Binary Maple

Bericht zur Anwendungsentwicklung

“The tree that is beside the running water is fresher and gives more fruit”.
- Teresa von Ávila (1515 - 1582)

Ein Projekt von
Simon Bullik und
Nils Malte Kiele

10. November 2019
Fachlehrer: Jürgen Wolkenhauer

Inhaltsverzeichnis

Allgemeines

Das von uns erarbeitete Programm samt Klassen, Methoden, Variablennamen und Kommentaren ist auf Englisch geschrieben, da dies dem internationalen Standard von Software entspricht und Problemen wie z. B. der Verwendung von Umlauten vorbeugt.

Zur Bearbeitung dieses Projekts wurde uns Zeit in der Schule zur Verfügung gestellt, die wir genutzt haben. Ein erheblicher Teil der Arbeit geschah aber außerhalb des Unterrichts.

Aufgabenstellung

Unsere Aufgabe war es, eine Applikation zu schreiben, die einen binären Suchbaum realisiert. Dieser soll visuell dargestellt und über eine Benutzeroberfläche verändert werden können. Es wurde erwartet, dass die Applikation in Java (Version 11 oder neuer) programmiert und die Benutzeroberfläche mithilfe von JavaFX und Graphviz erstellt wird. Die Anforderungen an den Baum sind, dass Elemente hinzugefügt und gelöscht werden können. Zudem soll nach Elementen im Baum effizient gesucht und alle Elemente sollen in

den Traversierungsarten In-Order Pre-Order, Post-Order und Level-Order durchlaufen werden können.

Die Werte (“Schlüssel”) des Baumes sollen in Knoten dargestellt werden. Zu umfangreiche Werte (wie z. B. lange Strings) sollen gekürzt dargestellt werden, damit der Baum nicht zu unübersichtlich wird. Auch große Bäume sollen möglichst anschaulich dargestellt werden; dazu sollte Zoom- oder Scroll-Funktionalität in der GUI vorhanden sein.

Neben einem einfachen binären Suchbaum ist für die volle Erfüllung der Erwartungen die Realisierung eines Rot-Schwarz-Baums gefordert, der seine Werte automatisch balanciert und somit die Entartung zu einer Liste verhindert.

Technische Ausstattung

Das Programm wurde in der IDE IntelliJ 2019 und auf Basis von Java 11 entwickelt. Um das Programm korrekt ausführen zu können, muss eine kompatible Java-Version (11 oder neuer) auf dem ausführenden Rechner vorhanden sein.

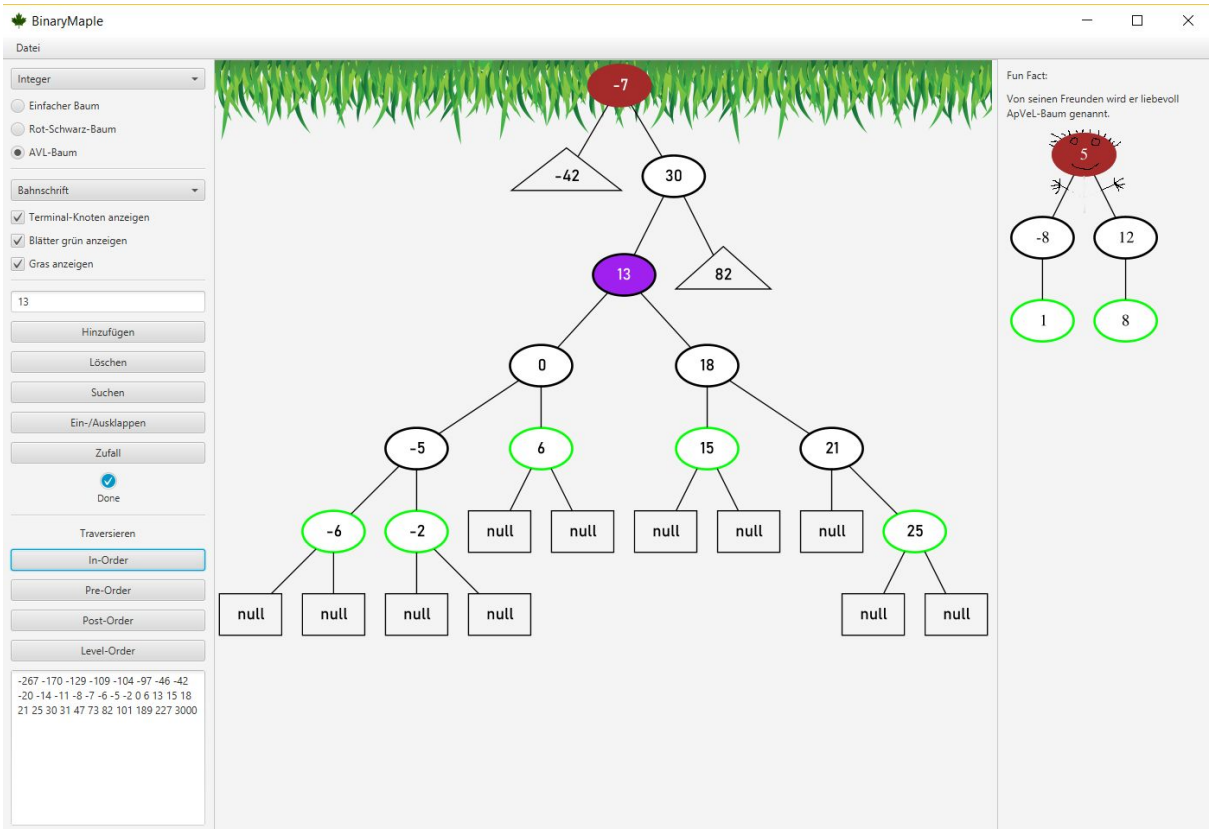
Die Programmabhängigkeiten werden mithilfe von Maven durch die pom.xml-Datei aufgelöst. Der Benutzer kommt hierbei nicht in direkten Kontakt mit den benötigten Bibliotheken.

