Dokumentation Java 1

13 Buchverleih

Fachbereich
Gebäudetechnik und Informatik
Angewandte Informatik

Projektmitglieder

Stephanie Wachs
Josephina Burger
Lucas-Manfred Herpe

Bearbeitungszeitraum

15.01.2025 - 07.03.2025

Inhaltsverzeichnis

1. Aufgabenstellung

Ziel unseres Projekts war die Entwicklung eines Bibliothekssystems zur Verwaltung von Büchern und Personen. Die folgenden Anforderungen wurden dabei berücksichtigt:

Es soll ein Bibliothekssystem abgebildet werden, in dem Bücher und Personen verwaltet werden. Ein Buch hat einen Titel, einen Autor, ein Genre und einen Zustand. Personen haben einen Vor- und Nachnamen, ein Geburtsdatum sowie eine Adresse. Ausgeliehene Bücher müssen innerhalb eines Monats zurückgegeben werden. Wenn ein Buch nicht pünktlich zurückgebracht wird, fällt eine Gebühr pro angefangene Woche an. Sollte ein Buch beschädigt zurückgegeben werden, wird eine zusätzliche Gebühr fällig.

Die Geschäftslogik umfasst dabei folgende Punkte:

- Das Verwalten von Büchern (Hinzufügen, Bearbeiten und Löschen von Büchern).
- Das Verwalten von Personen (Hinzufügen, Bearbeiten und Löschen von Personendaten).
- Das Ausleihen von Büchern an Personen.
- Das Zurückgeben von Büchern einschließlich der Berechnung von potentiellen Strafgebühren bei verspäteter Rückgabe oder Beschädigung des Buches.
- Offene Gebühren werden bei der betreffenden Person erfasst und verwaltet.
- Bücher und Personen sind entsprechend ihres Titels oder Namens lexikographisch vergleichbar.
- Der Filter für Bücher soll folgende Kriterien erfüllen:
    - Genre
    - Autor
    - Titel (enthält Textbausteine)
- Der Filter für Personen soll folgende Kriterien erfüllen:
    - Ausgeliehene Bücher
    - Offene Gebühren in Höhe von (von – bis)
    - Anzahl der ausgeliehenen Bücher (von – bis)
- Es ist eine geeignete String-Repräsentation der Personenkartei und der Leihliste umzusetzen.

2. Implementierung

Im Rahmen der Implementierung haben wir uns entschieden, eine klare logische Unterteilung zwischen Datenmodellen und Geschäftslogik vorzunehmen. Diese Trennung ermöglicht eine klare Strukturierung des Codes, bei der die Datenmodelle (*Person, Book, Address*) die reine Datenhaltung und grundlegende Validierungslogik übernehmen, während die *LibrarySystem*-Klasse die komplexeren Geschäftsregeln und Interaktionen verwaltet. Die Modelle repräsentieren in erster Linie die Struktur der Daten, beinhalten jedoch auch grundlegende logische Funktionen. Die Geschäftslogik ist hauptsächlich für die Verarbeitung und das Management der Anwendung zuständig.

Im Rahmen unseres Projekts haben wir die Modelle *Person*, *Book* und *Address* definiert, sowie die *LibrarySystem*-Klasse für die Geschäftslogik und die *BookFilter*- und *PersonFilter*-Klassen, die für das Filtern von Büchern und Personen zuständig sind. Im Folgenden wird der Aufbau der Modelle sowie die Funktionen dieser Klassen genauer beschrieben.

Dem Klassendiagramm sind die Struktur, die Attribute der einzelnen Klassen sowie die Methoden zu entnehmen. Eine vergrößerte Ansicht des Diagramms ist im Anhang enthalten.

