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 5.12.2017 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 5.12.2017 um 08:00 Uhr an Ihre Tutorin/Ihren Tutor.
  Stellen Sie sicher, dass Ihr Programm von `javac` **akzeptiert** wird und über eine geeignete `main`-Methode verfügt, um mit `java` ausgeführt zu werden. Verändern Sie außerdem den gegebenen Code nur an den angegebenen Stellen. 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.

Tutoraufgabe 1 (Rekursive Datenstrukturen):

In dieser Aufgabe geht es um einfach verkettete Listen als Beispiel für eine dynamische Datenstruktur. Wir legen hier besonderen Wert darauf, dass eine einmal erzeugte Liste nicht mehr verändert werden kann. Achten Sie also in der Implementierung darauf, dass die Attribute der einzelnen Listen-Elemente `nur` im Konstruktor geschrieben werden.
Für diese Aufgabe benötigen Sie die Klasse `ListExercise.java`, welche Sie von unserer Webseite herunterladen können.
In der gesamten Aufgabe dürfen Sie **keine Schleifen** verwenden (die Verwendung von Rekursion ist hingegen erlaubt). Ergänzen Sie in Ihrer Lösung für alle öffentlichen Methoden außer Konstruktoren und Selektoren geeignete `javadoc`-Kommentare.

a) Erstellen Sie eine Klasse `List`, die eine einfach verkettete Liste als rekursive Datenstruktur realisiert. Die Klasse `List` muss dabei mindestens die folgenden öffentlichen Methoden und Attribute enthalten:

- `static final List EMPTY` ist die einzige `List`-Instanz, die die *leere* Liste repräsentiert
- `List(List n, int v)` erzeugt eine neue Liste, die mit dem Wert `v` beginnt, gefolgt von allen Elementen der Liste `n`
- `List getNext()` liefert die von `this` referenzierte Liste ohne ihr erstes Element zurück
- `int getValue()` liefert das erste Element der Liste zurück

b) Implementieren Sie in der Klasse `List` die öffentlichen Methoden `int length()` und `String toString()`. Die Methode `length` soll die Länge der Liste zurück liefern. Die Methode `toString` soll eine textuelle Repräsentation der Liste zurück liefern, wobei die Elemente der Liste durch Kommata separiert hintereinander stehen. Beispielsweise ist die textuelle Repräsentation der Liste mit den Elementen 2, 3 und 1 der `String "2, 3, 1"`.

c) Ergänzen Sie diese Klasse darüber hinaus noch um eine Methode `getSublist`, welche ein Argument i vom Typ `int` erhält und eine unveränderliche Liste zurück liefert, welche die ersten i Elemente der aktuellen Liste enthält. Sollte die aktuelle Liste nicht genügend Elemente besitzen, wird einfach eine Liste mit allen Elementen der aktuellen Liste zurück gegeben.

d) Vervollständigen Sie die Methode `merge` in der Klasse `ListExercise.java`. Diese Methode erhält zwei Listen als Eingabe, von denen wir annehmen, dass diese bereits aufsteigend sortiert sind. Sie soll eine Liste zurück liefern, die alle Elemente der beiden übergebenen Listen in aufsteigender Reihenfolge enthält.

Hinweise:

- Verwenden Sie zwei Zeiger, die jeweils auf das kleinste noch nicht in die Ergebnisliste eingefügte Element in den Argumentlisten zeigen. Vergleichen Sie die beiden Elemente und fügen Sie das kleinere ein, wobei Sie den entsprechenden Zeiger ein Element weiter rücken. Sobald eine der Argumentlisten vollständig eingefügt ist, können die Elemente der anderen Liste ohne weitere Vergleiche hintereinander eingefügt werden.

e) Vervollständigen Sie die Methode `mergesort` in der Klasse `ListExercise.java`. Diese Methode erhält eine unveränderliche Liste als Eingabe und soll eine Liste mit den gleichen Elementen in aufsteigender Reihenfolge zurückliefern. Falls die übergebene Liste weniger als zwei Elemente enthält, soll sie unverändert zurück geliefert werden. Ansonsten soll die übergebene Liste mit der vorgegebenen Methode `divide` in zwei kleinere Listen aufgespalten werden, welche dann mit `mergesort` sortiert und mit `merge` danach wieder zusammengefügt werden.

