*In this chat, I provided the Ai copilot with the exact same information as in chat 2,3 BUT I instructed it to create the domain code first(in the others chats I didn’t tell something like that). The Ai really understood pretty well how and what domain classes to create ,BUT, it really didn’t use DAOs or Services and used a class named LibrarySystem for all of these(and this class didn’t use DAOs,overall the results were good but it used this class librarySystem so it wasn’t so good, and in the next one I will instruct the Ai to create the code for the DAOs and Services from the start, to then see if it makes it use them more, as I did with gpt 3.5 and it helped)*

# Phase 3. 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 name of all the domain classes with their attributes , 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'm here to assist you. Please provide the Gherkin Scenarios and the details of the domain classes. I'll do my best to help you with the implementation of the Step definitions for each feature in Java.

**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>()**

**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>()**

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

**private String description,**

**private int maxLendingItems,**

**private int maxLendingDays,**

**private Money dailyFine**

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

**private int itemNumber = 0,**

**private Book book,**

**private ItemState state = ItemState.NEW**

**5)enum class "ItemState" with no attrbutes outside the enum values**

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

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

**private LocalDate returnDate,**

**private Borrower borrower,**

**private Item item**

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

**private String firstName,**

**private String lastName**

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

Thank you for providing the domain classes. Here are the Java implementations for each class:

public class Book {

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>();

// getters and setters

}

public class Borrower {

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>();

// getters and setters

}

public class BorrowerCategory {

private String description;

private int maxLendingItems;

private int maxLendingDays;

private Money dailyFine;

// getters and setters

}

public class Item {

private int itemNumber = 0;

private Book book;

private ItemState state = ItemState.NEW;

// getters and setters

}

public enum ItemState {

// enum values

}

public class Loan {

private LocalDate loanDate = SystemDate.now();

private LocalDate returnDate;

private Borrower borrower;

private Item item;

// getters and setters

}

public class Person {

private String firstName;

private String lastName;

// getters and setters

}

Please provide the Gherkin Scenarios so I can assist you with the implementation of the Step definitions for each feature in Java.

**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**

**I want you to firstly create all the domain classes and the implementation code for them**

Sure, let's start by creating the domain classes.

Here is the `Book` class:

public class Book {

private ISBN isbn;

private String title;

private String publication;

private int publicationyear;

private Publisher publisher;

private Set<Item> items = new HashSet<>();

private Set<Author> authors = new HashSet<>();

// getters and setters

}

Here is the `Borrower` class:

public class Borrower {

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<>();

// getters and setters

}

Here is the `BorrowerCategory` class:

public class BorrowerCategory {

private String description;

private int maxLendingItems;

private int maxLendingDays;

private Money dailyFine;

// getters and setters

}

Here is the `Item` class:

public class Item {

private int itemNumber = 0;

private Book book;

private ItemState state = ItemState.NEW;

// getters and setters

}

Here is the `ItemState` enum:

public enum ItemState {

NEW, AVAILABLE, BORROWED

}

Here is the `Loan` class:

public class Loan {

private LocalDate loanDate = LocalDate.now();

private LocalDate returnDate;

private Borrower borrower;

private Item item;

// getters and setters

}

And finally, here is the `Person` class:

public class Person {

private String firstName;

private String lastName;

// getters and setters

}

Please note that I've made some assumptions about the missing classes (`ISBN`, `Publisher`, `Author`, `TelephoneNumber`, `EmailAddress`, `Address`, `Money`). You'll need to replace these with your actual classes or create them if they don't exist.

**create these as well**

Sure, let's create the missing classes.

Here is the `ISBN` class:

public class ISBN {

private String isbn;

// getters and setters

}

Here is the `Publisher` class:

public class Publisher {

private String name;

// getters and setters

}

Here is the `Author` class:

public class Author {

private String name;

// getters and setters

}

Here is the `TelephoneNumber` class:

public class TelephoneNumber {

private String number;

// getters and setters

}

Here is the `EmailAddress` class:

public class EmailAddress {

private String email;

// getters and setters

}

Here is the `Address` class:

public class Address {

private String street;

private String city;

private String state;

private String zip;

// getters and setters

}

And finally, here is the `Money` class:

public class Money {

private BigDecimal amount;

// getters and setters

}

Please note that these are basic implementations and you may need to add more fields or methods based on your specific requirements.

