Karlsruher Institut für Technologie

PD Dr. rer. nat. Robert Heinrich
Institute of Information Security and Dependability (KASTEL)
https://sdq.kastel.kit.edu/wiki/Programmieren/
programmieren-vorlesung@cs.kit.edu

Programmieren – Wintersemester 2024

Übungsblatt 4 Version 1.0	20 Punkte	Ausgabe: Abgabe: Abgabefrist:	20.12.2023, ca. 12:00 Uhr 10.01.2024, 12:00 Uhr 18.01.2024, 06:00 Uhr

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

[1] https://sdq.kastel.kit.edu/wiki/Programmieren/FAQ

Fragen stellen, und überprüfen Sie diese regelmäßig und eigenverantwortlich auf Änderungen. Beachten Sie zudem die Hinweise im ILIAS-Wiki[2].

In den *ILIAS-Foren* und auf *Artemis* 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[3] 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`.
- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen. Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 140 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.

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. Orientieren Sie sich zudem an den Bewertungskriterien im ILIAS-Wiki.

- Fügen Sie außer Ihrem u-Kürzel keine weiteren persönlichen Daten zu Ihren Abgaben hinzu.

[2] `https://ilias.studium.kit.edu/goto.php?target=wiki_2213632_Hauptseite`
[3] `https://artemis.praktomat.cs.kit.edu/`

- 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 10.01.2024, 12:00 Uhr, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Einreichungssystem bei der richtigen Aufgabe vor Ablauf der Abgabefrist am 18.01.2024, 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 Flugtickets (14 Punkte)

In dieser Aufgabe soll ein Buchungssystem für Flugtickets implementiert werden. Ein *Flugticket* ist vergleichbar einer Fahrkarte, welche im Luftverkehr als Beförderungsdokument ausgestellt wird. Ein Flugticket dient über die Belegfunktion hinaus zu Kontrollzwecken, ob der auf dem Flugticket ausgewiesene Flugpassagier einen gültigen Anspruch auf den darin genannten Transport in einem Luftfahrzeug besitzt. Hierbei kann ein Luftfahrzeug über seine eindeutige *Flugzeugkennung* identifiziert werden. Ein *Flughafen* bezeichnet Start- und Landeplätze für Luftfahrzeug mit der gesamten Infrastruktur, auf welchem regelmäßiger kommerzieller Flugverkehr stattfindet und auch als Verknüpfungspunkt verschiedener Flugrouten dient. Hierbei kann jeder Flughafen über eine eindeutige Zeichenkette identifiziert werden. Eine *Flugroute* ist die Verbindung zweier Flughäfen, welche regelmäßig durch Luftfahrzeuge bedient wird.

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.

A.1.0.1 `<Flugzeugkennung>` Eine natürliche Zahl aus dem geschlossenen Intervall $[1, 99999]$, welche bei der Ausgabe immer mit führenden Nullen auf insgesamt fünf Stellen aufgeführt wird.

A.1.0.2 `<Flugpassagierkennung>` Eine natürliche Zahl, welche für den ersten Flugpassagier bei 1 beginnt und für jeden weiteren Kunden um 1 hochgezählt wird.

A.1.0.3 `<Vorname>` und `<Nachname>` Name des Flugpassagiers, welcher jeweils durch eine beliebige Zeichenkette ohne Zeilenumbruch und ohne Leerzeichen angegeben werden.

A.1.0.4 `<Preis>` Die Kosten für ein Flugticket, welche durch eine positive reelle Zahl mit einem zweistelligen Dezimalteil, getrennt durch einen Punkt, in Euro, ausgegeben wird.

A.1.0.5 `<Rechnungsnummer>` Eine natürliche Zahl, welche für die erste Rechnung bei 1 beginnt und für jeden weitere Rechnung um 1 hochgezählt wird.

A.1.0.6 `<Kundennummer>` Eine natürliche Zahl, welche für den ersten Kunden bei 1 beginnt und für jeden weiteren Kunden um 1 hochgezählt wird.

A.1.0.7 `<Start>` und `<Ziel>` Bezeichner für den Start- und Zielflughafen einer Flugroute, jeweils in Form einer Zeichenkette ohne Zeilenumbruch und ohne Leerzeichen.

A.2 Befehle

Nach dem Start nimmt Ihr Buchungssystem über die Konsole mittels der Standardeingabe Eingaben entgegen, welche im Folgenden näher spezifiziert werden. Nach Abarbeitung einer Eingabe wartet Ihr Buchungssystem auf weitere Eingaben, bis das Buchungssystem 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 Buchungssystem wie erwartet fortfahren und wieder auf die nächste Eingabe warten. Jede Fehlermeldung muss mit `Error ,` beginnen und darf keine Zeilenumbrüche enthalten. Den weiteren Text der Fehlermeldung dürfen Sie frei wählen, er sollte jedoch sinnvoll sein.

