Informatik II - Übung 7

Pascal Schärli

othello@pascscha.ch

06.11.2020

ETH zürich

Nachbesprechung

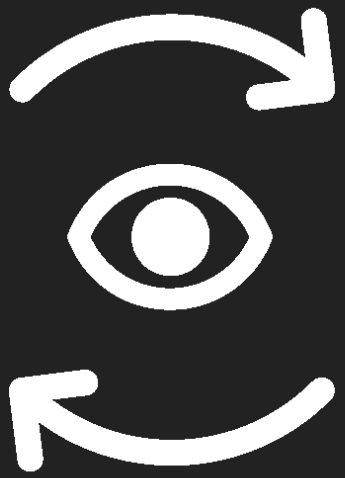
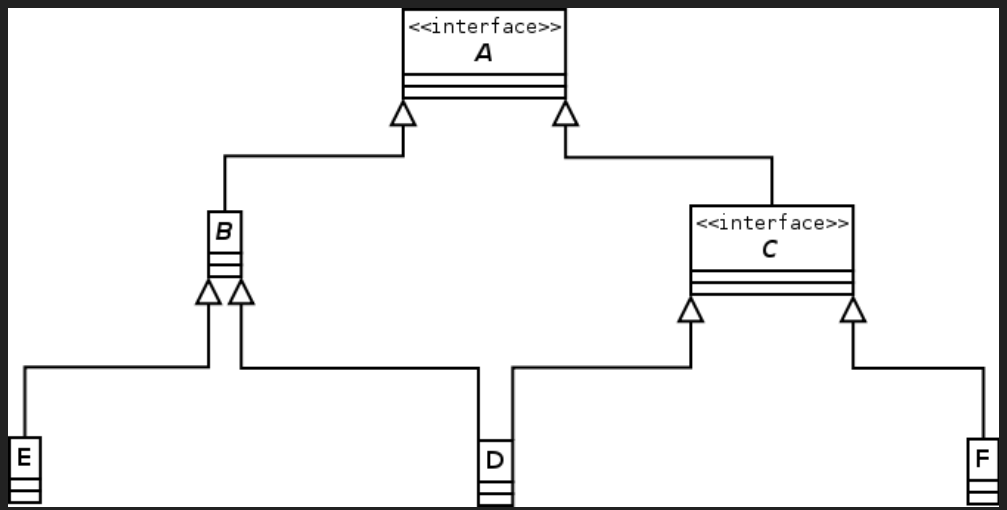


Bild: look again by corpus delicti from the Noun Project

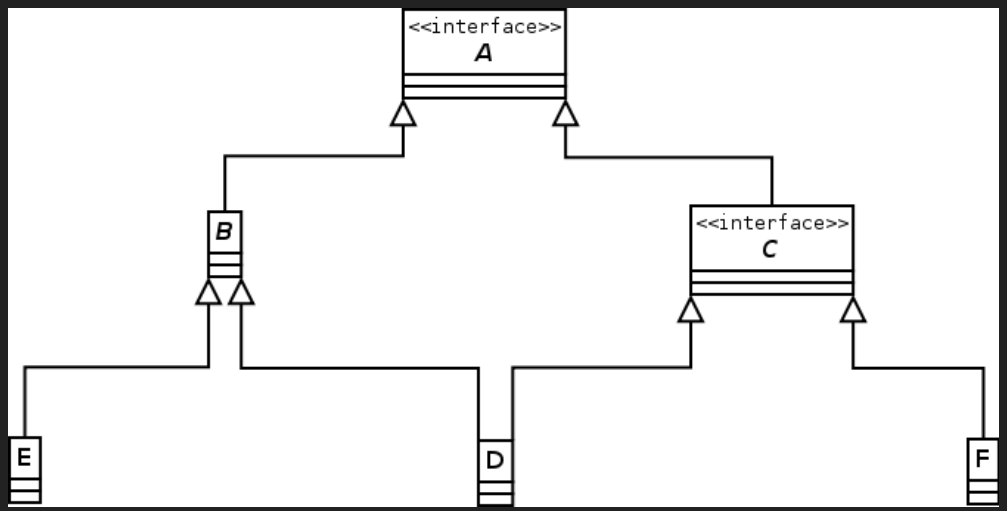
Klassen




```
1 D d = new D();
2 C c = d;
3 //E e = d;
```

Eine Zuweisung ist dann möglich ist, falls die neue Klasse (links) weiter oben im selben Ast wie die bereits existierende Klasse (rechts) ist.

Klassen




```
1 B b = new D();
2 C c = (C) b;
3 //E e = (E) b;
```

Beim Casten kommt es also nicht darauf an, was Java denkt, dass das Objekt ist, sondern als was es erstellt wurde. Solange alle nötigen Informationen vorhanden sind, ist ein Cast möglich.

Polymorphie

```java
package u6a3;

/**
 * abstract class for geometric objects
 */
public abstract class GeometricObject implements Comparable {
    public abstract int area();

    public boolean smallerThan(Comparable rhs) {
        GeometricObject other = (GeometricObject) rhs;
        return this.area() < other.area();
    }
}
```

Polymorphie

```
1  private GenericList insertSorted(GenericList list, Object value) {
2      if (list == null) return new GenericList(value, null);
3
4      Comparable lhs = (Comparable) value;
5      Comparable rhs = (Comparable) list.value;
6      if (lhs.smallerThan(rhs)) return new GenericList(value, list);
7
8      list.next = insertSorted(list.next, value);
9      return list;
10 }
11
12 public GenericList sort(GenericList list) {
13     if (list == null) return null;
14     return insertSorted(sort(list.next), list.value);
15 }
```

Best of Vorlesung

Generics

Eine ArrayList kann beliebige Typen speichern:

```
1 ArrayList myList = new ArrayList();
2
3 myList.add(42);
4 myList.add("I <3 Info II");
5
6 int i = (Integer)myList.get(0);
7 String s = (String)myList.get(1);
```

- Flexibel, aber Casts sind gefährlich
- Falls man nicht mehr weiss, was wo in der Liste ist, kann es zu Chaos führen
- $\rightarrow$ Generics

Generics

Man kann einer ArrayList sagen, dass sie nur einen bestimmten Typ speichern darf:

```
1 ArrayList<Integer> myList = new ArrayList<Integer>();
2 myList.add(42);
3 int i = myList.get(0);
4 myList.add("Hello World");
```

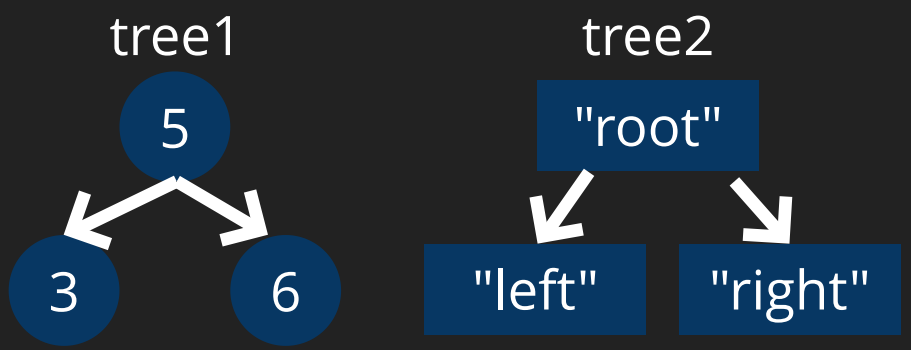
✕ Compile-Error

- Vorteile:
  - Typsicherheit
  - keine Casts nötig
- Nachteile:
  - Weniger Flexibel

Generics - Beispiel

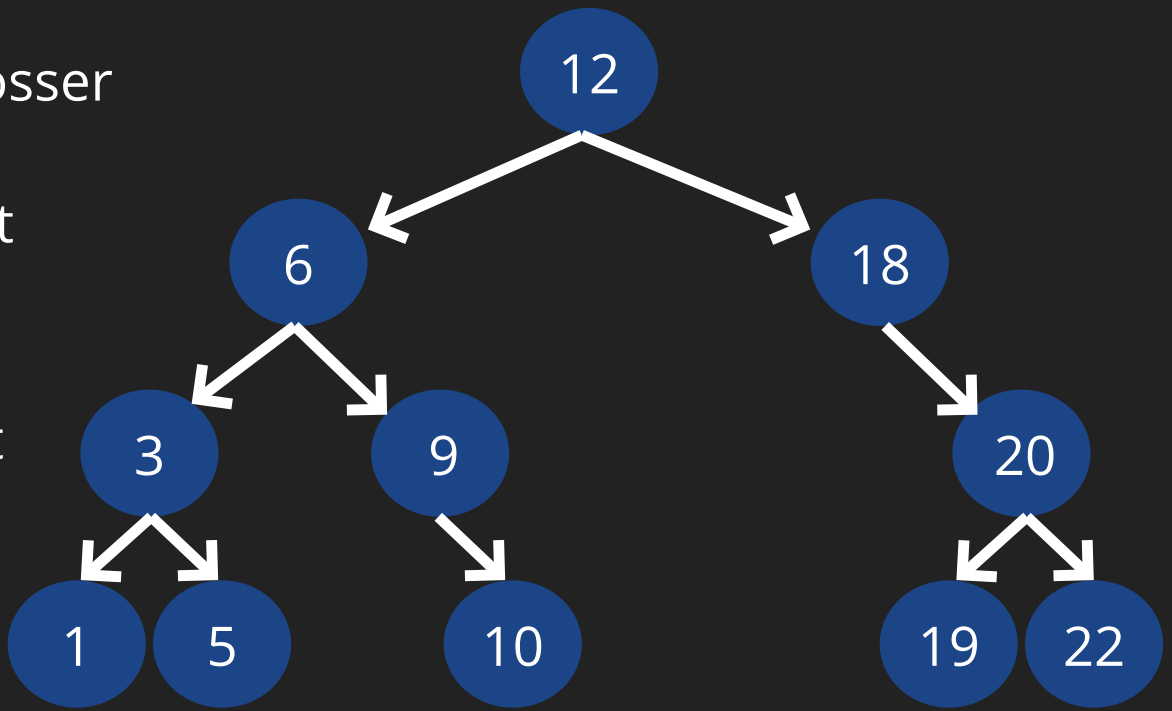
```java
public class Tree<T> {

  private T value;
  Tree right;
  Tree left;

  public Tree(T value) {
    right = null;
    left = null;
    this.value = value;
  }

  public T getValue() {
        return value;
  }

  public void setValue(T v) {
        value = v;
  }
}
```

```java
Tree<Integer> tree1 = new Tree<Integer>(5);
tree1.right = new Tree<Integer>(3);
tree1.left = new Tree<Integer>(6);

Tree<String> tree2 = new Tree<String>("root");
tree2.right = new Tree<String>("left");
tree2.left = new Tree<String>("right");
```



