Στη συνομιλία αυτή, παρέθεσα στο LLM τις ίδιες ακριβώς πληροφορίες όπως στη Συνομιλία 1, αλλά δεν το καθοδήγησα να δημιουργήσει πρώτα τον κώδικα για το Domain, Services και DAOs. Ωστόσο, όπως σε όλες τις προηγούμενες συνομιλίες, το LLM τα δημιούργησε από μόνο του. Τα αποτελέσματα του κώδικα ήταν αρκετά καλά και παρόμοια με αυτά των άλλων συνομιλιών. Το LLM χρησιμοποίησε την αρχιτεκτονική που του είχα δώσει στην αρχή της συνομιλίας, παρείχε πολλές λεπτομέρειες στις απαντήσεις του και χρειάστηκε πολύ λίγα μηνύματα για να δημιουργήσει όλα τα Step Definitions για όλα τα features που παρείχα.

Phase 4 Chat 2

User

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?

ChatGPT

Your plan sounds clear, and I'm ready to help. Here is a breakdown of how we can proceed:

1. **Provide Domain Classes**: Share the Java code for your domain classes including their attributes and methods.

2. **Gherkin Scenarios**: Provide the Gherkin scenarios for the features of your library system application.

3. **Implementation of Step Definitions**: I will provide the Java implementation code for the Step definitions for each feature based on the Gherkin scenarios.

4. **Architecture Details**: I will ensure the code is structured into Domain classes, Data Access Objects (DAOs) for handling the database operations, and Service classes for the business logic.

Feel free to start by providing the domain classes and the Gherkin scenarios. Once I have this information, I'll proceed with creating the Step definitions and necessary implementation code.

User

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

ChatGPT

Great, I have noted down the details of your domain classes. Now, please provide the Gherkin scenarios for the features of your library system application. Once I have those, I'll be able to help you implement the Step definitions and necessary service, DAO, and domain logic in Java.

User

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

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

ChatGPT

Based on the given domain classes and Gherkin scenarios, we will follow a structured approach to implement the Step definitions and necessary service, DAO, and domain logic in Java.

Here is the plan:

1. **Data Access Objects (DAOs)**: Create DAOs for each of the domain classes to handle the database operations.

2. **Services**: Create service classes to encapsulate the business logic.

3. **Step Definitions**: Implement the step definitions based on the Gherkin scenarios.

Step 1: Create Data Access Objects (DAOs)

BorrowerDAO.java

```java
```

```
}


import java.util.HashMap;

import java.util.Map;


public class BorrowerDAO {

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


    public void save(Borrower borrower) {

        borrowers.put(borrower.getBorrowerNo(), borrower);

    }


    public Borrower findById(int borrowerNo) {

        return borrowers.get(borrowerNo);

    }


    public void delete(int borrowerNo) {

        borrowers.remove(borrowerNo);

    }


    public boolean exists(int borrowerNo) {

        return borrowers.containsKey(borrowerNo);

    }

}
```

~~~

Step 2: Create Service Classes

BorrowerService.java

```java
public class BorrowerService {
    private BorrowerDAO borrowerDAO;

    public BorrowerService(BorrowerDAO borrowerDAO) {
        this.borrowerDAO = borrowerDAO;
    }

    public void registerBorrower(Borrower borrower) throws Exception {
        if (borrowerDAO.exists(borrower.getBorrowerNo())) {
            throw new Exception("Borrower already exists");
        }
        borrowerDAO.save(borrower);
    }

    public void updateBorrower(Borrower borrower) throws Exception {
        if (!borrowerDAO.exists(borrower.getBorrowerNo())) {
~~~

```
            throw new Exception("Borrower does not exist");
        }
        borrowerDAO.save(borrower);
    }

    public void deleteBorrower(int borrowerNo) throws Exception {
        if (!borrowerDAO.exists(borrowerNo)) {
            throw new Exception("Borrower does not exist");
        }
        Borrower borrower = borrowerDAO.findById(borrowerNo);
        if (borrower.countPendingItems() > 0) {
            throw new Exception("Borrower has pending items");
        }
        borrowerDAO.delete(borrowerNo);
    }
}
```

Step 3: Implement Step Definitions

BorrowerStepDefinitions.java

