Στη συνομιλία αυτή, παρέθεσα στο LLM τα features του συστήματος, τις κλάσεις Domain του συστήματος με τις ιδιότητές τους, και την περιγραφή του συστήματος. Όπως σε κάθε άλλη φάση με αυτό το LLM, το LLM κατάλαβε εξαιρετικά γρήγορα τι έπρεπε να κάνει και πώς να συνδέσει τις κλάσεις. Δημιούργησε όλο τον κώδικα για τις κλάσεις Domain, Services και DAOs χωρίς να χρειαστεί καθοδήγηση, και στη συνέχεια, με μόνο 2 μηνύματα, δημιούργησε όλα τα Step Definitions για όλα τα features. Συνολικά, ο κώδικας ήταν αρκετά καλός, πολλές φορές πολύ κοντά στον πραγματικό κώδικα του συστήματος, και περιλάμβανε πολλές λεπτομέρειες σε κάθε βήμα, χωρίς να καταφύγει στην εύκολη λύση με απλές δηλώσεις ή εκτυπώσεις στους ορισμούς βημάτων.

Phase 3 Chat 1

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 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?

ChatGPT

No, I don't have any questions. I'm ready to help you with the implementation of the Step definitions for your library system application. Please provide me with the Gherkin Scenarios, the name of the domain classes along with their attributes, and any other details you think are necessary.

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>()

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 fields:

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 fields:

private String firstName,

private String lastName

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

ChatGPT

Great, I have noted down the domain classes and their attributes. Please go ahead and provide the Gherkin Scenarios for the features of your library application, and I will help you with the implementation of the Step definitions in Java, adhering to the specified architecture with Domain classes, Data Access Objects (DAOs), and Service classes.

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

Step Definitions for Feature: Borrower handling by the system

Below are the Java step definitions using Cucumber framework:

Domain Classes

First, ensure you have the domain classes defined as follows (simplified for brevity):

```java
public class Book {
    private String 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
}

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
}

public class BorrowerCategory {
    private String description;
    private int maxLendingItems;
    private int maxLendingDays;
    private Money dailyFine;
    // Getters and Setters
}

public class Item {
    private int itemNumber;

    private Book book;

    private ItemState state = ItemState.NEW;

    // Getters and Setters

}


public enum ItemState {

    NEW, AVAILABLE, BORROWED, RETURNED;

}


public class Loan {

    private LocalDate loanDate = LocalDate.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

}
```

DAO Interfaces

Create Data Access Objects interfaces to manage persistence.

```java
public interface BorrowerDAO {
    Borrower findByBorrowerNo(int borrowerNo);
    void save(Borrower borrower);
    void update(Borrower borrower);
    void delete(Borrower borrower);
}
```

Service Class

A service class to handle business logic.

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

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


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


    public void updateBorrower(Borrower borrower) throws Exception {
        if (borrowerDAO.findByBorrowerNo(borrower.getBorrowerNo()) == null) {
            throw new Exception("Borrower does not exist");
        }
        borrowerDAO.update(borrower);
    }


    public void deleteBorrower(int borrowerNo) throws Exception {
        Borrower borrower = borrowerDAO.findByBorrowerNo(borrowerNo);
        if (borrower == null) {
            throw new Exception("Borrower does not exist");
        }
        borrowerDAO.delete(borrower);
    }
```

```
}
```

Step Definitions

The step definitions for the feature using the Cucumber framework.

```java
import static org.junit.Assert.*;

import java.util.HashMap;
import java.util.Map;

import cucumber.api.java.en.*;

public class BorrowerHandlingSteps {
    private BorrowerService borrowerService;
    private Map<Integer, Borrower> borrowers;
    private Borrower currentBorrower;
    private Exception exception;

    public BorrowerHandlingSteps() {
        // Assuming BorrowerDAO is properly implemented
        BorrowerDAO borrowerDAO = new InMemoryBorrowerDAO();
        borrowerService = new BorrowerService(borrowerDAO);

        borrowers = new HashMap<>();

        exception = null;

    }


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

    public void george_Red_is_not_registered_as_a_borrower() {

        currentBorrower = null;

    }


    @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 {

            currentBorrower = new Borrower();

            currentBorrower.setBorrowerNo(1); // Unique borrower number

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

            borrowerService.registerBorrower(currentBorrower);

        } catch (Exception e) {

            exception = e;

        }

    }


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

    public void the_system_successfully_stores_the_borrowers_details() {
        assertNull(exception);
        assertNotNull(borrowers.get(1));
    }

