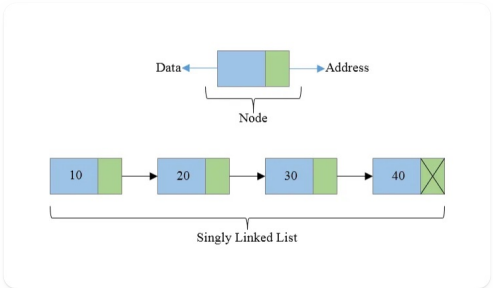
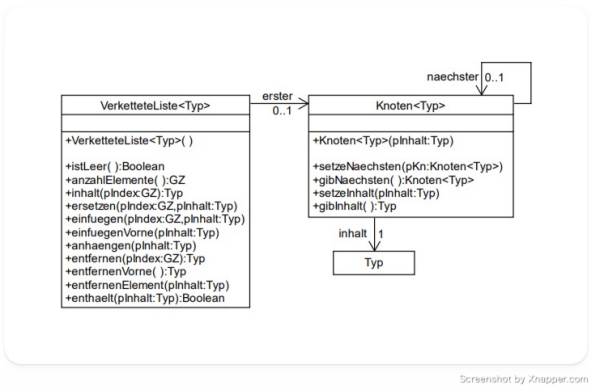


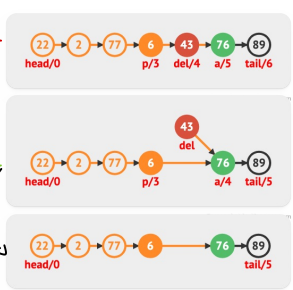
(Einfache) Verkettete Liste

Eine Liste ist entweder leer oder sie besteht sie besteht aus einem Knoten.
↳ enthält → Objekt von Datentyp A
→ Verweis auf eine Liste desselben types
→ Zeiger im Letzen Element zeigt auf die leere Liste (Nullzeiger)



Element Löschen

```
OPERATION entfernenElement(pInhalt:Typ)
  Lokale Variablen: vorheriger:Knoten<Typ>, aktueller:Knoten<Typ>

  vorheriger ← NICHTS
  aktueller ← erster                          ← Loop bis Ende (oder Abbruch)

  SOLANGE aktueller ≠ NICHTS
    WENN aktueller.gibInhalt() = pInhalt      ← Wenn 1 Knoten ist → verschiebe Zeiger
      WENN vorheriger = NICHTS
        erster ← aktueller.gibNaechsten()
      SONST
        vorheriger.setzeNaechsten(aktueller.gibNaechsten())   ← Verknüpfe Vorherigen Knoten mit den nächsten, um den aktuellen zu entfernen
      ENDE WENN
      ABBRUCH
    ENDE WENN

    vorheriger ← aktueller
    aktueller ← aktueller.gibNaechsten()
  ENDE SOLANGE
ENDE OPERATION
```



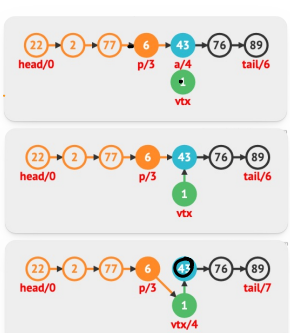
↳ Ende

Element suchen

```
OPERATION enthaelt(pInhalt:Typ):Boolean
  Lokale Variablen: gefunden:Boolean, knoten:Knoten<Typ>

  gefunden ← falsch
  knoten ← erster

  SOLANGE knoten ≠ NICHTS UND gefunden = falsch
    WENN knoten.gibInhalt( ) = pInhalt
      gefunden ← wahr
    SONST
      knoten ← knoten.gibNaechsten( )
    ENDE WENN
  ENDE SOLANGE

  RÜCKGABE gefunden
ENDE OPERATION
```

