated**, one of the following things happens:

If it should be **confiscated** (if the **total value** of the bag is **over $3000**), the bag is **added to the airport’s confiscated bags**. If not, the bag gets added to the airport’s **checked bags**. Any other bags, whose indices are not listed in the command input are **left with the passenger** (and eventually board the plane along with the passenger).

After checking in any bags, the **passenger** is **added** to the **trip**.

The command returns **"Checked in {username} with {bagsToCheckInCount-confiscatedBagsCount}/{bagsToCheckInCount} checked in bags"**.

**TakeOff**

For every trip, the following is performed:

**"{trip.Id}:"** is appended to the output.

If the trip is **already completed** (the command could be called several times), the trip is **skipped** entirely and nothing is done. If the trip has **not been completed yet**, all the **carry-on baggage** is **loaded** onto the **plane**.

Then, if the plane is **overbooked**, **passengers at random seats (indices) start getting ejected until the plane isn’t overbooked anymore**. Also, all ejected passengers’ **carry-on baggage** is automatically **confiscated**. The **random generator seed** is **always the number 1337**. After the bags are confiscated, their **total value** is calculated, and **"Confiscated {confiscatedBagsCount} bags (${bagsTotalValue})"** is appended to the output of the command.

Then, the trip is completed and **"Successfully transported {airplanePassengersCount} passengers from {source} to {destination}."** is appended to the output of the command.

After all trips are completed, **"Confiscated bags: {totalConfiscatedBagsCount} ({totalConfiscatedItemsCount} items) => ${totalConfiscatedItemsValue}"** is appended to the output of the command.

**Tasks III:**

**Reflection**

You need to refactor the given factories and implement new ones. Factories must **use reflection**, so it will be easy for us to follow the **Open/Closed Principle**. You are required to implement **two factories**:

* **AirplaneFactory**
* **ItemFactory**

Feel free to make additional factories for the other entities, even though they will not be tested.

Your task is to implement these factories in such a way that it will be **easy to extend the number of concrete types of each entity**.

**NOTE:** **Make sure you reference the Calling Assembly, instead of the Executing Assembly, since the code that’s going to be calling your factories in the tests depends on this!**

**No static** factoriesareallowed**!**

**Unit Testing**

Like you saw at the beginning, there is а class, which does not need refactoring - **FlightController.** This is the class, against which you need to **write unit tests**. In your skeleton, you are provided with a **perfectly working** **FlightController**, but it still needs to be **tested**, because in **Judge**, we have prepared some **bugs**, and you need to catch them in your unit tests.

You are provided with a **unit test project** in the **project skeleton**. **DO NOT modify its NuGet packages**.

Note: The **FlightController** you need to test is in the **global namespace**, as are any entities, which it depends on.

**Remove any using statements** **pointing towards the Travel namespace** before submitting your code.

Do not use the **AirportController or any factories** in your tests, as it’s not a part of the classes you are provided with. You are only provided the **FlightController** **and its direct dependencies**.

Do **NOT** use **Mocking** in your unit tests!

**Input**

* The input will come from the console in the form of commands, in the format specified above - each command on new line
* The input sequence ends when you receive the command “**END**”
* **Any** type of command, except “**END**” can be given at any time.

**Output**

The **output** of each command must be printed **on a new line**.

If an exception is thrown because of invalid state, they should be printed in the following format: “**ERROR: {exceptionMessage}**”.

After the “**END**” command is received, end the program.

**Constraints**

* All input lines will be **valid** commands with **valid** arguments.
* There will be **at most** **50** commands
* **All** rules **specified above** will be **strictly** **followed**, there will be **NO** unexpected input or conditions

