

Στην πρώτη συνομιλία με το GPT-4o, παρείχα τις ίδιες πληροφορίες που είχα δώσει σε άλλες συνομιλίες στη φάση 1 με διάφορα LLMs (features + σύντομη περιγραφή του συστήματος), αλλά χωρίς να δώσω επιπλέον οδηγίες για τη δημιουργία του κώδικα Domain ή άλλων στοιχείων, όπως έκανα σε άλλες περιπτώσεις. Τα αποτελέσματα ήταν αρκετά ικανοποιητικά για την ποσότητα των πληροφοριών που παρείχα. Το LLM κατανόησε άμεσα την εργασία και δημιούργησε το πρώτο feature με ευκολία. Ειδικότερα, το LLM δεν παρέλειψε κανένα σενάριο και ανέπτυξε ένα χαρακτηριστικό που δεν είχα παρατηρήσει σε προηγούμενες συνομιλίες: για κάθε απάντηση βασισμένη σε διαφορετικό feature, θυμόταν την αρχική εντολή που ανέφερε την αρχιτεκτονική (Domain, DAOs, Services) και προσαρμόστηκε αναλόγως. Δημιούργησε τον κώδικα για DAOs, Services και Domain, κάνοντας αναγκαίες αλλαγές με βάση τις νέες πληροφορίες. Επιπλέον, για πρώτη φορά, το LLM παρείχε και τις εξαρτήσεις για την προσθήκη του Cucumber στο σύστημα, κάτι που δεν είχε γίνει σε προηγούμενες συνομιλίες. Συνολικά, τα αποτελέσματα ήταν αρκετά καλά για μια πρώτη συνομιλία. Το LLM χρησιμοποίησε assertions στα βήματα GIVEN, κάτι που ίσως δεν είναι ιδανικό, αλλά κατανόησε γρήγορα τι απαιτείτο και δημιούργησε τον απαιτούμενο κώδικα με πολύ λίγες καθοδηγήσεις, χρησιμοποιώντας αποτελεσματικά τα DAOs, τις κλάσεις Domain και τα Services.

Phase 1. 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 then I will ask you to provide me the implementation code of the Step definitions . The architecture layers of the system are a domain layer containing the domain classes, the DAO layer containing the DAO classes and a service layer containing service classes that help with the logic of the system. In the next message , I will provide you with the features/scenarios and then I want you to implement the step definitions code for each step

ChatGPT

Sure, I'd be happy to help with that. Please go ahead and provide the Gherkin scenarios, and I'll assist you 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

The second feature is :

Feature: Delayed Return Notification

User Story: As a Library Manager

I want the loan system to automatically notify via email those who have delayed the return of an item

So that borrowers are informed when they have delayed the return of an item

Scenario: Notifying the borrower via email

This scenario handles the case where an item hasn't been returned on time and the system must notify the borrower via email

Given George Red has borrowed the item Harry Potter

And Harry Potter's due date has passed

And George Red has an email address

When the system executes the delayed return notification process

Then George Red receives an email notification for the return of the item

Scenario: Borrower does not have an email address

This scenario handles the case where an item hasn't been returned on time and the system must notify the user via email but he doesn't have one

Given George Red has borrowed the item Harry Potter

And Harry Potter's due date has passed

And George Red does not have an email address

When the system executes the delayed return notification process

Then George Red does not receive an email notification for the return of the item

The third feature is :

Feature: Loaning items

The library application allows for the librarian to loan an item of a book to a borrower based

on some conditions of the system

User story: As a library member

I want to be able to borrow items

So that I can study them at home

Scenario: Successful loaning of an item

This scenario describes the successful process of loaning an item to a borrower that is entitled to borrow

Given the library has the item Harry Potter available

And George Red is a registered borrower

And George Red has 2 pending items to be returned

And George Red has been assigned a maximum lending limit of 5

When George Red borrows the item Harry Potter

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

And George Red's pending items increase to 3

Scenario: Borrower can borrow only one item due to his lending limit

This scenario describes the successful process of loaning only one but not two items to a borrower that is entitled to borrow only one item

Given the library has the items Harry Potter and Moby Dick available

And George Red is a registered borrower

And George Red has 2 pending items to be returned

And George Red has been assigned a maximum lending limit of 3

When George Red tries to borrow both items

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

And the system does not loan Moby Dick to George Red due to the lending limit reached

And George Red's pending items increase to 3

Scenario: Item not found

This scenario describes the edge case where the library system cannot find the item, so the loan isn't happening