Es wird empfohlen, die GUI auf einem möglichst großen Bildschirm darzustellen, damit genug Platz ist, um alle Schaltflächen anzuzeigen und somit die volle Funktionalität des Programms genutzt werden kann.

Benutzeranleitung

Zur Ausführung des Programms empfehlen wir die IDE *IntelliJ*. Zunächst müssen alle Abhängigkeiten korrekt aufgelöst werden - unter *Java* macht *Maven* dies selbstständig.

Um die Anwendung zu starten, kann die mitgelieferte Startkonfiguration in IntelliJ genutzt oder einer der beiden Befehle aus der Readme ausgeführt werden.
Der Start der GUI kann einige Sekunden dauern. Sobald die Benutzeroberfläche dargestellt wird, kann die Benutzung des Programms beginnen. Die Anordnung der GUI-Elemente ändert sich im Verlauf der Benutzung nicht; es stehen also alle Funktionalitäten auf einen Blick zur Verfügung, was im folgenden genauer erklärt ist.



Die GUI ist in drei Bereiche aufgeteilt, die nun im Detail beschrieben werden.

1. In der linken Navigationsleiste werden alle Interaktionsmöglichkeiten mit dem Baum zur Verfügung gestellt. Diese werden nun von oben nach unten genauer erklärt.
    a. **Dropdown-Menü zur Auswahl des Datentyps**: Hier kann zwischen drei verschiedenen Datentypen gewählt werden (Integer, Double oder String), die die Inhalte der Knoten beschreiben. Der dargestellte Baum kann zu einem beliebigen Zeitpunkt immer nur Datentypen einer Art besitzen, damit das Vergleichen von Werten (was für die Anordnung des Baums wichtig ist) klar definiert wird.
    b. **Radio-Button-Menü zur Auswahl der Baumart**: Die Anordnung der Werte wird maßgeblich von der Art des binären Suchbaums beeinflusst. Hier kann zwischen der einfachen (unbalancierten) Variante, dem Rot-Schwarz-Baum (Rebalancierung durch Farben der Knoten) und dem AVL-Baum (Rebalancierung durch Höhenunterschiede der Teilbäume von Knoten) gewählt werden. Je nach Auswahl ändert sich auch die Info-Ansicht rechts in der GUI. Ein bereits vorhandener Baum wird mit all seinen Werten beim Ändern des Baumtyps in eine andere Darstellung überführt. Hierbei kann es zu Änderungen in der Anordnung der Werte kommen, aber die Regeln eines Binärbaums (kleinere Elemente eines Knotens links, größere rechts) werden nicht verletzt.
    c. **Schriftart**: In diesem Dropdown-Menü kann zwischen einer Vielzahl von Schriftarten gewählt werden, in denen die Knoten des Baums dargestellt werden können.

