01584: Grundpraktikum Programmierung

durchgeführt vom
Lehrgebiet Unternehmensweite Softwaresysteme
im Sommersemester 2020

Aufgabenstellung

Inhaltsverzeichnis

1. Einleitung

Willkommen beim Programmierpraktikum im Sommersemester 2020. Sie haben sich erfolgreich angemeldet und können somit mit der Bearbeitung der Praktikumsaufgabe beginnen.

Die Aufgabe besteht darin, ein Verfahren zu konzipieren und zu implementieren, das die vollständige Ausleuchtung eines gegebenen Raumes mit einer möglichst geringen Anzahl von Lampen ermöglicht.

Im Rahmen des Programmierpraktikums wird sehr viel Wert auf das Verständnis der Aufgabenstellung sowie auf eine systematische Herangehensweise gelegt. Neben der eigentlichen Aufgabenstellung werden Ihnen deshalb wichtige und nützliche Hinweise zum Vorgehen in den nachfolgenden Kapiteln dieses Dokuments gegeben. Um Ihnen den Einstieg zu erleichtern, wird Ihnen ein Musterworkspace zur Verfügung gestellt, in dem bereits bestimmte Architekturvorgaben implementiert sind.

Wir empfehlen Ihnen eine gute Planung der einzelnen Aktivitäten des Programmierpraktikums, bevor Sie mit der Implementierung beginnen. Eine solche Zeitplanung ist nützlich, um laufend Ihren Fortschritt überprüfen zu können und bei Verzögerungen rechtzeitig geeignete Gegenmaßnahmen zu ergreifen.

Um etwaige Fragen im Zusammenhang mit der Programmiersprache Java und eventuelle Schwierigkeiten mit der Aufgabenstellung wird sich vor allen Dingen Frau Frank kümmern. Bei organisatorischen oder sonstigen Probleme können Sie ebenfalls Frau Frank ansprechen.

Wir wünschen Ihnen viel Erfolg beim Programmierpraktikum!

2. Organisation und Abgaben

2.1. Betreuung

Die Betreuung des Praktikums erfolgt über die folgenden zwei Newsgroups.

In der Betreuungsnewsgroup **feu.informatik.kurs.1584.betreuung.ss** werden Ankündigungen und allgemeine Informationen, beispielsweise zu Änderungen des organisatorischen Ablaufs, bekanntgegeben. Schauen Sie mindestens einmal pro Woche in diese Newsgroup, um sich über aktuelle Mitteilungen zu informieren.

Verwenden Sie die Diskussionsnewsgroup **feu.informatik.kurs.1584.diskussion.ss**, wenn Sie Fragen zum Praktikum haben oder anderen Studierenden helfen möchten. In dieser Newsgroup können alle technischen und inhaltlichen Probleme diskutiert werden.

Die Praktikumsbetreuung ist über die E-Mail-Adresse **propra@fernuni-hagen.de** zu erreichen. Fragen zur Programmierung werden grundsätzlich nicht per E-Mail beantwortet. Nutzen Sie dafür ausschließlich die Diskussionsnewsgroup, damit andere Studierende ebenfalls von der Diskussion profitieren können.

2.2. Terminplan

Die einzelnen Termine sowie die zum Termin zu erbringenden Leistungen werden im Folgenden detailliert benannt.

Datum	Termin	zu erbringende Leistungen
01.04.2020	Ausgabe der Aufgabenstellung und offizieller Arbeitsbeginn	
20.07.2020, 23:59 Uhr	Abgabe des kompletten Programms	- ausführbares Programm - Quellcode digital und in ausgedruckter Form - Dokumentation im Programmcode
21.07.2020 bis 13.08.2020	Bewertung der eingereichten Lösungen	
18.09.2020	Präsenzphase in Hagen	- Testat, mündliche Prüfung

2.3. Präsenztag

Die genaue Terminplanung für die Präsenzphase erfolgt nach Bewertung der Programme. Falls wir bei der Bewertung feststellen, dass Ihre Lösung nicht den Mindestanforderungen genügt, werden Sie nicht zur Präsenzphase eingeladen. Die Implementierung wird mit bis zu 100 erreichbaren Punkten bewertet. Die Mindestanforderungen gelten als erfüllt, wenn mindestens 50 Punkte erreicht wurden. Für alle organisatorischen Fragen bezüglich des Präsenztages wenden Sie sich bitte an die E-Mail-Adresse **propra@fernuni-hagen.de**.

Der Präsenztag beinhaltet ein Testat in schriftlicher Form im Umfang von 60 Minuten. Mit dem Testat wird anhand (einfacher) Aufgaben überprüft, ob Sie über die für die Implementierung der Lösung erforderlichen Programmierkenntnisse in Java verfügen. Wenn Sie Ihr Pro-

gramm selbständig konzipiert und implementiert haben, wird Ihnen das Testat keine Probleme bereiten.

2.4. Abgabe des Programms

Beim Programmierpraktikum handelt es sich um eine benotete Prüfungsleistung. Für das Bestehen des Praktikums erwarten wir von Ihnen die folgenden drei Dinge:

1. Eigenständige und fehlerfreie Implementierung des in der Aufgabenstellung beschriebenen Programms. **Gruppenarbeiten sind unzulässig**.
2. Termingerechte Abgabe der implementierten und dokumentierten Lösung. Die Abgabe des Programms muss neben der Implementierung den unten genannten Anforderungen genügen. Genauere Informationen zu der Abgabe erhalten Sie von uns per E-Mail.
3. Sie müssen uns zeigen, dass Sie das von Ihnen eingereichte Programm selbst entwickelt und verstanden haben. Dazu müssen Sie am Präsenztag die zuvor beschriebenen Leistungen erfolgreich erbringen.

Neben der Abgabe des Programms sind zusätzlich Ausdrucke der von Ihnen geschriebenen Quelltexte postalisch einzureichen. Die Einsendung hat ebenfalls bis zum oben genannten Abgabetag zu erfolgen. Es gilt das Datum des Poststempels. Legen Sie Ihrer Sendung bitte das vollständig ausgefüllte Deckblatt, das Sie im Anhang der Aufgabenstellung finden, bei.

Die Abgabe des Programms umfasst ein einziges ausführbares JAR-Archiv, den implementierten Musterworkspace und die dazugehörige Dokumentation im Programmcode. Die einzelnen Dateien der Abgabe müssen als ZIP-Archiv abgegeben werden. Vermeiden Sie Umlaute im Dateinamen. Das Archiv hat zwingend der folgenden Namenskonvention sowie der angegebenen hierarchischen Struktur zu folgen:

- Archivname: **<Matrikelnummer_Name_Vorname>.zip**
    - Ausführbares JAR-Archiv: **ProPra.jar**
    - Quellcodeverzeichnis: **Workspace** .

Das von Ihnen entwickelte Programm muss mit Hilfe des Befehls **Java -jar ProPra.jar** aufrufbar sein. Es ist empfehlenswert, den in der Entwicklungsumgebung verfügbaren Exportmechanismus zur Erstellung des ausführbaren Programms zu verwenden.

Die geforderten Kommandozeilenparameter müssen in der Form „**Java -jar ProPra.jar param1=value1 param2=value2 param3=value3**“ angegeben werden können.

Manipulationsversuche mit den eingesandten Programmen, z.B. durch Aufspielen von Computerviren oder anderen Programmkomponenten, die nicht der Lösung der Programmieraufgabe dienen, führen - unabhängig von Schadensersatzansprüchen seitens der FernUniversität – dazu, dass das Praktikum als nicht bestanden gewertet wird.

Aus gegebenem Anlass müssen wir darauf hinweisen, dass das Kopieren von Quellcode aus ähnlichen Projekten aus dem Internet unter keinen Umständen gestattet ist und mit einem so-

fortigen Ausschluss geahndet wird. Wir besitzen sowohl die entsprechenden Werkzeuge als auch die notwendige Sachkenntnis, um solche 'Refactorings' zu erkennen.

