

**Organizing Event and Volunteering System**

**S18052 – Elif Dilara Aygün**

**Contents**

[User Requirements 3](#_Toc43254250)

[Use Case Diagram 4](#_Toc43254251)

[Analytical Class Diagram 5](#_Toc43254252)

[Design Class Diagram 6](#_Toc43254253)

[The Scenario of selected use case 7](#_Toc43254254)

[The Activity Diagram for Organizing an Event 8](#_Toc43254255)

[The State Diagram for Event class 9](#_Toc43254256)

[The Sequence diagram for Organizing an event 10](#_Toc43254257)

[The GUI Design 11](#_Toc43254258)

[Discussions about design decisions 13](#_Toc43254259)

# User Requirements

The software is developed for organizers, volunteers and sponsors to use. By using the software, it will be easier to organize an event, volunteer in any event and be a sponsor.

The organizer is able to organize an event for the specific date, change the date of the event or remove the event from the system. Event can be free or paid. In case of free event organizer should specify the promo code for it. They can organize more than one event. Organizer may also join an organizer’s group and be the manager of any group. The organizer of the event can be a company or a person.

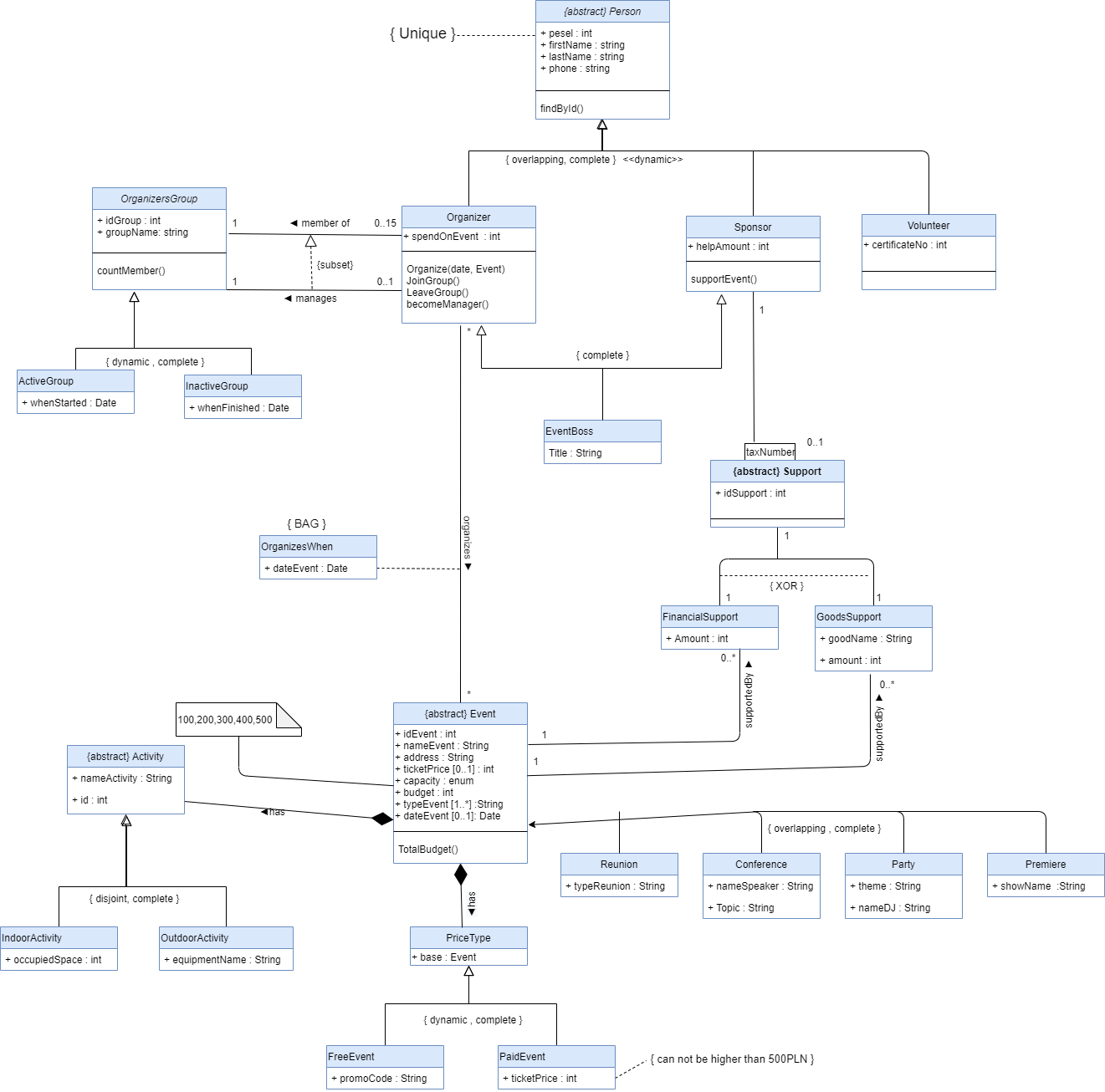
The volunteer is able to display current events, choose an event to volunteer among them. They may volunteer in many events at different times.

