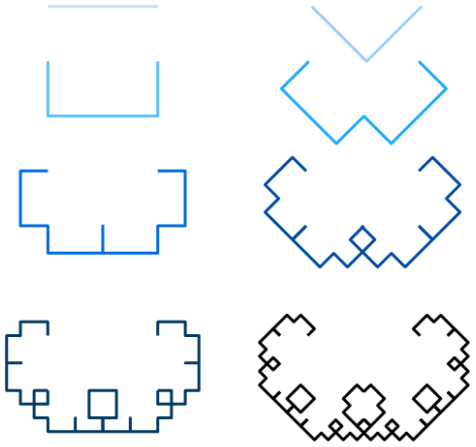
Prof. Dr. J. Giesl
S. Dollase, M. Hark, D. Korzeniewski

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 30.11.2018 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 30.11.2018 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.
- Für einige Programmieraufgaben benötigen Sie die Java Klasse `SimpleIO`. Diese können Sie auf der Website herunterladen.
- 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 (Fraktale):

In dieser Aufgabe soll die fraktale Struktur der Lévy–C–Kurven mit Hilfe der Klasse `Canvas` gezeichnet werden, die auf der Website zu finden ist. Verwenden Sie das Programm Javadoc, um die Schnittstellendokumentation dieser Klasse zu erzeugen.
Eine Lévy–C–Kurve 0. Ordnung besteht aus einer geraden Linie. Kurven i-ter Ordnung werden gebildet, indem man zunächst eine Lévy–C–Kurve $(i-1)$-ter Ordnung zeichnet. Vom letzten Strich dieser Kurve aus zeichnet man dann in einem Winkel von $90° - 90° \cdot (i-1)$ eine weitere Lévy–C–Kurve $(i-1)$-ter Ordnung. Ein Winkel von $0°$ bedeutet hierbei, dass die letzte Linie der Kurve gerade in die erste Linie der zweiten Kurve über geht. Positive Winkel bedeuten eine Ecke im Uhrzeigersinn, negative Winkel eine Ecke entgegen dem Uhrzeigersinn. Die folgende Grafik zeigt Lévy–C-Kurven der Ordnungen 0 bis 7. In ihrer Implementierung werden die Kurven zum Teil *anders gedreht* sein.



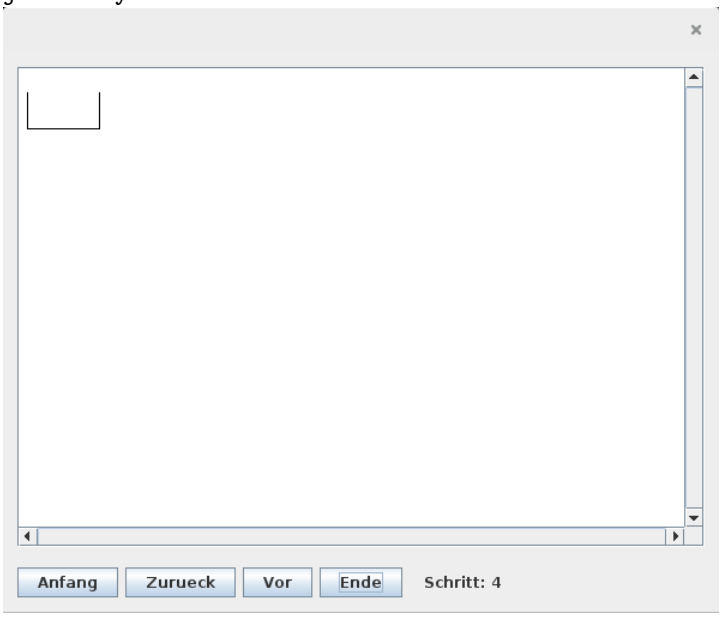
[1]

Sie dürfen in dieser Aufgabe keine Schleifen verwenden. Die Verwendung von Rekursion ist hingegen erlaubt. Implementieren Sie die statische Methode `levyCKurve` in der Klasse `LevyC`, welche folgende Parameter erhält:

- eine Referenz `c` auf ein `Canvas` Objekt
- eine `int`-Zahl, welche die gewünschte Ordnung der Kurve angibt.
- eine `int`-Zahl, welche die Länge einer Lévy–C-Kurve 0. Ordnung angibt.