```java
```

```
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;
import static org.junit.Assert.*;

public class BorrowerStepDefinitions {
    private BorrowerDAO borrowerDAO = new BorrowerDAO();
    private BorrowerService borrowerService = new BorrowerService(borrowerDAO);
    private Borrower borrower;
    private Exception exception;

    @Given("George Red is not registered as a borrower")
    public void george_red_is_not_registered_as_a_borrower() {
        borrower = new Borrower(1, /*other details*/);
        borrowerDAO.delete(borrower.getBorrowerNo());
    }

    @When("George Red gets registered in the system with a unique borrower number and his details")
    public void george_red_gets_registered_in_the_system() {
        try {
            borrowerService.registerBorrower(borrower);
        } catch (Exception e) {
            exception = e;
        }
    }

    @Then("the system successfully stores the borrower's details")
    public void the_system_successfully_stores_the_borrower_s_details() {
        assertNull(exception);
        assertNotNull(borrowerDAO.findById(borrower.getBorrowerNo()));
    }

    @Given("George Red is registered as a borrower")
    public void george_red_is_registered_as_a_borrower() {
        borrower = new Borrower(1, /*other details*/);
        borrowerDAO.save(borrower);
    }

    @When("the system attempts to register George Red with a unique borrower number and his
details")
    public void the_system_attempts_to_register_george_red() {
        try {
            borrowerService.registerBorrower(borrower);
        } catch (Exception e) {
            exception = e;
        }
    }

    @Then("the system informs that the user already exists")
    public void the_system_informs_that_the_user_already_exists() {
        assertNotNull(exception);
        assertEquals("Borrower already exists", exception.getMessage());
    }

    @When("George Red updates his borrowing details")
    public void george_red_updates_his_borrowing_details() {
        borrower.setEmailAddress(new EmailAddress("newemail@example.com"));
        try {
            borrowerService.updateBorrower(borrower);
        } catch (Exception e) {
            exception = e;
        }
    }

    @Then("the system saves the changes")
```

```
    public void the_system_saves_the_changes() {
        assertNull(exception);
        assertEquals("newemail@example.com",
borrowerDAO.findById(borrower.getBorrowerNo()).getEmailAddress().getEmail());
    }


    @When("George Red tries to update his borrowing details")
    public void george_red_tries_to_update_his_borrowing_details() {
        try {
            borrowerService.updateBorrower(borrower);
        } catch (Exception e) {
            exception = e;
        }
    }


    @Then("the system displays an error message indicating that George Red does not exist")
    public void the_system_displays_an_error_message() {
        assertNotNull(exception);
        assertEquals("Borrower does not exist", exception.getMessage());
    }


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

```
public void the_system_deletes_george_red_s_account() {

    try {

        borrowerService.deleteBorrower(borrower.getBorrowerNo());

    } catch (Exception e) {

        exception = e;

    }

}


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

public void the_system_removes_george_red_s_details() {

    assertNull(exception);

    assertNull(borrowerDAO.findById(borrower.getBorrowerNo()));

}


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

public void the_system_attempts_to_delete_george_red_s_account() {

    try {

        borrowerService.deleteBorrower(borrower.getBorrowerNo());

    } catch (Exception e) {

        exception = e;

    }

}
```

```
    @Then("the system informs that the borrower does not exist")
    public void the_system_informs_that_the_borrower_does_not_exist() {
        assertNotNull(exception);
        assertEquals("Borrower does not exist", exception.getMessage());
    }

    @Given("George Red has pending items")
    public void george_red_has_pending_items() {
        Loan loan = new Loan(LocalDate.now().minusDays(5), null, borrower, new Item(1, /*other details*/));
        borrower.getLoans().add(loan);
    }

    @Then("the system does not remove George Red's details")
    public void the_system_does_not_remove_george_red_s_details() {
        assertNotNull(exception);
        assertEquals("Borrower has pending items", exception.getMessage());
    }

    @Then("the system informs about the pending items")
    public void the_system_informs_about_the_pending_items() {
        assertNotNull(exception);
        assertEquals("Borrower has pending items", exception.getMessage());
    }
}
```

The implementation includes the data access object for the `Borrower`, the service class handling the business logic, and the step definitions for the feature "Borrower handling by the system". Similar steps can be followed for implementing the rest of the features and scenarios. Let me know if you want the implementations for other features or any specific part elaborated.

