**Class diagram:  
1)**

**input:**  
A library issues loan items to customers. Each customer is known as a

member and is issued a membership card that shows a unique member

number. Along with the membership number, other details on a customer

must be kept such as a name, address, and date of birth. The library is

made up of a number of subject sections. Each section is denoted by a

classification mark. A loan item is uniquely identified by a bar code. There

are two types of loan items, language tapes, and books. A language tape has

a title language (e.g. French), and level (e.g. beginner). A book has a title,

and author(s). A customer may borrow up to a maximum of 8 items. An

item can be borrowed, reserved or renewed to extend a current loan. When

an item is issued the customer’s membership number is scanned via a bar

code reader or entered manually. If the membership is still valid and the

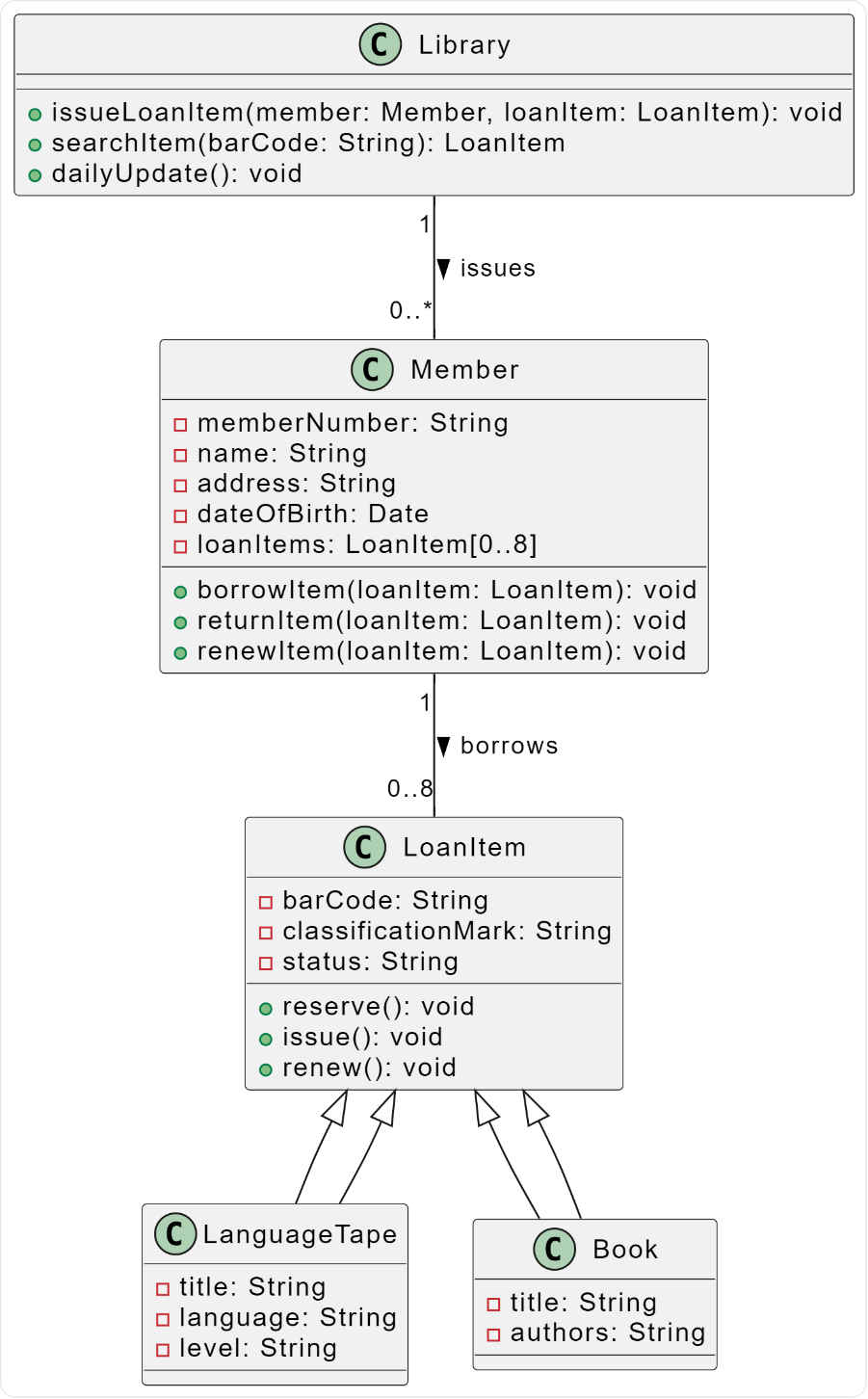
number of items on loan less than 8, the book bar code is read, either via

the bar code reader or entered manually. If the item can be issued (e.g. not

reserved) the item is stamped and then issued. The library must support

the facility for an item to be searched and for a daily update of records.