A.2.1 Der `add`-Befehl

Dieser Befehl fügt dem Buchungssystem eine neue buchbare Flugroute hinzu.

A.2.1.1 Eingabeformat `add <Flugzeugkennung> <Start> <Ziel> <Preis>`

A.2.1.2 Ausgabeformat Im Erfolgsfall wird `OK` ausgegeben.

A.2.2 Der `remove`-Befehl

Mit dem Befehl ist es möglich, Flugrouten aus den Buchungssystem zu entfernen. Gestrichene Flugrouten mit einer Flugzeugkennung können jedoch zu einem späteren Zeitpunkt mittels `add`-Befehl dem System wieder hinzugefügt werden.

A.2.2.1 Eingabeformat `remove <Flugzeugkennung>`

A.2.2.2 Ausgabeformat Im Erfolgsfall wird `OK` ausgegeben.

A.2.3 Der `list-routes`-Befehl

Der parameterlose Befehl gibt die Liste aller buchbaren Flugrouten auf der Konsole aus.

A.2.3.1 Eingabeformat `list-routes`

A.2.3.2 Ausgabeformat `<Flugzeugkennung> <Start> <Ziel> <Preis>`

Die buchbaren Flugrouten werden zeilenweise, analog des Eingabeformats des `add`-Befehls, aufsteigend, sortiert nach der `<Flugzeugkennung>`, ausgegeben. Groß- und Kleinschreibung des Start- und Zielflughafens stimmen mit der zuvor mittels `add`-Befehl getätigten Eingabe überein. Für den Fall, dass noch keine Flugroute hinzugefügt wurden, findet keine Ausgabe statt.

A.2.4 Der `find-shortest-trips`-Befehl

Dieser Befehl erlaubt alle transitive Routen bestehend aus mehreren Flügen zu finden, um zum Beispiel von einem Startflughafen zu einem Zielflughafen zu kommen, wenn diese nicht direkt mit einem Flug verbunden sind. Der Befehl gibt alle Routen (bestehend aus einem oder mehreren Einzelflügen) sortiert aufsteigend nach der Anzahl ihrer Einzelflüge aus. Auf einer Route darf kein Flughafen doppelt besucht werden. Besteht ein Direktflug, ist dies also die kürzeste Route. Haben zwei Routen gleich viele Einzelflüge, dann ist das sekundäre Kriterium für die Sortierung die Flugzeugkennung der Einzelflüge in Reihenfolge der Route (auch aufsteigend sortiert).

A.2.4.1 Eingabeformat `find-shortest-trips <Start> <Ziel>`

A.2.4.2 Ausgabeformat Jede transitive Route wird als eigene Zeile im folgenden Format ausgegeben: "`<Umstiege> Transfers: <Route>`", wobei `<Umstiege>` für die Anzahl der Zwischenstopps in der transitiven Route steht. Für jede Route `<Route>` werden die Bezeichner der Flughäfen der Route konkateniert durch die Zeichenkette " `-<Flugzeugkennung>->` " (z.B. " `-00010->` ") ausgegeben. Die Reihenfolge der Flughäfen entspricht hierbei der Route von `<Start>` nach `<Ziel>`. Besteht keine mögliche Route oder ist `<Start>` identisch mit `<Ziel>`, wird nichts ausgegeben.

A.2.5 Der `find-cheapest-trips`-Befehl

Dieser Befehl erlaubt alle transitive Routen bestehend aus mehreren Flügen zu finden, um zum Beispiel von einem Startflughafen zu einem Zielflughafen zu kommen, wenn diese nicht direkt mit einem Flug verbunden sind. Der Befehl gibt alle Routen (bestehend aus einem oder mehreren Einzelflügen) sortiert aufsteigend nach der Summe der Kosten ihrer Einzelflüge aus. Auf einer Route darf kein Flughafen doppelt besucht werden. Haben zwei Routen die gleiche Summe, dann ist das sekundäre Kriterium für die Sortierung die Flugzeugkennung der Einzelflüge in Reihenfolge der Route (auch aufsteigend sortiert).

A.2.5.1 Eingabeformat `find-cheapest-trips <Start> <Ziel>`

A.2.5.2 Ausgabeformat Jede transitive Route wird als eigene Zeile im folgenden Format ausgegeben: "`<Preis>: <Route>`". Für jede Route werden die Bezeichner der Flughäfen der Route konkateniert durch die Zeichenkette " `-<Flugzeugkennung>->` " (z.B. " `-00010->` ") ausgegeben. Die Reihenfolge der Flughäfen entspricht hierbei der Route von `<Start>` nach `<Ziel>`. Besteht keine mögliche Route oder ist `<Start>` identisch mit `<Ziel>`, wird nichts ausgegeben.

