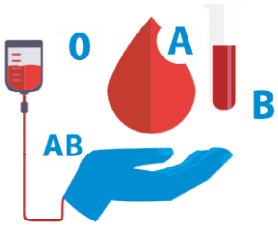


Alicia du Buisson

Information Systems Management 354 Project

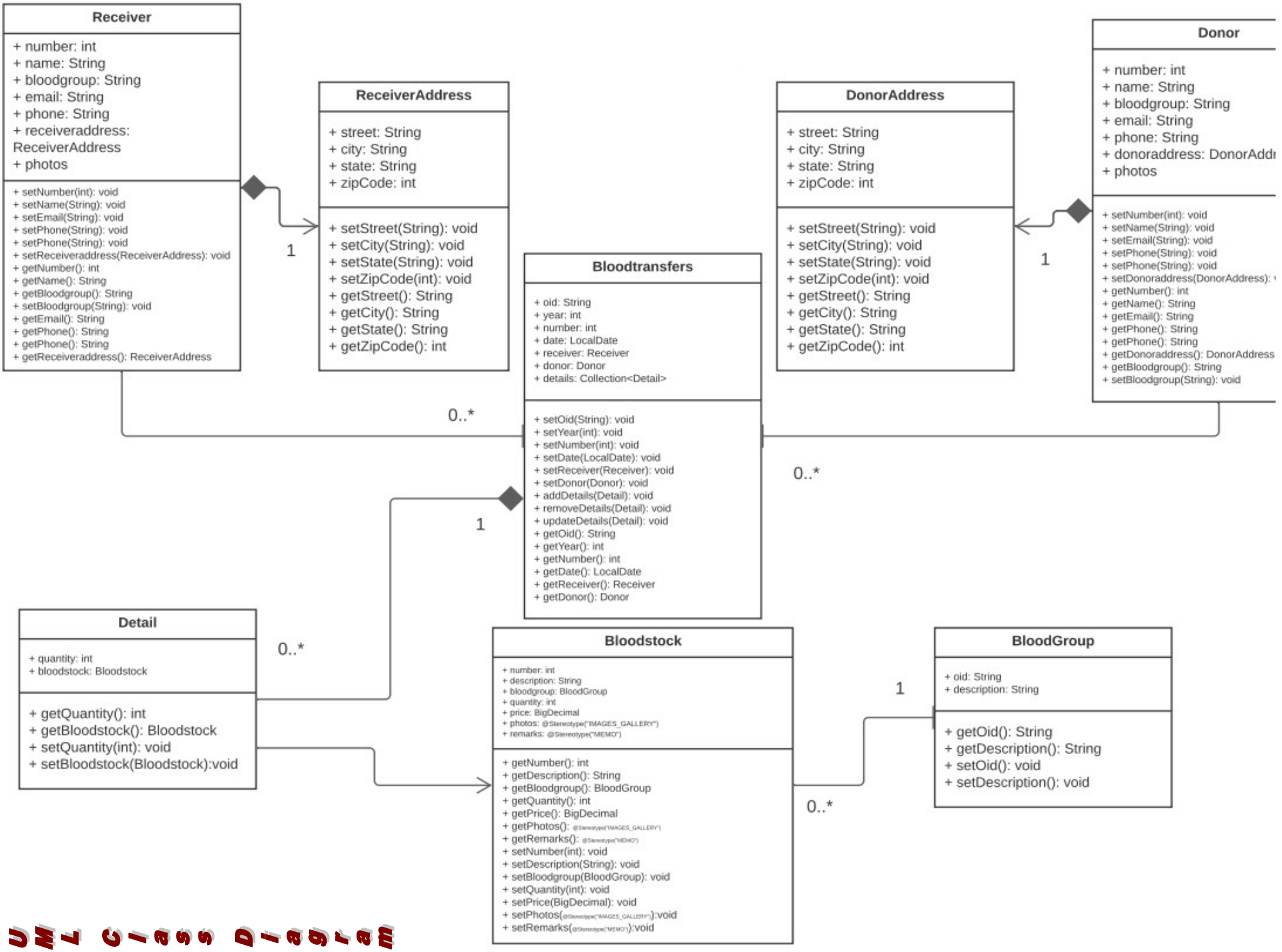
Blood Bank Information System

November 2021

High-level overview

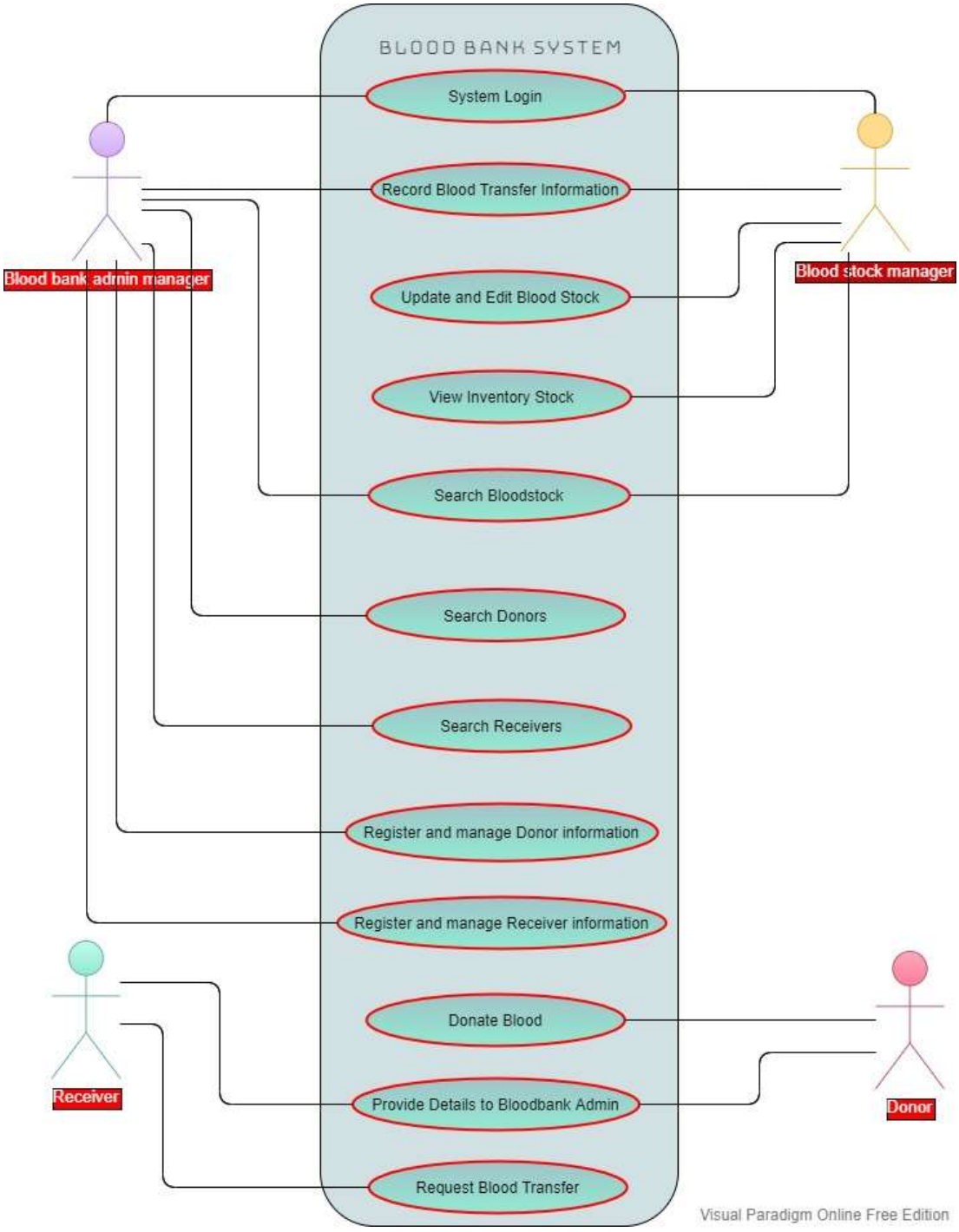
The prototype I wanted to create main purpose was to address the challenges faced by blood banks by providing a management system to manage their information and data more effectively. Blood banks make sure that donated blood are preserved and kept safe and that the supply of every blood type is available for patients for a general or emergency blood transfer. Blood transfusion safety and efficiency is still a big health issue in many parts of the world. Many blood banks still make use of paper based systems to store and manage their information. This is makes room for a lot of error. It is neither efficient nor effective and it puts a lot of patient's lives in jeopardy.

