Karlsruher Institut für Technologie

Prof. Dr.-Ing. Anne Koziolek
Institute of Information Security and Dependability (KASTEL)
https://sdq.kastel.kit.edu/wiki/Programmieren/
programmieren-vorlesung@cs.kit.edu

# Programmieren – Sommersemester 2023

| | | | |
|---|---|---|---|
| **Übungsblatt 4**<br>Version 1.1 | 20 Punkte | Ausgabe:<br>Abgabe:<br>Abgabefrist: | 06.06.2023, ca. 12:00 Uhr<br>14.06.2023, 12:00 Uhr<br>22.06.2023, 06:00 Uhr |

## Änderungen v1.1

Kleinere Änderungen:

- Die Pakete `java.time` und `java.time.format` sind jetzt auch erlaubt.
- Kleinere Tippfehler wurden in Bsp. 1 und und Bsp. 2 in Unterabschnitt A.4 behoben.

## Geschlechtergerechte Sprache

Wenn das generische Maskulinum gewählt wurde, geschieht dies zur besseren Lesbarkeit und zum einfachen Verständnis der Aufgabenstellung. Sofern nicht anders angegeben, beziehen sich Angaben im Sinne der Gleichbehandlung auf Vertretende aller Geschlechter.

## Plagiarismus

Es werden nur selbstständig angefertigte Lösungen akzeptiert. Das Einreichen fremder Lösungen, seien es auch nur teilweise Lösungen von Dritten, aus Büchern, dem Internet oder anderen Quellen, ist ein Täuschungsversuch und führt zur Bewertung „nicht bestanden“. Ausdrücklich ausgenommen hiervon sind Quelltextschnipsel von den Vorlesungsfolien und aus den Lösungsvorschlägen des Übungsbetriebes in diesem Semester. Alle benutzten Hilfsmittel müssen vollständig und genau angegeben werden. Alles, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, muss deutlich kenntlich gemacht werden.

Studierende, die den ordnungsgemäßen Ablauf einer Erfolgskontrolle stören, können von der Erbringung der Erfolgskontrolle ausgeschlossen werden. Ebenso stellt unter anderem die Weitergabe von Teilen von Testfällen oder Lösungen bereits eine Störung des ordnungsgemäßen Ablaufs dar. Auch diese Art von Störungen können ausdrücklich zum Ausschluss der Erfolgskontrolle führen.

## Kommunikation und aktuelle Informationen

In unseren *FAQs*[1] finden Sie einen Überblick über häufig gestellte Fragen und die entsprechenden Antworten zum Modul „Programmieren“. Bitte lesen Sie diese sorgfältig durch, noch bevor Sie Fragen stellen, und überprüfen Sie diese regelmäßig und eigenverantwortlich auf Änderungen.

In den *ILIAS-Foren* veröffentlichen wir gelegentlich wichtige Neuigkeiten. Eventuelle Korrekturen von Aufgabenstellungen werden ebenso auf diesem Weg bekannt gemacht. Das aktive Beobachten der Foren wird daher vorausgesetzt.

Überprüfen Sie das Postfach Ihrer *KIT-Mailadresse* regelmäßig auf neue E-Mails. Sie erhalten unter anderem eine Zusammenfassung der Korrektur per E-Mail an diese Adresse. Alle Anmerkungen können Sie anschließend im Online-Einreichungssystem[2] einsehen.

## Bearbeitungshinweise

Bitte beachten Sie, dass das erfolgreiche Bestehen der verpflichtenden Tests für eine erfolgreiche Abgabe von Übungsblatt 4 notwendig ist. Ihre Abgabe wird automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist. Sie müssen zuerst die verpflichtenden Tests bestehen, bevor die anderen Tests ausgewertet werden können. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Verwenden Sie ausschließlich *Java SE 17*.
- Sofern in einer Aufgabe nicht ausdrücklich anders angegeben, verwenden Sie keine Elemente der Java-Bibliotheken. Ausgenommen ist die Klasse `java.util.Scanner` und alle Elemente aus den folgenden Paketen: `java.lang`, `java.io`, `java.util` und `java.util.regex` sowie `java.time` und `java.time.format`.
- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen. Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 120 Zeichen einhalten.
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie alle Regeln zu Variablen-, Methoden- und Paketbenennung ein.
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute.
- Nutzen Sie nicht das `default`-Package.
- `System.exit()`, `Runtime.exit()` oder ähnliches dürfen nicht verwendet werden.
- Halten Sie die Regeln zur Javadoc-Dokumentation ein.
- Halten Sie auch alle anderen Checkstyle-Regeln ein.

