The Stuff Lending System

A large problem for the environment is that we have too much stuff that we seldom use or need. It is wasteful to buy items that you use once or twice, instead it would be better if we could borrow such items from our friends, coworkers or neighbours etc.
One problem that arise in this situation is to know where the stuff actually is at a particular moment in time, what stuff is available for lending etc. In this assignment you will create design and implement the basics of such a system.
The general idea is that you have a number of members that offer a number of items for lending. For every item they get a number of credits in the system. Credits can be used to lend items. The items have a cost per day credits are transfered from the lender to the item owner when a lending contract is established. The lending contract defines the item and the time period (days) when the item should be picked up and returned to the owner. This establishes a basic economy in the system; mebers add value by adding items, and lending them to others, and can in return lend more items themselves.
A special case is when an owner of items needs to reserve an item they themselves own (i.e. the item is no longer available for others to lend) in this case the owner does not pay any credits.
Currently the system does not need to handle bad intentions such as registering non available items, not returning items etc.

Creating a working economy is a hard task and the goal here is merely to offer a basic proof of concept that could later be tested more in depth.

Task

Design and implement the stuff lending system. Implementation (Java source code), class-, object- and sequence-diagrams are to be created and presented. The sequence diagrams should show how a model-view-controller separation is achieved (i.e. start in the UI) and how the different requirements are met. The design and implementation should match *perfectly*. The focus is not to create a very usable or fancy user interface but to have a robust and well-documented design that can handle change and follows the MVC, GRASP and possibly GoF patterns. The application should be a java console application.

OBS: It is not permitted to use any type of framework, however, the standard class libraries, etc. are permitted. Basically, you should design and code your own application.

Functional Requirements

1. Member

    1. Create with a name, email and mobile phone number. A *unique* member id should be generated and assigned to the new member and the day of creation should be recorded.
        1. The member id should be 6 alpha-numeric characters.
        2. The email adress and phone number needs to be unique (no other members can have the same email or phone number).
    2. Delete a member.
    3. Change a member's information.
    4. Look at a specific members full information.
    5. List all members in a simple way (Name, email, current credits, and number of owned items)
    6. List all members in a verbose way (Name, email, information of all owned items (including who they are currently lent to and the time period))

2. Item

    1. Create a new item for a member, the item should have a category (Tool, Vehicle, Game, Toy, Sport, Other), a name, a short description, the day of creation should be recorded, and a cost per day to lend the item.
        1. When created the owning member gets 100 credits.
    2. Delete an item
    3. Change an item’s information.
    4. View an items information including the contracts for an item (historical and future)

3. Contract

    1. Establish a new lending contract with a starting day, an ending day and an item.
        1. Credits should be transfered according to the number of days and the price per day of the item
        2. Can only be done if the lender has enough credits.
        3. Can obly be done if the item is available during the time period

4. Time

    1. Time is handled as a day counter, 0 is the first day and is set when the system starts. Time is not connected to the system in this proof of concept.
    2. Advance day. In order to properly test the system there needs to be a way to advance the current day without relying on the system time.

5. *Prepare* the design for persistence, i.e. add a persistence interface. Implement a hard coded "loading" of some members with items, i.e. create some hard-coded data. You should **not** implement any persistent loading or saving to file or database etc.