    @Given("George Red is registered as a borrower")
    public void george_Red_is_registered_as_a_borrower() {
        currentBorrower = new Borrower();
        currentBorrower.setBorrowerNo(1);
        currentBorrower.setPerson(new Person("George", "Red"));
        borrowers.put(1, currentBorrower);
    }

    @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(currentBorrower);
        } 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() {
        try {
            currentBorrower.getPerson().setTelephone(new TelephoneNumber("1234567890"));
            borrowerService.updateBorrower(currentBorrower);
        } catch (Exception e) {
            exception = e;
        }
    }

    @Then("the system saves the changes")
    public void the_system_saves_the_changes() {
        assertNull(exception);
        assertEquals("1234567890",
borrowers.get(1).getPerson().getTelephone().getNumber());
    }

    @When("George Red tries to update his borrowing details")
    public void george_Red_tries_to_update_his_borrowing_details() {
        try {
            currentBorrower = new Borrower();
            currentBorrower.setBorrowerNo(2); // New borrower number
            currentBorrower.setPerson(new Person("George", "Red"));
            borrowerService.updateBorrower(currentBorrower);
        } 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_indicating_that_George_Red_does_not_exist() {
        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(1);
        } 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(borrowers.get(1));
    }


    @When("the system attempts to delete George Red's account")
    public void the_system_attempts_to_delete_George_Red_s_account() {
        try {
            borrowerService.deleteBorrower(2);
        } 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());
    }
}
```

~~~

In-Memory DAO Implementation

To test the above steps, you may need an in-memory DAO implementation.

```java
public class InMemoryBorrowerDAO implements BorrowerDAO {
    private Map<Integer, Borrower> database = new HashMap<>();

    @Override
    public Borrower findByBorrowerNo(int borrowerNo) {
        return database.get(borrowerNo);
    }

    @Override
    public void save(Borrower borrower) {
        database.put(borrower.getBorrowerNo(), borrower);
    }

    @Override
    public void update(Borrower borrower) {
        database.put(borrower.getBorrowerNo(), borrower);
    }
~~~

```
    @Override
    public void delete(Borrower borrower) {
        database.remove(borrower.getBorrowerNo());
    }
}
```

Summary

This implementation includes the domain classes, a DAO interface, a service class for business logic, and Cucumber step definitions. Modify as needed to fit your specific requirements and test setup. Provide the same structure for the other scenarios and features.

User

Provide all the step definitions for all the other features as well

ChatGPT

Step Definitions for Feature: Delayed Return Notification

Step Definitions

```java
import cucumber.api.java.en.*;
import java.time.LocalDate;
import static org.junit.Assert.*;
```

```
public class DelayedReturnNotificationSteps {

    private BorrowerService borrowerService;

    private LoanService loanService;

    private Borrower currentBorrower;

    private Loan currentLoan;

    private Exception exception;

    private EmailService emailService;

    private boolean emailSent;


    public DelayedReturnNotificationSteps() {

        // Assuming the necessary DAOs are properly implemented

        BorrowerDAO borrowerDAO = new InMemoryBorrowerDAO();

        LoanDAO loanDAO = new InMemoryLoanDAO();

        borrowerService = new BorrowerService(borrowerDAO);

        loanService = new LoanService(loanDAO);

        emailService = new EmailService();

        exception = null;

        emailSent = false;

    }


    @Given("George Red has borrowed the item Harry Potter")

    public void george_Red_has_borrowed_the_item_Harry_Potter() {

        currentBorrower = new Borrower();
        currentBorrower.setBorrowerNo(1);
        currentBorrower.setPerson(new Person("George", "Red"));
        Item harryPotter = new Item();
        harryPotter.setItemNumber(1);
        harryPotter.setBook(new Book("Harry Potter", "JK Rowling", 1997));
        currentLoan = new Loan();
        currentLoan.setBorrower(currentBorrower);
        currentLoan.setItem(harryPotter);
        loanService.save(currentLoan);
    }

    @Given("Harry Potter's due date has passed")
    public void harry_Potter_s_due_date_has_passed() {
        currentLoan.setReturnDate(LocalDate.now().minusDays(5));
    }

    @Given("George Red has an email address")
    public void george_Red_has_an_email_address() {
        currentBorrower.setEmailAddress(new EmailAddress("george.red@example.com"));
    }

    @When("the system executes the delayed return notification process")
    public void the_system_executes_the_delayed_return_notification_process() {
        try {
            emailService.sendDelayedReturnNotification(currentLoan);
            emailSent = true;
        } 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() {
        assertTrue(emailSent);
        assertNull(exception);
    }

    @Given("George Red does not have an email address")
    public void george_Red_does_not_have_an_email_address() {
        currentBorrower.setEmailAddress(null);
    }

    @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() {
        assertFalse(emailSent);
        assertNotNull(exception);
        assertEquals("Email address not found", exception.getMessage());
    }
}
```