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| RegisterTrip Sofia London LightAirplane  RegisterPassenger Pesho  RegisterPassenger Pesho  RegisterPassenger Ivan  RegisterBag Pesho Toothbrush Jewelery  RegisterBag Pesho CellPhone Laptop  CheckIn Pesho SofiaLondon1 0  TakeOff  END | Registered trip SofiaLondon1  Registered Pesho  ERROR: Passenger Pesho already registered!  Registered Ivan  Registered bag with Toothbrush, Jewelery for Pesho  Registered bag with CellPhone, Laptop for Pesho  Checked in Pesho with 1/1 checked in bags  SofiaLondon1:  Successfully transported 1 passengers from Sofia to London.  Confiscated bags: 0 (0 items) => $0 |
| RegisterTrip Sofia London LightAirplane  RegisterPassenger Pesho  RegisterBag Pesho Laptop Laptop Laptop Laptop Laptop Laptop  CheckIn Pesho SofiaLondon1  CheckIn Pesho SofiaLondon1  TakeOff  END | Registered trip SofiaLondon1  Registered Pesho  Registered bag with Laptop, Laptop, Laptop, Laptop, Laptop, Laptop for Pesho  Checked in Pesho with 0/0 checked in bags  ERROR: Pesho is already checked in!  SofiaLondon1:  Successfully transported 1 passengers from Sofia to London.  Confiscated bags: 0 (0 items) => $0 |
| RegisterTrip Sofia London LightAirplane  RegisterTrip London Sofia MediumAirplane  RegisterPassenger Pesho  RegisterPassenger Pesho  RegisterPassenger Gosho  RegisterPassenger Ivan  RegisterPassenger Penka  RegisterPassenger Boiko  RegisterPassenger Goshko  RegisterBag Pesho Toothbrush Jewelery CellPhone  RegisterBag Gosho Toothbrush Jewelery Laptop  RegisterBag Gosho Colombian  RegisterBag Ivan Toothbrush TravelKit CellPhone  RegisterBag Penka Toothbrush Jewelery CellPhone  CheckIn Pesho SofiaLondon1 0  CheckIn Pesho SofiaLondon1  CheckIn Gosho SofiaLondon1 1  CheckIn Ivan SofiaLondon1 0  CheckIn Penka SofiaLondon1 0  CheckIn Boiko SofiaLondon1  CheckIn Goshko SofiaLondon1  TakeOff  RegisterPassenger Pesho1  RegisterPassenger Gosho1  RegisterPassenger Ivan1  RegisterPassenger Penka1  RegisterPassenger Boiko1  RegisterPassenger Goshko1  RegisterBag Pesho1 Toothbrush Jewelery CellPhone  RegisterBag Gosho1 Toothbrush Jewelery Laptop  RegisterBag Gosho1 Colombian  RegisterBag Ivan1 Toothbrush TravelKit CellPhone  RegisterBag Penka1 Toothbrush Jewelery CellPhone  CheckIn Pesho1 LondonSofia2 0  CheckIn Gosho1 LondonSofia2 1  CheckIn Ivan1 LondonSofia2 0  CheckIn Penka1 LondonSofia2 0  CheckIn Boiko1 LondonSofia2  CheckIn Goshko1 LondonSofia2  TakeOff  END | Registered trip SofiaLondon1  Registered trip LondonSofia2  Registered Pesho  ERROR: Passenger Pesho already registered!  Registered Gosho  Registered Ivan  Registered Penka  Registered Boiko  Registered Goshko  Registered bag with Toothbrush, Jewelery, CellPhone for Pesho  Registered bag with Toothbrush, Jewelery, Laptop for Gosho  Registered bag with Colombian for Gosho  Registered bag with Toothbrush, TravelKit, CellPhone for Ivan  Registered bag with Toothbrush, Jewelery, CellPhone for Penka  Checked in Pesho with 1/1 checked in bags  ERROR: Pesho is already checked in!  Checked in Gosho with 0/1 checked in bags  Checked in Ivan with 1/1 checked in bags  Checked in Penka with 1/1 checked in bags  Checked in Boiko with 0/0 checked in bags  Checked in Goshko with 0/0 checked in bags  SofiaLondon1:  Overbooked! Ejected Gosho  Confiscated 1 bags ($3303)  Successfully transported 5 passengers from Sofia to London.  LondonSofia2:  Successfully transported 0 passengers from London to Sofia.  Confiscated bags: 2 (4 items) => $53303  Registered Pesho1  Registered Gosho1  Registered Ivan1  Registered Penka1  Registered Boiko1  Registered Goshko1  Registered bag with Toothbrush, Jewelery, CellPhone for Pesho1  Registered bag with Toothbrush, Jewelery, Laptop for Gosho1  Registered bag with Colombian for Gosho1  Registered bag with Toothbrush, TravelKit, CellPhone for Ivan1  Registered bag with Toothbrush, Jewelery, CellPhone for Penka1  Checked in Pesho1 with 1/1 checked in bags  Checked in Gosho1 with 0/1 checked in bags  Checked in Ivan1 with 1/1 checked in bags  Checked in Penka1 with 1/1 checked in bags  Checked in Boiko1 with 0/0 checked in bags  Checked in Goshko1 with 0/0 checked in bags  Confiscated bags: 3 (5 items) => $103303 |