Advanced Programming

Lab Sheet 3 – Java & UML

Introduction

This lab session re-enforces the learning from this week's lecture on UML. You will be working on some small parts of some hotel booking management software. In the first exercise, you will be coding some classes to a design depicted in a UML class model. In the second, you will be creating a UML class model from some previously-written code.

Task 1: Coding from a UML Model

Two of the classes found in our hotel booking system are the Booking class and the BookingManager class. The BookingManager class maintains a list (e.g. an ArrayList) of all bookings made, and has various utility methods for finding information out about the bookings on its list. The Booking class contains basic information about a hotel booking, with most of its methods being simple getters and setters. A UML class model of the two classes is depicted below.

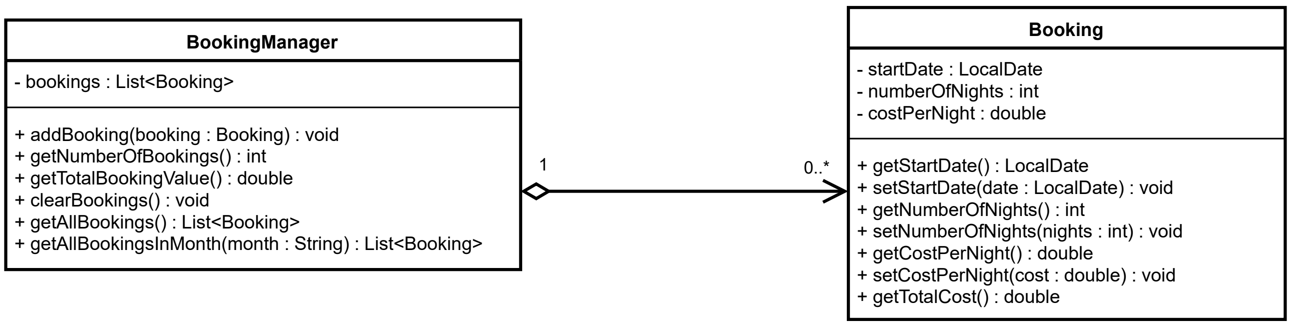
BookingManager
- bookings : List<Booking>
+ addBooking(booking : Booking) : void
+ getNumberOfBookings() : int
+ getTotalBookingValue() : double
+ clearBookings() : void
+ getAllBookings() : List<Booking>
+ getAllBookingsInMonth(month : String) : List<Booking>
1
0..*
Booking
- startDate : LocalDate
- numberOfNights : int
- costPerNight : double
+ getStartDate() : LocalDate
+ setStartDate(date : LocalDate) : void
+ getNumberOfNights() : int
+ setNumberOfNights(nights : int) : void
+ getCostPerNight() : double
+ setCostPerNight(cost : double) : void
+ getTotalCost() : double

Figure 1: UML Class Model of part of a Hotel Booking System

Create the two classes depicted above. Some of the methods are a little tougher than others! You can read about the LocalDate class, which is in the java.time package, here. Pay particular attention to the BookingManager's getAllBokingsInMonth() method, which is trickier than it first appears.

You will need to also create a class with a main() method that creates a BookingManager and some Bookings before testing each of the BookingManager's methods to make sure you implemented them correctly.

Task 2: Modelling Existing Code

The code depicted below is another part of the hotel booking system. The QuoteCostCalculator class is responsible for deciding how much to quote prospective guests for particular dates. It relies on some constants defined in the HotelInfo class and the BookingManager class to do its work.

Extend the UML class diagram depicted in Figure 1, previously, with the classes that you now know are part of the booking system. I would recommend using http://www.draw.io/ for this task.

```
import java.time.LocalDate;

public class QuoteCostCalculator {

        BookingManager manager;

        public QuoteCostCalculator(BookingManager manager) {
                this.manager = manager;
        }

        public double calculateQuoteFor(LocalDate date, int nights) throws HotelFullException {
                return calculateBaseNightlyCostFor(date,nights) * nights;
        }

        public boolean isFull(LocalDate date) {
                return !(manager.getAllBookingsOn(date).size() < HotelInfo.MAX_ROOMS);
        }

        public boolean isQuiet(LocalDate date) {
                return manager.getAllBookingsOn(date).size() < HotelInfo.DESIRED_OCCUPANCY;
        }

        public boolean isBusy(LocalDate date) {
                return manager.getAllBookingsOn(date).size() > HotelInfo.DESIRED_OCCUPANCY;
        }

        private double calculateBaseNightlyCostFor(LocalDate date, int nights) throws HotelFullException {
                boolean quiet = false;
                boolean busy = false;
                LocalDate currentDate = date;
                int nightsRemaining = nights;

                while ( nightsRemaining > 0 ) {
                        if ( isFull(currentDate) ) { throw new HotelFullException(); }
                        else if ( isQuiet(currentDate) ) { quiet = true; }
                        else if ( isBusy(currentDate) ) { busy = true; }
                        currentDate = currentDate.plusDays(1);
                        nightsRemaining--;
                }
                double cost = HotelInfo.BASE_PRICE;

                if ( quiet ) { cost -= (cost * (HotelInfo.QUIET_DATE_DISCOUNT / 100)); }
                else if ( busy ) { cost += (cost * (HotelInfo.BUSY_DATE_UPLIFT / 100)); }
                if ( nights >= HotelInfo.LONG_BOOKING_LENGTH ) {
                        cost -= (cost * (HotelInfo.LONG_BOOKING_DISCOUNT / 100));
                }

                return cost;
        }

}
```

Figure 2: Code Listing for the QuoteCostCalculator class

Too Easy?

There are a number of interesting extensions that can be made to the BookingManager class from Task 1. Some suggestions are:

Try adding a getAllBookingsOn() method that accepts a LocalDate as a parameter, and returns a list of all bookings that include the specified date.

Try adding a second, overloaded, getAllBookingsOn() method that accepts a day of the week String as a parameter, capable of returning all bookings that start on (e.g.) a Saturday.

Try adding a method capable of producing some simple reports and statistics on bookings to System.out. These statistics could include the average value of bookings that start on a particular day of the week, the average booking length, the median cost per night of bookings, etc.

Too Hard?

If you're struggling to read the model, it might be worth going back over the lecture notes from last week, which covered several types of UML model, and included some material on both coding from models, and modelling existing code. You could also take a look at this tutorial for some extra reading material.