Stacks und Queues

Behälter, Stacks, Stackpaare, De-Rekursivierung, Ausdrucksauswertung, FORTH, Postscript, Queues, Kanäle, Producer-Consumer

Behälter sind ...

... Sammlungen gleichartiger Objekte

- Operationen mit Behältern:
  - Objekte
    - aufnehmen (add)
    - löschen (remove)
    - suchen
  - Behälter durchlaufen, um z.B.
    - Objekte zählen
    - ein Objekt suchen
  - mehrere Behälter
    - zu einem zusammenfügen
    - Behälter kopieren
    - Teilbehälter bilden

© H. Peter Gumm, Philipps-Universität Marburg

Mögliche Merkmale von Behälter

- keine Reihenfolge:
  - Mengen
  - Bags
- mit Standardreihenfolge:
  - Bäume
  - Listen
- mit direktem Zugriff:
  - Maps
  - Arrays
- mit Einfüge/Entnahmereihenfolge:
  - Stacks
  - Queues

© H. Peter Gumm, Philipps-Universität Marburg

Behälter mit Zugriffsreihenfolge

- Ein Stapel (engl.: Stack) von Briefen oder Tellern
  - Man greift immer nur den obersten
  - Das ist der zuletzt abgelegte
- Ein Schlange von Personen an der Kasse
  - Der vorderste kommt zuerst dran
  - Der am längsten in der Schlange steht
- Eine Schlange am Flugschalter
  - Der vorderste kommt zuerst dran, aber...
  - Die Besatzung darf sich vordrängeln, sie hat Priorität

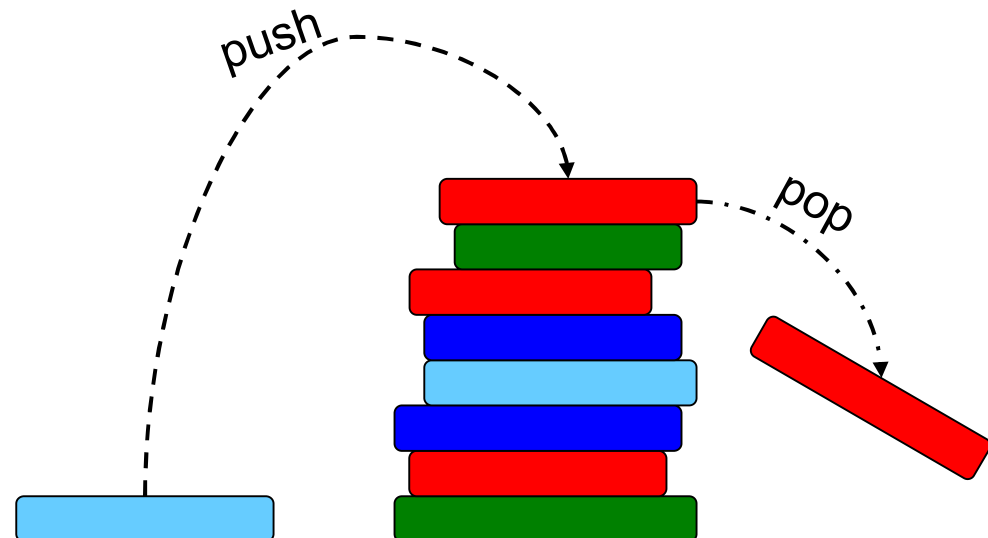
Stack

Set

© H. Peter Gumm, Philipps-Universität Marburg

Stack - Stapel

- direkter Zugriff nur auf letztes eingefügte Element
  - Last in – First Out (LIFO)
    - Stapel von Tellern
    - Stapel von Briefen
- Manche Leute sagen: „Keller“
  - Zuletzt eingelagerte Kartoffeln werden zuerst gegessen
- Stacks sind sehr wichtige Datenstrukturen der Informatik
  - Statt insert und remove sagt man
  - push und pop



