

Στη συνομιλία αυτή, χρησιμοποιήθηκε η προσέγγιση της παροχής των features ένα προς ένα, όπως έγινε με τα άλλα LLM. Αυτή η τεχνική στόχευε στο να δούμε αν το LLM θα επικεντρωνόταν περισσότερο σε κάθε μεμονωμένο feature και θα βελτιώνε την ποιότητα του κώδικα του. Συνολικά, τα αποτελέσματα ήταν σχεδόν τα ίδια με αυτά που είχατε πάρει πριν ή, σε κάποιες περιπτώσεις, ήταν ακόμα χειρότερα. Το LLM φάνηκε να κατανοεί γρήγορα τα features και να δημιουργεί τις απαντήσεις αμέσως, αλλά η προσέγγιση της παροχής των features ένα προς ένα δεν είχε θετική επίδραση στην ποιότητα του κώδικα που παρήχθη. Η παροχή όλων των πληροφοριών από την αρχή θα μπορούσε να είχε βελτιώσει την ποιότητα του κώδικα, δεδομένου ότι το LLM θα είχε πλήρη εικόνα του συστήματος από την αρχή. Η τεχνική της παροχής features ένα προς ένα δεν έφερε την επιθυμητή βελτίωση και συνεπώς δεν θα χρησιμοποιηθεί σε μελλοντικές συνομιλίες, καθώς δεν είχε θετική επίδραση στην παραγωγή καλύτερου κώδικα.

Phase 1 Chat 3

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 one at a time 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

That sounds great! I'm ready to help you with the implementation of the Step definitions for your library system application in Java. Please provide the first feature/scenario, and I'll guide you through the architecture with Domain classes, Data Access Objects (DAOs), and Service classes, along with the implementation of the Step definitions.

User

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

ChatGPT

Sure, let's start with the implementation of the Step definitions for the "Borrower handling by the system" feature. Here's the breakdown:

1. ****Domain Classes****
2. ****Data Access Objects (DAOs)****
3. ****Service Classes****
4. ****Step Definitions****

1. Domain Classes

Borrower.java

```
```java
```

```
public class Borrower {

 private String name;

 private String borrowerNumber;

 private String details;

 private boolean hasPendingItems;

 // Constructors, getters, and setters
```

```

public Borrower(String name, String borrowerNumber, String details) {

 this.name = name;

 this.borrowerNumber = borrowerNumber;

 this.details = details;

 this.hasPendingItems = false;

}

public String getName() { return name; }

public void setName(String name) { this.name = name; }

public String getBorrowerNumber() { return borrowerNumber; }

public void setBorrowerNumber(String borrowerNumber) { this.borrowerNumber =
borrowerNumber; }

public String getDetails() { return details; }

public void setDetails(String details) { this.details = details; }

public boolean hasPendingItems() { return hasPendingItems; }

public void setPendingItems(boolean hasPendingItems) { this.hasPendingItems =
hasPendingItems; }

}

...

```

## ### 2. Data Access Objects (DAOs)

#### BorrowerDAO.java

```
```java
```

```
import java.util.HashMap;
```

```
import java.util.Map;
```

```
public class BorrowerDAO {
```

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

```
    public void save(Borrower borrower) {
```

```
        borrowers.put(borrower.getBorrowerNumber(), borrower);
```

```
    }
```

```
    public Borrower findByBorrowerNumber(String borrowerNumber) {
```

```
        return borrowers.get(borrowerNumber);
```

```
    }
```

```
    public void deleteByBorrowerNumber(String borrowerNumber) {
```

```
        borrowers.remove(borrowerNumber);
```

```
    }
```

```
    public boolean exists(String borrowerNumber) {
```

```
        return borrowers.containsKey(borrowerNumber);
```

```
    }
```

```
}
```

```
...
```

3. Service Classes

BorrowerService.java

```
```java
```

```
public class BorrowerService {
```

```
 private BorrowerDAO borrowerDAO;
```

```
 public BorrowerService(BorrowerDAO borrowerDAO) {
```

```
 this.borrowerDAO = borrowerDAO;
```

```
 }
```

```
 public String registerBorrower(Borrower borrower) {
```

```
 if (borrowerDAO.exists(borrower.getBorrowerNumber())) {
```

```
 return "User already exists";
```

```
 }
```

```
 borrowerDAO.save(borrower);
```

```
 return "Borrower registered successfully";
```

```
 }
```

```
 public String updateBorrowerDetails(Borrower borrower) {
```

```
 if (!borrowerDAO.exists(borrower.getBorrowerNumber())) {
```

```
 return "Borrower does not exist";
```

```

 }

 borrowerDAO.save(borrower);

 return "Borrower details updated successfully";
}

public String deleteBorrower(String borrowerNumber) {

 Borrower borrower = borrowerDAO.findByBorrowerNumber(borrowerNumber);

 if (borrower == null) {

 return "Borrower does not exist";