Element einfügen

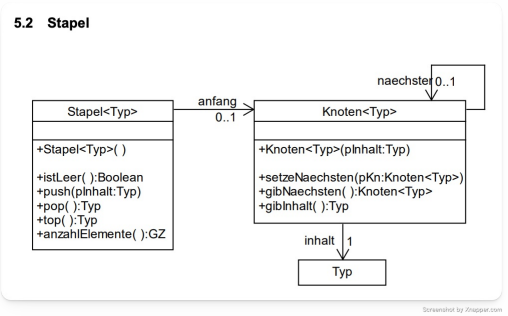
```
OPERATION einfuegen(pIndex:GZ, pInhalt:Typ)
  Lokale Variablen: neuerKnoten:Knoten<Typ>, aktueller:Knoten<Typ>, zaehler:GZ

  neuerKnoten ← NEU Knoten<Typ>(pInhalt)

  WENN pIndex = 0                             ← Wenn an Position 1
    neuerKnoten.setzeNaechsten(erster)
    erster ← neuerKnoten
  SONST
    aktueller ← erster
    zaehler ← 0

    SOLANGE aktueller ≠ NICHTS UND zaehler < pIndex - 1   ← Bewege zu Pos vor Einfügestell
      aktueller ← aktueller.gibNaechsten()
      zaehler ← zaehler + 1
    ENDE SOLANGE

    WENN aktueller ≠ NICHTS                   ← Wenn existiert
      neuerKnoten.setzeNaechsten(aktueller.gibNaechsten())   ← Neuer Knoten zeigt auf nächsten Knoten
      aktueller.setzeNaechsten(neuerKnoten)   ← Vorheriger Knoten zeigt auf den neuen Knoten
    ENDE WENN
  ENDE WENN
ENDE OPERATION
```



↳ Ende

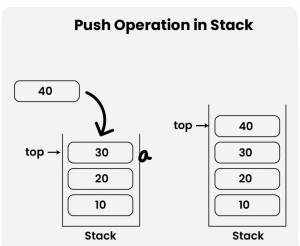
Stapel

Visualisierung



Stapelspeicher arbeitet nach LIFO-Prinzip
(↳ last-in – first-out)

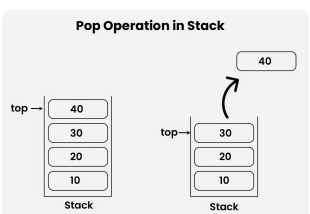
↳ zuletzt hinzugefügte Element zuerst entfernt wird.



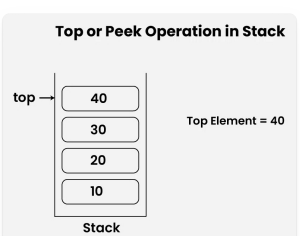

```
OPERATION push(pInhalt: Typ)
  Lokale Variablen: neuerKnoten: Knoten<Typ>

  neuerKnoten ← NEU Knoten<Typ>(pInhalt)
  neuerKnoten.setzeNaechsten(anfang)
  anfang ← neuerKnoten
ENDE OPERATION
```

Elemente zählen

```
OPERATION anzahlElemente(): GZ
  Lokale Variablen: zaehler: GZ, knoten: Knoten<Typ>

  zaehler ← 0
  knoten ← anfang

  SOLANGE knoten ≠ NICHTS
    zaehler ← zaehler + 1
    knoten ← knoten.gibNaechsten()
  ENDE SOLANGE

  RÜCKGABE zaehler
ENDE OPERATION
```

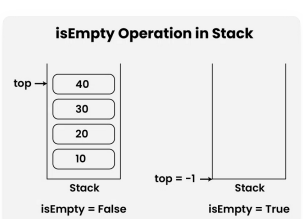



```
OPERATION pop(): Typ
  Lokale Variablen: inhalt: Typ

  WENN istLeer() = wahr
    RÜCKGABE NICHTS
  ENDE WENN

  inhalt ← anfang.gibInhalt()
  anfang ← anfang.gibNaechsten()

  RÜCKGABE inhalt
ENDE OPERATION
```




```
OPERATION top(): Typ
  WENN istLeer() = wahr
    RÜCKGABE NICHTS
  ENDE WENN

  RÜCKGABE anfang.gibInhalt()
ENDE OPERATION
```