© H. Peter Gumm, Philipps-Universität Marburg

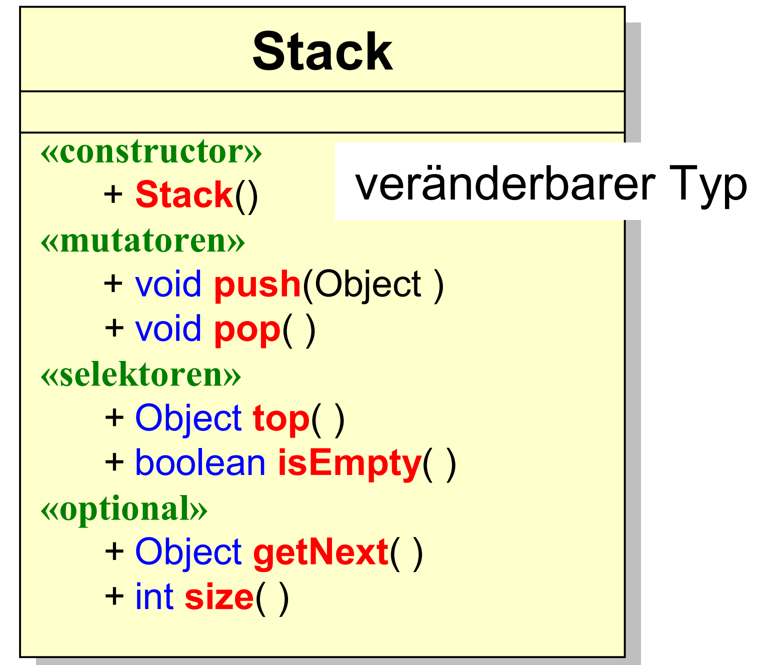
Stack - Spezifikation

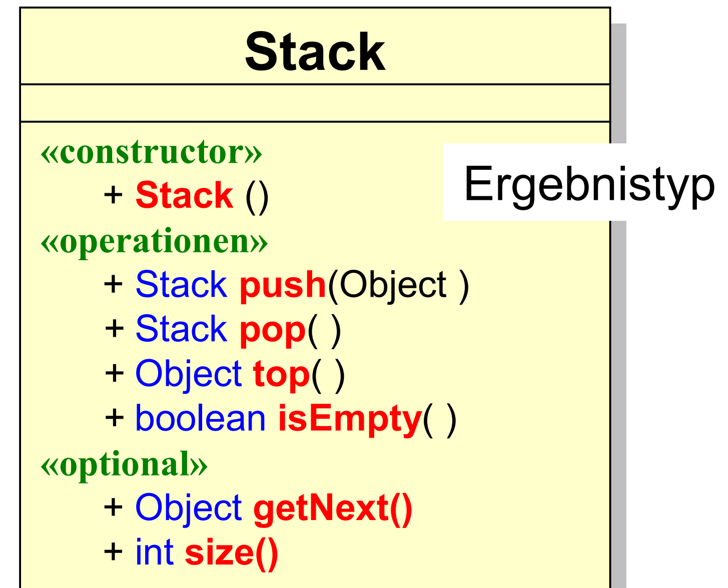
- LIFO-Behälter von Objekten
- Sort – Stack S von Objekten

- Operationen:
  - push : $Stack \times Object \to Stack$
  - pop : $Stack \to Stack$
  - emptyStack : $\to Stack$
  - top : $Stack \to Object$
  - isEmpty : $Stack \to boolean$

- Gleichungen
  - top(push(s,e)) = e
  - pop(push(s,e))= s
  - isEmpty(emptyStack)=true
  - isEmpty(push(s,e)) = false

```
E getNext() { E o = top(); pop(); return o; }
```