 }

 if (borrower.hasPendingItems()) {

 return "Cannot delete borrower with pending items";

 }

 borrowerDAO.deleteByBorrowerNumber(borrowerNumber);

 return "Borrower deleted successfully";

}

}

```

```

4. Step Definitions

BorrowerSteps.java

```
```java
```



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

```
import io.cucumber.java.en.*;
```

```
import io.cucumber.java.Before;
```

```
public class BorrowerSteps {
```

```
 private BorrowerDAO borrowerDAO;
```

```
 private BorrowerService borrowerService;
```

```
 private String responseMessage;
```

```
 @Before
```

```
 public void setUp() {
```

```
 borrowerDAO = new BorrowerDAO();
```

```
 borrowerService = new BorrowerService(borrowerDAO);
```

```
 }
```

```
 @Given("{string} is not registered as a borrower")
```

```
 public void is_not_registered_as_a_borrower(String name) {
```

```
 assertFalse(borrowerDAO.exists(name));
```

```
 }
```

```
 @Given("{string} is registered as a borrower")
```

```
 public void is_registered_as_a_borrower(String name) {
```

```

 Borrower borrower = new Borrower(name, "unique-borrower-number", "details");

 borrowerDAO.save(borrower);

 assertTrue(borrowerDAO.exists("unique-borrower-number"));
 }

```

@When("{string} gets registered in the system with a unique borrower number and his details")

```

 public void
 gets_registered_in_the_system_with_a_unique_borrower_number_and_his_details(String
 name) {

```

```

 Borrower borrower = new Borrower(name, "unique-borrower-number", "details");

 responseMessage = borrowerService.registerBorrower(borrower);
 }

```

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

```

 public void the_system_successfully_stores_the_borrower_s_details() {

 assertEquals("Borrower registered successfully", responseMessage);
 }

```

@When("the system attempts to register {string} with a unique borrower number and his details")

```

 public void
 the_system_attempts_to_register_with_a_unique_borrower_number_and_his_details(Strin
 g name) {

```

```

 Borrower borrower = new Borrower(name, "unique-borrower-number", "details");

```

```
 responseMessage = borrowerService.registerBorrower(borrower);
}
```

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

public void the_system_informs_that_the_user_already_exists() {
 assertEquals("User already exists", responseMessage);
}
```

```
@When("{string} updates his borrowing details")

public void updates_his_borrowing_details(String name) {
 Borrower borrower = new Borrower(name, "unique-borrower-number", "updated
details");
 responseMessage = borrowerService.updateBorrowerDetails(borrower);
}
```

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

public void the_system_saves_the_changes() {
 assertEquals("Borrower details updated successfully", responseMessage);
}
```

```
@When("{string} tries to update his borrowing details")

public void tries_to_update_his_borrowing_details(String name) {
 Borrower borrower = new Borrower(name, "unique-borrower-number", "updated
details");
```

```
 responseMessage = borrowerService.updateBorrowerDetails(borrower);
 }


```

```
@Then("the system displays an error message indicating that {string} does not exist")
```

```
public void the_system_displays_an_error_message_indicating_that_does_not_exist(String name) {
 assertEquals("Borrower does not exist", responseMessage);
}


```

```
@When("the system deletes {string}'s account")
```

```
public void the_system_deletes_s_account(String name) {
 responseMessage = borrowerService.deleteBorrower("unique-borrower-number");
}


```

```
@Then("the system removes {string}'s details")
```

```
public void the_system_removes_s_details(String name) {
 assertEquals("Borrower deleted successfully", responseMessage);
}


```

```
@When("the system attempts to delete {string}'s account")
```

```
public void the_system_attempts_to_delete_s_account(String name) {
 responseMessage = borrowerService.deleteBorrower("unique-borrower-number");
}


```

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