[1] `https://sdq.kastel.kit.edu/wiki/Programmieren/FAQ`
[2] `https://artemis.praktomat.cs.kit.edu/`

Diese folgenden Bearbeitungshinweise sind relevant für die Bewertung Ihrer Abgabe. Dennoch wird Ihre Abgabe durch das Abgabesystem *nicht* automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist.

- Fügen Sie außer Ihrem u-Kürzel keine weiteren persönlichen Daten zu Ihren Abgaben hinzu.
- Beachten Sie, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung als auch Funktionalität bewertet werden. Halten Sie die Hinweise zur Modellierung im ILIAS-Wiki ein.
- Programmcode muss in englischer Sprache verfasst sein.
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich.
- Die Kommentare sollen einheitlich in englischer oder deutscher Sprache verfasst werden.
- Geben Sie im Javadoc-Autoren-Tag nur Ihr u-Kürzel an.
- Wählen Sie aussagekräftige Namen für alle Ihre Bezeichner.

## Checkstyle

Das Online-Einreichungssystem überprüft Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung der Checkstyle-Regeln. Es gibt speziell markierte Regeln, bei denen das Online-Einreichungssystem die Abgabe mit null Punkten bewertet, da diese Regeln verpflichtend einzuhalten sind. Andere Regelverletzungen können zu Punktabzug führen. Sie können und sollten Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki im ILIAS beschreibt, wie Checkstyle verwendet werden kann.

## Abgabehinweise

Die Abgabe im Online-Einreichungssystem wird am 14.06.2023, 12:00 Uhr, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Einreichungssystem bei der richtigen Aufgabe vor Ablauf der Abgabefrist am 22.06.2023, 06:00 Uhr, hochzuladen. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung dahingehend zu testen, und verwenden Sie das Forum, um eventuelle Unklarheiten zu klären. Falls Sie mit Git abgeben, *muss immer* auf den `main`-Branch gepusht werden.

- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe A in Einzelarbeit mit der entsprechenden Ordnerstruktur im zugehörigen Verzeichnis ab.
- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe B in Einzelarbeit mit der entsprechenden Ordnerstruktur im zugehörigen Verzeichnis ab.

# Aufgabe A: Buchungssystem für Car-Sharing (13 Punkte)

In dieser Aufgabe soll ein Buchungssystem für einen Car-Sharing-Anbieter implementiert werden. Der Anbieter betreibt hierbei eine Flotte von Fahrzeugen, die von den Kunden gebucht werden können. Hierbei kann ein Fahrzeug der Flotte über seine eindeutige *Fahrzeugnummer* identifiziert werden. Fahrzeuge werden in vier Kategorien eingeteilt. Eine Buchung ist immer von einem Kunden für ein bestimmtes Fahrzeug in einem gewissen Zeitraum. Ein Kunde kann mehrere Fahrzeuge zur gleichen Zeit buchen. Jedoch kann ein Fahrzeug nicht zur gleichen Zeit von zwei Kunden gebucht werden. Eine *Buchungsbestätigung* ist ein Nachweis, dass ein Fahrzeug für einen Zeitraum gebucht wurde. Eine *Station* ist ein Standort für ein oder mehrere Fahrzeuge aus der Flotte. Dort müssen die gebuchten Fahrzeuge abgeholt und abgegeben werden.

## A.1 Platzhalter

Beachten Sie, dass bei der Beschreibung der Eingabe- und Ausgabeformate die Wörter zwischen spitzen Klammen (`<` und `>`) für Platzhalter stehen, welche bei der konkreten Ein- und Ausgabe durch Werte ersetzt werden. Diese eigentlichen Werte enthalten bei der Ein- und Ausgabe keine spitzen Klammern.