Binäre Suchbäume

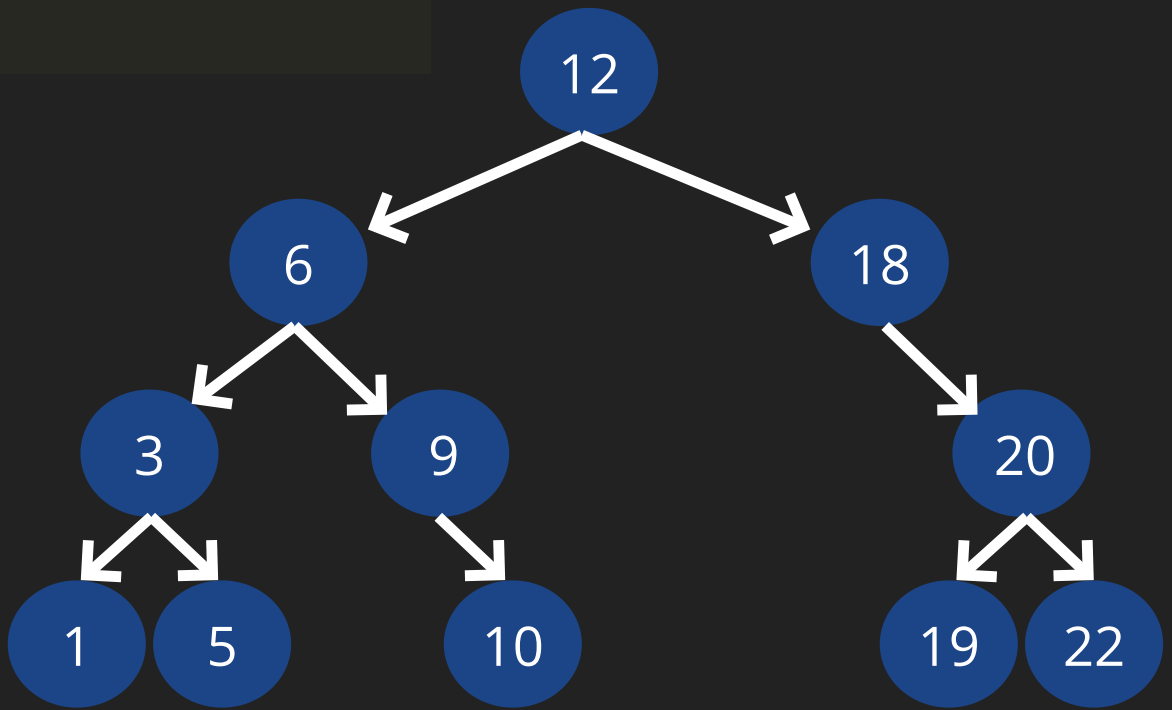
- Alle Elemente im linken Ast sind kleiner
- Alle Elemente im rechten Ast sind grösser
- $\rightarrow$ kleinstes Element ganz links
- $\rightarrow$ grösstes Element ganz rechts



Binäre Suchbäume - Element Finden

```
1 Element Finden:
2   Falls die Wurzel das gesuchte Element ist
3     -> gefunden!
4
5   Falls der geschte Wert kleiner als die Wurzel ist
6     -> Links weitersuchen
7
8   Falls der geschte Wert grösser als die Wurzel ist
9     -> Rechts weitersuchen
```

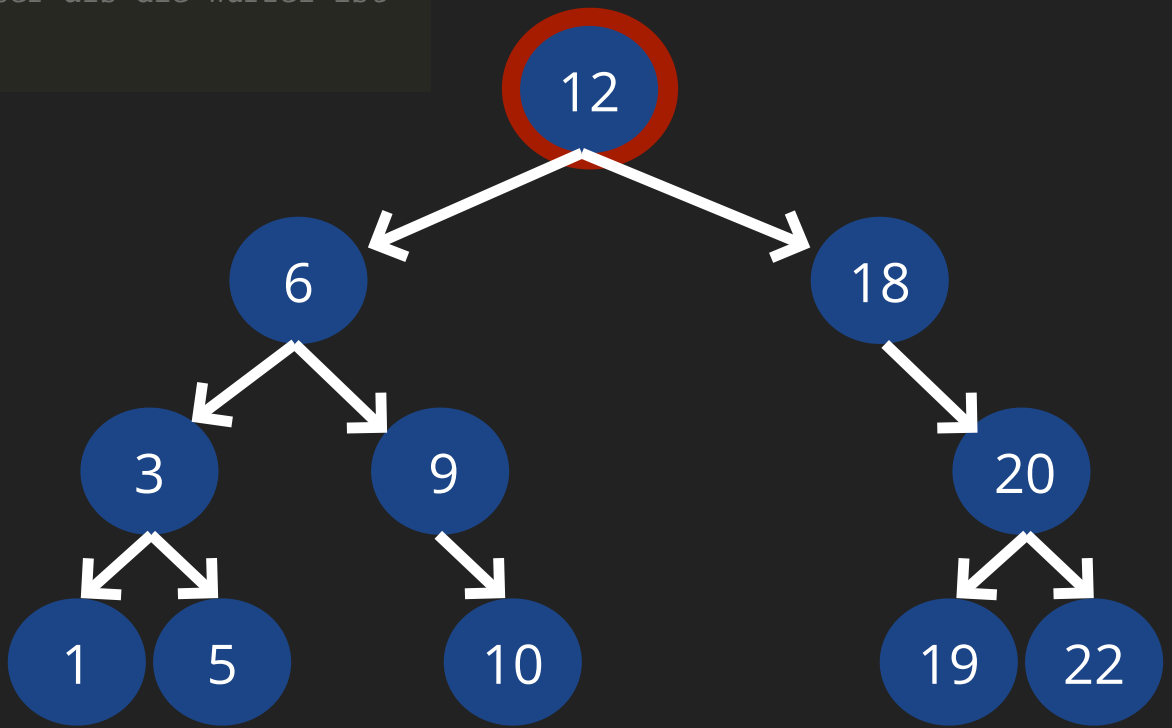
Gesucht: 9



Binäre Suchbäume - Element Finden

```
1 Element Finden:
2   Falls die Wurzel das gesuchte Element ist
3     -> gefunden!
4
5   Falls der geschte Wert kleiner als die Wurzel ist
6     -> Links weitersuchen
7
8   Falls der geschte Wert grösser als die Wurzel ist
9     -> Rechts weitersuchen
```

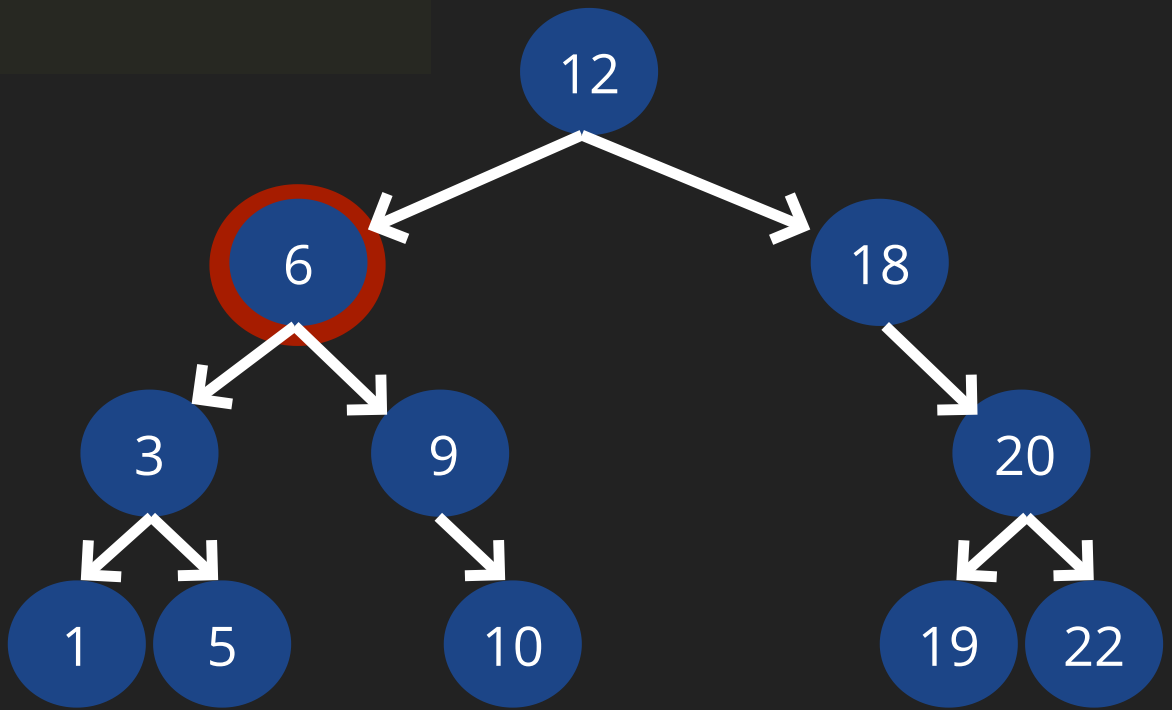
Gesucht: 9



Binäre Suchbäume - Element Finden

```
1 Element Finden:
2   Falls die Wurzel das gesuchte Element ist
3     -> gefunden!
4
5   Falls der geschte Wert kleiner als die Wurzel ist
6     -> Links weitersuchen
7
8   Falls der geschte Wert grösser als die Wurzel ist
9     -> Rechts weitersuchen
```

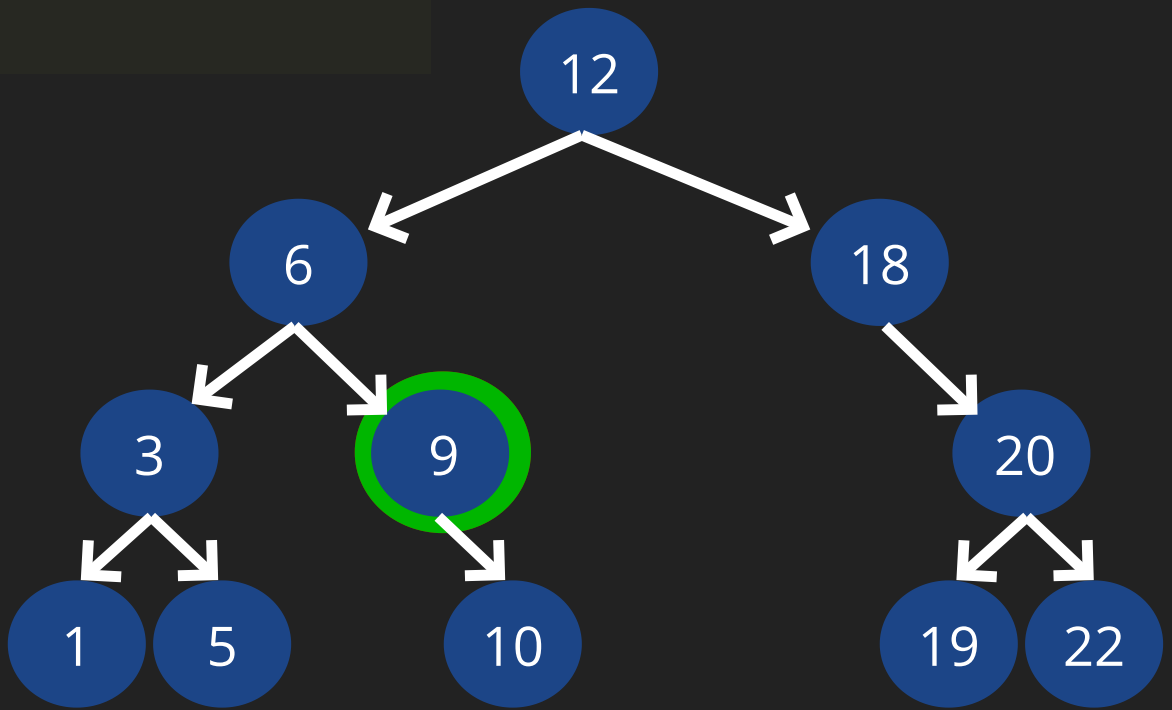
Gesucht: 9



Binäre Suchbäume - Element Finden

```
1 Element Finden:
2   Falls die Wurzel das gesuchte Element ist
3     -> gefunden!
4
5   Falls der geschte Wert kleiner als die Wurzel ist
6     -> Links weitersuchen
7
8   Falls der geschte Wert grösser als die Wurzel ist
9     -> Rechts weitersuchen
```

