## C# OOP Exam – Hotel Booking Application

## Overview

You have to create a simple Hotel Booking application. It should be able to keep data about the available **Rooms** in different **Hotels** and to give information about the **type** and **category rate** of a hotel. Guests should be able to check the room **availability** and **bed capacity** andto **make new bookings**. The application keeps data for all the **bookings** and the **turnover** of every hotel.

## Setup

* Upload **only the** BookingAppproject in every problem **except** **Unit Tests**
* **Do not modify the interfaces or their namespaces**
* 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
* 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 have to implement their functionality in the **correct classes**.

There are **3** types of entities in the application: **Room, Booking** and **Hotel**. There should also be **RoomRepository, BookingRepository** and **HotelRepository**.

**Room**

The **Room** is a **base class** of any **type of room** and it **should not be able to be instantiated**.

**Data**

* **BedCapacity -** **int**
  + Property which represents the maximum amount of people which could be accommodated in the **Room.** Depends on the room type
* **PricePerNight – double**
  + **PricePerNight** cannot be **negative**. If so, throw new **ArgumentException** with message : "Price cannot be negative!".
  + Set **PricePerNight** initial value to **zero**.

**Constructor**

The constructor of the **Room** class should accept the following parameters:

int bedCapacity

**Behavior**

**void SetPrice(double price)**

This method sets the PricePerNight value when needed.

**Child Classes**

There are **three** actual types of **Room**:

**DoubleBed**

Has **BedCapacity of 2**.

The constructorshould take no values upon initialization.

**Studio**

Has **BedCapacity of 4**.

The constructorshould take no values upon initialization.

**Apartment**

Has **BedCapacity of 6**.

The constructorshould take no values upon initialization.

**Booking**

**Data**

* **Room - IRoom**
  + The room where the **Booking** will be accomodated
* **ResidenceDuration – int**
  + **ResidenceDuration** must be greater than **zero**. If **NOT**, throw new **ArgumentException** with message: "Duration cannot be negative or zero!".
* **AdultsCount – int**
  + The count of **Adults** cannot be less than 1. If so, throw new **ArgumentException** with message: "Adults count cannot be negative or zero!".
* **ChildrenCount – int**
  + The count of **Children** cannot be less than 0. If so, throw new **ArgumentException** with message: "Children count cannot be negative!".
* **BookingNumber – int,** returns the booking number, which is set by the constructor upon creating every new **Booking**.

**Constructor**

The constructorshould take the following values upon initialization:

IRoom room, int residenceDuration, int adultsCount, int childrenCount, int bookingNumber

**Behavior**

**string BookingSummary()**

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

"Booking number: {BookingNumber}

Room type: {RoomType}

Adults: {AdultsCount} Children: {ChildrenCount}

Total amount paid: {TotalPaid():F2} $"

**HINT:** TotalPaid() => MathRound(ResidenceDuration\*PricePerNight, 2), print TotalPaid() on the Console with **two decimal places** after decimal point.

**Hotel**

**Data**

* **FullName – string**
  + If the name **is null or whitespace**, throw an **ArgumentException** with message: "Hotel name cannot be null or empty!"
* **Category - int**
  + If the category is less than 1 or greater than 5**,** throw an **ArgumentException** with a message:

"Category should be between 1 and 5 stars!"

* **Turnover – double**
  + Returns the **Sum** of **all booking amounts(ResidenceDuration\*PricePerNight)** paid in the **Hotel,** rounded to the second decimal place
* **Rooms – IRepository<IRooms>** which holds information about all available rooms for the **Hotel**
* **Bookings – IRepository<IBooking>** which holds information about all bookings made for the **Hotel**

**Constructor**

The constructorshould take the following values upon initialization:

string fullName, int category

**RoomRepository**

The **RoomRepository** is a **class** which represents collection of bookings.

**Data**

* **Some private field might be helpful**

**Behavior**

**void AddNew(IRoom room)**

* Adds new **Room** to the repository.

**IRoom Select(string roomTypeName)**

* Returns a Room which is entity of type with the given room type name

**IReadonlyCollection<IRoom> All()**

* Returns a ReadonlyCollection of all rooms, that have been added to the repository**.**

**Constructor**

The constructor should not take any values upon initialization.

**HotelRepository**

The **HotelRepository** is a **class** which represents collection of hotels.

**Data**

* **Some private field might be helpful**

**Behavior**

**void AddNew(IHotel hotel)**

* Adds new **Hotel** to the repository.

**IHotel Select(string hotelName)**

* Returns a hotel which has the given hotelName or returns default value

**IReadonlyCollection<IHotel> All()**

* Returns a ReadonlyCollection of all hotels, that have been added to the repository**.**

**Constructor**

The constructor should not take any values upon initialization.