d. **Darstellungsoptionen**: Es werden drei Möglichkeiten angeboten, die Darstellung zu individualisieren. Diese sind hier kurz aufgeführt.
    i. **Terminal-Knoten anzeigen**: Ist dieser Modus aktiviert, so haben alle Knoten mit Wert zwei Kinder, möglicherweise Null-Knoten (auch Terminal-Knoten genannt)
    ii. **Blätter grün anzeigen**: Alle Knoten mit Werten, die weder links noch rechts einen Kindknoten mit Wert haben (sogenannte *Blätter*), können mit dieser Option durch einen grünen Rand hervorgehoben werden.
    iii. **Gras anzeigen**: Damit sich der Benutzer noch mehr zu Hause in unserer Oberfläche fühlt, bieten wir die Möglichkeit, die Wurzel des Baums mit frischem Gras umgeben zu lassen.
e. **Eingabefeld & Interaktions-Button**: Hiermit kann der Benutzer direkt mit dem Baum interagieren. Zunächst muss ein Wert passenden Datentyps in das Eingabefeld geschrieben werden. Mit diesem Wert kann der Baum nun auf eine bestimmte Art beeinflusst werden, je nachdem, welcher Button gedrückt wird.
    i. **Hinzufügen**: Hiermit wird der gewünschte Wert dem Baum hinzugefügt. Bedingung für die erfolgreiche Ausführung ist, dass der Wert noch nicht im Baum existiert und er dem angegebenen Datentyp entspricht.
    ii. **Löschen**: Hiermit wird der gewünschte Wert aus dem Baum gelöscht, falls er existiert.
    iii. **Suchen**: Durch Betätigen dieser Schaltfläche wird der eingegebene Wert, wenn er im Baum existiert, farblich hervorgehoben (im Beispiel oben ist das der Knoten mit Wert 13)
    iv. **Ein-/Ausklappen**: Um den Baum übersichtlich zu halten, können Teilbäume ein- und später wieder ausgeklappt werden werden. Diese werden dann als dreieckige Knoten dargestellt (im Beispiel oben sind die Knoten mit den Werten -42 und 82 eingeklappt).
    v. **Zufall**: Um den Baum schnell mit Werten zu füllen, können mit einem Klick auf diesen Button so viele Knoten mit Zufallswerten dem Baum hinzugefügt werden, wie im Eingabefeld in Form einer Zahl angegeben sind.
f. **Statusanzeige**: Dem Benutzer wird unter den Interaktions-Buttons visuell dargestellt, ob die Applikation momentan eine Operation des Benutzers ausführt (drehender Kreis) oder nicht (Haken, siehe Beispielbild).
g. **Schaltflächen zum Traversieren**: Die vier Buttons mit der Aufschrift “In-Order”, “Pre-Order”, “Post-Order” und “Level-Order” ermöglichen das Durchlaufen aller Werte des aktuell dargestellten Baums in verschiedenen Reihenfolgen, die im Kontext von binären Suchbäumen klar definiert sind. Das Ergebnis der Traversierung wird in darunterliegenden Textfeld dargestellt und kann zur besseren Ansicht bei vielen Knoten einfach herauskopiert werden (im Beispiel oben sind die Werte in der In-Order angegeben). Zu beachten ist, dass auch die Werte von Knoten aufgeführt werden, die zurzeit wegen eingeklappter Teilbäume nicht zu sehen sind.

2. Der größte Bereich in der Mitte der Applikation stellt den aktuellen Baum dar. Im Beispiel oben haben wir die verschiedenen Darstellungserweiterungen wie das Anzeigen von Gras, grünen Blättern, Terminal-Knoten, einem gesuchten Wert und eingeklappter Teilbäume beispielhaft angewendet. Passt der Baum nicht vollständig auf die Benutzeroberfläche, so kann durch horizontale und vertikale Scroll-Balken der Ausschnitt des sichtbaren Baums einfach verändert werden. Um die Visualisierung des Baums abzurunden, ist die Wurzel (der Wert in der obersten Ebene) immer braun dargestellt, ähnlich zum Stamm eines echten Baums.
3. Am rechten Rand der GUI stellen wir dem Benutzer eine Auswahl von Informationen in Form von Text und Bild dar, die dabei hilft, sich in die aktuelle Baumart hineinzuversetzen. Die verschiedenen Charaktereigenschaften der Bäume werden hier spielerisch aufgezeigt und visualisiert.

