Informatik

Abstrakte Datentypen - Keller und Schlange

ADT Keller (= Stapel, stack)

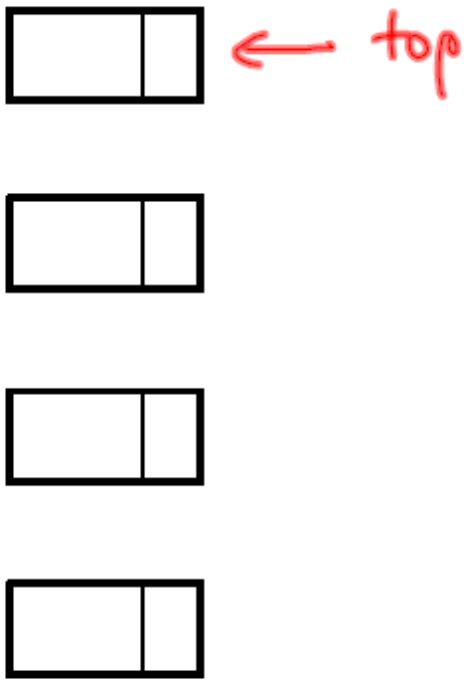
Prinzip LIFO: Last in, First Out

Ein Keller ist eine (ggf. leere) Folge von Elementen zusammen mit einem so genannten (ggf. undefinierten) Top-Element.

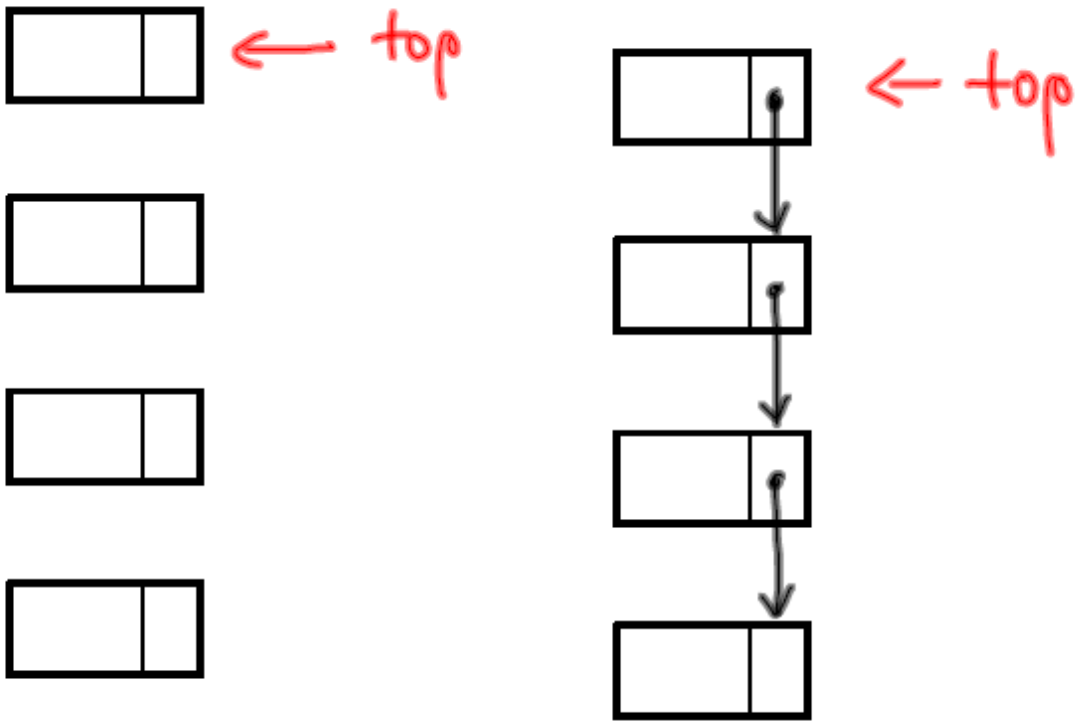
Schnittstelle des ADT Keller:

empty	:	liefert true, falls Keller leer
push	:	legt Element auf Keller
top	:	liefert oberstes Kellerelement
pop	:	entfernt oberstes Kellerelement

Wie sollen wir verzeigern?



