HotelProgram

January 31, 2025

1 Simple Hotel Program

1.1 Class

- We'll have the **Hotel** class representing Individual hotels
- **Room** class

1.2 Hotel Class Overview

☒ Name field (Store the name of the hotel)
☒ Location field (Store the location of the hotel)
☒ List of **room** objects
☒ Int **occupiedCnt** to keep track of how many rooms in a hotel

1.3 Specific Hotel Class overview

☒ Two Constructor
  – `__init__` => hotel *name* and *location*
    * Assign **numOfRooms** to zero (Number of rooms in the hotel) - **10 element** array
☒ **AddRoom** method create each room with **required information**
  – *room number*, *bed type*, *smoking/non-smoking* and *room rate*
    * 5 rooms with different characteristics
  – **occupied** attribute Boolean field set to False when room is created
☒ Increment **numOfRoom** *instance variable*
  – Also incremented occupiedRoomCnt once a room is filled

1.4 Exception Handling?

- Make sure smoking is s/n (look at add room)

[1]:
```
# Let's build the Hotel blueprint first
from abc import ABC, abstractmethod

class HotelABC(ABC):
    # Required ClassMethod
    @abstractmethod
    def isFull(self):
        # Must return a Boolean
        pass
```

```
    @abstractmethod
    def isEmpty(self):
        # Must return Boolean
        pass

    @abstractmethod
    def addRoom(self, roomnumber, bedtype, smoking, price):
        pass

    @abstractmethod
    def addReservation(self, occupantName, smoking, bedtype):
        pass

    @abstractmethod
    def cancelReservation(self, occupantName):
        pass

    @abstractmethod
    def _findReservation(self, occupantName):
        pass

    @abstractmethod
    def printReservationList(self):
        # Assuming we're printing all rooms
        pass

    @abstractmethod
    def getDailySales(self):
        pass

    @abstractmethod
    def occupancyPercentage(self):
        pass

    # Setters and getters (means name and location are private variables)
    @abstractmethod
    def getName(self):
        # Get name of Hotel
        pass

    @abstractmethod
    def setName(self, new_hotel_name):
        # Set name of Hotel
        pass

    @abstractmethod
```

```
    def getLocation(self):
        # Get location of Hotel
        pass

    @abstractmethod
    def setLocation(self, new_hotel_location):
        # Get location of Hotel
        pass

# Now that we have our blueprint out, let's build the actual Hotel class
class Hotel(HotelABC):
    # Constructor
    def __init__(self, name='', location='', occupiedCnt=0, numOfRooms=0):
        # Remember if have we setter and getter it's most likely a private␣
 ↪variable
        self.__name = name
        self.__location = location
        self.occupiedCnt = occupiedCnt
        self.numOfRooms = numOfRooms
        self.theRooms = []

    # Building setter and getters for location and name
    def getName(self):
        return self.__name

    def getLocation(self):
        return self.__location

    def setName(self, new_hotel_name):
        self.__name = new_hotel_name
        return self.getName()

    def setLocation(self, new_hotel_location):
        self.__location = new_hotel_location
        return self.getLocation()

    # Building the other Hotel logic
    def isFull(self):
        # We could check if all the rooms in the hotel are occupied with␣
 ↪occupiedCnt vs the numOfRooms
        return self.occupiedCnt == self.numOfRooms

    def isEmpty(self):
        # We could check if all the rooms in the hotel are unoccupied with␣
 ↪occupiedCnt being 0
        return self.occupiedCnt == 0
```

```
    # addRoom, addReservation, cancelReservation, findReservation we'll work␣
↪with after we create our Room Class
    def addRoom(self, roomnumber, bedtype, smoking, price):
        # I think they want us to build a room object within hotel and append␣
↪to our room list
        room_obj = Room(roomnumber, bedtype, price, smoking)
        self.theRooms.append(room_obj)
        # Once we addRoom we need to increase the number of room in the hotel
        self.numOfRooms += 1
        print(f'{room_obj} Added to Rooms')

    def addReservation(self, occupantName, smoking, bedtype):
        # We need to filter and check if there are any rooms available with␣
↪these parameters
        # We don't want occupied rooms
        FOUND_ROOM = False
        unoccupied_room = [r for r in self.theRooms if not r.isOccupied()]
        for room in unoccupied_room:
            if room.getSmoking() == smoking and room.getBedType() == bedtype:
                # The Room is NOT occupied, and available
                room.setOccupant(occupantName)
                room.setOccupied(True)
                # If a room is occupied make sure we increment the count of␣
↪occupied
                self.occupiedCnt += 1
                FOUND_ROOM = True
                # We could break since we found an available room
                break

        print('Found Room') if FOUND_ROOM else 'Room Not Found'

    def cancelReservation(self, occupantName):
        # Now we could actually use findReservation to find the room associated␣
↪with the name (if any)
        res = self._findReservation(occupantName)
        if res != -1:
            # Remember we returned -1 if the room wasn't found
            found_res = self.theRooms[res]
            found_res.setOccupant('')
            found_res.setOccupied(False)
            # Now if they cancel, we need to reduce the occupiedCount
            self.occupiedCnt -= 1
            print(f'{occupantName}\'s room: {found_res.getRoomNumber()} was␣
↪cancelled')
        else:
            print(f'{occupantName}\'s room not found')
```

```
    # Private method
    def _findReservation(self, occupantName):
        # We need to loop with enumerate for that index value
        """
            # We'll only search the rooms that are occupied (we could use␣
↪filter()
            # occupied_rooms = list(filter(lambda  r : r.isOccupied() ,self.
↪theRooms))

            WE cannot filter at the moment because we need the correct index to␣
↪access the room. Perhaps if we have an id with mysql db then we could filter␣
↪to optimize it
        """

        for index, reservation in enumerate(self.theRooms):
            if occupantName == reservation.getOccupant():
                # Must return index
                return index

        # No matches at this point
        # I think we're supposed to return -1 because it's not found
        return -1

    # My helper method to find occupied rooms
    def occupiedRooms(self):
        return list(filter(lambda r: r.isOccupied(), self.theRooms))

    def printReservationList(self):
        # Printing all the rooms that are occupied
        # Here is where we could implement out filter logic
        occupied_rooms = self.occupiedRooms()
        if len(occupied_rooms) >= 1:
            print('Reservation List: ')
            for room in occupied_rooms:
                print(room)
                print()
        else:
            print('No Reservation Found')

    def getDailySales(self):
        # Again we're only adding up the rates for all the OCCUPIED rooms
        occupied_rooms = self.occupiedRooms()
        if len(occupied_rooms) >= 1:
            # We can calculate the sales
            total_sales = 0
            for room in occupied_rooms:
```

```
                total_sales += room.getRoomRate()
            return f'Total Sales For Today: ${total_sales}'
        else:
            print('No Sales For Today :(')

    def occupancyPercentage(self):
        try:
            return f'{(self.occupiedCnt / self.numOfRooms) * 100}%'
        except ZeroDivisionError as e:
            print('No rooms Added To The Hotel Yet')
            raise

    # String Formatting
    def __str__(self):
        # Building all the Room String first
        hotel_details = f"""
        Hotel Name: {self.getName()}
        Number of Rooms: {self.numOfRooms}
        Number of Occupied Rooms: {self.occupiedCnt}

        Room Details:

        """

        # Join all the elements in theRooms array into a new line for each␣
↪element
        # We needed to do str(r) because r itself is a Room object
        room_details = "\n".join(str(r) for r in self.theRooms)

        # Include room details if there are rooms otherwise, we'll just return␣
↪the hotel details ONLY
        return (hotel_details + room_details) if len(room_details) >= 1 else␣
↪hotel_details
```