**Based on the above, create the step definitions for the feature: Borrower Handling by the system**

Sure, let's create the step definitions for the feature: Borrower Handling by the system.

Here is the `StepDefinitions` class:

import io.cucumber.java.en.Given;

import io.cucumber.java.en.Then;

import io.cucumber.java.en.When;

public class StepDefinitions {

private Borrower borrower;

private LibrarySystem librarySystem = new LibrarySystem();

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

public void georgeRedIsNotRegisteredAsABorrower() {

borrower = librarySystem.findBorrower("George Red");

if (borrower != null) {

librarySystem.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(librarySystem.generateUniqueBorrowerNumber());

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

librarySystem.registerBorrower(borrower);

}

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

public void theSystemSuccessfullyStoresTheBorrowersDetails() {

Borrower registeredBorrower = librarySystem.findBorrower("George Red");

assert registeredBorrower != null;

}

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

public void georgeRedIsRegisteredAsABorrower() {

borrower = librarySystem.findBorrower("George Red");

if (borrower == null) {

borrower = new Borrower();

borrower.setBorrowerNo(librarySystem.generateUniqueBorrowerNumber());

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

librarySystem.registerBorrower(borrower);

}

}

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

public void theSystemAttemptsToRegisterGeorgeRedWithAUniqueBorrowerNumberAndHisDetails() {

Borrower newBorrower = new Borrower();

newBorrower.setBorrowerNo(librarySystem.generateUniqueBorrowerNumber());

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

librarySystem.registerBorrower(newBorrower);

}

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

public void theSystemInformsThatTheUserAlreadyExists() {

Borrower registeredBorrower = librarySystem.findBorrower("George Red");

assert registeredBorrower != null;

}

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

public void georgeRedUpdatesHisBorrowingDetails() {

borrower.setTelephoneNumber(new TelephoneNumber("1234567890"));

librarySystem.updateBorrower(borrower);

}

@Then("the system saves the changes")

public void theSystemSavesTheChanges() {

Borrower updatedBorrower = librarySystem.findBorrower("George Red");

assert updatedBorrower.getTelephoneNumber().getNumber().equals("1234567890");

}

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

public void georgeRedTriesToUpdateHisBorrowingDetails() {

borrower = new Borrower();

borrower.setBorrowerNo(librarySystem.generateUniqueBorrowerNumber());

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

borrower.setTelephoneNumber(new TelephoneNumber("1234567890"));

librarySystem.updateBorrower(borrower);

}

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

public void theSystemDisplaysAnErrorMessageIndicatingThatGeorgeRedDoesNotExist() {

Borrower updatedBorrower = librarySystem.findBorrower("George Red");

assert updatedBorrower == null;

}

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

public void theSystemDeletesGeorgeRedsAccount() {

librarySystem.deleteBorrower(borrower);

}

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

public void theSystemRemovesGeorgeRedsDetails() {

Borrower deletedBorrower = librarySystem.findBorrower("George Red");

assert deletedBorrower == null;

}

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

public void theSystemAttemptsToDeleteGeorgeRedsAccount() {

librarySystem.deleteBorrower(borrower);

}

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

public void theSystemInformsThatTheBorrowerDoesNotExist() {

Borrower deletedBorrower = librarySystem.findBorrower("George Red");

assert deletedBorrower == null;

}

@Given("George Red has pending items")

public void georgeRedHasPendingItems() {

Item item = new Item();

item.setBook(new Book());

item.setState(ItemState.BORROWED);

borrower.getLoans().add(new Loan(item, borrower));

}

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

public void theSystemDoesNotRemoveGeorgeRedsDetails() {

Borrower existingBorrower = librarySystem.findBorrower("George Red");

assert existingBorrower != null;

}

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

public void theSystemInformsAboutThePendingItems() {

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

}

}

Please note that I've made some assumptions about the `LibrarySystem` class. You'll need to replace these with your actual methods or create them if they don't exist.