User

Implement all the step definitions of all the features I provided you with

ChatGPT

Sure, I'll provide the complete implementation of the step definitions for all the features and scenarios you provided. To keep the response organized, I'll group the step definitions by feature.

Step Definitions for Borrower Handling

BorrowerStepDefinitions.java

```java
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;
import static org.junit.Assert.*;

import java.time.LocalDate;
import java.util.HashSet;

public class BorrowerStepDefinitions {
    private BorrowerDAO borrowerDAO = new BorrowerDAO();
    private BorrowerService borrowerService = new BorrowerService(borrowerDAO);
    private Borrower borrower;
    private Exception exception;

    @Given("George Red is not registered as a borrower")
    public void george_red_is_not_registered_as_a_borrower() {
        borrower = new Borrower(1, /* TelephoneNumber */, /* EmailAddress */, /* Address */, /* BorrowerCategory */, new Person("George", "Red"), new HashSet<>());
        borrowerDAO.delete(borrower.getBorrowerNo());
    }

    @When("George Red gets registered in the system with a unique borrower number and his details")
    public void george_red_gets_registered_in_the_system() {
        try {
            borrowerService.registerBorrower(borrower);
        } catch (Exception e) {
            exception = e;
        }
    }


    @Then("the system successfully stores the borrower's details")
    public void the_system_successfully_stores_the_borrower_s_details() {
        assertNull(exception);
        assertNotNull(borrowerDAO.findById(borrower.getBorrowerNo()));
    }


    @Given("George Red is registered as a borrower")
    public void george_red_is_registered_as_a_borrower() {
        borrower = new Borrower(1, /* TelephoneNumber */, /* EmailAddress */, /* Address */, /* BorrowerCategory */, new Person("George", "Red"), new HashSet<>());
        borrowerDAO.save(borrower);
    }
    @When("the system attempts to register George Red with a unique borrower number and his details")
    public void the_system_attempts_to_register_george_red() {
        try {
            borrowerService.registerBorrower(borrower);
        } catch (Exception e) {
            exception = e;
        }
    }

    @Then("the system informs that the user already exists")
    public void the_system_informs_that_the_user_already_exists() {
        assertNotNull(exception);
        assertEquals("Borrower already exists", exception.getMessage());
    }

    @When("George Red updates his borrowing details")
    public void george_red_updates_his_borrowing_details() {
        borrower.setEmailAddress(new EmailAddress("newemail@example.com"));
        try {
            borrowerService.updateBorrower(borrower);
        } catch (Exception e) {
```

```
        exception = e;

      }

    }


    @Then("the system saves the changes")

    public void the_system_saves_the_changes() {

      assertNull(exception);

      assertEquals("newemail@example.com",
borrowerDAO.findById(borrower.getBorrowerNo()).getEmailAddress().getEmail());

    }


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

    public void george_red_tries_to_update_his_borrowing_details() {

      try {

        borrowerService.updateBorrower(borrower);

      } catch (Exception e) {

        exception = e;

      }

    }


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

    public void the_system_displays_an_error_message() {
```

```
    assertNotNull(exception);
    assertEquals("Borrower does not exist", exception.getMessage());
}

@When("the system deletes George Red's account")
public void the_system_deletes_george_red_s_account() {
    try {
        borrowerService.deleteBorrower(borrower.getBorrowerNo());
    } catch (Exception e) {
        exception = e;
    }
}

@Then("the system removes George Red's details")
public void the_system_removes_george_red_s_details() {
    assertNull(exception);
    assertNull(borrowerDAO.findById(borrower.getBorrowerNo()));
}

@When("the system attempts to delete George Red's account")
public void the_system_attempts_to_delete_george_red_s_account() {
    try {
        borrowerService.deleteBorrower(borrower.getBorrowerNo());
    } catch (Exception e) {
        exception = e;
    }
}

@Then("the system informs that the borrower does not exist")
public void the_system_informs_that_the_borrower_does_not_exist() {
    assertNotNull(exception);
    assertEquals("Borrower does not exist", exception.getMessage());
}

@Given("George Red has pending items")
public void george_red_has_pending_items() {
    Item item = new Item(1, new Book(/* ISBN */, "Harry Potter", "Publication", 2020, new Publisher(), new HashSet<>(), new HashSet<>()), ItemState.LOANED);
    Loan loan = new Loan(LocalDate.now().minusDays(5), null, borrower, item);
    borrower.getLoans().add(loan);
}

@Then("the system does not remove George Red's details")
public void the_system_does_not_remove_george_red_s_details() {
    assertNotNull(exception);
    assertEquals("Borrower has pending items", exception.getMessage());
}

@Then("the system informs about the pending items")
public void the_system_informs_about_the_pending_items() {
    assertNotNull(exception);
    assertEquals("Borrower has pending items", exception.getMessage());
}
}
```