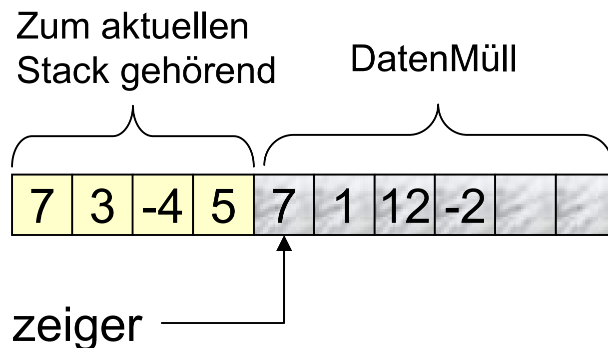




© H. Peter Gumm, Philipps-Universität Marburg

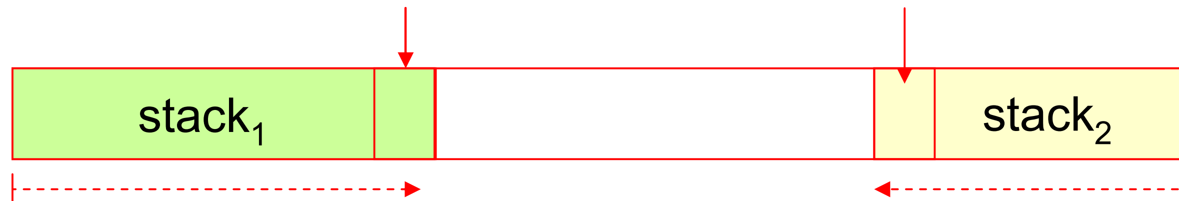
Bounded Stack

- Wir implementieren einen Stack als bounded stack
  - d.h. mit einer begrenzten Kapazität
  - Diese wird bei dem Aufruf des Constructors bestimmt
- Wir wählen die destruktive Variante der Stackoperationen
  - void push(Element e)
  - void pop()



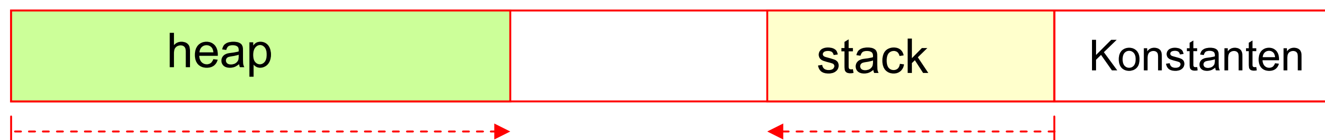

```java
public class BStack{
  // Stack repräsentiert durch theStack[0 .. zeiger-1]
   private Object[] theStack;
   private int zeiger;

  /** Konstruktor für Stack gewünschter Grösse */
   public BStack(int kapazität){
       theStack = new Object[kapazität];
   }
   /** Prüft ob Stack leer ist */
   public boolean isEmpty(){ return zeiger==0;}

   /** Legt neues Element auf den Stack */
   public void push(Object e){
      if(zeiger >= theStack.length)
         throw new StackException("push on full stack");
      else theStack[zeiger++] = e;
    }
   /** Entferne oberstes Element */
   public void pop(){
       if(zeiger>0) zeiger--;
       else throw new StackException("pop on empty stack");
    }
   /** Liefert oberstes Element */
   public Object top(){
        if(isEmpty())
           throw new StackException("top of empty stack");
        else return theStack[zeiger-1];
    }
   /** Liefert oberstes Element und entfernt es vom Stack */
   public Object getNext(){
```

Stackpaare



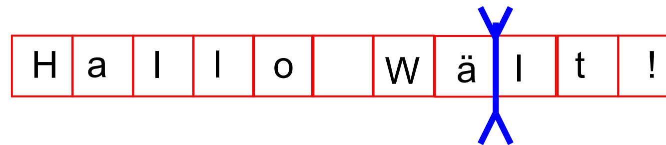
- Zwei Stacks kann man in einem Array unterbringen
  - einer wächst von links nach rechts
  - der andere von rechts nach links
  - Überlauf erst, wenn Summe der Längen > Länge des Arrays
- Optimale Ausnutzung des vorhandenen Platzes
- Viele Programmiersprachen
  - z.B. Pascal

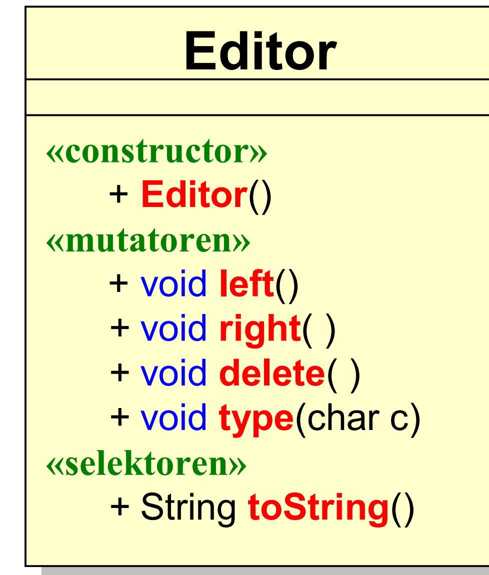
  verwalten so
  - den heap
    - dynamischer Speicher
  - den runtime-stack
    - für Methodenaufrufe
    - lokale Variablen