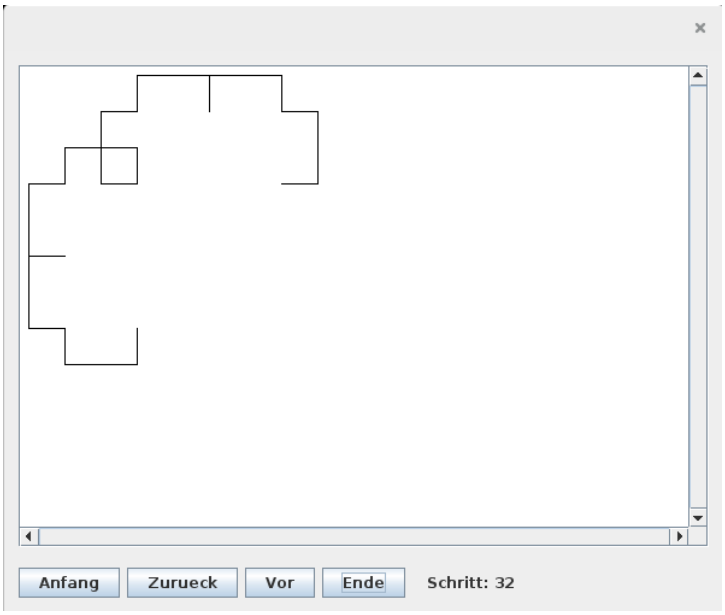
Diese Methode soll eine Lévy–C-Kurve der spezifizierten Ordnung zeichnen.
Zum Testen Ihrer Implementierung enthält die Klasse `LevyC` schon eine `main`-Methode. Das Programm bekommt bis zu zwei Parameter. Der erste gibt die Ordnung der Kurve an, der zweite die Länge der Kurven 0. Ordnung aus denen sie zusammengesetzt werden soll. Aus der `main`-Methode wird die Methode `levyCKurve` entsprechend aufgerufen. Sie können ihre Implementierung mit folgenden Aufrufen testen (darunter finden Sie Abbildungen, die Sie als Ergebnis zu diesen Aufrufen erhalten sollten):

- `java LevyC 2 30`

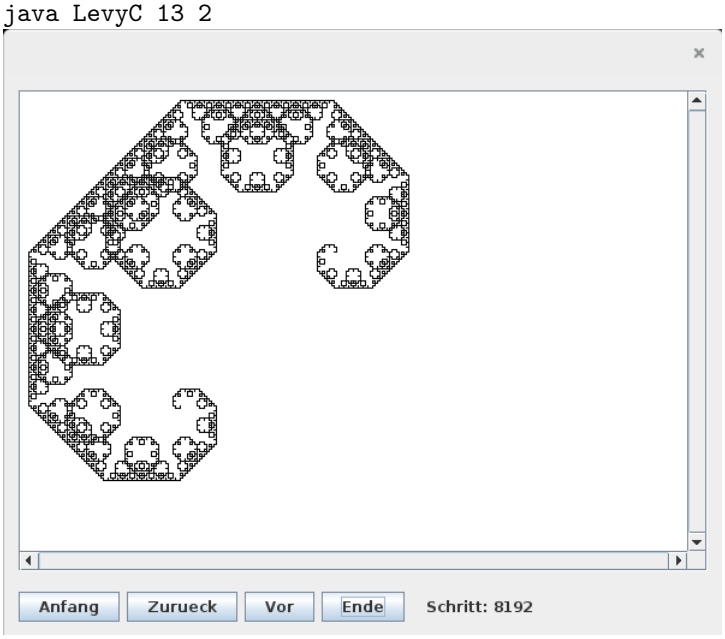


- `java LevyC 5 30`

[1] Bild lizenziert unter CC BY-SA 3.0, Autor: Gandalf61 at English Wikipedia, Quelle: `https://en.wikipedia.org/wiki/File:Levy_C_construction.png`



- `java LevyC 13 2`



Hinweise:

- Implementieren Sie die Methode `LevyC` rekursiv.
- Klicken Sie einmal auf die Schaltfläche Ende, um das Ergebnis anzuzeigen.
- Mit den Schaltflächen Vor und Zurueck können Sie die Zeichnung schrittweise auf- bzw. abbauen. Der Ablauf entspricht dabei dem Ablauf Ihres Programms. Die Schaltflächen Anfang und Ende springen zum Anfang bzw. Ende des Ablaufs.