Hinweise:

- Sie können die ausführbare `main`-Methode verwenden, um das Verhalten Ihrer Implementierung zu überprüfen. Um beispielsweise die unveränderliche Liste `[2,4,3]` sortieren zu lassen, rufen Sie die `main`-Methode durch `java ListExercise 2 4 3` auf.

Aufgabe 2 (Rekursive Datenstrukturen): (3+5+5+3+5+5 = 26 Punkte)

Um den Umgang mit rekursiven Datenstrukturen zu üben, verwenden wir in dieser Aufgabe *Polynome von Literalen*. Ein Polynom ist wie folgt induktiv definiert:

- Eine Gleitkommazahl ist ein Literal (eine sogenannte Konstante).
- Eine Variable ist ein Literal.
- Eine Potenz eines Literals hat ein Literal als Basis und einen nicht-negativen ganzzahligen Exponenten.
- Eine Potenz eines Literals ist ein Monom.
- Wenn a eine Potenz ist und b ein Monom, dann ist auch $a * b$ ein Monom.
- Monome sind auch Polynome.
- Wenn m ein Monom ist und p ein Polynom, dann ist auch $m + p$ ein Polynom.

Beispielsweise sind (3.2) und x Literale. Eine Potenz ist z.B. $x\hat{}3$. Damit ist dann $x\hat{}3 * (3.2)\hat{}2$ ein Monom. Schließlich ist $x\hat{}3 * (3.2)\hat{}2 + x\hat{}0 + (5.0)\hat{}1$ ein Polynom.
Auf der Homepage finden Sie den Aufzählungstypen `Typ` und die Klassen `Literal`, `Power`, `Monomial` und `Polynomial`. Wir repräsentieren Literale als Objekte der Klasse `Literal`, wobei hierfür die Attribute die folgende Bedeutung haben:

- **typ** gibt an, ob das Literal eine Konstante oder eine Variable ist.

- **value** gibt den Wert einer Konstanten an. Bei Variablen spielt dieses Attribut keine Rolle und wird auf `0` gesetzt.
- **name** gibt den Namen einer Variablen an. Bei Konstanten spielt dieses Attribut keine Rolle und wird auf `""` gesetzt.

Die Variable x ist also durch ein `Literal`-Objekt mit `type = VAR`, `name = x` und `value = 0.` repräsentiert, die Konstante (2.0) durch ein `Literal`-Objekt mit `type = VALUE`, `name = ""` und `value = 2.0`.
Wir repräsentieren Potenzen als Objekte der Klasse `Power`, wobei hierfür die Attribute die folgende Bedeutung haben:

- **literal** beinhaltet das Literal.
- **exponent** ist der Exponent. Wir betrachten nur nicht-negative Exponenten.

Die Potenz `x^3` ist also ein `Power`-Objekt mit Attribut `exponent = 3` und `literal = x`, welches wie eben dargestellt ist.
Wir repräsentieren Monome als Objekte der Klasse `Monomial`, wobei hierfür die Attribute die folgende Bedeutung haben:

- **factor** beinhaltet die linkeste Potenz.
- **factors** beinhaltet das restliche Monom.

Monome sind also als einfach verkettete Liste von Potenzen dargestellt. Das Monom $x * y * z$ ist also als `Monomial`-Objekt mit `factor = x^1` und `factors = y^1*z^1` dargestellt.
Wir repräsentieren Polynome als Objekte der Klasse `Polynomial`, wobei hierfür die Attribute die folgende Bedeutung haben:

- **summand** beinhaltet den ersten Summanden.
- **summands** beinhaltet das restliche Polynom.

Polynome sind also einfach verkettete Listen von Monomen.
Das Polynom `x^3 + x*y*z + (0.0)` ist also durch ein `Polynomial`-Objekt mit `summand = x^3` und `summands = x^1*y^1*z^1 + (0.0)^1` dargestellt.
Verwenden Sie in dieser Aufgabe keine Schleifen, sondern ausschließlich Rekursion!
Verändern Sie in den Methoden das jeweilige Eingabe-Objekt (`this`) nicht, sondern erzeugen Sie bei Bedarf neue Objekte.
Ergänzen Sie in Ihrer Lösung für alle öffentlichen Methoden außer Konstruktoren und Selektoren geeignete `javadoc`-Kommentare.