```
public void the_system_informs_that_the_borrower_does_not_exist() {
```

```
 assertEquals("Borrower does not exist", responseMessage);
```

```
}
```

```
@Given("{string} has pending items")
```

```
public void has_pending_items(String name) {
```

```
 Borrower borrower = borrowerDAO.findByBorrowerNumber("unique-borrower-number");
```

```
 if (borrower != null) {
```

```
 borrower.setPendingItems(true);
```

```
 borrowerDAO.save(borrower);
```

```
 }
```

```
}
```

```
@Then("the system does not remove {string}'s details")
```

```
public void the_system_does_not_remove_s_details(String name) {
```

```
 assertTrue(borrowerDAO.exists("unique-borrower-number"));
```

```
}
```

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

```
public void the_system_informs_about_the_pending_items() {
```

```
 assertEquals("Cannot delete borrower with pending items", responseMessage);
```

```
}
```

```
}
...
```

This setup covers the required feature scenarios. Make sure to include the necessary imports and ensure your project is configured to use JUnit and Cucumber for testing. If you need further details or modifications, let me know!

## **User**

**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**

ChatGPT

Let's implement the "Delayed Return Notification" feature. This will involve adding functionality to check for overdue items and send notifications via email if the borrower has a registered email address. We'll follow the same structure as before with Domain classes, DAOs, Service classes, and Step Definitions.

### 1. Domain Classes

#### Borrower.java

We'll extend the `Borrower` class to include an email address.

```
```java
```

```
public class Borrower {  
  
    private String name;  
  
    private String borrowerNumber;  
  
    private String details;  
  
    private String email;  
  
    private boolean hasPendingItems;  
}
```

```
// Constructors, getters, and setters
```

```
public Borrower(String name, String borrowerNumber, String details, String email) {
```

```
    this.name = name;
```

```
    this.borrowerNumber = borrowerNumber;
```

```
    this.details = details;
```

```
    this.email = email;
```

```
    this.hasPendingItems = false;
```

```
}
```

```
public String getName() { return name; }
```

```
public void setName(String name) { this.name = name; }
```

```
public String getBorrowerNumber() { return borrowerNumber; }
```

```
public void setBorrowerNumber(String borrowerNumber) { this.borrowerNumber =  
borrowerNumber; }
```

```
public String getDetails() { return details; }
```

```
public void setDetails(String details) { this.details = details; }
```

```
public String getEmail() { return email; }
```

```
public void setEmail(String email) { this.email = email; }
```

```
public boolean hasPendingItems() { return hasPendingItems; }
```

```
public void setPendingItems(boolean hasPendingItems) { this.hasPendingItems =  
hasPendingItems; }
```

```
}
```

```
...
```


Item.java

We'll add an `Item` class to represent borrowed items.

```
```java
```

```
import java.time.LocalDate;
```

```
public class Item {
```

```
 private String title;
```

```
 private LocalDate dueDate;
```

```
 // Constructors, getters, and setters
```

```
 public Item(String title, LocalDate dueDate) {
```

```
 this.title = title;
```

```
 this.dueDate = dueDate;
```

```
 }
```

```
 public String getTitle() { return title; }
```

```
 public void setTitle(String title) { this.title = title; }
```

```
 public LocalDate getDueDate() { return dueDate; }
```

```
 public void setDueDate(LocalDate dueDate) { this.dueDate = dueDate; }
```

```
}
```

```
```
```

2. Data Access Objects (DAOs)

BorrowerDAO.java

We add methods to handle items borrowed by the borrower.

```
```java
```

```
import java.util.HashMap;
```

```
import java.util.Map;
```

```
public class BorrowerDAO {
```

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

```
 public void save(Borrower borrower) {
```

```
 borrowers.put(borrower.getBorrowerNumber(), borrower);
```

```
 }
```

```
 public Borrower findByBorrowerNumber(String borrowerNumber) {
```

```
 return borrowers.get(borrowerNumber);
```

```
 }
```

```
 public void deleteByBorrowerNumber(String borrowerNumber) {
```

```
 borrowers.remove(borrowerNumber);
```

```
 }
```

```

public boolean exists(String borrowerNumber) {

 return borrowers.containsKey(borrowerNumber);

}

public void addItemToBorrower(String borrowerNumber, Item item) {

 Borrower borrower = borrowers.get(borrowerNumber);

 if (borrower != null) {

 borrower.setPendingItems(true); // Simplified for the sake of example

 }

}

}

'''

```

### ### 3. Service Classes

#### #### EmailService.java

We'll need a service to handle email notifications.