Given the item Harry Potter is in the library but not in the system

And George Red is a registered borrower

When George Red tries to borrow the item Harry Potter

Then the system returns an error due to the item's status

And the system withdraws the item Harry Potter

Scenario: The borrower is not eligible to borrow

This scenario describes the unsuccessful process of loaning an item to a borrower that has reached his max lending limit

Given the library has the item Harry Potter available

And George Red is a registered borrower

And George Red has 3 pending items to be returned

And George Red has been assigned a maximum lending limit of 3

When George Red tries to borrow the item Harry Potter

Then the system doesn't allow the loan

And George Red's pending items remain 3

and the last feature is :

Feature: Returning a borrowed item to the library

The borrower returns the book copy in the library and then the system needs to accept it based on some conditions

User story: As a librarian

I want to accurately record the return of one or more books

So that books are returned on time, and late returns are fined

Scenario: Successful return of an item

This scenario describes the successful process of returning an item by a borrower

Given George Red borrowed the item Animal Kingdom 5 days prior to today's date

And George Red has been assigned maximum lending days of 10

When the return of Animal Kingdom is processed

Then the system marks the state of Animal Kingdom as AVAILABLE

And George Red has one less pending item

And George Red does not pay a fine

And the return date of the loan is set to today's date

Scenario: Late return of an item from a borrower

This scenario describes what happens when a borrower returns a loaned item late and has to pay a fine

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

And George Red has been assigned maximum lending days of 7

When the return of Animal Kingdom is processed

Then the system marks the state of Animal Kingdom as AVAILABLE

And George Red pays a fine based on the borrower category

And the return date of the loan is set to today's date

ChatGPT

Great! I'll provide the implementation code for the step definitions of each scenario using Java and Cucumber. Here is the step definition code for the "Borrower handling by the system" feature.

First, ensure you have the necessary dependencies in your `pom.xml` for Cucumber:


```
``xml
```

```
<dependency>
```

```
  <groupId>io.cucumber</groupId>
```

```
  <artifactId>cucumber-java</artifactId>
```

```
  <version>6.10.2</version>
```

```
  <scope>test</scope>
```

```
</dependency>
```

```
<dependency>
```

```
  <groupId>io.cucumber</groupId>
```

```
  <artifactId>cucumber-junit</artifactId>
```

```
  <version>6.10.2</version>
```

```
  <scope>test</scope>
```

```
</dependency>
```

```
``
```

```
### Borrower Handling Step Definitions
```

```
``java
```

```
package com.library.steps;
```

```
import com.library.domain.Borrower;
```

```
import com.library.service.BorrowerService;
```

```
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 BorrowerSteps {
```

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

```
    private Borrower borrower;
```

```
    private String registrationResult;
```

```
    private String updateResult;
```

```
    private String deletionResult;
```

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

```
    public void georgeRedIsNotRegisteredAsABorrower() {
```

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

```
        assertNull(borrower);
```

```
    }
```

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

```
    public void georgeRedGetsRegisteredInTheSystem() {
```

```
        borrower = new Borrower("George Red", "unique_borrower_number", "details");
```

```
        registrationResult = borrowerService.registerBorrower(borrower);
```

```
}
```

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

```
public void theSystemSuccessfullyStoresTheBorrowersDetails() {
```

```
    assertEquals("Success", registrationResult);
```

```
    assertNotNull(borrowerService.findBorrowerByName("George Red"));
```

```
}
```

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

```
public void georgeRedIsRegisteredAsABorrower() {
```

```
    borrower = new Borrower("George Red", "unique_borrower_number", "details");
```

```
    borrowerService.registerBorrower(borrower);
```

```
    assertNotNull(borrowerService.findBorrowerByName("George Red"));
```

```
}
```

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

```
public void theSystemAttemptsToRegisterGeorgeRed() {
```

```
    registrationResult = borrowerService.registerBorrower(borrower);
```

```
}
```

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

```
public void theSystemInformsThatTheUserAlreadyExists() {
```

```
    assertEquals("User already exists", registrationResult);
```

```
}
```

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

```
public void georgeRedUpdatesHisBorrowingDetails() {
```

```
    borrower.setDetails("new details");
```

```
    updateResult = borrowerService.updateBorrower(borrower);
```

```
}
```

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

```
public void theSystemSavesTheChanges() {
```

```
    assertEquals("Update success", updateResult);
```

```
    assertEquals("new details", borrowerService.findBorrowerByName("George Red").getDetails());
```

```
}
```