3. Vollständige Ausleuchtung

3.1. Motivation und Aufgabenstellung

In einem Raum sollen Lampen so platziert werden, dass alle Stellen des Raumes beleuchtet werden. Räume sind begrenzt durch rechtwinklig zueinander stehende Wände. Im Grundriss ergibt sich ein Polygon, das beliebig verwinkelt sein kann. Ein Raum ist dann vollständig beleuchtet, wenn sowohl alle Wände als auch alle Flächen vollständig beleuchtet sind. Es darf innerhalb des Raumes keine Flächen geben, die im Schatten liegen.

Die Ausleuchtung soll effizient sein, das heißt, sie soll mit möglichst wenigen Lampen erreicht werden. Lampen sind punktförmige Lichtquellen, die in alle Richtungen strahlen. Sie können frei im Raum oder auch direkt an Ecken oder Wänden positioniert werden. Es gibt diesbezüglich keine weiteren Restriktionen wie etwa vorgegebene Deckenraster oder ästhetische Vorgaben. Vereinfachend wird davon ausgegangen, dass Lichtstrahlen von einer Lampe ausgehend in alle Richtungen unendlich weit reichen, bis sie auf eine Wand treffen. Möbel und sonstige Einrichtungsgegenstände sind nicht zu berücksichtigen. Eine Stelle im Raum ist beleuchtet, wenn ein Lichtstrahl von mindestens einer Lampe ungehindert zu dieser Stelle gelangen kann.

Die folgende Abbildung 1 soll die Problemstellung anhand eines Beispiels verdeutlichen.

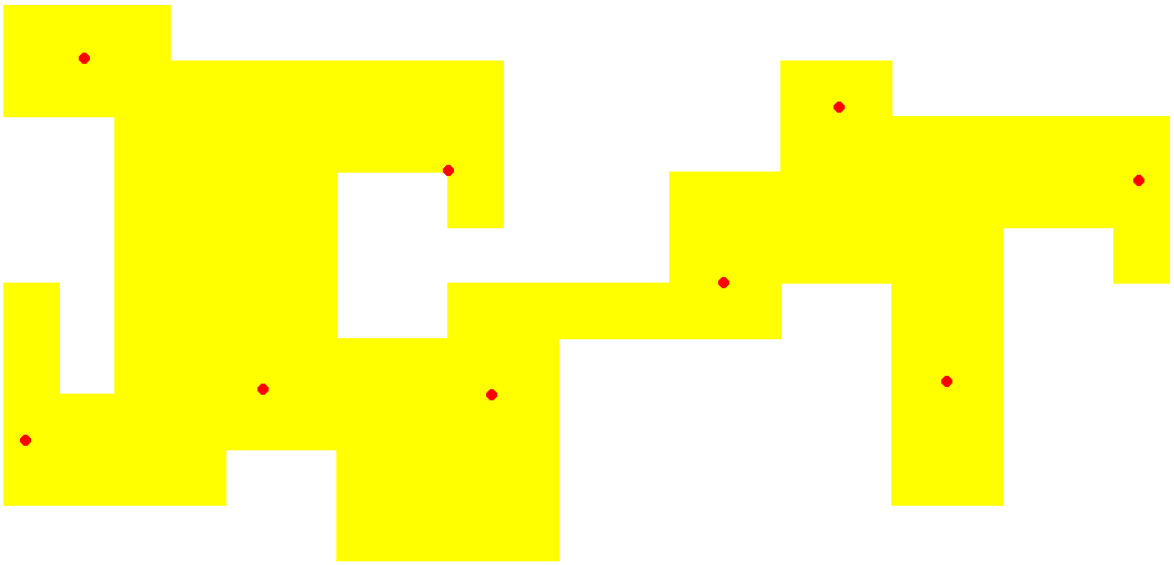

Abbildung 1: Grundriss eines ausgeleuchteten Raums

Abbildung 1 zeigt den Grundriss eines vollständig ausgeleuchteten Raumes. In diesem Beispiel wird eine vollständige Ausleuchtung durch den Einsatz von neun Lampen erreicht. Die Positionen der verwendeten Lampen sind durch rote Punkte gekennzeichnet. Es sei ange-

merkt, dass der hier dargestellte Raum auch bereits mit sechs Lampen vollständig ausgeleuchtet werden kann.

Im Programmierpraktikum sollen Sie ein Verfahren zur Bestimmung einer vollständigen Ausleuchtung mit einer geringen Anzahl von Lampen entwickeln. Die konkrete Form der zu beleuchtenden Räume ist im Voraus nicht bekannt. Dementsprechend soll das Verfahren allgemein für alle Räume mit im rechten Winkel zueinander stehenden Wänden anwendbar sein.

Folgende Anforderungen werden im Rahmen des Programmierpraktikums an Sie gestellt:

- Sie beschäftigen sich bei der Lösung der Programmieraufgabe mit der Schwierigkeit des Algorithmenentwurfs und kommen dabei insbesondere mit der Größe des Lösungsraums in Kontakt. Da der Lösungsraum typischerweise sehr groß ist, ist eine geeignete Heuristik zu entwickeln, die es ermöglicht, eine zulässige Lösung in akzeptabler Zeit zu finden. Der Entwurf dieser Heuristik erfordert eine intensive Auseinandersetzung mit der Problemstellung.
- Unter Anwendung der Prinzipien der objektorientierten Programmierung ist der Softwareentwurf so zu konzipieren, dass das Lösungsverfahren an möglichst vielen Stellen durch Konfiguration und Implementierung von Erweiterungspunkten anpassbar ist. Dadurch soll die Möglichkeit geschaffen werden, verschiedene Heuristiken experimentell zu untersuchen.
- Bei der Implementierung des Lösungsverfahrens ist dessen Effizienz zu berücksichtigen, um eine qualitativ hochwertige zulässige Lösung möglichst schnell zu ermitteln. Dazu sind zum einen geeignete Datenstrukturen zur Speicherung von Laufzeitinformationen zu verwenden. Andererseits ist bei der Implementierung des Algorithmus selbst und bei der Lösung von Unterproblemen sowie bei der Anwendung von Heuristiken auf eine möglichst zeiteffiziente Abarbeitung zu achten.
- Dem Prinzip eines guten Softwareentwurfs folgend, sind die Bestandteile des Lösungsverfahrens durch geeignete Testklassen abzusichern. Zur Ausführung der Tests wird ein verbreitetes Rahmenwerk zum Komponententest verwendet.
- Es ist eine Dateneingabe und -ausgabe auf der Basis von XML-Dokumenten zu realisieren.
- Eine ermittelte Lösung ist in einer grafischen Ausgabe darzustellen.
- Das Lösungsverfahren ist in ein Hauptprogramm einzubetten. Die Konfiguration des Programms erfolgt vollständig über die einzulesende Eingabedatei und durch Kommandozeilenparameter. Eine grafische Benutzeroberfläche zur Bedienung des Hauptprogramms ist daher nicht erforderlich.

Nachfolgend wird die Aufgabenstellung konkretisiert. Verschiedene Hinweise zur Lösung werden in Abschnitt 3.2 gegeben, während in Kapitel 4 Hinweise zur Durchführung des Programmierpraktikums zu finden sind.

3.2. Konkretisierung der Aufgabenstellung

Im Programmierpraktikum ist ein Lösungsverfahren zur Bestimmung einer effizienten Ausleuchtung eines Raumes zu konzipieren und zu implementieren. Das Lösungsverfahren soll den in Abschnitt 3.1 genannten funktionalen und nicht-funktionalen Anforderungen genügen.

Das zu entwickelnde Lösungsverfahren ist in ein Hauptprogramm einzubetten, das eine Probleminstanz einliest, gegebenenfalls das Lösungsverfahren startet und die Lösung in einer grafischen Darstellung anzeigt.

Die Anforderungen an die zu implementierenden Bestandteile werden im Folgenden genauer beschrieben. Das erfolgreiche Bestehen des Programmierpraktikums erfordert zwingend die vollständige Umsetzung der geforderten Funktionalitäten.

3.2.1. Lösungsrepräsentation

Es ist eine programminterne Repräsentation des gegebenen Raumes zu entwickeln. Die Repräsentation soll dabei insbesondere eine effiziente Ausführung des Lösungsverfahrens unterstützen. Es ist dabei zu beachten, dass es eventuell erforderlich ist, die Lösungsrepräsentation während der Entwicklungsphase anzupassen.