```
```java
```

```

public class EmailService {

    public void sendEmail(String to, String subject, String body) {

        // Simulated email sending logic

        System.out.println("Email sent to " + to + " with subject: " + subject);

    }

}

```

```
}
```

```
...
```

NotificationService.java

This service handles the notification logic.

```
```java
```

```
import java.time.LocalDate;
```

```
public class NotificationService {
```

```
 private BorrowerDAO borrowerDAO;
```

```
 private EmailService emailService;
```

```
 public NotificationService(BorrowerDAO borrowerDAO, EmailService emailService) {
```

```
 this.borrowerDAO = borrowerDAO;
```

```
 this.emailService = emailService;
```

```
 }
```

```
 public void processDelayedReturnNotifications() {
```

```
 for (Borrower borrower : borrowerDAO.borrowers.values()) {
```

```
 if (borrower.hasPendingItems() && borrower.getEmail() != null) {
```

```
 String subject = "Return Overdue Item";
```

```
 String body = "Dear " + borrower.getName() + ",\nPlease return your overdue item.";
```

```
 emailService.sendEmail(borrower.getEmail(), subject, body);
```

```
 }
 }
 }
}
...
```

#### ### 4. Step Definitions

##### #### NotificationSteps.java

We'll implement the Cucumber step definitions.

```
```java
```

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

```
import io.cucumber.java.en.*;
```

```
import io.cucumber.java.Before;
```

```
import java.time.LocalDate;
```

```
public class NotificationSteps {
```

```
    private BorrowerDAO borrowerDAO;
```

```
    private EmailService emailService;
```

```
    private NotificationService notificationService;
```

```
    private Borrower currentBorrower;
```

```
private String itemTitle;
```

```
private boolean emailSent;
```

```
@Before
```

```
public void setUp() {
```

```
    borrowerDAO = new BorrowerDAO();
```

```
    emailService = new EmailService() {
```

```
        @Override
```

```
        public void sendEmail(String to, String subject, String body) {
```

```
            emailSent = true;
```

```
            super.sendEmail(to, subject, body);
```

```
        }
```

```
    };
```

```
    notificationService = new NotificationService(borrowerDAO, emailService);
```

```
    emailSent = false;
```

```
}
```

```
@Given("{string} has borrowed the item {string}")
```

```
public void has_borrowed_the_item(String borrowerName, String itemTitle) {
```

```
    this.itemTitle = itemTitle;
```

```
    currentBorrower = new Borrower(borrowerName, "unique-borrower-number", "details",  
    "george.red@example.com");
```

```
    borrowerDAO.save(currentBorrower);
```

```
        borrowerDAO.addItemToBorrower("unique-borrower-number", new Item(itemTitle,
LocalDate.now().minusDays(1)));

    }
```

```
@Given("{string}'s due date has passed")

public void s_due_date_has_passed(String itemTitle) {

    // No additional implementation needed as the due date is already set in the previous
step

}
```

```
@Given("{string} has an email address")

public void has_an_email_address(String borrowerName) {

    currentBorrower.setEmail("george.red@example.com");

    borrowerDAO.save(currentBorrower);

}
```

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

public void the_system_executes_the_delayed_return_notification_process() {

    notificationService.processDelayedReturnNotifications();

}
```

```
@Then("{string} receives an email notification for the return of the item")

public void receives_an_email_notification_for_the_return_of_the_item(String
borrowerName) {
```

```

        assertTrue(emailSent);
    }

    @Given("{string} does not have an email address")

    public void does_not_have_an_email_address(String borrowerName) {

        currentBorrower.setEmail(null);

        borrowerDAO.save(currentBorrower);
    }

    @Then("{string} does not receive an email notification for the return of the item")

    public void does_not_receive_an_email_notification_for_the_return_of_the_item(String
borrowerName) {

        assertFalse(emailSent);
    }
}

'''

```

This setup covers the required feature scenarios. Ensure your project is configured to use JUnit and Cucumber for testing. If you need further details or modifications, let me know!

User

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

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

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

ChatGPT

Let's implement the "Loaning items" feature. We'll build upon the existing structure with Domain classes, DAOs, Service classes, and Step Definitions. Here's how to handle the scenarios for loaning items to borrowers, considering various conditions like lending limits and item availability.

1. Domain Classes

Item.java

```
```java
```

```
import java.time.LocalDate;
```

```
public class Item {
```

```
 private String title;
```

```
 private boolean available;
```

```
 private LocalDate dueDate;
```

```
 public Item(String title) {
```

```
 this.title = title;
```

```
 this.available = true;
```

```
 }
```

```
 public String getTitle() { return title; }
```

```
 public boolean isAvailable() { return available; }
```

```
 public void setAvailable(boolean available) { this.available = available; }
```

```
 public LocalDate getDueDate() { return dueDate; }

 public void setDueDate(LocalDate dueDate) { this.dueDate = dueDate; }

}

...