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

```
public void georgeRedTriesToUpdateHisBorrowingDetails() {
```

```
    borrower = new Borrower("George Red", "unique_borrower_number", "new details");
```

```
    updateResult = borrowerService.updateBorrower(borrower);
```

```
}
```

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

```
public void theSystemDisplaysAnErrorMessage() {
```

```
    assertEquals("User does not exist", updateResult);
```

```
}
```

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

```
public void theSystemDeletesGeorgeRedAccount() {
```

```
    deletionResult = borrowerService.deleteBorrower("George Red");
```

```
}
```

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

```
public void theSystemRemovesGeorgeRedDetails() {
```

```
    assertEquals("Deletion success", deletionResult);
```

```
    assertNull(borrowerService.findBorrowerByName("George Red"));
```

```
}
```

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

```
public void theSystemAttemptsToDeleteGeorgeRedAccount() {
```

```
    deletionResult = borrowerService.deleteBorrower("George Red");
```

```
}
```

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

```
public void theSystemInformsThatTheBorrowerDoesNotExist() {
```

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

```
}
```

```
@Given("George Red has pending items")
```

```
public void georgeRedHasPendingItems() {
```

```
    borrower.setPendingItems(1);
```

```
    borrowerService.updateBorrower(borrower);
```

```
}
```

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

```
public void theSystemDoesNotRemoveGeorgeRedDetails() {
```

```
    assertEquals("Cannot delete, pending items", deletionResult);
```

```
    assertNotNull(borrowerService.findBorrowerByName("George Red"));
```

```
}
```

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

```
public void theSystemInformsAboutThePendingItems() {
```

```
    assertEquals("Cannot delete, pending items", deletionResult);
```

```
}
```

```
}
```

```
...
```

BorrowerService Implementation Example

This is a simple example of what the `BorrowerService` class might look like. It should interact with your DAO layer to perform operations on the borrowers.

```
``java
```

```
package com.library.service;
```

```
import com.library.dao.BorrowerDao;
```

```
import com.library.domain.Borrower;
```

```
public class BorrowerService {
```

```
    private BorrowerDao borrowerDao = new BorrowerDao();
```

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

```
        if (borrowerDao.findByName(borrower.getName()) != null) {
```

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

```
        }
```

```
        borrowerDao.save(borrower);
```

```
        return "Success";
```

```
    }
```

```
    public Borrower findBorrowerByName(String name) {
```

```
        return borrowerDao.findByName(name);
```

```
    }
```

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

```

    Borrower existingBorrower = borrowerDao.findByName(borrower.getName());

    if (existingBorrower == null) {

        return "User does not exist";

    }

    borrowerDao.update(borrower);

    return "Update success";

}

public String deleteBorrower(String name) {

    Borrower borrower = borrowerDao.findByName(name);

    if (borrower == null) {

        return "Borrower does not exist";

    }

    if (borrower.getPendingItems() > 0) {

        return "Cannot delete, pending items";

    }

    borrowerDao.delete(borrower);

    return "Deletion success";

}

}

...

```

BorrowerDao Implementation Example

This is a simple example of what the `BorrowerDao` class might look like. This should interact with your database.

```
```java
```

```
package com.library.dao;
```

```
import com.library.domain.Borrower;
```

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

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

```
public class BorrowerDao {
```

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

```
 public Borrower findByName(String name) {
```

```
 return borrowerDatabase.get(name);
```

```
 }
```

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

```
 borrowerDatabase.put(borrower.getName(), borrower);
```

```
 }
```

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

```
 borrowerDatabase.put(borrower.getName(), borrower);
 }

 public void delete(Borrower borrower) {
 borrowerDatabase.remove(borrower.getName());
 }
}

...

```

### ### Borrower Domain Class

This is a simple example of what the `Borrower` class might look like.

```
```java
package com.library.domain;

public class Borrower {

    private String name;

    private String borrowerNumber;

    private String details;

    private int pendingItems;
}

```

```
public Borrower(String name, String borrowerNumber, String details) {  
  
    this.name = name;  
  
    this.borrowerNumber = borrowerNumber;  
  
    this.details = details;  
  
    this.pendingItems = 0;  
  
}
```

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

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

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

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

```

public int getPendingItems() {

    return pendingItems;

}

public void setPendingItems(int pendingItems) {

    this.pendingItems = pendingItems;

}

}

...