Blood banks are data driven organisations. The data of patients and donors and keeping track of bloodstocks is a very important aspect in the healthcare system. A lot of people's lives depend on the data safety and the efficient storing and management of these records. Creating an online blood bank information system to make the blood bank more efficient and to better the lives of patients seemed like a very purposeful project to take on. The Information System provides the blood bank with a web based application to manage, edit and administrate valuable data of donors, patients and the blood bank stock in a quick and user-friendly manner. Overall the system assists in the management of blood transfer and donor data that needs to be stored in to the database for medical record regulation, to find patient-donor matches and to assist in patient identification. The system also makes it possible for the blood bank to add and register donors and patients into the database. The database can also be used search for donors and receivers who can be candidates in a blood transfer according to their blood group/type and to search for the blood stock in the database of the blood bank. The stock of the blood bank is stored into a database and this is also very useful to the blood bank, since blood banks can also make this data available or authorize certain medical professionals or hospitals to have access to this data. By doing this they can also search availability and stock of blood for patients in need and request for patients to be registered into the database. It also provides donors with medical records of their donations and provides patients with their medical record. Reports of donors, seekers, total consumption of the blood units etc. can be generated as a pdf. The system has a lot of built in features like search engines to search the database for donors and patients with the same blood type, other identifying attributes can also be viewed from the interface and taken in to account when selecting potential donors and receivers for a blood transfer, for example close distances to each other (address) for emergencies and convenience and available bloodstock. The systems allows for the date, year and extra descriptions regarding blood transfers to be stored. With the efficient way to storing and managing this data, the system will improve blood transfusion efficiency, confidentiality and overall safety. A web based applications like this will reduce the probability of error and this will have a profound impact on people's lives.

