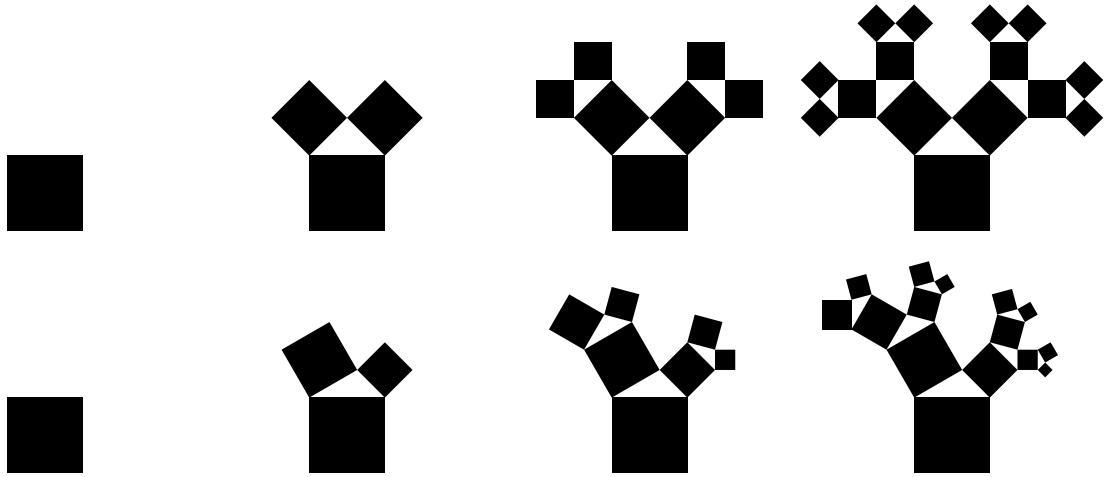
Prof. Dr. J. Giesl
S. Dollase, M. Hark, D. Cloerkes

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).



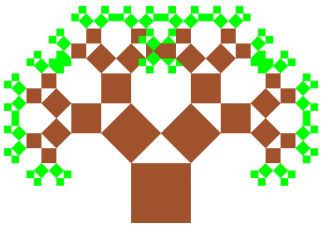
Zur Berechnung der jeweils nächsten Quadratlängen eignet sich der Sinussatz. Sei ℓ die Länge des unteren Quadrats. Dann folgt aus dem Sinussatz, dass die Länge des linken Quadrats genau $\frac{\sin(\beta)\cdot\ell}{\sin(180°-\alpha-\beta)}$ und die Länge des rechten Quadrats genau $\frac{\sin(\alpha)\cdot\ell}{\sin(180°-\alpha-\beta)}$ 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

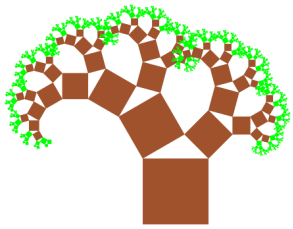
- eine `int`-Zahl `leftAngle`, welche dem Winkel α entspricht
- eine `int`-Zahl `rightAngle`, welche dem Winkel β entspricht
- eine `int`-Zahl `switchLength`, welche die maximale Länge von Quadraten angibt, welche grün gezeichnet werden sollen

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.

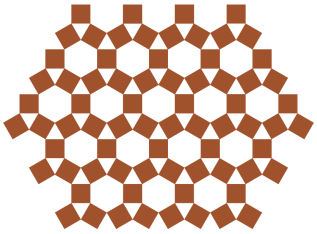
Dokumentieren Sie die Methode `paintPythagorasTree`, indem Sie die Implementierung wie auf Blatt 5 mit Javadoc-Kommentaren ergänzen, auch für jeden Parameter. Stellen Sie sicher, dass `javadoc` kompiliert.
Zum Testen Ihrer Implementierung eignen sich die folgenden Aufrufe Ihres Programms (darüber finden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

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.

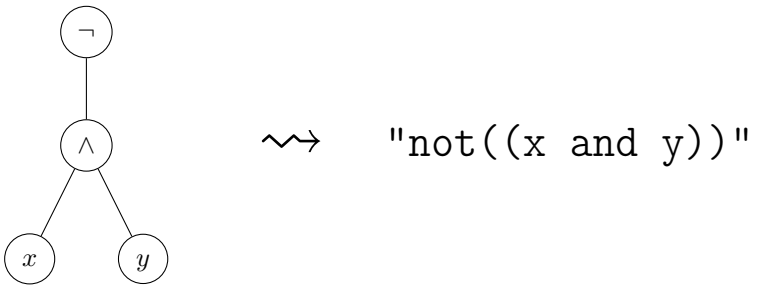
Hinweis: Überlegen Sie, wie Sie mit Hilfe der vorhandenen Elemente Disjunktionen modellieren können.

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:



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

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