Prof. Dr. J. Giesl

S. Dollase, M. Hark, 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 oben links!**
- 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 Freitag, den 23.11.2018 um 12:00** 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 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 Freitag, den 23.11.2018 um 12:00 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.
- 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.
- Aufgaben, die mit einem * markiert sind, sind Sonderaufgaben mit erhöhtem Schwierigkeitsgrad. Sie tragen nicht zur Summe der erreichbaren Punkte bei, die für die Klausurzulassung benötigt werden, jedoch werden Ihnen die in solchen Aufgaben erreichten Punkte ganz normal gutgeschrieben.

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 (Digitale Signaturen): (1 + 7 + 7 + 4 + 2* + 4 = 23 + 2* Punkte)

Alice möchte für einige Leute digitale Zertifikate ausstellen. Diese Zertifikate möchte Alice später überprüfen können, um sich davon zu überzeugen, dass sie wirklich von ihr ausgestellt wurden.
Natürlich könnte Alice alle Daten speichern und zum Überprüfen in ihrer Datenbank nachsehen. Das klingt für Alice aber viel zu aufwendig, daher entscheidet sie sich, digitale Signaturen zu verwenden. Jedes Zertifikat enthält daher einen Inhalt und eine Signatur. Alice benötigt nun zur Überprüfung nur ein *Secret*[1]. Aus dem Inhalt des Zertifikats und dem Secret wird mit Hilfe eines speziellen Algorithmus und einer kryptografischen Hashfunktion eine Signatur berechnet.
Ohne das Secret von Alice ist es praktisch unmöglich, die korrekte Signatur zu erzeugen. Um später zu prüfen, dass sie das Zertifikat selbst ausgestellt hat, muss sie also nur die Signatur erneut berechnen und mit der Signatur des Zertifikats abgleichen. Andere können diese Überprüfung nicht vornehmen, da man das Secret zur Überprüfung benötigt. Es handelt sich also um ein symmetrisches Signaturverfahren.[2]
In dieser Aufgabe werden die nötigen Klassen implementiert, um mit Hilfe der in Java verfügbaren Hashfunktionen digital signierte Zertifikate zu erzeugen.
Ein Zertifikat besteht aus drei Teilen: Einem *Header*, der den Namen des verwendeten Signaturalgorithmus enthält, einen *Body*, der Paare von Schlüsseln und Werten enthält, und schließlich der *Signatur*. Header und Body bilden zusammen den Inhalt des Zertifikats. Der Aufbau von Zertifikaten wird also durch die folgende Grammatik in EBNF beschrieben:

```
Header = ("HMAC_MD5"|"HMAC_SHA1"|"HMAC_SHA256")
Key_Value_Pair = "'" String "': '" String "'"
Body = Key_Value_Pair { "," Key_Value_Pair }
Signatur = Hexadezimalzahl
```

[1] Ein Passwort oder kryptografischer Schlüssel
[2] Dies ist im Gegensatz zu asymmetrischen Verfahren, bei denen es einen privaten Schlüssel zum Erzeugen und einen öffentlichen Schlüssel zum Überprüfen der Signaturen gibt.

```
Inhalt = Header ";" Body ";"
Zertifikat = Inhalt Signatur
```

Die folgenden beiden Ausdrücke sind syntaktisch korrekte Zertifikate.

```
HMAC_MD5;'i': '42','string': 'string';9b55eef8e51651bb7f265204d281354c

HMAC_SHA256;'i': '1','str': 'value';
93d0de88bef2c1cc63d485e330e067d3096aaa6fcc8c722d2de4d87991a0b54e
```

Auf der Webseite finden Sie eine Klasse `Hasher.java`, die Sie nutzen können. Die Klasse vereinfacht den Umgang mit den Hashfunktionen von `Java` und bietet zusätzlich Methoden, um zwischen `byte[]` und Strings mit Hexadezimalzahlen zu konvertieren. Alle öffentlichen Methoden dieser Klasse verfügen über ausführliche Erklärungen als `Javadoc` Kommentare.
Zusätzlich gibt es eine `main`-Methode in der Klasse `Hasher`, mit der Sie Ihren Code testen können.

Hinweise:

- Beachten Sie bei Ihrem Entwurf der Klassen 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 zu den von Ihnen geschriebenen Klassen hinzufügen.
- Die vorgegebene Klasse `Hasher` **darf nicht verändert werden!**
- Halten Sie sich unbedingt an die vorgegebenen Namen für Klassen, Methodennamen und so weiter, damit Ihr Code getestet werden kann.
- Ihr Code muss nicht robust sein. Wenn falsche Parameter übergeben werden, darf sich Ihr Code beliebig verhalten. Falls z.B. ein String erwartet wird, der eine Zahl darstellt, aber `"a"` übergeben wird, darf Ihr Programm abstürzen.

a) Schreiben Sie einen Aufzählungstyp (Enum) `Algorithm`. Dieser soll die drei möglichen Signaturalgorithmen `HMAC_MD5`, `HMAC_SHA1` und `HMAC_SHA256` repräsentieren können.

b) Zunächst benötigen wir eine einfache Klasse für die Schlüssel-Wert-Paare, die den Body des Zertifikats bilden. Schreiben Sie dazu eine Klasse `Pair`. Ein `Pair`-Objekt soll jeweils einen Schlüssel und einen Wert (als Strings) speichern, die über die Selektoren `getKey` bzw. `getValue` gelesen werden können.