```

#### Borrower.java

We'll extend the `Borrower` class to include the borrowing limit and pending items.

```
```java

```

```
import java.util.List;

```

```
public class Borrower {

```

```
    private String name;

```

```
    private String borrowerNumber;

```

```
    private String details;

```

```
    private String email;

```

```
    private int pendingItems;

```

```
    private int maxLendingLimit;

```

```
    public Borrower(String name, String borrowerNumber, String details, String email, int
maxLendingLimit) {

```

```
        this.name = name;

```

```
        this.borrowerNumber = borrowerNumber;

```

```
        this.details = details;

```

```
        this.email = email;

```

```
        this.maxLendingLimit = maxLendingLimit;

        this.pendingItems = 0;
    }

    public String getName() { return name; }

    public String getBorrowerNumber() { return borrowerNumber; }

    public String getDetails() { return details; }

    public String getEmail() { return email; }

    public int getPendingItems() { return pendingItems; }

    public int getMaxLendingLimit() { return maxLendingLimit; }

    public void incrementPendingItems() {

        this.pendingItems++;

    }

    public void decrementPendingItems() {

        if (pendingItems > 0) {

            this.pendingItems--;

        }

    }

}

'''
```

2. Data Access Objects (DAOs)

ItemDAO.java

```
```java
```

```
import java.util.HashMap;
```

```
import java.util.Map;
```

```
public class ItemDAO {
```

```
 private Map<String, Item> items = new HashMap<>();
```

```
 public void save(Item item) {
```

```
 items.put(item.getTitle(), item);
```

```
 }
```

```
 public Item findByTitle(String title) {
```

```
 return items.get(title);
```

```
 }
```

```
 public boolean exists(String title) {
```

```
 return items.containsKey(title);
```

```
 }
```

```
}
```

```
```
```

BorrowerDAO.java

We'll keep the existing functionality and add methods to handle the borrower items.

```
```java
```

```
import java.util.HashMap;
```

```
import java.util.Map;
```

```
public class BorrowerDAO {
```

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

```
 public void save(Borrower borrower) {
```

```
 borrowers.put(borrower.getBorrowerNumber(), borrower);
```

```
 }
```

```
 public Borrower findByBorrowerNumber(String borrowerNumber) {
```

```
 return borrowers.get(borrowerNumber);
```

```
 }
```

```
 public boolean exists(String borrowerNumber) {
```

```
 return borrowers.containsKey(borrowerNumber);
```

```
 }
```

```
}
```

```
```
```

3. Service Classes

LoanService.java

This service handles the logic for loaning items.

```
```java
```

```
import java.time.LocalDate;
```

```
public class LoanService {
```

```
 private BorrowerDAO borrowerDAO;
```

```
 private ItemDAO itemDAO;
```

```
 public LoanService(BorrowerDAO borrowerDAO, ItemDAO itemDAO) {
```

```
 this.borrowerDAO = borrowerDAO;
```

```
 this.itemDAO = itemDAO;
```

```
 }
```

```
 public String loanItem(String borrowerNumber, String itemTitle) {
```

```
 Borrower borrower = borrowerDAO.findByBorrowerNumber(borrowerNumber);
```

```
 Item item = itemDAO.findByTitle(itemTitle);
```

```
 if (item == null || !item.isAvailable()) {
```

```
 return "Error: Item not found or unavailable.";
```



```

 }

 if (borrower == null) {

 return "Error: Borrower not found.";

 }

 if (borrower.getPendingItems() >= borrower.getMaxLendingLimit()) {

 return "Error: Borrower has reached the maximum lending limit.";

 }

 item.setAvailable(false);

 item.setDueDate(LocalDate.now().plusDays(14)); // Setting a due date of 14 days

 borrower.incrementPendingItems();

 return "Success: Item loaned to " + borrower.getName() + " with due date " +
item.getDueDate();

 }

}

...