Hinweise:

- Testen Sie Ihre Implementierung mit Hilfe der gegebenen `main`-Methode in der Klasse `Polynomial`, die noch auskommentiert ist. Nutzen Sie dazu die noch auskommmentierte Methode `Polynom parse(String input)` mit ihren ebenfalls noch auskommentierten Hilfsmethoden. Diese Methode liest ein Polynom aus einem String ein. Konstanten müssen dazu in Klammern geschrieben werden, um Sie von den Exponenten der Potenzen zu unterscheiden.
- Achten Sie auf den korrekten Umgang mit dem `null`-Pointer.
- Verwenden Sie in Ihrer Implementierung an den passenden Stellen eine `switch`-Anweisung zur Fallunterscheidung anhand des Typs des jeweiligen Literals. Da der Compiler nicht erkennt, dass Sie alle Möglichkeiten durch `case`-Ausdrücke berücksichtigt haben, ist es in nicht-`void` Methoden unter Umständen nötig, eine zusätzliche (unerreichbare) `return`-Anweisung in einem letzten `default`-Fall zu schreiben. In diesen Fällen ist es erlaubt, beliebige Werte (beispielsweise `null`) zurückzugeben.
- Ignorieren Sie die folgenden möglichen Probleme und gehen Sie davon aus, dass diese nicht auftreten:
  - Über- und Unterläufe
  - unpräzise `double`-Arithmetik

Implementieren Sie nun die Klassen `Literal`, `Power`, `Monomial` und `Polynomial` wie folgt:

a) Schreiben Sie für jede Klasse drei Konstruktoren, sowie für jedes Attribut eine `get`-Methode. In den `.java`-Dateien auf der Website finden Sie bereits die Signaturen. Alle Klassen haben einen Kopierkonstruktor, der ein gegebenes Objekt kopiert. Desweiteren gibt es folgende Konstruktoren.

- Die Klasse `Literal` hat weiterhin einen Konstruktor, der eine Konstante erzeugt und einen Konstruktor, der eine Variable erzeugt.
- Die Klasse `Power` hat einen Konstruktor, der ein Literal zu einer Potenz mit Exponent 1 liftet und einen Konstruktor, der aus einem gegebenen Literal und einem Exponenten eine Potenz erzeugt. Falls der Exponent im formalen Parameter negativ ist, wird stattdessen eine Potenz mit dem Exponent 0 erzeugt.
- Die Klasse `Monomial` hat einen Konstruktor, der eine Potenz zu einem Monom liftet, d.h. ein Produkt mit nur einem Faktor darstellt. Außerdem gibt es einen Konstruktor, der aus einer Potenz und einem Monom ein neues Monom generiert.
- Die Klasse `Polynomial` hat einen Konstruktor, der ein Monom zu einem Polynom liftet, d.h. eine Summe mit nur einem Summanden darstellt. Außerdem gibt es einen Konstruktor, der aus einem Monom und einem Polynom ein neues Polynom erzeugt.

Hinweise:

- Nutzen Sie anstelle von `null` die statischen Objekte `public static final Monomial ONE` und `public static final Polynomial ZERO`, die das leere Monom (interpretiert als der Wert 1) und das leere Polynom (interpretiert als der Wert 0) repräsentieren.

b) Wie Sie vielleicht bemerkt haben, ist die Darstellung des Polynoms `0` nicht eindeutig. Die default Darstellung ist das leere Polynom `ZERO`. Allerdings ist auch eine Summe 0, in der jeder Summand 0 ist. Schreiben Sie deshalb eine Methode `public boolean isZero()`, die genau dann `true` liefert, wenn alle Summanden des Polynoms 0 oder `null` sind.