[2]:
```
# Let's start working on the Rooms Class
class RoomsABC(ABC):

    @abstractmethod
    def getBedType(self):
        pass

    @abstractmethod
    def getSmoking(self):
        pass

    @abstractmethod
    def getRoomNumber(self):
```

```
        pass

    @abstractmethod
    def getRoomRate(self):
        pass

    @abstractmethod
    def getOccupant(self):
        pass

    @abstractmethod
    def setOccupied(self, new_occupied):
        # Boolean
        pass

    @abstractmethod
    def setOccupant(self, new_occupant):
        pass

    @abstractmethod
    def setRoomNum(self, new_room_num):
        pass

    @abstractmethod
    def setBedType(self, new_bedtype):
        pass

    @abstractmethod
    def setRate(self, new_room_rate):
        pass

    @abstractmethod
    def setSmoking(self, new_smoking):
        pass

    @abstractmethod
    def isOccupied(self):
        # Boolean
        pass

class Room(RoomsABC):
        # Constructor
        def __init__(self, roomNum, bedType, rate, smoking, occupantName='',␣
 ↪occupied=False):
            # We have setter and getter for all of these; therefore, they'll be␣
 ↪private variables
            self.__roomNum = roomNum
```

```
        self.__bedType = bedType
        self.__rate = rate
        # This is an optional ARgument because it could be unoccupied
        self.__occupantName = occupantName
        self.__smoking = smoking
        self.__occupied = occupied

    # Getters
    def getBedType(self):
        return self.__bedType

    def getSmoking(self):
        return self.__smoking

    def getRoomNumber(self):
        return self.__roomNum

    def getRoomRate(self):
        return self.__rate

    def getOccupant(self):
        return self.__occupantName

    def isOccupied(self):
        return self.__occupied

    # Setters
    def setOccupied(self, new_occupied):
        self.__occupied = new_occupied
        return self.isOccupied()

    def setOccupant(self, new_occupant):
        self.__occupantName = new_occupant
        return self.getOccupant()

    def setRoomNum(self, new_room_num):
        self.__roomNum = new_room_num
        return self.getRoomNumber()

    def setBedType(self, new_bedtype):
        self.__bedType = new_bedtype
        return self.getBedType()

    def setRate(self, new_room_rate):
        self.__rate = new_room_rate
        return self.getRoomRate()
```

```
        def setSmoking(self, new_smoking):
            self.__smoking = new_smoking
            return self.getSmoking()


        # String for Room
        def __str__(self):
            # Need to check if Occupied or not
            message = f"""
                Room Number: {self.getRoomNumber()}
                Occupant Name: {self.getOccupant() if self.isOccupied() else␣
↪'Not Occupied'}
                Smoking Room: {self.getSmoking()}
                Bed Type: {self.getBedType()}
                Rate: {self.getRoomRate()}
            """
            return message
```