```

The provided implementation is a starting point. You may need to adjust it to fit the exact architecture and functionality of your existing library system. If you have any specific details or requirements, please let me know so I can tailor the solution accordingly.

User

Now, based on all the above and with the same pattern, create the step definitions for the feature: Loaning items

ChatGPT

Sure! Here are the step definitions for the feature "Loaning items" using Java and Cucumber.

Loaning Items Step Definitions

```

```java

```

```

package com.library.steps;

```

```
import com.library.domain.Borrower;

import com.library.domain.Item;

import com.library.service.BorrowerService;

import com.library.service.ItemService;

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 LoaningItemsSteps {

 private BorrowerService borrowerService = new BorrowerService();

 private ItemService itemService = new ItemService();

 private Borrower borrower;

 private Item item;

 private String loanResult;

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

 public void theLibraryHasTheItemAvailable(String itemName) {

 item = itemService.findItemByName(itemName);

 if (item == null) {

 item = new Item(itemName, true);

 itemService.addItem(item);

 }

 }

}
```

```
}

 assertTrue(item.isAvailable());

}
```

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

public void georgeRedIsARegisteredBorrower() {

 borrower = new Borrower("George Red", "unique_borrower_number", "details");

 borrowerService.registerBorrower(borrower);

 assertNotNull(borrowerService.findBorrowerByName("George Red"));

}
```

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

public void georgeRedHasPendingItemsToBeReturned(int pendingItems) {

 borrower.setPendingItems(pendingItems);

 borrowerService.updateBorrower(borrower);

 assertEquals(pendingItems, borrowerService.findBorrowerByName("George
Red").getPendingItems());

}
```

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

public void georgeRedHasBeenAssignedAMaximumLendingLimit(int lendingLimit) {

 borrower.setLendingLimit(lendingLimit);

 borrowerService.updateBorrower(borrower);

}
```

```
 assertEquals(lendingLimit, borrowerService.findBorrowerByName("George
Red").getLendingLimit());
 }
```

```
@When("George Red borrows the item {string}")
```

```
public void georgeRedBorrowsTheItem(String itemName) {

 item = itemService.findItemByName(itemName);

 loanResult = itemService loanItemToBorrower(item, borrower);

}
```

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

```
public void theSystemSuccessfullyLoansTheItemToGeorgeRed(String itemName) {

 assertEquals("Loan success", loanResult);

 assertFalse(itemService.findItemByName(itemName).isAvailable());

}
```

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

```
public void georgeRedPendingItemsIncreaseTo(int newPendingItems) {

 assertEquals(newPendingItems, borrowerService.findBorrowerByName("George
Red").getPendingItems());

}
```

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

```
public void theLibraryHasTheItemsAvailable(String itemName1, String itemName2) {
```

```
Item item1 = new Item(itemName1, true);

Item item2 = new Item(itemName2, true);

itemService.addItem(item1);

itemService.addItem(item2);

assertTrue(item1.isAvailable());

assertTrue(item2.isAvailable());

}
```

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

public void georgeRedTriesToBorrowBothItems() {

 Item item1 = itemService.findItemByName("Harry Potter");

 Item item2 = itemService.findItemByName("Moby Dick");

 loanResult = itemService loanItemToBorrower(item1, borrower);

 loanResult += itemService loanItemToBorrower(item2, borrower);

}
```

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

public void theSystemSuccessfullyLoansTheItemToGeorgeRedWithADueDateSet(String
itemName) {

 assertEquals("Loan success", loanResult.contains("Loan success") ? "Loan success" : "");

 assertFalse(itemService.findItemByName(itemName).isAvailable());

}
```

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



```

 public void theSystemDoesNotLoanToGeorgeRedDueToTheLendingLimitReached(String
itemName) {

 assertEquals("Lending limit reached", loanResult.contains("Lending limit reached") ?
"Lending limit reached" : "");

 assertTrue(itemService.findItemByName(itemName).isAvailable());

 }

```

```

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

```

```

public void theItemIsInTheLibraryButNotInTheSystem(String itemName) {

 item = new Item(itemName, true);

 // Not adding to the item service to simulate not in the system

}

```

```

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

```

```

public void georgeRedTriesToBorrowTheItem(String itemName) {

 loanResult = itemService loanItemToBorrower(item, borrower);

}

```

```

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

```

```

public void theSystemReturnsAnErrorDueToTheItemStatus() {

 assertEquals("Item not found", loanResult);

}

```

```

@Then("the system withdraws the item {string}")

