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

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 **Montag, den 2.12.2019, um 12:00 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 abgeben.
- In Aufgabe 2 müssen Sie in Java programmieren und **eine** `.java`-Datei vervollständigen. **Drucken** Sie diese aus **und** laden Sie sie fristgerecht im `RWTHmoodle`-Lernraum "Programmierung (Übung - Tutorium)" hoch. Aufgabe 2 ist ausdrücklich **keine** VPL-Aufgabe!

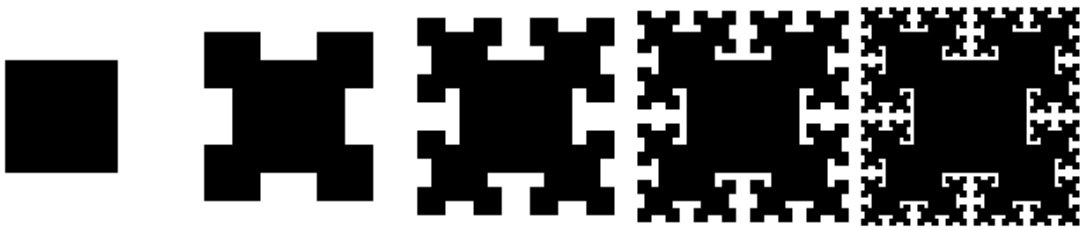
  Stellen Sie sicher, dass Ihr Programm mit `javac` **fehlerfrei kompiliert**, ansonsten werden keine Punkte vergeben.
- In Aufgabe 4 müssen Sie in Java programmieren und `.java`-Dateien anlegen. **Drucken** Sie diese aus **und** laden Sie sie fristgerecht im `RWTHmoodle`-Lernraum "Programmierung (Übung - Tutorium)" in der entsprechenden VPL-Aufgabe hoch. Sie können auch direkt den VPL-Editor benutzen, um die Aufgabe zu lösen.

  Stellen Sie sicher, dass Ihr Programm in **VPL fehlerfrei kompiliert**, ansonsten werden keine Punkte vergeben.

  Es ist ausreichend, wenn eine Person Ihrer Abgabegruppe den Programmcode in VPL hochlädt. Schreiben Sie alle Matrikelnummern der Abgabegruppe als Kommentar in Ihren Programmcode. Schreiben Sie außerdem auf Ihre schriftliche Abgabe die Matrikelnummer derjenigen Person, deren Programmcode in VPL bewertet werden soll.
- Einige Hausaufgaben müssen im Spiel Codescape gelöst werden. Klicken Sie dazu im `RWTHmoodle`-Lernraum "Programmierung (Übung - Tutorium)" rechts im Block "Codescape" auf den angegebenen Link. Diese Aufgaben werden getrennt von den anderen Hausaufgaben gewertet.

Tutoraufgabe 1 (Fraktale):

In dieser Aufgabe geht es darum, eine fraktale Struktur grafisch darzustellen. Die Struktur besteht aus Quadraten, die nach einem bestimmten Vorgehen angeordnet werden. Das Vorgehen lässt sich rekursiv beschreiben: Im ersten Level ($n = 1$) besteht die Struktur aus nur einem Quadrat. Für $n \geq 2$ gilt: Die Struktur des n-ten Levels entsteht dadurch, dass erst ein Quadrat und danach auf dessen vier Ecken jeweils eine verkleinerte Variante des $(n-1)$-ten Levels der Struktur gezeichnet wird. Hierbei haben die größten Quadrate des $(n-1)$-ten Levels die halbe Kantenlänge des größten Quadrates des n-ten Levels. In folgendem Bild sind die ersten fünf Level der fraktalen Struktur dargestellt.



