Programmieren 1

Dr. Arne Schmidt

Wintersemester 2023/2024
Abgabe: 19.01.2024
Rückgabe: ab 22.01.2024

Übungsblatt 4

Abgabe der Lösungen bis zum 19.01.2024 um 18:00 Uhr im Gitlab vom IBR. Es werden nur ausführbare und rechtzeitig eingereicht Lösungen bewertet. Python-Programme müssen mit python3 interpretierbar, und Java-Programme mit Java 11 kompilierbar sein. Sofern nicht anders in der Aufgabe angegeben, dürfen **keine** Standardfunktionen benutzt oder Bibliotheken importiert werden, wenn sie nicht in der Vorlesung behandelt wurden.

Sofern nicht anders in der Aufgabe beschrieben, soll im Git die Ordnerstruktur wie vom Blatt vorgegeben beibehalten werden. D.h. für eine Aufgabe x von Blatt y sollen die verwendeten Source-Dateien (.py oder .java) in dem Ordner blatt[y]/pflichtaufgabe[x]/ gespeichert werden.

Hausaufgabe 1 (Smart Home): **(6 Punkte)**

In einem Haus werden Geräte oft auf verschiedene Arten eingeschaltet: manchmal reicht es einen einfachen Knopf (`Button`) zu drücken, manchmal muss ein Schalter (`Switch`) betätigt werden. Schon diese beiden Arten unterscheiden sich signifikant. Während ein Knopf nur gedrückt werden kann, ändert sich der Schalter von einer An-Position zu einer Aus-Position (oder umgekehrt). Ein Schalter kann ein System nur dann einschalten, wenn die Position von „aus“ auf „ein“ wechselt. Ansonsten schaltet dieser das System aus. Ein Knopf startet ein System immer, welches nach Durchführung gestoppt wird. Die Funktion `poll` betätigt den Knopf bzw. Schalter.

Um die Morgenroutine zu vereinfachen, wollen wir ein Smart-Home-System einführen, welches verschiedene Systeme am Morgen automatisiert startet. Die Systeme beinhalten entweder auf Knopfdruck gestartete Systeme, die nach der Durchführung wieder beendet werden, oder durch einen Schalter gestartete Systeme, die erst ausgeschaltet werden, wenn der Schalter erneut betätigt wird. Genauer sollen folgende Systeme betrachtet werden:

- Schlafzimmerlicht per Schalter
- Badezimmerlicht per Schalter
- Duschwasserheizsystem per Schalter
- Kaffeemaschine per Knopfdruck

a) Implementiere Klassen und Interfaces entsprechend dem Klassendiagramm aus Abbildung 1 innerhalb des Package smarthome.

   Auf der Konsole soll ausgegeben werden, was die Systeme durchführen, sobald sie ein- bzw. ausgeschaltet werden. Außer diesen Ausgaben müssen die Methoden `run` und `stop` nichts weiter tun.

b) Implementiere eine main-Methode, in welcher ein Array mit den vier oben genannten Systemen angelegt wird. Diese sollen anschließend alle gestartet werden. Danach sollen alle Systeme mit Schalter der Reihe nach deaktiviert werden.