**What is the implementation code for the library system?**

The `LibrarySystem` class would be responsible for managing the borrowers and their loans. Here's a basic implementation:

import java.util.HashMap;

import java.util.Map;

public class LibrarySystem {

private Map<String, Borrower> borrowers = new HashMap<>();

private int uniqueBorrowerNumber = 0;

public Borrower findBorrower(String name) {

return borrowers.get(name);

}

public void registerBorrower(Borrower borrower) {

if (borrowers.containsKey(borrower.getPerson().getFirstName() + " " + borrower.getPerson().getLastName())) {

throw new RuntimeException("Borrower already exists");

}

borrowers.put(borrower.getPerson().getFirstName() + " " + borrower.getPerson().getLastName(), borrower);

}

public void updateBorrower(Borrower borrower) {

if (!borrowers.containsKey(borrower.getPerson().getFirstName() + " " + borrower.getPerson().getLastName())) {

throw new RuntimeException("Borrower does not exist");

}

borrowers.put(borrower.getPerson().getFirstName() + " " + borrower.getPerson().getLastName(), borrower);

}

public void deleteBorrower(Borrower borrower) {

if (!borrowers.containsKey(borrower.getPerson().getFirstName() + " " + borrower.getPerson().getLastName())) {

throw new RuntimeException("Borrower does not exist");

}

if (!borrower.getLoans().isEmpty()) {

throw new RuntimeException("Borrower has pending items");

}

borrowers.remove(borrower.getPerson().getFirstName() + " " + borrower.getPerson().getLastName());

}

public int generateUniqueBorrowerNumber() {

return ++uniqueBorrowerNumber;

}

}

This class maintains a map of borrowers, keyed by their name. It provides methods to find, register, update, and delete borrowers. It also generates unique borrower numbers.

Please note that this is a basic implementation and you may need to add more methods or modify existing ones based on your specific requirements. For example, you might want to add methods to handle loans and items.

**now, give me the step definitions for the feature: Item Loaning**

Sure, let's create the step definitions for the feature: Item Loaning.

Here is the `StepDefinitions` class:

import io.cucumber.java.en.Given;

import io.cucumber.java.en.Then;

import io.cucumber.java.en.When;

public class StepDefinitions {

private Borrower borrower;

private Item item;

private LibrarySystem librarySystem = new LibrarySystem();

@Given("the library has the item Harry Potter available")

public void theLibraryHasTheItemHarryPotterAvailable() {

item = new Item();

item.setBook(new Book());

item.setState(ItemState.AVAILABLE);

librarySystem.addItem(item);

}

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

public void georgeRedIsARegisteredBorrower() {

borrower = librarySystem.findBorrower("George Red");

if (borrower == null) {

borrower = new Borrower();

borrower.setBorrowerNo(librarySystem.generateUniqueBorrowerNumber());

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

librarySystem.registerBorrower(borrower);

}

}

@Given("George Red has {int} pending items to be returned")

public void georgeRedHasPendingItemsToBeReturned(int pendingItems) {

for (int i = 0; i < pendingItems; i++) {

Item pendingItem = new Item();

pendingItem.setBook(new Book());

pendingItem.setState(ItemState.BORROWED);

borrower.getLoans().add(new Loan(pendingItem, borrower));

}

}

@Given("George Red has been assigned a maximum lending limit of {int}")

public void georgeRedHasBeenAssignedAMaximumLendingLimitOf(int maxLendingLimit) {

BorrowerCategory category = new BorrowerCategory();

category.setMaxLendingItems(maxLendingLimit);

borrower.setCategory(category);

}

@When("George Red borrows the item Harry Potter")

public void georgeRedBorrowsTheItemHarryPotter() {

librarySystem.loanItem(borrower, item);

