OfficeShare API (stage 1) Design Document
10/9/14
Author: William Passidomo

Introduction

This document is to be used as the design for a program which allows Office Space, which would otherwise go unused, to be presented to the open marketplace where, eventually, those in need of such Office Spaces will be able to reserve them for their own use for a predetermined period of timefu

Overview

The problem this program aims to solve is one which computers might be most adept at not solving, but minimizing, and that is the problem of inefficiency. When a future home owner enters the marketplace, it would seem logical that they express their "needs" for a property as the amount which may satisfy their highest reasonably conceived need for space. It is obvious where gross inefficient enters this equation, the home buyer will come to find (something they likely knew when they purchased) that they have need for the full potential of the home (space-wise) on rare occasion. This may not describe everyone who purchases a home, but at least a non-trivial portion. By matching those in need of a work space with those who have extra room, allows the homeowner to recoup on his or her investment.

In creating this program, we aim to fulfill this match in dynamic and secure fashion by allowing those which potential office space to post nearly instantaneously their offering and *eventually* allowing those in the temporary Office Space market to peruse the offerings

Requirements

1) to support the creation, update and deletion of Office Space Providers
   Office Space Providers are Users who have 1 or more Office Spaces (see below) which they are willing to rent. Office Space Providers must have a name, a means of contact, an Image of themselves and a PayPal Account through which they may accept payment for their space.
   Office Space Providers may be reviewed by Users on overall quality, these reviews must be submitted by another user and include a 1-5 star rating, a comment and a time stamp

2) to support the create, update, and deletion of Office Spaces
   Office Spaces are offered by one Office Space Providers and include a variety of fields which describe the Office Space. Office Spaces are required to have

one or more Images of the physical space, a Capacity which lists the square footage of the whole space, the maximum number of people who can occupy the space at once and the number of workspaces (more that one person may occupy one workspace) and a rate which notes a per day rate in dollars over a fixed period for with the space may be rented (i.e 40 dollars per day for 4 days) and a total amount for the whole period derived from the period and rate (i.e 160 dollars). Office Space Providers may chose to list features (both physical and services) available in their Office Spaces, separated into private features, which the renter will have sole access to and common features, which the renter will have joint access to along with the homes occupants. Features may include bathroom, wifi, coffee, or parking. Each Office Space must have their full address listed for the sake of search and authentication.

3) a nonfunctional requirement is that Objects must meet the requirements at all times. Im not sure how to say this clearer, but for example, an OfficeSpace must be associated with at least one Image. this means that while the programs should allow you to remove and add pictures, it should not allow you to remove the last picture so there are no pictures left in the collection. This concept is a nonfunctional requirement for all of the functional requirements listed in 1 and 2

Implementation

Class Directory

Address (includes static inner class AddressBuilder)

The Address class encapsulates the fields pertaining to the location of the OfficeSpace. Fields required for any Address class are; Street1, city, state, zip, and country code. Since there are so many required fields and a large number of optional fields, such as street2, po box, apartment number and coordinates, the Builder pattern was used to implements the class. Notice the static inner class "AddressBuilder" which features many of the same functions (getters and setters) as the Address class with the additional build() function which checks that the AddressBuilder has all the required fields and returns the "approved" Address object

Capacity

Capcity class is simply class meant for organizing and checking data pertaining to the capacity of an OfficeSpace. Capacity throws and IllegalValueError if one of the datas being passed (either square feet, people limit, or workspaces) is 0 or less, which would not be allowed.

Contact

Contact, like Capacity is a simple class meant for organizing and checking data pertaining to a OfficeSpaceProvider's (User) phone number, cell number, email address etc.

FacilityType

FacilityType's main purpose will be to validate and control the updating and deletion of FacilityType instances.