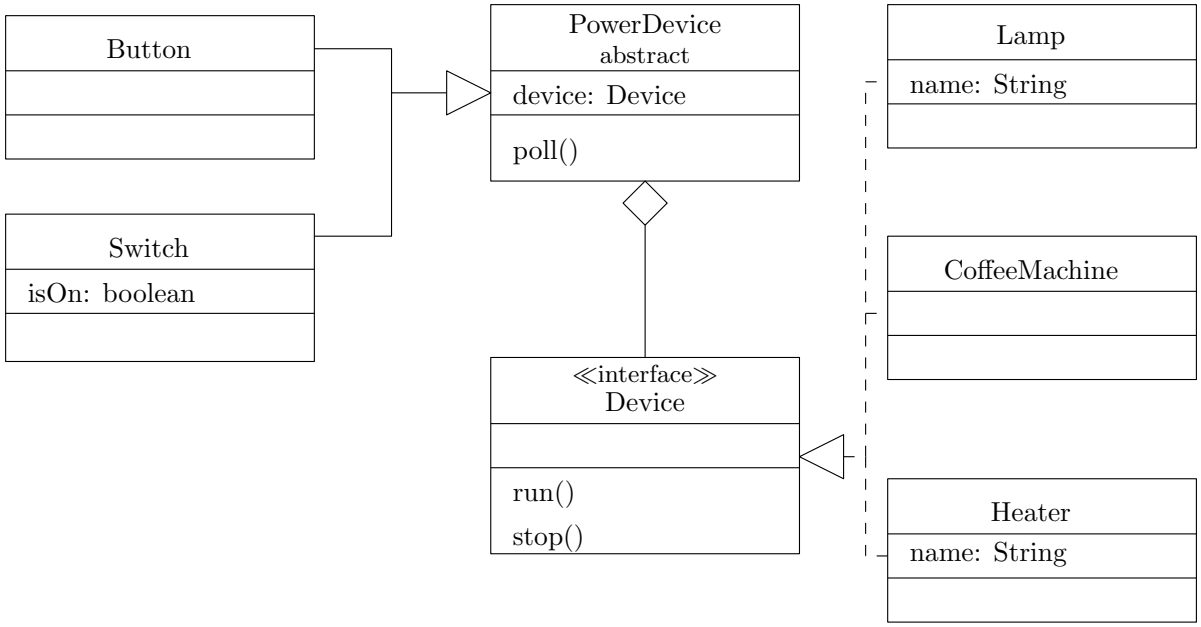


Abbildung 1: Klassendiagramm für das Smart-Home

Hausaufgabe 2 (Binäre Suchbäume): **(9 Punkte)**
Ein binärer Suchbaum (siehe Abbildung 2) ist im Gegensatz zu einer verketteten Liste eine geordnete Datenstruktur. Ein **Element** in einem binären Suchbaum besteht dabei aus drei Zeigern (`left`, `right` und `parent`) und einem Datensatz (`data`).

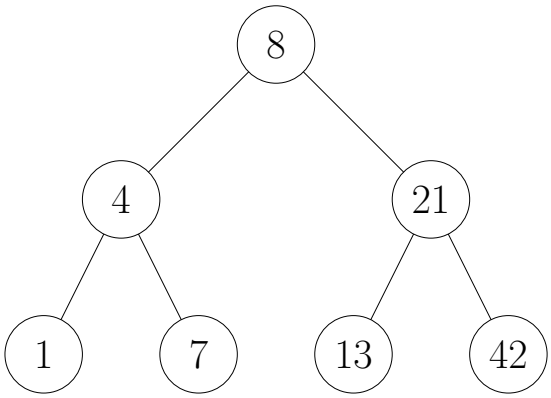
Der Datensatz eines Elements besteht aus zwei Werten: Dem Schlüssel (`key`) und dem eigentlichen Wert (`val`). Diese werden zusammengefasst in der Klasse `Pair`. Damit Inhalte eines Elements verglichen werden können, implementiert `Pair` die Schnittstelle `Comparable` von Java. Zwei Paare werden dann einfach über die jeweiligen keys verglichen.

Für ein Element e mit Datensatz d zeigen die drei Zeiger auf jeweils andere Elemente des binären Suchbaum:

- `left` zeigt auf ein Element, dessen Inhalt kleiner ist als d. Auch *linkes Kind* genannt.
- `right` zeigt auf ein Element, dessen Inhalt größer ist als d. Auch *rechtes Kind* genannt.
- `parent` zeigt auf ein Element e', sodass e entweder linkes oder rechtes Kind von e' ist. Auch *Vaterknoten* genannt.

Jeder dieser Zeiger kann `null` sein. Dabei gilt besonders:

- Besitzt ein Element keinen Vaterknoten (d.h. parent == null), ist dieses die *Wurzel* des gesamten Baumes. Alle anderen Elemente besitzen einen Vaterknoten.
- Jedes Element ist die Wurzel des eigenen Teilbaumes (enthält das Element selbst und alle seine Nachkommen).
- Für ein Element mit Schlüsselwert s besitzt jedes Element im Teilbaum des linken Kindes einen kleineren Schlüsselwert als s. Analog sind im rechten Teilbaum alle Schlüsselwerte größer als s.
- Besitzt ein Element weder ein rechtes, noch ein linkes Kind, so heißt dieses Element *Blatt*.
- Die maximale Anzahl an Schritten von der Wurzel zu einem Blatt ist die *Höhe* des Baumes.



inorder Ausgabe: 1, 4, 7, 8, 13, 21, 42
preorder Ausgabe: 8, 4, 1, 7, 21, 13, 42
postorder Ausgabe: 1, 7, 4, 13, 42, 21, 8

Abbildung 2: Ein binärer Suchbaum. Für jedes Element entspricht hier der Schlüssel dem val-Wert. Rechts die drei möglichen Ausgaben.

Für binäre Suchbäume wollen wir nun eine **Utility Class** entwickeln, welche folgende Methoden bereitstellt:

`Element createBinaryTree(Pair[] data, int left, int right)` Konstruiert einen binären Suchbaum mit minimaler Höhe aus einem sortierten Array data auf dem Index-Bereich `left` (inklusive) bis `right` (exklusive). Dies geschieht wie folgt:

Erstelle aus dem in der Mitte stehenden Wert x von data (Index m = (left + right)/2 abgerundet) ein Element e_x des binären Suchbaumes. Wiederhole das Vorgehen rekursiv auf dem Bereich l bis m sowie den Bereich m+1 bis r. Dadurch erhält man zwei Elemente e_l und e_r des binären Suchbaumen, welche dem linken bzw. rechten Kind von x entsprechen. Setze zum Schluss die parent-Zeiger von e_l und e_r auf e_x.

Für den Fall, dass l == r gilt, soll ein null-Pointer zurückgegeben werden.