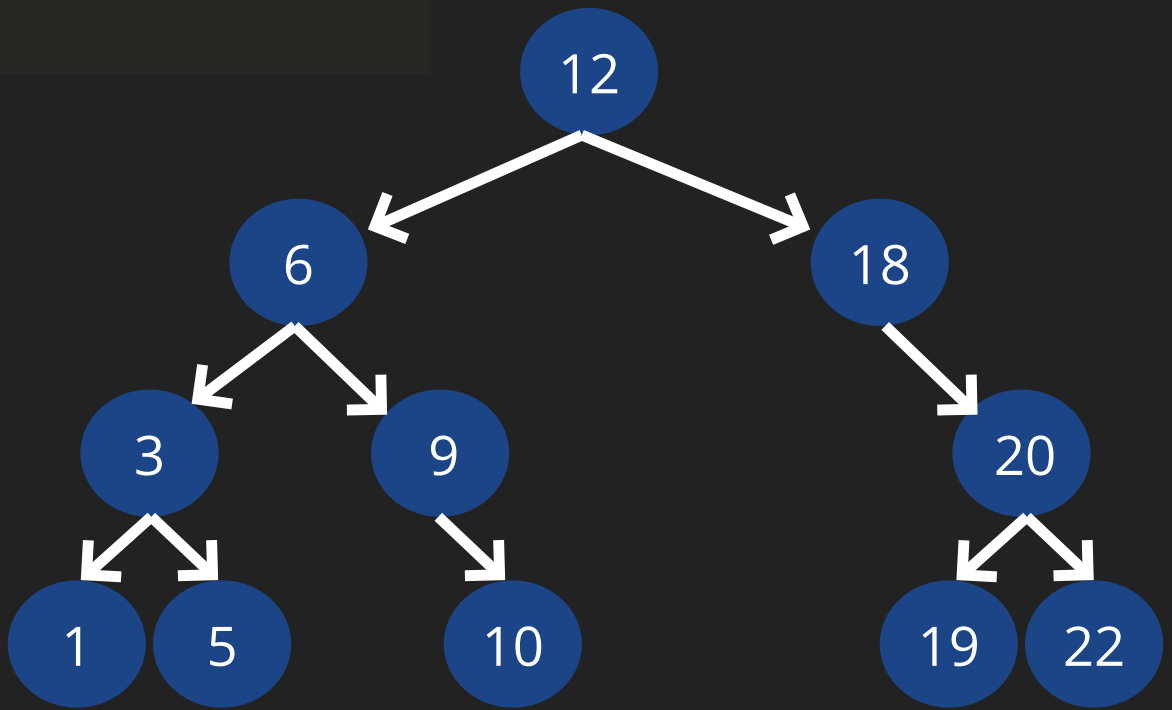
Gesucht: 9



Binäre Suchbäume - Element Finden

```
1 Element Finden:
2   Falls die Wurzel das gesuchte Element ist
3     -> gefunden!
4
5   Falls der geschte Wert kleiner als die Wurzel ist
6     -> Links weitersuchen
7
8   Falls der geschte Wert grösser als die Wurzel ist
9     -> Rechts weitersuchen
```

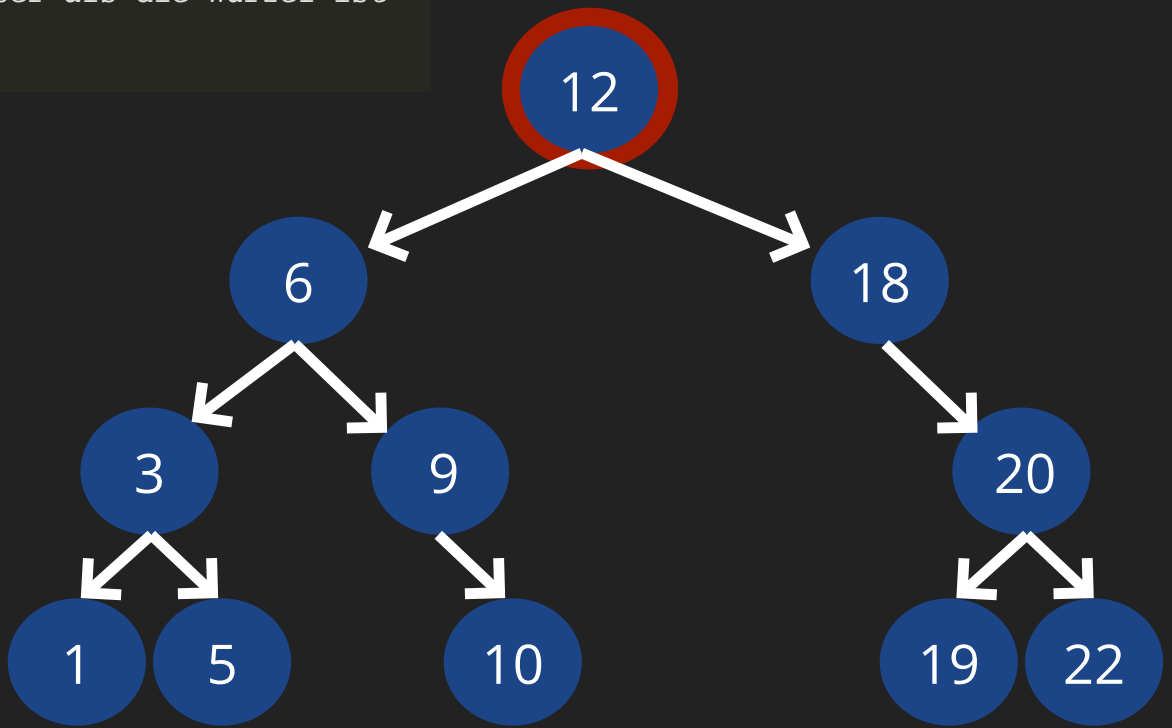
Gesucht: 15



Binäre Suchbäume - Element Finden

```
1 Element Finden:
2   Falls die Wurzel das gesuchte Element ist
3     -> gefunden!
4
5   Falls der geschte Wert kleiner als die Wurzel ist
6     -> Links weitersuchen
7
8   Falls der geschte Wert grösser als die Wurzel ist
9     -> Rechts weitersuchen
```

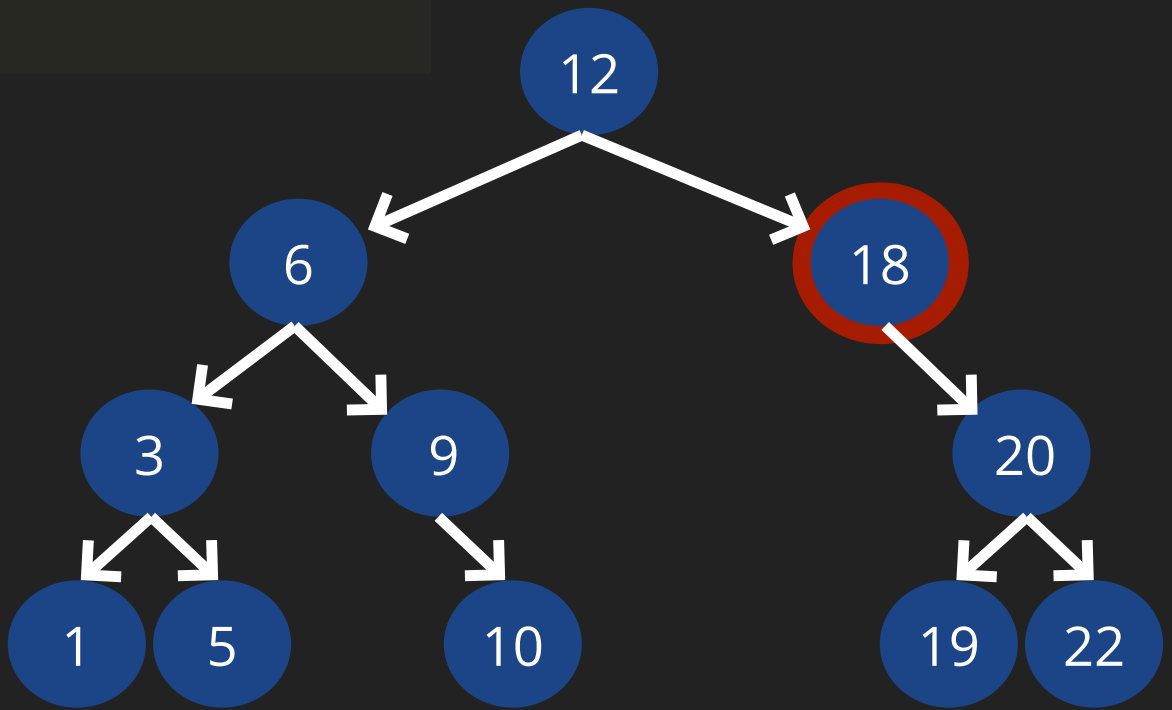
Gesucht: 15



Binäre Suchbäume - Element Finden

```
1 Element Finden:
2   Falls die Wurzel das gesuchte Element ist
3     -> gefunden!
4
5   Falls der geschte Wert kleiner als die Wurzel ist
6     -> Links weitersuchen
7
8   Falls der geschte Wert grösser als die Wurzel ist
9     -> Rechts weitersuchen
```