↳ Interface ?

Ein Raum wird durch die geordnete Menge seiner Eckpunkte definiert. Jeder Punkt wird durch seine X- und Y-Koordinaten bestimmt. Zwischen zwei benachbarten Eckpunkten liegt jeweils eine gerade Wand. Benachbarte Wände stoßen an ihrem gemeinsamen Eckpunkt im rechten Winkel aufeinander. Folgt man den Eckpunkten eines Raumes, findet man somit abwechselnd senkrechte und waagerechte Wände.

Die Dateneingabe und Datenausgabe erfolgt XML-basiert. Eine Probleminstanz wird mit dem Wurzelelement <Raum> definiert. Ein Raum wird durch eine eindeutige <ID> identifiziert. Unterhalb einer Aggregation <ecken> sind Elemente <Ecke> mit ihren Koordinaten <x> und <y> enthalten. In der XML-Datei direkt aufeinander folgende Ecken sind im Raum benachbart. Zusätzlich sind die erste und letzte Ecke benachbart. Die Ecken können entweder mit oder entgegen dem Uhrzeigersinn sortiert sein.

Die als Lösung identifizierte Menge von Lampen inklusive deren Positionen wird unterhalb des optionalen Elements <lampen> abgelegt. Zur Speicherung werden Elemente vom Typ <Lampe> verwendet, deren Koordinaten in Elementen <x> und <y> gespeichert werden.

Listing 1 zeigt die zugehörige Dokumenttypdefinition (DTD). Die angegebene DTD legt die Dokumentstruktur fest, die von den Dokumenten unbedingt einzuhalten ist.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <!ELEMENT Raum (ID, ecken, lampen?)>
03 <!ELEMENT ID (#PCDATA)>
04 <!ELEMENT ecken (Ecke*)>
05 <!ELEMENT lampen (Lampe*)>
06 <!ELEMENT Ecke (x, y)>
07 <!ELEMENT Lampe (x, y)>
08 <!ELEMENT x (#PCDATA)>
09 <!ELEMENT y (#PCDATA)>
```

Listing 1: DTD zur Angabe von Probleminstanzen und Lösungen

Als Beispiel wird ein Raum mit zehn Ecken und einer zulässigen aber nicht optimalen Lösung mit zwei Lampen angegeben.

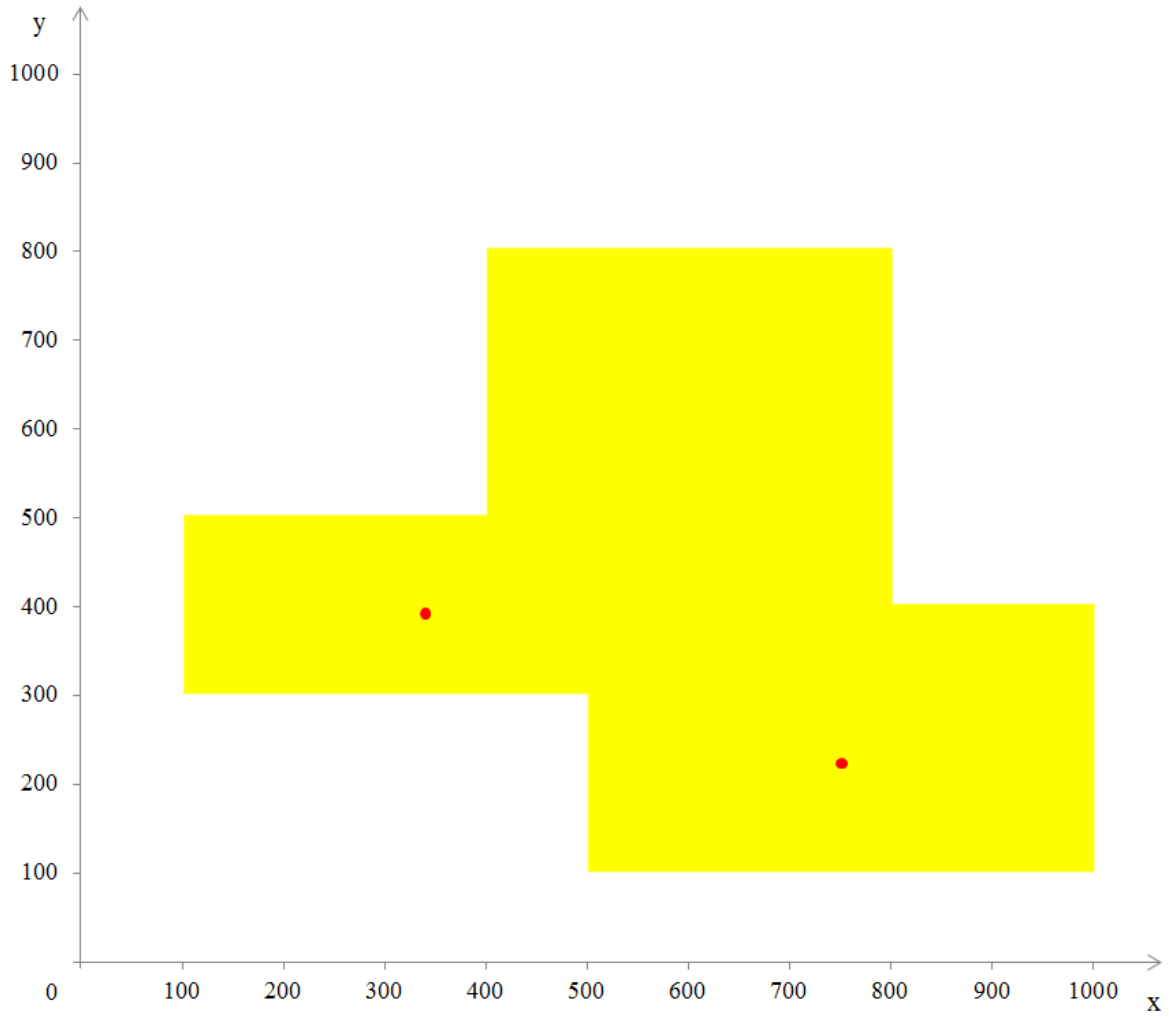

Abbildung 2: Beispiel Raum mit Lösung

Die in Abbildung 2 gezeigte Probleminstanz und die Lösung werden durch das in XML-Dokument in Listing 2 repräsentiert.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <Raum>
03     <ID>2</ID>
04     <ecken>
05         <Ecke>
06             <x>100</x>
07             <y>300</y>
08         </Ecke>
09         <Ecke>
10             <x>500</x>
11             <y>300</y>
12         </Ecke>
13         <Ecke>
14             <x>500</x>
15             <y>100</y>
16         </Ecke>
17         <Ecke>
18             <x>1000</x>
19             <y>100</y>
20         </Ecke>
21         <Ecke>
22             <x>1000</x>
23             <y>400</y>
24         </Ecke>
25         <Ecke>
26             <x>800</x>
27             <y>400</y>
28         </Ecke>
29         <Ecke>
30             <x>800</x>
31             <y>800</y>
32         </Ecke>
33         <Ecke>
34             <x>400</x>
35             <y>800</y>
36         </Ecke>
37         <Ecke>
38             <x>400</x>
39             <y>500</y>
40         </Ecke>
41         <Ecke>
42             <x>100</x>
43             <y>500</y>
44         </Ecke>
45     </ecken>
46     <lampen>
47         <Lampe>
48             <x>338.46</x>
49             <y>392.31</y>
50         </Lampe>
51         <Lampe>
```

```
52              <x>751.15</x>
53              <y>225.03</y>
54          </Lampe>
55      </lampen>
56 </Raum>
```

Listing 2: Gültiges XML-Dokument für eine Probleminstanz mit Lösung

Eine Probleminstanz, d.h. die Definition eines Raumes, wird ebenfalls durch die in Listing 1 angegebene DTD repräsentiert. Das zugehörige XML-Dokument, das ausschließlich eine Probleminstanz enthält, beinhaltet dann aber nicht das optionale Element <lampen> und dessen Kindelemente.