↗ gibt das oberste Element zurück




```
OPERATION istLeer(): Boolean
  RÜCKGABE anfang = NICHTS
ENDE OPERATION
```

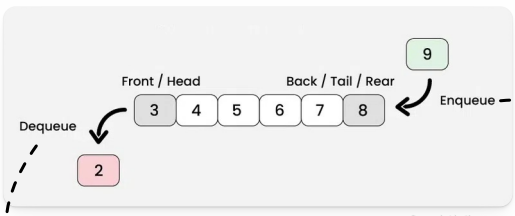
Warteschlange

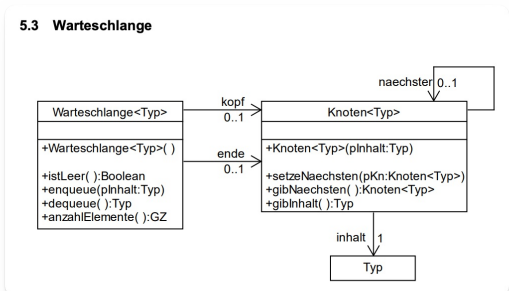
Visualisierung

↳ Warteschlange arbeitet nach FIFO

↳ First-In-First-Out

zuerst hinzukommendes Element wird auch als erstes entfernt






```
OPERATION dequeue(): Typ
  Lokale Variablen: inhalt: Typ

  WENN istLeer() = wahr
    RÜCKGABE NICHTS
  ENDE WENN

  inhalt ← kopf.gibInhalt()
  kopf ← kopf.gibNaechsten()

  WENN kopf = NICHTS
    ende ← NICHTS
  ENDE WENN

  RÜCKGABE inhalt
ENDE OPERATION
```

- null

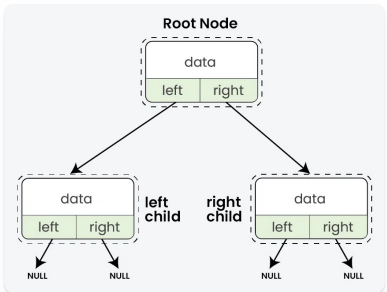
```
OPERATION enqueue(pInhalt: Typ)
  Lokale Variablen: neuerKnoten: Knoten<Typ>

  neuerKnoten ← NEU Knoten<Typ>(pInhalt)

  WENN istLeer() = wahr
    kopf ← neuerKnoten
    ende ← neuerKnoten
  SONST
    ende.setzeNaechsten(neuerKnoten)
    ende ← neuerKnoten
  ENDE WENN
ENDE OPERATION
```

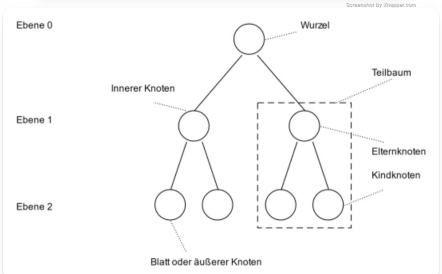
```
OPERATION anzahlElemente(): GZ
  Lokale Variablen: zaehler: GZ, knoten: Knoten<Typ>

  zaehler ← 0
  knoten ← kopf

  SOLANGE knoten ≠ NICHTS
    zaehler ← zaehler + 1
    knoten ← knoten.gibNaechsten()
  ENDE SOLANGE

  RÜCKGABE zaehler
ENDE OPERATION
```

```
OPERATION istLeer(): Boolean
  RÜCKGABE kopf = NICHTS
ENDE OPERATION
```

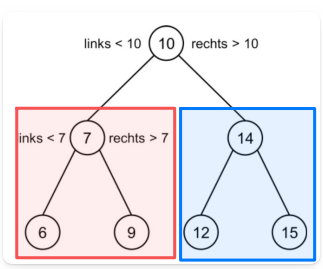
Binärbaum

Visualisierung





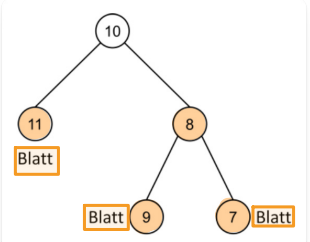
Sortiert



↳ Der linke Teilbaum enthält nur Knoten, die kleiner sind als Wurzel des T.