Aufgabe 2 (Fraktale): (3 Punkte)

Auch in dieser Aufgabe soll eine fraktale Struktur mit Hilfe der Klasse `Canvas` gezeichnet werden. Diesmal geht es um Hilbert-Kurven. Eine solche Kurve 0. Ordnung ist die leere Kurve. Kurven n-ter Ordnung für $n > 0$

werden gebildet, indem man vier Hilbert-Kurven $(n-1)$-ter Ordnung kombiniert. Dabei unterscheidet man zwischen Links-Kurven und Rechts-Kurven.
Für $n > 0$ wird eine Hilbert-Links-Kurve n-ter Ordnung wie folgt konstruiert:

- Zunächst wird eine Drehung um $-90°$ durchgeführt.
- Dann wird eine Hilbert-Rechts-Kurve $(n-1)$-ter Ordnung gezeichnet.
- Anschließend eine gerade Linie
- Dann eine $90°$ Drehung
- Eine Hilbert-Links-Kurve $(n-1)$-ter Ordnung
- Eine gerade Linie
- Eine Hilbert-Links-Kurve $(n-1)$-ter Ordnung
- Eine $90°$ Drehung
- Wieder eine gerade Linie
- Eine Hilbert-Rechts-Kurve $(n-1)$-ter Ordnung
- Die nächste Kurve setzt dann im Winkel von $-90°$ an.

Für $n > 0$ werden Hilbert-Rechts-Kurven n-ter Ordnung ähnlich erzeugt:

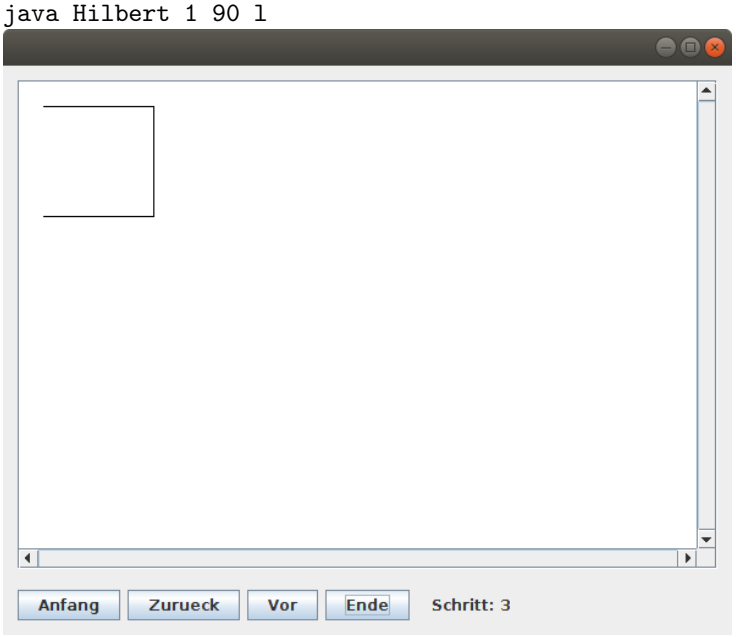
- Zunächst wird eine Drehung um $90°$ durchgeführt.
- Dann wird eine Hilbert-Links-Kurve $(n-1)$-ter Ordnung gezeichnet.
- Anschließend eine gerade Linie
- Dann eine $-90°$ Drehung
- Eine Hilbert-Rechts-Kurve $(n-1)$-ter Ordnung
- Eine gerade Linie
- Eine Hilbert-Rechts-Kurve $(n-1)$-ter Ordnung
- Eine $-90°$ Drehung
- Wieder eine gerade Linie
- Eine Hilbert-Links-Kurve $(n-1)$-ter Ordnung
- Die nächste Kurve setzt dann im Winkel von $90°$ an.