```

```
public void theSystemWithdrawsTheItem(String itemName) {

 assertNull(itemService.findItemByName(itemName));

}
```

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

public void georgeRedTriesToBorrowTheItemAgain(String itemName) {

 item = itemService.findItemByName(itemName);

 loanResult = itemService loanItemToBorrower(item, borrower);

}
```

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

public void theSystemDoesntAllowTheLoan() {

 assertEquals("Loan not allowed", loanResult);

}
```

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

public void georgeRedPendingItemsRemain(int pendingItems) {

 assertEquals(pendingItems, borrowerService.findBorrowerByName("George
Red").getPendingItems());

}

}

...
```

### ItemService Implementation Example

Here's a basic example of what the `ItemService` class might look like:

```
```java
```

```
package com.library.service;
```

```
import com.library.dao.ItemDao;
```

```
import com.library.domain.Borrower;
```

```
import com.library.domain.Item;
```

```
public class ItemService {
```

```
    private ItemDao itemDao = new ItemDao();
```

```
    public Item findItemByName(String name) {
```

```
        return itemDao.findByName(name);
```

```
    }
```

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

```
        itemDao.save(item);
```

```
    }
```

```
    public String loanItemToBorrower(Item item, Borrower borrower) {
```

```

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

        return "Item not found";

    }

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

        return "Lending limit reached";

    }

    item.setAvailable(false);

    itemDao.update(item);

    borrower.setPendingItems(borrower.getPendingItems() + 1);

    // Assuming updateBorrower method is available in borrowerService

    BorrowerService borrowerService = new BorrowerService();

    borrowerService.updateBorrower(borrower);

    return "Loan success";

}

}

'''

```

ItemDao Implementation Example

Here's a basic example of what the `ItemDao` class might look like:

```

```java

package com.library.dao;

```

```
import com.library.domain.Item;
```

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

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

```
public class ItemDao {
```

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

```
 public Item findByName(String name) {
```

```
 return itemDatabase.get(name);
```

```
 }
```

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

```
 itemDatabase.put(item.getName(), item);
```

```
 }
```

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

```
 itemDatabase.put(item.getName(), item);
```

```
 }
```

```
}
```

```
...
```

### ### Item Domain Class

Here's a simple example of what the `Item` class might look like:

```
```java
package com.library.domain;

public class Item {

    private String name;

    private boolean available;

    public Item(String name, boolean available) {

        this.name = name;

        this.available = available;

    }

    public String getName() {

        return name;

    }

    public boolean isAvailable() {

        return available;

    }

}
```

```

    }

    public void setAvailable(boolean available) {

        this.available = available;

    }

}

...

```

This implementation follows the same pattern as the previous feature. Adjust the code as needed to fit your existing architecture and functionality. If you have specific details or requirements, please let me know so I can tailor the solution accordingly.

User

Now, do the same for the feature: Returning a borrowed item in the library

ChatGPT

Sure! Here are the step definitions for the feature "Returning a borrowed item to the library" using Java and Cucumber.

Returning a Borrowed Item Step Definitions

```

```java

package com.library.steps;

import com.library.domain.Borrower;

import com.library.domain.Item;

```

```
import com.library.service.BorrowerService;
```

```
import com.library.service.ItemService;
```

```
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;
```

```
public class ReturningItemSteps {
```

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

```
 private ItemService itemService = new ItemService();
```

```
 private Borrower borrower;
```

```
 private Item item;
```

```
 private String returnResult;
```

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

```
 public void georgeRedBorrowedTheItemDaysPriorToToday(String itemName, int daysAgo) {
```

```
 item = new Item(itemName, false);
```

```
 item.setBorrowDate(LocalDate.now().minusDays(daysAgo));
```

```
 itemService.addItem(item);
```

```
 borrower = new Borrower("George Red", "unique_borrower_number", "details");
```



```
 borrowerService.registerBorrower(borrower);

 borrower.setPendingItems(1);

 borrowerService.updateBorrower(borrower);
}
```

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

public void georgeRedHasBeenAssignedMaximumLendingDays(int maxLendingDays) {

 borrower.setMaxLendingDays(maxLendingDays);

 borrowerService.updateBorrower(borrower);
}
```

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

public void theReturnOfIsProcessed(String itemName) {

 item = itemService.findItemByName(itemName);

 returnResult = itemService.returnItem(item, borrower);
}
```

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

public void theSystemMarksTheStateOfAsAVAILABLE(String itemName) {

 item = itemService.findItemByName(itemName);

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