Um den Workflow für den Benutzer zu verbessern, können für die wichtigsten Interaktionen auch folgende Shortcuts (Tastenkombinationen) verwendet werden.

1. ALT + H Wert hinzufügen
2. ALT + L Wert löschen
3. ALT + S Wert suchen und hervorheben
4. ALT + K Wert ein- oder ausklappen
5. ALT + Z Zufallswerte hinzufügen

6. ALT + E Baumart zu einfachem Baum ändern
7. ALT + R Baumart zu Rot-Schwarz-Baum ändern
8. ALT + A Baumart zu AVL-Baum ändern

9. ALT + T Terminal-Knoten anzeigen oder ausblenden
10. ALT + B Blätter farblich markieren oder nicht
11. ALT + G Gras anzeigen oder ausblenden

Über dem Reiter “Datei” oben links in der Applikation kann weitere Programmfunktionialität genutzt werden.

1. Ein (zuvor gespeicherter) Baum kann aus eine XML-Datei geladen werden.
2. Der aktuell dargestellte Baum kann als XML-Datei gespeichert werden. Dabei bleibt der Datentyp der Werte sowie die Art des Baums (Einfach, Rot-Schwarz oder AVL) erhalten.
3. Der aktuell dargestellte Baum kann lokal auf dem Computer des Benutzers gespeichert werden, z. B. als Bild. Hierbei kann aus verschiedenen Dateiformaten gewählt werden. Neben PNG und SVG kann der Baum unter anderem als DOT (Format von Graphviz, um einen Graphen in Textform darzustellen) gespeichert werden.

Funktionsbeschreibung

Unser Programm erfüllt die Anforderungen der Aufgabenstellung. Darüber hinaus haben wir eine Vielzahl von weiteren Funktionalitäten, Features und Details implementiert, die die Anforderungen übertreffen. Diese sind hier aufgeführt:

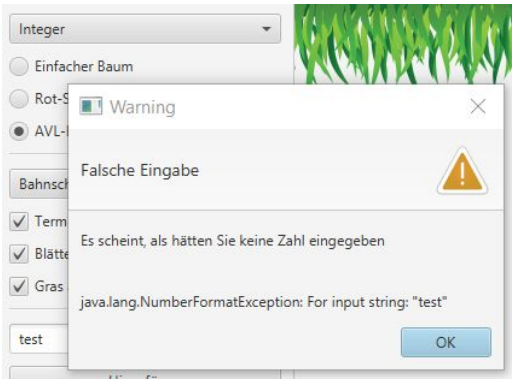
1. Es wurden von uns drei verschiedene Arten von binären Suchbäumen mit den wichtigsten Funktionalitäten wie dem korrekten Einfügen und Löschen von Knoten implementiert:
    a. klassischer Baum (ohne Selbst-Balancierung)
    b. Red-Black-Tree
    c. AVL-Tree
2. Werte können einem Baum **hinzugefügt** und wieder **gelöscht** werden, wobei mögliche Änderungen in der Anordnung des Baums berücksichtigt werden.
3. Es kann nach einem Wert **gesucht** werden und dieser wird farblich hervorgehoben.
4. Teilbäume von Knoten können an jeder Stelle ein- und wieder ausgeklappt werden, was die Übersichtlichkeit enorm erhöhen kann. **Eingeklappte Knoten** werden als Dreiecke dargestellt, mit dem Wert der “lokalen Wurzel” noch vorhanden im Dreieck selbst. Somit ist einfach erkennbar, wo ein Teilbaum eingeklappt wurde und mithilfe welchen Knotenwertes dieser wieder ausgeklappt werden kann. Auch mit verschachtelten eingeklappten Bäumen geht unsere Applikation intuitiv um. Eine Beispielsituation hierfür ist:
    a. Knoten A befindet sich im Teilbaum von Knoten B.
    b. A wird eingeklappt.
    c. B wird eingeklappt.
    d. A wird wieder ausgeklappt.
    e. B wird wieder ausgeklappt. Ohne Schritt (d) wäre A nachwievor eingeklappt, aber mit Schritt (d) sind nun sowohl A als auch B wieder ausgeklappt.