The sponsor is able to see events and choose one among them to support. The sponsor can support event financially or provide some goods which are required for the event.

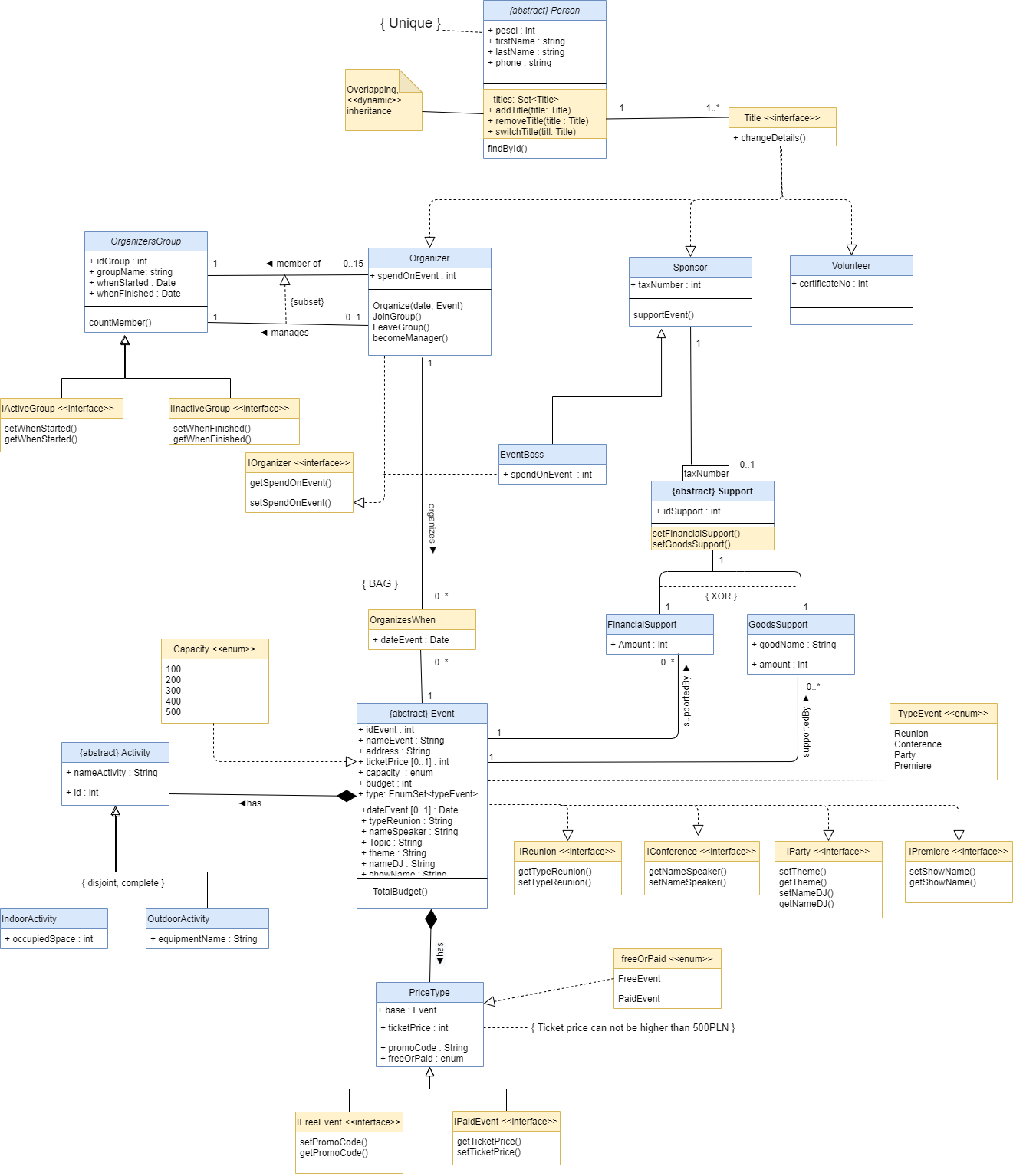
# Use Case Diagram

******

# Analytical Class Diagram

******

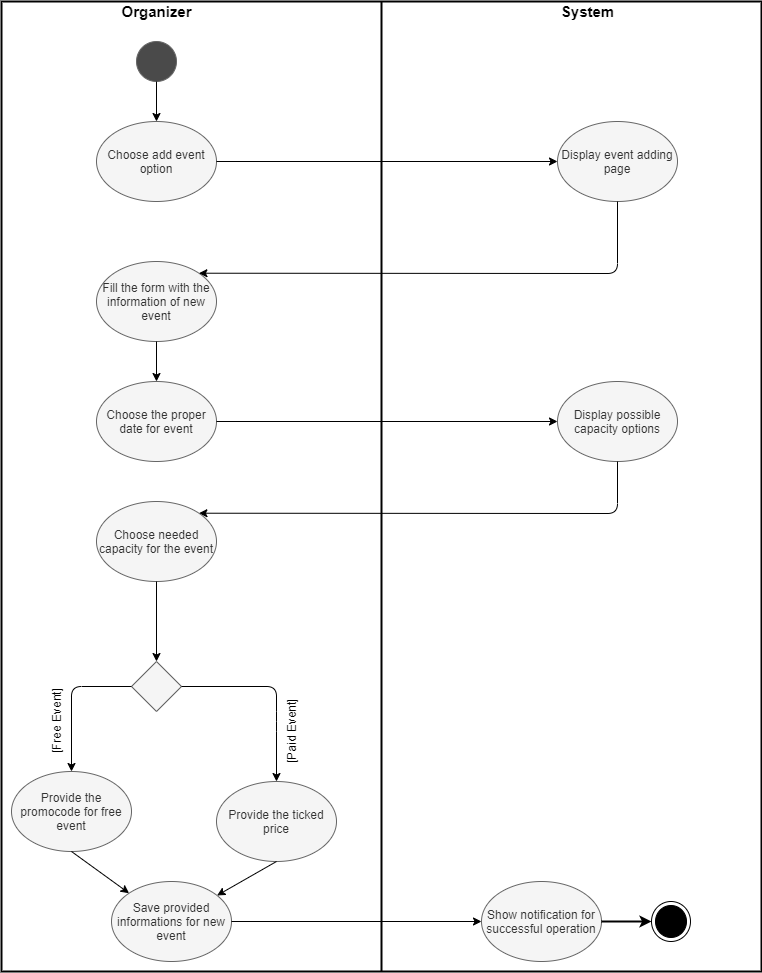
# Design Class Diagram

******

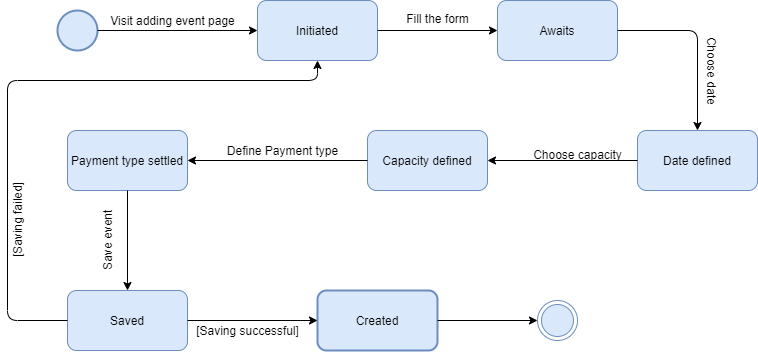
# The Scenario of Organizing a new Event

Organizer would like to organize a new event. In order to do this, organizer should basically add new event with defining proper date for it. So Organizer starts with choosing the add event button on the main page. And when the adding new event page is displayed, he should provide the right information for new event such as name of the event, address, etc. Then organizer must define the proper date and choose the needed capacity for the event. Once it is done, the organizer should choose the payment type. Payment type can be either no payment which is for free events or paid event. For both cases, organizer should provide more information such as promocode for free event and ticket price for paid event. After filling the form fully, organizer can save it to the system. Ones it is saved, there will be notification pop-up for successful operation.