```

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

public void georgeRedHasOneLessPendingItem() {

 assertEquals(0, borrowerService.findBorrowerByName("George
Red").getPendingItems());

}

```

```

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

public void georgeRedDoesNotPayAFine() {

 assertEquals("Return success, no fine", returnResult);

}

```

```

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

public void theReturnDateOfTheLoanIsSetToTodaysDate() {

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

}

```

```

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

public void georgeRedPaysAFineBasedOnTheBorrowerCategory() {

 assertEquals("Return success, fine applied", returnResult);

}

}

...

```

### ItemService Implementation Example (Updated)

Here's an updated example of the `ItemService` class that includes the return item logic:

```
```java

package com.library.service;

import com.library.dao.ItemDao;

import com.library.domain.Borrower;

import com.library.domain.Item;

import java.time.LocalDate;

import java.time.temporal.ChronoUnit;

public class ItemService {

    private ItemDao itemDao = new ItemDao();

    public Item findItemByName(String name) {

        return itemDao.findByName(name);

    }

    public void addItem(Item item) {

        itemDao.save(item);

    }

}
```

```
}
```

```
public String loanItemToBorrower(Item item, Borrower borrower) {
```

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

```
        return "Item not found";
```

```
    }
```

```
    if (borrower.getPendingItems() >= borrower.getLendingLimit()) {
```

```
        return "Lending limit reached";
```

```
    }
```

```
    item.setAvailable(false);
```

```
    item.setBorrowDate(LocalDate.now());
```

```
    itemDao.update(item);
```

```
    borrower.setPendingItems(borrower.getPendingItems() + 1);
```

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

```
    borrowerService.updateBorrower(borrower);
```

```
    return "Loan success";
```

```
}
```

```
public String returnItem(Item item, Borrower borrower) {
```

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

```
        return "Item not borrowed";
```

```
    }
```

```
    item.setAvailable(true);
```

```

        item.setReturnDate(LocalDate.now());

        itemDao.update(item);

        borrower.setPendingItems(borrower.getPendingItems() - 1);

        BorrowerService borrowerService = new BorrowerService();

        borrowerService.updateBorrower(borrower);

        long    daysBorrowed    =    ChronoUnit.DAYS.between(item.getBorrowDate(),
LocalDate.now());

        if (daysBorrowed > borrower.getMaxLendingDays()) {

            return "Return success, fine applied";

        }

        return "Return success, no fine";

    }

}

'''

```

BorrowerService Implementation Example (Updated)

Here's an updated example of the `BorrowerService` class that includes the max lending days logic:

```

'''java

package com.library.service;

```

```
import com.library.dao.BorrowerDao;
```

```
import com.library.domain.Borrower;
```

```
public class BorrowerService {
```

```
    private BorrowerDao borrowerDao = new BorrowerDao();
```

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

```
        if (borrowerDao.findByName(borrower.getName()) != null) {
```

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

```
        }
```

```
        borrowerDao.save(borrower);
```

```
        return "Success";
```

```
    }
```

```
    public Borrower findBorrowerByName(String name) {
```

```
        return borrowerDao.findByName(name);
```

```
    }
```

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

```
        Borrower existingBorrower = borrowerDao.findByName(borrower.getName());
```

```
        if (existingBorrower == null) {
```

```
            return "User does not exist";
```

```

    }

    borrowerDao.update(borrower);

    return "Update success";
}

public String deleteBorrower(String name) {

    Borrower borrower = borrowerDao.findByName(name);

    if (borrower == null) {

        return "Borrower does not exist";

    }

    if (borrower.getPendingItems() > 0) {

        return "Cannot delete, pending items";

    }

    borrowerDao.delete(borrower);

    return "Deletion success";

}

}

'''

```

Borrower Domain Class (Updated)

Here's an updated example of the `Borrower` class that includes the max lending days logic:

```
```java
```

```
package com.library.domain;
```

```
public class Borrower {
```

```
 private String name;
```

```
 private String borrowerNumber;
```

```
 private String details;
```

```
 private int pendingItems;
```

```
 private int lendingLimit;
```

```
 private int maxLendingDays;
```

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

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

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

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

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

```
 }
```

```
 public String getName() {
```

```
 return name;
```

```
 }
```



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

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

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

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

```
public void setPendingItems(int pendingItems) {
 this.pendingItems = pendingItems;
}
```

```
public int getLendingLimit() {
 return lendingLimit;
}
```

```
public void setLendingLimit(int lendingLimit) {
 this.lendingLimit = lendingLimit;
}
```

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