Step Definitions for Delayed Return Notification

DelayedReturnNotificationStepDefinitions.java

```java
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;
import static org.junit.Assert.*;

import java.time.LocalDate;
import java.util.HashSet;


public class DelayedReturnNotificationStepDefinitions {
    private BorrowerDAO borrowerDAO = new BorrowerDAO();
    private NotificationService notificationService = new NotificationService(borrowerDAO);
    private Borrower borrower;
    private Book book;
    private Item item;
    private Exception exception;


    @Given("George Red has borrowed the item Harry Potter")
    public void george_red_has_borrowed_the_item_harry_potter() {
        book = new Book(/* ISBN */, "Harry Potter", "Publication", 2020, new Publisher(), new HashSet<>(), new HashSet<>());
        item = new Item(1, book, ItemState.LOANED);
        borrower = new Borrower(1, /* TelephoneNumber */, new EmailAddress("george@example.com"), /* Address */, /* BorrowerCategory */, new Person("George", "Red"), new HashSet<>());
        Loan loan = new Loan(LocalDate.now().minusDays(10), null, borrower, item);
        borrower.getLoans().add(loan);
        borrowerDAO.save(borrower);
    }
```

```
@Given("Harry Potter's due date has passed")

public void harry_potter_s_due_date_has_passed() {

    // This is already handled in the previous step

}


@Given("George Red has an email address")

public void george_red_has_an_email_address() {

    assertNotNull(borrower.getEmailAddress());

}


@When("the system executes the delayed return notification process")

public void the_system_executes_the_delayed_return_notification_process() {

    try {

        notificationService.notifyDelayedReturn();

    } catch (Exception e) {

        exception = e;

    }

}


@Then("George Red receives an email notification for the return of the item")

public void george_red_receives_an_email_notification_for_the_return_of_the_item() {
        assertNull(exception);
        // Add assertion to check if the email was sent (implementation dependent)
    }

    @Given("George Red does not have an email address")
    public void george_red_does_not_have_an_email_address() {
        borrower.setEmailAddress(null);
        borrowerDAO.save(borrower);
    }

    @Then("George Red does not receive an email notification for the return of the item")
    public void george_red_does_not_receive_an_email_notification_for_the_return_of_the_item() {
        assertNull(exception);
        // Add assertion to check if the email was not sent (implementation dependent)
    }
}
```

Step Definitions for Loaning Items

LoaningItemsStepDefinitions.java

~~~
```java
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;
import static org.junit.Assert.*;

import java.util.HashSet;

public class LoaningItemsStepDefinitions {
    private BorrowerDAO borrowerDAO = new BorrowerDAO();
    private BookDAO bookDAO = new BookDAO();
    private LoanService loanService = new LoanService(borrowerDAO, bookDAO);
    private Borrower borrower;
    private Book bookHarryPotter;
    private Book bookMobyDick;
    private Item itemHarryPotter;
    private Item itemMobyDick;
    private Exception exception;

    @Given("the library has the item Harry Potter available")
    public void the_library_has_the_item_harry_potter_available() {
~~~

```
        bookHarryPotter = new Book(/* ISBN */, "Harry Potter", "Publication", 2020, new Publisher(), new HashSet<>(), new HashSet<>());

        itemHarryPotter = new Item(1, bookHarryPotter, ItemState.AVAILABLE);

        bookHarryPotter.addItem(itemHarryPotter);

        bookDAO.save(bookHarryPotter);

    }


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

    public void george_red_is_a_registered

ChatGPT

_borrower() {

        borrower = new Borrower(1, /* TelephoneNumber */, new EmailAddress("george@example.com"), /* Address */, new BorrowerCategory("Standard", 5, 10, new Money(10)), new Person("George", "Red"), new HashSet<>());

        borrowerDAO.save(borrower);