3.2.2. Lösungsverfahren

Lampen können überall im Raum sowie auf Ecken und Wänden platziert werden. Damit ist die Menge möglicher Positionen unendlich groß. Ein vollständiges Durchsuchen des Lösungsraums, d.h. ein Betrachten aller möglichen Kombinationen von Lampen, ist aus diesem Grund nicht möglich.

Es ist stattdessen zunächst erforderlich, geeignete Kandidaten für Positionen von Lampen zu bestimmen. Die Kandidaten sind so auszuwählen, dass Lampen an den Positionen bestimmte Teilbereiche des Raumes ausleuchten. Ihre Anzahl sollte nicht zu groß sein, damit die Auswahl der Positionen für eine vollständige Ausleuchtung des Raumes im nachfolgenden Schritt schneller durchgeführt werden kann.

Für die Kandidatenauswahl bieten sich verschiedene Vorgehensweisen an, die sich im Hinblick auf Rechenaufwand und Qualität in Bezug auf eine möglichst geringe Anzahl benötigter Lampen unterscheiden können. Um in vertretbarer Rechenzeit eine Lösung zu finden, ist es empfehlenswert, dem im Folgenden vorgeschlagenen Verfahren zu folgen.

Zunächst werden alle Teilrechtecke des Raumes betrachtet. Jedes Teilrechteck enthält alle Punkte, von denen es vollständig beleuchtet werden kann. Durch die Bildung der Schnittflächen können Lampenpositionen gefunden werden, von denen aus möglichst viele Teilrechtecke des Raumes beleuchtet werden können.

Das gewählte Verfahren vereinfacht das Problem insofern, dass eine diagonale Ausbreitung des Lichts einer Lampe in Teilrechtecke des Raumes, in denen diese Lampe nicht enthalten ist, nicht berücksichtigt wird. Dies führt dazu, dass bei einigen Probleminstanzen möglicherweise nicht die Lösung mit der tatsächlich minimalen Anzahl von Lampen gefunden werden kann. Andererseits ermöglicht diese Vereinfachung eine deutliche Reduzierung der Rechenzeit.

Für die Bestimmung der Teilbereiche des Raumes ist es sinnvoll, aus den gegebenen Eckpunkten die Wände abzuleiten und diese geordnet nach ihrer Orientierung (Nord, Ost, Süd, West) in entsprechenden Containern zu speichern. Die Teilbereiche des Raumes findet man von den verschieden orientierten Wänden ausgehend in der im Folgenden beschriebenen Weise.

Von den Nordwänden ausgehend bestimmt man zunächst die nordwestliche Ecke des Teilbereichs, die durch den Schnittpunkt der Nordwand oder ihrer gedachten Verlängerung mit der nächstgelegenen Westwand gegeben ist. Anschließend bestimmt man den Schnittpunkt der Nordwand oder ihrer gedachten Verlängerung mit der nächstgelegenen Ostwand als den nordöstlichen Eckpunkt. Mit den Schnittpunkten der von den Eckpunkten ausgehenden senkrechten Geraden und der nächstgelegenen Südwand bzw. deren gedachter Verlängerung können schließlich die beiden südlichen Eckpunkte ermittelt werden. Das Vorgehen bei der von den West-, Süd- und Ostwänden ausgehenden Suche ist analog. Teilrechtecke sollen von allen Wänden des Raumes ausgehend ermittelt werden. Abbildung 3 zeigt exemplarisch einige auf diese Weise ermittelte Teilbereiche.

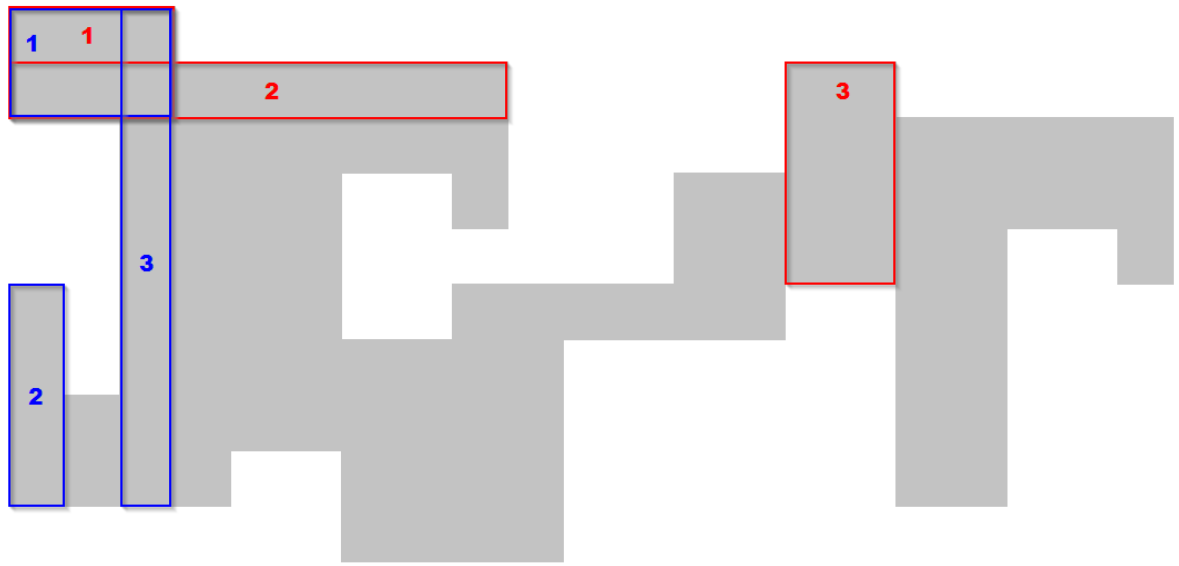

Abbildung 3: Bestimmung von Teilbereichen

In Abbildung 3 sind der Übersichtlichkeit halber nur die Teilbereiche abgebildet, die sich ergeben, wenn man die Rechtecke von den jeweils ersten drei Nord- bzw. Westwänden ausgehend ermittelt. Die roten Rechtecke ergeben sich bei der von den Nordwänden ausgehenden Suche. Von den Westwänden ausgehend, findet man die blauen Teilbereiche.

Geeignete Kandidaten für Positionen von Lampen sind solche Bereiche, die gemeinsame Schnittflächen mit möglichst vielen Rechtecken aufweisen. Abbildung 4 zeigt einen Ausschnitt des oben dargestellten Raumes.

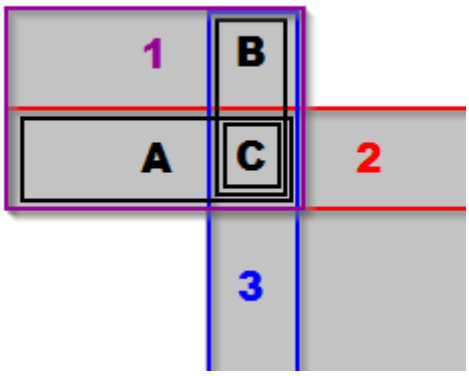

Abbildung 4: Kandidaten für Positionen von Lampen

In dem Ausschnitt findet man die Schnittfläche A, von der aus die beiden Rechtecke 1 und 2 vollständig beleuchtet werden können. Aus der Schnittfläche B heraus können die Rechtecke 1 und 3 vollständig beleuchtet werden. Es handelt sich hierbei allerdings nicht um geeignete Kandidaten für die Position einer Lampe. Die Rechtecke 1, 2 und 3 können nämlich ebenso gut aus der Schnittfläche C heraus beleuchtet werden. Schnittfläche C dominiert in dieser Hinsicht die Schnittflächen A und B. Nur Schnittfläche C stellt in diesem Ausschnitt einen Kandidaten für die Position einer Lampe dar.

Zur Identifizierung der Kandidaten müssen Sie also Schnittflächen der im vorigen Schritt ermittelten Teilrechtecke bilden und bezüglich der sie schneidenden Teilrechtecke miteinander vergleichen. Nur dann, wenn die Menge der geschnittenen Rechtecke einer Fläche keine Teilmenge der Menge der geschnittenen Rechtecke einer anderen Fläche ist, handelt es sich um einen Kandidaten.

