C# OOP Exam

Christmas Pastry Shop

Overview

As we all love delicacies, today you were chosen to build a simple Christmas pastry shop software system. This system must have support for **Delicacy**, **Cocktail** and **Booth**. The project will consist of **model classes** and a **controller class**, which manages the **interaction** between the **delicacies**, **cocktails** and **booths**.

Setup

- Upload **only the `ChristmasPastryShop`** project in every task **except Unit Tests.**
- **Do not modify the interfaces or their packages.**
- 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** in the concrete class than the interface has defined.
- Make sure you have **no public fields** anywhere.
- **Exception messages** and **output messages** can be found in the **`"Utilities"`** folder.
- For solving this problem use **Visual Studio 2019, Visual Studio 2022** and **netcoreapp 3.1**

Task 1: Structure (50 points)

For this task's evaluation logic in the methods isn't included.

You are given **4** interfaces, and you must implement their functionality in the **correct classes**.

There are 3 types of entities and 3 repositories in the application: **`Booth`, `Delicacy`, `Cocktail` and a `Repository(BoothRepository, DelicacyRepository, CocktailRepository)` for each of them**:

Delicacy

`Delicacy` is a **base class** for any **type of `Delicacy`,** and it **should not be able to be instantiated**.

Data

- **`Name`** - **`string`**
    - If the name **is null or whitespace,** throw an **`ArgumentException`** with a message "`Name cannot be null or whitespace!`"
- **`Price`** - **`double`**

`Override ToString() method:`

Override the existing method **`ToString()`** and modify it, so the returned string must be in the following format:

`"{delicacyName} – {current price - formatted to the second decimal place} lv"`

Note: Do not use `"\r\n"` for a new line.

Constructor

The constructor of the **`Delicacy`** should take the following parameters upon initialization:

`string delicacyName, double price`

Child Classes

There are several concrete types of **Delicacy**:

Gingerbread

The **Gingerbread has a** constant value for **`gignerbreadPrice = 4.00`**

The constructor of the **Gingerbread** should take the following parameters upon initialization:

`string delicacyName`

Stolen

The **Stolen has a** constant value for **`stolenPrice = 3.50`**

The constructor of the **Stolen** should take the following parameters upon initialization:

`string delicacyName`

Cocktail

`Cocktail` is a **base class** for any **type of `Cocktail`** and it **should not be able to be instantiated**.

Data

- **Name** - **`string`**
  - If the name **is null or whitespace,** throw an **`ArgumentException`** with message "`Name cannot be null or whitespace!`"
- **Size** - **`string`**
  - Possible values: **`"Small"`**, **`"Middle"`**, **`"Large"`**. **`this.Size`** will be validated later in the **`Controller`** class.
- **Price** - **`double`**
  - If the **Size** is set to **`"Large"`**, the **Price** is set to be equal to the passed **value**
  - If the **Size** is set to **`"Middle"`**, the **Price** is equal to **⅔** of the passed **value** (example: ⅔ * 13.50 = 9.00)
  - If the **Size** is set to **`"Small"`**, the **Price** is equal to **⅓** of the passed **value** (example: ⅓ * 10.50 = 3.50)

`Override ToString() method:`

Override the existing method **`ToString()`** and modify it, so the returned string must be in the following format:

`"{cocktailName} ({size}) - {cocktailPrice - formatted to the second decimal place} lv"`

Note: Do not use `"\r\n"` for a new line.

Constructor

A **`Cocktail`** should take the following values upon initialization:

`string cocktailName, string size, double price`

Child Classes

There are several concrete types of **`Cocktail`**:

MulledWine

The **MulledWine** has **constant value** for price of **`Large MulledWine = 13.50`**

The constructor of the **MulledWine** should take the following parameters upon initialization:

`string cocktailName, string size`

Hibernation

The **Hibernation** has **constant value** for price of **Large `Hibernation - 10.50`**

The constructor of the **Hibernation** should take the following parameters upon initialization:

```
string cocktailName, string size
```