Verwenden Sie zur graphischen Darstellung die Klasse `Canvas`, welche in der Datei `Canvas.java` definiert ist. Sobald ein Objekt der Klasse `Canvas` erzeugt wird, wird dieses in einem Fenster auf dem Bildschirm angezeigt. Die Klasse `Canvas` stellt (unter anderem) die Methoden `move` und `square` zur Verfügung, die ein

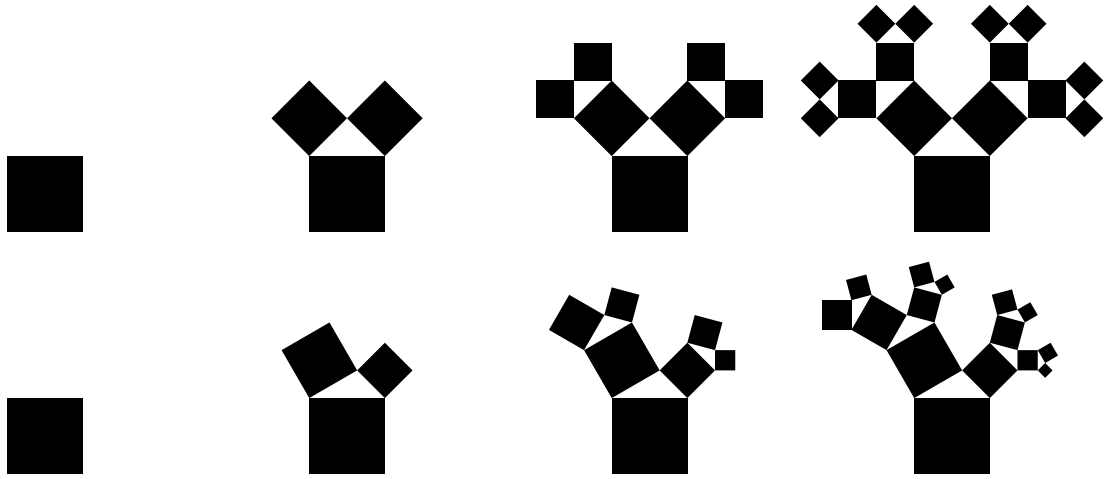
bzw. zwei `double`-Zahlen als Parameter bekommen. Mit `move` verändern Sie die Zeichenposition, mit `square` nehmen Sie eine Zeichenoperation im angezeigten Fenster vor. Sie können die einzelnen Zeichenschritte über die Schaltflächen Vor und Zurueck nacheinander ansehen. Die Schaltflächen `Anfang` und `Ende` zeigen die leere Fläche bzw. das fertige Bild.
Sei `c` ein Objekt der Klasse `Canvas`. Dann bewegt beispielsweise der Aufruf `c.move(3, 4)` die aktuelle Zeichenposition (d. h. die Position, an der als nächstes gezeichnet wird) um 3 Einheiten nach rechts und 4 Einheiten nach unten. Ein Aufruf `c.square(5)` zeichnet ein gefülltes Quadrat mit einer Seitenlänge von 5 Einheiten, wobei der Mittelpunkt des Quadrats auf der aktuellen Zeichenposition liegt.
Implementieren Sie die statische Methode `paintFractal` in der Klasse `Squares`, welche eine Referenz `c` auf ein Objekt der Klasse `Canvas`, eine `int`-Zahl `level` und eine `double`-Zahl `length` übergeben bekommt und mit Hilfe des `Canvas` Objekts `c` die beschriebene fraktale Struktur des `level`-ten Level zeichnet, wobei die Kantenlänge des größten Quadrats `length` beträgt.
Die `main`-Methode der Klasse `Squares` ist bereits gegeben, sie nimmt die beiden Parameter `level` und `length` entgegen und prüft, ob beide Parameter ≥ 1 sind. Außerdem ruft die `main`-Methode die von Ihnen zu implementierende Methode `paintFractal` auf.
Sie dürfen in dieser Aufgabe *keine* Schleifen verwenden. Die Verwendung von Rekursion ist hingegen erlaubt.
Dokumentieren Sie die Methode `paintFractal`, indem Sie die Implementierung wie auf Blatt 5 mit Javadoc-Kommentaren ergänzen, auch für jeden Parameter. Stellen Sie sicher, dass `javadoc` kompiliert.