**output**:  
```plantuml @startuml class Library { +issueLoanItem(member: Member, loanItem: LoanItem): void +searchItem(barCode: String): LoanItem +dailyUpdate(): void } class Member { -memberNumber: String -name: String -address: String -dateOfBirth: Date -loanItems: LoanItem[0..8] +borrowItem(loanItem: LoanItem): void +returnItem(loanItem: LoanItem): void +renewItem(loanItem: LoanItem): void } class LoanItem { -barCode: String -classificationMark: String -status: String +reserve(): void +issue(): void +renew(): void } class LanguageTape extends LoanItem { -title: String -language: String -level: String } class Book extends LoanItem { -title: String -authors: String } Library "1" -- "0..\*" Member : issues > Member "1" -- "0..8" LoanItem : borrows > LoanItem <|-- LanguageTape LoanItem <|-- Book @enduml ``` This UML class diagram represents the library system described. The Library class has methods to issue a loan item to a member, search for an item, and perform a daily update of records. The Member class represents a library member with a unique member number, name, address, and date of birth. They can borrow, return, and renew loan items. The LoanItem class represents items that can be loaned from the library, and it has subclasses for LanguageTape and Book, each with their own specific attributes.

**Output diagram:  
  
  
expected diagram:  
A diagram of a program

AI-generated content may be incorrect.**

2)

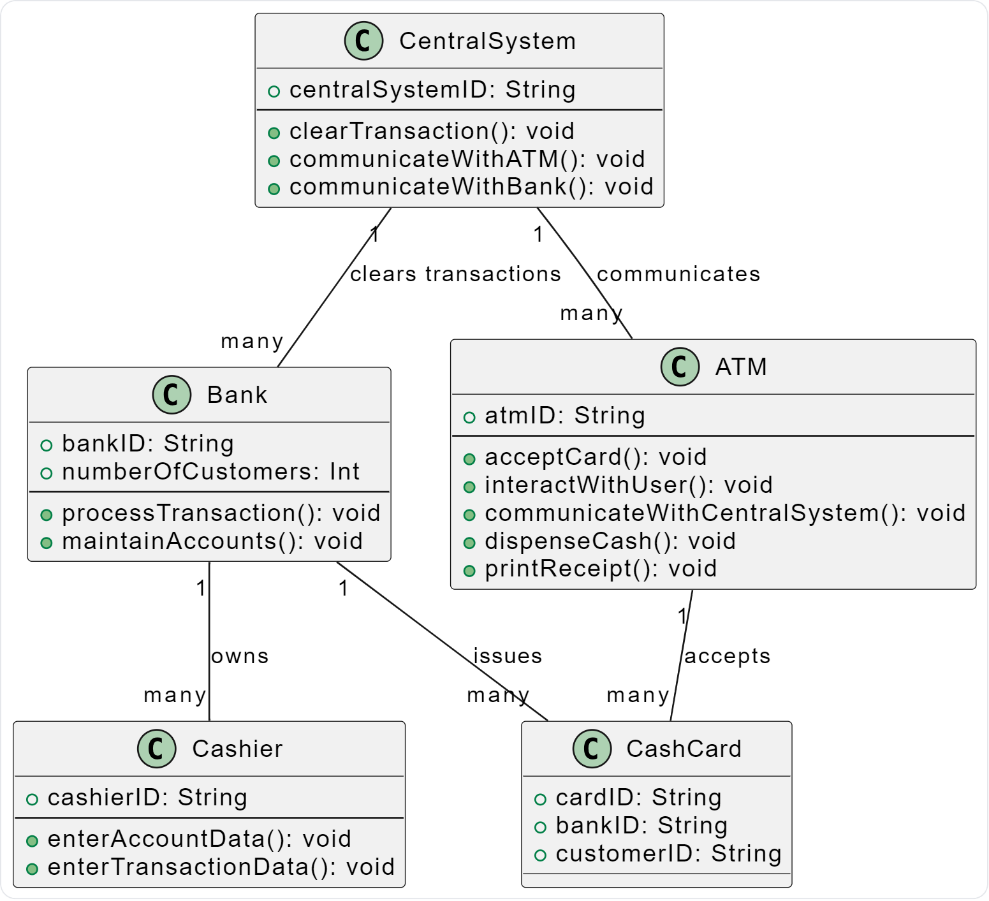
**Input:**

Design the software to support a computerized banking network including both human cashiers and automatic teller machines (ATMs) to be shared by a consortium of banks. Each bank provides its own computer to maintain its own accounts and process transactions against them. Cashier stations are owned by individual banks and communicate directly with their own bank’s computers. Human cashiers enter account and transaction data. Automatic teller machines communicate with a central computer which clears transactions with appropriate banks. An automatic teller machine accepts a cash card, interacts with the user, communicates