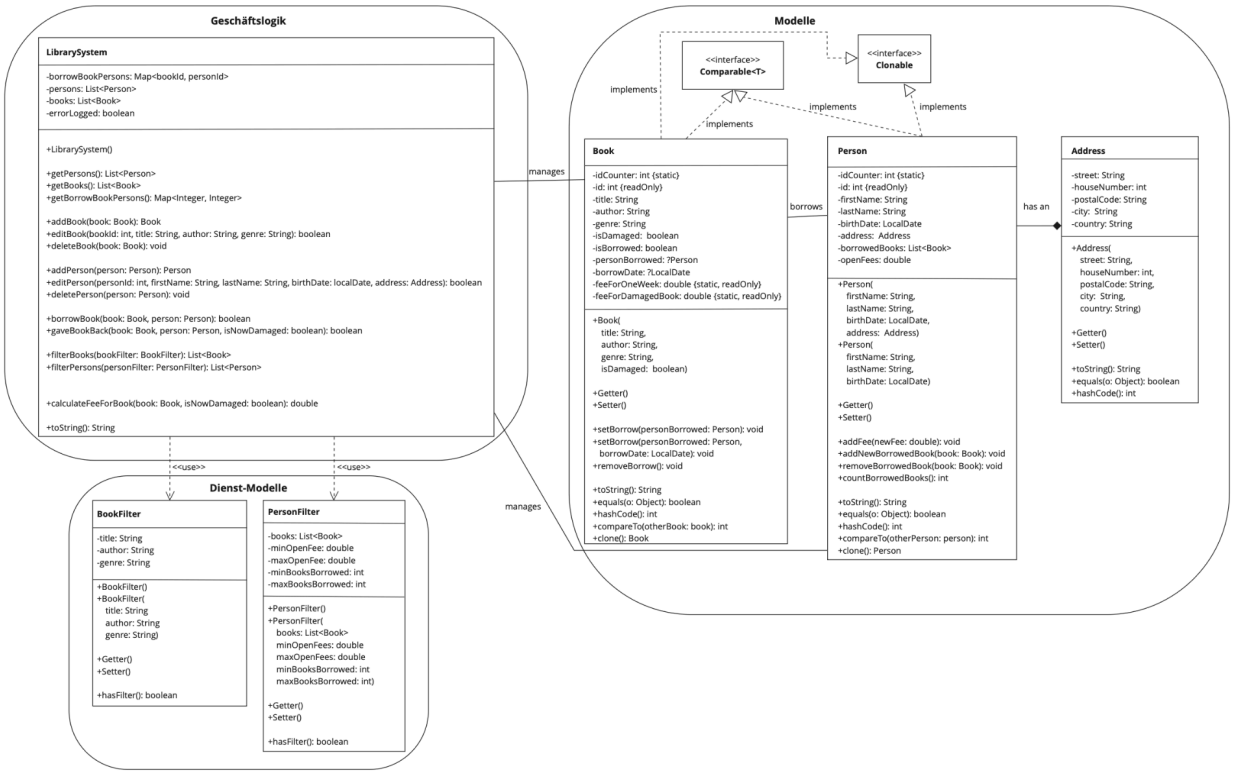

Abbildung 1: Klassendiagramm des Buchverleihs

2.1 Personen

Gemäß der Aufgabenstellung, besitzt eine Person einen Vor- und Nachnamen, ein Geburtsdatum und eine Adresse. Zur eindeutigen Identifizierung der Person entschieden wir uns jeder Person außerdem eine ID zu geben. Hierzu verwenden wir eine statische Klassenvariable *idCounter*, da die ID für jede neue Person eindeutig sein soll und automatisch hochgezählt wird. Um einem Personen-Objekt dann eine ID zuzuweisen, verwenden wir eine Instanzvariable, welche final ist, damit diese im Nachhinein nicht mehr geändert werden kann. Die Attribute jeder Instanz können über die entsprechenden Getter- und Setter-Methoden gelesen und verändert werden.

Die Adresse wird durch die separate Klasse Address repräsentiert und kann beim Erstellen einer Person optional angegeben werden. Die Klasse *Address* umfasst die Attribute Straße, Hausnummer, Postleitzahl, Stadt und Land sowie die entsprechenden Getter- und Setter-Methoden. Zwei *Address*-Objekte gelten als gleich, wenn alle ihre Attribute übereinstimmen (überschriebene equals Methode).

Das *Person* Modell enthält außerdem einen kleinen logischen Teil, da in ihm sowohl die Liste der ausgeliehenen Bücher (*borrowedBooks*) als auch die offenen, ausstehenden Gebühren (*openFees*) gespeichert werden. Das Modell bietet die Methoden *addFee*, *addNewBorrowedBook*, *removeBorrowedBook* und *countBorrowedBooks*, um die offenen Gebühren und ausgeliehenen Bücher zu verwalten. Man hätte die Logik in ein separates Modell auslagern können. Unsere Vorüberlegung hierzu sind dem folgenden Bild zu entnehmen:

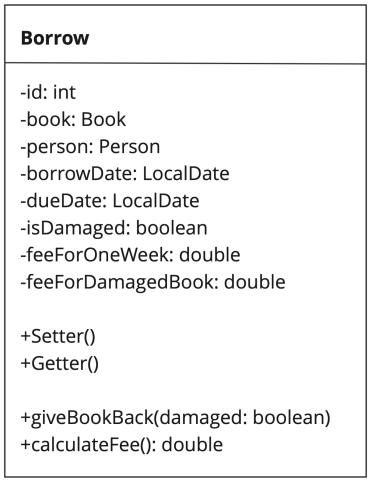

Abbildung 2: Alternative Borrow-Klasse

Wir haben uns jedoch bewusst dagegen entschieden, da die Auslagerung dieser Logik in ein separates Modell in einem vergleichsweise kleinen Projekt die Komplexität unnötig erhöht hätte. Die enge Verbindung zwischen einer Person und ihren ausgeliehenen Büchern sowie ihren offenen Gebühren rechtfertigt die Integration dieser Attribute direkt in die *Person*-Klasse.

Mit der Methode clone lässt sich eine flache Kopie von einer Person- Instanz erstellen. Zwei Person-Objekte gelten als gleich, wenn ihre *ID*, ihre *openFees*, ihr *firstName*, *lastName*, ihr *birthDate*, ihre *Address* und ihre *borrowedBooks* übereinstimmen (überschriebene equals Methode).

2.2 Bücher

Ein Buch besitzt einen Titel, einen Autor und ein Genre. Das Genre haben wir als *String* implementiert. Obwohl die Verwendung von Enumerationen zur Kategorisierung der Genres möglich gewesen wäre, haben wir uns bewusst dagegen entschieden. Der Grund dafür ist, dass es eine große Anzahl an möglichen Genres gibt, die nicht auf eine vordefinierte Auswahl beschränkt werden sollten. Die Verwendung eines *String* ermöglicht uns daher eine größere Flexibilität bei der Vergabe und Verwaltung der Genres.

Zusätzlich werden der Zustand des Buches (*isDamaged*), die Person, die das Buch ausgeliehen hat (*personBorrowed*), und das Ausleihdatum (*borrowDate*) gespeichert. Die letzten beiden Attribute sind optional, da es auch Bücher geben kann, die im Moment nicht ausgeliehen sind.

Die Gebühren für verspätete Rückgaben (*feeForOneWeek*) und beschädigte Bücher (*feeForDamagedBook*) werden als statische, finale Klassenvariablen in der Book-Klasse definiert. Diese Werte gehören zur Klasse selbst, da sie für alle Bücher gelten, nur einmal initialisiert werden und nachträglich nicht geändert werden sollen.

Zur eindeutigen Identifizierung eines Buches verwenden wir eine ID. Die ID wird genauso erstellt, wie bei der *Person* Klasse. Die Modellierung erfolgt in der *Book*-Klasse, die auch Getter und Setter bereitstellt, um gezielte Modifikationen vorzunehmen. Zwei Book-Objekte gelten als gleich, wenn *id*, *title*, *author*, *genre*, *isDamaged*, *personBorrowed*, und *borrowDate* übereinstimmen (überschriebene equals Methode). Mit der Methode clone lässt sich eine flache Kopie von einer Buch- Instanz erstellen.

Auch hier haben wir uns bewusst dafür entschieden, einen Teil der Logik im *Book*-Modell zu implementieren. Dies wird durch die Attribute *personBorrowed* und *borrowDate* sowie die Methoden *setBorrow(Person personBorrowed)* und *removeBorrow*() verdeutlicht.

2.3 Logik / Buchsystem

Die Geschäftslogik des Buchverleihs wird im *LibrarySystem* umgesetzt. Wie bereits dargestellt, wird ein Teil der Logik auch in den Modellen verarbeitet.

Das *LibrarySystem* hat eine Liste für Bücher, die ausgeliehen werden können, sowie eine Liste von Personen, die im System hinterlegt sind. Wir haben uns dabei bewusst für eine ArrayList entschieden. Diese eignet sich gut, da wir davon ausgehen, dass der Buchverleih klein ist und die Anzahl der Bücher und Personen überschaubar bleibt. ArrayLists bieten in diesem Kontext eine unkomplizierte und effiziente Möglichkeit, unsere dynamischen Sammlungen zu speichern und zu verwalten. Sie ermöglichen uns, dank der geplanten

überschaubaren Anzahl an Büchern und Personen, ein schnelles Hinzufügen, Entfernen und Durchsuchen von Elementen, was ideal für unsere aktuellen Anforderungen ist.