Der folgende Aufruf eignet als Test, als Ausgabe ist das Fraktal ganz rechts im obenstehenden Bild zu erwarten.

```
java Squares 5 100
```

Aufgabe 2 (Fraktale): (16 Punkte)

Auch in dieser Aufgabe soll eine fraktale Struktur mithilfe der Klasse `Canvas` gezeichnet werden. Diesmal geht es um sogenannte Pythagoras Bäume. Ein Pythagoras Baum im ersten Level ($n = 1$) ist wiederum nur ein Quadrat einer vorgegebenen Basislänge. Für $n \geq 2$ gilt: Ein Pythagoras Baum des n-ten Levels besteht aus einem Quadrat und zwei Pythagoras Unterbäumen des $(n-1)$-ten Levels. Die jeweils ersten Quadrate dieser beiden Unterbäume werden oberhalb des Quadrates im n-ten Level nebeneinander positioniert. Dabei wird das linke Quadrat um einen Winkel α gegen den Uhrzeigersinn und das rechte Quadrat um einen Winkel β im Uhrzeigersinn rotiert, sodass die drei Quadrate zwischen sich ein leeres Dreieck formen. Beide Winkel müssen positiv und kleiner als 90° sein und dürfen in Summe nicht mehr als 120° ergeben. Basierend auf der Länge des Quadrates im n-ten Level und diesen beiden Winkeln müssen die Längen der beiden Quadrate in den beiden Unterbäumen berechnet werden. Die folgende Grafik zeigt die ersten vier Level eines Pythagoras Baums mit den Winkeln $\alpha = 45°$ und $\beta = 45°$ (oben) sowie $\alpha = 30°$ und $\beta = 45°$ (unten).



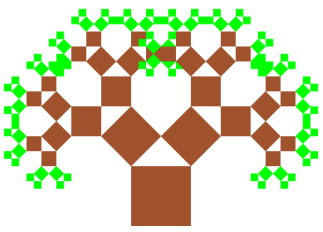
Zur Berechnung der jeweils nächsten Quadratlängen eignet sich der Sinussatz. Sei ℓ die Länge des unteren Quadrats. Dann folgt aus dem Sinussatz, dass die Länge des linken Quadrats genau $\frac{\sin(\beta)\cdot\ell}{\sin(180°-\alpha-\beta)}$ und die Länge des rechten Quadrats genau $\frac{\sin(\alpha)\cdot\ell}{\sin(180°-\alpha-\beta)}$ ist. Um den Sinus eines Winkels `angle` (angegeben in Grad) in Java zu berechnen, können Sie den Ausdruck `Math.sin(Math.toRadians(angle))` verwenden. (Die Funktion `Math.toRadians(angle)` ist nötig, weil `Math.sin` einen Winkel im Bogenmaß erwartet und nicht als Gradzahl.)

Die Klasse `Canvas` bietet neben den bereits aus der vorigen Aufgabe bekannten Methoden `move` und `square` zusätzlich eine Methode `rotate` an, welche einen Winkel in Grad übergeben bekommt. Diese Methode rotiert die Ausrichtung für alle weiteren Zeichenoperationen um den übergebenen Winkel im Uhrzeigersinn (bei negativen Winkeln entsprechend gegen den Uhrzeigersinn).
Sei `c` ein Objekt der Klasse `Canvas`. Dann zeichnen beispielsweise die aufeinander folgenden Aufrufe `c.rotate(45)` und `c.square(5)` ein um 45° rotiertes Quadrat mit Seitenlänge 5 mit der aktuellen Position als Mittelpunkt. Analog wird die aktuelle Position durch die aufeinander folgenden Aufrufe `c.rotate(90)` und `c.move(1,0)` um eine Einheit nach unten statt nach rechts verschoben. Durch `rotate` wird also das gesamte Koordinatensystem rotiert.
Außerdem bietet die Klasse `Canvas` zwei Farben `BROWN` und `GREEN` als statische Attribute an, welche mittels der Methode `chooseColor` ausgewählt werden können. Der Aufruf `c.chooseColor(Canvas.BROWN)` führt also im obigen Beispiel dazu, dass alle weiteren Zeichenoperationen mit brauner Farbe durchgeführt werden. Diese Farben sollen dazu genutzt werden, große Quadrate braun und kleine Quadrate grün zu färben, damit die grafische Qualität der Pythagoras Bäume erhöht wird.
Implementieren Sie die statische Methode `paintPythagorasTree` in der Klasse `Pythagoras`, welche folgende Parameter erhält:

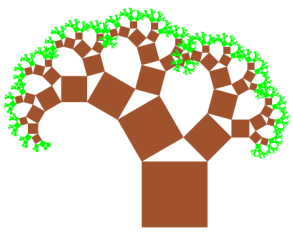
- eine Referenz `c` auf ein `Canvas` Objekt
- eine `int`-Zahl `level`, welche das gewünschte Level des Pythagoras Baums angibt
- einen `double`-Wert `length`, welcher die Länge des ersten Quadrats des Pythagoras Baums angibt
- eine `int`-Zahl `leftAngle`, welche dem Winkel α entspricht
- eine `int`-Zahl `rightAngle`, welche dem Winkel β entspricht
- eine `int`-Zahl `switchLength`, welche die maximale Länge von Quadraten angibt, welche grün gezeichnet werden sollen

Diese Methode soll einen Pythagoras Baum des spezifizierten Levels zeichnen, wobei das erste Quadrat die übergebene Länge hat und die beiden jeweils folgenden Quadrate wie oben beschrieben um die Winkel α und β rotiert werden. Außerdem sollen Quadrate mit einer Seitenlänge größer als `switchLength` in braun und alle anderen Quadrate in grün gezeichnet werden.
Die `main`-Methode der Klasse `Pythagoras` ist bereits gegeben. Sie nimmt die oben aufgelisteten fünf Parameter entgegen und prüft, ob die entsprechenden Anforderungen erfüllt sind. Werden weniger als fünf Parameter übergeben, dann werden für die fehlenden Parameter default-Werte gewählt. Im Anschluss ruft die `main`-Methode die von Ihnen zu implementierende Methode `paintPythagorasTree` auf.
Sie dürfen in dieser Aufgabe *keine* Schleifen verwenden. Die Verwendung von Rekursion ist hingegen erlaubt.

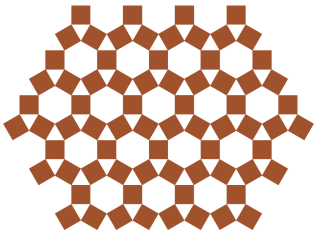
Dokumentieren Sie die Methode `paintPythagorasTree`, indem Sie die Implementierung wie auf Blatt 5 mit Javadoc-Kommentaren ergänzen, auch für jeden Parameter. Stellen Sie sicher, dass `javadoc` kompiliert.
Zum Testen Ihrer Implementierung eignen sich die folgenden Aufrufe Ihres Programms (darüber finden Sie Abbildungen, die Sie als Ergebnis zu diesen Aufrufen erhalten sollten):



`java Pythagoras 7 100 45 45 20`



`java Pythagoras 10 100 30 45 10`



`java Pythagoras 7 50 60 60`

Beachten Sie die Abgabemodalitäten in den Allgemeinen Hinweisen auf dem Titelblatt.

Tutoraufgabe 3 (Boolesche Binärbaume):

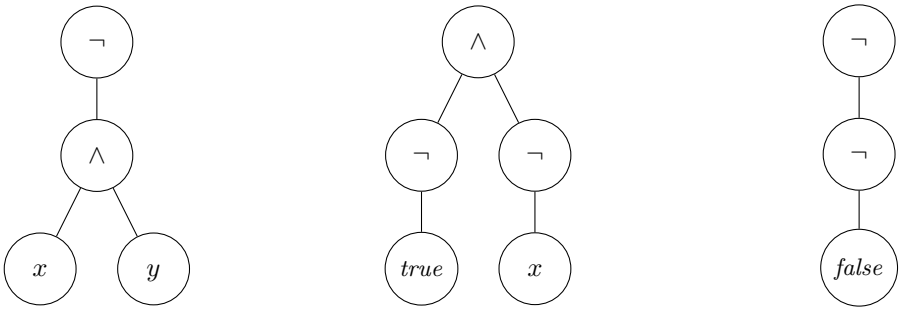
In dieser Aufgabe sollen eine Datenstruktur für boolesche Binärbäume sowie einige rekursive Algorithmen darauf implemetiert werden.

Ein boolescher Binärbaum repräsentiert in dieser Aufgabe eine boolesche Formel, die aus Variablen, den konstanten Wahrheitswerten `true` und `false`, dem binären Konjunktionsoperator $\wedge$ und dem unären Negationsoperator $\neg$ aufgebaut ist. Der Disjunktionsoperator $\vee$ soll durch die anzulegende Datenstruktur explizit *nicht* modelliert werden. Dies stellt aber keine echte Einschränkung dar, da alle booleschen Funktionen bereits über der Operatorenmenge $\{\wedge, \neg\}$ ausgedrückt werden können; diese ist *funktional vollständig*.