Positive Winkel bedeuten eine Drehung im Uhrzeigersinn, negative Winkel eine Drehung entgegen dem Uhrzeigersinn.
Wie in der vorigen Aufgabe dürfen Sie in dieser Aufgabe keine Schleifen verwenden. Die Verwendung von Rekursion ist hingegen erlaubt.
Implementieren Sie die statischen Methoden `hilbertLinks` und `hilbertRechts` in der Klasse `Hilbert`, welche folgende Parameter erhalten:

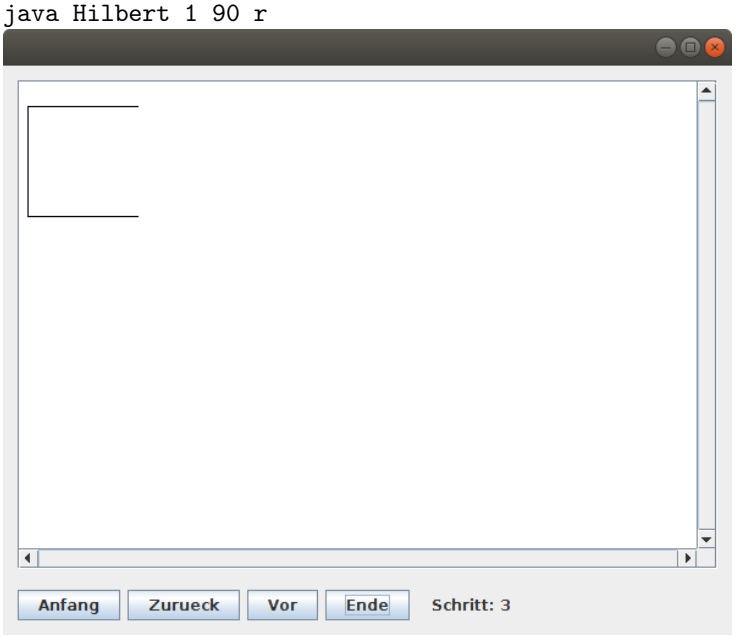
- eine Referenz `c` auf ein `Canvas` Objekt
- eine `int`-Zahl, welche die gewünschte Ordnung der Kurve angibt.
- eine `int`-Zahl, welche die Länge der geraden Linien angibt.

Diese Methoden sollen eine Hilbert-Kurve der spezifizierten Ordnung und Richtung zeichnen.
Zum Testen Ihrer Implementierung enthält die Klasse `Hilbert` schon eine `main`-Methode. Das Programm bekommt bis zu drei Parameter. Der erste gibt die Ordnung der Kurve an, der zweite die Länge der geraden Linien, die zu zeichnen sind. Der dritte Parameter gibt an, ob es eine Links-Kurve (`l`) oder eine Rechts-Kurve (`r`) sein soll. Aus der `main`-Methode wird die Methode `hilbertLinks` bzw. `hilbertRechts` entsprechend aufgerufen. Sie können Ihre Implementierung mit folgenden Aufrufen testen (darunter finden Sie Abbildungen, die Sie als Ergebnis zu diesen Aufrufen erhalten sollten):