    Beim Exportieren des Baums als Bild werden eingeklappte Knoten wie auch in der GUI mit Dreiecken dargestellt.
5. Um die Erstellung größerer Bäume schnell und einfach zu gestalten und wenn die genauen Werte nicht relevant für den Benutzer sind, kann durch das **zufällige Hinzufügen** einer durch den Benutzer bestimmten Menge von Werten ein Baum beachtlicher Größe in nur wenigen Momenten erstellt werden.
6. Alle Knoten des Baums können mit den vier **Traversierungsarten** In-Order Pre-Order, Post-Order und Level-Order mit einem einfach Knopfdruck durchlaufen werden. Das Ergebnis wird direkt in der Benutzeroberfläche ausgeben, aus der man die Daten einfach kopiert kann.
7. Bäume werden mit zentriert formatierten Werten in den Knoten und eindeutigen Verbindungen zu Kinder- und Elter-Knoten **anschaulich dargestellt**.
8. Über **vertikale und horizontale Scrollbalken** kann der aktuelle Ausschnitt eines größeren Baums verändert werden. Dem Benutzer wird damit erlaubt, einfach durch die Darstellung des aktuellen Baums zu navigieren.
9. Wir haben uns bewusst gegen die Kürzung von Inhalten (z. B. bei langen Strings) entschieden, da eine Zeichengrenze hier relativ wahllos gewählt wäre und oftmals

die gesamte Information von Knoten relevant sind. Um trotzdem für Übersichtlichkeit zu sorgen und sehr lange, schmale Knoten zu vermeiden, kann es **Knoten unterschiedlicher Größe** geben. Dabei werden längere Inhalte (bei Zahlen wie bei Strings) in mehreren Zeilen dargestellt. Somit bleibt das Verhältnis von Höhe zu Breite der Knoten angemessen und Werte müssen nicht gekürzt werden.

10. Es kann zwischen **verschiedenen Datentypen** ausgewählt werden. Direkt in der GUI kann also z. B. von Integers als Knotenwerte zu Strings gewechselt werden.
11. Es wurden **zusätzliche visuelle Elemente** wie das Gras und die grün gefärbten Blätter sowie die braune Wurzel des Baums implementiert, um die Benutzeroberfläche ansprechender zu gestalten.
12. Bäume können als **XML-Datei gespeichert und geladen** werden. Beim Laden eines vorhandenen Baums wird von der Applikation automatisch die Art des Baums und der verwendete Datentyp erkannt und angepasst, sodass es zu keinen Konflikten kommt, wenn der Benutzer z. B. einen String-Baum im Integer-Modus laden möchte.
13. Bäume können in verschiedene Formate, unter anderem als Bilder, **exportiert** und lokal auf dem PC gespeichert werden - kein Grund mehr für nervige Screenshots.
14. **Tastatur-Shortcuts** erleichtern die Benutzung, da der Benutzer wahlweise eine einfache Tastenkombination statt einer Schaltfläche bedienen kann.
15. Dem Benutzer wird angezeigt, in welchem **Status** sich das Programm bei der Ausführung einer Aktion (z. B. Hinzufügen eines Werts) befindet.

16. Das Verlangen des Benutzers, seinen oder ihren ganz persönlichen Baum zu erstellen, wird durch die **Auswahl der Schriftart** befriedigt. Hier kann in einem übersichtlichen Dropdown-Menü zwischen einer großen Menge von Schriftarten gewählt werden, in denen die Knoten des Baums dargestellt werden können.
17. Beim Betätigen der Schaltflächen “Hinzufügen” und “Zufall” wird ein **Sound-Effekt** in Form eines sanften Vogelgezwitschers abgespielt. Schließlich handelt dieses Projekt von Bäumen und da fühlen sich auch Vögel wohl.
18. Buttons und Schaltflächen sind **möglichst intuitiv** angelegt worden, sodass alle Funktionalitäten des Programms auf einen Blick erkennbar und einfach anzuwenden sind.
19. Fehlerhafter Umgang des Benutzers mit der Applikation wird in Form von **Fehlermeldungen als Pop-Up** dargestellt. Dabei wird der Grund des Fehlers mithilfe eines klar verständlichen Textes und der entsprechenden *Exception* angezeigt.



Erläuterung des Programms

Abhängigkeiten

Unser Programm ist von einer Vielzahl von Bibliotheken abhängig, die im folgenden näher erläutert sind.