**`<Fahrzeugnr>`:** Eine natürliche Zahl aus dem offenen Intervall $(0, 1000)$, welche bei der Ausgabe immer mit führenden Nullen auf drei Stellen aufgefüllt wird. Jede Fahrzeugnummer ist genau einem Fahrzeug zugeordnet.

**`<Kundennr>`:** Eine natürliche Zahl exklusive 0, welche für den ersten Kunden bei 1 beginnt und für jeden weiteren Kunden um 1 hochgezählt wird.

**`<Vorname>` und `<Nachname>`:** Name des Kunden, welcher jeweils durch eine beliebige, Zeichenkette ohne Zeilenumbruch und ohne Semikolon angegeben werden.

**`<Preis>`:** Die Kosten für eine Buchung, welche durch eine positive reelle Zahl mit einem zweistelligen Nachkommaanteil, getrennt durch einen Punkt, angegeben werden.

**`<Stundenpreis>`:** Die Kosten für eine Stunde, welche durch eine positive reelle Zahl mit einem zweistelligen Nachkommaanteil, getrennt durch einen Punkt, angegeben werden.

**`<Rechnungsnr>`:** Eine natürliche Zahl exklusive 0, welche für die erste Rechnung bei 1 beginnt und für jeden weitere Rechnung um 1 hochgezählt wird.

**`<Stationsname>`:** Eine Zeichenkette ohne Zeilenumbruch und ohne Semikolon.

**`<Stationsnr>`:** Eine natürliche Zahl exklusive 0, welche für die erste Station bei 1 beginnt und für jede weitere Station um 1 hochgezählt wird.

**`<Kategorie>`:** Die Kategorie des Fahrzeugs, entweder `small`, `medium`, `large` oder `transport`. Die Stundenpreise der Fahrzeugkategorien ermitteln sich wie in Tabelle A.1 gelistet.

**`<Datum>`:** Eine Datumsangabe als Zeichenkette im Format JJ-MM-TT, z.B. `2023-05-24`.

**`<Uhrzeit>`:** Eine Uhrzeit als Zeichenkette im 24-Stunden-Format, z.B. `15:30`. Erlaubt sind nur Uhrzeiten im 30-Minuten-Takt, startend von `00:00` bis `23:30`.

| Kategorie | Stundenpreis |
|---|---|
| small | 2,90€ |
| medium | 4,30€ |
| large | 6,60€ |
| transport | 9,90€ |

Tabelle A.1: Fahrzeugpreise nach Fahrzeugkategorie.

**`<Dauer>:`** Eine Buchungsdauer ist immer in festen Stunden angegeben und deswegen eine natürliche Zahl exklusive 0 (Mindestdauer eine Stunde).

## A.2 Befehle

Nach dem Start nimmt Ihr Programm über die Konsole die nachfolgenden Befehle unter Verwendung der Standardeingabe[3] entgegen, welche im Folgenden näher spezifiziert werden. Sie können zu diesem Zweck die Klasse `Scanner`[4] aus dem Paket `java.util` verwenden. Alle Befehle werden auf dem aktuellen Zustand des Programms ausgeführt. Nach Abarbeitung einer Eingabe wartet Ihr Programm auf weitere Eingaben, bis das Programm irgendwann durch die Eingabe der Zeichenfolge `quit` beendet wird.

Achten Sie darauf, dass durch die Ausführung der folgenden Befehle die gegebene Spezifikation nicht verletzt werden und geben Sie in diesen Fällen immer eine aussagekräftige Fehlermeldung aus. Wenn die Benutzereingabe nicht dem vorgegebenen Format entspricht, ist auch eine Fehlermeldung auszugeben. Nach der Ausgabe einer Fehlermeldung soll das Programm wie erwartet fortfahren und wieder auf die nächste Eingabe warten. Jede Fehlermeldung muss mit `ERROR:` gefolgt von einem Leerzeichen beginnen und darf keine Zeilenumbrüche enthalten. Den weiteren Text der Fehlermeldung dürfen Sie frei wählen, er sollte jedoch sinnvoll sein.