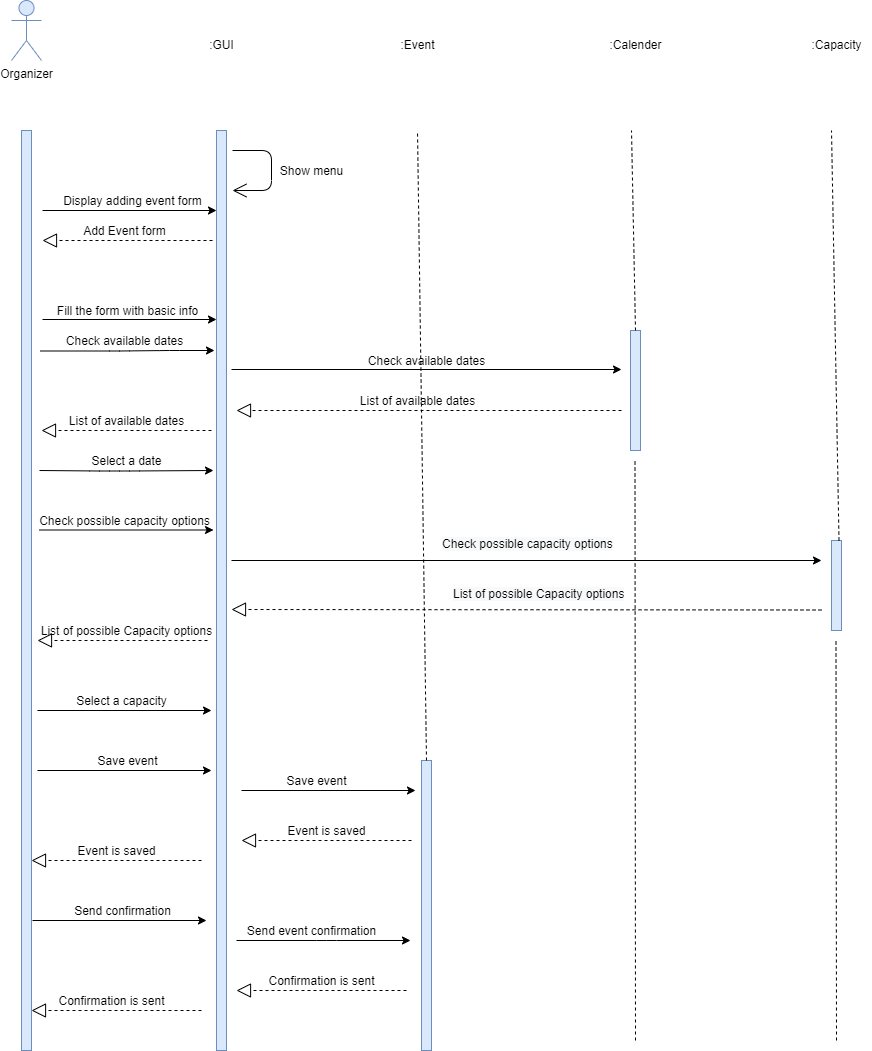
# The Activity Diagram for Organizing a new Event