Hinweise:

- Gehen Sie davon aus, dass ein Literal genau dann 0 ist, wenn es die Konstante `0.0` ist.
- Gehen Sie davon aus, dass eine Potenz genau dann 0 ist, wenn entweder die Basis `null` ist oder sowohl der Exponent größer 0 und die Basis 0 ist.
- Ein Monom ist genau dann 0, wenn es mindestens eine Potenz enthält, die 0 ist.
- Beachten Sie die korrekte Behandlung des `null`-Pointers. Summanden eines Polynoms, Faktoren eines Monoms oder Literale einer Potenz können `null` sein.

c) Schreiben Sie in jeder Klasse eine Methode `toString()`. Für eine Variable soll nur der Name der Variablen zurückgegeben werden. Eine Konstante soll in Klammern zurückgegeben werden.

Eine Potenz soll als String `base^exponent` ausgegeben werden. Ist der Exponent `0`, so soll `1` ausgedruckt werden. Ist der Exponent `1`, so soll nur das Literal ausgedruckt werden.

Ein Monom soll durch das Symbol `*` getrennt ausgegeben werden, das leere Monom als `1`.

Ein Polynom soll durch das Symbol `+` getrennt ausgegeben werden, das Nullpolynom als `0`.

Hinweise:

- Achten Sie in *dieser und den weiteren Teilaufgaben* auf den richtigen Umgang mit dem `null`-Pointer. In einem Polynom wird ein Summand, der `null` ist stets wie eine 0 behandelt und in einem Monom wird ein Faktor, der `null` ist, stets wie eine 1 behandelt.
- Sie können die Ausgabe optimieren, in dem Sie Summanden, die 0 sind, oder Faktoren, die 1 sind, ignorieren.

d) Schreiben Sie Methoden `int getDegree()`, die den Grad eines Literals, einer Potenz, eines Monoms bzw. eines Polynoms zurückgeben. Der Grad ist wie folgt induktiv definiert:

- Eine Konstante hat Grad 0.
- Eine Variable hat Grad 1.

- Eine Potenz $p\hat{}e$ hat den Grad $\text{Grad}(p) \cdot e$. Falls die Basis p `null` ist, hat die Potenz den Grad 0.
- Ein Monom hat als Grad die Summe der Grade der Faktoren. Falls das Monom jedoch 0 ist, so hat es Grad 0.
- Ein Polynom hat als Grad das Maximum der Grade der Summanden.

Ruft man also `monomial.getDegree()` auf, wobei `monomial` das `Monomial`-Objekt $x\hat{}2 * y * (3.0)\hat{}5$ ist, so ist das Ergebnis `3`. Ruft man hingegen `polynomial.getDegree()` auf, wobei `polynomial` das `Polynomial`-Objekt $x\hat{}2 * y * (3.0)\hat{}5 + y * z\hat{}8 + (2.0)$ ist, so ist das Ergebnis `9`, denn $x\hat{}2 * y * (3.0)\hat{}5$ hat Grad 3, $y * z\hat{}8$ hat Grad 9 und (2.0) hat Grad 0.

Hinweise:

- Nutzen Sie die Methode `isZero()`, um das Nullpolynom bzw. Monome, die 0 sind korrekt zu behandeln.
- Benutzen Sie die Methode `Math.max(int a, int b)` , die das Maximum zweier Integer berechnet. Wenn Sie z.B. das Maximum von x und y bestimmen wollen, können Sie einfach `Math.max(x,y)` in ihr Programm schreiben, sofern x und y automatisch zu Integern gecastet werden können.

e) Schreiben Sie eine Methode `Polynomial substitute(String toSubstitute, double value)`, die alle Vorkommen einer Variablen in einem Polynom durch die Konstante mit dem Wert `value` ersetzt. Wenn das Polynom die eingegebene Variable nicht enthält, dann wird eine Kopie des Polynoms zurückgegeben. So liefert der Aufruf `polynomial.substitute("y", 3.0)`, wobei `polynomial` das `Polynomial`-Objekt $x\hat{}2 * y * (3.0)\hat{}5 + y * z\hat{}8 + 2.0$ ist, den Rückgabewert $x\hat{}2 * (3.0) * (3.0)\hat{}5 + (3.0) * z\hat{}8 + (2.0)$. Hingegen liefert der Aufruf `polynomial.substitute("a", 0.)` auf demselben Objekt den Rückgabewert $x\hat{}2 * y * (3.0)\hat{}5 + y * z\hat{}8 + (2.0)$.