Booth

Data

- **`BoothId`** – **`int`** the booth number
- **`Capacity`** - **`int`** the booth capacity
  It can't be **less or equal to zero**. In these cases, throw an **`ArgumentException`** with message: **"`Capacity has to be greater than 0!`"**
- **`DelicacyMenu – DelicacyRepository`** all available to order delicacies
- **`CocktailMenu – CocktailRepository`** all available to order cocktails
- **`CurrentBill – double`**, set initial value to zero and increase the **`CurrentBill`** after every successful order (UpdateCurrentBill method)
- **`Turnover – double,`** set initial value to zero the **Turnover** should be increased, after paying the **`CurrentBill`** upon leaving the **Booth**
  - **If no orders** have been made to the specific **Booth**, **return zero**.
- **`IsReserved - boolean`** returns **true** if the **Booth** is **reserved**, otherwise returns **false.** Set its **initial** value to **False.**

Behavior

`void UpdateCurrentBill(double amount)`

When ordering new item, adds the amount(itemPrice) to the CurrentBill.

`void Charge()`

Increases the **`Turnover`** with the amount of the **`CurrentBill`** and sets the **`CurrentBill`** to **zero**.

`void ChangeStatus()`

Changes the IsReserved property:

- If its value is True, then sets it to False
- If its value is False, then sets it to True

Override ToString() method:

Override the existing method **ToString()** and modify it, so the returned string must be in the following format:

"Booth: {boothId}
Capacity: {boothCapacity}
Turnover: {turnoverAmount - formatted to the second decimal place} lv
-Cocktail menu:
--{cocktail$_1$.ToString()}
--{cocktail$_2$.ToString()}
…
--{cocktail$_N$.ToString()}
-Delicacy menu:
--{delicacy$_1$.ToString()}
--{delicacy$_2$.ToString()}
…
--{delicacy$_N$.ToString()}"

Note: Do not use "\r\n" for a new line.

Constructor

A **Booth** should take the following values upon initialization:

```
int boothId, int capacity
```

DelicacyRepository

The repository holds information about the delicacies.

Data

- **Models – IReadOnlyCollection<IDelicacy>**

Behavior

void AddModel(IDelicacy delicacy)

Adds an entity in the collection.

CocktailRepository

The repository holds information about the cocktails.

Data

- **Models – IReadOnlyCollection<ICocktail>**

Behavior

void AddModel(ICocktail cocktail)

Adds an entity in the collection.

BoothRepository

The repository holds information about the booths.

Data

- **Models – IReadOnlyCollection<IBooth>**

Behavior

`void AddModel(IBooth booth)`

Adds an entity in the collection.

Task 2: Business Logic (150 points)

The Controller Class

The business logic of the program should be concentrated around several **commands**. You are given interfaces, which you have to implement in the correct classes.

The first interface is **IController**. You must create a **Controller** class, which implements the interface and implements all of its methods. The constructor of **Controller** does not take any arguments. The given methods should have the logic described for each in the Commands section. When you create the **Controller** class, go into the **Engine** class constructor and uncomment the "`this.controller = new Controller();`" line.

Data

You need to keep track of some things, this is why you need some private fields in your controller class:

- **booths – BoothRepository**

Commands

There are several commands, which control the business logic of the application. They are stated below.

AddBooth Command

Parameters

- **capacity - int**

Functionality

Booth constructor will be expecting as first parameter **boothId.** So it should be created by taking the count of the already added booths in the **BoothRepository** + 1.

Creates a new **Booth** with the given **capacity.** Adds the newly created **Booth** to the **BoothRepository** and returns:

"Added booth number **{boothId}** with capacity **{capacity}** in the pastry shop!"

AddDelicacy Command

Parameters

- **boothId – int,** only valid boothId will be received as parameter
- **delicacyTypeName – string**

- **delicacyName – string**

Functionality

Creates a new **IDelicacy** from the proper type with the given name.

- If the given delicacyType is not supported in the application, return the following message: **"Delicacy type {type} is not supported in our application!"**
- If a **Delicacy** with the given delicacyName already exists in the delicacy repository, return the following message **"{delicacyName} is already added in the pastry shop!"**
- If the delicacy is created successfully, it is added to the DelicacyMenu of the Booth with the given boothId. Returns the following message:

  "{delicacyTypeName} {delicacyName} added to the pastry shop!"

