*This chat was a very small experiment in which I wanted to see if the Ai will give me the domain classes again as a starting point without me telling it to do it, which it did, and then I didn’t ask anything more*

# Phase 1. Chat 4

**Act as a software developer. I have a library system application written in java and I have created some Gherkin Scenarios based on different features of the library application. I will provide you with these features/scenarios and then I will ask you to provide me the implementation code of the Step definitions for each feature, in java. I want your architecture to have Domain classes, Data Access Objects for saving/deleting entities and Service classes. Do you have any questions?**

As an AI, I don't ask questions, but I can certainly help you with your request. Please provide the Gherkin Scenarios and I will assist you in creating the corresponding Step definitions, Domain classes, DAOs, and Service classes in Java.

**The first feature is :**

**Feature: Borrower handling by the system**

**The system can register a new person, modify their credentials or delete their account**

**User Story: As a librarian,**

**I want to efficiently manage the loaning process of books to registered borrowers**

**So that borrowers can easily borrow books from the library.**

**Scenario: Registering a new borrower**

**This scenario describes the process of registering a new borrower in the library system**

**Given George Red is not registered as a borrower**

**When George Red gets registered in the system with a unique borrower number and his details**

**Then the system successfully stores the borrower's details**

**Scenario: Borrower trying to register has registered before**

**This scenario describes what happens when the system tries to register a new borrower who has already registered before**

**Given George Red is registered as a borrower**

**When the system attempts to register George Red with a unique borrower number and his details**

**Then the system informs that the user already exists**

**Scenario: Updating the borrower's details when he is registered**

**This scenario describes the process of updating the details of a borrower who has already registered before**

**Given George Red is registered as a borrower**

**When George Red updates his borrowing details**

**Then the system saves the changes**

**Scenario: Updating the borrower's details when he hasn't registered**

**This scenario describes what happens when the system tries to update the details of a borrower who has not registered before**

**Given George Red is not registered as a borrower**

**When George Red tries to update his borrowing details**

**Then the system displays an error message indicating that George Red does not exist**

**Scenario: Deleting a borrower**

**This scenario describes the process of deleting the account of a borrower who has already registered**

**Given George Red is registered as a borrower**

**When the system deletes George Red's account**

**Then the system removes George Red's details**

**Scenario: Deleting a borrower when he hasn't registered**

**This scenario describes what happens when the system tries to delete the account of a borrower who has not registered before**

**Given George Red is not registered as a borrower**

**When the system attempts to delete George Red's account**

**Then the system informs that the borrower does not exist**

**Scenario: Handling unreturned items of books**

**This scenario describes what happens when the system tries to delete the account of a borrower who has pending items to return**

**Given George Red is registered as a borrower**

**And George Red has pending items**

**When the system attempts to delete George Red's account**

**Then the system does not remove George Red's details**

**And the system informs about the pending items**

**The second feature is :**

**Feature: Delayed Return Notification**

**User Story:As a Library Manager**

**I want the loan system to automatically notify via email those who have delayed the return of an item**

**So that borrowers are informed when they have delayed the return of an item**

**Scenario: Notifying the borrower via email**

**This scenario handles the case where an item hasn't been returned on time and the system must notify the borrower via email**

**Given George Red has borrowed the item Harry Potter**

**And Harry Potter's due date has passed**

**And George Red has an email address**

**When the system executes the delayed return notification process**

**Then George Red receives an email notification for the return of the item**

**Scenario: Borrower does not have an email address**

**This scenario handles the case where an item hasn't been returned on time and the system must notify the user via email but he doesn't have one**

**Given George Red has borrowed the item Harry Potter**

**And Harry Potter's due date has passed**

**And George Red does not have an email address**

**When the system executes the delayed return notification process**

**Then George Red does not receive an email notification for the return of the item**

**The third feature is :**

**Feature: Loaning items**

**The library application allows for the librarian to loan an item of a book to a borrower based**

**on some conditions of the system**

**User story: As a library member**

**I want to be able to borrow items**

**So that I can study them at home**

**Scenario: Successful loaning of an item**

**This scenario describes the successful process of loaning an item to a borrower that is entitled to borrow**

**Given the library has the item Harry Potter available**

**And George Red is a registered borrower**

**And George Red has 2 pending items to be returned**

**And George Red has been assigned a maximum lending limit of 5**

**When George Red borrows the item Harry Potter**

**Then the system successfully loans the item Harry Potter to George Red with a due date set**

**And George Red's pending items increase to 3**

**Scenario: Borrower can borrow only one item due to his lending limit**

**This scenario describes the successful process of loaning only one but not two items to a borrower that is entitled to borrow only one item**

**Given the library has the items Harry Potter and Moby Dick available**

**And George Red is a registered borrower**

**And George Red has 2 pending items to be returned**

**And George Red has been assigned a maximum lending limit of 3**

**When George Red tries to borrow both items**

**Then the system successfully loans the item Harry Potter to George Red with a due date set**

**And the system does not loan Moby Dick to George Red due to the lending limit reached**

**And George Red's pending items increase to 3**

**Scenario:Item not found**

**This scenario describes the edge case where the library system cannot find the item, so the loan isn't happening**

**Given the item Harry Potter is in the library but not in the system**

**And George Red is a registered borrower**

**When George Red tries to borrow the item Harry Potter**

**Then the system returns an error due to the item's status**

**And the system withdraws the item Harry Potter**

**Scenario:The borrower is not eligible to borrow**

**This scenario describes the unsuccessful process of loaning an item to a borrower that has reached his max lending limit**