JavaFX

JavaFX ist ein Framework zur Erstellung von graphischen Oberflächen mit *Java*. Seit der Version 11 ist es Teil vom *OpenJDK* und wird unter dem Projektnamen *OpenJFX* weiterentwickelt.

graphviz-java

Graphviz ist eine Open-Source-Visualisierungssoftware für Graphen. *graphviz-java* ist eine Open-Source-Bibliothek für *Java*, um die die Software *Graphviz* mit *Java* benutzen zu können. Die Bibliothek bringt die JavaScript-Bibliothek *viz.js* mit, welche auf der *JavaScript Engine V8* oder *Nashorn* ausgeführt wird. Aus diesem Grund braucht man keine lokale Installation von Graphviz.

logback-classic

Da die *Graphviz*-Bibliothek dieses Modul für das Loggen in der Konsole benötigt, haben wir uns auch dazu entschieden, Fehler hierüber in der Konsole ausgeben zu lassen.

xstream

Dies ist eine Deserialisierungs- und Serialisierungsbibliothek für *XML* in *Java*, die wir dafür benutzen, den aktuellen Baum als *XML*-Datei speichern und wieder laden zu können.

commons-lang3

Diese Bibliothek stellt uns einige Klassen für die zufällige Generierung von Werten zu Verfügung. Vor allem für die Erstellung zufälliger Strings war dies hilfreich.

annotations

Mit Hilfe dieser Bibliothek von *JetBrains* haben wir *Annotations* hinzugefügt, welche anzeigen, ob Variablen *Null* sein können oder nicht. Durch die Integration der IDE wurden uns so Warnungen angezeigt, wenn wir keine *Null*-Checks machen aber der Wert *Null* sein kann und vice versa. So haben wir *NullPointerExceptions* vermeiden können.

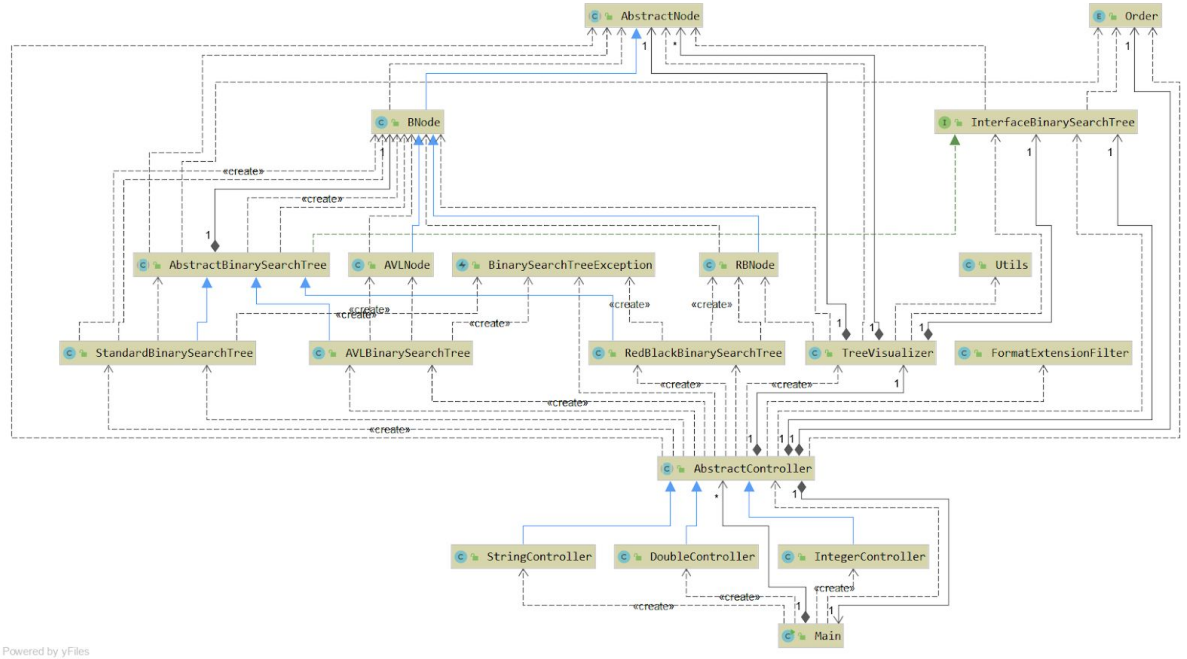
lombok

Diese Bibliothek benutzen wir, um automatisch *Getter* und *Setter* mittels *Annotations* zu generieren.

junit-jupiter

Um unsere Implementierungen der Bäume zu testen, haben wir mit Hilfe von *JUnit* entsprechende Tests geschrieben.

