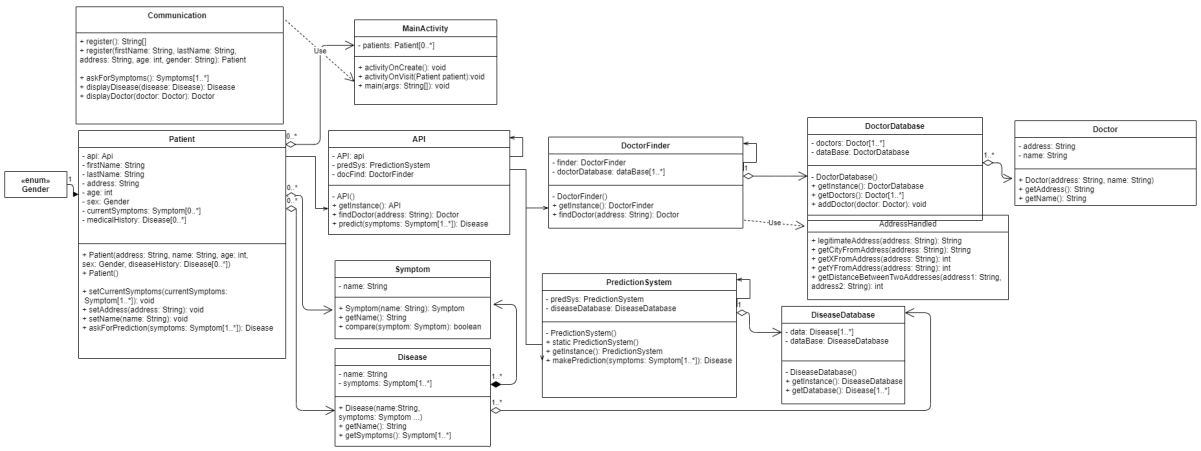
Application Description

The app serves users with two main functionalities - making predictions about diseases based on symptoms and finding doctors nearby. The app uses a simple database consisting of Diseases and their symptoms; in order to predict the illness, it matches the symptoms with the database and returns the disease with the highest number of matches. For the doctor finding system, it compares the address of the patient with the address of the doctors inside the database and it returns the nearest found doctor.

UML Diagram



Please click here to see the diagram in full size.

The diagram consists of 11 classes. All crucial methods and functions are included in the diagram. MainActivity uses static methods from the Communication class for printing messages to users and reading input from them. In the MainActivity there are stored Patient's objects which are denoted by the aggregation arrow. The Patient has aggregated its Symptoms and Diseases. There is an enum Gender which indicates the sex of the Patient. Since symptoms must have some cause and they cannot exist without any Disease their relation with Disease is composition.

Patient calls methods from the API class with which it is associated. API is associated with Singleton classes PredictionSystem and DoctorFinder. Prediction system has DiseaseDatabase with at least one Disease. Since the Disease is independent of the database they are connected through aggregation.

The DoctorFinder class uses static methods from the AddressHandler and it is connected with DoctorDatabse via an aggregation relationship. The DoctorDatabase consists of Doctor objects; because the Doctor is independent of the database, they are aggregated as well.

Program Description

Core system

There are 3 core classes that are in charge of the application's main task - API, DoctorFinder and PredictionSystem. All those 3 classes implement the singleton design pattern since there is only one such system that handles requests for all users. Moreover, the Singleton design pattern disables the creation of new unnecessary instances of the API, DoctorFinder and Prediction system, which might be heavy and slow down the program. Using Singleton also ensures that the instance will not be replaced (might be accidental) by a different object which could have caused the wrong functionality and lead to non-accurate predictions and searches.

API

The class API provides the user with access to the two systems/classes - DoctorFinder and PredictionSystems (following the Single responsibility principle). For every user, the same database for finding doctors is used as well as the same database of diseases is used for all predictions.

DoctorFinder

This class finds the nearest doctor nearby the patient base on their addresses. Furthermore, it provides the patient with a doctor's credentials and address, so that the user can easily visit the doctor personally. Inside the DoctorFinder there is a database, which consists of objects of Doctor class. There are only general practitioners in the database, since the application models real-life use-case, where the patient must at first contact some general practitioners and only after that they can visit specialists (inspired by the health care system in the Czech Republic).

Prediction System

This class compares a Patient’s symptoms with its database and returns the most probable disease. The class is opened for being used with new illnesses since the database of illnesses and the prediction system are separated.

Patient

The patient class represents users of the program. Every patient has some personal information, such as name, surname, age, sex, current symptoms and history of past diseases. They can edit the data, if necessary in their profile. Every patient is guaranteed to have access to the Api class, from which they can request the prediction of diseases or find the nearest doctor.

Doctor

Doctor class consists only of address and name since it is a representation of a business card of the general practitioner inside the system. The Patient can request the contact for the doctor so that they can make an appointment in person.

Communication

The Communication class handles the I/O between the system and the user. Its separation from the main logic of the program allows easier change of the code for different platforms.

MainActivity

Taking into consideration aspects of the Android app, we have created a MainActivity class that imitates the behaviour of the Android structure. This class consists of two important methods - activityOnCreate, which initializes the data in that class e.g. it registers the users, and second activityOnVisit, which run the whole system. In order to test and show the functionality of our program, we have included also the main method in this class.