Wie sollen wir verzeigern?



Anwendungsbeispiel: Korrektheit der Klammerung mittels Keller bestimmen:

$(((a + b) \cdot c + (a + c) \cdot 2) - 3) \cdot 5$ korrekt geklammert

$(((a + b) \cdot c + (a + c) \cdot 2)) - 3) \cdot 5$ nicht korrekt geklammert

```
class Keller:
    def __init__(self):
```

```
class Keller:
    def __init__(self):
        self.tp = None

    def empty(self):
```

```
class Keller:
    def __init__(self):
        self.tp = None

    def empty(self):
        return self.tp is None

    def push(self, x):
```

```
class Keller:
    def __init__(self):
        self.tp = None

    def empty(self):
        return self.tp is None

    def push(self, x):
        hilf = Eintrag()
        hilf.inhalt = x
        hilf.next = self.tp
        self.tp = hilf

    def top(self):
```

```
class Keller:
    def __init__(self):
        self.tp = None

    def empty(self):
        return self.tp is None

    def push(self, x):
        hilf = Eintrag()
        hilf.inhalt = x
        hilf.next = self.tp
        self.tp = hilf

    def top(self):
        if self.empty(): raise RuntimeError("Fehler: Keller ist leer")
        return self.tp.inhalt

    def pop(self):
```

```
class Keller:
    def __init__(self):
        self.tp = None

    def empty(self):
        return self.tp is None

    def push(self, x):
        hilf = Eintrag()
        hilf.inhalt = x
        hilf.next = self.tp
        self.tp = hilf

    def top(self):
        if self.empty(): raise RuntimeError("Fehler: Keller ist leer")
        return self.tp.inhalt

    def pop(self):
         if self.empty(): raise RuntimeError("Fehler: Keller ist leer")
         self.tp = self.tp.next
```

ADT Schlange

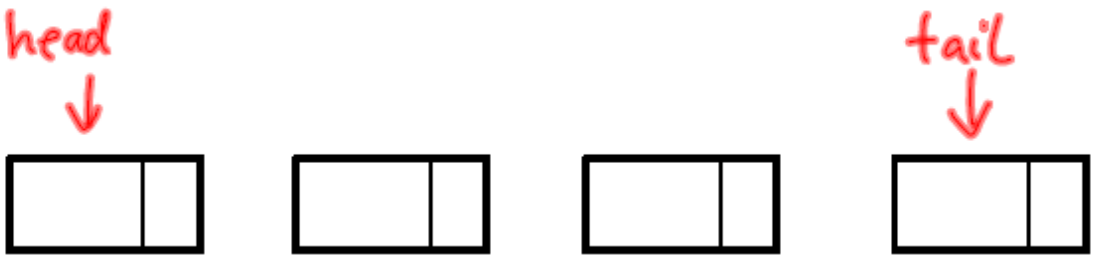
Eine Schlange ist eine (ggf. leere) Folge von Elementen zusammen mit einem so genannten (ggf. undefinierten) Front-Element.