Jeder Knoten kann null, ein oder zwei Kinder haben. Diese Kinder sind wiederum boolesche Binärbäume. Außerdem kann er mit einer Variable vom Typ `String` markiert sein. Was ein bestimmter Baumknoten repräsentiert, wird implizit über die Anzahl der Kinder ausgedrückt, die nicht den Wert `null` haben:

- Hat ein Knoten keine solchen Kinder, so steht er für seine Variable. Die beiden Kinder sind dann `null`. Die Variable darf nicht der leere String `""` sein.
- Hat ein Knoten ein nicht-`null`-Kind, so steht er für die Negation der Formel, die durch sein Kind repräsentiert wird. Hierbei soll durch Ihre Implementierung sichergestellt werden, dass ein Negationsknoten stets sein erstes Kind nutzt. Das erste Kind ist dann nicht `null`, das zweite Kind ist `null` und die Variable soll auf den leeren String `""` gesetzt sein.
- Hat ein Knoten zwei nicht-`null`-Kinder, so repräsentiert er die Konjunktion der beiden Formeln, die durch seine beiden Kinder repräsentiert werden. Wiederum soll die Variable der leere String `""` sein.
- Es gibt weiterhin Knoten mit den konstanten Wahrheitswerten `true` und `false`. Diese gelten als Variablenknoten, tragen als Attribut Variable aber `"true"` bzw. `"false"`. Das heißt auch, dass echte Variablen diese beiden Strings als Namen nicht annehmen dürfen.

In Abbildung 2 finden Sie zwei Beispiele für solche booleschen Binärbäume.



(a) $\neg(x \wedge y)$ (b) $\neg true \wedge \neg x$ (c) $\neg\neg false$

Abbildung 2: Beispiele für boolesche Binärbäume mit den jeweils repräsentierten Formeln

Ihre Implementierung sollte mindestens die folgenden Methoden beinhalten. Sie sollten dabei die Konzepte der Datenkapselung berücksichtigen. Hilfsmethoden müssen als `private` deklariert werden. In dieser Aufgabe dürfen Sie die in der Klasse `Utils` zur Verfügung gestellten Hilfsfunktionen, aber keine Bibliotheksfunktionen verwenden. Sie finden die Klasse im Lernraum.

In dieser Aufgabe dürfen Sie *keine* Schleifen verwenden. Die Verwendung von Rekursion ist hingegen erlaubt.

a) Erstellen Sie eine Java-Klasse `BoolTreeNode` mit den Attributen `variable` (vom Typ `String`), `child1` und `child2` (vom Typ `BoolTreeNode`).

b) Erstellen sie die folgenden drei Konstruktoren, um Objekte vom Typ `BoolTreeNode` zu erzeugen.

```
BoolTreeNode(String variableInput)
BoolTreeNode(BoolTreeNode negated)
BoolTreeNode(BoolTreeNode conjunct1, BoolTreeNode conjunct2)
```

Alle drei Konstruktoren sollen für den Benutzer der Klasse nicht sichtbar sein. Setzen Sie die Attribute wie in der Einleitung beschrieben. Sie müssen sich nicht um die Zulässigkeit der Eingaben kümmern. Dies wird in der nächsten Teilaufgabe behandelt.

c) Jetzt kommen wir zu den Methoden, mit denen der Benutzer der Klasse neue Knoten erzeugen kann:

```
BoolTreeNode boolTreeTrueNode()
BoolTreeNode boolTreeFalseNode()
BoolTreeNode boolTreeVariableNode(String variableInput)
BoolTreeNode boolTreeNotNode(BoolTreeNode negated)
BoolTreeNode boolTreeAndNode(BoolTreeNode conjunct1,
                             BoolTreeNode conjunct2)
```

Greifen Sie dafür auf die soeben erstellten Konstruktoren zurück. Sie müssen jetzt auch überprüfen, ob die übergebenen Parameter den Anforderungen entsprechen. Kann ein Knoten wegen einer Anforderungsverletzung nicht erstellt werden, so ist eine aussagekräftige Fehlermeldung auszugeben und `null` zurückzugeben. Nutzen Sie für Fehlermeldungen die statische Methode `error` aus der Klasse `Utils`. Entscheiden Sie begründet, ob Sie diese Methoden statisch implementieren. Setzen Sie jeweils auch einen sinnvollen Zugriffsmodifikator (z.B. `public` oder `private`) .

