*In this chat, I provided the AI the same exact information as in the previous chats, but I DIDN’T instruct it to create the domain/ daos/ services code first, to see if it would make any difference, which it didn’t at all, it created almost the same code for the first feature, so I ended the conversation because there werent any improvements.*

# Phase 4. Chat 3

**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 the code of all the domain classes with their attributes and methods that you need, 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?**

I'm sorry, but as an AI, I don't ask questions. However, I'm ready to assist you. Please provide the Gherkin Scenarios and the code of all the domain classes with their attributes and methods. I'll then provide you with the implementation code of the Step definitions for each feature, in Java, adhering to the architecture you specified.

**The domain classes are:**

**1)class "Book" with the attributes:**

**private ISBN isbn,**

**private String title,**

**private String publication,**

**private int publicationyear,**

**private Publisher publisher,**

**private Set<Item> items = new HashSet<Item>(),**

**private Set<Author> authors = new HashSet<Author>(),**

**and with the methods:**

**Setters and Getters for every attribute and a constructor,**

**public Set<Item> getItems(),**

**public void addItem(Item item),**

**public void removeItem(Item item),**

**Set<Item> friendItems(),**

**public void addAuthor(Author author),**

**public void removeAuthor(Author author),**

**Set<Author> friendAuthors()**

**2)class "Borrower" with the attributes:**

**private int borrowerNo,**

**private TelephoneNumber telephone,**

**private EmailAddress emailAddress,**

**private Address address,**

**private BorrowerCategory category,**

**private Person person = new Person(),**

**private Set<Loan> loans = new HashSet<Loan>(),**

**and with the methods:**

**Setters and Getters for every attribute and a constructor,**

**public Set<Loan> getLoans(),**

**Set<Loan> friendLoans(),**

**public int countPendingItems(),**

**public boolean canBorrow(),**

**public LocalDate getLoanDue(LocalDate loanDate),**

**public Money getDailyFine()**

**3)class "BorrowerCategory" with the attributes:**

**private String description,**

**private int maxLendingItems,**

**private int maxLendingDays,**

**private Money dailyFine,**

**and with the methods:**

**Setters and Getters for every attribute and a constructor,**

**public LocalDate getLoanDue(LocalDate loanDate),**

**public boolean canBorrow(int pendingItems)**

**4)class "Item" with the attributes:**

**private int itemNumber = 0,**

**private Book book,**

**private ItemState state = ItemState.NEW**

**and with the methods:**

**Setters and Getters for every attribute and a constructor,**

**public Loan borrow(Borrower borrower),**

**public void available(),**

**public void withdraw(),**

**public void lost(),**

**5)enum class "ItemState" with the Enum values :**

**NEW,AVAILABLE,LOANED,LOST,WITHDRAWN**

**6)class "Loan" with the attributes:**

**private LocalDate loanDate = SystemDate.now(),**

**private LocalDate returnDate,**

**private Borrower borrower,**

**private Item item**

**and with the methods:**

**Setters and Getters for every attribute and a constructor,**

**public LocalDate getDue(),**

**public boolean isPending(),**

**public void returnItem(),**

**public boolean isOverdue(),**

**public long daysToDue(),**

**public long getOverdue(),**

**public Money getFine(),**

**7)class "Person" with the attributes:**

**private String firstName,**

**private String lastName,**

**and with the methods:**

**Setters and Getters for every attribute and a constructor**

**In the next message I will provide you with the features/scenarios**

Thank you for providing the domain classes. I'm ready to assist you with the implementation of the Step definitions once you provide the Gherkin Scenarios.

**The Gherkin scenarios are :**

**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 create the step definitions for each feature. Here are the step definitions for the first feature "Borrower handling by the system":

import io.cucumber.java.en.Given;

import io.cucumber.java.en.Then;

import io.cucumber.java.en.When;

public class BorrowerHandlingSteps {

private Borrower borrower;

private BorrowerService borrowerService = new BorrowerService();

@Given("George Red is not registered as a borrower")

public void georgeRedIsNotRegisteredAsABorrower() {

borrower = borrowerService.findBorrowerByName("George Red");

if (borrower != null) {

borrowerService.deleteBorrower(borrower);

}

}

@When("George Red gets registered in the system with a unique borrower number and his details")

public void georgeRedGetsRegisteredInTheSystemWithAUniqueBorrowerNumberAndHisDetails() {

borrower = new Borrower();

borrower.setBorrowerNo(borrowerService.generateUniqueBorrowerNumber());

borrower.setPerson(new Person("George", "Red"));

borrowerService.saveBorrower(borrower);

}

@Then("the system successfully stores the borrower's details")

public void theSystemSuccessfullyStoresTheBorrowersDetails() {

Borrower savedBorrower = borrowerService.findBorrowerByName("George Red");

assert savedBorrower != null;

}

// Continue with the rest of the scenarios in a similar manner

}

Please note that the `BorrowerService` class is assumed to be a service class that handles the operations related to the `Borrower` class. It should contain methods like `findBorrowerByName(String name)`, `deleteBorrower(Borrower borrower)`, `saveBorrower(Borrower borrower)`, and `generateUniqueBorrowerNumber()`.