UML-Klassendiagramm



Das obige Diagramm verdeutlicht die interne Struktur unserer Anwendung. Wir haben darauf geachtet, eine Balance zwischen möglichst wenig Code-Redundanz und klarer Programmstruktur herzustellen. Für eine elegante Lösung halten wir die jetzige Struktur.

Klassen wie *AbstractNode*, *InterfaceBinarySearchTree*, *AbstractBinarySearchTree*, *Order* und *BinarySearchTreeException* befinden sich auf einer Ordner-Ebene, da sie keiner genauen Baumarten zugewiesen sind, sondern allgemein gelten. Es ist zu beachten, dass sich auch die Klasse *BNode* hier befindet. Als klassischer Knoten eines einfachen Binärbaums gilt seine Funktionalität wie z. B. *getLeft()* und *getRight()* für alle Bäume.

In den drei Ordnern *standard*, *rb* und *avl* sind klar getrennt voneinander die Implementationen der entsprechenden Bäume zu finden. Der einfache Baum verwendet direkt die *BNode*-Klasse, da die Funktionalität hier vollkommen ausreicht. Für die balancierenden Baumarten haben wir jeweils eine eigene Knotenklasse im entsprechenden Ordner erstellt, die *BNode* erweitert (mit Farbe oder Höhen der linken und rechten Teilbäume). Dies ist ein typischer Fall von Vererbung.

Auch bei den Baumklassen ist Vererbung zu beobachten. *AbstractBinarySearchTree* implementiert die Klasse *InterfaceBinarySearchTree* und stellt Methoden wie z. B. *traverse()* und *getRoot()* zur Verfügung. Diese Methoden verbinden alle verschiedenen Arten der binären Suchbäume. Auf eine Implementation von *addValue()* und *delValue()* wird an dieser

Stelle aber verzichtet, da hier verschiedene Arten der Balancierung und Rotation verwendet werden müssen. Diese beiden Methoden werden also den entsprechenden Baumklassen in den Ordnern *standard*, *rb* und *avl* überlassen, welche *AbstractBinarySearchTree* erweitern.

Damit die Bäume verschiedene Datentypen aufnehmen können, werden *Generics* verwendet. Somit kann auf übersichtliche Art und Weise wie z. B.
StandardBinarySearchTree<Integer> tree = new StandardBinarySearchTree<>();
ein Baum mit dem gewünschten Datentyp (hier Integer) erstellt werden. Auch wenn in unserer GUI nur drei Datentypen zur Verfügung stehen, können theoretisch ohne weiteren Aufwand Bäume mit jeder Art von Datentyp erstellt werden, solange die Klasse des Datentyps *Comparable* erweitert und mit Objekten des gleichen Datentyps vergleichbar ist.

Um die verschiedenen Datentypen anzeigen zu können, verwenden wir verschiedene Controller. Der *AbstractController* macht die Hauptarbeit. Jedes Element hat hier seine Aktionen implementiert. *Abstract* ist der Controller, weil es zwei datentypspezifische *abstrakte* Methoden gibt, welche im *StringController*, *IntegerController* und *DoubleController* implementiert sind:

1. Parsen der Eingabe des Benutzer (entweder Text, Ganzzahl oder Kommazahl)
2. Generieren von Zufallswerten in dem spezifischen Typ

Um die Erweiterbarkeit so einfach wie möglich zu halten, muss ein neuer Controller nur den *AbstractController* implementieren und zur Liste *controllers* in der *Main*-Klasse hinzugefügt werden. Der Rest passiert automatisch.

Tests

Im Ordner “test” stellen wir Tests für alle logischen Komponenten der verschiedenen binären Suchbäume zur Verfügung. Die Tests für die verschiedenen Arten der Knoten (z. B. *BNode*, *AVLNode*) und die Baumklassen (z. B. *StandardBinarySearchTree* und *AVLBinarySearchTree*) wurden so erstellt, dass die Funktionalität aller relevanten Methoden überprüft wird.