Step Definitions for Feature: Loaning Items

Step Definitions

```java
import cucumber.api.java.en.*;
import java.util.HashMap;
import java.util.Map;
import static org.junit.Assert.*;

public class LoaningItemsSteps {
    private BorrowerService borrowerService;
    private ItemService itemService;
    private LoanService loanService;
    private Borrower currentBorrower;
    private Item currentItem;
    private Map<Integer, Item> items;
    private Exception exception;
```

```
public LoaningItemsSteps() {

    // Assuming the necessary DAOs are properly implemented

    BorrowerDAO borrowerDAO = new InMemoryBorrowerDAO();

    ItemDAO itemDAO = new InMemoryItemDAO();

    LoanDAO loanDAO = new InMemoryLoanDAO();

    borrowerService = new BorrowerService(borrowerDAO);

    itemService = new ItemService(itemDAO);

    loanService = new LoanService(loanDAO);

    items = new HashMap<>();

    exception = null;

}


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

public void the_library_has_the_item_Harry_Potter_available() {

    currentItem = new Item();

    currentItem.setItemNumber(1);

    currentItem.setBook(new Book("Harry Potter", "JK Rowling", 1997));

    currentItem.setState(ItemState.AVAILABLE);

    items.put(1, currentItem);

    itemService.save(currentItem);

}
```

```
    }


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

    public void george_Red_is_a_registered_borrower() {

        currentBorrower = new Borrower();

        currentBorrower.setBorrowerNo(1);

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

        borrowerService.save(currentBorrower);

    }


    @Given("George Red has 2 pending items to be returned")

    public void george_Red_has_pending_items() {

        currentBorrower.getLoans().add(new Loan(LocalDate.now().minusDays(2), null,
currentBorrower, currentItem));

        currentBorrower.getLoans().add(new Loan(LocalDate.now().minusDays(2), null,
currentBorrower, new Item()));

    }


    @Given("George Red has been assigned a maximum lending limit of 5")

    public void george_Red_maximum_lending_limit() {

        BorrowerCategory category = new BorrowerCategory();

        category.setMaxLendingItems(5);

        currentBorrower.setCategory(category);

    }


    @When("George Red borrows the item Harry Potter")
    public void george_Red_borrows_the_item_Harry_Potter() {
        try {
            loanService.loanItem(currentBorrower, currentItem);
        } catch (Exception e) {
            exception = e;
        }
    }

    @Then("the system successfully loans the item Harry Potter to George Red with a due date set")
    public void system_successfully_loans_the_item() {
        assertNull(exception);
        assertEquals(ItemState.BORROWED, currentItem.getState());
        assertNotNull(currentLoan.getLoanDate());
    }

    @Then("George Red's pending items increase to 3")
    public void george_Red_pending_items_increase() {
        assertEquals(3, currentBorrower.getLoans().size());
    }

    @Given("the library has the items Harry Potter and Moby Dick available")
    public void library_has_multiple_items() {
        currentItem = new Item();
        currentItem.setItemNumber(1);
        currentItem.setBook(new Book("Harry Potter", "JK Rowling", 1997));
        currentItem.setState(ItemState.AVAILABLE);
        items.put(1, currentItem);
        itemService.save(currentItem);

        Item mobyDick = new Item();
        mobyDick.setItemNumber(2);
        mobyDick.setBook(new Book("Moby Dick", "Herman Melville", 1851));
        mobyDick.setState(ItemState.AVAILABLE);
        items.put(2, mobyDick);
        itemService.save(mobyDick);
    }

    @Given("George Red has been assigned a maximum lending limit of 3")
    public void george_Red_max_lending_limit() {
        BorrowerCategory category = new BorrowerCategory();
        category.setMaxLendingItems(3);
        currentBorrower.setCategory(category);
    }

    @When("George Red tries to borrow both items")
    public void george_Red_tries_to_borrow_both_items() {
        try {
            loanService.loanItem(currentBorrower, items.get(1));
            loanService.loanItem(currentBorrower, items.get(2));
        } catch (Exception e) {
            exception = e;
        }
    }

    @Then("the system does not loan Moby Dick to George Red due to the lending limit reached")
    public void system_does_not_loan_due_to_limit() {
        assertNotNull(exception);
        assertEquals("Lending limit reached", exception.getMessage());
        assertEquals(ItemState.AVAILABLE, items.get(2).getState());
    }

    @Then("George Red's pending items increase to 3")
    public void george_Red_pending_items() {
        assertEquals(3, currentBorrower.getLoans().size());
    }

    @Given("the item Harry Potter is in the library but not in the system")
    public void item_not_in_system() {
        currentItem = null; // Item is not in the system
```

```
}

@When("George Red tries to borrow the item Harry Potter")
public void george_Red_tries_to_borrow_item() {
    try {
        loanService.loanItem(currentBorrower, currentItem);
    } catch (Exception e) {
        exception = e;
    }
}

@Then("the system returns an error due to the item's status")
public void system_returns_error_due_to_status() {
    assertNotNull(exception);
    assertEquals("Item not found", exception.getMessage());
}

@Then("the system withdraws the item Harry Potter")
public void system_withdraws_item() {
    assertNull(items.get(1));
}

@Given("George Red has 3 pending items to be returned")
```

```
    public void george_Red_3_pending_items() {
        currentBorrower.getLoans().add(new Loan(LocalDate.now().minusDays(2), null, currentBorrower, currentItem));
        currentBorrower.getLoans().add(new Loan(LocalDate.now().minusDays(2), null, currentBorrower, new Item()));
        currentBorrower.getLoans().add(new Loan(LocalDate.now().minusDays(2), null, currentBorrower, new Item()));
    }

    @Given("George Red has been assigned a maximum lending limit of 3")
    public void george_Red_max_limit_
ChatGPT
3() {
        BorrowerCategory category = new BorrowerCategory();
        category.setMaxLendingItems(3);
        currentBorrower.setCategory(category);
    }

    @When("George Red tries to borrow the item Harry Potter")
    public void george_Red_tries_to_borrow_item_harry_potter() {
        try {
            loanService.loanItem(currentBorrower, currentItem);
        } catch (Exception e) {
            exception = e;
        }
    }

    @Then("the system doesn't allow the loan")
    public void system_does_not_allow_loan() {
        assertNotNull(exception);
        assertEquals("Lending limit reached", exception.getMessage());
    }

    @Then("George Red's pending items remain 3")
    public void george_Red_pending_items_remain_3() {
        assertEquals(3, currentBorrower.getLoans().size());
    }
}
```