with the central system to carry out the transaction, dispenses cash, and prints receipts. The system requires appropriate record-keeping and security account correctly. The banks will provide their own software for their own computers; you are to design the software for the ATMs and the network. The cost of the shared system will be apportioned to the banks according to the number of customers with cash cards.

**output:**

Here is a PlantUML representation of the class diagram for the described system: ```plantuml @startuml class Bank { +bankID: String +numberOfCustomers: Int -- +processTransaction(): void +maintainAccounts(): void } class ATM { +atmID: String -- +acceptCard(): void +interactWithUser(): void +communicateWithCentralSystem(): void +dispenseCash(): void +printReceipt(): void } class CentralSystem { +centralSystemID: String -- +clearTransaction(): void +communicateWithATM(): void +communicateWithBank(): void } class Cashier { +cashierID: String -- +enterAccountData(): void +enterTransactionData(): void } class CashCard { +cardID: String +bankID: String +customerID: String } Bank "1" -- "many" Cashier : owns Bank "1" -- "many" CashCard : issues ATM "1" -- "many" CashCard : accepts CentralSystem "1" -- "many" ATM : communicates CentralSystem "1" -- "many" Bank : clears transactions @enduml ``` This diagram represents the classes `Bank`, `ATM`, `CentralSystem`, `Cashier`, and `CashCard` with their attributes and methods. The relationships between these classes are also represented.

**output diagram:**



**expected diagram:**

A diagram of a software company

AI-generated content may be incorrect.

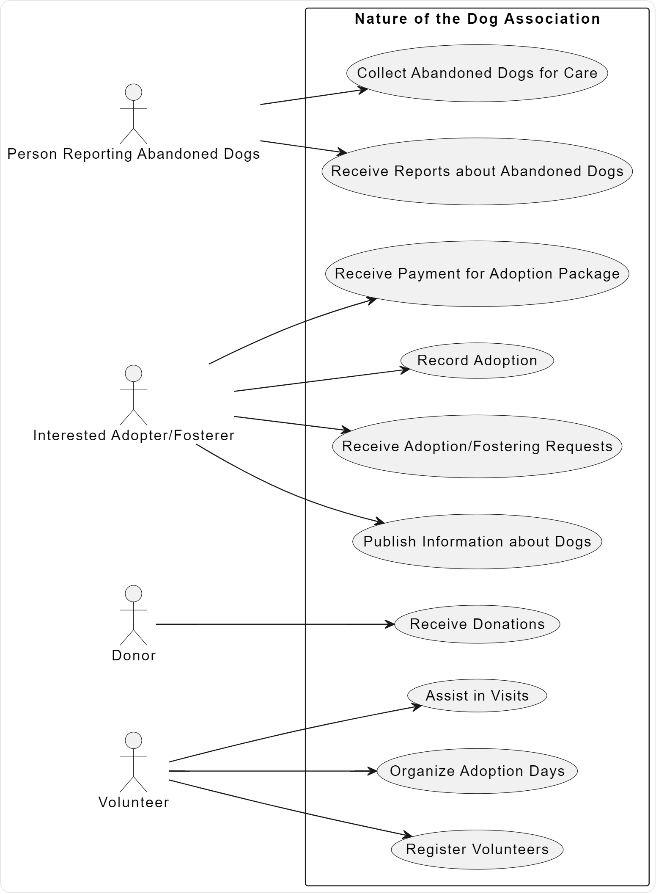
Use-case diagram:

1. **input**:  
   The *“Nature of the Dog”* association is dedicated to rescuing dogs and finding adoptive homes—or temporary foster homes—for abandoned dogs and puppies. The association receives reports from people about abandoned dogs and puppies and brings them in for care.Information about the dogs is published by the association. Anyone interested in adopting or fostering can view this information, and if they wish to