2 Testing Hotel

This is like a perfect scenario without any miss inputs or need for **Exception Handling**

Functions we tested: - [x] Creating the Hotel instance + `setName` and `setLocation` - [x] Adding Room (Added 5) - [x] Adding Reservation - Even tested Private function: `_findReservation` - [x] Cancelling Reservation - [x] Printing all Reservations that are **occupied** - [x] Daily Sales based on the people that **reserved** a room - [x] Hotel **String Representation** with `__str__` room objects

[3]:
```
# Testing

# Our Hotel
hotel = Hotel('Dynasty', 'NJ')
hotel.setName('Dynasty Hotel')
hotel.setLocation('NY')
print(hotel.getName(), hotel.getLocation())

# Adding a Rooms
testing_rooms = [
    {
        'roomNum': 102,
        'bedType': 'king',
        'smoking': 'n',
        'rate': 110.0
    },
    {
        'roomNum': 101,
        'bedType': 'queen',
```

```
        'smoking': 's',
        'rate': 100.0
    },
    {
        'roomNum': 103,
        'bedType': 'king',
        'smoking': 'n',
        'rate': 88.0
    },
    {
        'roomNum': 104,
        'bedType': 'twin',
        'smoking': 's',
        'rate': 100.0
    },
    {
        'roomNum': 105,
        'bedType': 'queen',
        'smoking': 'n',
        'rate': 99.0
    },
]

for room in testing_rooms:
    hotel.addRoom(room['roomNum'], room['bedType'], room['smoking'],␣
 ↪room['rate'])
```

```
Dynasty Hotel NY

                Room Number: 102
                Occupant Name: Not Occupied
                Smoking Room: n
                Bed Type: king
                Rate: 110.0
            Added to Rooms

                Room Number: 101
                Occupant Name: Not Occupied
                Smoking Room: s
                Bed Type: queen
                Rate: 100.0
            Added to Rooms

                Room Number: 103
                Occupant Name: Not Occupied
                Smoking Room: n
                Bed Type: king
                Rate: 88.0
```

```
Added to Rooms

   Room Number: 104
   Occupant Name: Not Occupied
   Smoking Room: s
   Bed Type: twin
   Rate: 100.0
Added to Rooms

   Room Number: 105
   Occupant Name: Not Occupied
   Smoking Room: n
   Bed Type: queen
   Rate: 99.0
Added to Rooms
```

[4]:
```
# Let's try addReservation
myReservation = {
    'name': 'Thy',
    'bedType': 'queen',
    'smoking': 'n'
}

hotel.addReservation(myReservation['name'], myReservation['smoking'],␣
 ↪myReservation['bedType'])
# Confirming the Reservation with findReservation
room_num = hotel._findReservation(myReservation['name'])
# Room found (-1 is NOT_FOUND)
print(hotel.theRooms[room_num]) if room_num != 1 else f'No Reservation for:␣
 ↪{myReservation['name']}'
```

```
Found Room

   Room Number: 105
   Occupant Name: Thy
   Smoking Room: n
   Bed Type: queen
   Rate: 99.0
```

[5]:
```
# Let's try cancelling
hotel.cancelReservation(myReservation['name'])
```

```
Thy's room: 105 was cancelled
```

[6]:
```
# Now testing printing ReservationList
# We just cancelled our reservation so there are none let's add a reservation
```

```
hotel.addReservation(myReservation['name'], myReservation['smoking'],␣
 ↪myReservation['bedType'])

# Wilson Reservation
wReservation = {
    'name': 'Wilson',
    'bedType': 'king',
    'smoking': 'n'
}

hotel.addReservation(wReservation['name'], wReservation['smoking'],␣
 ↪wReservation['bedType'])
hotel.printReservationList()
```

```
Found Room
Found Room
Reservation List:

                Room Number: 102
                Occupant Name: Wilson
                Smoking Room: n
                Bed Type: king
                Rate: 110.0



                Room Number: 105
                Occupant Name: Thy
                Smoking Room: n
                Bed Type: queen
                Rate: 99.0
```