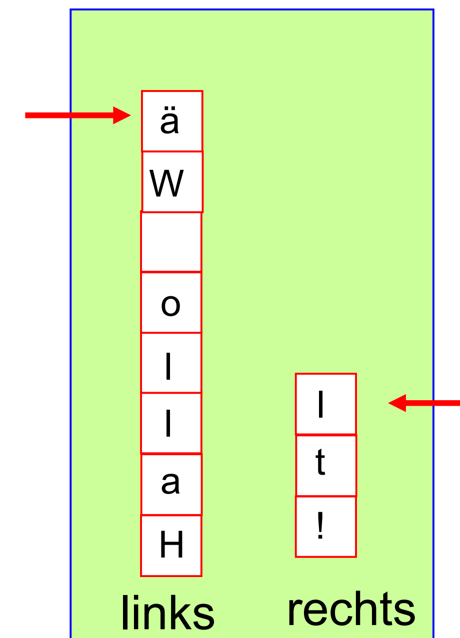
© H. Peter Gumm, Philipps-Universität Marburg

Anwendung: Text Editor





- Ein Texteditor besteht aus
  - einer Folge von Zeichen
  - einem Cursor
- Operationen
  - zur Bewegung des Cursors
  - zum Tippen oder Löschen
- Zur Implementierung eignet sich ein Stackpaar (links,rechts)
  - links : die Zeichen vor dem Cursor
  - rechts : die Zeichen rechts vom Cursor
- Positionierung und Operationen
  - left() = right.push(left.getNext())
  - right() = left.push(right.getNext());
  - delete() = left.pop()
  - type(c) = left.push(c)



© H. Peter Gumm, Philipps-Universität Marburg

Auswertung rekursiver Methoden

- Beim Eintritt
  - Sicherе alle lokalen Variablen und Parameter auf dem Stack
- Nach Beendigung
  - Lade sie wieder vom Stack

```
static void binär(int n){
    if(n < 10) schreibe(n);
    else{
        binär(n/2);
        schreibe(n%2);
}
```

```
void binär(int n){
  if (n<2) schreibe(n);
  else{
    binär(n/2);
    schreibe(n%2);
  }
}
```

```
push(n);
n=n/2
```

```
void binär(int n){
  if (n<2) schreibe(n);
  else{
    binär(n/2);
    schreibe(n%2);
  }
}
```

```
n=getNext();
```

```
push(n);
n=n/2
```

```
void binär(int n){
  if (n<2) schreibe(n);
  else{
    binär(n/2);
    schreibe(n%2);
  }
}
```

```
n=getNext();
```


© H. Peter Gumm, Philipps-Universität Marburg

Auswertung von binär(13)

binär(n) — n: 13 — Stack: (leer)

push(n);
n=n/2;
binär(n) — n: 6 — Stack: 13

push(n);
n=n/2;
binär(n) — n: 3 — Stack: 6, 13

push(n);
n=n/2;
binär(n) — n: 1 — Stack: 3, 6, 13

schreibe(n) — n: 1 — Stack: 3, 6, 13 — Ausgabe: 1

n=getNext();
schreibe(n%2) — n: 3 — Stack: 6, 13 — Ausgabe: 11

n=getNext();
schreibe(n%2) — n: 6 — Stack: 13 — Ausgabe: 110

n=getNext();
schreibe(n%2) — n: 13 — Stack: (leer) — Ausgabe: 1101

© H. Peter Gumm, Philipps-Universität Marburg

Anwendung: De-Rekursivierung

- Auf dem Stack abgelegte Elemente entnimmt man in umgekehrter Reihenfolge
- *WriteBinary* nutzt dies aus:
  - die Binärziffern werden in falscher Reihenfolge produziert und auf dem Stack abgelegt
  - dann werden sie vom Stack entnommen und ausgegeben

StackAnwendungen

Klasse Bearbeiten Werkzeuge Optionen

Übersetzen Rückgängig Ausschneiden Kopieren Einfügen Suchen... Weitersu

```
static void writeBinary(int n){
   BStack<Integer> s = new BStack(64);
   while (n > 0){
      s.push(n%2);
      n=n/2;
   }
   while(!s.isEmpty()){
     System.out.print(s.top());
     s.pop();
   }
}
```

BlueJ: Konsole - s...