Wir sind uns bewusst, dass bei einem deutlich größeren Buchverleih mit einer umfangreichen Anzahl an Büchern und Mitgliedern eine HashMap eine attraktive Alternative darstellen könnte. Die Möglichkeit, Bücher über eine Buch-ID als Schlüssel effizient zu suchen, zu aktualisieren und zu löschen, wäre in einem solchen Szenario von großem Vorteil. Allerdings haben wir die aktuelle Größe unseres Buchverleihs sorgfältig analysiert und sind zu dem Schluss gekommen, dass ArrayLists für unsere Bedürfnisse die optimale Balance zwischen Performance, Einfachheit und Wartbarkeit bieten.

Wenn eine Person ein Buch ausleiht, wird dies im *LibrarySystem* in einer *Map* gespeichert, die aus der ID der Person und der ID des Buches besteht. Eine *Map* eignet sich hier besonders gut, da sie eine schnelle Zuordnung zwischen einer Person und einem ausgeliehenen Buch ermöglicht. Die Buch-ID wird als Schlüssel verwendet, und die Person-ID dient als Wert. So kann jedes ausgeliehene Buch eindeutig einer Person zugeordnet werden, und eine schnelle Suche oder Modifikation der ausgeliehenen Bücher ist möglich.

Um Bücher und Personen zum Buchverleih hinzuzufügen, stehen die Methoden *addBook* und *addPerson* zur Verfügung. Diese fügen die entsprechenden Objekte der jeweiligen Liste (*books* oder *persons*) hinzu. Um die Daten eines Buch oder einer Person im Nachhinein zu bearbeiten, können die Methoden *editBook* und *editPerson* verwendet werden.

Bei *editPerson* können nur die persönlichen Daten (Vorname, Nachname, Geburtsdatum und Adresse) geändert werden. Änderungen an der Liste der ausgeliehenen Bücher (*borrowedBooks*) müssen über die Funktionen *borrowBook* und *giveBookBack* im *LibrarySystem* vorgenommen werden, um die Konsistenz der Daten zu gewährleisten.

Mit *editBook* können der Titel, der Autor und/oder das Genre eines Buches geändert werden. Andere Buchdaten (wie *isBorrowed*, *isDamaged*, *personBorrowed* und *borrowDate*) können nur über die Funktionen *giveBookBack* gesetzt werden, um die Daten im LibrarySystem korrekt und konsistent zu halten.

Bücher und Personen können über die Funktionen *deleteBook* und *deletePerson* aus den Buchverleih-Listen gelöscht werden. Die jeweiligen Instanzen bleiben jedoch bestehen – sie sind lediglich nicht mehr in den *ArrayLists books* oder *persons* enthalten. Wir haben uns bewusst für dieses Vorgehen entschieden, damit weiterhin mit der Person oder dem Buch gearbeitet werden kann, wenn gewünscht. Es ist zu beachten, dass Personen nur gelöscht werden können, wenn sie keine offenen Gebühren haben und keine Bücher ausgeliehen sind. Ein Buch kann nur gelöscht werden, wenn es nicht ausgeliehen ist.

Um ein Buch auszuleihen, steht die Funktion *borrowBook* zur Verfügung. Soll ein Buch zurückgegeben werden, kann die Funktion *gaveBookBack* verwendet, die gleichzeitig die Funktion *calculateFeeForBook* aufruft, um eventuell anfallende Gebühren zu berechnen.

Außerdem können die Bücher- und Personen-Listen (*books* und *persons*) gefiltert werden. Wir haben uns entschieden, jeweils Filter-Modelle zu verwenden - Klassen, von denen

Instanzen erstellt werden, in denen die Filterkriterien über Getter und Setter gesetzt werden. Dies ermöglicht eine flexible Filterung nach beliebigen Kombinationen von Kriterien.
Personen können dabei nach ausgeliehenen Büchern, offenen Gebühren und der Anzahl der ausgeliehenen Bücher gefiltert werden. Bücher lassen sich nach Genre, Autor und Titel filtern. Der Titel kann dabei auch nach Textbausteinen durchsucht werden.
In der Filterfunktion haben wir mit *Streams* gearbeitet (z.B. *person.stream()*). Dies erzeugt aus der Liste *persons* einen Stream, auf den dann verschiedene Methoden angewendet werden können. In unserem Fall nutzen wir den Filter-Mechanismus mit *.filter(...).* Um den Stream zu beenden und das Ergebnis als Liste zu sammeln, rufen wir .collect(Collectors.toList()) auf. Zusätzlich verwenden wir hier *Lambdas*. Diese ermöglichen es, anonyme Funktionen kompakt zu definieren. Der Parameter *person* wird in die Lambda-Funktion übergeben, auf dem dann Methoden wie *getOpenFees()* aufgerufen werden. Der Stream durchläuft dabei jede Person. Wenn der Ausdruck *true* ergibt, wird die Person in das Ergebnis aufgenommen, andernfalls wird sie aus dem Stream herausgefiltert.

