Bitte geben Sie AVLBaum.java und EditPanel.java bis zum 17.01.2021 ab!

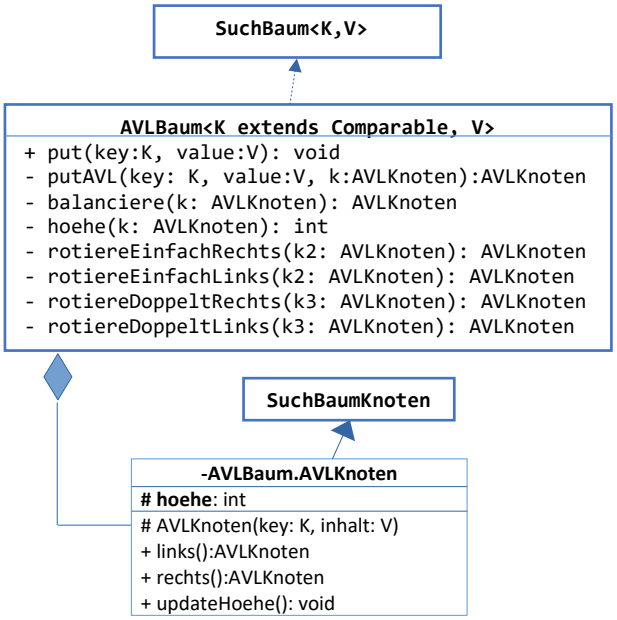
Aufgabe 28

AVL Baum

Wir implementieren einen AVL-Baum auf der Basis unserer Suchbaum Klasse.

Wichtig: bitte ändern Sie die Sichtbarkeit der Methode *SuchBaum.put(key, value, SuchBaumKnoten)* von *protected* zu *private*, damit wir sie nicht aus versehen rufen können!

Die von *SuchBaumKnoten* abgeleitete innere Klasse ***AVLKnoten*** hat die zusätzlichen Eigenschaft ***hoehe***.



Die Methoden ***links()*** und ***rechts()*** geben den jeweiligen Teilbaum als AVLKnoten zurück, indem sie einen Cast anwenden. Das erspart uns eine Vielzahl von Casts beui den Rotationen.

Die Methode ***updateHoehe*** setzt die Höhe des aktuellen Knotens auf 1 plus der Höhen seines höchsten Teilbaums. **Sie ist nicht rekursiv!** Ein nicht existierender Teilbaum wird mit der Höhe 0 angerechnet.

Die ***AVLBaum***-Methode ***put*** wirft eine *IllegalArgumentException* falls *key* oder *value null* sind. Dann ruft sie die Methode *putAVL* und übergibt *(AVLKnoten)wurzel* als letzten Parameter. Das Ergebnis der Methode wird in *wurzel* gespeichert.

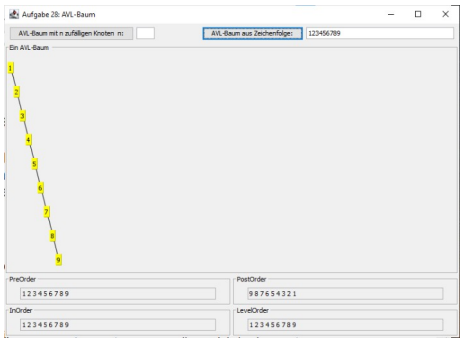
Die Methode ***putAVL*** sieht ebenfalls gleich aus wie die entsprechende Methode *put* in Suchbaum. Falls *k null* ist, wird ein neuer *AVLKnoten* zurückgegeben. Falls *key* und *k.key* gleich sind, wird *k.inhalt* ersetzt, ansonsten wird *k.setLinks* bzw. *k.setRechts* gerufen und als Parameter der rekursive Aufruf von *putAVL* übergeben.

Die letzte Anweisung ist anders: statt *k* geben wir *balanciere(k)* zurück.

Eine erste Version von ***balanciere*** gibt einfach nur **k** zurück. Unser AVL-Baum sollte jetzt ein gewöhnlicher Suchbaum sein, da noch keine Rotationen implementiert sind.

Schreiben Sie deshalb erst einmal die Klasse ***Aufgabe28***, die exakt der Klasse *Aufgabe26* entspricht und statt eines Suchbaums einen AVL-Baum erzeugt und überprüfen Sie dass der AVL Baum wie ein Suchbaum funktioniert wie rechts gezeigt.

Dann implementieren wir zunächst die **einfachen Rotationen**. Die Methode ***rotiereEinfachRechts(k2: AVLKnoten): AVLKnoten*** arbeitet nach folgendem Schema:



```
     k2                k1
    /  \              /  \
   k1   Z    -->     X    k2
  /  \                   /  \
 X    Y                 Y    Z
```

Zunächst wird *k2.links()* in der lokalen Variable ***k1*** gespeichert. Dann wird *k2.setLinks* gerufen um den rechten Teilbaum von *k1* als linken Teilbaum in *k2* einzuhängen. Schließlich wird k2 zum rechten Teilbaum von *k1* gesetzt. Jetzt muss noch für *k1* und *k2 updateHoehe* gerufen werden um die geänderten Höhen zu berechnen und am Ende wird *k1* zurückgegeben.