### A.2.1 Der `station`-Befehl

Dieser Befehl eröffnet eine neue Station für das Buchungssystem.

**Eingabeformat:** `station <Stationsname>`

**Ausgabeformat:** Im Erfolgsfall wird `OK: added <Stationsname> identified by <Stationsnr>` ausgegeben.

### A.2.2 Der `add`-Befehl

Dieser Befehl fügt einer Station ein neues buchbares Fahrzeug hinzu.

**Eingabeformat:** `add <Fahrzeugnr>;<Stationsnr>;<Kategorie>`

[3] `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/System.html#in`
[4] `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Scanner.html`

**Ausgabeformat:** Im Erfolgsfall wird `OK: added <Fahrzeugnr> to <Stationsname> which has <size> cars` ausgegeben.

`<size>` ist hierbei die Anzahl der Fahrzeuge an der Station nach dem Ausführen des add-Befehls.

### A.2.3 Der `remove`-Befehl

Mit dem Befehl ist es möglich, buchbare Fahrzeuge aus dem Buchungssystem zu entfernen. Entfernte Fahrzeuge können jedoch zu einem späteren Zeitpunkt mittels `add`-Befehl dem System wieder hinzugefügt werden.

**Eingabeformat:** `remove <Fahrzeugnr>`

**Ausgabeformat:** Im Erfolgsfall wird `OK: removed <Fahrzeugnr>` ausgegeben.

Buchung oder andere Verweise für das entferne Fahrzeug bleiben erhalten. Allerdings wird das Auto nicht mehr bei der Station gelistet und kann nicht mehr gebucht werden.

### A.2.4 Der `list-stations`-Befehl

Der Befehl gibt die Liste aller Stationen auf der Konsole aus.

**Eingabeformat:** `list-stations`

**Ausgabeformat:** `<Stationsnr>;<Stationsname>;<size>`

Die Stationen werden zeilenweise aufsteigend sortiert nach der `<Stationsnr>` ausgegeben. Groß- und Kleinschreibung des Stationsnamens stimmen mit der zuvor mittels `add`-Befehl getätigten Eingabe überein. Für den Fall, dass noch keine Stationen hinzugefügt wurden, findet keine Ausgabe statt. `<size>` ist hierbei die Anzahl der Fahrzeuge an der Station.

### A.2.5 Der `list-cars`-Befehl

Der Befehl gibt die Liste aller Fahrzeuge einer Station auf der Konsole aus.

**Eingabeformat:** `list-cars <Stationsnr>`

**Ausgabeformat:** `<Fahrzeugnr>;<Stationsname>;<Stationsnr>;<Kategorie>`

Die Fahrzeuge werden zeilenweise aufsteigend sortiert nach der `<Fahrzeugnr>` ausgegeben. Groß- und Kleinschreibung des Stationsnamens stimmen mit der zuvor mittels `add`-Befehl getätigten Eingabe überein. Für den Fall, dass noch keine Fahrzeuge hinzugefügt wurden, findet keine Ausgabe statt.

### A.2.6 Der `book`-Befehl

Mit dem Befehl kann ein Kunde ein Fahrzeug buchen. Jeder erfolgreichen Buchung wird eine neue Rechnungsnummer zugewiesen. Auch wird mit jeder erfolgreichen Buchung ein neuer Kunde mit einer eindeutigen Kundennummer angelegt, es sei denn ein Kunde mit dem gleichen Namen hat bereits früher eine Buchung getätigt.

**Eingabeformat:**
`book <Fahrzeugnr>;<Vorname>;<Nachname>;<Datum>;<Uhrzeit>;<Dauer>`

**Ausgabeformat:** `OK: booked <Rechnungsnr>;<Kundennr> for <Preis>`

Im Erfolgsfall werden die zugehörige Rechnungs- und Kundennummern angezeigt. Zudem wird der Preis mit ausgegeben, wobei es sich um das Produkt des Stundenpreises, der Fahrzeugkategorie und der Buchungsdauer handelt.