Gesucht: 15

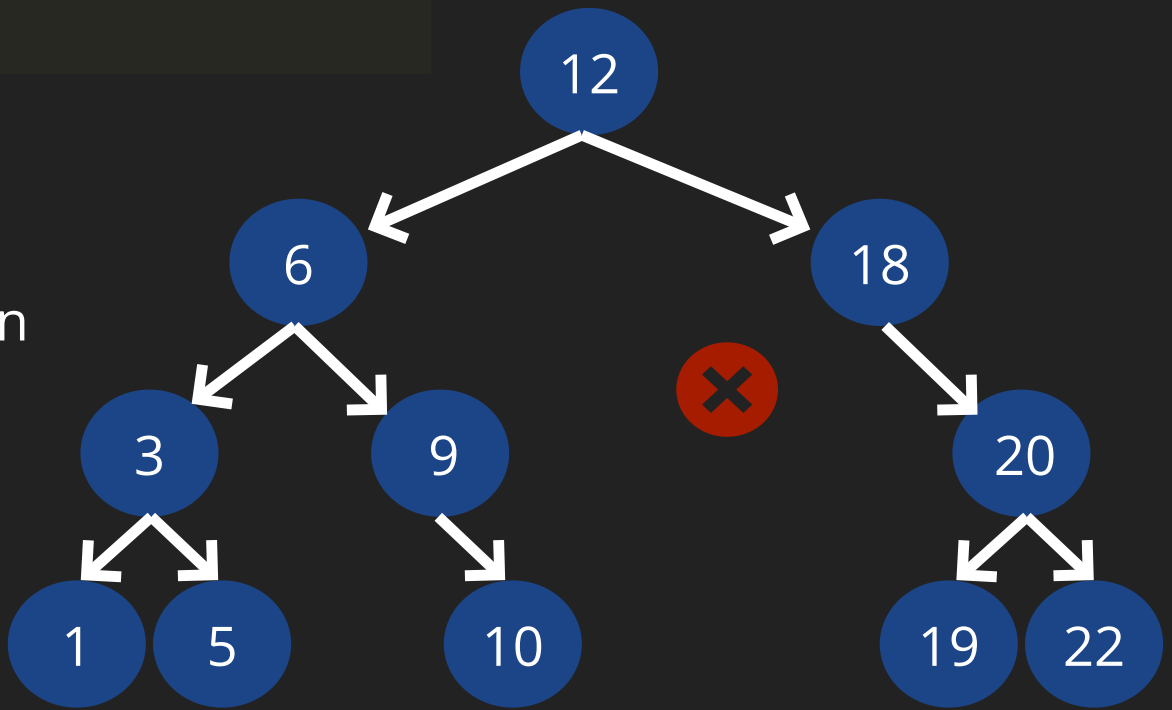


Binäre Suchbäume - Element Finden

```
1 Element Finden:
2   Falls die Wurzel das gesuchte Element ist
3     -> gefunden!
4
5   Falls der geschte Wert kleiner als die Wurzel ist
6     -> Links weitersuchen
7
8   Falls der geschte Wert grösser als die Wurzel ist
9     -> Rechts weitersuchen
```

Gesucht: 15

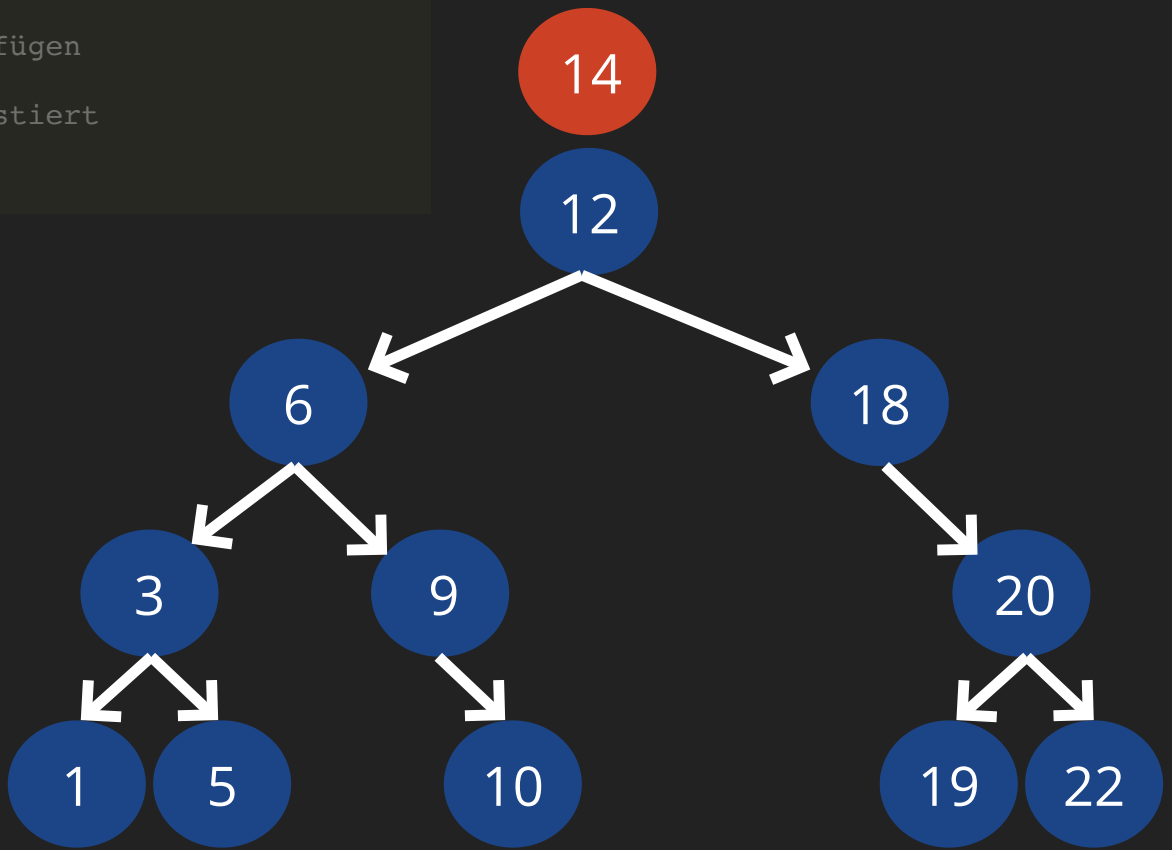
→ Nicht vorhanden



Binäre Suchbäume - Element Einfügen

```
1 Element Finden:
2   Falls grösser
3     -> im rechten Teilbaum einfügen
4
5   Falls kleiner
6     -> im linken Teilbaum einfügen
7
8   Falls Knoten noch nicht existiert
9     -> fertig
```

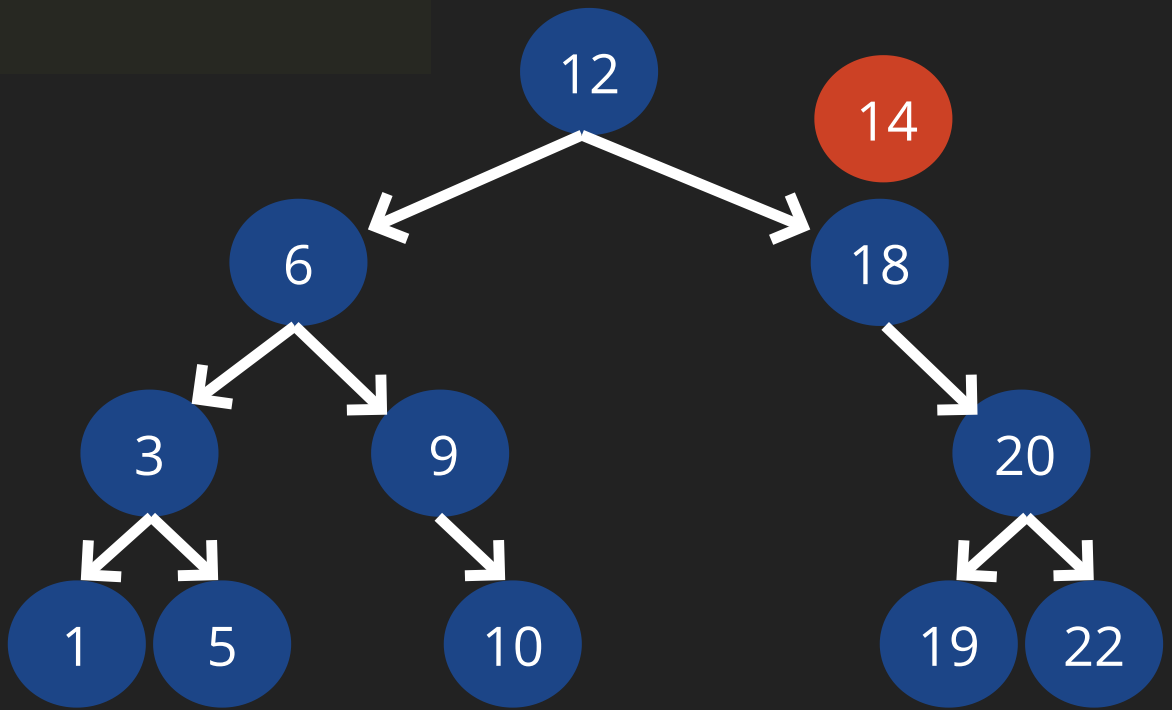
Einfügen: 14



Binäre Suchbäume - Element Einfügen

```
1 Element Finden:
2   Falls grösser
3     -> im rechten Teilbaum einfügen
4
5   Falls kleiner
6     -> im linken Teilbaum einfügen
7
8   Falls Knoten noch nicht existiert
9     -> fertig
```

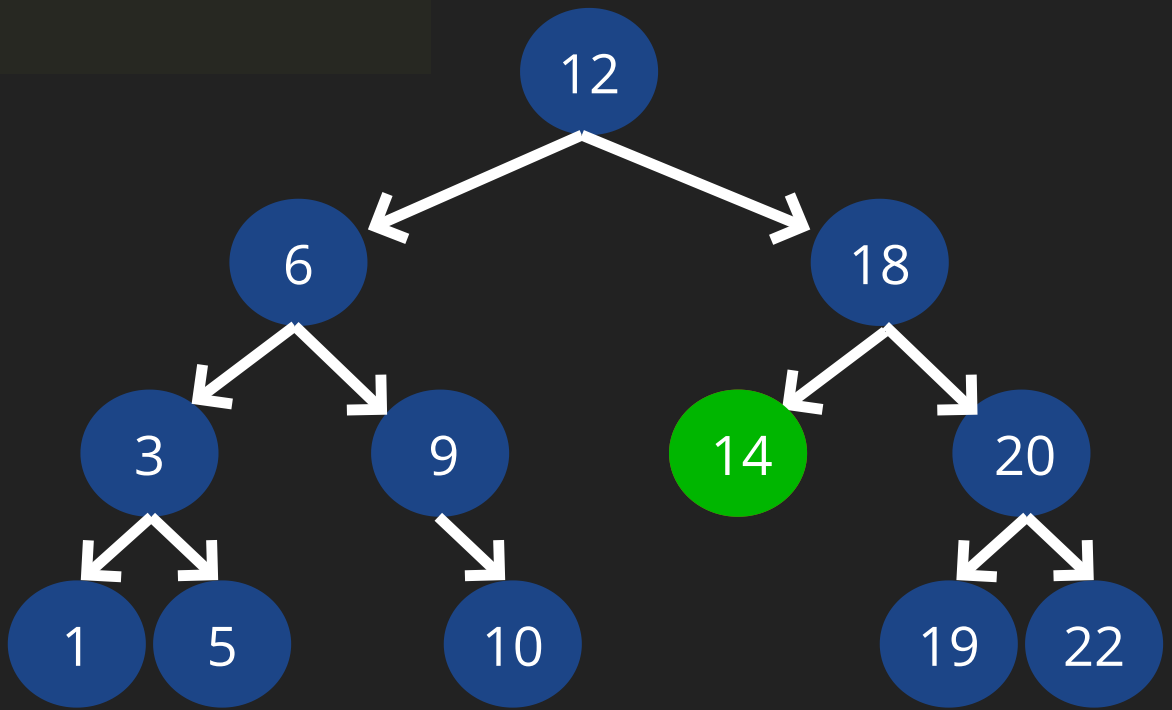
Einfügen: 14



Binäre Suchbäume - Element Einfügen

```
1 Element Finden:
2   Falls grösser
3     -> im rechten Teilbaum einfügen
4
5   Falls kleiner
6     -> im linken Teilbaum einfügen
7
8   Falls Knoten noch nicht existiert
9     -> fertig
```

