Prof. Dr. J. Giesl

M. Hark, J. Hensel, D. Korzeniewski

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **2 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Dienstag, den 21.11.2017 um 08:00 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor oder unmittelbar vor Beginn in der Globalübung abgeben.
- In einigen Aufgaben müssen Sie in Java programmieren und `.java`-Dateien anlegen. **Drucken** Sie diese aus **und** schicken Sie sie per **E-Mail** vor Dienstag, dem 21.11.2017 um 08:00 Uhr an Ihre Tutorin/Ihren Tutor.
  Stellen Sie sicher, dass Ihr Programm von `javac` **akzeptiert** wird, ansonsten werden keine Punkte vergeben.
- Benutzen Sie in ihrem Code keine Umlaute, auch nicht in Strings und Kommentaren. Diese führen oft zu Problemen, da diese Zeichen auf verschiedenen Betriebssystemen unterschiedlich behandelt werden. Dadurch kann es dazu führen, dass Ihr Programm bei Ihrer Tutorin/Ihrem Tutor bei Verwendung von Umlauten nicht mehr von `javac` akzeptiert wird.
- Für einige Programmieraufgaben benötigen Sie die Java Klasse `SimpleIO`. Diese können Sie auf der Website herunterladen.
- Halten Sie sich beim Lösen von Programmieraufgaben an die auf der Website zur Vorlesung verfügbaren Codekonventionen. Verstöße, die zu unleserlichem Code führen, können zu Punktabzug führen.
- Einige Hausaufgaben müssen im Spiel Codescape gelöst werden:
  https://codescape.medien.rwth-aachen.de/progra/
  Diese Aufgaben werden getrennt von den anderen Hausaufgaben gewertet.

Tutoraufgabe 1 (Werkzeugkasten):

In dieser Aufgabe wird eine Klasse implementiert, die eine Werkzeugkiste verwaltet. In einer Werkzeugkiste können Materialien (Schrauben, Nieten, etc.), einfache Werkzeuge (Zangen, Schraubendreher, etc.) und Elektrowerkzeuge (Bohrmaschinen, Schleifgerät, etc.) sein. Die meisten Materialien sind sehr klein. Deswegen können alle Materialien zusammen in einem einzelnen Fach der Werkzeugkiste untergebracht werden. Elektrowerkzeuge hingegen sind sehr groß. Jedes Elektrowerkzeug nimmt daher immer je drei benachbarte Fächer ein.
Beachten Sie in allen Teilaufgaben die Prinzipien der Datenkapselung.

a) Schreiben Sie einen Aufzählungstyp `Tool` für die drei Arten von Werkzeugen: `PowerTool`, `SimpleTool` und `Materials`.

b) Schreiben Sie eine Klasse `Toolbox`, die vier Attribute hat: ein Array von `Tool`-Objekten als Fächer der Werkzeugkiste, eine ganzzahlige Variable für die freie Kapazität der Werkzeugkiste, ein String als Name der Werkzeugkiste und eine Konstante, die angibt, wie viele Fächer ein Elektrowerkzeug belegt.

Schreiben Sie außerdem zwei Konstruktoren:

- Ein Konstruktor, der eine Kapazität übergeben bekommt und eine leere Werkzeugkiste mit entsprechender Kapazität erstellt.

- Ein Konstruktor, der eine beliebige Anzahl `Tool`-Objekte übergeben bekommt und eine Werkzeugkiste erstellt, die genau diese Werkzeuge enthält und keine zusätzlichen freien Fächer hat. Freie Fächer entstehen hier also nur, falls auch `null` als `Tool`-Objekt übergeben wird. Die Einschränkung, dass Elektrowerkzeuge immer drei Fächer benötigen, kann hier ignoriert werden, denn bei idealer Platzeinteilung kann man oft viel mehr auf gleichem Raum unterbringen.

Beide Konstruktoren bekommen außerdem einen String übergeben, der als Name der Werkzeugkiste gesetzt wird.

c) Schreiben Sie Methoden, um die freie Kapazität zu lesen, um den Namen der Kiste zu lesen und um das Werkzeug in Fach `i` zu lesen. Falls `i` keine gültige Fachnummer ist, soll `null` zurückgegeben werden. Schreiben Sie außerdem eine Methode, um den Namen zu ändern.

d) Schreiben Sie eine Hilfsmethode `checkRoomForPowerTool`, die den ersten Index `i` ermittelt, an dem ein Elektrowerkzeug in die Werkzeugkiste passen würde. Als Rückgabewert hat die Methode einen `boolean`, der angibt, ob drei freie Plätze in Folge gefunden werden konnten. Als Eingabe bekommt die Methode ein Objekt der Klasse `Wrapper`, die auf der Webseite zur Verfügung steht. Sie speichert einen `int`-Wert und bietet Getter und Setter Methoden für diesen `int`-Wert. Als Seiteneffekt soll dieses `Wrapper`-Objekt so geändert werden, dass es den gefundenen Index `i` speichert.