[7]:
```
# Daily Sales (Based on 2 reservation from Thy & Wilson we'd have 110.0 + 99.0)
hotel.getDailySales()
```

[7]:
```
'Total Sales For Today: $209.0'
```

[8]:
```
# Checking Occupancy Percentage
hotel.occupancyPercentage() # 40% makes sense because we have 5 rooms and 2 of␣
 ↪them are occupied
```

[8]:
```
'40.0%'
```

[9]:
```
# String Representation of Hotel
print(hotel)
```

```
Hotel Name: Dynasty Hotel
Number of Rooms: 5
Number of Occupied Rooms: 2

Room Details:


        Room Number: 102
        Occupant Name: Wilson
        Smoking Room: n
        Bed Type: king
        Rate: 110.0


        Room Number: 101
        Occupant Name: Not Occupied
        Smoking Room: s
        Bed Type: queen
        Rate: 100.0


        Room Number: 103
        Occupant Name: Not Occupied
        Smoking Room: n
        Bed Type: king
        Rate: 88.0


        Room Number: 104
        Occupant Name: Not Occupied
        Smoking Room: s
        Bed Type: twin
        Rate: 100.0


        Room Number: 105
        Occupant Name: Thy
        Smoking Room: n
        Bed Type: queen
        Rate: 99.0
```

3 Handling Exceptions

We just want to make sure they're supplying the correct arguments.

Made a function to compare **types**: - Hotel Name and Location must be a **string** - Rate has to be

a **float** (double) - Room Number must be an **int** - Smoking should either be: 'n' or 's'

We also took into account of `occupancyPercentage`: - Made sure there were Rooms before we end up **dividing by zero**

[10]:
```
from colorama import Fore,Style
def checkValidInput(obj,exceptedType):
    if not type(obj) == exceptedType:
        raise TypeError(f"{Fore.RED} {obj} must be of type {exceptedType}␣
 ↪{Style.RESET_ALL}")
    return True

testing_rooms = [
    {
        'roomNum': 102,
        'bedType': 'king',
        'smoking': 'n',
        'rate': 110.0
    },
    {
        'roomNum': 101,
        'bedType': 'queen',
        'smoking': 'y', # Not n/s should bring up an error (ValueError)
        'rate': 100.0
    },
    {
        'roomNum': 103,
        'bedType': 'king',
        'smoking': 'n',
        'rate': 88  # Int (Should bring up an Error) Expected: Float
    },
    {
        'roomNum': '104',   # String (Should bring up an error) Expected: Int
        'bedType': 'twin',
        'smoking': 's',
        'rate': 100.0
    },
    {
        'roomNum': 105,
        'bedType': 'queen',
        'smoking': 'n',
        'rate': 99.0
    },
]

try:
    # Guarding type
    hotel_name = 'Beach Marriot'
```

```
    hotel_location = 'Pensacola'
    if checkValidInput(hotel_name, str) and checkValidInput(hotel_location,␣
 ↪str):
        hotel = Hotel(hotel_name, hotel_location )
        for room in testing_rooms:
            # Making sure Rate is a float (double) and smoking is either n/s
            if checkValidInput(room['rate'], float) and␣
 ↪checkValidInput(room['roomNum'], int) and (room['smoking'] == 'n' or␣
 ↪room['smoking'] == 's'):
                hotel.addRoom(room['roomNum'], room['bedType'],␣
 ↪room['smoking'], room['rate'])
            else:
                raise ValueError(f'{Fore.RED} Smoking "{room["smoking"]}" is␣
 ↪not valid {Style.RESET_ALL}')
except Exception as e:
    print(e)
```

```
                Room Number: 102
                Occupant Name: Not Occupied
                Smoking Room: n
                Bed Type: king
                Rate: 110.0
            Added to Rooms
 Smoking "y" is not valid
```

[11]:
```
# Assume we haven't added any Rooms so let's run occupy Percentage (Divide by 0)
secondHotel = Hotel('Dynasty Hotel', 'NY')
print(secondHotel)
try:
    # Again we except ZeroDivisionError but also raised for this Exception block
    secondHotel.occupancyPercentage()
except Exception as e:
    print(Fore.RED + str(e) + Style.RESET_ALL)
```

```
        Hotel Name: Dynasty Hotel
        Number of Rooms: 0
        Number of Occupied Rooms: 0

        Room Details:


No rooms Added To The Hotel Yet
division by zero
```

[11]:
```

```