Der Befehl kann nur erfolgreich sein, falls keine überlappende Buchung für das Fahrzeug besteht. Im Fehlerfall sollte die Fehlermeldung indizieren, mit welcher Buchung ein Problem besteht. Falls der Endzeitpunkt einer Buchung dem Startzeitpunkt einer anderen entspricht, zählt dies noch nicht als Überlappung.

### A.2.7 Der `check-available`-Befehl

Mit dem Befehl kann ein Kunde bei einer Station überprüfen, welche Fahrzeuge verfügbar sind. Nicht verfügbar sind Autos, die bereits zur gleichen Zeit oder überlappend gebucht wurden.

**Eingabeformat:** `check-available <Stationsnr>;<Datum>;<Uhrzeit>;<Dauer>`

**Ausgabeformat:** `<Fahrzeugnr>;<Kategorie>;Stundenpreis>`

Die Fahrzeuge werden zeilenweise aufsteigend sortiert nach der `<Fahrzeugnr>` ausgegeben. Für den Fall, dass keine Fahrzeuge verfügbar sind, findet keine Ausgabe statt.

### A.2.8 Der `list-bookings`-Befehl

Der parameterlose Befehl gibt eine Liste aller bisher getätigten Buchungen auf der Konsole auf.

**Eingabeformat:** `list-bookings`

**Ausgabeformat:**
`<Kundennr>;<Fahrzeugnr>;<Rechnungsnr>;<Datum>;<Uhrzeit>;<Dauer>;<Preis>`

Alle Buchungen werden zeilenweise ausgegeben. Die Ausgabe erfolgt aufsteigend sortiert nach der Kundennummer. Bei identischer Kundennummer wird absteigend nach der Rechnungsnummer sortiert. Für den Fall, dass noch keine Buchungen vorgenommen wurden, findet keine Ausgabe statt.

### A.2.9 Der `bill`-Befehl

Der Befehl gibt eine Liste aller Buchungen eines Kunden in einem bestimmten Jahr auf der Konsole aus.

**Eingabeformat:** `bill <Kundennr>;<Jahr>`

**Ausgabeformat:**
`<Kundennr>;<Fahrzeugnr>;<Rechnungsnr>;<Datum>;<Uhrzeit>;<Dauer>;<Preis>`

Alle Buchungen werden zeilenweise ausgegeben. Die Ausgabe erfolgt absteigend sortiert nach der Rechnungsnummer. Abschließend wird eine weitere Zeile ausgegeben:

**Ausgabeformat (Zusammenfassung):** `Sum: <Summe>`

Hierbei ist `<Summe>` die Summe der Preise aller Buchungen des Jahres.

### A.2.10 Der `quit`-Befehl

Der parameterlose Befehl ermöglicht es, das Buchungssystem vollständig zu beenden. Dabei findet keine Konsolenausgabe statt. Beachten Sie, dass hierfür keine Methoden wie `System.exit()` oder `Runtime.exit()` verwendet werden dürfen. Zudem ist der Befehl kein außerplanmäßiges Verhalten des Programms.

## A.3 Command-Pattern-Template

Zur Umsetzung der Befehle wird eine unfertige Implementierung des Kommando-Entwurfsmusters vorgegeben. Diese Implementierung können Sie nutzen, um die Aufgabe zu lösen. Es ist Ihnen zudem erlaubt, diese zu verändern. Stellen Sie sicher, dass auch der vorgegebene Code die Anforderungen an eine saubere Abgabe erfüllt. Dies betrifft unter anderem auch Javadoc-Kommentare.

## A.4 Beispielinteraktion

Der Beispielablauf zeigt zwei verschiedene Spiele nach Start des Programmes. Die Zeilennummern und die Trennlinie sind kein Bestandteil der Benutzerschnittstelle, sie dienen lediglich zur Orientierung für die gegebene Beispielinteraktion. Nutzereingaben sind mit dem Zeichen `>` gekennzeichnet. Die Zeichen `%>` (Prozent-Zeichen und Größer-Zeichen gefolgt von einem Leerzeichen) stellt die Kommandozeile dar. Der Name des Programms `CityMobil` dient nur als Beispiel und ist nicht vorgeschrieben.