`void inorder(Element root)` Durchläuft den binären Suchbaum mit Wurzel root. Dabei wird zunächst rekursiv auf dem linken Teilbaum fortgefahren. Ist dieser abgearbeitet, wird der val-Wert von root ausgegeben und danach rekursiv auf dem rechten Teilbaum fortgefahren.

`void preorder(Element root)` Durchläuft den binären Suchbaum mit Wurzel root. Dabei wird zunächst der val-Wert von root ausgegeben und danach rekursiv auf dem linken und dann auf dem rechten Teilbaum der Vorgang wiederholt.

`void postorder(Element root)` Durchläuft den binären Suchbaum mit Wurzel root. Dabei wird zunächst rekursiv auf dem linken und dann auf dem rechten Teilbaum fortgefahren. Sind beide Teilbäume abgearbeitet, wird der val-Wert von root ausgegeben.

`int getHeight(Element root)` Gibt die Höhe des Suchbaums zurück. Blätter besitzen die Höhe 1. Für jedes andere Element entspricht die Höhe 1 plus dem Maximum der Höhen seiner Kinder (null Element besitzt Höhe 0).

a) Implementiere die Klassen **Element**, **Pair** und **BinaryTreeUtils** in dem Paket **binary-Tree**. Achte dabei auf korrekte Modifizierer der Klassen und Methoden. Zur Übersicht betrachte das Klassendiagramm aus Abbildung 3. Konstruktoren, Get- und Set-Methoden sind nach Bedarf eigenständig zu ergänzen.

b) Entwirf eine main-Methode, in welcher ein binärer Baum erzeugt wird, der die inorder-Ausgabe „ein binaerer Baum ist eine dynamische Datenstruktur“ besitzt. Dabei soll jedes Element im Suchbaum nur ein Wort repräsentieren.

Erstelle dazu zunächst ein `Pair`-Array, welches nicht korrekt nach den keys sortiert ist. Sortiere das Arrays dann in der main-Methode und lass danach den Suchbaum erzeugen.

c) Führe in der main-Methode jede Ausgabe-Methode (`inrOder`, `preOrder`, `postOrder`) für den erstellten Baum einmal aus. Zum Schluss soll die Höhe des Suchbaums mit `getHeight` bestimmt und ausgegeben werden.

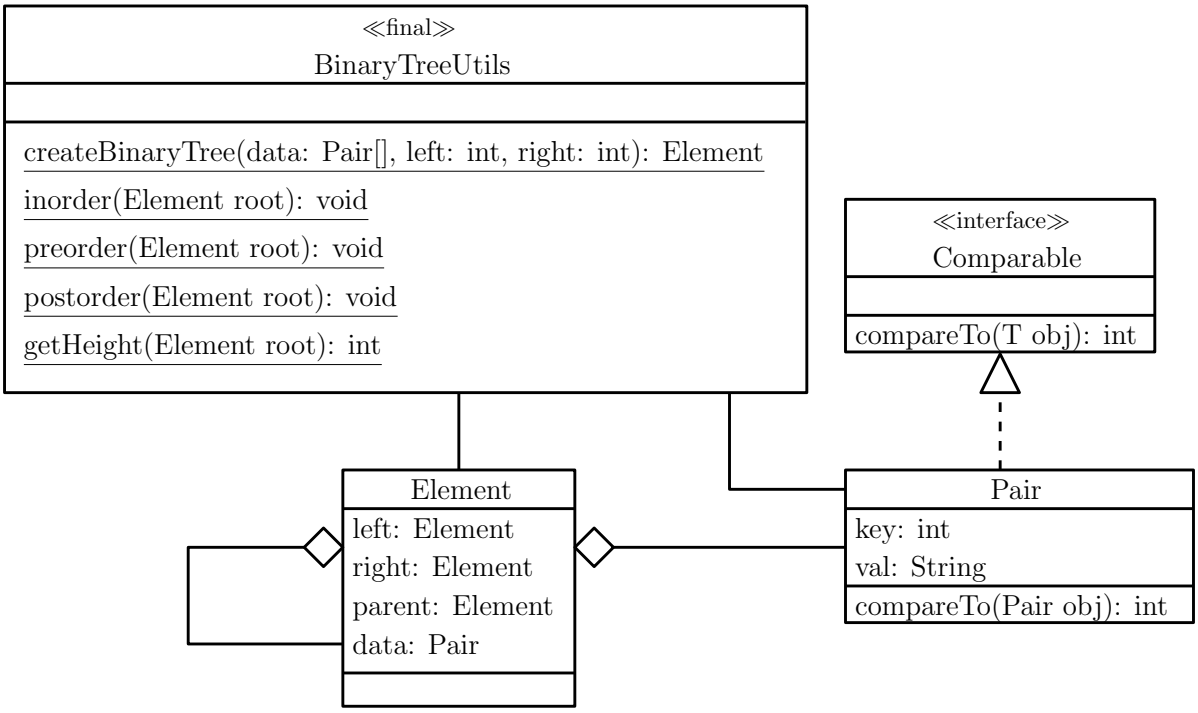


Abbildung 3: Klassendiagramm für einen binären Suchbaum. Das Comparable-Interface existiert bereits in Java.