Step Definitions for Feature: Returning a Borrowed Item to the Library

Step Definitions

```java
import cucumber.api.java.en.*;
import java.time.LocalDate;
import static org.junit.Assert.*;


public class ReturningBorrowedItemsSteps {
    private BorrowerService borrowerService;
    private ItemService itemService;
    private LoanService loanService;
    private Borrower currentBorrower;
    private Item currentItem;
    private Loan currentLoan;
    private Exception exception;

    public ReturningBorrowedItemsSteps() {
        // Assuming the necessary DAOs are properly implemented
        BorrowerDAO borrowerDAO = new InMemoryBorrowerDAO();
        ItemDAO itemDAO = new InMemoryItemDAO();
        LoanDAO loanDAO = new InMemoryLoanDAO();
        borrowerService = new BorrowerService(borrowerDAO);
        itemService = new ItemService(itemDAO);
        loanService = new LoanService(loanDAO);
        exception = null;
    }

    @Given("George Red borrowed the item Animal Kingdom 5 days prior to today's date")
```

```
public void george_Red_borrowed_animal_kingdom() {

    currentBorrower = new Borrower();

    currentBorrower.setBorrowerNo(1);

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

    currentItem = new Item();

    currentItem.setItemNumber(1);

    currentItem.setBook(new Book("Animal Kingdom", "Author Name", 2020));

    currentItem.setState(ItemState.BORROWED);

    currentLoan = new Loan();

    currentLoan.setLoanDate(LocalDate.now().minusDays(5));

    currentLoan.setBorrower(currentBorrower);

    currentLoan.setItem(currentItem);

    loanService.save(currentLoan);

}


@Given("George Red has been assigned maximum lending days of 10")

public void george_Red_max_lending_days_10() {

    BorrowerCategory category = new BorrowerCategory();

    category.setMaxLendingDays(10);

    currentBorrower.setCategory(category);

}


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

```
public void return_of_animal_kingdom_processed() {

    try {

        loanService.returnItem(currentBorrower, currentItem);

