## Team name and list of team members:

Team name: **Voltron**Team members:

* Mihail Aleksiev
* Nilolay Genov
* Stamo Petkov
* Svetlin Panov
* Todor Miroleskov

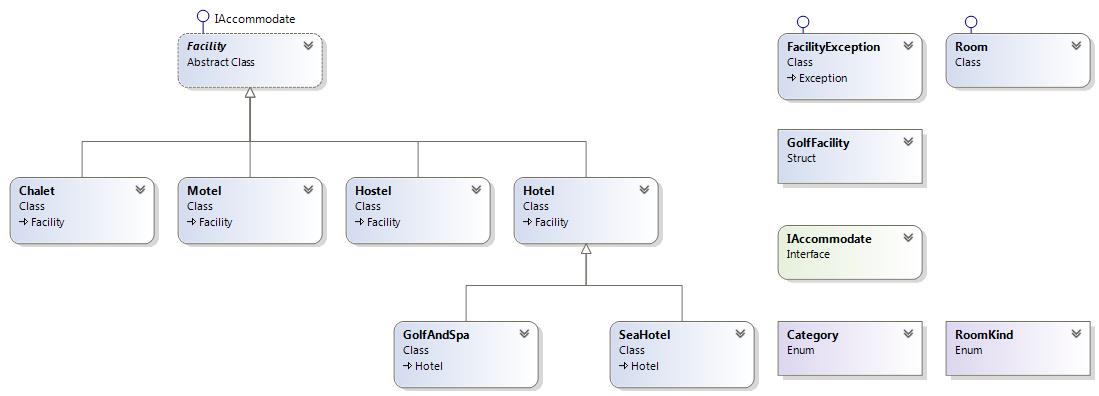
## Project purpose:

Our project consists of two parts.

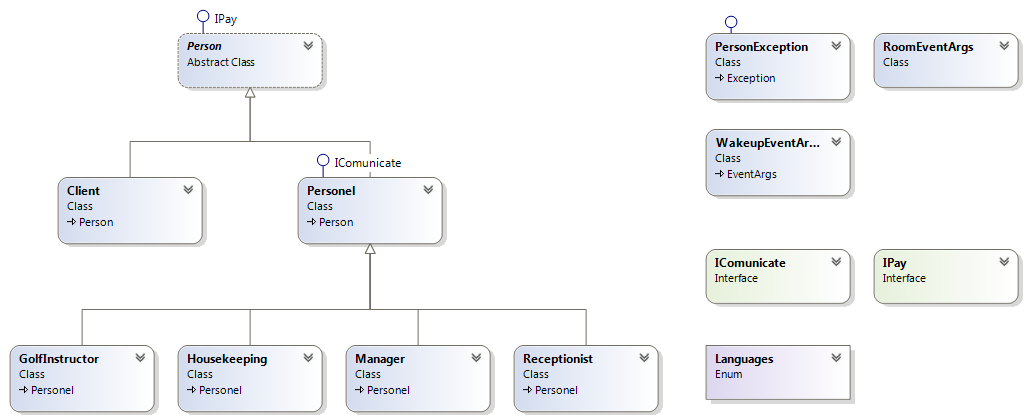
The first part is about a different types of facilities, their personal and the people who want to take a rest in some of the facilities. First of all we implement abstract class named “Facility” which is parent of “Hotel”, “Motel”, “Hostel” and “Chalet” classes. The classes “Sea Hotel” and “GolfAndSpa” are inherited from the “Hotel” class. For the “Room” class we have implemented a couple of properties that say the number of the room, if the room is cleaned or not, if it is free or not, the numbers of the beds in it, and if it is a single, double, triple room or an apartment. We have also put enumeration for the facilities. You can pick from one star to five star holiday house.

In the second part of the project we implement a couple of classes with few methods in them. The main class here is “Person” which is inherited by the classes “Client” and “Personal”. For example every facility has its “Personal” which is parent of the “Receptionist”, “Housekeeping” and “Manager” classes. Every person in our project, no matter personal or client, has its ID and Name. The receptionist can checkin / checkout clients using methods of the “IAccommodate” interface, and also to check how much money the client have to ensure that if he/she can afford a room or not and to take money from the customers’ bill by the interface “IPay”. Also the receptionist greets the client by the interface “IComunicate” in four different languages which are defined in the enumeration “Languages”. The “Housekeeping” class is defined to clean the rooms and is called by the event “CleanRoom” when a room is freed. The “Manager” class has methods to hire and release personal. The trick here is that only the “Facility” class can hire a new manager.

## Class diagrams:



**Facility class diagram.**



**Person class diagram.**

## The URL of your TFS repository:

<https://stamopetkov.visualstudio.com/>

## Other information:

An interesting method is SayIt() – method of the “ICommunicate” interface. It takes as parameters Language in which personal wants to say something and a phrase in predefined base language and translates the phrase via Google Translate. Unfortunately Google has closed it’s Translate API, so we needed to find a workaround. It is using Webclient with phrase and language pair sent as GET parameters to request the translation page from Google Translate. After receiving the page, the Webclient have to convert it from char[] to string and to do this it have to know the encoding. The worst thing is that Google returns different codepages for different languages, so we needed to specify codepages for converting char[] to string. Than it is using SubString method to cut the info we don’t need and returns translated phrase. If there is no internet connection, the method will throw PersonalException with message that says “I can’t translate it!. No connection to my brain.!”.

In our project we use the so called Factory Method Design Pattern. It defines an interface for creating objects, which are created during the run-time of the program, the type of the objects is defined by its subclasses. For example when a client checks in a hotel the receptionist gives you a key to a room, and in this case the client can be looked as a “room” factory.

We implement the Facade Design Pattern, which is also called the façade pattern. Its purpose is to provide simplified interface to a long body of code. In this way we can easily operate, test and understand a big amount of code such as a library.

The Composite Pattern is used to show a tree structure, and every element in this structure can perform some tasks. For example in our project this is the “Person” class which is divided into three separate classes – “Receptionist”, “Manager” and “Housekeeping”.

For running through a container and access it’s elements we use the Iterator Design Pattern. It provides a way to access collections elements sequentially without exposing its underlying representation.