**BookingRepository**

The **BookingRepository** is a **class** which represents collection of bookings.

**Data**

* **Some private field might be helpful**

**Behavior**

**void AddNew(IBooking booking)**

* Adds new **Booking** to the repository.

**IBooking Select(string bookingNumberToString)**

* Returns a booking which has the given **bookingNumber** or returns default value

**IReadonlyCollection<IBooking> All()**

* Returns a ReadonlyCollection of all bookings, that have been added to the repository**.**

**Constructor**

The constructor should not take any values upon initialization.

## 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.

**Note: The Controller class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!**

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:

* **hotels – HotelRepository**

**Commands**

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

**AddHotel Command**

**Parameters**

* **hotelName - string**
* **category - int**

**Functionality**

Creates a **Hotel** with the given **name** and star **category**.The method should return one of the following messages:

* **If the** hotel **with the given name exists return:** "Hotel {hotelName}is already registered in our platform."
* If the hotel is successfully created, store the hotel in the appropriate collection and return: "{category} stars hotel {hotelName} is registered in our platform and expects room availability to be uploaded."

**UploadRoomTypes Command**

**Parameters**

* **hotelName - string**
* **roomTypeName - string**

**Functionality**

Uploads new room type for the given hotel.

* If hotel with such name doesn’t exist, returns: "Profile {hotelName} doesn’t exist!"
* If the given type is already created, returns: "Room type is already created!"
* If the room type is not correct, throw new **ArgumentException** with message: "Incorrect room type!"
* If all the given data is correct, create a room from the given type and add it to the **RoomRepository** of the **Hotel** with the given name, return: "Successfully added {roomType} room type in {hotelName} hotel!"

**SetRoomPrices Command**

**Parameters**

* **hotelName - string**
* **roomTypeName – string**
* **price - double**

**Functionality**

Sets prices to the given room type for the given hotel.

* If hotel with such name doesn’t exist, returns: "Profile {hotelName} doesn’t exist!"
* If the room type is not correct, throw new **ArgumentException** with message: "Incorrect room type!"
* If the given type is not created yet, returns: "Room type is not created yet!"
* You can set the room price only once. If it is already set, throw new **InvalidOperationException** with message: "Price is already set!"
* If the price is set successfully, return message: "Price of {roomType} room type in {hotelName} hotel is set!"

**BookAvailableRoom Command**

**Parameters**

* **adults – int**
* **children – int**
* **duration - int**
* **category - int**

**Functionality**

A reservation is made in the **first** room, which answers all the following conditions:

* **First, order** all the hotels **by FullName** alphabetically
* **Second**, **take** only the rooms which have their **PricePerNight** set (PricePerNight > 0 )
* **Third, order** all taken rooms from previous step **by** **bed capacity** ascending,
* **Finally**, **choose** from ordered rooms, the room with the **lowest capacity where** the guests will fit

If none of the available hotels corresponds to the given category, returns: "{category} star hotel is not available in our platform."

If none of the rooms can fit the requested guests, return message: "We cannot offer appropriate room for your request."

If the booking is successful, the method returns message: "Booking number {bookingNumber} for {hotelName} hotel is successful!"

Also for successful booking you should add the new **Booking** in the **BookingRepository** of the selected hotel**. Every new Booking should have booking number equal to the total number of the already added bookings to the selected hotel increased by one:**

**bookingNumber = totalBookingAppBookingsCount + 1;**

**HotelReport**

**Parameters**

* **hotelName – string**

**Functionality**

Returns on the console information about hotel with the given name and all the bookings made for this hotel.

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

* If there are no registered hotels with this name in the platform, return: "Profile {hotelName} doesn’t exist!"
* If the **Hotel** is found, return the following information for the hotel and **BookingSummary()** for every **Booking,** separated by **empty** new line. If the **Hotel** has **no** bookings in its **BookingRepository**, print "none" (look at the last example for reference), instead of **BookingSummary()** for each **Booking** ():

"Hotel name: {hotelName}

--{Category} star hotel

--Turnover: {hotelTurnover : F2} $

--Bookings:

Booking number: {Booking1.BookingNumber}

Room type: {RoomType}

Adults: {AdultsCount} Children: {ChildrenCount}

Total amount paid: {totalPaid} $

Booking number: {Booking2.BookingNumber}

Room type: {RoomType}

Adults: {AdultsCount} Children: {ChildrenCount}

Total amount paid: {totalPaid} $

...

