The implementation process started right after completing design, according to the waterfall model. Work has been divided and one part of the group started with the model classes, while the other part was designing GUI and proceeded with its implementation. We found this approach more efficient due to the parallel system development. Moreover, having too many people at one task was problematic, because of merge conflicts in git and coding ideas missmatch.

Now I will present few interesting code examples. First one is the Member List. This class is responsible for primary storing all members signed up to VIPASSANA, as an ArrayList.

Lists classes, beside storing objects, provide basic operations on lists, such as adding and deleting objects, searching through lists with desired criteria or generating new, specialized lists. Here two methods can be seen. getListOfMembersWhoHasntPaid() searches in the list and checks if a member has paid. If not, the object is added to a new list, which will be returned and passed further inside the system. getListOfEmails() generates an email list of all members. It is used when the administrator wants to send a newsletter.

Another important part of implementation is the Controller. This is connection between model and view. One of the most extensively used type of method is the one, returning DefaultTableModel. Hence, we are using tables to present data in GUI, table model is needed. When a table is about to be created, view calls one of the static methods in the controller which returns an appropriate model for the table. Firstly, column names are defined. Secondly, we get desired data from model using VIAManager, and the list is parsed into a two-dimensional array, where each row represents one table entry. Thirdly, model is created. However, we want to have more control over the table and to achieve it, DefaultTableModel class is extended with an anonymous inner class, overriding getColumnClass() and isCellEditable() methods. In getColumnClass() the type of each column is defined. For example, column with index 3 is type Boolean, which will result in displaying it as a column of checkboxes. In isCellEditable() editing is only enabled for the column with checkboxes.

The last class presented is FileManager. Here is saving to and reading from files handled. The system has two types of reading data, binary and text files. The purpose of text files reading is to load initial data into system. It was much faster to write methods for this and prepare text files, then to insert data manually. This approach was very helpful during testing, when we already had data to work with. In this case, information about member objects are written in a text file in a specific format. Each line is read, separated and information is stored in variables. When all data is retrieved, Member object is created and added to the list.

The second part of FileManager is binary files reading and storing. Since all of the classes in the model implement Serializable interface, they can be stored as a list object into binary file using Java Object Serialization.

readEventFile() method is an example of reading binary file using ObjectsInputStream. Firstly, ObjectInputStream instance is created using FileInputStream. Secondly, the whole EventList object is read. Due to using IDs in the system, ID in Event class must be restored.

And that is all from the implementation part. Now Miśka will talk about the process.