```
public void setMaxLendingDays(int maxLendingDays) {
 this.maxLendingDays = maxLendingDays;
}
}
...
```

### ### Item Domain Class (Updated)

Here's an updated example of the `Item` class that includes the borrow and return dates logic:

```
```java
```

```
package com.library.domain;
```

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

```
public class Item {

    private String name;

    private boolean available;

    private LocalDate borrowDate;

    private LocalDate returnDate;

    public Item(String name, boolean available) {

        this.name = name;

        this.available = available;

    }

    public String getName() {

        return name;

    }

    public boolean isAvailable() {

        return available;

    }

    public void setAvailable(boolean available) {

        this.available = available;

    }

}
```

```
}
```

```
public LocalDate getBorrowDate() {  
    return borrowDate;  
}
```

```
public void setBorrowDate(LocalDate borrowDate) {  
    this.borrowDate = borrowDate;  
}
```

```
public LocalDate getReturnDate() {  
    return returnDate;  
}
```

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

ItemDao Implementation Example (Unchanged)

The `ItemDao` class remains the same as in the previous example:

```
```java
```

```
package com.library.dao;
```

```
import com.library.domain.Item;
```

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

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

```
public class ItemDao {
```

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

```
 public Item findByName(String name) {
```

```
 return itemDatabase.get(name);
```

```
 }
```

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

```
 itemDatabase.put(item.getName(), item);
```

```
 }
```

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

```
 itemDatabase.put(item.getName(), item);
```

```
 }
```

```
}
```

```
...
```

This implementation follows the same pattern as the previous features and includes the logic for returning items and handling fines for late returns. Adjust the code as needed to fit your existing architecture and functionality. If you have specific details or requirements, please let me know so I can tailor the

ChatGPT

solution accordingly.

**User**

**At last, do the same for the feature: Delayed Notification**

ChatGPT

Sure! Here are the step definitions for the feature "Delayed Return Notification" using Java and Cucumber.

### Delayed Return Notification Step Definitions

```
```java
```

```
package com.library.steps;
```

```
import com.library.domain.Borrower;
```

```
import com.library.domain.Item;
```

```
import com.library.service.BorrowerService;
```

```
import com.library.service.ItemService;
```

```
import com.library.service.NotificationService;
```

```
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;
```

```
public class DelayedReturnNotificationSteps {
```

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

```
    private ItemService itemService = new ItemService();
```

```
    private NotificationService notificationService = new NotificationService();
```

```
    private Borrower borrower;
```

```
    private Item item;
```

```
    private boolean notificationResult;
```

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

```
    public void georgeRedHasBorrowedTheItem(String itemName) {
```

```
        item = new Item(itemName, false);
```

```
        item.setBorrowDate(LocalDate.now().minusDays(10)); // Assume the item was borrowed  
10 days ago
```

```
        itemService.addItem(item);
```

```
        borrower = new Borrower("George Red", "unique_borrower_number", "details");
```

```
    borrowerService.registerBorrower(borrower);

    borrower.setPendingItems(1);

    borrowerService.updateBorrower(borrower);
}
```

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

```
public void itemsDueDateHasPassed(String itemName) {
```

```
    item = itemService.findItemByName(itemName);
```

```
    assertTrue(item.getBorrowDate().plusDays(borrower.getMaxLendingDays()).isBefore(LocalDate.now()));
```

```
}
```

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

```
public void georgeRedHasAnEmailAddress() {
```

```
    borrower.setEmail("george.red@example.com");
```

```
    borrowerService.updateBorrower(borrower);
```

```
    assertNotNull(borrowerService.findBorrowerByName("George Red").getEmail());
```

```
}
```

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

```
public void theSystemExecutesTheDelayedReturnNotificationProcess() {
```

```
    notificationResult = notificationService.notifyDelayedReturn(borrower, item);
```

```
}
```



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

```
public void georgeRedReceivesAnEmailNotificationForTheReturnOfTheItem() {  
    assertTrue(notificationResult);  
}
```

```
@Given("George Red does not have an email address")
```

```
public void georgeRedDoesNotHaveAnEmailAddress() {  
    borrower.setEmail(null);  
    borrowerService.updateBorrower(borrower);  
    assertNull(borrowerService.findBorrowerByName("George Red").getEmail());  
}
```

```
@Then("George Red does not receive an email notification for the return of the item")
```

```
public void georgeRedDoesNotReceiveAnEmailNotificationForTheReturnOfTheItem() {  
    assertFalse(notificationResult);  
}  
}  
...
```