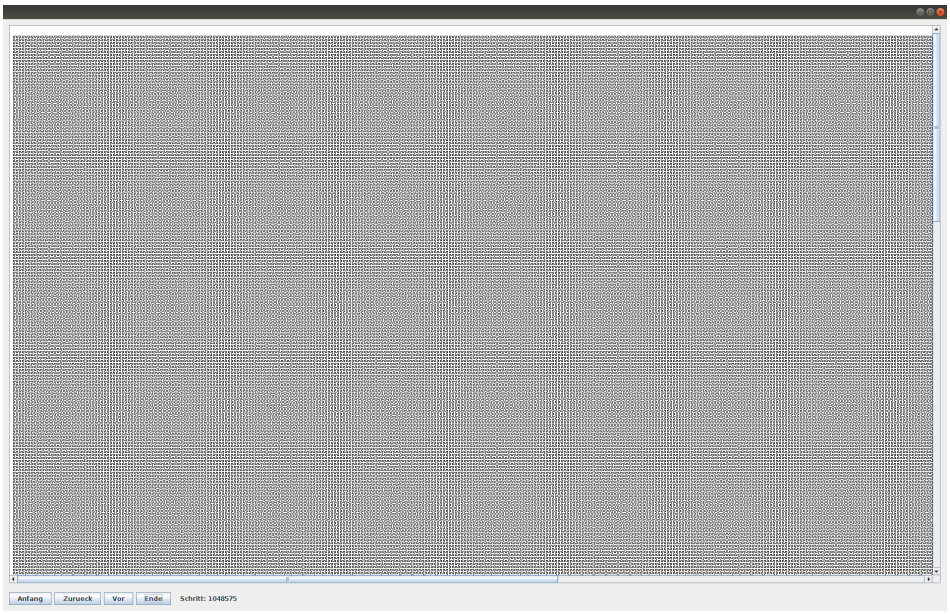
- `java Hilbert 1 90 l`



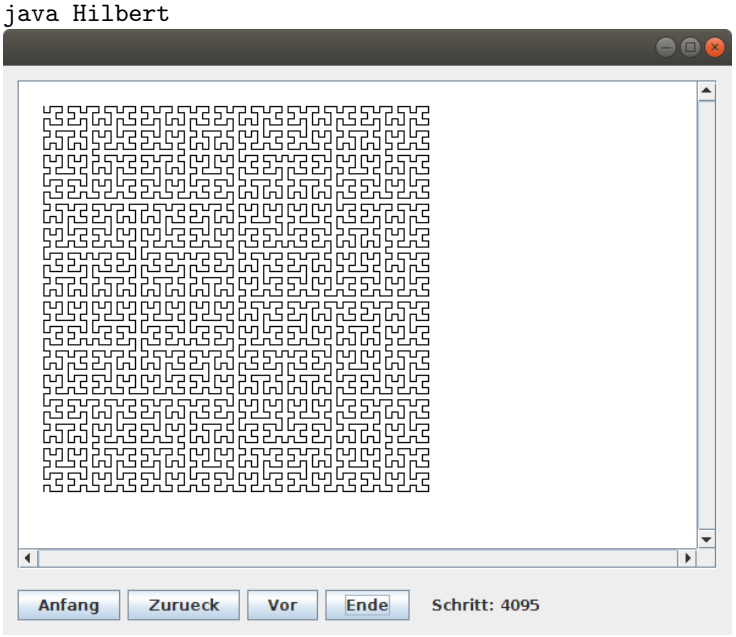
- `java Hilbert 1 90 r`



- `java Hilbert 10 3 l`



- `java Hilbert`



Hinweise:

- Klicken Sie einmal auf die Schaltfläche Ende, um das Ergebnis anzuzeigen.
- Mit den Schaltflächen Vor und Zurueck können Sie die Zeichnung schrittweise auf- bzw. abbauen. Der Ablauf entspricht dabei dem Ablauf Ihres Programms. Die Schaltflächen Anfang und Ende springen zum Anfang bzw. Ende des Ablaufs.

Tutoraufgabe 3 (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 zurückliefern. Die Methode `toString` soll eine textuelle Repräsentation der Liste zurückliefern, 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 zurückliefert, 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 zurückliefern, 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.

