Büchereiverwaltung

0. Inbetriebnahme

OpenJDK 22 oder vergleichbar und GIT sollten auf dem Zielgerät installiert sein Die Umgebungsvariable "JAVA_HOME" sollte auf den Pfad der JDK Installation verweisen (z.B. "C:\Program Files\Java\jdk-22")

```
setx JAVA_HOME "PFAD ZUR JDK" – Windows
export JAVA_HOME=/usr/bin/java - Linux
```

Das Repository mit

```
git clone https://github.com/KevStr06/ASE.git
```

in einen Ordner der Wahl clonen (Der Ordner benötigt Schreibrechte; Sonderzeichen und lange übergeordnete Ordnernamen können Probleme bereiten)

Ein(e) Eingabeaufforderung / Terminal im Ordner "Buechereiverwaltung" öffnen

Je nach Eingabeaufforderung / Terminal einen der Befehle zur Initialisierung ausführen:

```
mvnw clean install - Windows CMD
./mvnw clean install - PowerShell, Linux, etc.
```

Nun wurde die Tests bereits aufgeführt. Diese können beliebig wiederholt werden.

```
mvnw test - Windows CMD
./mvnw test - PowerShell, Linux, etc.
```

Das automatisierte Demoskript starten:

```
mvnw exec:java -f ./0-plugins/0-presentation/ - Windows CMD
./mvnw exec:java -f ./0-plugins/0-presentation/ - PowerShell,
Linux, etc.
```

Durch die Ausführung werden im aktuellen Ausführungsverzeichnis drei .save Dateien erzeugt und anschließend für jeden User ein Mailto Link erzeugt. In der Mail wird jedes ausgeliehene Buch mit der Anzahl der verbleibenden Tage angegeben.

1. Domain Driven Design

1.1 Ubiquitous Language

Begrifflichkeit	Fachliche Bedeutung	Regeln
Buch	Ein konkretes Buch, welches von einem Nutzer ausgeliehen werden kann.	- Kann maximal von einem Nutzer ausgeliehen werden
Bibliothekar	Eine Person, die berechtigt ist, neue Bücher hinzuzufügen, alte Bücher zu löschen und einzusehen welche Nutzer welche Bücher ausgeliehen haben. Ist der Nutzer der Software.	
Nutzer	Eine Person, die berechtigt ist Bücher auszuleihen, zurückzugeben und Bücher auf eine Merkliste hinzuzufügen.	
Ausleihformular	Beinhalten ein Buch, Nutzer und ein Rückgabedatum.	- Das Buch darf nicht bereits ausgeliehen sein
Rückgabedatum	Datum bis zu welchem ein Nutzer ein Buch zurückbringen muss.	- Muss in der Zukunft liegen
Merkliste	Beinhaltet alle Bücher, die ein Nutzer sich vermerkt hat.	
ISBN	Internationale Standardbuchnummer	- Besteht insgesamt aus 17 Zeichen - Beinhaltet genau vier „-“

Prozesse	
Ausleihen	Der Prozess des Ausleihens eines Buches von einem konkreten Nutzer, dabei wird ein Ausleihformular erstellt.
Zurückgeben	Der Prozess des Zurückgebens eines Buches eines konkreten Nutzers, dabei wird das dazugehörige Ausleihformular gelöscht.
Nutzer erstellen	Der Prozess in dem ein Nutzer in der Bibliothek registriert wird
Buch erstellen	Der Prozess in dem ein Buch in der Bibliothek registriert wird
Speichern	Der Prozess in dem alle Bücher, Nutzer und Ausleihformulare gespeichert werden.
Laden	Der Prozess in dem die gespeicherten Elemente geladen werden.

1.2 Verwendete taktische Muster des DDD

1.2.1 Value Objects

ReturnDate

Das Rückgabe des Ausleihformulars ist als Value Object implementiert, da es nur aus einem Datum besteht und nicht verändert wird, da beim Erstellen eines Ausleihformulars ein konkretes Rückgabedatum definiert wird und dieses nicht verändert wird, da es nicht die Möglichkeit gibt den Ausleihzeitrum zu verlängern.

1.2.2 Entities

LoanAgreement

Das Ausleihformular ist als Entity realisiert, da es Soft Referenzen besitzt, aber Teil des Buch Aggregates ist.

1.2.3 Aggregates

User

Der Nutzer ist ein Aggregat, da in dieser Klasse mehrere Value Objects und Entities zusammengeführt werden.

1.2.4 Repositories

Book Repository

Das Book Repository beinhaltet alle Bücher, um somit zu gewährleisten, dass die Bücher weiterhin existieren, auch wenn sie nicht mehr ausgeliehen sind. Des Weiteren kann somit in anderen Repositories auf bereits erstellte Bücher zugegriffen werden.

1.2.5 Domain Services

BookRenter

Die Klasse BookRenter wurde als Domain Service implementiert, da sich die Aufgabe dieses Services in der Domain Layer abspielt und sich somit die Logik und Validierung an einer Stelle befinden.

2. Clean Architecture

2.1 Domain Layer

Bildet den Kern des Systems mit der Geschäftslogik und Regeln. In dieser Schicht ist die Obiquitous Language implementiert.

2.2 Application Layer

Diese Schicht bildet die Brücke zwischen der Präsentationsschicht und der Repositories und ist verantwortlich für die Koordination der Anwendungslogik. In dieser Implementierung fungieren die Services als Schnittstelle für den Zugriff auf die Repositories, wodurch die Domain-Schicht nur diejenigen Funktionen implementieren muss, die fachlich relevant sind und ihren Repository-Schnittstellen entsprechen.

2.3 Plugin Layer

Persistence Layer

Diese Schicht ist zuständig für das Speicher und Laden der Objekte, dafür wir die Bibliothek Jackson verwendet und das ganze als JSON gespeichert.