AddCocktail Command

Parameters

- **boothId – int,** only valid boothId will be received as parameter
- **cocktailTypeName – string**
- **cocktailName – string**
- **size – string**

Functionality

Creates a new **ICocktail** from the proper type with the given name.

- If the given cocktailType is not supported in the application, return the following message: **"Cocktail type {type} is not supported in our application!"**
- If the given size is different from the supported in the application (**"Small", "Middle", "Large"**), return the following message: **"{size} is not recognized as valid cocktail size!"**
- If a **Cocktail** with the given **cocktailName && size** already exists in the cocktail repository, return the following message **"{size} {cocktailName} is already added in the pastry shop!"**
- If the **Cocktail** is created successfully, , it is added to the CocktailMenu of the Booth with the given boothId and returns the following message:

  "{size} {cocktailName} {cocktailTypeName} added to the pastry shop!"

ReserveBooth Command

Parameters

- **countOfPeople – int**

Functionality

- Order all booths from the BoothRepository, which are not reserved && their capacity is enough for the number of people provided, **by capacity ascending, and the by boothId, decsending**
- Take the first available **Booth**.
- If there is no such booth returns: **"No available booth for {numberOfPeople} people!"**
- If an available **Booth** is found, sets the **IsReserved** status to true and returns the following message:

"Booth {boothId} has been reserved for {numberOfPeople} people!"

TryOrder Command

Parameters

- **boothId – int,** only valid boothId will be received as parameter
- **order – string**

Functionality

The second prameter **(order)** will be a string sequence, seperated by "/":

- The first element of the sequence will be the **itemTypeName**
- The second element will be **itemName**
- The third element will be the **count of the ordered pieces**
- The fourth will exist only if the item is an **ICocktail**. The element (if such exists) will be the **size** of the **Cocktail**.

Finds the booth with the given boothId and finds the item from the given type with the given name. Before confirming the order, the method must make the following validations, in the following order:

- If the given **itemTypeName** is not existing in our application, return the following message: **"{itemTypeName} is not recognized type!"**
- If an item with the given **itemName** is not added in the according **IRepository** yet, return the following message: **"There is no {itemTypeName} {itemName} available!"**

If all validations pass, try to order the given item:

- If the item is cocktail:
  - Check if cocktail from the given itemTypeName, with the given itemName and the desired size is available:
    - If not, return the following message: **"There is no {size} {itemName} available!"**
  - **If all the validations pass,** the **CurrentBill is increased** with the price of the desired item, multiplied by the desired pieces **and the following message is returned:**

    "Booth {boothId} ordered {pieces} {itemName}!"
- If the item is delicacy:
  - Check if delicacy from the given itemTypeName and the given itemName is available :
    - If not, return the following message: **"There is no {itemTypeName} {itemName} available!"**
  - **If all the validations pass,** the **CurrentBill is increased** with the price of the desired item, multiplied by the desired pieces **and the following message is returned:**

    "Booth {boothId} ordered {pieces} {itemName}!"

LeaveBooth Command

Parameters

- **boothId – int,** only valid boothId will be received as parameter

Functionality

- Finds the **Booth** with the given **boothId**.

- Gets the **CurrentBill** and adds it to the **Turnover** of the **Booth**. Sets the **CurrentBill** to **zero.** This can be done through the **Charge()** method
- Sets the **IsReserved** status to false. This can be done through the **ChangeStatus()** method
- After all returns the following message:

"Bill {currentBill:f2} lv"

"Booth {boothId} is now available!"

BoothReport Command

Parameters

- **boothId – int,** only valid boothId will be received as parameter

Returns a string report for the **Booth** with the given **boothId**:

"Booth: {boothId}

Capacity: {boothCapacity}

Turnover: {turnover:f2} lv

-Cocktail menu:

--{cocktail$_1$.ToString()}

--{cocktail$_2$.ToString()}

…

--{cocktail$_N$.ToString()}

-Delicacy menu:

--{delicacy$_1$.ToString()}

--{delicacy$_2$.ToString()}

…

--{delicacy$_N$.ToString()}"

Note: Do not use "\r\n" for a new line.

