Software Engineering Übung 06

Design Prinzipien 2

Übung von Jonathan Lippert und Magnus Dierking

Tag der Einreichung: 18. Dezember 2020

Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Software Engineering Übung 06
Design Prinzipien 2

Übung von Jonathan Lippert und Magnus Dierking

Tag der Einreichung: 18. Dezember 2020

Darmstadt

1 Aufgabe

a

i

Siehe Abbildung 1.1

ii

LCOM = 3 von 11

b

Bei erstem Betrachten der Implementation erschien deren Kohäsion nicht optimal zu sein. Drei nicht zusammenfallende Verantwortungen sind alle in einer Klasse enthalten, die noch dazu ein sehr allgemeines Konzept beschreibt, was eine hohe Kohäsion sehr wünschenswert machen würde.
Der berechnete Wert spiegelt dies gut wieder, da er mit 3 auf jeden Fall nicht optimal ist. Angesichts der hohen Methodenanzahl in der Klasse hätte er aber dem Empfinden nach noch höher ausfallen können. Die Verantwortungen der Klasse `Book` sind:

1. Verwaltung der Daten eines einzelnen Buches
2. Verwaltung der Daten Bezüglich der Ausleihe eines einzelnen Buches
3. Exportieren der Daten eines einzelnen Buches als csv-Datei

Der ermittelte **LCOM**- Wert deckt sich also hier genau mit der Anzahl der Verantwortungen der Klasse.

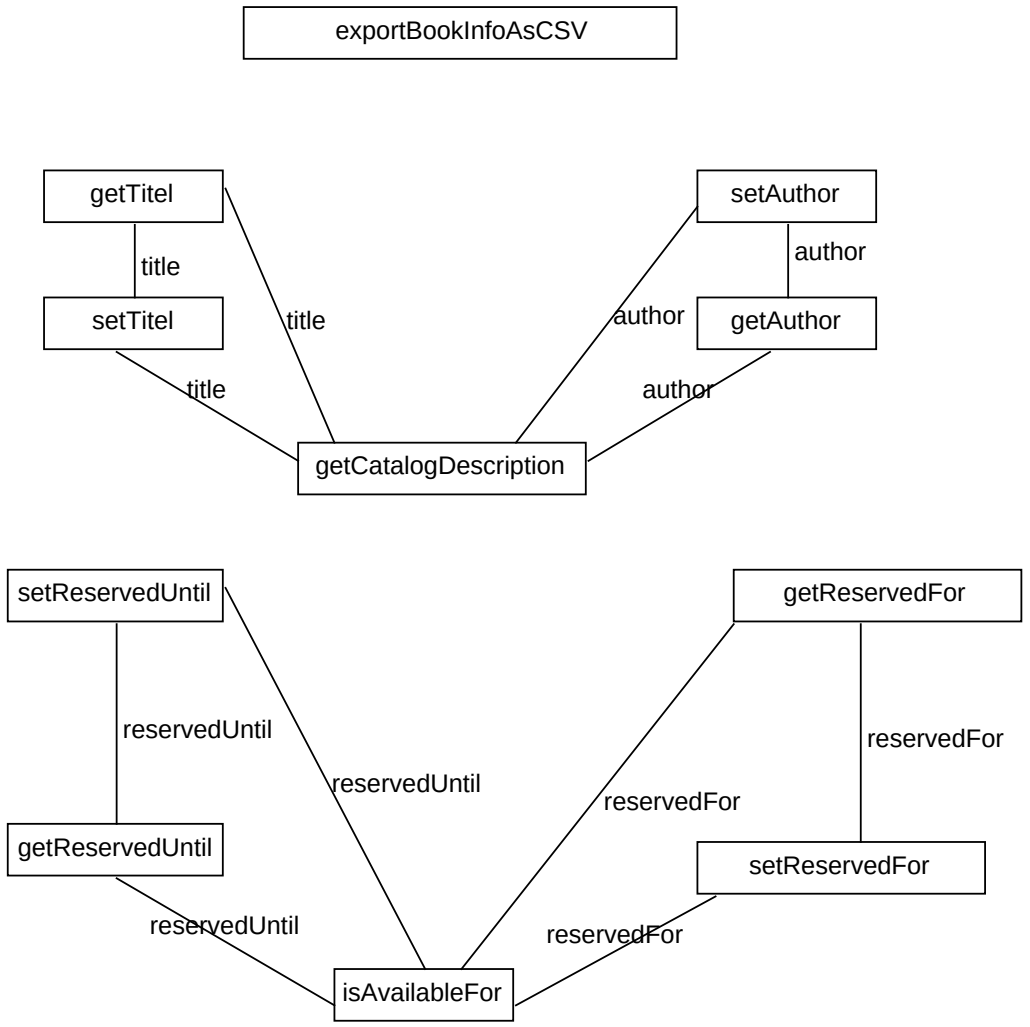

Abbildung 1.1: Zusammenhangsgraph der Klasse Book für das LCOM Verfahren.

c

Listing 1.1: Klasse Book

```
1  package org.library;
2
3  import java.util.Date;
4
5  import org.library.users.Client;
6
7  public class Book implements LibraryItem {
8
9    private String title;
10   private String author;
```

```

  public Book(String title, String author){
    this.title = title;
    this.author=author;
  }

  // getters
  public String getTitle() { return title; }

  public String getAuthor() { return author; }

  // implementiert interface LibraryItem
  @Override
  public String getCatalogDescription() {
  return title + " by " + author;
  }
}
```

Listing 1.2: Klasse Reservation

```
package org.library;
import java.util.Date;
import org.library.users.Client;

public class Reservation {

  private Client reservedFor;
  private Date reservedUntil;
  private Book book;

  public Reservation(Client reservedFor,Date reservedUntil, Book book){
    this.reservedFor=reservedFor;
    this.reservedUntil = reservedUntil;
    this.book = book;
  }
}
```

Listing 1.3: RerservationManager

```
  package org.library;
  import java.util.Date;
  import org.library.users.Client;

  public class RerservationManager {


  public static boolean isAvailableFor(Client c, Date from) {
  if (reservedFor != null || reservedUntil.after(from)) {
  return false;
  }
  return c.numberOfBorrowedBooks() < 3;
  }

```

```
15  }
```

Listing 1.4: BookExporter

```
1  package org.library;
2
3  public class BookExporter{
4    public static void exportBookInfoAsCSV(org.library.formats.CSVExporter
         exporter, Book book) throws java.io.IOException {
5      exporter.writeLine(book.getTitle(), book.getAuthor());
6    }
7  }
```

d

Würde die beschrieben Methode implementiert werden, so ergäbe das **LCOM**-Verfahren stets einen niedrigeren Wert. Jede Methode die auf irgendein Attribut der Klasse zugreift wird durch die neue Methode in eine einzige Zusammenhangs-Komponente integriert. Lediglich Klassen, die kein Attribut direkt ansprechen wie `exportBookInfoAsCSV()` sind hiervon nicht betroffen. Unter diesen Umständen wäre ein Wert über 1 also inakzeptabel und würde sofortiges Refactoring nach sich ziehen. Für eine genauere Aussage über die Kohäsion müsste Debug-Methoden aber aus dem Graph entfernt werden, da sie die Aussagekraft des Ergebnisses stark senken.