Hinweise:

- Nutzen Sie die Methode `isZero()`, um das Nullpolynom korrekt zu behandeln.
- Beachten Sie die induktive Definition eines Polynoms. Schreiben Sie geeignete Hilfsmethoden in den Klassen `Monomial`, `Power` und `Literal`.

f) Schreiben Sie eine Methode `double evaluate(double default)`, die ein Polynom auswertet, indem alle noch vorhandenen Variablen durch den Wert `default` ersetzt werden. Beispielsweise wertet der Aufruf `polynom.evaluate(1.)`, wobei `polynom` das `Polynomial`-Objekt $(2.0) * x\hat{}2 * y + (1.0)$ ist, zu 3.0 aus.

Hinweise:

- Nutzen Sie die Methode `isZero()`, um das Nullpolynom korrekt zu behandeln.
- Beachten Sie die induktive Definition eines Polynoms. Schreiben Sie geeignete Hilfsmethoden in den Klassen `Monomial`, `Power` und `Literal`.
- Nutzen Sie die Methode `Math.pow(double base, double exponent)`, die die Potenz *base*ˆ*exponent* berechnet. Sie können in Ihrem Code `Math.pow(x,y)` aufrufen, um x^y zu berechnen, wenn x und y automatisch zu Gleitkommazahlen gecastet werden können.

Tutoraufgabe 3 (Hierarchie):

Ziel dieser Aufgabe ist die Erstellung einer Hierarchie zur Verwaltung von verschiedenen Möglichkeiten, seinen Urlaub zu verbringen.

- Ein Urlaub ist in unserem Modell ein Aktivurlaub, eine Kreuzfahrt, ein Strandurlaub oder ein Kultururlaub. Von jedem Urlaub wissen wir die Dauer (in Tagen) und den Preis des Urlaubs.
- Ein Aktivurlaub zeichnet sich dadurch aus, dass man viele Kalorien verbraucht. Deswegen ist zu jedem Aktivurlaub bekannt, wie viele Gramm Körpergewicht man durch die Anstrengung pro Tag abnimmt.
- Ein Skiurlaub ist ein spezieller Aktivurlaub. Für einen solchen Urlaub ist bekannt, wie viele Ski-Pisten in unmittelbarer Nähe zur Verfügung stehen.

- Ein Wanderurlaub ist ebenfalls ein spezieller Aktivurlaub. Die Kilometer-Anzahl der verfügbaren Wanderwege ist für jeden Wanderurlaub bekannt.
- Zusätzlich gibt es die Möglichkeit, seinen Urlaub mit einer Kreuzfahrt zu verbringen. Hierfür ist der Name des Schiffes und die Anzahl der Sterne des Bord-Restaurants bekannt.
- Alternativ zu den Aktivurlauben und Kreuzfahrten gibt es auch die Möglichkeit eines Strandurlaubs. Die Länge des Strandes zeichnet diesen Urlaub aus.
- Ein besonderer Strandurlaub ist der Party-Strandurlaub, der besonders bei jungen Leuten beliebt ist. Für einen solchen Urlaub ist besonders relevant, wie viele verschiedene Clubs direkt am Strand liegen.
- Zu guter Letzt kann auch ein Kultururlaub gebucht werden. Bei einem solchen Urlaub ist angegeben, wie viele Museen besichtigt werden können.
- Jeder Urlaub soll eine Methode `ausgabe()` besitzen, die eine Beschreibung des Urlaubs als `String` zurückgibt. Begründen Sie Ihre Entscheidung bezüglich der Frage, in welchen Klassen diese Methode implementiert werden sollte.
- Was wäre ein Urlaub ohne Urlauber: Sie sind durch ihren Namen gekennzeichnet und haben die Möglichkeit, Urlaube zu buchen.