Feature, CommonFeature, PrivateFeature

Feature is the parent class of CommonFeature and PrivateFeature. CommonFeature and PrivateFeature are categorized two different ways in the program, but are both just simply consist of a String identifier and will eventually have methods which control who and how their instances (collected in static Lists in each class) may be created and deleted based on Authorization Tokens

Name

the Name class has various fields such as firstName, middleInitial, and nickName which describe the name of the OfficeSpaceProvider. The purpose of this class is mostly of organizing and checking data pertaining to the name of an

OfficeSpace (includes static inner class OfficeSpaceBuilder)

OfficeSpace class is by far the most verbose of the classes. OfficeSpace Class represents the physical Office Space being offered by an Office Space Provider. Since there are a large number of fields, some required and some not, this class employs the Builder patter with its nested Class OfficeSpaceBuilder. In order to instantiate an instance of OfficeSpace, first instantiate an instance of OfficeSpaceBuilder and use the 'setters' to provide values for the minimally required fields. the minimally required fields are: capacity, 1 Rate in rates, address, officeOwner, facilityType and 1 image in images. after the minimally required fields are satisfied, Client may instantiate an OfficeSpace object by calling the build() method on the OfficeSpaceBuilder Object which will check to see whether the minimal requirements are met for an OfficeSpace Object, and if so, will return an OfficeSpace Object

OfficeSpaceProvider (includes static inner class OfficeSpaceProviderBuilder)

OfficeSpaceProvider represents a User (a class which is not addressed in this Sprint) who, in real life, has extra room in their house which they would like to rent as an OfficeSpace. In this class, each instance may have multiple OfficeSpace instances associated with it. OfficeSpaceProvider, like OfficeSpace and Address have a number of required fields and a number of non required fields. the required fields consist of name, contact, payPalAccount and an Image. due to the large number of required and non required fields, OfficeSpaceProvider employs the Builder pattern with its nested Class OfficeSpaceProviderBuilder. Im sure by now reading his document, it is clear how the Builder pattern has been implemented and OfficeSpaceProvider class is no different. For more description, refer to OfficeSpace or Address

PayPalAccount

PayPalAccount is a simple class meant for organizing and checking data pertaining to the PayPalAccount of an OfficeSpaceProvider.

Rate

Rate is a simple class meant for organizing and checking data pertaining to an individual rate offered for an OfficeSpace by an OfficeSpaceProvider. a Rate consist of a period in days and a rate which represents the dollars per day. The method getTotalCost() returns the rate over the full period.

Rating

an instance of a Rating class represents the feedback from a User (not yet implemented in this Sprint) towards a OfficeSpace or an OfficeSpaceProvider

Implementation Details

for this Implementation I notices that there where a number of classes, notably Address, OfficeSpace and OfficeSpaceProvider, which each had a number of required and optional fields. rather than telescoping the constructors and/or relying on the Client to pass in a large number of parameters for the constructor I made liberal use of the Builder pattern. I used the Builder Pattern in the 3 classes mentioned earlier and I found that it gives a degree of readability to the code, although it did seem to lengthen it, and it gave me a clear means to enforce the requirements on those objects, which would have been difficult otherwise

also noteworthy in this Implimentation is that I made the 2 "main classes" OfficeSpace and OfficeSpaceProvider the only ones with access through the API. I chose this in order to maintain the requirements as it provided less entry points. I believe having less entry points would be a good design technique in most any API. In the case of the Address class, which makes use of the Builder Pattern, I gave the client reference to an AddressBuilder object through the newAddress() function in the OfficeSpace class. Through this reference, the client is able to access the methods, including build(), in the Address.AddressBuilder class as well as reference to a new Address object which is returned by the build() method.

Both OfficeSpace and OfficeSpaceProvider are objects which may be rated, so I implemented a Rateable interface and implemented it in both classes. This interface provided simple get and submit methods for ratings as well as a simple averageRating() method which was to return the average 'stars' of the rating. Having both OfficeSpace and OfficeSpaceProvider implement Rateable made it simple to have an association to an object implementing Rateable to represent the subject of the Rating, rather than having to rig it to need 1 of the two classes or something shady like that.

Use Cases

The OfficeShare API supports 2 primary use cases;

1) Administrators update and review collections such as CommonFeatures and SharedFeatures
2) Office Providers may list, udate, and delete Office Spaces as well as their own account
3) ***not implemented in this Sprint*** Office Renters may create, update and delete their own account as well as browse and book Office Spaces (Office Providers may have this same functionality

Risks

Security will be an issue when deployed onto a network.

Storage will quickly become a problem unless a datebase structure in implemented