A.2.6 Der `book`-Befehl

Mit dem Befehl kann ein Kunde ein Flugticket 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 Bestellung getätigt.

A.2.6.1 Eingabeformat `book <Flugzeugkennung> <Vorname> <Nachname>`

A.2.6.2 Ausgabeformat `<Rechnungsnummer> <Kundennummer>`

Im Erfolgsfall werden die zugehörige Rechnungs- und Kundennummern angezeigt. Rechnungsnummern für die mittels `remove`-Befehl gelöschten Flugrouten werden nicht mit ausgegeben und auch nicht wiederverwendet.

A.2.7 Der list-bookings-Befehl

Der parameterlose Befehl gibt eine Liste aller bisher getätigten Buchungen auf der Konsole aus. Die mittels `remove`-Befehl gelöschten Flugrouten werden nicht mit ausgegeben.

A.2.7.1 Eingabeformat `list-bookings`

A.2.7.2 Ausgabeformat `<Kundennummer> <Flugzeugkennung> <Rechnungsnummer>`

Alle Flugticketbuchungen 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.2.8 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.

A.3 Beispielinteraktion

Die Zeilennummern und die Trennlinie sind kein Bestandteil der Benutzerschnittstelle, sie dienen lediglich zur Orientierung für die gegebene Beispielinteraktion. Die Eingabezeilen werden mit dem Größer-als-Zeichen (`>`) gefolgt von einem Leerzeichen eingeleitet, diese beiden Zeichen sind ebenfalls kein Bestandteil des eingegebenen Befehls, sondern dienen der Unterscheidung zwischen Ein- und Ausgabezeilen.

Beispielinteraktion

```
> add 1 LAX FRA 659.99
OK
> book 1 Max Mustermann
1 1
> book 1 Mareike Musterfrau
2 2
> list-bookings
1 00001 1
2 00001 2
> list-routes
00001 LAX FRA 659.99€
> remove 1
OK
> quit
```

Aufgabe B: Adventskalender II (6 Punkte)

In dieser Aufgabe werden Sie erneut eine Datenstruktur für einen Adventskalender implementieren. Die Datenstruktur ähnelt hierbei der Datenstruktur des letzten Übungsblatts. In diesem Übungsblatt sollen jedoch Exceptions und Generics verwendet werden. Es werden Exceptions geworfen, um Randfälle abzufangen. Zudem soll die Datenstruktur generisch sein, also keinen festen Datentyp als Inhalt der Türen vorgeben. Sie dürfen vom letzten Blatt Ihre eigene Lösung oder die Musterlösung wiederverwenden. Verwenden Sie jedoch **keine Arrays**!

Wie bereits zuvor enthält der Adventskalender Süßigkeiten, die jedoch nun durch einen Typparameter `T` repräsentiert werden. Jede Süßigkeit ist hierbei hinter einer Tür versteckt, die einem Tag im Dezember zugeordnet ist (zum Beispiel vom 01.12. bis zum 24.12. bei 24 Süßigkeiten). Eine Tür kann nur dann geöffnet werden, wenn das aktuelle Datum des entsprechenden Tags schon erreicht wurde. So können zum Beispiel am 4. Dezember nur die ersten vier Türen geöffnet werden. Jede Tür kann hierbei nur einmal geöffnet werden. Der aktuelle Tag wird mit den Süßigkeiten in der Datenstruktur intern gespeichert.

Ihnen werden zwei Interfaces für die Datenstruktur vorgegeben. Hierbei sollen Sie eine konkrete Implementierung (Schlüsselwort `implements`) von der Schnittstelle `CountdownCalendar` erstellen. Es ist nicht erlaubt, die Schnittstellen `CountdownCalendar` und `Copyable` zu verändern.

Implementieren Sie hierbei folgenden Methoden:

- `public AdventCalendar(List<T> candies)` ist ein Konstruktor und erstellt einen Kalender mit den in `candies` übergebenen Süßigkeiten. Wenn die Liste `null` ist, leer ist oder eines der Elemente der Liste `null` ist, soll eine Exception mit passender Fehlermeldung geworfen werden. Die n-te Süßigkeit in der Liste ist hierbei dem n-ten Tag zugeordnet. Je nach Größe der Liste wird der *Maximaltag* bestimmt und der Adventskalender hat dementsprechend viele Türen (wichtig: die Größe ist *nicht* auf 24 beschränkt). Nach Aufruf des Konstruktors ist der aktuelle Tag der Tag *vor* dem ersten Dezember.
- `public int getDay()` gibt den aktuellen Tag als Ganzzahl zwischen 0 und dem Maximaltag zurück. Der Tag vor dem ersten Dezember wird durch den Wert 0 repräsentiert. Der erste Tag, an dem eine Türe geöffnet werden kann, wird also durch den Wert 1 repräsentiert.
- `public int size()` ist eine neue Methode und gibt die Gesamtanzahl der Türen zurück. Dieser Wert entspricht damit dem Maximaltag.
- `public void nextDay()` erhöht den intern gespeicherten Tag. Damit wird gesteuert, welche Tür geöffnet werden kann. Der Tag kann nur bis zum Maximaltag erhöht werden. Danach wird beim Aufruf der Methode stattdessen eine passende Exception geworfen.
- `public void nextDays(int days)` erhöht den intern gespeicherten Tag um mehrere Tage. Das Parameter `days` bestimmt hierbei die Anzahl der erhöhten Tage. Tag kann nur bis zum Maximaltag erhöht werden. Würde die Erhöhung über den Maximaltag hinaus gehen, werden keine Tage erhöht und stattdessen eine passende Exception geworfen. Ist `days` kleiner oder gleich 0, wird direkt eine passende Exception geworfen.

- `public boolean isDoorOpen(int number)` überprüft, ob die Tür mit der passenden Nummer bereits geöffnet wurde. Ist dies der Fall, wird `true` zurückgegeben. Ansonsten wird `false` zurückgegeben. Ist die Nummer echt kleiner eins oder echt größer als der Maximaltag, wird stattdessen eine passende Exception geworfen.
- `public T openDoor(int number)` öffnet die Tür mit der passenden Nummer und gibt die Süßigkeit als Zeichenkette zurück. Wurde die Tür bereits geöffnet, wird `null` zurückgegeben. Ist die Nummer echt größer als der aktuelle Tag, wird eine passende Exception geworfen. Ist die Nummer echt kleiner eins oder echt größer als der Maximaltag, wird eine passende Exception geworfen.
- `public List<T> openDoors(List<Integer> numbers)` öffnet alle Türen deren Nummer in der Liste enthalten ist und gibt die Süßigkeiten in gleicher Reihenfolge in einer Liste zurück. Wurde eine Tür bereits geöffnet, wird die Tür übersprungen, also kein Element in die Liste hinzugefügt. Ist eine Nummer echt größer als der aktuelle Tag, wird eine passende Exception geworfen. Ist eine Nummer echt kleiner eins oder echt größer als der Maximaltag, wird eine passende Exception geworfen.
- `public String toString()` gibt den Adventskalender in Textform zurück. Eine bereits geöffnete Tür wird durch eine öffnende eckigen Klammer gefolgt von *drei Leerzeichen* und abschließend einer schließenden eckigen Klammer dargestellt. Eine ungeöffnete Tür wird durch eine öffnende eckige Klammer gefolgt der Textdarstellung (`toString()`) der Süßigkeit und abschließend einer schließende eckige Klammer dargestellt. Die Türen werden in der Reihe der Tage konkateniert ausgegeben. Mit Ausnahme der letzten Tür wird nach jeweils vier Türen ein Zeilenumbruch angehängt. Ansonsten werden *keine* Trennzeichen verwendet.

Folgende Methoden der Klasse `AdventCalendar` sollen auch implementiert werden, diese unterscheiden sich aber in ihrer Beschreibung nicht von dem vorherigen Übungsblatt:

- `public int numberOfUnopenedDoors()` gibt die Anzahl der bisher noch nicht geöffneten Türen zurück, die zum aktuellen Tag geöffnet werden könnten. Damit ist der Rückgabewert immer zwischen 0 und der Ausgangsgröße des Kalenders. Ist zum Beispiel der Tag der 9. Dezember und es wurden bisher drei Türen geöffnet, wird 6 zurückgegeben.
- `public void reset()` setzt den Kalender zum Ursprungszustand zurück. Der interne Tag ist nun wieder der Tag *vor* dem ersten Dezember. Alle Türen sind wieder geschlossen und die Objekte hinter den Türen entsprechen dem Zustand nach dem Aufruf des Konstruktors. *Tipp:* Verwenden Sie an passender Stelle die Methode `copy()` der Schnittstelle `Copyable`.

Hinweise: In dieser Aufgabe sollen Sie erneut sinnvolle Javadoc-Kommentare für die Klasse und alle öffentlichen Methoden schreiben, die nicht eine bestehende Signatur überschreiben. Implementieren Sie keine weiteren öffentlichen Methoden und keine öffentlichen Attribute. Erstellen Sie keine weiteren Klassen. Private Attribute und private Methoden sind erlaubt. Verändern Sie nicht die vorgegebene Schnittstelle. Intern dürfen neben Listen zur Repräsentation der Daten auch andere Datentypen verwendet werden, dies ist jedoch optional. **Verwenden Sie passende Exception-Klassen mit aussagekräftiger Nachricht. Verwenden sie keine Arrays!**