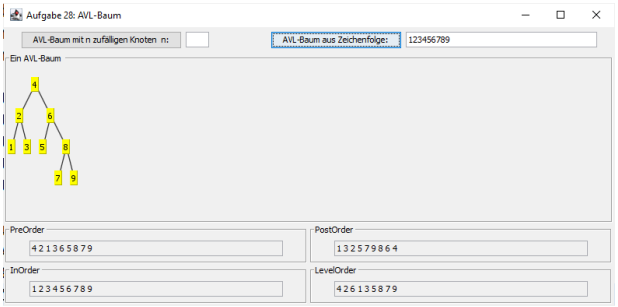
Die Methode ***rotiereEinfachLinks*** arbeitet analog nach dem folgenden Schema:

```
    k2                  k1
   /  \                /  \
  Z    k1     -->     k2   Y
      /  \           /  \
     X    Y         Z    X
```

Um die einfachen Rotationen anzuwenden, schreiben wir zunächst die Methode ***hoehe(k:AVLKnoten)*** welche *0* zurückgibt, falls *k null* ist und ansonsten *k.hoehe* zurückgibt.

Dann erweitern wir die Methode ***balanciere*** wie folgt:

- wir berechnen den balance-Faktor von k unter Verwendung der Methode *hoehe*,
- falls der balance-Faktor größer 1 ist rufen wir *k=rotiereEinfachRechts(k)*
- falls der balance-Faktor kleiner -1 ist rufen wir *k=rotiereEinfachLinks(k)*
- zum Schluss rufen wir *k.updateHoehe()*



Jetzt sollte der AVL Baum für alle Beispiel mit einfachen Rotationen, z.B. die Zeichenfolge 123456789 schon einen richtigen AVL Baum liefern wie rechts gezeigt.

Die Methode ***rotiereDoppeltRechts(k3: AVLKnoten): AVLKnoten*** arbeitet nach folgendem Schema:

```
  k3                 k2
   /                /  \
 k1       -->      k1   k3
   \
    k2
```

Wir rufen also zunächst r*otiereEinfachLinks(k3.links())* und speichern das Ergebnis in der lokalen Variable *k2*. Dann machen wir *k2* zum linken Knoten von *k3*. Schließlich wird *k3* nach rechts rotiert und das Ergebnis in k2 gespeichert. Nachdem die Höhen von k3 und k2 neu berechnet wurden, wird k2 zurückgegeben.

Die Methode ***rotiereDoppeltLinks*** arbeitet analog nach folgendem Schema:

```
  k3                 k2
    \               /  \
     k1     -->    k3   k1
     /
    k2
```

Um die beiden Methoden anzuwenden, erweitern wir *balanciere* wie folgt:

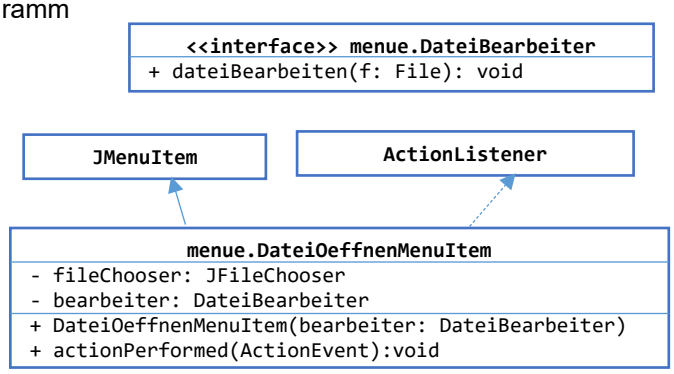
- wir rotieren nur dann einfach nach Rechts, wenn der Pfad unterhalb von *k* keinen Knick hat, also die Höhe von *k.links().links()* größer oder gleich der Höhe von *k.links().rechts()* ist, ansonsten wird doppelt nach Rechts rotiert.
- Analog wird nur dann einfach nach Links rotiert, wenn die Höhe von *k.rechts().rechts()* größer oder gleich der Höhe von *k.rechts().links()* ist, ansonsten ist wieder eine Doppelrotation nach Links nötig.

Damit ist unser AVL Baum fertig – testen Sie ihn ausgiebig.

Aufgabe 29

Datei lesen, kleiner Text-Editor

Das Programm dieser Aufgabe ist der Prototyp eines kleinen Text-Editors. Es liest Textdateien ein und stellt sie in einer *JTabbedPane* dar. Es ist wichtig, das Programm von Anfang an gut zu strukturieren:



Das Interface ***DateiBearbeiter*** fordert die Methode *dateiBearbeiten* mit einem Parameter vom Typ *File*.

Die Klasse ***DateiOeffnenMenuItem*** ist ein Menüeintrag mit dem Titel „Öffnen“ der sein eigener *ActionListener* ist.