d) Erstellen Sie eine Methode, um die Tiefe eines `BoolTreeNode` zu bestimmen.

```
public int depth()
```

Ein Variablenknoten habe dabei Tiefe 0, ein Negationsknoten eine um eins höhere Tiefe als sein Kind, und ein Konjunktionsknoten habe eine um eins höhere Tiefe als das Tiefste der beiden Kinder. Setzen Sie diese rekursive Definition der Tiefe einer Formel in Ihrer Implementierung auch rekursiv um.

Hinweis: Sie finden in der Klasse `Utils` eine Methode `max`, um das Maximum von zwei `int`-Werten zu bestimmen.

e) Erstellen Sie folgende Methoden, um zu bestimmen, von welcher Art ein Knoten ist.

```
public boolean isLeaf()
public boolean isTrueLeaf()
public boolean isFalseLeaf()
public boolean isNegation()
public boolean isConjunction()
```

Dabei soll `isLeaf()` genau dann `true` zurückgeben, wenn der Knoten ein Variablenknoten ist (d.h. wenn beide Kinder des Knotens den Wert `null` haben). Die Methoden `isTrueLeaf()` und `isFalseLeaf()` sollen genau dann `true` zurückgeben, wenn der Knoten den konstanten Wahrheitswert `true` bzw. `false` repräsentiert. Durch `isNegation()` soll genau dann `true` zurückgegeben werden, wenn der Knoten genau ein nicht-`null`-Kind hat (und zwar das erste). Die Methode `isConjunction()` soll genau dann `true` zurückgeben, wenn der Knoten zwei nicht-`null`-Kinder hat.

f) In dieser Teilaufgabe soll die boolesche Formel, die durch den Baum repräsentiert wird, ausgewertet werden.

```
public boolean evaluate(String... trueVars)
```

Dabei soll der Baum unverändert bestehen bleiben. Die konstanten Wahrheitswerte, die Konjunktion und die Negation haben die übliche Semantik. Der Methode wird weiterhin eine Menge von Strings übergeben. Dies sind die Namen der Variablen, die zu `true` evaluiert werden. Alle anderen Variablen werden zu `false` evaluiert.

Hinweis: Sie finden die statische Methode `evaluateVariable(String elem, String... strings)` in der Klasse `Utils`. Diese führt die Evaluation einer Variable `elem` bei gegebenen (mit `true` belegten) Variablen `strings` durch. Hier werden auch die Fälle `"true"` und `"false"` behandelt.

g) Wenn eine Formel häufig ausgewertet wird, kann es sinnvoll sein, die Formel im Voraus zu vereinfachen, um die folgenden Auswertungen effizienter ausführen zu können. Die folgenden Teilaufgabe (wie auch die späteren Aufgaben 4(c)-(g)) beschäftigt sich mit verschiedenen Möglichkeiten der Vereinfachung einer booleschen Formel.

Hinweise:

- In all diesen Teilaufgaben soll die Vereinfachung nicht nur auf dem Wurzelknoten durchgeführt werden, sondern auf dem gesamten Baum.
- Alle dieser Vereinfachungsmethoden geben einen Wert vom Typ `boolean` zurück. Dieser soll angeben, ob irgendwo im Baum eine solche Vereinfachung stattgefunden hat. Wir werden später (in Aufgabe 4(h)) Gebrauch von diesem Rückgabewert machen.
- Wenn Sie im Zuge einer Vereinfachung einen Knoten entfernen, dann darf dieser entfernte Knoten nachher nicht mehr als Kind eines anderen Knotens auftauchen. Machen Sie sich Gedanken, auf welche Werte diese Einträge in den einzelnen Fällen gesetzt werden sollen.
- Wenn in einer Teilaufgabe eine bestimmte Vereinfachungsregel angegeben ist, die auf beiden Seiten einer Konjunktion arbeitet, dann soll die Implemtierung symmetrisch erfolgen. Das heißt, dass es keine Rolle spielen soll, wenn das erste und das zweite Kind vertauscht werden.
- Benutzen Sie `isLeaf()`, `isTrueLeaf()`, `isFalseLeaf()`, `isNegation()` und `isConjunction()`.
- x steht im Folgenden für eine Variable, die durch einen Variablenknoten (ohne Kinder) repräsentiert wird. φ steht für eine boolesche Formel, die durch einen beliebigen Knoten repräsentiert werden kann.