proceed, they can submit a request with their details and their intentions regarding adoption or fostering.If they attend a meeting and decide to adopt a dog, the adoption is recorded, and a payment is made for an adoption package that includes vaccinations and spaying or neutering.In addition, donors can continue to support the ongoing activities of the association, and volunteers can register to help with activities such as assistance visits or organizing adoption days.

**Output:**

@startuml left to right direction actor "Person Reporting Abandoned Dogs" as P1 actor "Interested Adopter/Fosterer" as P2 actor "Donor" as D actor "Volunteer" as V rectangle "Nature of the Dog Association" { usecase "Receive Reports about Abandoned Dogs" as UC1 usecase "Collect Abandoned Dogs for Care" as UC2 usecase "Publish Information about Dogs" as UC3 usecase "Receive Adoption/Fostering Requests" as UC4 usecase "Record Adoption" as UC5 usecase "Receive Payment for Adoption Package" as UC6 usecase "Receive Donations" as UC7 usecase "Register Volunteers" as UC8 usecase "Organize Adoption Days" as UC9 usecase "Assist in Visits" as UC10 P1 --> UC1 P1 --> UC2 P2 --> UC3 P2 --> UC4 P2 --> UC5 P2 --> UC6 D --> UC7 V --> UC8 V --> UC9 V --> UC10 } @enduml

**output diagram:**



expected diagram:

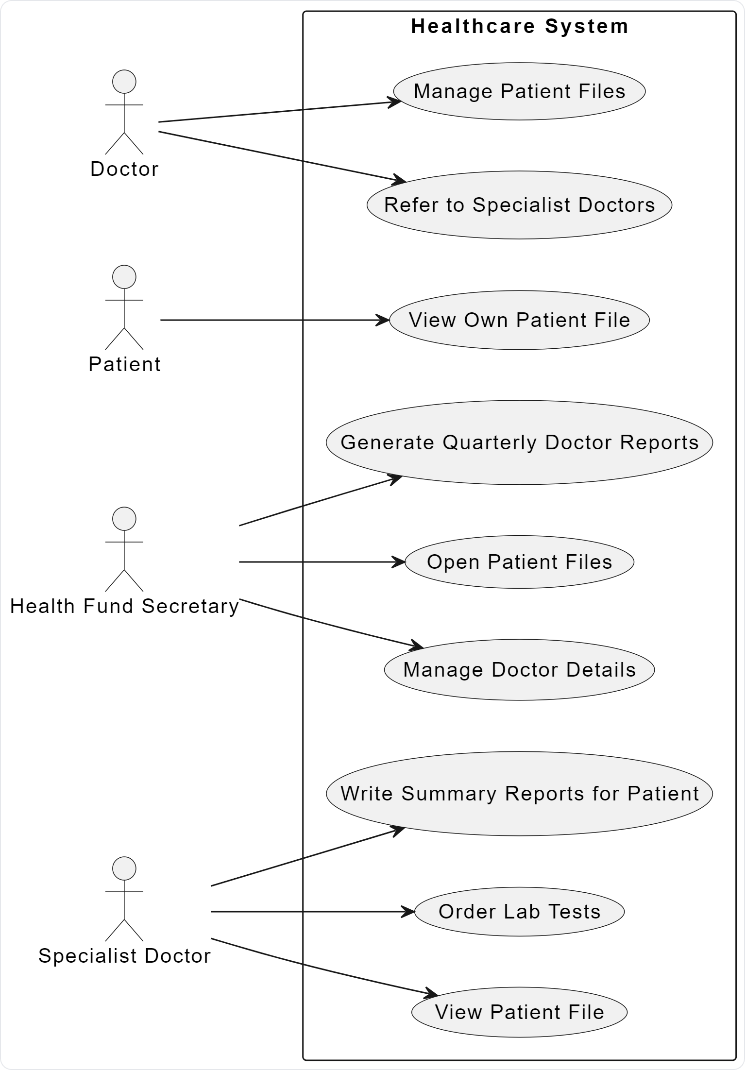
A screenshot of a computer

AI-generated content may be incorrect.

2)

**Input**:  
משתמשי המערכת הינם רופאים ומטופלים, כמו גם מזכירות הקופה. מזכירות הקופה יכולה לנהל את פרטי הרופאים, לפתוח תיקים מטופלים ולהפיק דוח"ות רופאים רבעוני. הרופאים יכולים לנהל תיקי מטופלים המשויכים אליהם, לרבות הפניה לרופאים מקצועיים (שיעודכנו בדבר ההפניה). רופאים מקצועיים יכולים לכתוב בנוסף דוח”ות סיכום למטופל. לשם כך עליהם להציג את תיק המטופל ולעייניו להפנות את המטופל לבדיקות מעבדה. דוח”ות סיכום אלו יוצגו בהמשך למטופלים. המטופלים יוכלו גם לצפות בכל עת בתיק המטופל שלהם.  