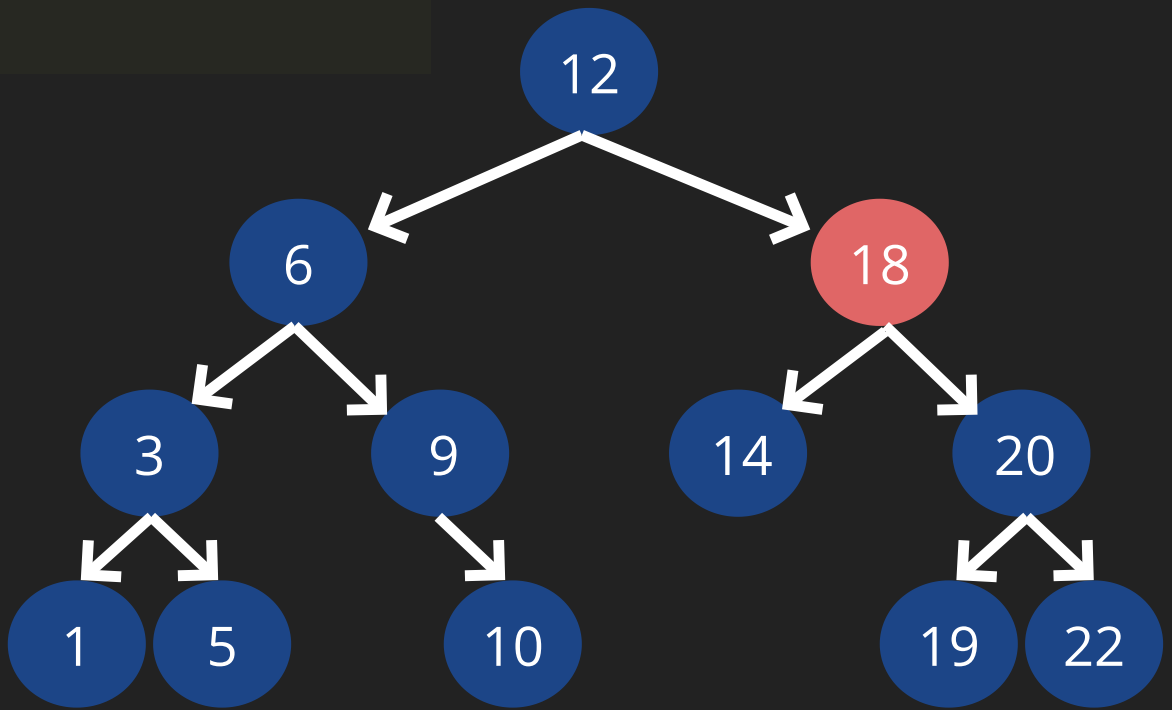
Einfügen: 14



Binäre Suchbäume - Element Entfernen

```
1 Element entfernen - Methode 1
2   Ersetzen durch kleinstes Element
3   im rechten Teilbaum
4
5 Element entfernen - Methode 2
6   Ersetzen durch grösstes Element
7   im linken Teilbaum
```

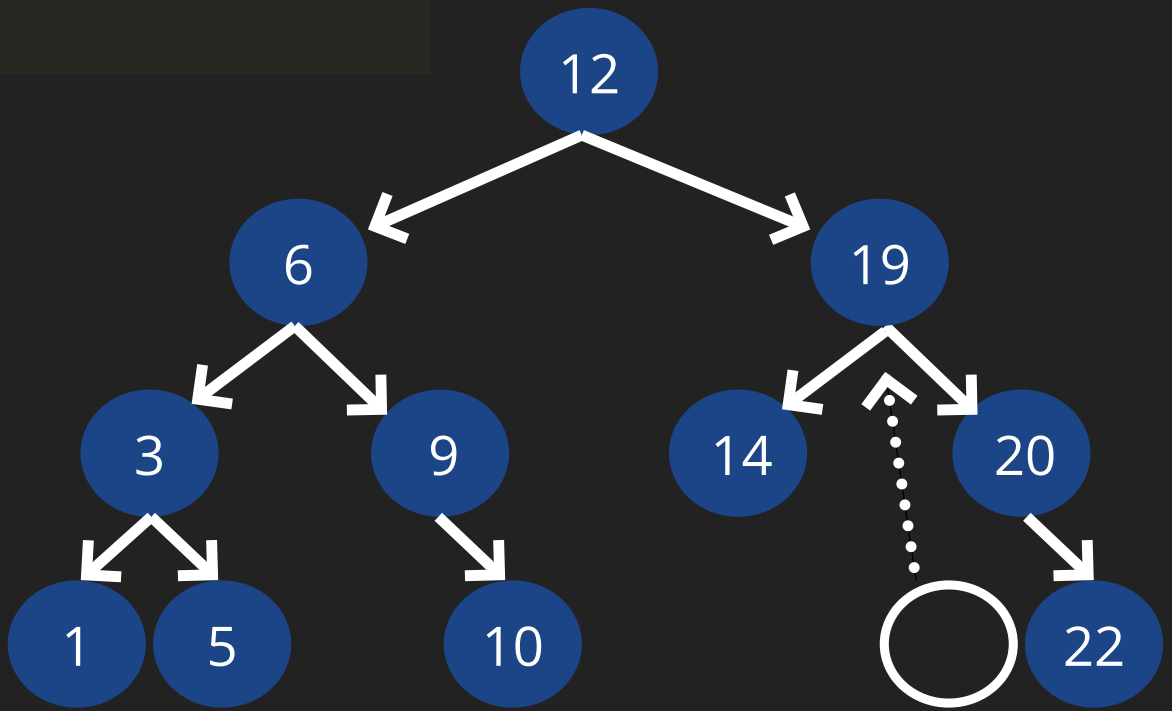
Entfernen: 18



Binäre Suchbäume - Element Entfernen

```
1 Element entfernen - Methode 1
2   Ersetzen durch kleinstes Element
3   im rechten Teilbaum
4
5 Element entfernen - Methode 2
6   Ersetzen durch grösstes Element
7   im linken Teilbaum
```

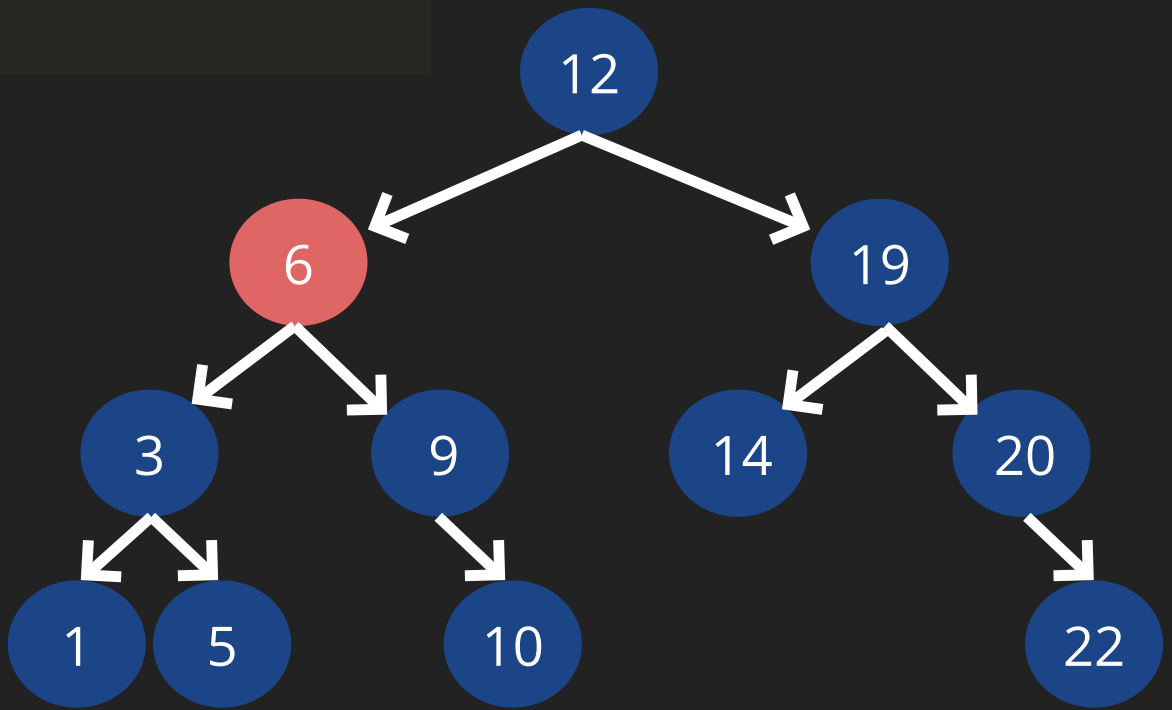
Entfernen: 18



Binäre Suchbäume - Element Entfernen

```
1 Element entfernen - Methode 1
2   Ersetzen durch kleinstes Element
3   im rechten Teilbaum
4
5 Element entfernen - Methode 2
6   Ersetzen durch grösstes Element
7   im linken Teilbaum
```