Einmal erstellt, sollen Schlüssel und Wert eines Objekts nicht mehr geändert werden können.

Es soll zwei Konstruktoren geben, die jeweils als ersten Parameter den Schlüssel als String erhalten und als zweiten Parameter einen String oder ein `Integer`-Objekt (das dann in einen entsprechenden String überführt werden muss).

Schließlich soll es eine `toString` Methode geben, die eine Stringrepräsentation des Objekts gemäß der oben gezeigten EBNF Regeln für das Nicht-Terminal `Key_Value_Pair` erzeugt, und eine Methode `fromString` für die umgekehrte Richtung.
Der Aufruf `Pair.fromString("'a': 'something'")` sollte also ein `Pair`-Objekt mit dem Schlüssel `"a"` und dem Wert `"something"` zurückgeben.

Hinweise:

- In der Klasse `String` gibt es eine Methode `split(String pattern)`. Diese liefert ein Array von Strings, das entsteht, wenn man den String bei jedem vorkommen von `pattern` aufspaltet. Ruft man `split(",")` z.B. auf dem String `"a,bcd,e,"` auf, erhält man das Array `{"a", "bcd", "e", ""}`.
- Gehen Sie davon aus, dass weder der Schlüssel noch der Wert das Symbol ' enthalten. Andernfalls darf sich Ihr Code beliebig verhalten.

c) Schreiben Sie eine Klasse `Certificate`, die ein Zertifikat repräsentiert.

Ein Zertifikat besteht aus einem `Algorithm`, der für die Signatur verwendet werden soll, einer beliebigen, aber für ein Objekt festen Anzahl von Schlüssel-Wert-Paaren und einer Signatur. Der `Algorithm` und die

Schlüssel-Wert-Paare sollen dem Konstruktor übergeben werden. Nutzen sie `vararg`-Parameter, damit eine beliebige Anzahl Schlüssel-Wert-Paare übergeben werden kann.

Die Signatur soll außerhalb der Klasse nicht zugreifbar sein. Auch der `Algorithm` ist extern nicht relevant. Für die Schlüssel-Wert-Paare soll es statt Selektoren eine Funktion `get` geben, die einen Schlüssel als String übergeben bekommt und den zugehörigen Wert zurückliefert, falls es ein entsprechendes Schlüssel-Wert-Paar in dem Zertifikat gibt. Ansonsten soll `null` zurückgegeben werden.

Außerdem soll die Klasse eine Methode `fromString` besitzen, die ein Objekt der Klasse aus einem String gemäß der EBNF Regeln für das Nicht-Terminal `Zertifikat` erstellt.
Ruft man zum Beispiel `Certificate.fromString` auf dem ersten Beispiel-Zertifikat oben auf, sollte man ein Objekt vom Typ `Certificate` erhalten, bei dem der Algorithm `HMAC_MD5` ist, die Signatur `9b55eef8e51651bb7f265204d281354c` und das genau zwei Schlüssel-Wert-Paare besitzt. Die Schlüssel sollten `"i"` und `"string"` sein und die zugehörigen Werte `"42"` bzw. `"string"`.

Hinweise:

- Gehen Sie davon aus, dass weder die Schlüssel noch die Werte die Symbole `'` oder `;` enthalten. Andernfalls darf sich Ihr Code beliebig verhalten.

d) Schreiben Sie in der Klasse `Certificate` eine Methode `getSignedString(String secret)`, die eine Signatur als String aus dem gegebenen `secret` mit dem Algorithm des aktuellen `Certificate`-Objekts erstellt, die Signatur des aktuellen `Certificate`-Objekts mit der Hexadezimalstring-Darstellung dieser String-Signatur belegt und das vollständige Zertifikat als String zurück liefert, wie in der EBNF-Grammatik angegeben. Hat das aktuelle `Certificate`-Objekt bereits eine Signatur, z.B. weil es aus einem String erstellt wurde, soll diese nicht verändert werden!