Im nachfolgenden Schritt suchen Sie die kleinste Kombination von Kandidaten, an deren Positionen platzierte Lampen zusammen alle Rechtecke des Raumes vollständig beleuchten. Auch wenn die Menge möglicher Positionen für Lampen derart eingeschränkt wird, kann der Lösungsraum abhängig von der Probleminstanz sehr groß werden. Eine vollständige Betrachtung aller möglichen Kombinationen von Lampen ist daher oft nicht mit vertretbarem Zeitaufwand realisierbar. Implementieren Sie stattdessen einen auf Backtracking basierenden Algorithmus.

Für die Beschreibung des Algorithmus gehen wir davon aus, dass an jeder Position aus der Menge der Kandidaten eine Lampe platziert ist, die entweder ein- oder ausgeschaltet sein kann. Eine Funktion prüft auf Basis der Zustände der Lampen, ob der Raum vollständig ausgeleuchtet und somit eine zulässige Lösung gefunden wurde. Ist dies nicht der Fall, kommt es zu rekursiven Aufrufen der Funktion mit einer Lampe in ein- beziehungsweise ausgeschaltetem Zustand. Falls bereits eine Lösung mit einer geringeren oder gleich großen Anzahl eingeschalteter Lampen gefunden wurde, findet keine weitere Rekursion statt. Eine grobe Beschreibung des Algorithmus ist in Listing 3 angegeben. Weitere Informationen und Beispiele zu Backtracking-Algorithmen sind in [4] zu finden.

```
01 INPUT Lampen          : Collection<Lampe>
02       Idx             : Integer                //Lampen-Index
03       Beleuchtet      : Collection<Integer>    //Bel. Rechtecke
04
05 PROCEDURE sucheLösung( Lampen, Idx, Beleuchtet ) {
06     if(Alle Rechtecke des Raumes sind in Beleuchtet enthalten) {
07         Merke Lampen  //zulässige Lösung gefunden
08     } else {
09         if(In Lampen sind weniger Lampen eingeschaltet
10            als bei der bisher besten gefundenen Lösung) {
11             Lampen[Idx] einschalten
12             Beleuchtet += Lampen[Idx].beleuchtet
13             sucheLösung( Lampen, (Idx+1), Beleuchtet )
14             Lampen[Idx] ausschalten
15             Beleuchtet -= Lampen[Idx].beleuchtet
16             sucheLösung( Lampen, (Idx+1), Beleuchtet )
17     }
```

Listing 3: Backtracking-Algorithmus zur Auswahl von Lampenpositionen

Die Softwarearchitektur soll die leichte Änderbarkeit des Lösungsverfahrens gewährleisten. Entwerfen Sie eine entsprechende Softwarearchitektur unter Berücksichtigung der Prinzipien der objektorientierten Softwareentwicklung. Beachten Sie, dass das Lösungsverfahren so effizient wie möglich zu implementieren ist. Dazu zählt beispielsweise, dass geeignete Datenstrukturen verwendet werden und auch Algorithmen zur Lösung von Unterproblemen weder Speicher- noch Zeitressourcen verschwenden.

3.2.3. Grafische Ausgabe

Die zuvor ermittelte beziehungsweise eingelesene Lösung soll in einem Fenster grafisch dargestellt werden. Erstellen Sie dazu ein Fenster, in dem der Grundriss des durch die Probleminstanz beschriebenen Raumes sowie die Positionen der Lampen dargestellt werden. Der Grundriss ist sinnvoll skaliert darzustellen, um die Anzeige des gesamten Raumes innerhalb eines Fensters zu ermöglichen. In dem Dialogfenster soll außerdem in Textform die Anzahl der Lampen ausgegeben werden. Die Anwendung soll unter Verwendung des Kreuzes im roten Rechteck am rechten oberen Fensterrand geschlossen werden. An die grafische Ausgabe werden ansonsten keine weiteren Anforderungen gestellt.

3.2.4. Hauptprogramm

Das Hauptprogramm wird über drei Kommandozeilenparameter gesteuert. Der erste Parameter steuert den Ablauf des Hauptprogramms und wird weiter unten genau spezifiziert. Der zweite Parameter identifiziert die zu verarbeitende XML-Datei, die eine Probleminstanz und optional eine zur Probleminstanz zugehörige Lösung enthält. Als dritter Parameter wird ein Zeitlimit in Sekunden als Abbruchkriterium für den Algorithmus übergeben.

Für den Ablaufparameter **r** wird folgende Festlegung getroffen:

- „s“ (solve): für die durch die XML-Datei beschriebene Probleminstanz wird eine Lösung ermittelt. Die Positionen der Lampen werden in der angegebenen XML-Datei gespeichert. Wenn in der XML-Datei bereits eine Lösung enthalten ist, so ist diese zu überschreiben.
- „sd“ (solve & display): wie „s“, nur dass der Raum sowie die ermittelten Positionen der Lampen zusätzlich in der grafischen Oberfläche angezeigt werden.
- „v“ (validate): durch diese Option wird geprüft, ob der in der angegebenen XML-Datei enthaltene Raum durch die ebenso dort angegebenen Lampen vollständig ausgeleuchtet ist. Die Validierung erfolgt nach den in Unterabschnitt 3.2.2 beschriebenen Vorgaben. Das bedeutet, dass Lösungen, die lediglich bei Berücksichtigung von diagonaler Ausbreitung des Lichts zulässig sind, als unzulässig betrachtet werden. Das Ergebnis der Prüfung sowie die Anzahl und Positionen der Lampen werden ausgegeben. Falls die angegebene XML-Datei keinen zulässigen Raum enthält, wird eine Fehlermeldung ausgegeben. Die Ausgabe erfolgt in der Kommandozeile.
- „vd“ (validate & display): wie „v“, nur dass der Raum und die Lampen nach der Validierung zusätzlich in der grafischen Oberfläche angezeigt werden.
- „d“ (display): der in der XML-Datei enthaltene Raum und die Lampen werden in der grafischen Oberfläche angezeigt. Falls die angegebene XML-Datei keinen zulässigen Raum enthält, wird eine Fehlermeldung auf der Kommandozeile ausgegeben.

Der Eingabedateiparameter **if** (Input-File) ist ein String, der den Pfad der Eingabedatei beinhaltet. Der Parameter für das Zeitlimit **l** ist eine positive natürliche Zahl, welche die maximale Rechenzeit in Sekunden angibt. Ein Beispielparameteraufruf kann demnach wie folgt aussehen: **r=s if="bin\\raum_1.xml" l=30**.

3.2.5. API-Modus

Im API-Modus soll das Lösungsverfahren durch ein weiteres Programm aufgerufen und in dessen Adressraum ausgeführt werden können. Implementieren Sie dazu die in Listing 4 angegebene Klasse.

```
01 package fernuni.propra.algorithm;
02
03 public class Ausleuchtung implements IAusleuchtung {
04
05     /**
06      * Überprüft die eingegebene Lösung auf Korrektheit
07      * @param xmlFile Dokument mit der Lösung, die validiert…
08      * @return true, falls die eingegebene Lösung korrekt ist
09      */
10     @Override
11     public boolean validateSolution(String xmlFile) {
12         // TODO Logik implementieren
13     }
14
15     /**
16      * Ermittelt eine Lösung zu den eingegebenen Daten
17      * @param xmlFile Dokument, das die zu lösende Problem…
```

```
18       * @param timeLimit Zeitlimit in Sekunden
19       * @return Anzahl der Lampen der ermittelten Lösung
20       */
21      @Override
22      public int solve(String xmlFile, int timeLimit) {
23          // TODO Logik implementieren
24      }
25  }
```

Listing 4: Schnittstellenspezifikation der Algorithmenkomponente

Beachten Sie, dass keine Änderungen an der Klassen- und Methodendefinition und der Paketzuordnung vorgenommen werden dürfen.

Das Ziel der Methode validateSolution(String xmlFile) besteht in der Überprüfung der in der Datei xmlFile gespeicherten Lösung. Als Rückgabewert liefert die Methode den Wert true falls die Lampen den Raum vollständig beleuchten.

