Telerik Academy

C# OOP - August 2017 - Travel Agency System

General Description

Implement a journey and ticket tracking system for a famous travel agency called Traveller. The application already accepts commands and outputs text for each submitted command, you just need to write the OOP. You can create different models (Bus, Airplane, Train, Ticket, Journey), as well as listing them. Make sure to follow all the good Object Orientated Programming practices and conventions that we have talked about during the lectures and don't let the length of this description intimidate you, read it carefully and start hacking!

Classes

Read the description carefully before you proceed to the code base. Read the code base carefully before you start implementing the down-mentioned requirements.

Implement and validate the following:

- **Train** has:
  - PassangerCapacity that is a number representing quantity in the interval of [30, 150].
    - Exception message: A train cannot have less than 30 passengers or more than 150 passengers.
  - Carts that is a number representing quantity in the interval of [1, 15].
    - Exception message: A train cannot have less than 1 cart or more than 15 carts.
  - Type that is a set of fixed values in the interval of [Land, Air, Sea].
  - PricePerKilometer that is a number representing currency.
  - Should be convertable to **string** in the format:

```
Train ----
Passenger capacity: VALUE
Price per kilometer: VALUE
Vehicle type: VALUE
Carts amount: VALUE
```

- **Airplane** has:
  - PassangerCapacity that is a number representing quantity.
  - Type that is a set of fixed values in the interval of [Land, Air, Sea].
  - HasFreeFood that is a boolean.
  - PricePerKilometer that is a number representing currency.

- Should be convertable to **string** in the format:

```
Airplane ----
Passenger capacity: VALUE
Price per kilometer: VALUE
Vehicle type: VALUE
Has free food: VALUE
```

- **Bus** has:
  - PassangerCapacity that is a number representing quantity in the interval of [10, 50].
    - Exception message: A bus cannot have less than 10 passengers or more than 50 passengers.
  - PricePerKilometer that is a number representing currency.
  - Type that is a set of fixed values in the interval of [Land, Air, Sea].
  - Should be convertable to **string** in the format:

```
Bus ----
Passenger capacity: VALUE
Price per kilometer: VALUE
Vehicle type: VALUE
```

- **Journey** has:
  - StartLocation that is a string with length in the interval of [5, 25].
    - Exception message: The StartingLocation's length cannot be less than 5 or more than 25 symbols long.
  - Destination that is a string with length in the interval of [5, 25].
    - Exception message: The Destination's length cannot be less than 5 or more than 25 symbols long.
  - Distance that is a number representing quantity in the interval of [5, 5000].
    - Exception message: The Distance cannot be less than 5 or more than 5000 kilometers.
  - Vehicle that is the vehicle used in the journey.
  - CalculateTravelCosts() that returns a currency calculated by:
    - Multiplying the Distance by the Vehicle's price per kilometer.
  - Should be convertable to **string** in the format:

```
Journey ----
Start location: VALUE
Destination: VALUE
Distance: VALUE
Vehicle type: VALUE
Travel costs: VALUE
```

- **Ticket** has:
    - Journey that is the journey the ticket is sold for.
    - AdministrativeCosts that is a number representing currency.
    - CalculatePrice() that returns a currency calculated by:
        - Multiplying the AdministrativeCosts by the Journey's travel costs.
    - Should be convertable to **string** in the format:

```
Ticket ----
Destination: VALUE
Price: VALUE
```

- **TravellerFactory** has:
    - **CreateBus(...)** that needs to be implemented.
    - **CreateAirplane(...)** that needs to be implemented.
    - **CreateTrain(...)** that needs to be implemented.
    - **CreateJourney(...)** that needs to be implemented.
    - **CreateTicket(...)** that needs to be implemented.

Additional validations

The laws of physics and finances dictate that:

- A vehicle with less than 1 passenger or more than 800 passengers cannot exist!
    - Exception message: A vehicle with less than 1 passengers or more than 800 passengers cannot exist!
- A vehicle with a price per kilometer lower than $0.10 or higher than $2.50 cannot exist!
    - Exception message: A vehicle with a price per kilometer lower than $0.10 or higher than $2.50 cannot exist!

In your case, there is such a vehicle, but think about these rules more generally. This system could be extended in the future to accommodate more vehicles.