```

#### ### 4. Step Definitions

##### #### LoanSteps.java

Implementing the Cucumber step definitions for the loaning scenarios.

```
```java
```

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

```
import io.cucumber.java.en.*;
```

```
import io.cucumber.java.Before;
```

```
import java.time.LocalDate;
```

```
public class LoanSteps {
```

```
    private BorrowerDAO borrowerDAO;
```

```
    private ItemDAO itemDAO;
```

```
    private LoanService loanService;
```

```
    private String responseMessage;
```

```
    private String itemTitle;
```

```
    @Before
```

```
    public void setUp() {
```

```
        borrowerDAO = new BorrowerDAO();
```

```
        itemDAO = new ItemDAO();
```

```
        loanService = new LoanService(borrowerDAO, itemDAO);
```

```
    }
```

```
    @Given("the library has the item {string} available")
```

```
public void the_library_has_the_item_available(String title) {  
  
    Item item = new Item(title);  
  
    itemDAO.save(item);  
  
}
```

```
@Given("{string} is a registered borrower")
```

```
public void is_a_registered_borrower(String name) {  
  
    Borrower borrower = new Borrower(name, "unique-borrower-number", "details",  
"george.red@example.com", 5);  
  
    borrowerDAO.save(borrower);  
  
}
```

```
@Given("{string} has {int} pending items to be returned")
```

```
public void has_pending_items_to_be_returned(String name, int pendingItems) {  
  
    Borrower borrower = borrowerDAO.findByBorrowerNumber("unique-borrower-  
number");  
  
    for (int i = 0; i < pendingItems; i++) {  
  
        borrower.incrementPendingItems();  
  
    }  
  
    borrowerDAO.save(borrower);  
  
}
```

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

```
public void has_been_assigned_a_maximum_lending_limit_of(String name, int limit) {
```

```

        Borrower borrower = borrowerDAO.findByBorrowerNumber("unique-borrower-
number");

        borrower = new Borrower(borrower.getName(), borrower.getBorrowerNumber(),
borrower.getDetails(), borrower.getEmail(), limit);

        borrowerDAO.save(borrower);
    }

```

```

@When("{string} borrows the item {string}")

```

```

public void borrows_the_item(String name, String title) {

```

```

    responseMessage = loanService loanItem("unique-borrower-number", title);

```

```

    itemTitle = title;

```

```

}

```

```

@Then("the system successfully loans the item {string} to {string} with a due date set")

```

```

public void the_system_successfully_loans_the_item_to_with_a_due_date_set(String
itemTitle, String name) {

```

```

    Item item = itemDAO.findByTitle(itemTitle);

```

```

    Borrower borrower = borrowerDAO.findByBorrowerNumber("unique-borrower-
number");

```

```

    assertFalse(item.isAvailable());

```

```

    assertEquals(LocalDate.now().plusDays(14), item.getDueDate());

```

```

    assertEquals(3, borrower.getPendingItems());

```

```

    assertEquals("Success: Item loaned to " + name + " with due date " + item.getDueDate(),
responseMessage);

```

```

}

```

```

    @Then("the system does not loan {string} to {string} due to the lending limit reached")

    public void the_system_does_not_loan_to_due_to_the_lending_limit_reached(String
itemTitle, String name) {

        assertEquals("Error: Borrower has reached the maximum lending limit.",
responseMessage);

        Borrower borrower = borrowerDAO.findByBorrowerNumber("unique-borrower-
number");

        assertEquals(3, borrower.getPendingItems());

    }

```

```

    @Given("the item {string} is in the library but not in the system")

    public void the_item_is_in_the_library_but_not_in_the_system(String title) {

        // This item exists in the library but not in the DAO

        // No need to do anything here as it's handled in the service logic

    }

```

```

    @When("{string} tries to borrow the item {string}")

    public void tries_to_borrow_the_item(String name, String title) {

        responseMessage = loanService loanItem("unique-borrower-number", title);

    }

```

```

    @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() {

```

```

        assertEquals("Error: Item not found or unavailable.", responseMessage);
    }

    @Given("{string} is a registered borrower with {int} pending items to be returned")

    public void is_a_registered_borrower_with_pending_items_to_be_returned(String name,
int pendingItems) {

        Borrower borrower = new Borrower(name, "unique-borrower-number", "details",
"george.red@example.com", 3);