}

@Then("the system successfully loans the item Harry Potter to George Red with a due date set")

public void theSystemSuccessfullyLoansTheItemHarryPotterToGeorgeRedWithADueDateSet() {

Loan loan = borrower.getLoans().stream().filter(l -> l.getItem().equals(item)).findFirst().orElse(null);

assert loan != null;

assert loan.getReturnDate() != null;

}

@Then("George Red's pending items increase to {int}")

public void georgeRedSPendingItemsIncreaseTo(int pendingItems) {

assert borrower.getLoans().size() == pendingItems;

}

@When("George Red tries to borrow both items")

public void georgeRedTriesToBorrowBothItems() {

Item anotherItem = new Item();

anotherItem.setBook(new Book());

anotherItem.setState(ItemState.AVAILABLE);

librarySystem.addItem(anotherItem);

librarySystem.loanItem(borrower, item);

librarySystem.loanItem(borrower, anotherItem);

}

@Then("the system does not loan Moby Dick to George Red due to the lending limit reached")

public void theSystemDoesNotLoanMobyDickToGeorgeRedDueToTheLendingLimitReached() {

assert borrower.getLoans().size() < borrower.getCategory().getMaxLendingItems();

}

@Given("the item Harry Potter is in the library but not in the system")

public void theItemHarryPotterIsInTheLibraryButNotInTheSystem() {

item = new Item();

item.setBook(new Book());

item.setState(ItemState.NEW);

}

@When("George Red tries to borrow the item Harry Potter")

public void georgeRedTriesToBorrowTheItemHarryPotter() {

try {

librarySystem.loanItem(borrower, item);

} catch (Exception e) {

// handle exception

}

}

@Then("the system returns an error due to the item's status")

public void theSystemReturnsAnErrorDueToTheItemSStatus() {

assert item.getState() == ItemState.NEW;

}

@Then("the system withdraws the item Harry Potter")

public void theSystemWithdrawsTheItemHarryPotter() {

librarySystem.removeItem(item);

assert librarySystem.findItem(item.getItemNumber()) == null;

}

@Then("the system doesn't allow the loan")

public void theSystemDoesnTAllowTheLoan() {

assert borrower.getLoans().size() < borrower.getCategory().getMaxLendingItems();

}

@Then("George Red's pending items remain {int}")

public void georgeRedSPendingItemsRemain(int pendingItems) {

assert borrower.getLoans().size() == pendingItems;

}

}

Please note that I've made some assumptions about the `LibrarySystem` class. You'll need to replace these with your actual methods or create them if they don't exist.

**now, with the same pattern give me the step definitions for the feature: Returning a borrowed item to the library**

Sure, let's create the step definitions for the feature: Returning a borrowed item to the library.

Here is the `StepDefinitions` class:

import io.cucumber.java.en.Given;

import io.cucumber.java.en.Then;

import io.cucumber.java.en.When;

import java.time.LocalDate;

import java.time.temporal.ChronoUnit;

public class StepDefinitions {

private Borrower borrower;

private Item item;

private LibrarySystem librarySystem = new LibrarySystem();

@Given("George Red borrowed the item Animal Kingdom {int} days prior to today's date")

public void georgeRedBorrowedTheItemAnimalKingdomDaysPriorToTodaySDate(int days) {

borrower = librarySystem.findBorrower("George Red");

if (borrower == null) {

borrower = new Borrower();

borrower.setBorrowerNo(librarySystem.generateUniqueBorrowerNumber());

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

librarySystem.registerBorrower(borrower);

}

item = new Item();

item.setBook(new Book());

item.setState(ItemState.BORROWED);

librarySystem.addItem(item);

Loan loan = new Loan(item, borrower);

loan.setLoanDate(LocalDate.now().minus(days, ChronoUnit.DAYS));

borrower.getLoans().add(loan);

}

@Given("George Red has been assigned maximum lending days of {int}")

public void georgeRedHasBeenAssignedMaximumLendingDaysOf(int maxLendingDays) {

BorrowerCategory category = new BorrowerCategory();

category.setMaxLendingDays(maxLendingDays);

borrower.setCategory(category);