Optionen

```
11111010101
```

BlueJ: Methodenaufruf

void writeBinary(int n)

StackAnwendungen.writeBinary (2005)

Ok Abbrechen

© H. Peter Gumm, Philipps-Universität Marburg

Iterative Version mit Stack s

De-Rekursivierung

Eine *linear-rekursive* Funktion

$$f(x) = \begin{cases} g(x), \text{ falls } p(x) \\ h(x, f(r(x))), \text{ sonst} \end{cases}$$

Programmierung in Java

```
Y f(X x){
    if(p(x)) return g(x);
    else     return h(x,f(r(x)));
}
```

X und Y stehen für
Argument- bzw. Resultattyp

Rek

Klasse Bearbeiten Werkzeuge Optionen

Übersetzen Rückgängig Ausschneiden Kopieren Einfügen Suchen...

```
22  Y fIterativ(X x){
23      Y result;
24      X arg;
25      Stack<X> s = new Stack<X>();
26
27      while(!p(x)){
28        s.push(x);
29        x=r(x);
30      }
31      result = g(x);
32
33      while(!s.isEmpty()){
34          arg=s.pop();
35          result = h(arg,result);
36        }
37      return result;
38    }
```

In java.util.Stack :
pop() statt getNext()

Klasse übersetzt - keine Syntaxfehler

gespeichert

© H. Peter Gumm, Philipps-Universität Marburg

Arithmetische Ausdrücke

- Für die Angabe komplexer arithmetische Ausdrücke benutzt man
  - Präzedenzen
  - Klammern
  - Beispiele:
    - $(x+1)*x + 1/x * e^{-(x+1)} * \sin (1/x)$
- Dies macht die technische Auswertung kompliziert
  - Wie berechnen Sie solche Ausdrücke mit dem Taschenrechner?
    - $2*(x+x^2)$, $(x+1)^2/(x^2+1)$, ...
  - Geben Sie einen Algorithmus an, wie man solche Ausdrücke auswertet

© H. Peter Gumm, Philipps-Universität Marburg

Notation von Ausdrücken

- Infixnotation
  - Operationszeichen zwischen den Operanden
    - x + 3, 2*15, x and y
  - erfordert Klammern
    - (x+1)*15, x and (y or z), x / √(x+1), sin (x +1)
- Praefixnotation
  - Operationszeichen vorne
    - + x 3, * 2 15, and x y
  - erfordert keine Klammern
    (sofern Stelligkeit der Operatoren bekannt ist)
    - * + x 1 15, and x or y z, / x √ + x 1, sin + x 1
  - aber Operator muss auf Berechnung der Argument warten
  - wird in der Sprache LISP verwendet.
- Postfixnotation (auch UPN-umgekehrte polnische Notation)
  - Operationszeichen hinten
    - x 3 +, 2 15 *, x y and
  - erfordert keine Klammern
    - x 1 + 15 *, x y z or and, x x 1 + √ / , x 1 + sin
  - Operatoren haben gleich ihre fertigen Argumente

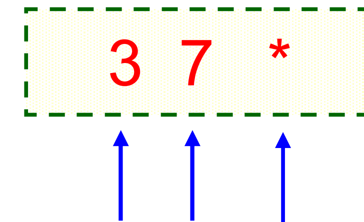
HP 35 der erste erschwingliche programmierbare Taschenrechner