3. Programming Principles

3.1 SOLID – Single Responsibility Principle

Das Single Responsibility Principle (SRP) besagt, dass eine Klasse oder ein Modul nur für eine einzige Verantwortlichkeit oder Aufgabe zuständig sein sollte. Dieses Prinzip wird gut durch die Klasse Book demonstriert, da diese nur dafür zuständig ist jeweils ein konkretes Buch zu repräsentieren, daher wird diese Klasse auch nur geändert, wenn etwas an dem Aufbau eines Buches zu ändern ist.

3.2 SOLID – Dependeny Inversion Principle

Das Dependency Inversion Principle (DIP) besagt, dass Module von abstrakten Konzepten abhängig sein sollten, nicht von konkreten Implementierungen. Dieses Prinzip lässt sich gut in dem Zusammenspiel der Klasse LoanAgreementManagementService, (kurz Service) dem Interface LoanAgreementRepository (kurz Repository) und der Klasse LoanAgreementJacksonRepository (kurz Jackson) erkennen. Die Klasse Service Klasse verwendet das Repository ohne sich um die konkreten Implementierungen kümmern zu müssen, den dafür ist die Jackson Klasse zuständig. Diese implementiert die im Repository definierten Funktionen.

3.3 GRASP – Pure Fabrication

Dieses Prinzip beschreibt die Trennung von technischen und Domänenwissen. Das ist gut erkennbar an den Jackson Repositorys diese bietet die technische Umsetzung unabhängig vom Domänenwissen.

3.4 GRASP – Low Coupling

Dieses Prinzip besagt, dass die Abhängigkeit von Klassen und Methoden möglichst gering seien soll, um somit den nächsten Befehl austauschbarer zu machen. Dies lässt

sich gut in den Service Klassen erkennen, da diese so aufgebaut sind, dass sie zwar mit den Repository Klassen zusammenhängen, aber unabhängig von der konkreten Implementierung dieser funktionieren.

3.5 YAGNI - You ain't gonna need it

Dieses Prinzip beschreibt das Weglassen von Methoden, dabei bezieht es sich besonders auf Methoden, welche man sonst Implementieren würde, weil man sie vielleicht einmal brauchen könnte. In diesem Projekt wurde beispielswiese bei den Bookmarks darauf verzichtet eine Funktion zu implementieren, dass alle vorgemerkten Bücher gelöscht werden können. Dies liegt daran, dass in der Realität nie seine gesamte Merkliste auf einmal löschen soll, es sei den der Nutzer wird gelöscht und dabei wird automatisch schon die gesamte Merkliste gelöscht.

4. Refactoring

4.1 Code Smells

Long Parameter List

Die Klasse Book weist den Code Smell Long Parameter List auf in ihrem Konstruktor, da der Konstruktor vier Übergabeparameter besitzt.

Primitive Obsession

Die Klasse Book weist den Code Smell Primitiv Obsession im Konstruktor auf, erkennbar an der Verwendung des Typs String anstelle der definierten Objekte. Konkret könnten „authorName“ und „authorSurname“ durch das Objekt Name ersetzt werden.

Duplicate Code

Für diesen Code Smell lassen sich im Code mehrere Beispiele finden, zwei sind hier genannt.

Die Klasse User weist den Code Smell Duplicate Code auf, zusehen ist diese an der Stelle der zwei Konstruktoren, welche fast gleich sind und sich reduzieren lassen.

Die Klasse LoanAgreement weist beim Konstruktor das gleiche auf, auch hier lassen sich die Konstruktoren durch Verweis auf den anderen reduzieren.

Coupler

Für diesen Code Smell habe ich keinen konkreten Namen gefunden.

In der Historie zu sehen im Commit „Change getAll functions for Books, Users and LoanAgreements“ wurde ursprünglich eine Liste der Books, User und LoanAgreements zurückgegeben, was anschließen durch die Rückgabe der Ids ersetzt wurde. Dadurch kann nicht direkt auf ein Book, User und LoanAgreement zugegriffen werden.

4.2 Refactoring

Remove Parameter, Introduce Parameter Object

Die Long Parameter List und die Primitive Obsession im Konstruktor der Klasse Book wurde im Commit „Change Book constructor“ behoben, indem authorName und Surname, welche zuvor Stings waren durch ein Name Objekt ersetzt wurden. Dadurch wurde die Anzahl der Übergabeparameter von vier auf drei reduziert und der Name ist automatisch validiert.

Extract Method

Der Duplicate Coder in der Klasse User durch die zwei Konstruktoren wurde im Commit „Change User constructor“ behoben, indem der private Konstruktor im öffentlichen Konstruktor aufgerufen wird.

5. Entwurfsmuster

Das Entwurfsmuster „Strategie“ wurde in diesem Projekt umgesetzt. Strategie ist ein Verhaltensmuster, welches die Möglichkeit beschreibt, das Verhalten des Programms während der Laufzeit flexibel zu gestalten und unabhängig von Klienten ändern zu können.

Das Entwurfsmuster wurde durch das Interface ContactMethodStrategy und die Klassen EmailContactStrategy und LetterContactStrategy umgesetzt. Diese Klassen erlauben es eine Nachricht an den User entweder per Mail oder per Post zu schicken, je nachdem, was beim Nutzer festgelegt ist. Dabei ist als Standard die EmailContactStragey hinterlegt. Die LetterContactStragey ist noch nicht implementiert und die Möglichkeit auf diese Kontakt Möglichkeit zu wechseln fehlt im aktuellen Stand des Projektes ebenso, aber würde sich durch eine Funktion in der Klasse User realisieren lassen. Die Funktion müsste die ContactMethodStrategy im User ändern.