Arrays: int [ ] arr = new int [10];

Datentyp (↑) Symbol (↑) Name (↑)

arr.length ⇒ Länge des Arrays arr [2] } Welt endet

Durchlaufen mit: for (int i = 0; i < arr.length; i++)

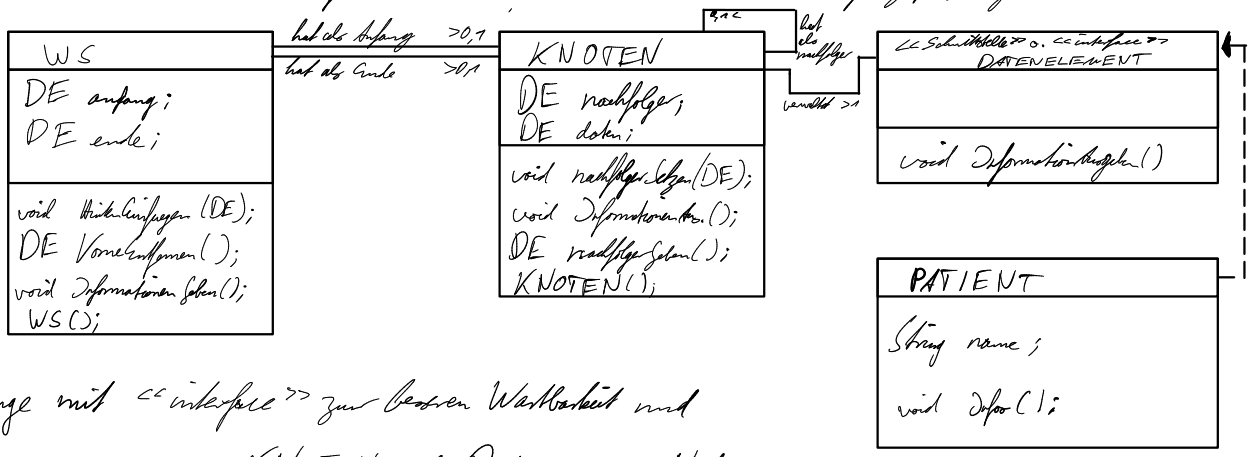
Stack: WS nach FILO

Warteschlange: (aus Feldern) → Jedes Element (z.B. Objekt Taxi; Argument ist arg; Aufrufen mittels Taxen auf!

⇒ Liste nach FIFO → Fürstin-First-out | Pro: wie in echt | Kontra: feste Größe, langsam

neue Idee:

DE ≙ Datenelement



⇒ Warteschlange mit << interface >> zur besseren Wartbarkeit und Wiederverwendbarkeit ⇒ KNOTEN sind Datenträger im Netz

Methode:

```
DE VorneEntfernen () {
    DE d = null
    if (anfang != null) {
        d = anfang.DEgeben();
        anfang = anfang.nachfolgerGeben();
        if (anfang == null) ende = null;
        return d;
    }}
```

```
void HintenEinfügen (DE dNeu) {
    KNOTEN neuK = new KNOTEN (dNeu);
    if (ende != null) ende.NachfolgerSetzen(neuK);
    else anfang = neuK;
    ende = neuK;
}
```

Rekursion: Bsp: Methode für Fakultäten: $n! = n \cdot (n-1)!$

⇒

```
int Fak (int n) {
    if (n > 1) {          // Abbruchbedingung
        return n * Fakultät (n-1); }   // Rekursionsschritt
    else return 1; }      // Rekursionsende
```

lineare Rekursion = max 1× aufruf

kaskadenartige Rekursion = min 2× aufruf

⇒ WS braucht rekursive Methoden um nachkommende, weitere KNOTEN zu befragen

⇒