4. BOOK BANK SYSTEM

Aim: To define problem, requirements and model a Book Bank System.

Problem Statement:

A Book Bank lends books and magazines to member, who is registered in the system. Also it handles the purchase of new titles for the Book Bank. Popular titles are brought into multiple copies. Old books and magazines are removed when they are out or date or poor in condition. A member can reserve a book or magazine that is not currently available in the book bank, so that when it is returned or purchased by the book bank, that person is notified. The book bank can easily create, replace and delete information about the tiles, members, loans and reservations from the system.

Requirements Specification:

Identifying Functional Requirements

Given a problem statement, the functional requirements could be identified by focusing on the following points:

- Identify the high level functional requirements simply from the conceptual understanding of the problem. For example, a Book Bank system, apart from anything else, should be able to issue and return books.
- Identify the cases where an end user gets some meaningful work done by using the system. For example, in a book bank, a user might use the "Search Book" functionality to obtain information about the books of his interest.
- If we consider the system as a black box, there would be some inputs to it, and some output in return. This black box defines the functionalities of the system. For example, to search for a book, user gives title of the book as input and get the book details and location as the output.
- Any high level requirement identified could have different sub-requirements. For example, "Issue Book" module could behave differently for different class of users, or for a particular user who has issued the book thrice consecutively.

From the above problem statement, even without doing any deep analysis, we might easily identify some of the basic functionality of the system: New user registration, user login, search book, reserve book, purchase, notify, issue, return, pay due, remove old book, update, replace, delete, user logout.

Identification of non-functional requirements

Let's try to identify a few non-functional requirements.

- **Performance Requirements:**
    - This system should remain accessible 24x7
    - At least 50 users should be able to access the system altogether at any given time
- **Security Requirements:**
    - This system should be accessible only within the institute LAN
    - The database of Book bank (BB) should not store any password in plain text -- a hashed value has to be stored
- **Software Quality Attributes**
- **Database Requirements**
- **Design Constraints:**
    - The BB has to be developed as a web application, which should work with Firefox 5, Internet Explorer 8, Google Chrome 12, Opera 10
    - The system should be developed using HTML 5

UML models:

Star UML tool is used to transform requirements into UML models and refine them into designs. A new model is opened and renamed as BookBank. The following models are designed to represent the requirements stated above.

Use Case diagram:

Association between Actors and Use Cases

A use case is triggered by an actor. Actors and use cases are connected through binary associations indicating that the two communicates through message passing.

An actor must be associated with at least one use case. Similarly, a given use case must be associated with at least one actor. Association among the actors are usually not shown. However, one can depict the class hierarchy among actors.

Guidelines for drawing Use Case diagrams

Following general guidelines could be kept in mind while trying to draw a use case diagram:

- Determine the system boundary
- Ensure that individual actors have well-defined purpose
- Use cases identified should let some meaningful work done by the actors

- Associate the actors and use cases -- there shouldn't be any actor or use case floating without any connection
- Use include relationship to encapsulate common behaviour among use cases , if any

Actors identified in this problem are Member, Non-member, Admin or Owner, StaffMember.

Class Diagram:

Relationships

Existing relationships in a system describe legitimate connections between the classes in that system.

- **Association**

  It is an instance level relationship that allows exchanging messages among the objects of both ends of association. A simple straight line connecting two class boxes represent an association. We can give a name to association and also at the both end we may indicate role names and multiplicity of the adjacent classes. Association may be uni-directional.

  Example

  In structure model for a system of an organization an employee (instance of 'Employee' class) is always assigned to a particular department (instance of 'Department' class) and the association can be shown by a line connecting the respective classes.

