Let’s take the design under consideration.

Basing on analysis, Model-View-Controller architecture pattern has been chosen. As the name suggests, the system has been divided into 3 parts: model, view and controller.

1. Let’s start with taking a closer look at the Model. Model is a part of the system that contains all classes describing the logics of it. As the class diagram shows, these are 17 classes. The classes are connected either by a simple association or by inheritance. You may ask why we have chosen association over composition. The reason of that is that we wanted to have only one instance of every object. It can be explained on the example of modifying an event. The changes are entered in the view and are expected to be implemented in the eventList as well. In the case of using inheritance there are two super classes: the event class, which is an abstract class and the Participant class. Taking a closer look at the event class it can be seen that it is a parent to 4 classes: Lecture, Seminar, Workshop and Trip. They differ to take an example by the amount of lecturers. As I have mentioned before, the Event class is an abstract class. The reason of that is, that it shouldn’t be possible to create an event on it’s own, but at the same time, the four mentioned classes (outline) share most of the fields and available methods. Another approach was used with the Participant class. It’s subclass is a member. Both a member and a participant can exist separately on their own and the difference between them is that a member is an upgraded participant, with more fields and methods.  
   Furthermore, this part of the system contains 4 singletons, which are: eventList, memberList, lecturerList and VIAManager. VIAManager is however not only a singleton, but also a Façade. All information in the remaining 16 classes can be reached from VIAManager. This design pattern has been chosen due to the choice of MVC architecture pattern.

What is more, although the File Manager can be considered as a separate part of the system, it can be also included in the Model. It is connected with the rest of the classes through VIAManager and is responsible for secondary storage of information. More about how it is done, will be said by Michał in the implementation part.

1. The following part of the system is the view. View is an user interface dedicated part. It was developed using the Swing library for GUI, due to the fact, that every member was familiar with it and it is more beginner friendly comparing to JavaFx. In order to achieve full control of what is displayed, Widow Builder hasn’t been used. The thought behind the layout was simplicity, clarity and Buddhistic spirituality. The application is divided into decision panels, form panels and lists. In decision panels user decides where to go next. Form panels are for creating events and signing up members and lecturers. List panels have informative function. They display content of the system. Specific instructions about how to use the application are concluded in the user guide.
2. Last but not least is the controller- a singleton that connects all parts of the system together. It is illustrated by the sequence diagram of searching for lecturers. In the view the user inputs searched phrase. Then The view calls the controller, which calls the Search engine. The search engine calls different classes and at last, it returns a ready ArrayList of searched lecturers back to the controller, which passes it further to the view where the information is being displayed.

Before ending the design part, I’d like to add that to make the system organized and clear, the code has been divided into 4 packages, each dedicated to a specific part of the system: model, view, controller, and in this case also file manager.