Die Methode solve(String xmlFile, int timeLimit) startet das Lösungsverfahren und ermittelt die Positionen von Lampen, die den Raum vollständig beleuchten. Die beste innerhalb des vorgegebenen Zeitlimits gefundene Lösung wird in der angegebenen XML-Datei abgespeichert. Die Methode liefert die Anzahl der platzierten Lampen zurück.

3.2.6. Testbarkeit

Bei der Implementierung des Lösungsverfahrens sind Algorithmen zur Lösung von unterschiedlichen Unterproblemen wie beispielsweise zur Bestimmung von Kandidaten für Lampenpositionen oder der Ausgabe der Lösung zu entwickeln. Kapseln Sie die Lösung von Unterproblemen in separaten Methoden. Bei der Kapselung ist dafür zu sorgen, dass die Lösung von Unterproblemen individuell getestet werden kann.

Entwickeln Sie für alle auftretenden Unterprobleme geeignete Testklassen, durch welche die Korrektheit der implementierten Methoden sichergestellt wird. Die Testklassen müssen auf Basis von Unit Tests entwickelt werden. Achten Sie darauf, dass Ihre Implementierung alle Testfälle besteht.

3.2.7. Empfohlenes Vorgehen

1. Versuchen Sie zunächst einmal probeweise, Lampen in den Grundrissen verschiedener Räume manuell zu platzieren. Wann ist ein Raum vollständig beleuchtet? Machen Sie sich mit der Problemstellung vertraut und versuchen Sie ein Gefühl dafür zu bekommen, wo günstige Positionen von Lampen liegen.
2. Konzipieren Sie anschließend den Ablauf des Lösungsverfahrens. Identifizieren Sie dafür die zu lösenden Unterprobleme. Entwerfen Sie aufbauend auf diesen Überlegungen eine geeignete Softwarearchitektur, die insbesondere den Anforderungen an die Anpassbarkeit des Lösungsverfahrens genügt.
3. Entwickeln Sie zunächst eine geeignete Problemkodierung, die in der Lage ist, den Grundriss eines beliebigen Raumes eindeutig zu repräsentieren. Konzipieren Sie bereits in dieser Phase die Lösung von auftretenden Unterproblemen. Achten Sie darauf, dass die entworfene Problemkodierung eine effiziente Lösung der Unterprobleme unterstützt.

4. Implementieren Sie die Eingabe- und Ausgabekomponenten, durch welche die XML-Dateien in den beschriebenen Formaten eingelesen und ausgegeben werden können. Beim Lesen einer Eingabedatei sind einerseits die eingelesenen Informationen in die Problemkodierung zu überführen. Andererseits ist die interne Problemkodierung geeignet aufzubereiten, um diese in das geforderte Ausgabeformat zu transformieren.
5. Realisieren Sie die grafische Ausgabe von Räumen und Positionen von Lampen. Testen Sie die Korrektheit der grafischen Ausgaben anhand der von der Kursbetreuung zur Verfügung gestellten Beispielinstanzen.
6. Implementieren Sie die Algorithmen für die zu lösenden Unterprobleme. Die dazu erforderlichen Testfälle sollen bereits während der Implementierung der Algorithmen zur Verfügung stehen. Achten Sie bei der Entwicklung der Algorithmen unbedingt darauf, dass diese effizient ausgeführt werden können. → hier findet man bestimmt schon die Unterprobleme
7. Entwickeln Sie den Backtracking-Algorithmus. Analog zur Implementierung der Algorithmen zur Lösung der Unterprobleme ist auch der Backtracking-Algorithmus so effizient wie möglich zu implementieren.

4. Hinweise zur Durchführung

Im Folgenden werden Vorgaben und Hinweise zur erfolgreichen Erstellung des Programms beschrieben. Außerdem werden Rahmenbedingungen, die bei der Implementierung zu beachten sind, dargestellt.

4.1. Allgemeine Vorgaben

In der Konzeption, der Programmierung und der Durchführung sind folgende Systeme und Bibliotheken in den genannten Versionen explizit erlaubt:

- Runtime: Java Standard Edition (SE) in Version 13.0.2
- Entwicklungsumgebung: Eclipse IDE 2019-12
- XML Parser: JDOM in Version 2.0.6
- Unit Test Framework: JUnit in Version 4.13.

Alle nötigen Bibliotheken werden in einem Musterworkspace zur Verfügung gestellt. Verwenden Sie zur Implementierung Ihrer Lösung ausschließlich diesen Musterworkspace. Im Musterworkspace sind bereits einige Java-Projekthüllen enthalten, die beim Softwareentwurf als Leitlinien dienen.

Bevor Sie die Anwendung zur Bewertung abgeben, sind Integrationstests durchzuführen. Das bedeutet, dass Sie zusätzlich zu den Unit Tests komponentenübergreifend testen müssen. In der einfachsten Form kann das manuell erfolgen, indem Sie die geforderten Funktionen einzeln überprüfen. Geben Sie Ihr Programm erst dann ab, wenn Sie sicher sind, alle Anforderungen erfüllt zu haben.

Es ist ebenfalls darauf zu achten, dass das Programm unabhängig vom Ort der Ablage auf der Festplatte geöffnet und kompiliert werden kann. Kopieren Sie dazu vor der Abgabe den kom-

pletten Workspace an eine andere Stelle der Festplatte Ihres Computers. Versuchen Sie nun, die Software zu öffnen und zu kompilieren. Führen Sie dann alle Unit Tests erneut aus.

4.2. Dokumentation und Richtlinien

Die Dokumentation des Programms erfolgt im Programmcode. Kommentare sollen einen fachlichen, logischen und konzeptionellen Charakter haben. Nutzen Sie dazu die in der Java Standard Edition (Java SE) enthaltene Javadoc-Erweiterung. Ein Tutorial zu Javadoc ist in [1] zu finden. Es ist zwingend erforderlich, mit Hilfe von Javadoc alle öffentlichen Klassen, Schnittstellen, deren Parameter und Rückgabewerte sowie alle öffentlichen Variablen von Klassen und Schnittstellen zu kommentieren. Es ist ebenfalls notwendig, bei der Implementierung komplexer Logik, wie beispielsweise bei der Implementierung oder der Validierung des Backtracking-Algorithmus, im Programmcode Java-Kommentare in Bezug auf das logische Vorgehen einzufügen. Das ermöglicht Dritten ein einfacheres Verständnis, hilft beim Nachvollziehen der Überlegungen des Entwicklers und vermeidet so Missverständnisse. Analog zum eigentlich Programm sind alle Testfälle so zu kommentieren, dass klar erkennbar ist, wofür der jeweilige Testfall verwendet werden kann.

4.3. Komponentenentwurf

Im Programmierpraktikum sollen Sie ein Lösungsverfahren zur Bestimmung einer effizienten Ausleuchtung eines Raumes entwickeln. Neben der eigentlichen Funktion soll das Programm so entwickelt werden, dass es objektorientierten Standards genügt. Das bedeutet, dass das Programm aus mehreren entkoppelten Komponenten besteht, die in definierten Beziehungen zueinander stehen.

Das Programm besteht typischerweise aus folgenden Komponenten:

- Main Component (Starten und Auswerten der Anwendung bzw. der Parameter)
- File Processing Component (Lesen und Schreiben von XML-Dateien sowie Datenbereitstellung)
- Algorithm Component (beinhaltet die Algorithmen zur Lösungsermittlung)
- User Interface Component (grafische Benutzeroberfläche)
- Test Component (enthält die einzelnen Testfälle).

Grundsätzlich ist zu empfehlen, vor der Programmierung einen groben Architekturentwurf zu erstellen. Das bedeutet, dass Sie die einzelnen Komponenten zum Beispiel unter Verwendung von UML zeichnen. Stellen Sie auch die Beziehungen der einzelnen Komponenten zueinander dar. Anschließend definieren Sie die Funktionen und Eigenschaften jeder Komponente. Beginnen Sie erst dann mit der Implementierung, nachdem Sie alle Funktionen des Systems einer Komponente zugeordnet haben. Es ist sinnvoll, zunächst für jede Komponente ein Java-Projekt in Eclipse anzulegen. Nachdem Sie das durchgeführt haben, referenzieren Sie die Projekte so untereinander, wie Sie es in der Architektur vorgesehen haben. Dadurch wird das Programm strukturierter, so dass auch bei fortschreitender Implementierung die Komponententrennung und der Überblick bestehen bleiben. Die genannten Komponenten sowie einige der Beziehungen sind bereits im Musterworkspace vorhanden.