Es gilt für jede Formel φ, dass $\neg\neg\varphi$ äquivalent zu φ ist. Erstellen Sie eine Methode zur Entfernung von doppelten Negationen.

```
public boolean removeDoubleNegations()
```

Nach Anwendung der Funktion sollen keine Knoten mehr im Baum vorkommen, die Formeln der Form $\neg\neg\varphi$ repräsentieren.

h) Dokumentieren Sie alle Methoden, die als `public` markiert sind, indem Sie die Implementierung mit Javadoc-Kommentaren ergänzen. Diese Kommentare sollten eine allgemeine Erklärung der Methode sowie weitere Erklärungen jedes Parameters und des `return`-Wertes enthalten. Verwenden Sie innerhalb des Kommentars dafür die Javadoc-Anweisungen `@param` und `@return`.

Benutzen Sie das Programm `javadoc`, um Ihre Javadoc-Kommentare in das HTML-Format zu übersetzen. Überprüfen Sie mit einem Browser, ob das gewünschte Ergebnis generiert wurde.

Aufgabe 4 (Boolesche Binärbaume): (2 + 2 + 3 + 3 + 3 + 4 + 2 + 2 + 2 = 23 Punkte)

In dieser Aufgabe erweitern wir die Datenstruktur der booleschen Binärbäume aus Aufgabe 3. Es gelten alle Erläuterungen und Hinweise, die dort zu finden sind. Außerdem können Sie alle Methoden aus Aufgabe 3 benutzen. (Diese werden Ihnen im Lernraum zur Verfügung gestellt, nachdem alle regulären Tutorien gehalten worden sind.) Erstellen Sie in jeder Teilaufgabe die dort angegebene Methode.

Die Lösung dieser Aufgabe muss in VPL hochgeladen werden. Beachten Sie dazu die entsprechenden allgemeinen Hinweise auf dem Titelblatt. Der VPL-Editor bietet unter *Ausführen* auch einige einfache Testfälle zum Testen der grundlegenden Funktionaliät Ihrer Implementierung.
Die Testfälle decken aber nicht alle möglichen Fälle ab. Um Ihre Implementierung selbst zu testen, können sie in der Klasse `BoolTreeNode` eine `main`-Methode schreiben und diese unter *Debuggen* mit `run BoolTreeNode` ausführen.

a) Wie erwähnt unterstützt unsere Datenstruktur Disjunktionen der Form $\varphi_1 \vee \varphi_2$ nicht direkt. Wir möchten trotzdem eine Methode zur Verfügung stellen, die einen Knoten erstellt, der $\varphi_1 \vee \varphi_2$ repräsentiert.

```
BoolTreeNode BoolTreeOrNode(BoolTreeNode disjunct1,
                            BoolTreeNode disjunct2)
```

Auch hier müssen Sie wie in Aufgabe 3(c) prüfen, ob die Eingaben den Anforderungen genügen. Setzen Sie außerdem einen sinnvollen Zugriffsmodifikator und entscheiden Sie, ob die Methode statisch sein soll. Vermeken Sie Ihre Begründungen jeweils als Kommentar.

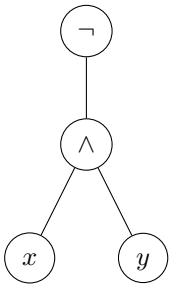
Hinweis: Überlegen Sie, wie Sie mit Hilfe der vorhandenen Elemente Disjunktionen modellieren können.

b) Zu jedem Objekt vom Typ `BoolTreeNode` soll eine String-Repräsentation berechnet werden.

```
public String toString()
```

Dabei soll der zurückgegebene String wie folgt aufgebaut werden: Ein Variablenknoten wird durch seine Variable repräsentiert, ein Negationsknoten durch `not(...)` und ein Konjunktionsknoten in Infixnotation durch `(... and ...)`, wobei jeweils die Repräsentationen der/des Kindknoten einzusetzen sind. Setzen Sie diese rekursive Definition in Ihrer Implementierung auch rekursiv um.