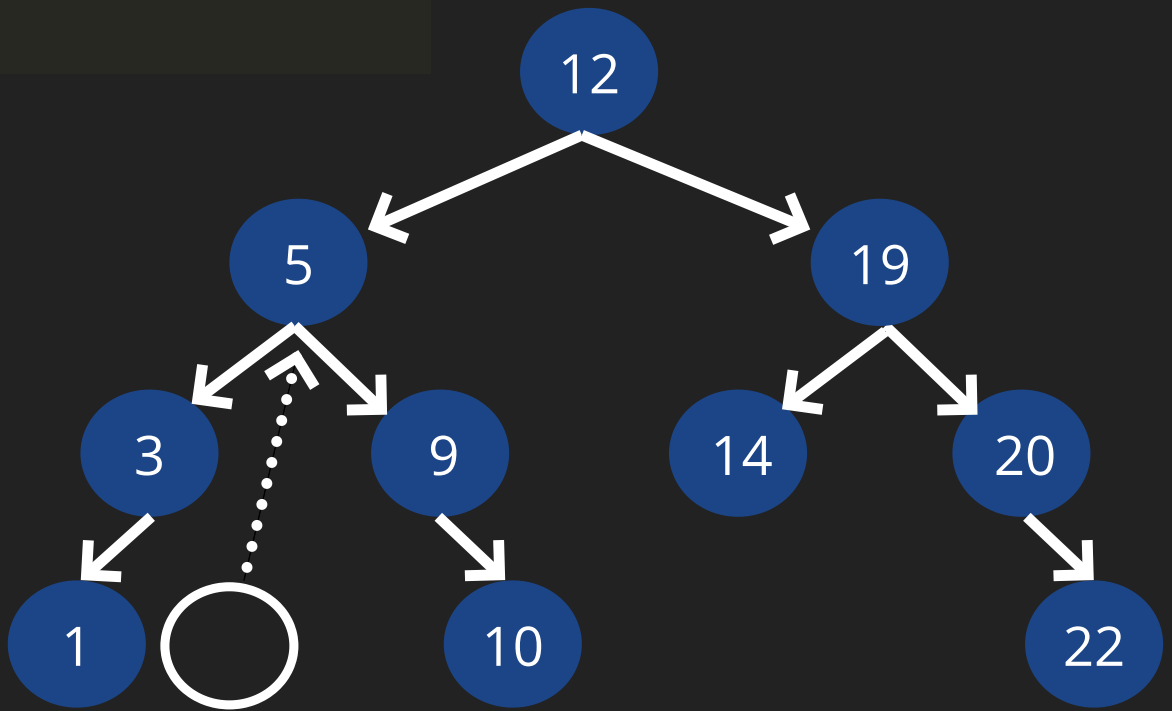
Entfernen: 6



Binäre Suchbäume - Element Entfernen

```
1 Element entfernen - Methode 1
2   Ersetzen durch kleinstes Element
3   im rechten Teilbaum
4
5 Element entfernen - Methode 2
6   Ersetzen durch grösstes Element
7   im linken Teilbaum
```

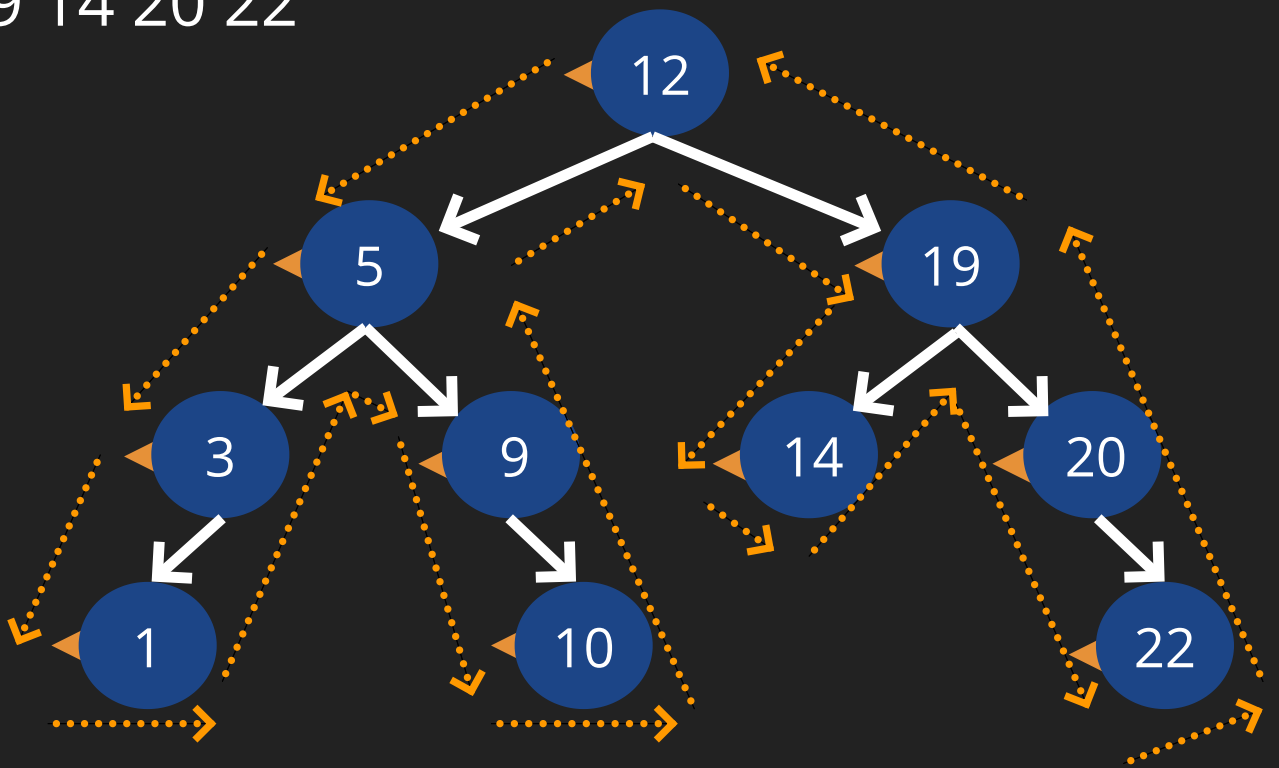
Entfernen: 6



Binäre Suchbäume - Preorder

```
1 Gib den Wert vom Knoten aus
2 Gib den Linken Teilbaum in Pre-Order aus
3 Gib den Rechte Teilbaum in Pre-Order aus
```

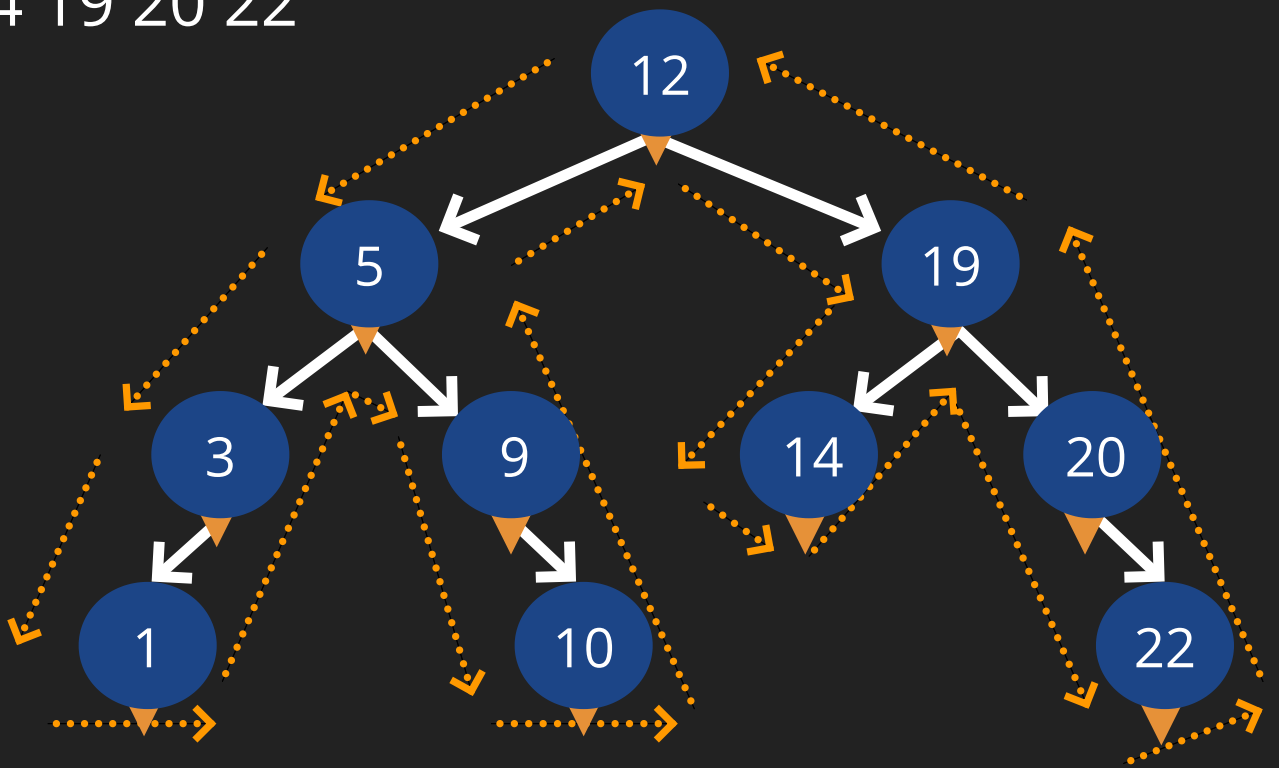
12 5 3 1 9 10 19 14 20 22



Binäre Suchbäume - Inorder

```
1 Gib den Linken Teilbaum in In-Order aus
2 Gib den Wert vom Knoten aus
3 Gib den Rechte Teilbaum in In-Order aus
```

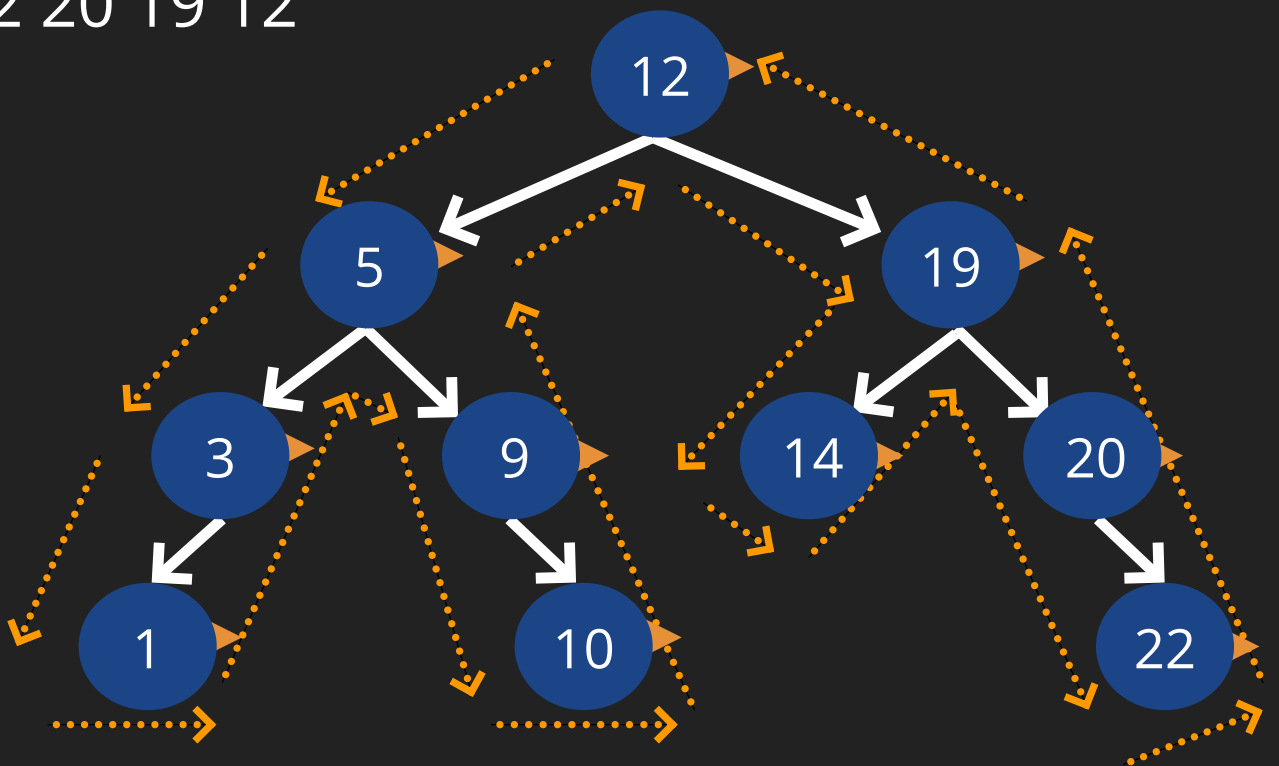
1 3 5 9 10 12 14 19 20 22



Binäre Suchbäume - Postorder

```
1 Gib den Linken Teilbaum in Post-Order aus
2 Gib den Rechte Teilbaum in Post-Order aus
3 Gib den Wert vom Knoten aus
```