Hint: Use the overriden Booth.ToString() method

Input / Output

You are provided with one interface, which will help with the correct execution process of your program. The interface is **Engine** and the class implementing this interface should read the input and when the program finishes, this class should print the output.

Input

Below, you can see the **format** in which **each command** will be given in the input:

- **AddBooth {capacity}**
- **AddDelicacy {delicacyTypeName} {delicacyName}**
- **AddCocktail {cocktailTypeName} {cocktailName} {size}**
- **ReserveBooth {countOfPeople}**
- **TryOrder {boothId} {order}**

- `LeaveBooth {boothId}`
- `BoothReport {boothId}`
- `Exit`

Output

Print the output from each command when issued. If an exception is thrown during any of the commands' execution, print the exception message.

Examples

Input

`AddBooth 5`
`AddDelicacy 1 Gingerbread Santabiscuit`
`AddCocktail 1 MulledWine Redstar Middle`
`ReserveBooth 4`
`TryOrder 1 MulledWine/Redstar/2/Middle`
`TryOrder 1 Gingerbread/Santabiscuit/2`
`LeaveBooth 1`
`BoothReport 1`
`Exit`

Output

`Added booth number 1 with capacity 5 in the pastry shop!`
`Gingerbread Santabiscuit added to the pastry shop!`
`Middle Redstar MulledWine added to the pastry shop!`
`Booth 1 has been reserved for 4 people!`
`Booth 1 ordered 2 Redstar!`
`Booth 1 ordered 2 Santabiscuit!`
`Bill 26.00 lv`
`Booth 1 is now available!`
`Booth: 1`
`Capacity: 5`
`Turnover: 26.00 lv`
`-Cocktail menu:`
`--Redstar (Middle) - 9.00 lv`
`-Delicacy menu:`
`--Santabiscuit - 4.00 lv`

Input
AddBooth 5 AddBooth 3 AddBooth 3 AddDelicacy 1 Stolen Sugarcookie AddDelicacy 2 Gingerbread Dwarfhat AddCocktail 3 Hibernation Bluewater Large AddCocktail 3 Hibernation Bluewater Small ReserveBooth 2 ReserveBooth 6 ReserveBooth 3 TryOrder 3 Hibernation/Bluewater/3/Middle TryOrder 2 Stolen/Sugarcookie/1 LeaveBooth 3 LeaveBooth 2 BoothReport 1 BoothReport 2 BoothReport 3 Exit
Output
Added booth number 1 with capacity 5 in the pastry shop! Added booth number 2 with capacity 3 in the pastry shop! Added booth number 3 with capacity 3 in the pastry shop! Stolen Sugarcookie added to the pastry shop! Gingerbread Dwarfhat added to the pastry shop! Large Bluewater Hibernation added to the pastry shop! Small Bluewater Hibernation added to the pastry shop! Booth 3 has been reserved for 2 people! No available booth for 6 people! Booth 2 has been reserved for 3 people! There is no Middle Bluewater available! There is no Stolen Sugarcookie available! Bill 0.00 lv Booth 3 is now available! Bill 0.00 lv Booth 2 is now available!

```
Booth: 1
Capacity: 5
Turnover: 0.00 lv
-Cocktail menu:
-Delicacy menu:
--Sugarcookie - 3.50 lv
Booth: 2
Capacity: 3
Turnover: 0.00 lv
-Cocktail menu:
-Delicacy menu:
--Dwarfhat - 4.00 lv
Booth: 3
Capacity: 3
Turnover: 0.00 lv
-Cocktail menu:
--Bluewater (Large) - 10.50 lv
--Bluewater (Small) - 3.50 lv
-Delicacy menu:
```

Task 3: Unit Tests (100 points)

You will receive a skeleton with two classes inside – **FootballPlayer** and **FootballTeam**. **FootballTeam** class will have some methods, fields, and constructors. Cover the whole class with the unit test to make sure that the class is working as intended. In Judge, you upload **.zip** to **football (**with **FootballTeamTests** inside**)** from the **skeleton**.