Notes:

- All validation intervals are inclusive (closed).
- The interfaces of these members have already been implemented.
- You can use whatever Exception type you deem fit.

Commands

All commands are case insensitive, except their parameters! Each command is represented in the code base as a separate class, that is invoked by the Engine.

You are given a set of commands. The following are already implemented:

- **createbus** [PassangerCapacity] [PricePerKilometer] - Creates a new Bus.
- **createtrain** [PassangerCapacity] [PricePerKilometer] [Carts] - Creates a new Train.
- **createjourney** [StartLocation] [Destination] [Distance] [VehicleID] - Creates a new Journey.
- **listjourneys** - Lists all stored journeys.
- **listtickets** - Lists all stored tickets.

And these are the commands you need to implement yourself:

- **createairplane** [PassangerCapacity] [PricePerKilometer] [HasFreeFood] - Creates a new Airplane.
- **createticket** [JourneyID] [AdministrativeCosts] - Creates a new Ticket.
- **listvehicles** - Lists all stored vehicles.

Notes:

- Commands are dynamically invoked with Reflection (something we have not talked about much so far).
    - Do not change the command names, as they are used to resolve the passed in the console strings.
    - When creating a new command class, follow the naming and inheritance conventions, applied in existing command classes.

Architecture

Let's talk a bit about how the system works (you are already provided with all of this stuff, there is no need to implement it). There is an Engine located in the Core namespace that has a loop that cycles until the exit command is submitted. With each cycle, it takes the input, passes it to the command parser that find the command with that name and executes it with those parameters. All commands are located in the Core.Commands namespace. The commands themselves use the TravellerFactory located in the Core.Factories namespace to create the needed objects. After the command executes, it returns a result message to the Engine that prints it to the console and then the cycle beings again. In the Engine, there is a try-catch block that catches every possible exception type and prints the exception's message to the console. Do not bother reading those classes, your focus should be on the Models and Commands namespaces, where you need to place the classes you create, using the provided interfaces in the Contracts namespace or implement the commands which are not ready yet. The result from the execution of every command is printed on the console after each step, so please use the input one line at a time.

Constraints

- You are allowed to create new and modify existing **classes, interfaces, enumerations and namespaces** in the Models namespace.
- You are allowed to modify the **TravellerFactory** in the Core.Factories namespace.
- You are allowed to create and modify existing classes Core.Commands namespace. **(Careful with the names!)**
- You are allowed to add and remove usings from every file in the solution.
- **You are NOT allowed to modify any other existing interfaces!**
- **You are NOT allowed to modify any other existing classes, enumerations and namespaces!**

Example

Test each command separately, after you implement it. When you are done, use the input below to fully test your application.

Input

```
createbus 10 0.7
createtrain 300 0.4 3
createairplane 250 1 true
createairplane 250 2.7 true
createtrain 80 0.4 3
listvehicles
createjourney Sofia vTurnovo 300 0
createjourney Sofia vTurnovo 3 0
createjourney vTurnovo Sofia 300 3
listjourneys
createticket 0 30
createticket 1 100
listtickets
exit
```

Output

```
Vehicle with ID 0 was created.
Specified argument was out of the range of valid values.
Parameter name: A train cannot have less than 30 passengers or more than 150
passengers.
Vehicle with ID 1 was created.
Specified argument was out of the range of valid values.
Parameter name: A vehicle with a price per kilometer lower than $0.10 or
higher than $2.50 cannot exist!
Vehicle with ID 2 was created.
Bus ----
Passenger capacity: 10
Price per kilometer: 0.7
Vehicle type: Land
####################
Airplane ----
Passenger capacity: 250
Price per kilometer: 1
Vehicle type: Air
Has free food: True
####################
Train ----
Passenger capacity: 80
Price per kilometer: 0.4
Vehicle type: Land
```

```
Carts amount: 3
Journey with ID 0 was created.
Specified argument was out of the range of valid values.
Parameter name: The Distance cannot be less than 5 or more than 5000
kilometers.
Failed to parse CreateJourney command parameters.
Journey ----
Start location: Sofia
Destination: vTurnovo
Distance: 300
Vehicle type: Land
Travel costs: 210.0
Ticket with ID 0 was created.
Failed to parse CreateTicket command parameters.
Ticket ----
Destination: vTurnovo
Price: 240.0
```