Booking number: { Bookingn.BookingNumber}

Room type: {RoomType}

Adults: {AdultsCount}

Children: {ChildrenCount}

Total amount paid: {totalPaid} $"

/

none

**HINT:** print hotelTurnover on the Console with **two decimal places** after the decimal point.

**Input**

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

* **AddHotel {hotelName} {category}**
* **UploadRoomTypes {hotelName} {roomType}**
* **SetRoomPrices {hotelName} {roomType} {price}**
* **AddRoomAvailability {hotelName} {roomsCount} {roomType}**
* **BookAvailableRoom {adultsCount} {childrenCount} {residenceDuration} {category}**
* **HotelReport**
* **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** |
| AddHotel Saint George 5  AddHotel Sunari Beach 3  AddHotel Alpine Slopes 4  UploadRoomTypes Saint George Apartment  UploadRoomTypes Alpine Slopes Studio  UploadRoomTypes Sunari Beach DoubleBed  UploadRoomTypes Sunari Beach Studio  SetRoomPrices Saint George Apartment 350  SetRoomPrices Sunari Beach DoubleBed 33  SetRoomPrices Alpine Slopes Studio 220  AddHotel Phoenix 3  UploadRoomTypes Phoenix Studio  BookAvailableRoom 2 0 3 3  HotelReport Sunari Beach  Exit |
| **Output** |
| 5 stars hotel Saint George is registered in our platform and expects room availability to be uploaded.  3 stars hotel Sunari Beach is registered in our platform and expects room availability to be uploaded.  4 stars hotel Alpine Slopes is registered in our platform and expects room availability to be uploaded.  Successfully added Apartment room type in Saint George hotel!  Successfully added Studio room type in Alpine Slopes hotel!  Successfully added DoubleBed room type in Sunari Beach hotel!  Successfully added Studio room type in Sunari Beach hotel!  Price of Apartment room type in Saint George hotel is set!  Price of DoubleBed room type in Sunari Beach hotel is set!  Price of Studio room type in Alpine Slopes hotel is set!  3 stars hotel Phoenix is registered in our platform and expects room availability to be uploaded.  Successfully added Studio room type in Phoenix hotel!  Booking number 1 for Sunari Beach hotel is successful!  Hotel name: Sunari Beach  --3 star hotel  --Turnover: 99.00 $  --Bookings:  Booking number: 1  Room type: DoubleBed  Adults:: 2 Children: 0  Total amount paid: 99.00 $ |
| **Input** |
| AddHotel Saint George 5  UploadRoomTypes Saint George Apartment  UploadRoomTypes Saint George Studio  UploadRoomTypes Saint George DoubleBed  SetRoomPrices Saint George Apartment 350  SetRoomPrices Saint George Studio 220  SetRoomPrices Saint George DoubleBed 150  BookAvailableRoom 2 0 3 5  BookAvailableRoom 2 1 4 5  BookAvailableRoom 3 1 5 5  BookAvailableRoom 5 1 1 5  BookAvailableRoom 4 1 2 5  HotelReport Saint George  Exit |
| **Output** |
| 5 stars hotel Saint George is registered in our platform and expects room availability to be uploaded.  Successfully added Apartment room type in Saint George hotel!  Successfully added Studio room type in Saint George hotel!  Successfully added DoubleBed room type in Saint George hotel!  Price of Apartment room type in Saint George hotel is **set!**  Price of Studio room type in Saint George hotel is set!  Price of DoubleBed room type in Saint George hotel is set!  Booking number 1 for Saint George hotel is successful!  Booking number 2 for Saint George hotel is successful!  Booking number 3 for Saint George hotel is successful!  Booking number 4 for Saint George hotel is successful!  Booking number 5 for Saint George hotel is successful!  Hotel name: Saint George  --5 star hotel  --Turnover: 3480.00 $  --Bookings:  Booking number: 1  Room type: DoubleBed  Adults: 2 Children: 0  Total amount paid: 450.00 $  Booking number: 2  Room type: Studio  Adults: 2 Children: 1  Total amount paid: 880.00 $  Booking number: 3  Room type: Studio  Adults: 3 Children: 1  Total amount paid: 1100.00 $  Booking number: 4  Room type: Apartment  Adults: 5 Children: 1  Total amount paid: 350.00 $  Booking number: 5  Room type: Apartment  Adults: 4 Children: 1  Total amount paid: 700.00 $ |
| **Input** |
| AddHotel Casa Domini 5  HotelReport Casa Domini  Exit |
| **Output** |
| Hotel name: Casa Domini  --5 star hotel  --Turnover: 0.00 $  --Bookings:  none |

1. **Task 3: Unit Tests (100 points)**

You will receive a skeleton with **Booking,** **Hotel** and **Room** classes inside. The classes will have some methods, fields and one constructor, which are working properly. You are **NOT ALLOWED** to change any class. Cover the whole **Hotel** class with unit tests to make sure that the class is working as intended.

You are provided with a **unit test project** in the **project skeleton**.

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