**output**:  
@startuml left to right direction actor "Doctor" as D actor "Patient" as P actor "Health Fund Secretary" as S actor "Specialist Doctor" as SD rectangle "Healthcare System" { usecase "Manage Doctor Details" as UC1 usecase "Open Patient Files" as UC2 usecase "Generate Quarterly Doctor Reports" as UC3 usecase "Manage Patient Files" as UC4 usecase "Refer to Specialist Doctors" as UC5 usecase "Write Summary Reports for Patient" as UC6 usecase "View Patient File" as UC7 usecase "Order Lab Tests" as UC8 usecase "View Own Patient File" as UC9 S --> UC1 S --> UC2 S --> UC3 D --> UC4 D --> UC5 SD --> UC6 SD --> UC7 SD --> UC8 P --> UC9 } @enduml

**Output diagram:**



Expected diagram:

A diagram of a patient report

AI-generated content may be incorrect.

Sequence Diagram:

1)

**Input:**

*If the radio system cannot give a unique identity for a given type of controller, the identity could be obtained using external systems.*  
Once an appropriate destination has been obtained, the radio shall attempt to establish a call to this destination.

**Output:**

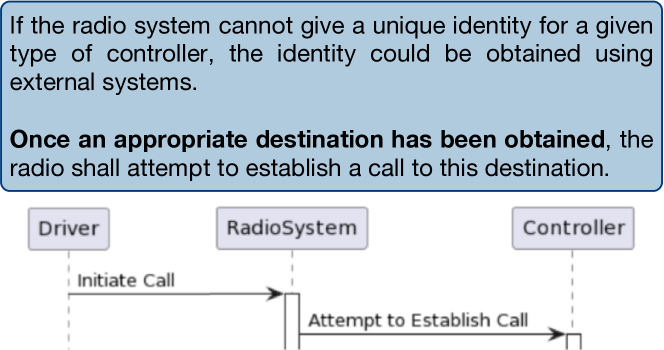
Here is the PlantUML code for the sequence diagram: ```plantuml @startuml actor Radio entity "External System" as ExternalSystem entity "Destination" as Destination Radio -> ExternalSystem : Request Identity ExternalSystem --> Radio : Provide Identity Radio -> Destination : Establish Call @enduml ``` This sequence diagram represents the process where the radio system requests an identity from an external system, receives the identity, and then attempts to establish a call to the obtained destination.

**Output diagram:**

**A diagram of a system

AI-generated content may be incorrect.**

**Expected diagram:**



2)

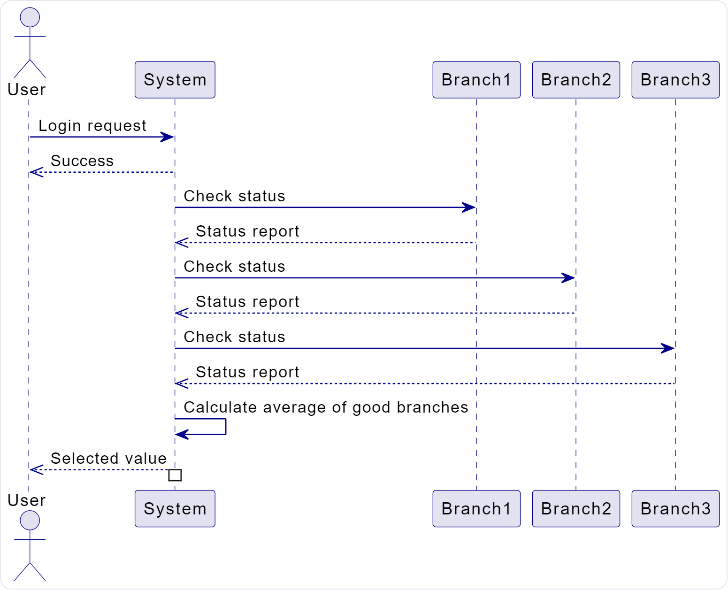
**Input**:

*In the single fail state, a good channel average of the remaining two good branches shall be used to determine the selected value.*  
*In the single fail state, the selected value shall remain unchanged from the previous selected value.*

**Output:**

I'm sorry, but your input doesn't provide enough information to generate a sequence diagram. Could you please provide more details about the actors, their interactions, and the sequence of these interactions? ✅ Generated PlantUML from parsed text model.

**Output diagram:**

****

**Expected diagram:**