4.3.1. Hauptkomponente

Die Main Component soll den zentralen Einstiegspunkt der Anwendung realisieren. Die Main-Methode des Hauptprogramms ist über drei Kommandozeilenparameter zu steuern. Der erste Parameter steuert den Ablauf des Hauptprogramms. Der zweite Parameter identifiziert die zu verarbeitende XML-Datei. Als dritter Parameter wird dem Hauptprogramm eine maximale Rechenzeit vorgegeben. Eine detaillierte Beschreibung der einzelnen Parameter ist in Unterabschnitt 3.2.4 zu finden.

Falls ein Fehler bei der Parametereingabe auftritt, ist das dem Anwender durch eine entsprechende Fehlermeldung zu signalisieren. Bei der Fehlermeldung ist keine technische Meldung auszugeben, sondern der Benutzer so detailliert wie möglich über den Fehler zu informieren. Guter Stil ist es, bei der Ausgabe von Fehlertexten dem Benutzer mögliche Lösungen anzubieten (z.B. „Bitte geben Sie eine Eingabedatei mit Hilfe des Parameters if=“BeispielFile.xml“ an.“).

4.3.2. Dateiverarbeitung

Die File Processing Component ist für die Dateneingabe und Datenausgabe verantwortlich. Sie hat die Aufgabe, die XML-Eingabedatei einzulesen und in eine interne Java-Objektstruktur abzulegen. Komponenten, welche die File Processing Component verwenden, wissen nicht, aus welcher Quelle die Daten stammen und in welchem Format sie in der Quelle abgelegt sind. Durch diese Entkopplung kann später problemlos die Dateiverarbeitung ausgetauscht werden, ohne dass andere Komponenten ihren Programmcode anpassen müssen. Im Programmierpraktikum erfolgt die Dateneingabe und -ausgabe im XML-Dateiformat (siehe Unterabschnitt 3.2.1).

Die File Processing Component muss ebenfalls dafür sorgen, dass eine durch das Programm ermittelte Lösung und die zugehörigen Daten zurück in die angegebene XML-Datei geschrieben werden.

Die im Listing 1 gezeigte DTD definiert die XML-Dokumentstruktur, in der die Probleminstanzen eingelesen bzw. die Ergebnisse ausgegeben werden. Verwenden Sie zur Implementierung der Dateneingabe- und Datenausgabefunktionalität entweder den DOM-Parser des im Musterworkspace bereits integrierten javax.xml Package oder die JDOM2-Bibliothek [2].

4.3.3. Algorithmenentwicklung

Um die Algorithm Component unabhängig nutzen zu können, muss diese über spezifizierte Schnittstellen angesprochen werden können. Die Grundidee besteht darin, dass andere Programme die Algorithm Component wiederverwenden können, ohne spezifische Details der Implementierung zu kennen. Aus diesem Grund müssen Schnittstellen und Eigenschaften vereinbart werden. Zu einer technischen Schnittstellenspezifikation gehören typischerweise der vollständige Schnittstellenname, die Übergabeparameter, die möglichen Fehler und der Rückgabeparameter.

Im Programmierpraktikum sind zwei Schnittstellen innerhalb einer fest definierten Klasse zu implementieren. Die Schnittstellenvorgaben sind im Listing 4 beschrieben und in der Algorithm Component im Musterworkspace bereits angelegt. Im Musterworkspace befinden sich genauere Beschreibungen der einzelnen Schnittstellenparameter. Beachten Sie, dass keine Änderungen an der Klassen- und Methodendefinition sowie der Paketzuordnung vorgenommen werden dürfen. Implementieren Sie deshalb den nötigen Programmcode in die bereits angelegte Struktur im Musterworkspace.

Entwickeln Sie für alle auftretenden Unterprobleme geeignete Testklassen, durch die die Korrektheit der implementierten Methoden möglichst umfassend sichergestellt wird. Achten Sie darauf, dass Ihre Implementierung alle Testfälle besteht.

Beachten Sie, dass das Lösungsverfahren so effizient wie möglich zu implementieren ist. Dazu zählt beispielsweise das Verwenden von geeigneten Datenstrukturen sowie die speicher- und zeitschonende Konzeption und Implementierung der Algorithmen.

4.3.4. Grafische Ausgabe

In der User Interface Component soll die grafische Ausgabe der Lösung erfolgen. Implementieren Sie ein Hauptfenster, in dem der Grundriss eines Raumes sowie die Positionen der Lampen angezeigt werden können. Die Darstellung sollte so skaliert sein, dass der gesamte Grundriss innerhalb eines Fenster auf einem Bildschirm mit einer Auflösung von 1024x768 Pixeln angezeigt werden kann. Bei einer Änderung der Fenstergröße soll die Darstellung nicht verzerrt sondern mit einem festen Seitenverhältnis skaliert werden. Die grafische Ausgabe ist so einfach wie möglich zu halten. Sie können in der JRE Systembibliothek enthaltene Toolkits wie AWT oder Swing verwenden. Um die Konzepte von Swing zu verstehen, kann das Tutorial [5] verwendet werden, das alle notwendigen Techniken für die Implementierung der Aufgabenstellung beschreibt.

4.3.5. Testbarkeit

In der Test Component werden die Tests für die einzelnen Komponenten implementiert. Im Normalfall wird für jede Komponente eine dazugehörige Testkomponente angelegt. Aus Vereinfachungsgründen wird im Programmierpraktikum jedoch nur eine Komponente verwendet.

Bei der Implementierung des Lösungsverfahrens sind Algorithmen zur Lösung von Unterproblemen zu entwickeln. Kapseln Sie die Lösung von Unterproblemen in separate Methoden, Klassen und Schnittstellen. Bei der Kapselung muss sichergestellt werden, dass die Lösungen der Unterprobleme individuell getestet werden können.

Die einzelnen Tests werden typischerweise in Testklassen entwickelt. Jede Testklasse gehört immer zu einer fachlichen bzw. thematischen Einheit (Unit). Aus diesem Grund ist es nötig, dass Sie verschiedene Testklassen für das Programm in der Test Component anlegen. Die Implementierung ist auf Basis des JUnit-Frameworks [3] durchzuführen. Die zugehörigen Bibliotheken sind bereits im Musterworkspace eingebunden.

Im Musterworkspace sind in der Test Component bereits einige Tests implementiert. Nutzen Sie die implementierten Tests, um Ihr Programm zu testen.

Zusätzlich dazu beinhalten einige Testfälle konkrete Probleminstanzen. Das kann bei der Entwicklung des Backtracking-Algorithmus nützlich sein, da die implementierten Tests mit Hilfe der API-Schnittstelle versuchen, eine Lösung zu ermitteln.

Auf Basis der bereits enthaltenen Tests sind möglichst viele weitere Tests zu entwickeln. Mit zunehmender Größe der Anwendung können Sie so sicherstellen, dass sich stets alle Funktionen wie gewünscht verhalten.