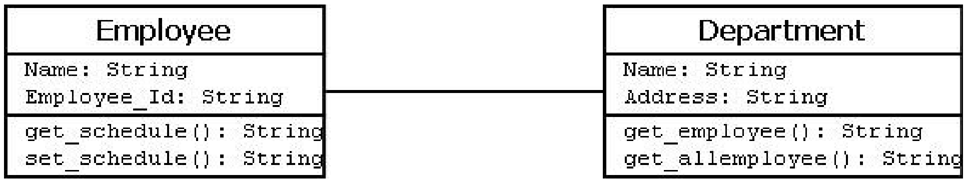


Figure -01:

- **Aggregation**

  It is a special form of association which describes a part-whole relationship between a pair of classes. It means, in a relationship, when a class holds some instances of related class, then that relationship can be designed as an aggregation.

  Example

  For a supermarket in a city, each branch runs some of the departments they have. So, the relation among the classes 'Branch' and 'Department' can be designed as an aggregation. In UML, it can be shown as in the Figure 02. below

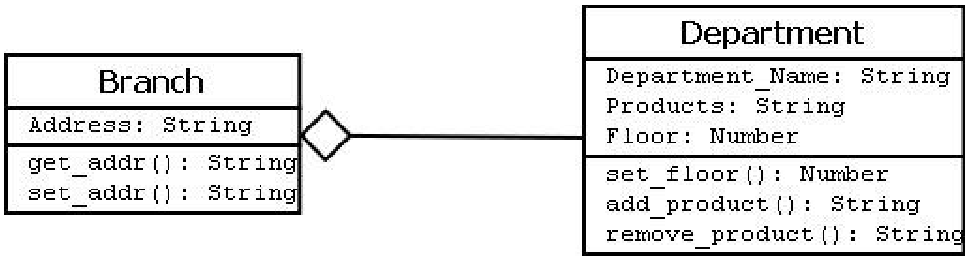


Figure-02:

- **Composition**

  It is a strong from of aggregation which describes that whole is completely owns its part. Life cycle of the part depends on the whole.

  Example

  Let consider a shopping mall has several branches in different locations in a city. The existence of branches completely depends on the shopping mall as if it is not exist any branch of it will no longer exists in the city. This relation can be described as composition and can be shown as below

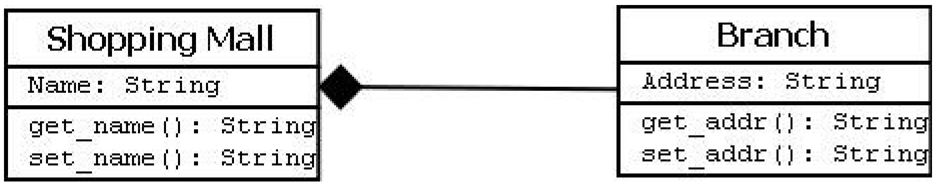
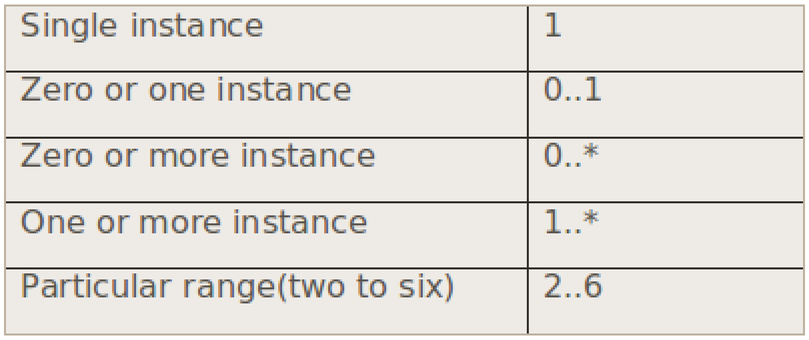


Figure-03:

- **Multiplicity**

  It describes how many numbers of instances of one class is related to the number of instances of another class in an association.