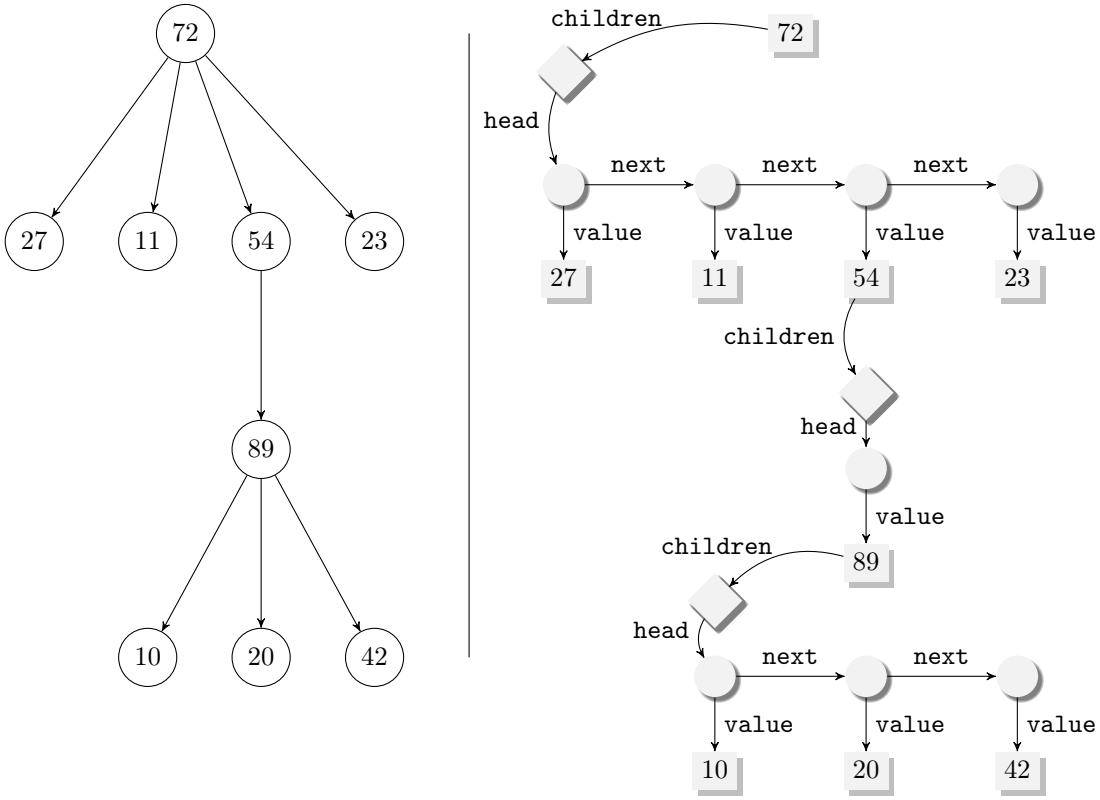
e) Vervollständigen Sie die Methode `mergesort` in der Klasse `ListExercise.java`. Diese Methode erhält eine unveränderliche Liste als Eingabe und soll eine Liste mit den gleichen Elementen in aufsteigender Reihenfolge zurückliefern. Falls die übergebene Liste weniger als zwei Elemente enthält, soll 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 4 (Rekursive Datenstrukturen): (2+4+6+6+6 = 24 Punkte)

Wir betrachten Vielwegbäume, in denen jeder Knoten beliebig viele Nachfolger haben kann. Ein solcher Baum ist in der folgenden Graphik links dargestellt:



Wir verwenden zur Repräsentation der Datenstruktur in Java die drei folgenden Klassen `TreeListElement`, `TreeList` und `Tree`, die Sie auf der Website finden:

```
public class TreeListElement{
  private Tree value;
  private TreeListElement next;

  public TreeListElement(Tree inputValue, TreeListElement inputNext){
    this.value = inputValue;
    this.next = inputNext;
  }
}
public class TreeList{
  private TreeListElement head;

  public TreeList(){
    this.head = null;
  }
}

public class Tree{
  private int label;
  private TreeList children;

  public Tree(int inputLabel, TreeList inputChildren){
    this.label = inputLabel;
    this.children = inputChildren;
  }
}
```