******

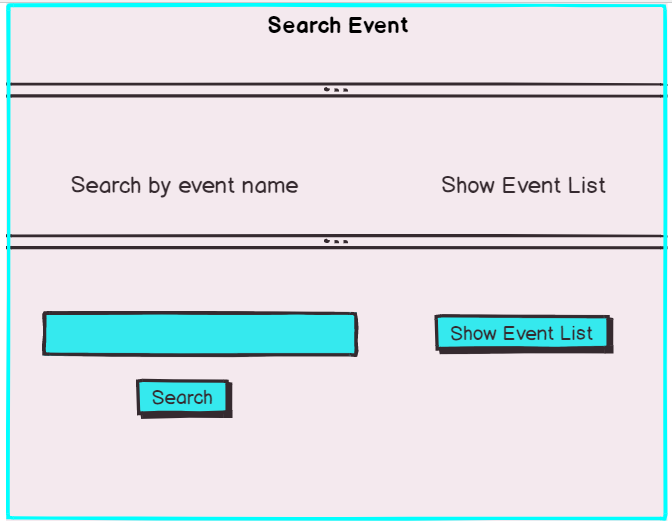
# The State Diagram for Event class

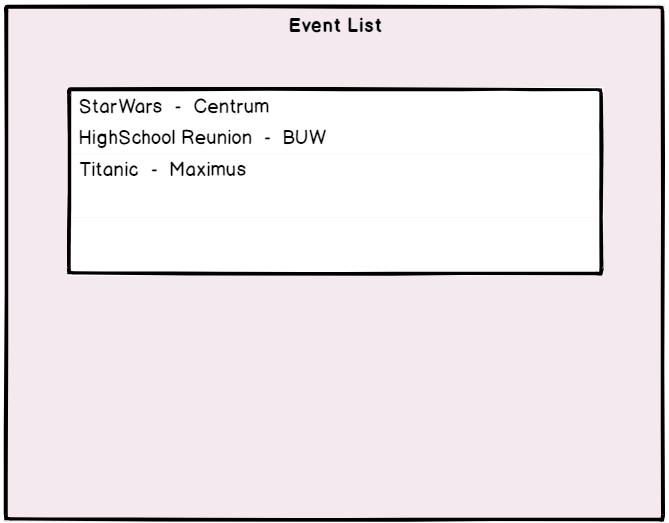
******

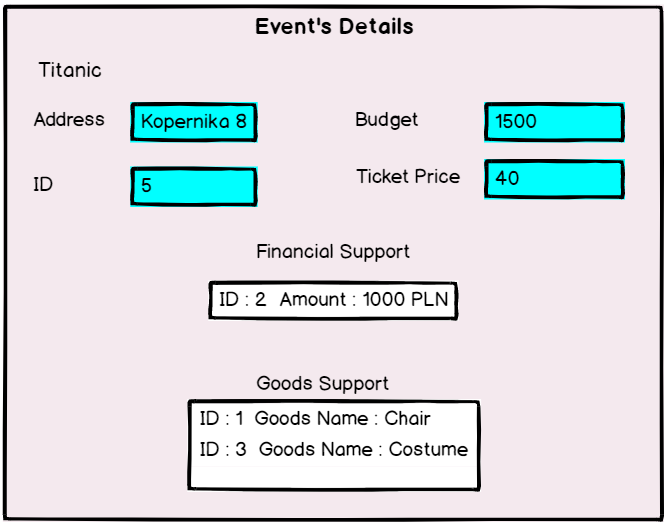
# The Sequence diagram for Organizing a new event

******

# The GUI Design



******

******

# Discussions about design decisions

* **Dynamic inheritance in OrganizersGroup class**

This functionality require a little manipulation of the structure. Dynamic inheritance allows objects to change and evolve over time. Since base classes provide properties and attributes for objects, changing base classes changes the properties and attributes of a class. In this system the organizer’s group maybe either active or inactive. And it is two-way possible change. Active group can become a in active group after defining the finish date or the other way around; inactive group can be an active group by specifying the start date. In order to implement such inheritance, flattening hierarchy approach has been used. OrganizersGroup class can implement two interfaces with required methods. IActiveGroup and IInActiveGroup. While creating the OrganizersGroup we should define the status of the group. And later on we are able to switch the status.

* **Activity class as composition**

Composition is a “belongs-to” type of relationship. So the object is part or member of the other object and their **lifecycles are tied. It means that if we destroy the owner object, its members also will be destroyed with it.** In our case it is not possible to create the activity without the event. The activity class has a private constructor and static createActivity() method. It is the only way of initializing the object of Activity class. This method requires Event object as a parameter. And it can not be null. So in the end if we destroy the Event object, the Activity will be destroyed as well.

* **XOR constraint in Support class**

XOR constraints is used to present some restrictions. In our case, the support can be either FinancialSupport or GoodsSupport. It can not be both at the same time. And it will be implemented using methods in the support class, setFinancialSupport() and setGoodsupport().

* ***Dynamic, Overlapping inheritance in Person class***

Overlapping means that an object may have 1+ roles at the same time. And dynamic means, those roles can be switched by time. In our case, the system knows about People that can be Organizers, Sponsors and Volunteers. A person have to belong to one of these classes but can belong to two or three, i.e. one Person can be a Organizer and a Sponsor at the same time. The implementation will be done by having a Title interface, Set<Title> to store titles and adding more to it. In case of dynamic changes we use switch(role : Role) method.