  Notation for different types of multiplicity:



Single instance	1
Zero or one instance	0..1
Zero or more instance	0..*
One or more instance	1..*
Particular range(two to six)	2..6

Figure-04:

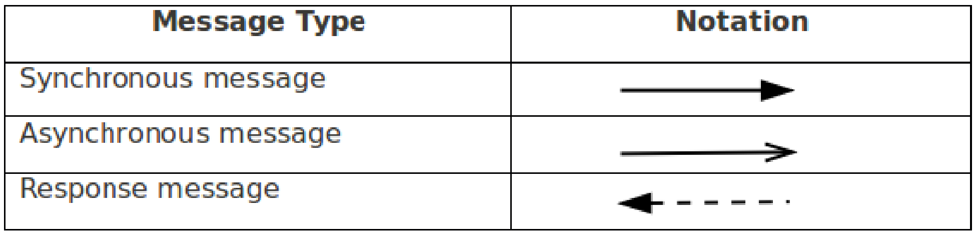
Some classes identified for BookBank application are Member, Useraccount, Book, Reservation, Loan, Admin, Transaction.

Sequence Diagram:

Messages

Messages are shown as an arrow from the life-line of sender object to the life-line of receiver object and labeled with the message name. Chronological order of the messages passing throughout the objects' life-line show the sequence in which they occur. There may exist some different types of messages :

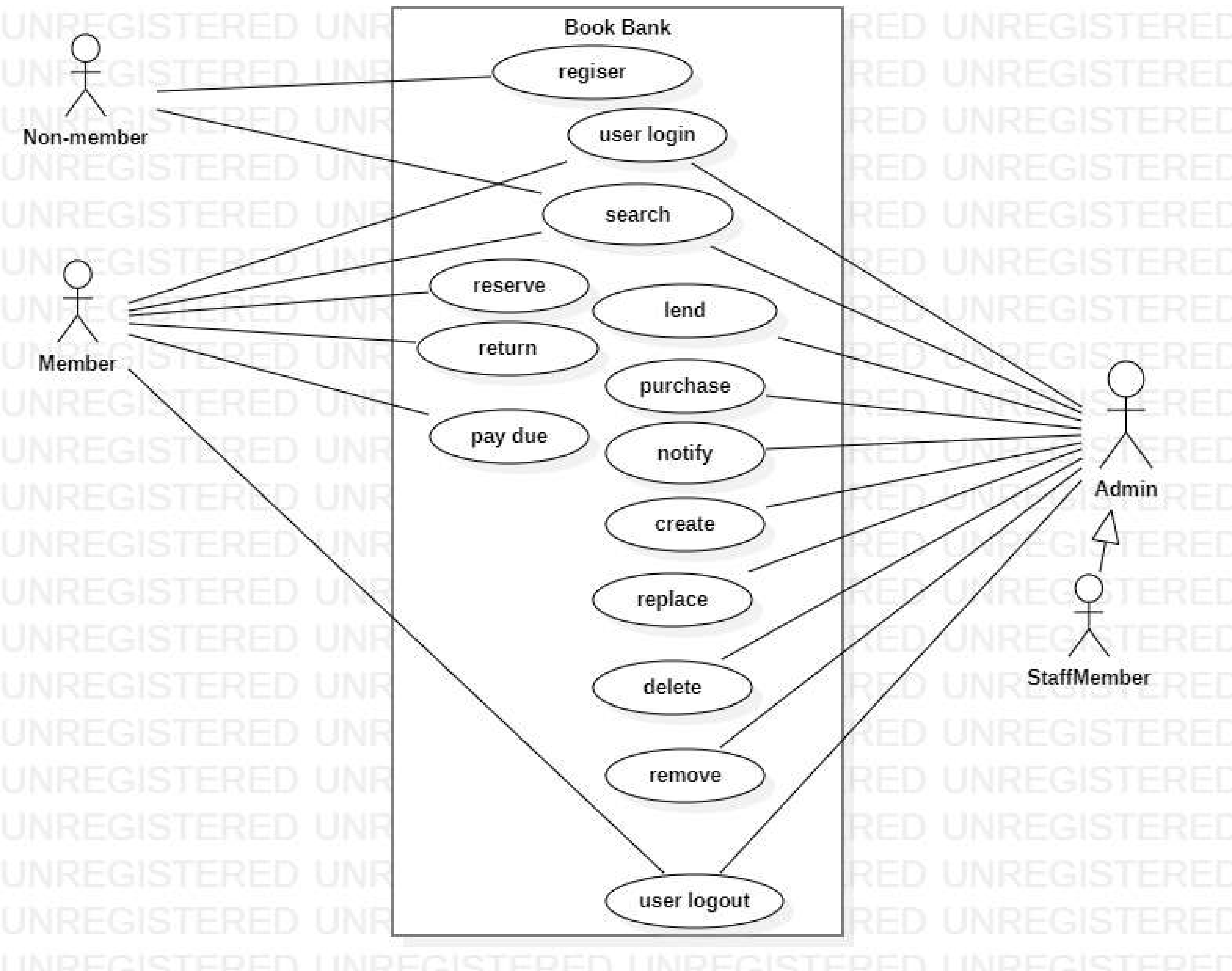
- **Synchronous messages:**Receiver start processing the message after receiving it and sender needs to wait until it is made. A straight arrow with close and fill arrow-head from sender life-line bar to receiver end, represent a synchronous message.
- **Asynchronous messages:**For asynchronous message sender needs not to wait for the receiver to process the message. A function call that creates thread can be represented as an asynchronous message in sequence diagram. A straight arrow with open arrow-head from sender life-line bar to receiver end, represent an asynchronous message.
- **Return message:**For a function call when we need to return a value to the object, from which it was called, then we use return message. But, it is optional, and we are using it when we are going to model our system in much detail. A dashed arrow with open arrow-head from sender life-line bar to receiver end, represent that message.
- **Response message:**One object can send a message to self. We use this message when we need to show the interaction between the same object.

