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. Die Einteilung Anzahl der Responsibilities = LCOM-Ergebnis ist sicherlich oft zutreffend, bei sehr schlechtem Code muss sie aber auch nicht zwangsläufig zutreffen. Mehrere Verantwortungen können sich durchaus überschneiden (sollten es aber generell nicht) oder die Methoden einzelner Verantwortungen können auch in keinerlei Bezug stehen (ebenfalls zu vermeiden).

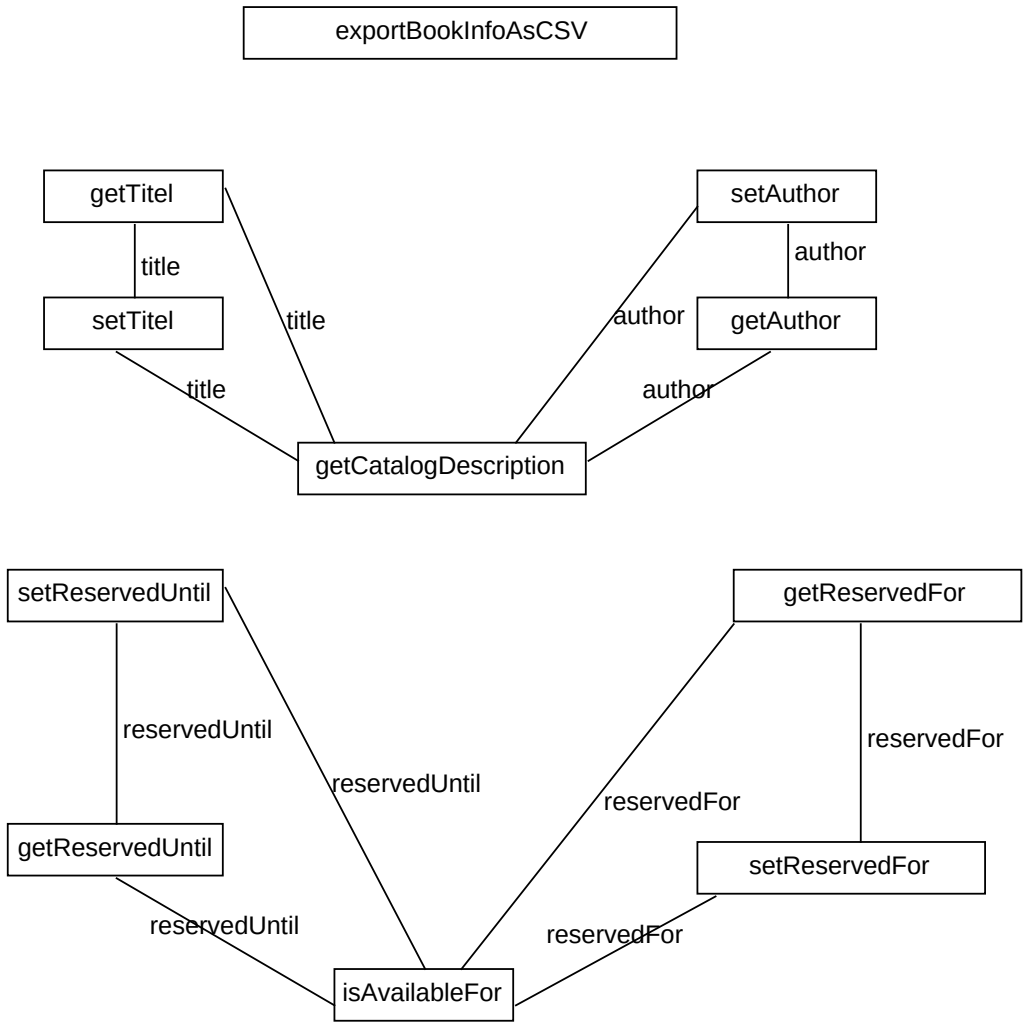

Abbildung 1.1: Zusammenhangsgraph der Klasse Book für das LCOM Verfahren.

c

```
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;
11
12 private Client reservedFor;
```

```
13 private Date reservedUntil;
14
15
16 // getters an setters
17 public String getTitle() { return title; }
18 public void setTitle(String title) { this.title = title; }
19
20 public String getAuthor() { return author; }
21 public void setAuthor(String author) { this.author = author; }
22
23 public Client getReservedFor() { return reservedFor; }
24 public void setReservedFor(Client reservedFor) { this.reservedFor = reservedFor; }
25
26 public Date getReservedUntil() { return reservedUntil; }
27 public void setReservedUntil(Date until) { this.reservedUntil = until; }
28
29 // implementiert interface LibraryItem
30 @Override
31 public String getCatalogDescription() {
32 return title + " by " + author;
33 }
34
35 // prueft, ob Buch vom Kunden geliehen werden kann
36 public boolean isAvailableFor(Client c, Date from) {
37 if (reservedFor != null || reservedUntil.after(from)) {
38 return false;
39 }
40 return c.numberOfBorrowedBooks() < 3;
41 }
42
43 // exportiert Buchinformation als csv Datei
44 public void exportBookInfoAsCSV(org.library.formats.CSVExporter exporter) throws
       java.io.IOException {
45 exporter.writeLine(getTitle(), getAuthor());
46 }
47 }
```

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.