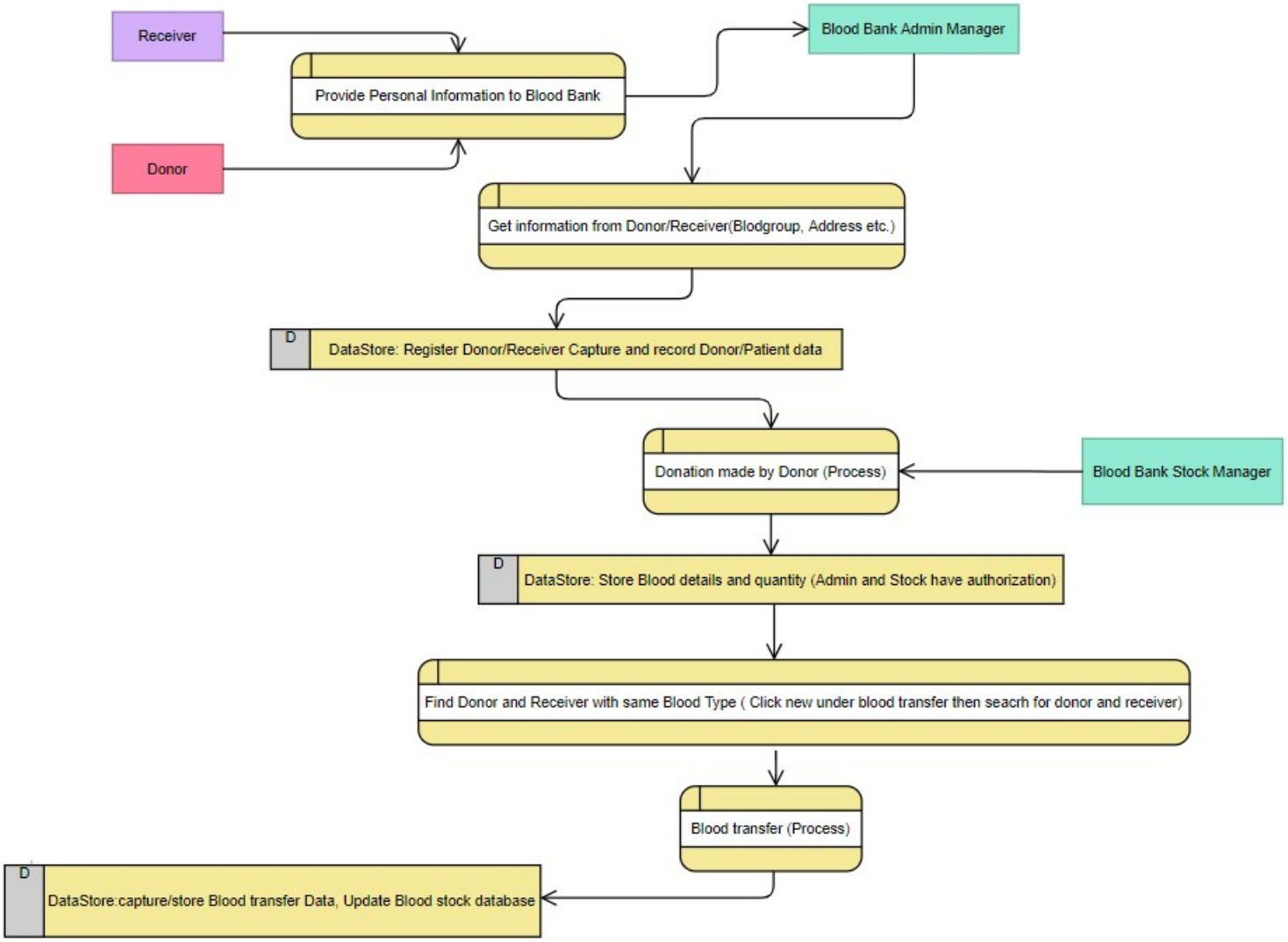


UML Class Diagram

Additional UML (1) : Use Case Diagram



Additional UML (2) : Data Flow Diagram



Code Explanation

Once the folder 21569479.zip is unzipped the contents will be available to open or import into OpenXava. Two folders will appear in the unzipped folder MACOSEX and bloodbank. The bloodbank folder contains a com folder com.yourcompany.bloodbank.model contains my 8 Java files containing my various classes. Once the bloodbank folder is run the output link can then be copied from OpenXava http://localhost:8080/Bloodbank into a web browser. The login details are as follows Username: admin, Password: admin

My code was written in OpenXava with the goal to create a web application for the maintenance of blood bank blood transfers between Donors and Receivers. I first identified my classes: Blood Transfers, Bloodstock, BloodGroup, Donor, Receiver, and Donor & ReceiverAddress. I declared a unique primary id for each of them by using the @id annotation in OpenXava. After analyzing how a blood bank works and creating a blueprint. I added the attributes and methods required for each of the classes in a UML diagram and started coding. The UML Diagram showed me the relationship i.e. @ManyToOne, @OneToOne between these classes. The OpenXava library annotation automatically maintains the attributes and generates a graphical user interface. By using the different annotations, my program can handle all basic updating, deletion, and searches in the database. The SQL code is automatically generated by OpenXava and in order for me to clear some errors with regards to deleting empty faulty fields, I had to use the built in Database Manager that OpenXava provides, to edit the database using SQL code commands. I created the two main entities in Bloodbank namely the Donor and Receiver. I added their information i.e. unique ID, name, BloodGroup, Address, Remarks and Photos (just creating attributes). Further, for the privacy purpose, Photos are not required, but fields like name and bloodgroup I used @required annotations, because these should be mandatory in able for the system to function correctly. By using @Embeddable instead of @Entity, I created two embedded Address classes for Donor and Receiver class, in this way DonorAddress will get deleted when we Delete Donor. Then I added the BloodStock entity for the Bloodbank, which can store the quantity and details of received blood. It can also be updated and edited by the admin manager on the interface. I created the @ManyToOne relationships between donor, receiver and bloodtransfers. When navigating to blood transfer and recording data when a new transfer is taking place (create new entity), the admin must choose a donor and receiver with the same blood type, by searching the database of all the registered Donors and Receivers. That's why I added their blood type attribute to a column to in the information table when searching for registered Donors and Receivers. I used @stereotype to specify the specific behaviour for each property. I wrote the code in such a way (@ElementCollection) that while creating the new blood transfer, the user can view and search all the bloodstock available in the database. The BloodGroup class keeps data about the category/blood group of the blood and if the blood bank is out of blood stock for a particular blood group it can be

written in the description, then the blood bank can contact or search for donors of that blood group/type. In case of emergency, they can check for donors with nearby address stored in the database. The date on which the transfer is made is automatically generated/calculated through the NextNumberForYearCalculator module. I imported the java time library for this and used the @DefualtValueCalculator annotation. I used the @View annotation to define the layout of attributes like number, date, year etc. when for example creating a new bloodtransfer record/entity.

Additional Features

There can be files or photos added under donor and patient information by browsing the computer or dropping files in the space provided. This helps to identify and store extra information on donors and receivers. This way a donor can also be identified by their photo/ID document. In order for the mail to be correct I have written @Stereotype("EMAIL") this stereotype ensures that the user types in a valid email, if the user types in an invalid email the user will be notified and required to type in a new one. The same goes for when a user enters a phone no, it must be valid. When a blood transfer is being made or added to the database the date on which the transfer is made is automatically generated through the NextNumberForYearCalculator module. I have added two embedded classes for the address of the receiver and Donor because it can make the system more maintainable , since addresses change quite often and can then be easily accessed and edited, it also makes deletion more “clean” because when the donor/receiver is deleted the Embedded class Donor/ReceiverAddress will also be deleted.