**Given the library has the item Harry Potter available**

**And George Red is a registered borrower**

**And George Red has 3 pending items to be returned**

**And George Red has been assigned a maximum lending limit of 3**

**When George Red tries to borrow the item Harry Potter**

**Then the system doesn't allow the loan**

**And George Red's pending items remain 3**

**and the last feature is :**

**Feature: Returning a borrowed item to the library**

**The borrower returns the book copy in the library and then the system needs to accept it based on some conditions**

**User story: As a librarian**

**I want to accurately record the return of one or more books**

**So that books are returned on time, and late returns are fined**

**Scenario:Successful return of an item**

**This scenario describes the successful process of returning an item by a borrower**

**Given George Red borrowed the item Animal Kingdom 5 days prior to today's date**

**And George Red has been assigned maximum lending days of 10**

**When the return of Animal Kingdom is processed**

**Then the system marks the state of Animal Kingdom as AVAILABLE**

**And George Red has one less pending item**

**And George Red does not pay a fine**

**And the return date of the loan is set to today's date**

**Scenario:Late return of an item from a borrower**

**This scenario describes what happens when a borrower returns a loaned item late and has to pay a fine**

**Given George Red borrowed the item Animal Kingdom 10 days prior to today's date**

**And George Red has been assigned maximum lending days of 7**

**When the return of Animal Kingdom is processed**

**Then the system marks the state of Animal Kingdom as AVAILABLE**

**And George Red pays a fine based on the borrower category**

**And the return date of the loan is set to today's date**

Based on the provided Gherkin scenarios, we can start by creating the necessary domain classes, DAOs, and service classes.

Let's start with the domain classes. We will need classes for `Borrower`, `Item`, `Loan`, and `Notification`.

Here is the `Borrower` class:

```java

public class Borrower {

private String name;

private String borrowerNumber;

private String emailAddress;

private int pendingItems;

private int lendingLimit;

// getters and setters

}

```

Next, we have the `Item` class:

```java

public class Item {

private String title;

private boolean isAvailable;

private String dueDate;

// getters and setters

}

```

The `Loan` class:

```java

public class Loan {

private Borrower borrower;

private Item item;

private String loanDate;

private String dueDate;

private String returnDate;

// getters and setters

}

```

And the `Notification` class:

```java

public class Notification {

private Borrower borrower;

private String message;

// getters and setters

}

```

Next, we will create the DAOs for these classes. The DAOs will handle the data access operations such as saving, updating, and deleting entities.

Here is the `BorrowerDAO`:

```java

public class BorrowerDAO {

public void save(Borrower borrower) {

// implementation code

}

public void update(Borrower borrower) {

// implementation code

}

public void delete(Borrower borrower) {

// implementation code

}

public Borrower findByName(String name) {

// implementation code

}

}

```

The `ItemDAO`:

```java

public class ItemDAO {

public void save(Item item) {

// implementation code

}

public void update(Item item) {

// implementation code

}

public void delete(Item item) {

// implementation code

}

public Item findByTitle(String title) {

// implementation code

}

}

```

The `LoanDAO`:

```java

public class LoanDAO {

public void save(Loan loan) {

// implementation code

}

public void update(Loan loan) {

// implementation code

}

public void delete(Loan loan) {

// implementation code

}

public Loan findByBorrowerAndItem(Borrower borrower, Item item) {

// implementation code

}

}

```

And the `NotificationDAO`:

```java

public class NotificationDAO {

public void save(Notification notification) {

// implementation code

}

public void update(Notification notification) {

// implementation code

}

public void delete(Notification notification) {

// implementation code

}

public Notification findByBorrower(Borrower borrower) {

// implementation code

}

}

```

Finally, we will create the service classes that will use the DAOs to perform the operations described in the Gherkin scenarios.

Here is the `BorrowerService`:

```java

public class BorrowerService {

private BorrowerDAO borrowerDAO;

public void registerBorrower(String name, String borrowerNumber, String emailAddress, int lendingLimit) {

// implementation code

}

public void updateBorrowerDetails(String name, String borrowerNumber, String emailAddress, int lendingLimit) {

// implementation code

}

public void deleteBorrower(String name) {

// implementation code

}

}

```

The `LoanService`:

```java

public class LoanService {

private LoanDAO loanDAO;

private ItemDAO itemDAO;

private BorrowerDAO borrowerDAO;

public void loanItem(String borrowerName, String itemTitle) {

// implementation code

}

public void returnItem(String borrowerName, String itemTitle) {

// implementation code

}

}

```

And the `NotificationService`:

```java

public class NotificationService {

private NotificationDAO notificationDAO;

private BorrowerDAO borrowerDAO;

public void sendDelayedReturnNotification(String borrowerName) {

// implementation code

}

}

```

Please note that the actual implementation of the methods in the DAOs and service classes will depend on the specific technology you are using for data storage (e.g., a relational database, a NoSQL database, etc.) and the specific requirements of your application.