NotificationService Implementation Example

Here's an example of the `NotificationService` class that handles sending email notifications:

```

```java

package com.library.service;

import com.library.domain.Borrower;

import com.library.domain.Item;

public class NotificationService {

 public boolean notifyDelayedReturn(Borrower borrower, Item item) {

 if (borrower.getEmail() == null) {

 return false;

 }

 // Simulate sending an email

 System.out.println("Sending email to " + borrower.getEmail() + " about delayed return of " + item.getName());

 return true;

 }

}

```

```

BorrowerService Implementation Example (Updated)

Here's an updated example of the `BorrowerService` class that includes email handling logic:

```
```java
```

```
package com.library.service;
```

```
import com.library.dao.BorrowerDao;
```

```
import com.library.domain.Borrower;
```

```
public class BorrowerService {
```

```
 private BorrowerDao borrowerDao = new BorrowerDao();
```

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

```
 if (borrowerDao.findByName(borrower.getName()) != null) {
```

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

```
 }
```

```
 borrowerDao.save(borrower);
```

```
 return "Success";
```

```
 }
```

```
 public Borrower findBorrowerByName(String name) {
```

```
 return borrowerDao.findByName(name);
```

```
 }
```

```
public String updateBorrower(Borrower borrower) {

 Borrower existingBorrower = borrowerDao.findByName(borrower.getName());

 if (existingBorrower == null) {

 return "User does not exist";

 }

 borrowerDao.update(borrower);

 return "Update success";

}
```

```
public String deleteBorrower(String name) {

 Borrower borrower = borrowerDao.findByName(name);

 if (borrower == null) {

 return "Borrower does not exist";

 }

 if (borrower.getPendingItems() > 0) {

 return "Cannot delete, pending items";

 }

 borrowerDao.delete(borrower);

 return "Deletion success";

}

}

...
```

### ### Borrower Domain Class (Updated)

Here's an updated example of the `Borrower` class that includes email logic:

```
```java
```

```
package com.library.domain;
```

```
public class Borrower {
```

```
    private String name;
```

```
    private String borrowerNumber;
```

```
    private String details;
```

```
    private int pendingItems;
```

```
    private int lendingLimit;
```

```
    private int maxLendingDays;
```

```
    private String email;
```

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

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

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

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

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

```
    }
```

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

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

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

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

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

```
public void setPendingItems(int pendingItems) {  
    this.pendingItems = pendingItems;  
}
```

```
}
```

```
public int getLendingLimit() {
```

```
    return lendingLimit;
```

```
}
```

```
public void setLendingLimit(int lendingLimit) {
```

```
    this.lendingLimit = lendingLimit;
```

```
}
```

```
public int getMaxLendingDays() {
```

```
    return maxLendingDays;
```

```
}
```

```
public void setMaxLendingDays(int maxLendingDays) {
```

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

```
}
```

```
public String getEmail() {
```

```
    return email;
```

```
}
```

```
public void setEmail(String email) {
```

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

```

Item Domain Class (Unchanged)

The `Item` class remains the same as in the previous example:

```
```java
package com.library.domain;

import java.time.LocalDate;

public class Item {

 private String name;

 private boolean available;

 private LocalDate borrowDate;

 private LocalDate returnDate;

 public Item(String name, boolean available) {

 this.name = name;
 }
}

```



```
 this.available = available;
 }
}
```

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

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

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

```
public LocalDate getBorrowDate() {
 return borrowDate;
}
```

```
public void setBorrowDate(LocalDate borrowDate) {
 this.borrowDate = borrowDate;
}
```

```
public LocalDate getReturnDate() {
 return returnDate;
}
```

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

#### ### ItemDao Implementation Example (Unchanged)

The `ItemDao` class remains the same as in the previous example:

```
```java  
  
package com.library.dao;  
  
import com.library.domain.Item;  
import java.util.HashMap;  
import java.util.Map;  
  
public class ItemDao {
```

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

public Item findByName(String name) {
    return itemDatabase.get(name);
}

public void save(Item item) {
    itemDatabase.put(item.getName(), item);
}

public void update(Item item) {
    itemDatabase.put(item.getName(), item);
}
}
...
```

This implementation follows the same pattern as the previous features and includes the logic for sending notifications for delayed returns. Adjust the code as needed to fit your existing architecture and functionality. If you have specific details or requirements, please let me know so I can tailor the solution accordingly.