Ein Beispiel für die Anwendung der `toString`-Methode finden Sie hier:



⇝ `"not((x and y))"`

Im Folgenden geht es um weitere Vereinfachungsmethoden wie `removeDoubleNegations` aus Aufgabe 3(g). Beachten Sie insbesondere die dortigen Hinweise.

c) Es gilt, dass $\neg \mathit{true}$ äquivalent zu *false* ist und dass $\neg \mathit{false}$ äquivalent zu *true* ist. Erstellen Sie eine Methode, die alle Vorkommen von $\neg \mathit{true}$ durch *false* ersetzt und alle Vorkommen von $\neg \mathit{false}$ durch *true*.

```
public boolean removeAtomicNegations()
```

d) Es gilt, dass $x \wedge x$ äquivalent zu x ist. Erstellen Sie eine Methode, die alle Idempotenzen $x \wedge x$ durch x ersetzt, wobei x eine Variable ist, die durch einen Variablenknoten repräsentiert wird. Andere Idempotenzen der Form $\varphi \wedge \varphi$ dürfen weiterhin vorkommen.

```
public boolean removeIdempotency()
```

e) Es gilt, dass $x \wedge \neg x$ äquivalent zu *false* ist. Erstellen Sie eine Methode, die alle Teilformeln $x \wedge \neg x$ und $\neg x \wedge x$ im Baum durch *false* ersetzt, wobei x eine Variable ist, die durch einen Variablenknoten repräsentiert wird. Andere Widersprüche der Form $\varphi \wedge \neg\varphi$ und $\neg\varphi \wedge \varphi$ dürfen weiterhin vorkommen.

```
public boolean findBasicContradictions()
```

f) Für jede Formel φ gilt, dass $\varphi \wedge \mathit{true}$ und $\mathit{true} \wedge \varphi$ äquivalent zu φ sind. Erstellen Sie eine Methode, die alle Teilformeln $\varphi \wedge \mathit{true}$ und $\mathit{true} \wedge \varphi$ durch φ ersetzt.

```
public boolean removeTrueConjuncts()
```

g) Es gilt, dass $\varphi \wedge \mathit{false}$ und $\mathit{false} \wedge \varphi$ äquivalent zu *false* sind. Erstellen Sie eine Methode, die alle Teilformeln $\varphi \wedge \mathit{false}$ und $\mathit{false} \wedge \varphi$ durch *false* ersetzt.

```
public boolean findFalseConjuncts()
```

h) Erstellen Sie eine Methode, die alle implementierten Vereinfachungen aus Aufgabe 3(g) und 4(c)-(g) wiederholt anwendet. Der Baum soll dabei so lange wie möglich vereinfacht werden.

```
public void reduce()
```

Hinweis: Beachten Sie, dass die einmalige Anwendung der Vereinfachungsmethoden nicht ausreicht. Um eine maximale Vereinfachung zu erreichen, dürfen Sie hier ausnahmsweise doch eine Schleife benutzen. Machen sie Gebrauch vom Rückgabewert der Vereinfachungsmethoden.

i) Dokumentieren Sie alle Methoden, die als `public` markiert sind, indem Sie die Implementierung mit Javadoc-Kommentaren ergänzen. Diese Kommentare sollten eine allgemeine Erklärung der Methode sowie weitere Erklärungen jedes Parameters und des `return`-Wertes enthalten. Verwenden Sie innerhalb des Kommentars dafür die Javadoc-Anweisungen `@param` und `@return`.

Benutzen Sie das Programm `javadoc`, um Ihre Javadoc-Kommentare in das HTML-Format zu übersetzen. Überprüfen Sie mit einem Browser, ob das gewünschte Ergebnis generiert wurde. (Falls `javadoc` ihre Abgabe nicht kompiliert, werden **keine** Punkte vergeben.) Bitte drucken Sie die generierten Dateien, der Umwelt zuliebe, *nicht* aus.

Aufgabe 5 (Deck 5): (Codescape)

Lösen Sie die Räume von Deck 5 des Spiels Codescape.
Ihre Lösung für Räume dieses Codescape Decks wird nur dann für die Zulassung gezählt, wenn Sie die Lösung bis Montag, den 2.12.2019, um 12:00 Uhr abschicken.