Prinzip FIFO: First in, First Out

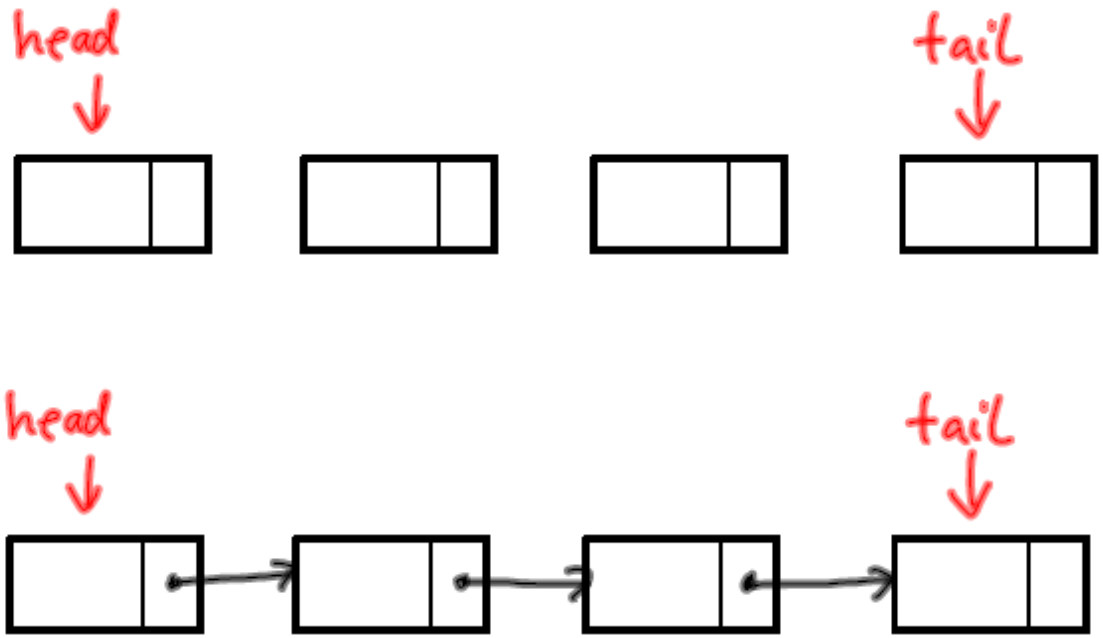
Schnittstelle des ADT Schlange:

empty	:	liefert true, falls Schlange leer
enq	:	fügt Element hinten ein
front	:	liefert vorderstes Element
deq	:	entfernt vorderstes Element

Wie sollen wir verzeigern?



Wie sollen wir verzeigern?



```
class Schlange:
    def __init__(self):
```

```
class Schlange:
    def __init__(self):
        self.head = None
        self.tail = None

    def empty(self):
```

```
class Schlange:
    def __init__(self):
        self.head = None
        self.tail = None

    def empty(self):
        return self.head is None

    def enq(self, x):
```

```
class Schlange:
    def __init__(self):
        self.head = None
        self.tail = None

    def empty(self):
        return self.head is None

    def enq(self, x):
        hilf = Eintrag()
        hilf.inhalt = x
        if self.empty():
            self.head = hilf
            self.tail = hilf
        else:
            self.tail.next = hilf
            self.tail = hilf

    def deq(self):
```

```
class Schlange:
    def __init__(self):
        self.head = None
        self.tail = None

    def empty(self):
        return self.head is None

    def enq(self, x):
        hilf = Eintrag()
        hilf.inhalt = x
        if self.empty():
            self.head = hilf
            self.tail = hilf
        else:
            self.tail.next = hilf
            self.tail = hilf

    def deq(self):
         if self.empty(): raise RuntimeError("Fehler: Schlange ist leer")
         self.head = self.head.next
         if self.head is None:
             self.tail = None

    def front(self):
```

```
class Schlange:
    def __init__(self):
        self.head = None
        self.tail = None

    def empty(self):
        return self.head is None

    def enq(self, x):
        hilf = Eintrag()
        hilf.inhalt = x
        if self.empty():
            self.head = hilf
            self.tail = hilf
        else:
            self.tail.next = hilf
            self.tail = hilf

    def deq(self):
         if self.empty(): raise RuntimeError("Fehler: Schlange ist leer")
         self.head = self.head.next
         if self.head is None:
             self.tail = None

    def front(self):
         if self.empty(): raise RuntimeError("Fehler: Schlange ist leer")
         return self.head.inhalt
```