**Beispiel 1: Stationen und Abrechnungen**

Beispielinteraktion

```
1  %> java CityMobil
2  > station Hauptbahnhof
3  OK: added Hauptbahnhof identified by 1
4  > station KIT
5  OK: added KIT identified by 2
6  > station Marktplatz
7  OK: added Marktplatz identified by 3
8  > add 133;2;medium
9  OK: added 133 to KIT which has 1 cars
10 > list-stations
11 1;Hauptbahnhof;0
12 2;KIT;1
13 3;Marktplatz;0
14 > add 1;2;transport
15 OK: added 001 to KIT which has 2 cars
16 > list-cars 2
17 001;KIT;2;transport
18 133;KIT;2;medium
19 > check-available 2;2023-06-06;16:30;5
20 001;transport;9.90
21 133;medium;4.30
22 > book 133;Manu;Musterperson;2023-06-06;16:30;5
23 OK: booked 1;1 for 21.50
24 > book 1;Manu;Musterperson;2023-12-21;12:00;3
25 OK: booked 2;1 for 29.70
26 > bill 1;2023
27 1;001;2;2023-12-21;12:00;3;29.70
28 1;133;1;2023-06-06;16:30;5;21.50
29 Sum: 51.20
30 > quit
```

### Beispiel 2: Überlappende Buchungen

Beispielinteraktion

```
%> java CityMobil
> station KIT-presidium
OK: added KIT-presidium identified by 1
> add 001;1;large
OK: added 001 to KIT-presidium which has 1 cars
> add 002;1;large
OK: added 002 to KIT-presidium which has 2 cars
> add 003;1;small
OK: added 003 to KIT-presidium which has 3 cars
> list-cars 1
001;KIT-presidium;1;large
002;KIT-presidium;1;large
003;KIT-presidium;1;small
> book 001;Holger;Hanselka;2023-10-01;10:00;10
OK: booked 1;1 for 66.00
> book 002;Holger;Hanselka;2023-10-01;10:00;10
OK: booked 2;1 for 66.00
> book 001;Alexander;Wanner;2023-10-01;14:30;2
ERROR: Cannot book due to overlapping booking!
> book 002;Alexander;Wanner;2023-10-01;14:30;2
ERROR: Cannot book due to overlapping booking!
> check-available 1;2023-10-01;14:30;2
003;small;2.90
> book 3;Alexander;Wanner;2023-10-01;14:30;2
OK: booked 3;2 for 5.80
> list-bookings
2;003;3;2023-10-01;14:30;2;5.80
1;002;2;2023-10-01;10:00;10;66.00
1;001;1;2023-10-01;10:00;10;66.00
> quit
```

# Aufgabe B: Generische Baum-Datenstruktur (7 Punkte)

Ein Baum ist eine spezielle Art von Graph, der hierarchisch organisiert ist und keine Zyklen enthält. Ein Baum hat einen speziellen Knoten, der als Wurzel bezeichnet wird. Von der Wurzel aus erstrecken sich Kanten zu anderen Knoten, die als Kindknoten bezeichnet werden. Jeder Knoten kann weitere Kindknoten haben, was zu einer hierarchischen Struktur führt. Ein Knoten kann mehrere Kindknoten haben, aber jeder Kindknoten hat nur einen Elternknoten.

In dieser Aufgabe implementieren Sie eine Datenstruktur für einen Baum (siehe Abbildung B.1). Diese Datenstruktur ist generisch, sie enthält also beliebige Elemente eines gemeinsamen Typs. Für diese Datenstruktur ist ihnen bereits eine Schnittstelle vorgegeben, deren Methodensignaturen gemäß ihren Javadoc-Kommentaren und den in dieser Aufgabe folgenden Hinweise implementieren werden müssen. Beachten Sie bei ihrer Implementierung besonders die erforderten Randfälle. Verwenden Sie sinnvolle Fehlermeldungen für Exceptions und schreiben Sie, wo erfordert, aussagekräftige Javadoc-Kommentare.