4.3.6. Empfohlenes Vorgehen

Das in Unterabschnitt 3.2.7 beschriebene Vorgehen wird nachfolgend um technische Aspekte erweitert:

1. Beginnen Sie zunächst mit dem Softwareentwurf. Zeichnen Sie sich dafür die in Abschnitt 4.3 beschriebenen Komponenten und deren Beziehungen auf. Danach ordnen Sie die Funktionen des Programms den einzelnen Komponenten zu. Vernachlässigen Sie dabei zunächst die technischen Rahmenbedingungen.
2. Konzipieren und entwerfen Sie ebenfalls die verschiedenen Anwendungsfälle des Programms. Dazu gehören beispielsweise das Ermitteln, die Validierung sowie das Anzeigen einer Lösung. Zeichnen Sie sich die einzelnen Anwendungsfälle auf, um dadurch ein Gefühl für die Komplexität und den Ablauf der einzelnen Schritte (Funktionen) zu bekommen. Als nächstes ordnen Sie die einzelnen Schritte den zuvor definierten Komponenten zu.
3. Konzipieren Sie ebenfalls vor der Implementierung mit Hilfe der Anwendungsfälle den Ablauf des Lösungsverfahrens. Identifizieren Sie dabei die zu lösenden Probleme bzw. Unterprobleme. Denken Sie bereits jetzt über mögliche algorithmische Lösungen nach. Entwerfen Sie aufbauend auf diesen Überlegungen eine geeignete Architektur, die insbesondere den Anforderungen an Anpassbarkeit und Entkopplung genügt.
4. Zusätzlich zur Anwendungsfallkonzeption ist nun die technische Konzeption der Komponenten durchzuführen. Dazu gehört die Spezifikation von Schnittstellen für die einzelnen Komponenten. Zu diesem Zeitpunkt müssen noch nicht zwingend alle Parameter und Werte der Schnittstellen endgültig festgelegt werden. Achten Sie darauf, die zuvor definierten Beziehungen zwischen den einzelnen Komponenten nicht zu verletzen.
5. Laden Sie den im Kursportal zur Verfügung gestellten Musterworkspace in der Entwicklungsumgebung. Im Workspace sind bereits die einzelnen Komponenten mit rudimentären Beziehungen zueinander eingefügt. Öffnen Sie den Workspace und assoziieren Sie, wie zuvor im Entwurf beschrieben, die Beziehungen der Komponenten.
6. Bevor Sie Logik implementieren, sind zunächst einzelne Testfälle mit Hilfe des JUnit-Frameworks zu erstellen. Das hilft Ihnen, gezielt nur die Funktionen zu prüfen, an denen Sie gerade arbeiten. Außerdem werden dadurch Seiteneffekte durch andere Komponenten vermieden. So ist es nicht erforderlich, für jede Testinstanz alle nötigen Schritte des Anwendungsfalls durchzuführen, sondern lediglich die sich gerade in Entwicklung be-

findliche Funktionalität kann getestet werden. Zu Beginn erscheint dieses Vorgehen umständlich, mit fortschreitender Entwicklung werden aber die Vorteile eines solchen Vorgehens sichtbar werden. Das wird insbesondere dann wichtig, wenn Sie Verbesserungen am bestehenden Programmcode vornehmen. Durch die Testfälle können Sie prüfen, ob die Logik der Systembausteine nach der Änderung identisch geblieben ist.

7. Implementieren Sie die Eingabe- und Ausgabefunktionen in der File Processing Component. Beim Lesen einer Eingabedatei sind einerseits die eingelesenen Informationen in die Problemkodierung zu überführen, andererseits ist die interne Problemkodierung geeignet aufzubereiten, um diese in das geforderte Ausgabeformat zu transformieren. Es ist sinnvoll, mit dem Datenmodell zu starten, um bereits erste Testdaten in der weiteren Entwicklung nutzen zu können.
8. Implementieren Sie die Methoden- bzw. Klassenrümpfe der zuvor identifizierten einzelnen Anwendungsfälle. Implementieren Sie ebenfalls die Aufrufreihenfolge der Methoden in Ihrem Programmcode bzw. in den einzelnen Komponenten des Programms. Alle Funktionen der einzelnen Anwendungsfälle sollen am Ende als Rumpf in Ihrem Programm implementiert sein. Die zuvor implementierten Datenklassen können so bereits in der Schnittstellenimplementierung verwendet werden. Nutzen und erweitern Sie während der Entwicklung kontinuierlich die zuvor implementierten Tests zur Überprüfung der gerade implementierten Validierungslogik.
9. Um die grafische Oberfläche zu entwickeln, können Sie ebenfalls die bereits implementierten Datenklassen verwenden. Als Daten für die Unittests können die bereits gelösten Probleminstanzen aus den implementierten Testfällen verwendet werden. Dadurch können Sie unabhängig vom Rest des Programms die grafische Ausgabe entwickeln.
10. Entwickeln Sie den Backtracking-Algorithmus. Beachten Sie, dass das Verfahren beendet wird, nachdem die Lösung gefunden wurde. Achten Sie bei der Entwicklung der Algorithmen unbedingt darauf, dass diese effizient ausgeführt werden können. Der Zeitaufwand wird durch die Größe des untersuchten Lösungsraums bestimmt. Deshalb ist bereits frühzeitig zur Reduzierung des Lösungsraums auf die Bestimmung einer kleinen Menge von Kandidaten für Lampenpositionen zu achten. Schauen Sie sich zunächst den in Unterabschnitt 3.2.2 angegebenen Pseudocode an, bevor Sie mit der Programmierung beginnen. Überlegen Sie, wie Sie die einzelnen Schritte umsetzen können. Beginnen Sie erst mit der Implementierung, nachdem Sie den Algorithmus verstanden haben. Denken Sie bei der Implementierung stets an die Komponententrennung und an die Austauschbarkeit der einzelnen Teilfunktionen des Algorithmus. Verwenden Sie dazu Schnittstellen oder abstrakte Klassen. Es empfiehlt sich, zunächst einige Testfälle für den Algorithmus zu entwickeln.
11. Nachdem der Algorithmus implementiert wurde, können Sie diesen in der vorgegebenen API-Klasse aus Listing 4 verwenden, um Probleminstanzen zu lösen bzw. übergebende Lösungen zu validieren. Gegebenenfalls ist es sinnvoll, bereits während der Entwicklung des Algorithmus die API-Klasse zu integrieren.
12. Im vorletzten Schritt müssen Sie alle zuvor getrennt entwickelten Komponenten integrieren. Dazu implementieren Sie die Anwendungsfälle vollständig. Achten Sie dabei un-

bedingt auf die Entkopplung der Komponenten. Verwenden Sie aus diesem Grund ausschließlich die implementierten Schnittstellen anstelle der konkreten Klassen.

13. Im letzten Schritt sind die Anforderungen der Main Component wie z.B. Konsoleneingabe und Dateiübergabe zu implementieren und die zuvor implementierten und getesteten Anwendungsfälle anzubinden.
14. Führen Sie intensive Integrationstests durch. Korrigieren Sie dabei auftretende Fehler in den jeweiligen Komponenten. Falls Sie während der Entwicklung auf Fehler stoßen, sind direkt entsprechende Tests zu implementieren.

5. Literatur

[1] Javadoc-Tutorial. 2019. http://www.oracle.com/technetwork/articles/java/index-137868.html.

[2] JDOM2-Bibliothek. 2019. http://www.jdom.org/.

[3] JUnit-Framework. 2019. http://junit.org/junit4.

[4] Saake, G., Sattler, K.-U. 2010. Algorithmen und Datenstrukturen: Eine Einführung mit Java. 4. Auflage, dpunkt.verlag, Heidelberg.

[5] Swing-Tutorial. 2019. http://zetcode.com/tutorials/javaswingtutorial/.

FernUniversität in Hagen

01584-2020

Bitte hier unbedingt Matrikelnummer und Adresse eintragen, sonst keine Bearbeitung möglich.

Postanschrift: FernUniversität in Hagen 58084

Name

Straße

PLZ, Ort

Bestimmungsland (nur bei Anschriften außerhalb Deutschlands)

EINGANGSSTEMPEL
FERNUNIVERSITÄT

MI

Bitte zurück an:
FernUniversität in Hagen
LG Unternehmensweite Softwaresysteme
D-58084 Hagen

01584: Grundpraktikum Programmierung

durchgeführt vom Lehrgebiet Unternehmensweite Softwaresysteme im Sommersemester 2020

Erklärung

Ich erkläre, dass ich die schriftliche Implementierung/Ausarbeitung zum Grundpraktikum Programmierung selbstständig und ohne unzulässige Inanspruchnahme Dritter fertiggestellt habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht.

Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Implementierung/Ausarbeitung wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Mit der Abgabe der elektronischen Fassung der endgültigen Version der Implementierung/Ausarbeitung nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

Datum, Unterschrift

Anlagen: Quellcode

Letzter Einsendetag: **20.07.2020** **(Datum des Poststempels)**