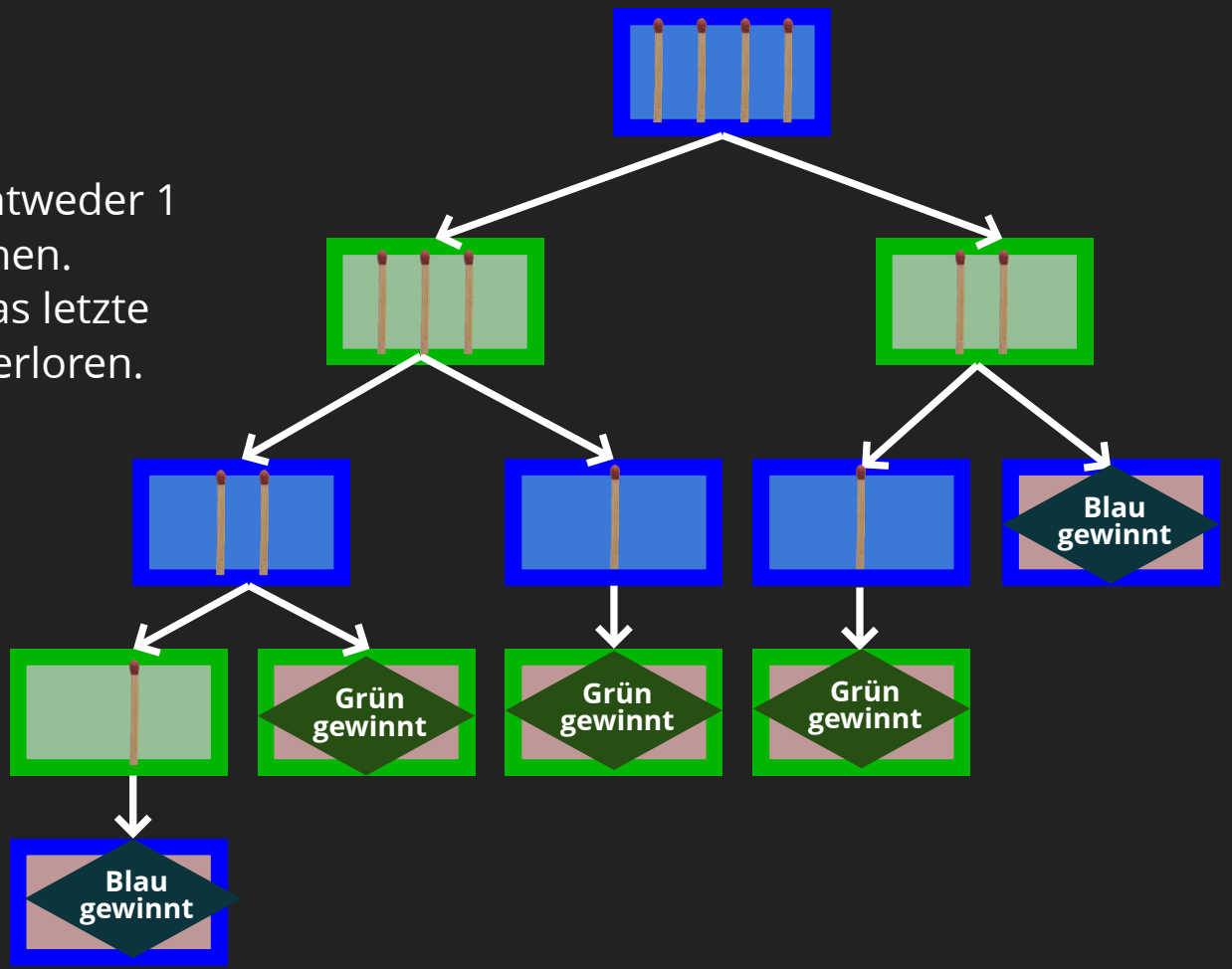
1 2 10 9 5 14 22 20 19 12



Spielbäume

Spielregeln:

- Die Spieler können abwechslungsweise entweder 1 oder 2 Zündhölzer ziehen.
- Der Spieler, welcher das letzte Zündholz nimmt hat verloren.

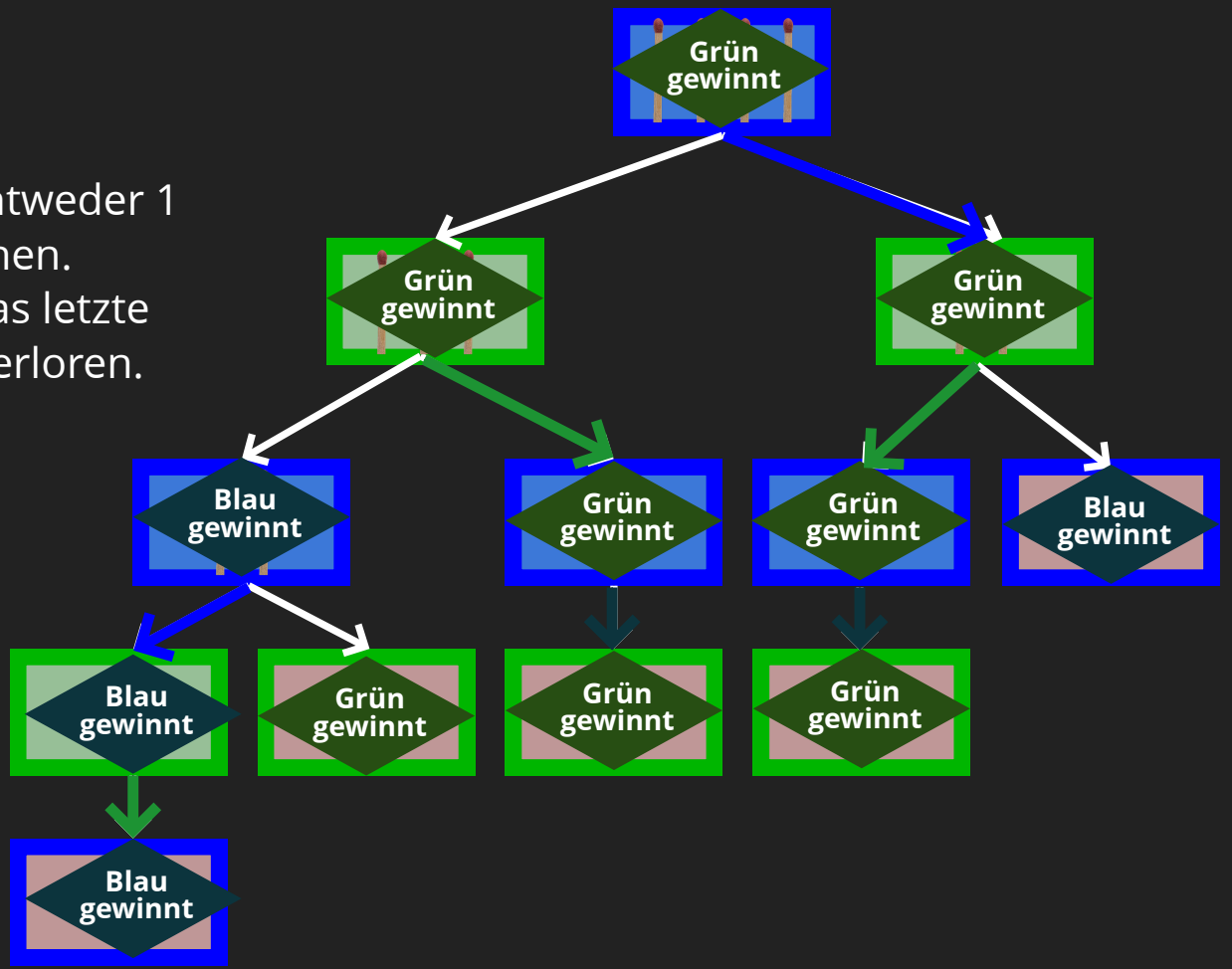


Spielbäume

Spielregeln:

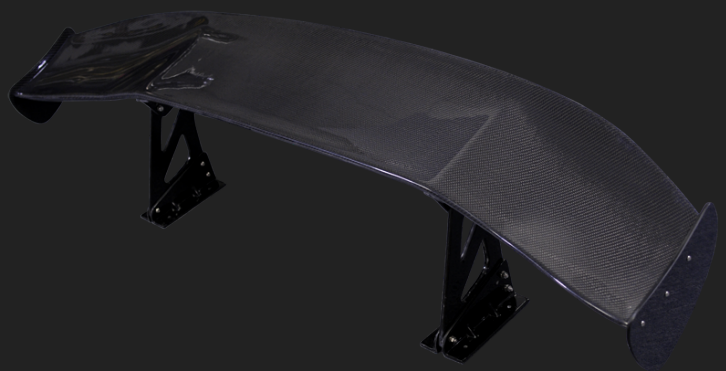
- Die Spieler können abwechslungsweise entweder 1 oder 2 Zündhölzer ziehen.
- Der Spieler, welcher das letzte Zündholz nimmt hat verloren.

⇒ Grün gewinnt!




K!