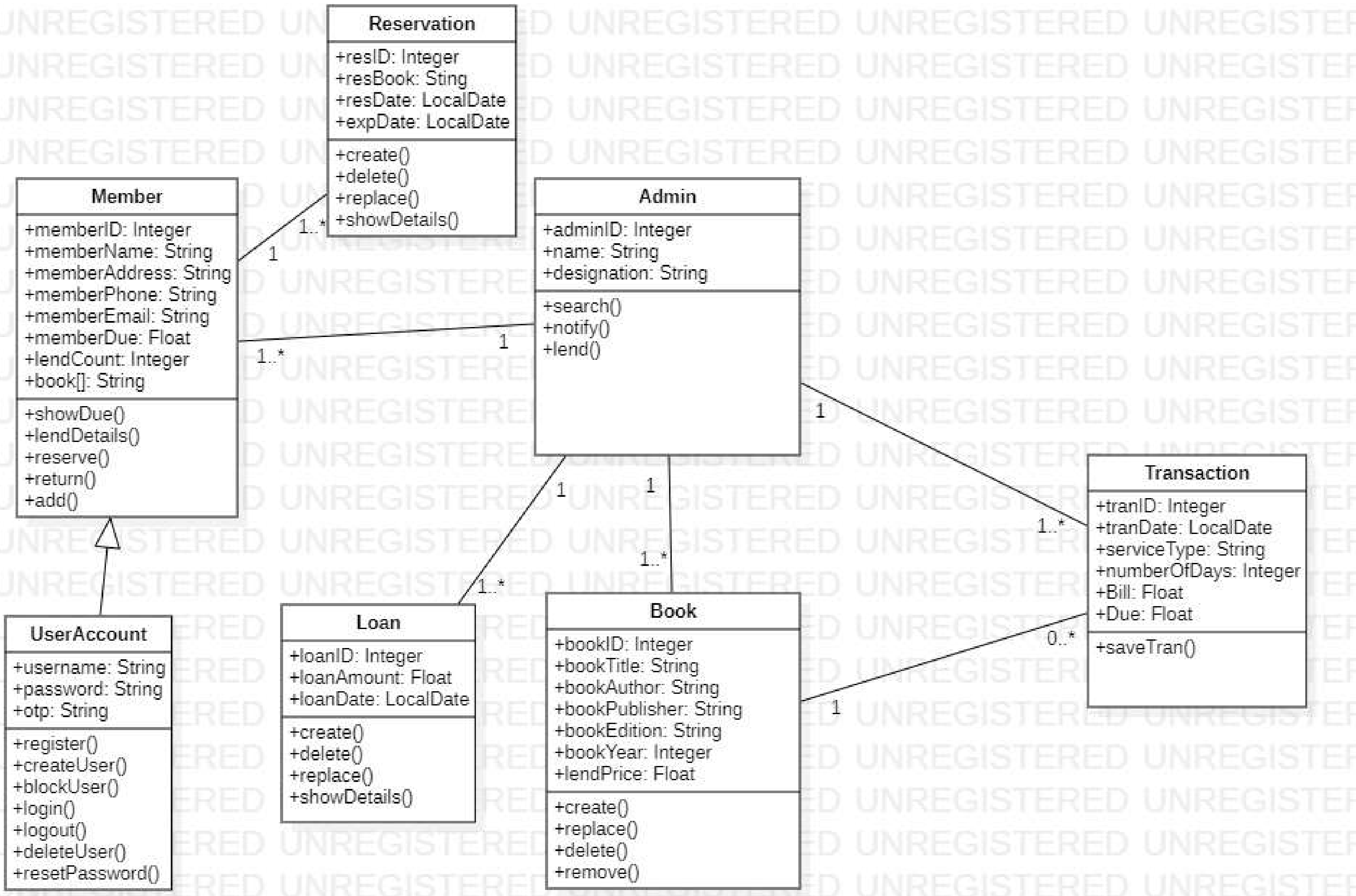


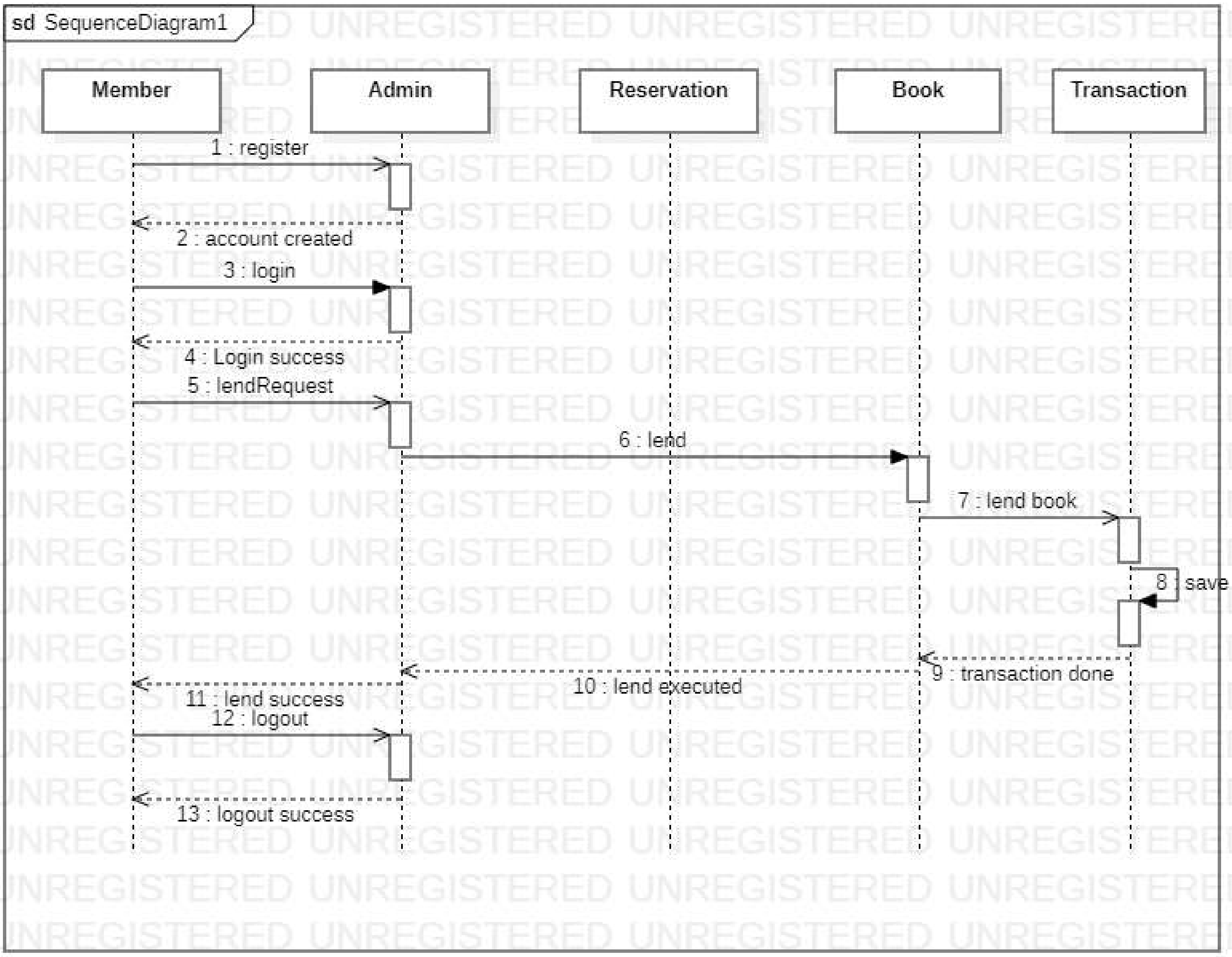
Message Type	Notation
Synchronous message	⟶
Asynchronous message	⟶
Response message	⇠

Figure-04:

(See: http://vlabs.iitkgp.ernet.in/se/)


Book Bank
regiser
user login
search
reserve
lend
return
purchase
pay due
notify
create
replace
delete
remove
user logout
Non-member
Member
Admin
StaffMember


Reservation
+resID: Integer
+resBook: Sting
+resDate: LocalDate
+expDate: LocalDate
+create()
+delete()
+replace()
+showDetails()
Member
+memberID: Integer
+memberName: String
+memberAddress: String
+memberPhone: String
+memberEmail: String
+memberDue: Float
+lendCount: Integer
+book[]: String
+showDue()
+lendDetails()
+reserve()
+return()
+add()
Admin
+adminID: Integer
+name: String
+designation: String
+search()
+notify()
+lend()
Transaction
+tranID: Integer
+tranDate: LocalDate
+serviceType: String
+numberOfDays: Integer
+Bill: Float
+Due: Float
+saveTran()
UserAccount
+username: String
+password: String
+otp: String
+register()
+createUser()
+blockUser()
+login()
+logout()
+deleteUser()
+resetPassword()
Loan
+loanID: Integer
+loanAmount: Float
+loanDate: LocalDate
+create()
+delete()
+replace()
+showDetails()
Book
+bookID: Integer
+bookTitle: String
+bookAuthor: String
+bookPublisher: String
+bookEdition: String
+bookYear: Integer
+lendPrice: Float
+create()
+replace()
+delete()
+remove()
1..*
1
1..*
1
1
1..*
1
1..*
1
1..*
0..*
1


sd SequenceDiagram1
Member
Admin
Reservation
Book
Transaction
1 : register
2 : account created
3 : login
4 : Login success
5 : lendRequest
6 : lend
7 : lend book
8 : save
9 : transaction done
10 : lend executed
11 : lend success
12 : logout
13 : logout success