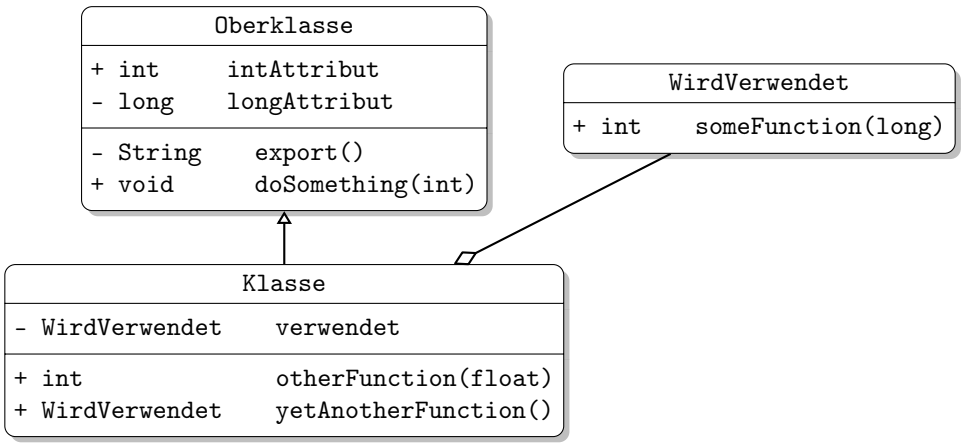

Abbildung 1: Graphische Notation zur Darstellung von Klassen.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Urlauben. Notieren Sie keine Konstruktoren. Um Schreibarbeit zu sparen, brauchen Sie keine Selektoren anzugeben. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen zusammengefasst werden.
Verwenden Sie hierbei die Notation aus Abb. 1. Eine Klasse wird hier durch einen Kasten beschrieben, in dem der Name der Klasse sowie Attribute und Methoden in einzelnen Abschnitten beschrieben werden. Weiterhin bedeutet der Pfeil $B \longrightarrow\!\!\triangleright\ A$, dass A die Oberklasse von B ist (also `class B extends A`) und $A \diamond\!\!- B$, dass A den Typen B in den Typen seiner Attribute oder in den Ein- oder Ausgabeparametern seiner Methoden verwendet. Benutzen Sie ein `-` um `private` und ein `+` um `public` abzukürzen.
Tragen Sie keine vordefinierten Klassen (String, etc.) oder Pfeile dorthin in Ihr Diagramm ein.

Aufgabe 4 (Hierarchie): (6 Punkte)

In dieser Aufgabe sollen Sie einen Teil der Schiffahrt modellieren.

- Ein Schiff kann ein Segelschiff oder ein Motorschiff sein. Jedes Schiff hat eine Länge und eine Breite in Metern und eine Anzahl an Besatzungsmitgliedern.

- Ein Segelschiff zeichnet sich durch die Anzahl seiner Segel aus.
- Die wichtigste Kennzahl eines Motorschiffs ist die PS-Stärke des Motors.
- Ein Kreuzfahrtschiff ist ein Motorschiff. Es hat eine Anzahl an Kabinen und kann das Schiffshorn ertönen lassen.
- Eine Yacht ist ein Segelschiff. Yachten können in Privatbesitz sein oder nicht.
- In einem aufwendigen Verfahren kann eine Yacht zu einem Kreuzfahrtschiff umgebaut werden.
- Schlepper sind Motorschiffe mit einer Anzahl von Schleppseilen. Ein Schlepper kann ein beliebiges Schiff ziehen.
- Frachter sind Motorschiffe, die durch ihre Containerstellplätze und die Größe des Treibstofftanks in Litern ausgezeichnet sind. Ein Frachter kann mit Containern be- und entladen werden.
- Ein Container zeichnet sich durch seine Zieladresse und das Gewicht seines Inhaltes aus. Zudem kann der Inhalt gefährlich sein oder nicht.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Schiffen. Notieren Sie keine Konstruktoren. Um Schreibarbeit zu sparen, brauchen Sie keine Selektoren anzugeben. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen zusammengefasst werden, falls dies sinnvoll ist. Ergänzen Sie außerdem geeignete Methoden, um das Beladen, Entladen, das Umbauen, das Ziehen und das Erklingen lassen des Horns abzubilden.
Verwenden Sie hierbei die Notation aus der entsprechenden Tutoriumsaufgabe.

Aufgabe 5 (Deck 5): (Codescape)

Lösen Sie die Räume von Deck 5 des Spiels Codescape.

Bitte nutzen Sie die Umfrage am Beginn dieses Decks, um der Gruppe “Medien für die Lehre” und uns Feedback zu Codescape zu geben.