Vorbesprechung

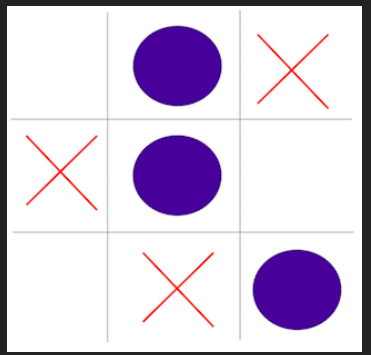


Array-Listen und Generics

- Gegeben: Liste von Übungsgruppen
  - groups = ArrayList $\leftarrow \{group1, group2, \dots\}$
    - group1 = ArrayList $\leftarrow \{student1, student2, \dots\}$
    - group2 = ArrayList $\leftarrow \{student21, student22, \dots\}$
    - $\dots$
- Jeder Student hat eine gewisse Punktzahl erreicht
  - myStudent.getPoints()
- Aufgabe: ArrayList aller Studente, welche eine gewisse Punktzahl erreicht haben
  1. filterRaw $\rightarrow$ Filter mit ArrayList als raw type implementieren
     $\rightarrow$ Viel Casten
  2. filterGeneric $\rightarrow$ Filter mit ArrayList als generic type implementieren
     $\rightarrow$ ArrayList<ArrayList<Student>>

Tic-Tac-Toe-Spielbaum

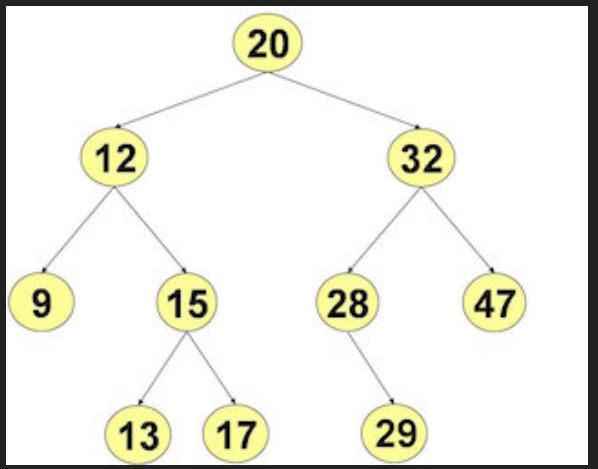
X ist am Zug - wer gewinnt?



- Spielbaum aufzeichnen
- Resultate von unten nach oben eintragen (1 $\rightarrow$ Sieg, 0 $\rightarrow$ Unentschieden, -1 $\rightarrow$ Niederlage)
- Wie wird das Spiel ausgehen, wenn beide perfekt spielen?
- Analog zu unserem Streichholz-Beispiel

Binäre Suchbäume

- Entfernt nacheinander die Elemente 15, 12 und 20
- Verwendet Strategie "Ersetzen durch kleinstes Element des rechten Teilbaums"
- Zeichnet den resultierenden Baum nach jedem Entfernen



Binäre Suchbäume 2

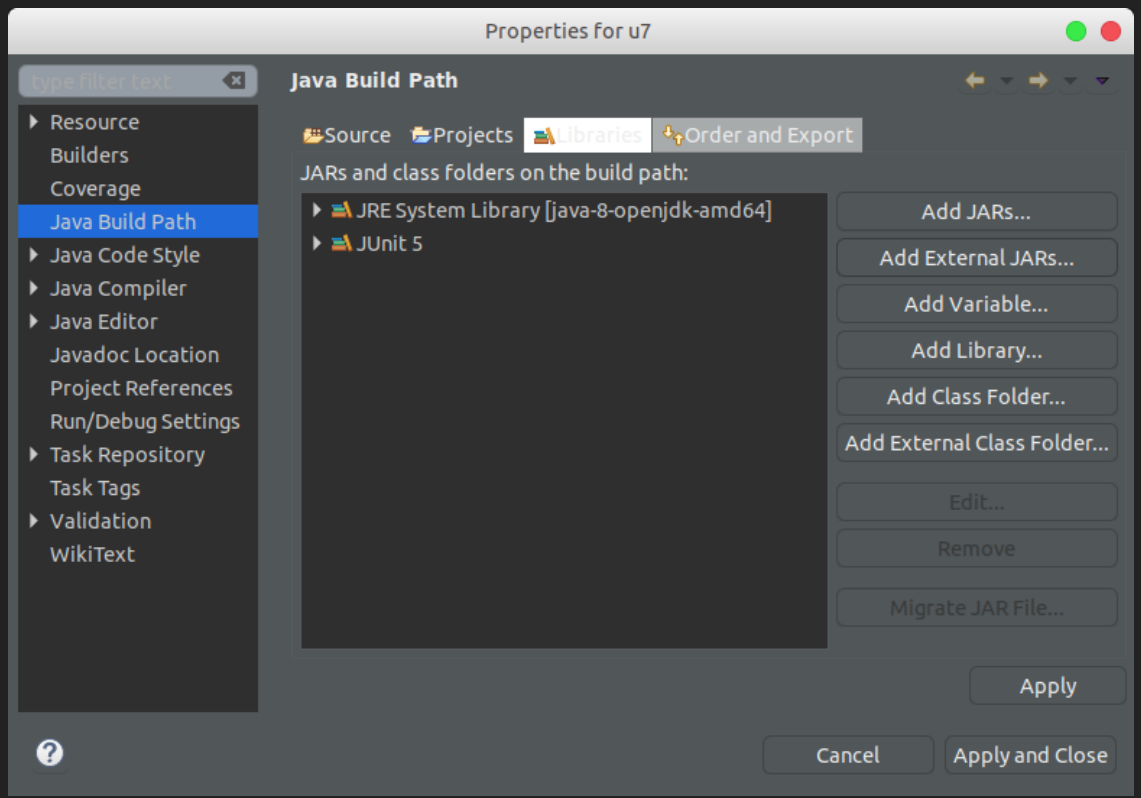
- Implementiert das Interface IBinarySerachTreeUtils<T>
- Generics $\rightarrow$ Verwendet T als euer Datentyp.
- Pro Knoten werde 2 Attribute gespeichert:
  - int key: Die id von eurem Element. Sortiert euren Baum nach diesem Attribut
  - T thing: Ein Objekt, welches im Baum gespeichert wird
- Die benötigten Algorithmen habe ich im Best-Of erklärt

Reversi Teil 1

- Super Möglichkeit praktische Programmierkentnisse zu sammeln
- Viel Prüfungsrelevanter Stoff wurde in den Reversi Übungen vertieft
- Sehr viel Spass
- Tolle Preise beim Turnier
- Schaut, ob ihr meinen Bot Trash.Can auf der Webseite schlagen könnt (machbar)
- Bei Fragen zögert nicht, Ich helfe *sehr* gerne

Reversi Teil 1

Das Reversi.jar File könnt ihr in den Properties wie J-Unit einfügen



Reversi Teil 1

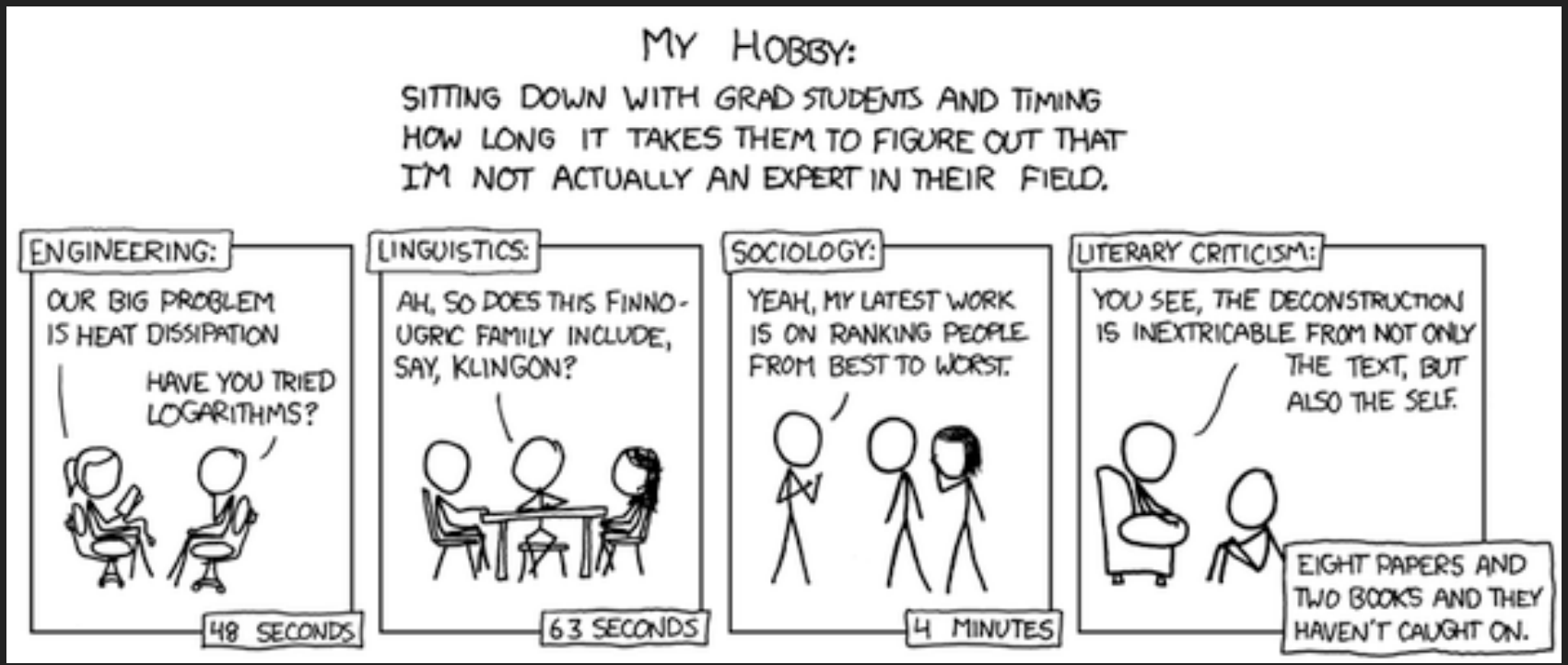
Um ein Spiel zu starten müsst Ihr dafür eine neue Run-Configuration erstellen:

1. Run $\rightarrow$ Run Configurations (im dropdown-Menu links vom Grünen Run-Knopf)
2. Rechtsklick auf JavaApplication $\rightarrow$ new Configuration
3. Name: Reversi (oder was auch immer)
   Project: u7
   Main class: reversi.Arena
4. Wählt Tab Arguments
5. Program arguments: -t 0 -r 50 TestGame u7a4.HumanPlayer u7a4.HumanPlayer
6. Apply
7. Run

Reversi Teil 1

- Implementiert einen "Random Player", welcher zufällig spielt
- Lasst euch vom "Human Player" inspirieren
- Benutzt Generics ArrayList<Coordinates> um die legalen Spielpositionen zu speichern
- Bei Fragen helfe ich sehr gerne

Viel Spass!


MY HOBBY:
SITTING DOWN WITH GRAD STUDENTS AND TIMING HOW LONG IT TAKES THEM TO FIGURE OUT THAT I'M NOT ACTUALLY AN EXPERT IN THEIR FIELD.
ENGINEERING:
OUR BIG PROBLEM IS HEAT DISSIPATION
HAVE YOU TRIED LOGARITHMS?
48 SECONDS
LINGUISTICS:
AH, SO DOES THIS FINNO-UGRIC FAMILY INCLUDE, SAY, KLINGON?
63 SECONDS
SOCIOLOGY:
YEAH, MY LATEST WORK IS ON RANKING PEOPLE FROM BEST TO WORST.
4 MINUTES
LITERARY CRITICISM:
YOU SEE, THE DECONSTRUCTION IS INEXTRICABLE FROM NOT ONLY THE TEXT, BUT ALSO THE SELF.
EIGHT PAPERS AND TWO BOOKS AND THEY HAVEN'T CAUGHT ON.