Eine Darstellung des Beispielbaums in unserer Datenstruktur ist in der Graphik rechts zu sehen. Dort sind Objekte der Klasse `TreeListElement` als (leere) Kreise, Objekte der Klasse `TreeList` als Rauten und Objekte der Klasse `Tree` als Quadrate dargestellt, die den gespeicherten `int`-Wert enthalten. Der dargestellte Baum hat also als Wurzel ein `Tree`-Objekt, dessen `label`-Attribut `72` ist und dessen `children`-Attribut auf eine vierelementige Liste zeigt. Die Elemente dieser Liste sind jeweils durch das Attribut `next` verbunden, während ihre `value`-Attribute jeweils auf `Tree`-Objekte zeigen. Der leere Baum wird durch `null` dargestellt. Die leere Liste wird durch das Element dargestellt, in welchem der `head`–Wert `null` ist. Attribute, deren Wert `null` ist, werden in der Graphik nicht gezeigt.
Sie können in jeder der folgenden Teilaufgaben beliebige Hilfsmethoden zur Klasse `Tree`, aber auch zu den Klassen `TreeList` und `TreeListElement` hinzufügen. Beachten Sie aber, dass diese Hilfsmethoden auch den Einschränkungen der Aufgabenstellung genügen müssen, da sonst keine Punkte vergeben werden. Insbesondere dürfen Sie beliebig viele eigene Konstruktoren implementieren. Sie dürfen jedoch keine vordefinierten Methoden benutzen.
Gehen Sie davon aus, dass alle dynamischen Objekte zykelfrei sind, d.h. dass Referenzen wie z.B. `next` oder `children` niemals auf Objekte zeigen, die bereits vorher in dem Baum oder der Liste vorkommen. Beachten Sie aber in Ihrer Implementierung, dass gewisse Attribute den Wert `null` haben können. Behandeln Sie diese Fälle so, dass es nicht zu einem Programmabsturz kommt. Benutzen Sie in dieser Aufgabe **keine** Schleifen sondern **ausschließlich Rekursion**. Versehen Sie in Ihrer Implementierung alle öffentlichen Methoden außer Konstruktoren und Selektoren mit `javadoc`–Kommentaren, um die Lesbarkeit zu erhöhen.
In der Klasse `Tree` gibt es eine `main`–Methode die Sie benutzen können, um Ihre Implementierung auszuprobieren. Sie können diese Methode beliebig verändern.

a) Schreiben Sie für alle Klassen geeignete get– und set–Methoden.

b) Schreiben Sie in der Klasse `Tree` eine Methode `public String toString()`. Der Ausgabestring soll die Form `"label->children"` haben, d.h. der gespeicherte Wert wird von einem Pfeil gefolgt ausgedruckt. Anschließend werden die Kinder des Knotens als Liste in eckigen Klammern durch Kommata getrennt aufgeführt. Rufen wir die Methode `toString()` auf dem oben abgebildeten Beispielbaum auf, so erhalten wir als Rückgabe:

```
72->[27->[],11->[],54->[89->[10->[],20->[],42->[]]],23->[]]
```

Hinweise:

- Falls ein `TreeListElement` das `value`–Attribut `null` hat, so wird dieser `null`–Wert bei der Ausgabe einfach ignoriert und nur die weiteren Werte im `next`–Attribut beatrachtet.

c) Schreiben Sie eine Methode `public int branchingDegree()`, die den Verzweigungsgrad eines Baumes bestimmt. Der Verzweigungsgrad ist dabei wie folgt definiert. Ein Baum ohne Kinder hat den Verzweigungsgrad 0. Ein Baum mit Kindern hat als Verzweigungsgrad das Maximum der Anzahl der Kinder und der Verzweigungsgrade der Kinder. Wird die Methode `branchingDegree()` auf dem oben abgebildeten Beispielbaum aufgerufen, liefert sie die Rückgabe `4`. Ruft man hingegen diese Methode auf dem Teilbaum auf, in dessen Wurzel der Wert `54` gespeichert ist, so ergibt sich der Wert `3`.

d) Schreiben Sie eine Methode `public boolean contains(int toSearch)`, die genau dann `true` zurückgibt, wenn `toSearch` als `label`–Attribut im entsprechenden Objekt der Klasse `Tree` vorkommt. Die Methode `contains` mit dem Argument `20` liefert auf dem oben abgebildeten Beispielbaum den Rückgabewert `true`, mit dem Argument `-1` aufgerufen liefert sie `false`.

e) Wie Sie sicher festgestellt haben, ist es sehr umständlich, beliebige Objekte der Klasse `Tree` mit Hilfe der Konstruktoren zu erzeugen. Um dies zu vereinfachen, soll in dieser Aufgabe eine Methode `public static Tree buildTree(int label, Tree... children)` geschrieben werden. Diese soll ein Objekt der Klasse `Tree` zurückgeben, dessen `label`–Attribut der Eingabewert `label` ist und dessen `children`–Attribut ein Objekt der Klasse `TreeList` ist, welches die im Eingabeparameter `children` spezifizierten Objekte der Klasse `Tree` in der eingegebenen Reihenfolge enthält. Möchte man also z.B. den Beispielbaum erzeugen, wird die Funktion folgendermaßen aufgerufen:

```
buildTree(72, buildTree(27), buildTree(11), buildTree(54,
  buildTree(89,buildTree(10),buildTree(20),buildTree(42))),
    buildTree(23))
```

Tutoraufgabe 5 (Klassenhierachie):

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.