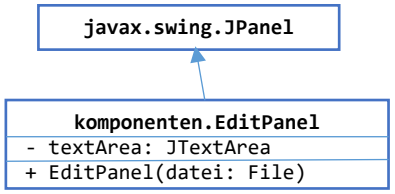
Der **Konstruktor** ruft *super("Öffnen")* und speichert seinen Parameter in der Instanzvariable. Der *JFileChooser* wird initialisiert und mit

setFileSelectionMode wird festgelegt, dass nur Dateien und keine Verzeichnisse gewählt werden können (JFileChooser.FILES_ONLY).

actionPerformed ruft *showOpenDialog* für den *JfileChooser,* wenn Sie dabei *this.getParent()* als ersten Parameter übergeben, wird der Dialog modal und erscheint über der Anwendung. Vergleichen Sie den Rückgabewert von *showOpenDialog* mit JFileChooser.APPROVE_OPTION um festzustellen, ob eine Datei ausgewählt wurde. Falls ja, rufen Sie *dateiLesen* des *DateiBearbeiters*.

Menüeinträge vom Typ *DateiOeffnenMenuItem* können wir jetzt allgemein dazu verwenden, um eine Datei auszuwählen und sie einem Objekt vom Typ *DateiBearbeiter* zu übergeben, der die Datei öffnet und etwas damit macht.

Die nächste Klasse ***EditPanel*** ist von *JPanel* abgeleitet.

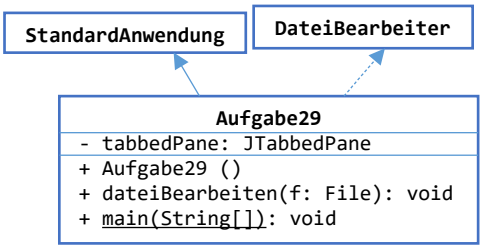


Der **Konstruktor** definiert sein Layout als *BorderLayout* und initialisiert die Instanzvariable *textArea*. Dann legt er eine *JScrollPane*, deren Viewport die *TextArea* ist, auf seine Oberfläche. Zum Schluss liest er den Inhalt der Datei in die Text-Area. Am einfachsten erzeugen Sie dazu ein *FileReader* Objekt, das Sie der Methode *read* der Text-Area übergeben, der zweite Parameter von *read* kann *null* sein. Dabei müssen die Fehler *IOException* und *FileNotFoundException* mit *try{ … } cach* abgefangen werden. Verwenden Sie *JOptionPane.showMessageDialog*, um dem Anwender die Fehler anzuzeigen. Wenn Sie der Methode *showMessageDialog* als ersten Parameter *this.getParent()* übergeben, wird der Dialog modal und erscheint über der Anwendung.

Die Anwendung ***Aufgabe29*** ist eine von *StandardAnwendung* abgeleitete Klasse.

In ***main*** rufen Sie *starteAnwendung*.

Der ***Konstruktor*** ruft super, gibt sich ein *BorderLayout*, initialisiert *tabbedPane* und legt es auf *this* und gibt sich ein Menü „Datei“ mit den Einträgen „Öffnen“ und „Beenden“.



Die Methode ***dateiBearbeiten*** speichert ein neues Objekt vom Typ *EditPanel* in der lokalen Variable ***neuesPanel***. Das neue Panel wird als neues Tab auf die *TabbedPane* gelegt (Methode *addTab*), als Name des Tabs nehmen Sie den Namen der Datei, den Sie mit *getName()* bekommen. Schließlich rufen Sie für die *TabbedPane* noch *setSelectedComponent(neuesPanel)*, damit das neue Panel angezeigt wird.

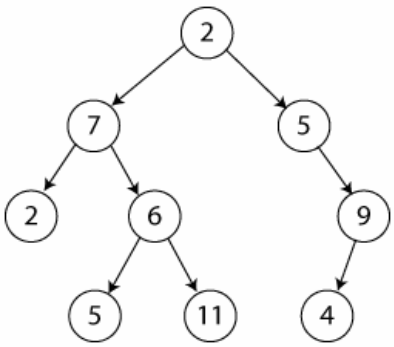
Mit wenig Aufwand haben wir einen kleinen Editor programmiert, mit dem man mehrere Dateien öffnen und bearbeiten kann – nur das Abspeichern fehlt noch.

Aufgabe 30

a) Geben Sie für den nebenstehenden Binärbaum (es ist kein Suchbaum) die Pre-, Post-, In- und Level-Order an.

b) Erzeugen Sie den AVL Baum aus den Buchstaben des Wortes „KASTENWAGEN“ und geben Sie für diesen Baum die Level- und Post-Order an.

b) Erzeugen Sie den AVL-Baum aus den Buchstaben des Wortes HEILIXBLECHLE und geben Sie seine Post-Order an.



Stellen Sie sich weitere Aufgaben zur Erzeugung von Such- und AVL-Bäumen aus Zahlen- oder Buchstabenfolgen bis Sie darin sicher sind.

Potentielle Prüfungsaufgaben

Programmieren Sie (ohne Hilkfsmittel!) einfache Methoden zur Klasse ***Baum<T>*** z.B.:

- tiefe(): int istBlatt(): boolean inOrder(Bearbeiter): void
- getVaterInhalt(): T anzahlKinder(): int hoehe(): int