    } catch (Exception e) {

        exception = e;

    }

}



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

public void system_marks_animal_kingdom_available() {

    assertEquals(ItemState.AVAILABLE, currentItem.getState());

}



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

public void george_Red_one_less_pending_item() {

    assertEquals(0, currentBorrower.getLoans().size());

}



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

public void george_Red_no_fine() {

    assertNull(exception);

}
```

```
    }


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

    public void return_date_set_to_today() {

        assertEquals(LocalDate.now(), currentLoan.getReturnDate());

    }


    @Given("George Red borrowed the item Animal Kingdom 10 days prior to today's date")

    public void george_Red_borrowed_animal_kingdom_10_days_ago() {

        currentLoan.setLoanDate(LocalDate.now().minusDays(10));

    }


    @Given("George Red has been assigned maximum lending days of 7")

    public void george_Red_max_lending_days_7() {

        BorrowerCategory category = new BorrowerCategory();

        category.setMaxLendingDays(7);

        currentBorrower.setCategory(category);

    }


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

    public void george_Red_pays_fine() {

        assertNotNull(exception);

        assertEquals("Late return fine", exception.getMessage());

    }

}
```

~~~

In-Memory DAO Implementations for Testing

```java
public class InMemoryItemDAO implements ItemDAO {
  private Map<Integer, Item> database = new HashMap<>();

  @Override
  public Item findByItemNumber(int itemNumber) {
    return database.get(itemNumber);
  }

  @Override
  public void save(Item item) {
    database.put(item.getItemNumber(), item);
  }

  @Override
  public void update(Item item) {
    database.put(item.getItemNumber(), item);
  }
~~~

```
    @Override

    public void delete(Item item) {

        database.remove(item.getItemNumber());

    }

}


public class InMemoryLoanDAO implements LoanDAO {

    private Map<Integer, Loan> database = new HashMap<>();


    @Override

    public Loan findByLoanId(int loanId) {

        return database.get(loanId);

    }


    @Override

    public void save(Loan loan) {

        database.put(loan.getItem().getItemNumber(), loan);

    }


    @Override

    public void update(Loan loan) {

        database.put(loan.getItem().getItemNumber(), loan);

    }
```

```java
    @Override
    public void delete(Loan loan) {
        database.remove(loan.getItem().getItemNumber());
    }
}
```

Services for Business Logic

```java
public class ItemService {
    private ItemDAO itemDAO;

    public ItemService(ItemDAO itemDAO) {
        this.itemDAO = itemDAO;
    }

    public void save(Item item) {
        itemDAO.save(item);
    }
}
```

```
public class LoanService {

    private LoanDAO loanDAO;



    public LoanService(LoanDAO loanDAO) {

        this.loanDAO = loanDAO;

    }



    public void save(Loan loan) {

        loanDAO.save(loan);

    }



    public void loanItem(Borrower borrower, Item item) throws Exception {

        if (item == null) {

            throw new Exception("Item not found");

        }

        if (borrower.getLoans().size() >= borrower.getCategory().getMaxLendingItems()) {

            throw new Exception("Lending limit reached");

        }

        item.setState(ItemState.BORROWED);

        Loan loan = new Loan();

        loan.setBorrower(borrower);

        loan.setItem(item);

        loanDAO.save(loan);
    }

    public void returnItem(Borrower borrower, Item item) throws Exception {
        Loan loan = loanDAO.findByLoanId(item.getItemNumber());
        if (loan == null) {
            throw new Exception("Loan not found");
        }
        LocalDate today = LocalDate.now();
        int daysBorrowed = (int) java.time.temporal.ChronoUnit.DAYS.between(loan.getLoanDate(), today);
        if (daysBorrowed > borrower.getCategory().getMaxLendingDays()) {
            throw new Exception("Late return fine");
        }
        loan.setReturnDate(today);
        item.setState(ItemState.AVAILABLE);
        loanDAO.update(loan);
    }
}

public class EmailService {
    public void sendDelayedReturnNotification(Loan loan) throws Exception {
        if (loan.getBorrower().getEmailAddress() == null) {
            throw new Exception("Email address not found");
        }
        // Simulate sending an email

        System.out.println("Email sent to: " + loan.getBorrower().getEmailAddress().getEmail());

    }

}
```

Summary

This implementation includes the complete step definitions for all the provided features and scenarios, using in-memory DAO implementations for testing purposes. The service classes encapsulate the business logic, and the step definitions use these services to fulfill the scenarios. Adjust the implementation details to fit your specific requirements and environment.