2.4 Anmerkung zu den Tests

Das *errorLogged*-Flag wurde in die Methoden *borrowBook* und *gaveBookBack* eingeführt, um sicherzustellen, dass Fehler, die während des Ausleihens oder Zurückgebens von Büchern auftreten, korrekt protokolliert und überwacht werden. Wenn in einer dieser Methoden eine Ausnahme geworfen wird, beispielsweise bei ungültigen Operationen oder fehlerhaften Zuständen, wird *errorLogged* auf true gesetzt. Dies ermöglicht es, im Testkontext sicherzustellen, dass der Fehler richtig erfasst wird, ohne dass das System unbemerkt bleibt, und bietet so eine zuverlässige Möglichkeit, Fehler zu überwachen und zu testen.

Im Test *testGaveBookBackthrowsException()* überprüfen wir, ob eine Exception geworfen wird. Eine Exception wird geworfen, wenn beim Aufruf von *removeBook* für eine *Personen*-Instanz ein *null*-Argument übergeben wird. Aufgrund der Implementierung der Methoden *addBook* und *borrowBook*, die sicherstellen, dass nur valide Buch-Objekte hinzugefügt und ausgeliehen werden, kann dieser Fall im normalen Ablauf eigentlich nicht auftreten. Allerdings ist es wichtig, dass die *removeBook*-Methode auch bei einem *null*-Argument eine Exception wirft, um unerwartetes Verhalten zu vermeiden. Um dies zu gewährleisten, haben wir in der gaveBookBack-Methode einen try-catch-Block eingefügt. So können wir sicherstellen, dass die Exception wie erwartet behandelt wird.

Die zweite *setBorrow*-Methode im *Book*-Modell wurde speziell hinzugefügt, um die Testbarkeit der Gebührenberechnung (*calculateFeeForBook* im *LibrarySystem*) zu verbessern. Sie ermöglicht es uns, in Testfällen ein beliebiges Ausleihdatum festzulegen, wodurch wir die Gebührenberechnung für verspätete und pünktliche Rückgaben gezielt simulieren können, ohne von der aktuellen Systemzeit abhängig zu sein. Dies ist entscheidend, um verschiedene Testfälle mit präzise gesteuertem Ausleihzeitraum abzudecken.

3. Arbeitsteilung

Die erfolgreiche Umsetzung unseres Projekts basierte auf einer klaren Aufgabenverteilung innerhalb des Teams. Die Tabelle zeigt eine detaillierte Aufschlüsselung der Verantwortlichkeiten und Beiträge jedes einzelnen Mitglieds:

gemeinsam	- Entwurf und Überarbeitung des Klassendiagramms - Entwurf der Ordnerstruktur im Projekt und Anlegen der .java Dateien
Stephanie Wachs	Klassen - Book - LibrarySystem (Grundstruktur) Methoden im LibrarySystem - borrowBook - gaveBookBack - calculateFeeForOneBook - toString Unit- Tests - PersonTest - addPerson, editPerson, deletePerson - Tests im LibrarySystem - personFilter - Tests im LibrarySystem - PersonFilterTest - calculateFeeForOneBook - Tests - borrowBook - Tests - BookTest JavaDocs - LibrarySystem - Book weitere Aufgaben - Korrektur und Hinweise zu dem Code der anderen Projektmitglieder - Ergänzungen in der Dokumentation
Lucas-Manfred Herpe	Klassen - BookFilter Methoden im LibrarySystem - addBook - editBook - deleteBook - filterBook Unit- Tests - BookTest - borrowBook - Tests - gaveBookBack - Tests
Josephina Burger	Klassen - Person - Address - PersonFilter Methoden im LibrarySystem - addPerson - editPerson - deletePerson

	- filterPerson - Vervollständigung der deleteBook-Methode Unit- Tests - addBook, editBook, deleteBook - Tests im LibrarySystem - bookFilter - Tests im Library System - BookFilterTest - gaveBookBack - Tests - AddressTest JavaDocs: - BookFilter - PersonFilter - Person - Address weitere Aufgaben: - Korrektur und Hinweise zu dem Code der anderen Projektmitglieder - Schreiben der Dokumentation

4. Anhang: Klassendiagramm