Schreiben Sie dazu zunächst eine nur intern verwendete Hilfsmethode `getHeaderBodyString`, die eine String-Repräsentation des Inhalts des aktuellen `Certificate`-Objekts gemäß der EBNF Regeln für das Nicht-Terminal `Inhalt` zurückliefert. Für ein Objekt mit den Schlüssel-Wert-Paaren `'a': '42'` und `'foo': 'bar'` und dem `Algorithm HMAC_MD5` würde die Methode den String
`HMAC_MD5;'a': '42','foo': 'bar';` zurückliefern.

Um die Signatur zu berechnen, verwenden Sie die passende Funktion aus der Klasse `Hasher`. Ist der `Algorithm` des aktuellen `Certificate`-Objekts z.B. `HMAC_SHA256`, müssten Sie
`Hasher.sha256Hmac(String toSign, String secret)` mit passenden Parametern aufrufen. Als erster Parameter sollte dabei das Ergebnis von `getHeaderBodyString` übergeben werden.

Schreiben Sie außerdem eine Methode `validateSignature(String secret)`. Diese soll erneut eine Signatur berechnen und diese mit der in dem Objekt gespeicherten Signatur vergleichen. Sind beide gleich, soll `true` zurückgeliefert werden, sonst `false`.

Hinweise:

- Sie können die Methode `substring(int beginIndex, int endIndex)` verwenden. Dieser erzeugt einen Teilstring, der genau die Zeichen von `beginIndex` bis `endIndex-1` enthält.
  `s.substring(0, s.length())` würde also den kompletten String `s` kopieren,
  `"abcd".substring(1,3)` würde den String `"bc"` zurückgeben.
- Verwenden Sie die Methode `byteArrayToHex` aus der Klasse `Hasher` um `byte`-Arrays in Strings (mit der entsprechenden Hexadezimalzahl) zu konvertieren. Ebenso enthält `Hasher` eine Methode für die umgekehrte Transformation.

e)* Alice bietet einen Webservice an, über den Bob und Chuck ihre Zertifikate überprüfen können. Leider ist Chuck nicht vertrauenswürdig und könnte versuchen, Zertifikate zu fälschen. Da Chuck die Zeit mitstoppen könnte, die eine Überprüfung dauert, muss Alice ihren Code gegen *Timing-Attacken* absichern. Bei einer Timing-Attacke stoppt Chuck die Laufzeit einer Operation und kann daraus Rückschlüsse auf geheime Informationen ziehen. In diesem Fall ist die kritische Information, wie weit die neu berechnete Signatur mit der vorhandenen übereinstimmt.

Die Methode `equals` der Klasse String vergleicht zwei Strings nur bis zum ersten Unterschied. Je länger der Vergleich dauert, desto mehr Bytes der Signatur sind richtig. Chuck kann also Byte für Byte die Signatur fälschen. So braucht er nur $256 \cdot 32$ Versuche, um eine SHA-256 Signatur zu fälschen, statt 256^{32} Versuche.

Doch es ist noch schlimmer! Für einen Angriff würde es schon reichen, wenn man weiß, ob nur ein einziges Byte der gefälschten Signatur auch in der korrekten Signatur an gleicher Position auftritt. Dies würde Chuck ermöglichen, die Signatur in höchstens $32 \cdot 32 + 256$ Versuchen zu fälschen, weil nicht jeder der 256 möglichen Byte-Werte in der aus 32 Byte bestehenden Signatur auftreten kann. Also muss es genau so lang dauern ein richtiges Byte zu verarbeiten, wie es braucht ein falsches Byte zu verarbeiten.

Verändern Sie die Methode `validateSignature(String secret)` so, dass keine der oben beschriebenen Timing-Attacken mehr möglich sind. Kommentieren Sie ihren Code, sodass klar wird, wie die Timing-Attacke vermieden wird.

Hinweise:

- Die Länge von `secret` oder des Inhalts ist keine kritische Information. In der Praxis werden grundsätzlich Geheimnisse mit der gleichen Länge wie die Block-Größe des Hash-Algorithmus verwendet. Die Block-Größe aller hier verwendeten Algorithmen ist 512 Bit.
- Diese Aufgabe benötigt große Präzision. Kleinste Fehler führen dazu, dass kritische Informationen verraten werden. Beispielsweise benötigt `true && 1 == 1` eine längere Zeit zur Auswertung als `false && 1 == 1`

f) Versehen Sie die Klassen `Certificate` und `Pair` mit einer ausführlichen Javadoc Dokumentation. Private Methoden und private Attribute müssen nicht zwingend dokumentiert werden. 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.