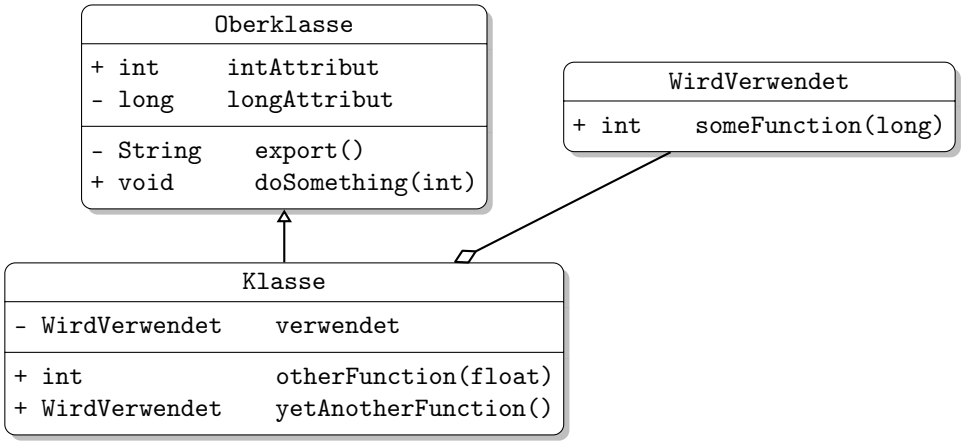


Abbildung 1: Graphische Notation zur Darstellung von Klassen.

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 6 (Klassenhierachie): (6 Punkte)

In dieser Aufgabe sollen Sie einen Teil der Möbelwelt modellieren.

- Ein Möbelstück hat ein Gewicht und ein bestimmtes Material. In dieser Aufgabe betrachten wir Sitzmöbel, Tische und Behältnismöbel.
- Ein Tisch hat eine Anzahl von Tischbeinen und eine bestimmte Länge und Breite.
- Ein Esstisch bietet einer gewissen Anzahl von Personen Platz. Er kann aber auch verlängert werden.
- Ein Sitzmöbelstück hat eine gewissen Anzahl von Sitzplätzen.
- Ein Stuhl ist ein Sitzmöbelstück. Stühle können an einen Tisch gestellt werden. Dies funktioniert nicht immer.
- Ein Schaukelstuhl ist ein besonderer Stuhl. Er kann schaukeln.
- Ein Behältnismöbelstück hat ein Volumen und kann gefüllt werden.
- Ein Schrank ist ein Behältnismöbelstück mit einer bestimmten Anzahl von Fächern. Einen Schrank kann man öffnen.

- Kleiderschränke können über einen Spiegel verfügen oder nicht. Ein Kleiderschrank kann aufgeräumt werden.
- Ein Bücherregal ist ein Behältnismöbelstück, welches natürlich Bücher enthält. Ein Buch kann entnommen werden, wenn es im Regal steht.
- Ein Buch hat einen Titel und eine Anzahl von Seiten. Bücher können gelesen werden.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Gegenständen. 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 Verlängern eines Esstischs, das Stellen eines Stuhls an einen Tisch, das Schaukeln eines Schaukelstuhls, das Füllen eines Behältnismöbelstücks, das Öffnen eines Schranks, das Aufräumen eines Kleiderschranks sowie das Entnehmen eines Buchs aus einem Regal zu modellieren.
Verwenden Sie hierbei die Notation aus der entsprechenden Tutoriumsaufgabe.

Aufgabe 7 (Codescape): (Codescape)

Lösen Sie die Räume von **Deck 4** des Spiels Codescape.
Ihre Lösung für Räume dieses Codescape Decks wird nur dann für die Zulassung gezählt, wenn Sie die Lösung bis Freitag, den 30.11.2018 um 12:00 abschicken.