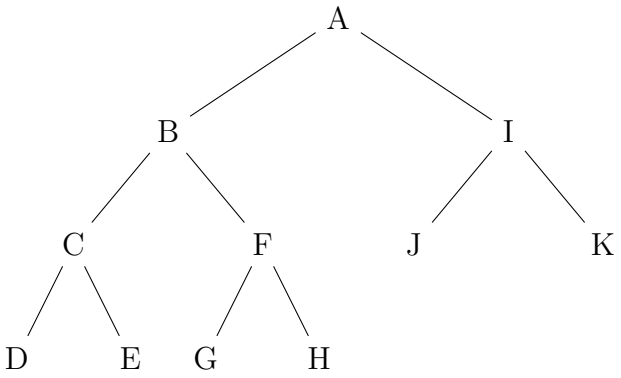


Abbildung B.1: Ein Beispiel für einen Baum mit 11 Knoten (A bis L). Der Knoten A ist die Wurzel. Die Knoten D, E, G, H, K und L sind Blätter.

## B.1 Konstruktor

Implementieren Sie mindestens einen Konstruktor, welcher das Wurzelelement als einziges Parameter entgegennimmt. Der Konstruktor erzeugt somit einen Baum mit einem einzigen Element.

## B.2 Methode `getRoot()`

Die Methode gibt das Wurzelelement des Baumes zurück. Für den Baum in Abbildung B.1 würde also das Objekt für den Knoten A zurückgegeben werden.

## B.3 Methode `getChildren(T parent)`

Die Methode gibt alle direkten Kinder eines Knotens zurück. Für den Baum in Abbildung B.1 würde also für den Elternknoten B die Kinder C und F zurückgegeben werden. Für den Knoten J jedoch nur eine leere Liste, da der Knoten keine Kinder hat.

## B.4 Methode `getParent(T child)`

Die Methode gibt den Elternknoten eines Knotens zurück. Für den Baum in Abbildung B.1 würde also für den Knoten B der Elternknoten A zurückgegeben werden.

## B.5 Methode `add(T element, T parent)`

Die Methode fügt einem bestimmten Elternknoten einen neuen Knoten als Kind hinzu. Abbildung B.2 illustriert dies für die Parameter D (neuer Knoten) und A (Elternknoten).


(a) vorher (b) nachher

Abbildung B.2: Zustand vor und nach dem Einfügen von dem Knoten D unter den Knoten A.

## B.6 Methode `removeLeaf(T leaf)`

Die Methode entfernt ein Blatt aus dem Baum. Abbildung B.3 illustriert dies für den Knoten C.


(a) vorher (b) nachher

Abbildung B.3: Zustand vor und nach dem Entfernen von dem Knoten C.

## B.7 Methode `removeSubtree(T parent)`

Die Methode entfernt einen Knoten aus dem Baum. Hierbei wird der komplette Teilbaum ausgehend von dem entfernten Knoten entfernt. Dies betrifft sowohl direkte als auch indirekte Kinder. Abbildung B.4 illustriert dies für den Knoten B, wobei die Knoten D, E, G und H auch entfernt werden.


(a) vorher (b) nachher

Abbildung B.4: Zustand vor und nach dem Entfernen von dem Knoten B.

## B.8 Methode `size()`

Diese Methode gibt die Anzahl der Elemente im Baum zurück. Das Root-Element zählt auch dazu. Für einen leeren Baum wird 0 zurückgegeben. Für den Baum in Abbildung B.1 würde also 11 zurückgegeben werden.

## B.9 Methode `height()`

Diese Methode gibt die Höhe des Baumes zurück. Dabei werden die Kanten zwischen der Wurzel und dem tiefsten Blatt gezählt (längster Pfad). Für einen leeren Baum wird 0 zurückgegeben. Für den Baum in Abbildung B.1 würde also 3 zurückgegeben werden.

## B.10 Hinweise

Die Javadoc-Kommentare der Schnittstelle verlangen bestimmtes Verhalten für Randfälle. Setzen Sie diese wie gefordert um. Ihre Datenstruktur muss `null` als Element akzeptieren. Zudem müssen auch identische Objekte mehrfach als verschiedene Knoten hinzugefügt werden können. Dies gilt zudem auch für `null`.