Tutoraufgabe 3 (Programmanalyse):

Lösen Sie die folgende Aufgabe ohne Einsatz eines Computers. Bedenken Sie, dass Sie in einer Prüfungssituation ebenfalls keinen Computer zur Verfügung haben.
Betrachten Sie das folgende Programm:

```
public class A {
    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(a1.f(5));
        System.out.println(a1.d);
        System.out.println(a1.f(Long.valueOf(2)));

        A a2 = new A(1,1);
        System.out.println(a2.i);
        System.out.println(a2.d);
    }

    private Integer i;
    private double d;

    public A() {
        this.i = 1;
        this.d = 4;
    }

    public A(Integer x, double y) {
        this.i = x;
        this.d = y;
    }

    public A(int x, double y) {
        this.i = 3;
        this.d = x + y;
    }

    public int f(Integer x) {
        return this.i + x;
    }

    public int f(double i) {
        this.d = i;
        return this.i;
    }
}
```

Geben Sie die Ausgabe dieses Programms an, wenn die `main`-Methode ausgeführt wird. Begründen Sie Ihre Antwort.

Aufgabe 4 (Programmanalyse): (4 Punkte)

Lösen Sie die folgende Aufgabe ohne Einsatz eines Computers. Bedenken Sie, dass Sie in einer Prüfungssituation ebenfalls keinen Computer zur Verfügung haben.
Betrachten Sie das folgende Programm:

```
public class B {
    public static void main(String[] args) {
        B b = new B();

        b.a(Long.valueOf(100));
        b.a(Double.valueOf(100));
        b.a(Integer.valueOf(100));

        double r1 = b.b(100D);
        int r2 = (int) b.b(100);

        c(Integer.valueOf(100), "0");
        c(100L, "0");
        c(100L, '0');
    }

    public void a(int p) {
        System.out.println("a1");
    }

    public void a(double p) {
        System.out.println("a2");
    }

    public void a(Double p) {
        System.out.println("a3");
    }

    public int b(double p) {
        System.out.println("b1");
        return 0;
    }

    public double b(int p) {
        System.out.println("b2");
        return 0;
    }

    public static void c(Long p1, int p2) {
        System.out.println("c1");
    }

    public static void c(long p1, String p2) {
        System.out.println("c2");
    }

    public static void c(Long p1, String p2) {
        System.out.println("c3");
    }
}
```

Geben Sie die Ausgabe dieses Programms an, wenn die `main`-Methode ausgeführt wird. **Begründen Sie Ihre Antwort!**

Tutoraufgabe 5 (Rekursion):

Betrachten Sie folgende Methode:

```
public static int arraySum(int[] a) {
    int res = 0;
    for (int i = 0; i < a.length; i++) {
        res += a[i];
    }
    return res;
}
```

Schreiben Sie eine statische Methode `arraySumRecursive`, welche ein `int`-Array erhält und eine `int`-Zahl zurückliefert, sodass für jedes `int`-Array `a` die Aufrufe `arraySum(a)` und `arraySumRecursive(a)` das gleiche Ergebnis liefern. Sie dürfen in dieser Aufgabe keine Schleifen verwenden. Die Verwendung von Rekursion ist hingegen erlaubt (inklusive der Definition von Hilfsmethoden).

Hinweise:

- Wenn Sie Ihr Programm mit der Anweisung `import java.util.Arrays;` beginnen, können Sie die vordefinierte statische Methode `Arrays.copyOfRange` benutzen. Hierbei liefert `Arrays.copyOfRange(a, i, j)` ein neues Array mit den Werten `a[i], a[i+1],..., a[j]` zurück.

Aufgabe 6 (Rekursion): (5 Punkte)

Gegeben sei die Klasse `Minimum` in der Datei `Minimum.java`. Vervollständigen sie die statische Methode `arrayMin`, die das kleinste Element eines `int`–Arrays zurückgibt. In dieser Aufgabe sei das kleinste Element des `int[]`–Arrays `null` bzw. des `int[]`–Arrays der Länge `0` die größte darstellbare `int`–Zahl. Das Benutzen von **Schleifen** ist **nicht gestattet**.

Hinweise:

- Den größten darstellbaren `int`–Wert können Sie durch den Aufruf `Integer.MAX_VALUE` erhalten.
- Wenn Sie Ihr Programm mit der Anweisung `import java.util.Arrays;` beginnen, können Sie die vordefinierte statische Methode `Arrays.copyOfRange` benutzen. Hierbei liefert `Arrays.copyOfRange(a, i, j)` ein neues Array mit den Werten `a[i], a[i+1],..., a[j-1]` zurück.
- Die Benutzung von vordefinierten Java–Methoden außer den hier explizit genannten **ist nicht gestattet.**
- Um Ihr Programm zu testen, legen Sie die auf der Website bereitgestellte Datei `Test.class` in dasselbe Verzeichnis wie `Minimum.java`. Führen sie nach dem Kompilieren von `Minimum.java` den Befehl `java Test` aus.

Aufgabe 7 (Codescape): (Codescape)

Lösen Sie die Räume von **Deck 3** des Spiels Codescape.
Ihre Lösung für Räume dieses Codescape Decks wird nur dann für die Zulassung gezählt, wenn Sie die Lösung bis Freitag, den 23.11.2018 um 12:00 abschicken.