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

            borrower.incrementPendingItems();

        }

        borrowerDAO.save(borrower);

    }

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

    public void s_pending_items_remain(String name, int pendingItems) {

        Borrower borrower = borrowerDAO.findByBorrowerNumber("unique-borrower-
number");

        assertEquals(pendingItems, borrower.getPendingItems());

    }

}

...

```

Summary

1. ****Domain Classes****: `Item`, `

User

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

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

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

Let's implement the "Returning a borrowed item to the library" feature. We will add necessary classes and methods to handle the return of items, calculate fines if the item is returned late, and update the system accordingly.

1. Domain Classes

Item.java

We'll add a method to set the item as available and a method to set the return date.

```
```java
```

```
import java.time.LocalDate;
```

```
public class Item {
```

```
 private String title;
```

```
 private boolean available;
```



```

private LocalDate dueDate;

private LocalDate returnDate;

public Item(String title) {

 this.title = title;

 this.available = true;

}

public String getTitle() { return title; }

public boolean isAvailable() { return available; }

public void setAvailable(boolean available) { this.available = available; }

public LocalDate getDueDate() { return dueDate; }

public void setDueDate(LocalDate dueDate) { this.dueDate = dueDate; }

public LocalDate getReturnDate() { return returnDate; }

public void setReturnDate(LocalDate returnDate) { this.returnDate = returnDate; }

}
...

```

#### #### Borrower.java

We need to add methods to handle fines and decrement pending items.

```
```java
```

```

public class Borrower {

    private String name;

```

```
private String borrowerNumber;
```

```
private String details;
```

```
private String email;
```

```
private int pendingItems;
```

```
private int maxLendingLimit;
```

```
private int maxLendingDays;
```

```
public Borrower(String name, String borrowerNumber, String details, String email, int  
maxLendingLimit, int maxLendingDays) {
```

```
    this.name = name;
```

```
    this.borrowerNumber = borrowerNumber;
```

```
    this.details = details;
```

```
    this.email = email;
```

```
    this.maxLendingLimit = maxLendingLimit;
```

```
    this.maxLendingDays = maxLendingDays;
```

```
    this.pendingItems = 0;
```

```
}
```

```
public String getName() { return name; }
```

```
public String getBorrowerNumber() { return borrowerNumber; }
```

```
public String getDetails() { return details; }
```

```
public String getEmail() { return email; }
```

```
public int getPendingItems() { return pendingItems; }
```

```
public int getMaxLendingLimit() { return maxLendingLimit; }
```

```
public int getMaxLendingDays() { return maxLendingDays; }
```

```
public void incrementPendingItems() {
```

```
    this.pendingItems++;
```

```
}
```

```
public void decrementPendingItems() {
```

```
    if (pendingItems > 0) {
```

```
        this.pendingItems--;
```

```
    }
```

```
}
```

```
public int calculateFine(int daysLate) {
```

```
    return daysLate * 1; // Assume a fine of 1 unit per day late, can be adjusted as needed
```

```
}
```

```
}
```

```
...
```

2. Data Access Objects (DAOs)

ItemDAO.java

We'll keep the existing functionality.

```
```java
```

```
import java.util.HashMap;

import java.util.Map;

public class ItemDAO {

 private Map<String, Item> items = new HashMap<>();

 public void save(Item item) {

 items.put(item.getTitle(), item);

 }

 public Item findByTitle(String title) {

 return items.get(title);

 }

 public boolean exists(String title) {

 return items.containsKey(title);

 }

}

...

```

#### BorrowerDAO.java

We'll keep the existing functionality.

```
```java
```

```

import java.util.HashMap;

import java.util.Map;


public class BorrowerDAO {

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


    public void save(Borrower borrower) {

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

    }


    public Borrower findByBorrowerNumber(String borrowerNumber) {

        return borrowers.get(borrowerNumber);

    }


    public boolean exists(String borrowerNumber) {

        return borrowers.containsKey(borrowerNumber);

    }

}

'''

```

3. Service Classes

ReturnService.java

This service handles the logic for returning items.