Der größte Testaufwand bestand bei den Methoden zum Einfügen und Löschen von Knoten des Rot-Schwarz-Baums und des AVL-Baums. Hier haben wir eine breite Spanne von Anordnungen der Knoten zueinander getestet und darauf geachtet, dass alle Rotationsfälle mindestens einmal korrekt ausgeführt werden müssen, damit die Tests erfolgreich durchlaufen werden.

Um einzelne Tests auszuführen, kann in *IntelliJ* der grüne Play-Button neben dem entsprechenden Test gedrückt werden. Zum Ausführen all unserer Tests muss der Wurzel-Ordner des Projekts angeklickt sein, gefolgt von der Tastenkombination Ctrl + Shift + F10. Eine weitere Möglichkeit gibt es durch *Maven*. Es muss nur der Lifecycle *test* entweder über die Oberfläche von *IntelliJ* gestartet werden oder über die Kommandozeile mit *mvn test*.

Einzelheiten zur Software

Um den Rahmen dieses Dokuments nicht zu sprengen, haben wir die Dokumentation der Software ausgelagert. Sie ist im *doc*-Ordner zu finden und wurde automatisch mithilfe von *Javadoc* erstellt. Bei den von uns erstellten *Javadoc*-Beschreibungen haben wir darauf geachtet, kurz und prägnant die Aufgabe der einzelnen Klassen und Methoden zu schildern.

Die Namen der Klassen und Methoden, die Ordnerstruktur unseres Projekts sowie ergänzende Kommentare zu wichtigen Stellen im Code tragen dazu bei, dass die von uns erarbeitete Software übersichtlich und (hoffentlich auch für Außenstehende) einfach verständlich ist.

Erkenntnisgewinn

Dieses Projekt hat unser Wissen in verschiedenen Bereichen erweitert, was im Folgenden näher aufgeschlüsselt ist:

1. Wir haben uns mit der Bibliothek **Graphviz** auseinandergesetzt und sie im Zusammenhang mit JavaFX eingesetzt.
2. Mit Buttons, Dropdowns, Bildern und Sounds haben wir die Funktionalitäten von **JavaFX** ausgiebig genutzt.
3. Unsere Horizont haben wir auch in **Java** erweitert. Mit Interfaces, abstrakten Klassen und Generics haben wir viele mächtige Werkzeuge in einem großen Ganzen zusammen spielen lassen.
4. Mit so vielen Abhängigkeiten wie in bisher noch keinem unserer Schulprojekte waren wir froh, **Maven** nutzen zu können und uns damit mehr Klarheit und Einfachheit zu verschaffen. Außerdem war es uns so möglich, mithilfe von verschiedenen Plugins z.B. über Maven die Tests ausführen zu lassen, die JavaFX Oberfläche zu starten und unsere JavaDoc Generierung zu konfigurieren damit man z.B. auch Graphviz-Bilder anzeigen kann.
5. Zum ersten Mal haben haben wir durchgängig **Javadoc** verwendet, um eine übersichtliche Dokumentation der Software automatisch generieren zu lassen. Das hat dem Projekt zusätzliche Struktur gegeben und uns dazu veranlasst, genauer über den Zweck jeder Klasse und Methode nachzudenken.
6. Um frühzeitig Fehler zu erkennen, haben wir das *Continuous Integration System* **Travis** für das Projekt konfiguriert. So wurden bei jedem Commit auf *GitHub* alle unsere Tests ausgeführt, um direkt Fehler finden zu können.
7. Auch nicht trivial war die **Implementierung** der tatsächlichen Funktionalität der jeweiligen binären Suchbäume. Durch stundenlange Beschäftigung mit verschiedenen Rotationsfällen und besonderen Situationen bei der Anordnung der Knoten haben wir den mächtigen Datentyp der Binärbäume genauer kennenlernen dürfen.
8. Ohne **Versionsverwaltungssystem** wäre diese Teamarbeit weniger flüssig abgelaufen. Durch die Verwendung von *Github* konnten wir unsere Fortschritte teilen und einen Überblick über Veränderungen im Programmcode behalten.

Uns hat dieses Projekt gefordert, aber nicht überfordert. Wir haben viel (Frei-)Zeit investiert, um ein Ergebnis zu erzielen, mit dem wir zufrieden sind. Mit Feuer und Flamme waren wir in 175 Commits dabei, der Informatikwelt den Ahornbaum zu geben, den sie verdient hat.

- **codingWhale & simonIT**