e) Diskutieren Sie die Sichtbarkeit der Methode `checkRoomForPowerTool`.

f) Schreiben Sie eine Methode `addTool`, die ein Werkzeug zu einer Werkzeugkiste hinzufügt. Elektrowerkzeuge werden an die erste Stelle gespeichert, an der drei Fächer in Folge frei sind. Das Objekt wird in jedes dieser drei Fächer geschrieben. Normale Werkzeuge werden an den ersten freien Platz geschrieben. Materialien werden nur dann neu hinzugefügt, wenn kein Fach mit Materialien gefunden wurde, bevor ein freies Fach gefunden wurde. Die Methode sollte außerdem die Kapazität aktualisieren.

g) Schreiben Sie ausführliche **javadoc**-Kommentare für die gesamte Klasse `Toolbox`.

Aufgabe 2 (Aufzug): (12 + 5 + 0.5 + 1.5 + 4 = 23 Punkte)

In dieser Aufgabe wird eine Aufzugsteuerung in Java geschrieben. Dazu finden Sie auf der Webseite die Datei `AufzugSteuerung.java`, die ein Code-Gerüst enthält.

Hinweise:

- Beachten Sie bei Ihrem Entwurf der Klasse die Datenkapselung und versehen Sie jedes Attribut und jede Methode mit `public`, `private`, `static` und `final` soweit sinnvoll.
- Sie dürfen, unter Beachtung der Datenkapselung und soweit nötig, Hilfsmethoden oder Attribute hinzufügen.

konj.只要

- Auf der Webseite werden die Dateien `Aufzug_Test.java` und `AufzugSteuerung_Test.java` bereitgestellt. Diese enthalten jeweils einen Test für einige Aufgabenteile. Beachten Sie dazu auch die Hinweise bei den Teilaufgaben.
- Denken Sie daran, alle geänderten Dateien neu zu compilieren, bevor Sie die Tests starten.
- Halten Sie sich unbedingt an die vorgegebenen Namen für Klassen, Methodennamen und so weiter, damit Ihr Code getestet werden kann.

a) Schreiben Sie eine Klasse, die einen Aufzug repräsentiert. Ein Aufzug ist dabei wie folgt definiert:

- Ein Aufzug hat ein aktuelles Stockwerk, eine Anzahl Personen, die aktuell im Aufzug sind, und eine Tür, die auf oder zu sein kann.
  Diese Eigenschaften können über Getter (`getAktuellesStockwerk`, `getAnzahlPersonen`, `isTuerAuf`) abgefragt werden.
- Außerdem hat der Aufzug ein maximales und minimales Stockwerk und eine maximale Anzahl an Passagieren. Diese Werte können sich aber nie ändern. Sie können über Getter-Methoden (`getMaxStockwerk`, `getMinStockwerk`, `getMaxPersonen`) abgefragt werden.

- Stockwerke werden durch ganze Zahlen repräsentiert, wobei ein Stockwerk selbstverständlich auch negativ sein kann.
- Es gibt Methoden (`tuerOeffnen`, `tuerSchliessen`), um die Türen zu öffnen und zu schließen.
- Es gibt Methoden (`einsteigen`, `aussteigen`), um die Anzahl der Passagiere zu ändern. Die Methode `einsteigen` bekommt ein Argument vom Typ `int`, das angibt, wie viele Personen einsteigen möchten, und gibt zurück, wie viele Passagiere nicht einsteigen konnten. Die Methode `aussteigen` erhält ebenfalls ein Argument vom Typ `int`, das die Anzahl Personen angibt, die aussteigen möchten. Diese Methode gibt nichts zurück. Beide Methoden sorgen dafür, dass nie weniger als 0 oder mehr als die maximale Anzahl an Passagieren im Aufzug sind. Beachten Sie außerdem, dass nur Passagiere ein- oder aussteigen können, wenn die Tür bereits geöffnet ist.
- Eine Methode `fahren` ermöglicht zu einem gegebenen Stockwerk zu fahren. Dazu müssen die Türen bereits zu sein. Liegt das angegeben Stockwerk außerhalb des vorgesehenen Bereichs, wird stattdessen das am nächsten daran liegende Stockwerk angefahren. Die Methode gibt zurück, ob der Aufzug tatsächlich gefahren ist.
- Alle Aufzüge haben eine ID. Diese beginnen bei 0 und werden fortlaufend an die Objekte der Klasse vergeben. Die ID eines Aufzugs kann über `getID` abgefragt werden.
- Die Klasse soll zwei Konstruktoren mit folgenden Signaturen haben:
  `Aufzug(int maxStockwerk, int minStockwerk, int maxPersonen)` und
  `Aufzug(int maxStockwerk, int maxPersonen)`.
  Beim zweiten Konstruktor wird angenommen, dass das minimale Stockwerk 0 ist. Ein Aufzug startet mit geschlossenen Türen und 0 Personen und befindet sich im minimalen Stockwerk.