    }


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

    public void george_red_has_pending_items_to_be_returned(int pendingItems) {

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

            Item item = new Item(i + 1, bookHarryPotter, ItemState.LOANED);

            Loan loan = new Loan(LocalDate.now().minusDays(5), null, borrower, item);
            borrower.getLoans().add(loan);
        }
    }

    @Given("George Red has been assigned a maximum lending limit of {int}")
    public void george_red_has_been_assigned_a_maximum_lending_limit_of(int maxLendingLimit) {
        borrower.getCategory().setMaxLendingItems(maxLendingLimit);
        borrowerDAO.save(borrower);
    }

    @When("George Red borrows the item Harry Potter")
    public void george_red_borrows_the_item_harry_potter() {
        try {
            loanService.loanItem(borrower.getBorrowerNo(), itemHarryPotter.getItemNumber());
        } catch (Exception e) {
            exception = e;
        }
    }

    @Then("the system successfully loans the item Harry Potter to George Red with a due date set")
    public void
the_system_successfully_loans_the_item_harry_potter_to_george_red_with_a_due_date_set()
{
        assertNull(exception);
        assertEquals(ItemState.LOANED, itemHarryPotter.getState());
        assertNotNull(itemHarryPotter.getBorrow().getDue());
    }

    @Then("George Red's pending items increase to {int}")
    public void george_red_s_pending_items_increase_to(int pendingItems) {
        assertEquals(pendingItems, borrower.countPendingItems());
    }

    @Given("the library has the items Harry Potter and Moby Dick available")
    public void the_library_has_the_items_harry_potter_and_moby_dick_available() {
        bookMobyDick = new Book(/* ISBN */, "Moby Dick", "Publication", 2020, new Publisher(),
new HashSet<>(), new HashSet<>());
        itemMobyDick = new Item(2, bookMobyDick, ItemState.AVAILABLE);
        bookMobyDick.addItem(itemMobyDick);
        bookDAO.save(bookMobyDick);
    }
```

```java
    @When("George Red tries to borrow both items")
    public void george_red_tries_to_borrow_both_items() {
        try {
            loanService.loanItem(borrower.getBorrowerNo(), itemHarryPotter.getItemNumber());
            loanService.loanItem(borrower.getBorrowerNo(), itemMobyDick.getItemNumber());
        } catch (Exception e) {
            exception = e;
        }
    }

    @Then("the system does not loan Moby Dick to George Red due to the lending limit reached")
    public void the_system_does_not_loan_moby_dick_to_george_red_due_to_the_lending_limit_reached() {
        assertNotNull(exception);
        assertEquals("Borrower has reached the maximum lending limit", exception.getMessage());
        assertEquals(ItemState.AVAILABLE, itemMobyDick.getState());
    }

    @Given("the item Harry Potter is in the library but not in the system")
    public void the_item_harry_potter_is_in_the_library_but_not_in_the_system() {
        bookHarryPotter = new Book(/* ISBN */, "Harry Potter", "Publication", 2020, new Publisher(), new HashSet<>(), new HashSet<>());
        itemHarryPotter = new Item(1, bookHarryPotter, ItemState.NEW);

        bookHarryPotter.addItem(itemHarryPotter);

    }


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

    public void george_red_tries_to_borrow_the_item_harry_potter() {

        try {

            loanService.loanItem(borrower.getBorrowerNo(), itemHarryPotter.getItemNumber());

        } catch (Exception e) {

            exception = e;

        }

    }


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

    public void the_system_returns_an_error_due_to_the_item_s_status() {

        assertNotNull(exception);

        assertEquals("Item is not available for loan", exception.getMessage());

    }


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

    public void the_system_withdraws_the_item_harry_potter() {

        assertEquals(ItemState.WITHDRAWN, itemHarryPotter.getState());
    }

    @When("George Red tries to borrow the item Harry Potter")
    public void george_red_tries_to_borrow_the_item_harry_potter_again() {
        try {
            loanService.loanItem(borrower.getBorrowerNo(), itemHarryPotter.getItemNumber());
        } catch (Exception e) {
            exception = e;
        }
    }