}

@When("the return of Animal Kingdom is processed")

public void theReturnOfAnimalKingdomIsProcessed() {

librarySystem.returnItem(borrower, item);

}

@Then("the system marks the state of Animal Kingdom as AVAILABLE")

public void theSystemMarksTheStateOfAnimalKingdomAsAVAILABLE() {

assert item.getState() == ItemState.AVAILABLE;

}

@Then("George Red has one less pending item")

public void georgeRedHasOneLessPendingItem() {

assert borrower.getLoans().stream().noneMatch(loan -> loan.getItem().equals(item));

}

@Then("George Red does not pay a fine")

public void georgeRedDoesNotPayAFine() {

assert borrower.getCategory().getDailyFine().getAmount().equals(BigDecimal.ZERO);

}

@Then("the return date of the loan is set to today's date")

public void theReturnDateOfTheLoanIsSetToTodaySDate() {

Loan loan = borrower.getLoans().stream().filter(l -> l.getItem().equals(item)).findFirst().orElse(null);

assert loan != null;

assert loan.getReturnDate().equals(LocalDate.now());

}

@Then("George Red pays a fine based on the borrower category")

public void georgeRedPaysAFineBasedOnTheBorrowerCategory() {

int daysLate = (int) ChronoUnit.DAYS.between(item.getLoan().getLoanDate().plusDays(borrower.getCategory().getMaxLendingDays()), LocalDate.now());

BigDecimal fine = borrower.getCategory().getDailyFine().getAmount().multiply(BigDecimal.valueOf(daysLate));

assert fine.compareTo(BigDecimal.ZERO) > 0;

}

}

Please note that I've made some assumptions about the `LibrarySystem` class. You'll need to replace these with your actual methods or create them if they don't exist.

**now,, give me the step definitions for the feature: Delayed Notification**

Sure, let's create the step definitions for the feature: Delayed Notification.

Here is the `StepDefinitions` class:

import io.cucumber.java.en.Given;

import io.cucumber.java.en.Then;

import io.cucumber.java.en.When;

import java.time.LocalDate;

import java.time.temporal.ChronoUnit;

public class StepDefinitions {

private Borrower borrower;

private Item item;

private LibrarySystem librarySystem = new LibrarySystem();

@Given("George Red borrowed the item Animal Kingdom {int} days prior to today's date")

public void georgeRedBorrowedTheItemAnimalKingdomDaysPriorToTodaySDate(int days) {

borrower = librarySystem.findBorrower("George Red");

if (borrower == null) {

borrower = new Borrower();

borrower.setBorrowerNo(librarySystem.generateUniqueBorrowerNumber());

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

librarySystem.registerBorrower(borrower);

}

item = new Item();

item.setBook(new Book());

item.setState(ItemState.BORROWED);

librarySystem.addItem(item);

Loan loan = new Loan(item, borrower);

loan.setLoanDate(LocalDate.now().minus(days, ChronoUnit.DAYS));

borrower.getLoans().add(loan);

}

@Given("George Red has been assigned maximum lending days of {int}")

public void georgeRedHasBeenAssignedMaximumLendingDaysOf(int maxLendingDays) {

BorrowerCategory category = new BorrowerCategory();

category.setMaxLendingDays(maxLendingDays);

borrower.setCategory(category);

}

@When("the system checks for overdue items")

public void theSystemChecksForOverdueItems() {

librarySystem.checkForOverdueItems();

}

@Then("the system sends a notification to George Red about the overdue item")

public void theSystemSendsANotificationToGeorgeRedAboutTheOverdueItem() {

assert borrower.getNotifications().stream().anyMatch(notification -> notification.getItem().equals(item));

}

@Then("the system does not send a notification to George Red")

public void theSystemDoesNotSendANotificationToGeorgeRed() {

assert borrower.getNotifications().stream().noneMatch(notification -> notification.getItem().equals(item));

}

}

Please note that I've made some assumptions about the `LibrarySystem` class. You'll need to replace these with your actual methods or create them if they don't exist.