Hinweise:

- Sie sollten jetzt die Datei `Aufzug_Test.java` compilieren können. Bei dem Aufruf `java Aufzug_Test` sollte anschließend als letzte Zeile
  `37 Tests abgeschlossen, davon 37 OK und 0 Fehler.`
  ausgegeben werden.
- Falls Fehler auftreten, hilft Ihnen eventuell die Fehlermeldung oder ein Blick in die `main`-Methode der Datei `Aufzug_Test.java` beim Finden des Fehlers.

b) Versehen Sie die Klasse `Aufzug` mit einer ausführlichen `javadoc` Dokumentation. Generieren Sie mit `javadoc` eine HTML Version der Dokumentation und schicken Sie diese zusammen mit dem Programmcode an ihre Tutorin/ihren Tutor. Die generierte HTML Version braucht *nicht* ausgedruckt zu werden.

c) Schreiben Sie einen Aufzählungstyp (Enum) `AufzugZustand`, der die drei Zustände der Aufzugsteuerung repräsentieren kann. Die Aufzugsteuerung kann die Zustände "Warten", "Hoch" und "Runter" haben.

d) Implementieren Sie den Konstruktor der Klasse `AufzugSteuerung`, sodass alle Attribute sinnvoll initialisiert werden. Insbesondere soll die Tür des Aufzugs geöffnet werden und das Attribut `stockwerkAngefragt` mit einem booleschen Array initialisiert werden, dass für so viele Einträge Platz hat wie es Stockwerke beim aktuellen Aufzug gibt. Zu Beginn ist der Aufzug im Zustand "Warten".

Hinweise:

- Sie sollten jetzt die Datei `AufzugSteuerung_Test.java` compilieren können, sofern die vorherigen Teilaufgaben korrekt gelöst wurden. Bei dem Aufruf `java AufzugSteuerung_Test` sollte anschließend die Ausgabe mit den Zeilen
  ```
  Konstruktor-Test:
  OK
  OK
  OK
  -- Fertig --
  ```
  beginnen. Fehler in den folgenden Tests sind hier noch möglich.
- Falls Fehler auftreten, hilft Ihnen eventuell die Fehlermeldung oder ein Blick in die `main`-Methode der Datei `AufzugSteuerung_Test.java` beim Finden des Fehlers.

e) Implementieren Sie die Methode `AufzugSteuerung.aufzugStarten()`. Diese soll wie folgt vorgehen:

(1) Ist die Steuerung im Zustand "Warten", dann wird das vom aktuellen Stockwerk nächstgelegene andere Stockwerk angefahren, das angefragt ist. Falls zwei gleich weit entfernte Stockwerke angefragt sind, fährt der Aufzug erst nach oben. Der Zustand wechselt dabei auf "Hoch" wenn der Aufzug hoch gefahren ist, und "Runter", wenn er runter gefahren ist. Falls kein Stockwerk angefragt ist, so ändert sich der Zustand nicht und der Aufzug bewegt sich nicht. Sind beispielsweise Stockwerk 3, 10 und 11 angefragt, während der Aufzug im 5. Stockwerk wartet, so wird 3 angefahren und der Zustand wechselt auf "Runter". Falls aber 3 und 7 angefragt sind, während der Aufzug im 5. Stockwerk wartet, so würde er nach 7 fahren und in den Zustand "Hoch" wechseln.

(2) Ist die Steuerung im Zustand "Hoch", so wird zum nächsten angefragten Stockwerk über dem aktuellen Stockwerk gefahren. Ist kein höheres angefragt, so wird zum nächsten angefragten Stockwerk unterhalb gefahren und der Zustand wechselt auf "Runter". Ist kein Stockwerk angefragt, wechselt der Zustand auf "Warten".

(3) Ist die Steuerung im Zustand "Runter", verhält sie sich symmetrisch zum Fall, wenn sie im Zustand "Hoch" ist.

Die Information, welche Stockwerke angefragt sind, befindet sich im Array `stockwerkAngefragt`. In allen Fällen soll der entsprechende Eintrag im Array `stockwerkAngefragt` auf `false` gesetzt werden, sobald der Aufzug dorthin gefahren ist. Achten Sie auch darauf, die Tür des Aufzugs zum Fahren zu schließen und danach wieder zu öffnen.
Nutzen Sie die gegebenen Hilfsmethoden in der Klasse `AufzugSteuerung` wenn möglich!

Hinweise:

- Bei dem Aufruf `java AufzugSteuerung_Test` sollte nun, wenn Sie alle Dateien neu compilieren, als letzte Zeile
  `16 Tests abgeschlossen, davon 16 OK und 0 Fehler.`
  ausgegeben werden.
- Falls Fehler auftreten, hilft Ihnen eventuell die Fehlermeldung oder ein Blick in die `main`-Methode der Datei `AufzugSteuerung_Test.java` beim Finden des Fehlers.

Aufgabe 3 (Deck 3): (Codescape)

Lösen Sie die Räume von Deck 3 des Spiels Codescape.