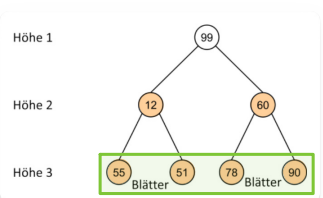
Der rechte Teilbaum enthält nur Knoten, die größer sind als Wurzel des T.

Voll



↳ Jeder Knoten:
- ist ein Blatt
- besitzt zwei Kinder

Vollständig



↳ Baum ist voll & alle Blätter auf gleicher Höhe

```
OPERATION ausgebenDatenPreorder() der Klasse Knoten
  inhalt.ausgebenDaten()

  WENN linkerTeilbaum ≠ NICHTS
    linkerTeilbaum.ausgebenDatenPreorder()
  ENDE WENN

  WENN rechterTeilbaum ≠ NICHTS
    rechterTeilbaum.ausgebenDatenPreorder()
  ENDE WENN
ENDE OPERATION
```

```
OPERATION ausgebenDatenInorder() der Klasse Knoten
  WENN linkerTeilbaum ≠ NICHTS
    linkerTeilbaum.ausgebenDatenInorder()
  ENDE WENN

  inhalt.ausgebenDaten()

  WENN rechterTeilbaum ≠ NICHTS
    rechterTeilbaum.ausgebenDatenInorder()
  ENDE WENN
ENDE OPERATION
```

```
OPERATION ausgebenDatenPostorder() der Klasse Knoten
  WENN linkerTeilbaum ≠ NICHTS
    linkerTeilbaum.ausgebenDatenPostorder()
  ENDE WENN

  WENN rechterTeilbaum ≠ NICHTS
    rechterTeilbaum.ausgebenDatenPostorder()
  ENDE WENN

  inhalt.ausgebenDaten()
ENDE OPERATION
```

Preorder

Wurzel → Links → Rechts

InOrder

Links → Wurzel → Rechts

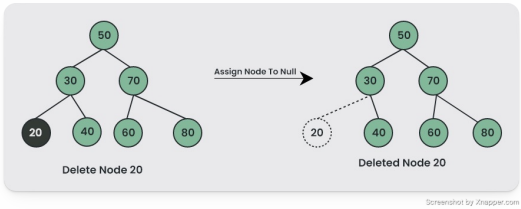
Postorder

Links - Rechts - Wurzel

Löschen von Knoten

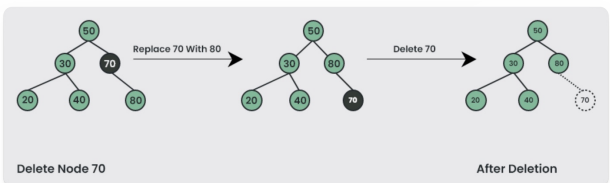
1. Blatt

↳ Einfach Knoten entfernen



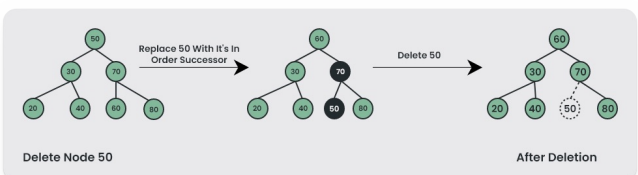
2. Knoten hat 1 Kind

↳ Knoten durch Blatt ersetzen



3. Knoten hat 2 Kinder

↳ mit kleinstem Element im rechten Teilbaum ersetzen & löschen



↳ geht auch wenn zu löschendes Element ein Kind ist!