You would need to create similar step definition classes for the other features as well. The step definitions should be written in a way that they interact with the domain classes and their corresponding service classes to perform the required operations.

**give me all the step definitions implementantion code for the feature : Borrower handling by the system, and give me the code for the borrower service you are using**

import java.util.List;

public class BorrowerService {

private BorrowerDAO borrowerDAO = new BorrowerDAO();

public Borrower findBorrowerByName(String name) {

List<Borrower> borrowers = borrowerDAO.getAllBorrowers();

for (Borrower borrower : borrowers) {

if (borrower.getPerson().getFirstName().equals(name)) {

return borrower;

}

}

return null;

}

public void deleteBorrower(Borrower borrower) {

borrowerDAO.delete(borrower);

}

public void saveBorrower(Borrower borrower) {

borrowerDAO.save(borrower);

}

public int generateUniqueBorrowerNumber() {

return borrowerDAO.getAllBorrowers().size() + 1;

}

}

**give me the rest of the scenarios as well**

Sure, let's continue with the rest of the scenarios for the feature "Borrower handling by the system":

@Given("George Red is registered as a borrower")

public void georgeRedIsRegisteredAsABorrower() {

borrower = borrowerService.findBorrowerByName("George Red");

if (borrower == null) {

borrower = new Borrower();

borrower.setBorrowerNo(borrowerService.generateUniqueBorrowerNumber());

borrower.setPerson(new Person("George", "Red"));

borrowerService.saveBorrower(borrower);

}

}

@When("the system attempts to register George Red with a unique borrower number and his details")

public void theSystemAttemptsToRegisterGeorgeRedWithAUniqueBorrowerNumberAndHisDetails() {

Borrower existingBorrower = borrowerService.findBorrowerByName("George Red");

if (existingBorrower != null) {

borrower = null;

} else {

borrower = new Borrower();

borrower.setBorrowerNo(borrowerService.generateUniqueBorrowerNumber());

borrower.setPerson(new Person("George", "Red"));

borrowerService.saveBorrower(borrower);

}

}

@Then("the system informs that the user already exists")

public void theSystemInformsThatTheUserAlreadyExists() {

assert borrower == null;

}

@When("George Red updates his borrowing details")

public void georgeRedUpdatesHisBorrowingDetails() {

if (borrower != null) {

borrower.setPerson(new Person("George", "Updated Red"));

borrowerService.saveBorrower(borrower);

}

}

@Then("the system saves the changes")

public void theSystemSavesTheChanges() {

Borrower updatedBorrower = borrowerService.findBorrowerByName("George Updated Red");

assert updatedBorrower != null;

}

@When("George Red tries to update his borrowing details")

public void georgeRedTriesToUpdateHisBorrowingDetails() {

if (borrower != null) {

borrower.setPerson(new Person("George", "Updated Red"));

borrowerService.saveBorrower(borrower);

}

}

@Then("the system displays an error message indicating that George Red does not exist")

public void theSystemDisplaysAnErrorMessageIndicatingThatGeorgeRedDoesNotExist() {

assert borrower == null;

}

@When("the system deletes George Red's account")

public void theSystemDeletesGeorgeRedsAccount() {

if (borrower != null) {

borrowerService.deleteBorrower(borrower);

borrower = null;

}

}

@Then("the system removes George Red's details")

public void theSystemRemovesGeorgeRedsDetails() {

Borrower deletedBorrower = borrowerService.findBorrowerByName("George Red");

assert deletedBorrower == null;

}

@When("the system attempts to delete George Red's account")

public void theSystemAttemptsToDeleteGeorgeRedsAccount() {

if (borrower != null) {

borrowerService.deleteBorrower(borrower);

borrower = null;

}

}

@Then("the system informs that the borrower does not exist")

public void theSystemInformsThatTheBorrowerDoesNotExist() {

Borrower deletedBorrower = borrowerService.findBorrowerByName("George Red");

assert deletedBorrower == null;

}

@Given("George Red has pending items")

public void georgeRedHasPendingItems() {

if (borrower != null) {

Loan loan = new Loan();

loan.setBorrower(borrower);

borrower.getLoans().add(loan);

}

}

@Then("the system does not remove George Red's details")

public void theSystemDoesNotRemoveGeorgeRedsDetails() {

Borrower existingBorrower = borrowerService.findBorrowerByName("George Red");

assert existingBorrower != null;

}

@Then("the system informs about the pending items")

public void theSystemInformsAboutThePendingItems() {

assert borrower.getLoans().size() > 0;

}

These step definitions cover all the scenarios for the feature "Borrower handling by the system". They interact with the `Borrower` domain class and the `BorrowerService` class to perform the required operations.