```
```java
```

```
import java.time.LocalDate;
```

```
import java.time.temporal.ChronoUnit;
```

```
public class ReturnService {
```

```
 private BorrowerDAO borrowerDAO;
```

```
 private ItemDAO itemDAO;
```

```
 public ReturnService(BorrowerDAO borrowerDAO, ItemDAO itemDAO) {
```

```
 this.borrowerDAO = borrowerDAO;
```

```
 this.itemDAO = itemDAO;
```

```
 }
```

```
 public String processReturn(String borrowerNumber, String itemTitle) {
```

```
 Borrower borrower = borrowerDAO.findByBorrowerNumber(borrowerNumber);
```

```
 Item item = itemDAO.findByTitle(itemTitle);
```

```
 if (item == null || borrower == null) {
```

```
 return "Error: Borrower or item not found.";
```

```
 }
```

```
 LocalDate today = LocalDate.now();
```

```

 item.setAvailable(true);

 item.setReturnDate(today);

 long daysLate = ChronoUnit.DAYS.between(item.getDueDate(), today);

 if (daysLate > 0) {

 int fine = borrower.calculateFine((int) daysLate);

 return "Success: Item returned. Fine of " + fine + " units due.";

 } else {

 borrower.decrementPendingItems();

 return "Success: Item returned. No fine.";

 }

 }

}

'''

```

#### ### 4. Step Definitions

##### #### ReturnSteps.java

Implementing the Cucumber step definitions for the return scenarios.

```
```java
```

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

```
import io.cucumber.java.en.*;
```

```
import io.cucumber.java.Before;
```

```
import java.time.LocalDate;
```

```
public class ReturnSteps {
```

```
    private BorrowerDAO borrowerDAO;
```

```
    private ItemDAO itemDAO;
```

```
    private ReturnService returnService;
```

```
    private String responseMessage;
```

```
    @Before
```

```
    public void setUp() {
```

```
        borrowerDAO = new BorrowerDAO();
```

```
        itemDAO = new ItemDAO();
```

```
        returnService = new ReturnService(borrowerDAO, itemDAO);
```

```
    }
```

```
    @Given("{string} borrowed the item {string} {int} days prior to today's date")
```

```
    public void borrowed_the_item_days_prior_to_today_s_date(String name, String title, int daysAgo) {
```

```
        Borrower borrower = new Borrower(name, "unique-borrower-number", "details",  
"george.red@example.com", 5, 10);
```

```
        borrower.incrementPendingItems();
```

```
        borrowerDAO.save(borrower);
```



```

        Item item = new Item(title);

        item.setAvailable(false);

        item.setDueDate(LocalDate.now().minusDays(daysAgo).plusDays(borrower.getMaxLendingDays()));

        itemDAO.save(item);

    }

```

```

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

    public void has_been_assigned_maximum_lending_days_of(String name, int maxLendingDays) {

        Borrower borrower = borrowerDAO.findByBorrowerNumber("unique-borrower-number");

        borrower = new Borrower(borrower.getName(), borrower.getBorrowerNumber(), borrower.getDetails(), borrower.getEmail(), borrower.getMaxLendingLimit(), maxLendingDays);

        borrowerDAO.save(borrower);

    }

```

```

    @When("the return of {string} is processed")

    public void the_return_of_is_processed(String title) {

        responseMessage = returnService.processReturn("unique-borrower-number", title);

    }

```

```
@Then("the system marks the state of {string} as AVAILABLE")
```

```
public void the_system_marks_the_state_of_as_AVAILABLE(String title) {
```

```
    Item item = itemDAO.findByTitle(title);
```

```
    assertTrue(item.isAvailable());
```

```
}
```

```
@Then("{string} has one less pending item")
```

```
public void has_one_less_pending_item(String name) {
```

```
    Borrower borrower = borrowerDAO.findByBorrowerNumber("unique-borrower-number");
```

```
    assertEquals(0, borrower.getPendingItems());
```

```
}
```

```
@Then("{string} does not pay a fine")
```

```
public void does_not_pay_a_fine(String name) {
```

```
    assertEquals("Success: Item returned. No fine.", responseMessage);
```

```
}
```

```
@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() {
```

```
    Item item = itemDAO.findByTitle("Animal Kingdom");
```

```
    assertEquals(LocalDate.now(), item.getReturnDate());
```

```
}
```

```
@Then("{string} pays a fine based on the borrower category")

public void pays_a_fine_based_on_the_borrower_category(String name) {

    assertTrue(responseMessage.startsWith("Success: Item returned. Fine of "));

}

}

...

```

Summary

This setup covers the required feature scenarios for returning a borrowed item to the library. Ensure your project is configured to use JUnit and Cucumber for testing. If you need further details or modifications, let me know!