    @Then("the system doesn't allow the loan")
    public void the_system_doesn_t_allow_the_loan() {
        assertNotNull(exception);
        assertEquals("Borrower has reached the maximum lending limit", exception.getMessage());
    }

    @Then("George Red's pending items remain {int}")
    public void george_red_s_pending_items_remain(int pendingItems) {
        assertEquals(pendingItems, borrower.countPendingItems());
    }
}
```

~~~

Step Definitions for Returning a Borrowed Item

ReturningItemsStepDefinitions.java

```java
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;
import static org.junit.Assert.*;

import java.time.LocalDate;
import java.util.HashSet;

public class ReturningItemsStepDefinitions {
    private BorrowerDAO borrowerDAO = new BorrowerDAO();
    private BookDAO bookDAO = new BookDAO();
    private ReturnService returnService = new ReturnService(borrowerDAO, bookDAO);
    private Borrower borrower;
    private Book bookAnimalKingdom;
    private Item itemAnimalKingdom;
    private Exception exception;
~~~

```
    @Given("George Red borrowed the item Animal Kingdom {int} days prior to today's date")
    public void george_red_borrowed_the_item_animal_kingdom_days_prior_to_today_s_date(int daysPrior) {
        bookAnimalKingdom = new Book(/* ISBN */, "Animal Kingdom", "Publication", 2020, new Publisher(), new HashSet<>(), new HashSet<>());
        itemAnimalKingdom = new Item(1, bookAnimalKingdom, ItemState.LOANED);
        bookAnimalKingdom.addItem(itemAnimalKingdom);
        borrower = new Borrower(1, /* TelephoneNumber */, new EmailAddress("george@example.com"), /* Address */, new BorrowerCategory("Standard", 5, 10, new Money(10)), new Person("George", "Red"), new HashSet<>());
        Loan loan = new Loan(LocalDate.now().minusDays(daysPrior), null, borrower, itemAnimalKingdom);
        borrower.getLoans().add(loan);
        borrowerDAO.save(borrower);
        bookDAO.save(bookAnimalKingdom);
    }

    @Given("George Red has been assigned maximum lending days of {int}")
    public void george_red_has_been_assigned_maximum_lending_days_of(int maxLendingDays) {
        borrower.getCategory().setMaxLendingDays(maxLendingDays);
        borrowerDAO.save(borrower);
    }


    @When("the return of Animal Kingdom is processed")
    public void the_return_of_animal_kingdom_is_processed() {
        try {
            returnService.returnItem(borrower.getBorrowerNo(),
itemAnimalKingdom.getItemNumber());
        } catch (Exception e) {
            exception = e;
        }
    }


    @Then("the system marks the state of Animal Kingdom as AVAILABLE")
    public void the_system_marks_the_state_of_animal_kingdom_as_available() {
        assertEquals(ItemState.AVAILABLE, itemAnimalKingdom.getState());
    }


    @Then("George Red has one less pending item")
    public void george_red_has_one_less_pending_item() {
        assertEquals(0, borrower.countPendingItems());
    }
    @Then("George Red does not pay a fine")
    public void george_red_does_not_pay_a_fine() {
        assertEquals(0, borrower.getLoans().iterator().next().getFine().getAmount().intValue());
    }

    @Then("the return date of the loan is set to today's date")
    public void the_return_date_of_the_loan_is_set_to_today_s_date() {
        assertEquals(LocalDate.now(), borrower.getLoans().iterator().next().getReturnDate());
    }

    @Then("George Red pays a fine based on the borrower category")
    public void george_red_pays_a_fine_based_on_the_borrower_category() {
        assertEquals(30, borrower.getLoans().iterator().next().getFine().getAmount().intValue());
    }
}
```

Conclusion

This implementation includes the data access object (DAO) for `Borrower` and `Book`, the service classes (`BorrowerService`, `NotificationService`, `LoanService`, and `ReturnService`) handling the business logic, and the step definitions for the features "Borrower handling by the system", "Delayed Return Notification", "Loaning items", and "Returning a borrowed item to the library".

Make sure to implement the remaining DAOs (`BookDAO`, `LoanDAO`) and the service classes (`NotificationService`, `LoanService`, and `ReturnService`) similarly. You may also need to add additional utility classes like `EmailService` to handle email notifications.

ChatGPT

Feel free to reach out if you need further assistance or more details on any part of the implementation.