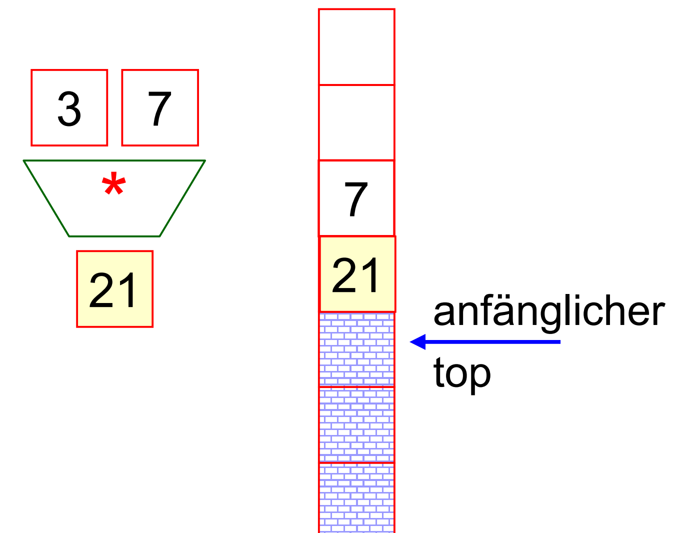
Ein Beispielprogramm z.B. bei www.hpmuseum.org/software/25simeq.htm

© H. Peter Gumm, Philipps-Universität Marburg

Expression-Auswertung mit Stack

- Um eine Operation mit k-Argumenten auszuwerten
  - Lege k Argumente auf den Stack
    - push
  - Operation entnimmt oberste k Elemente
    - getNext
  - Berechnet das Ergebnis
  - Legt das Ergebnis auf dem Stack ab.
- Netto-Veränderung des Stacks:
  - Wie zu Beginn, aber das Ergebnis ist hinzugekommen und liegt obenauf



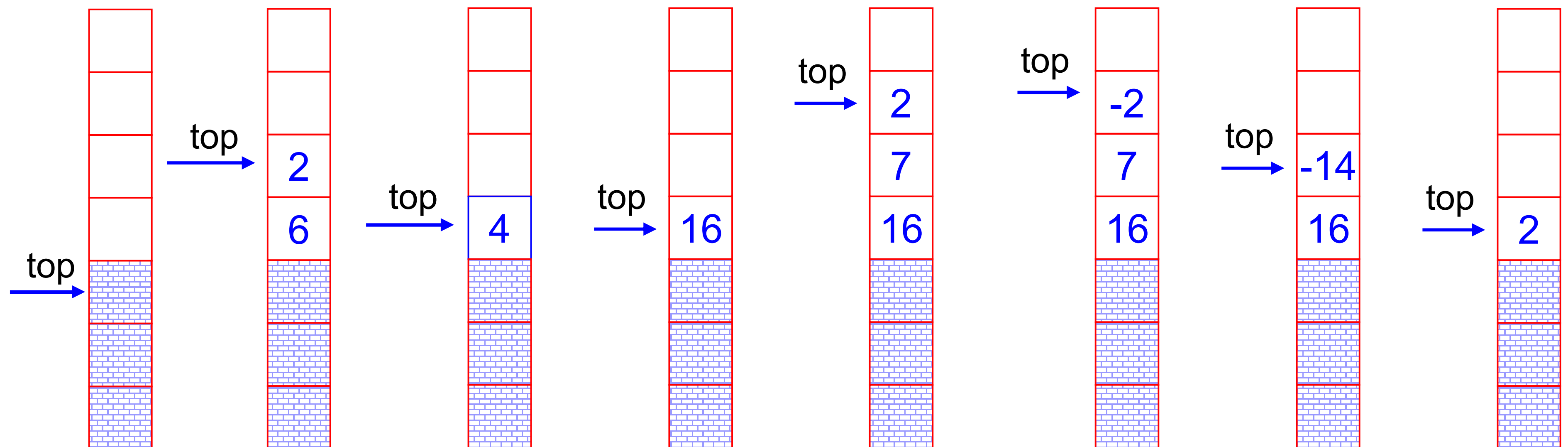


© H. Peter Gumm, Philipps-Universität Marburg

6 2 – ² 7 2 (±) * +

Postfix Ausdruck : 6 2 – ² 7 2 (±) * +

Stack-Maschinen Code: push(6) push(2) sub sqr push(7) push(2) neg mul add

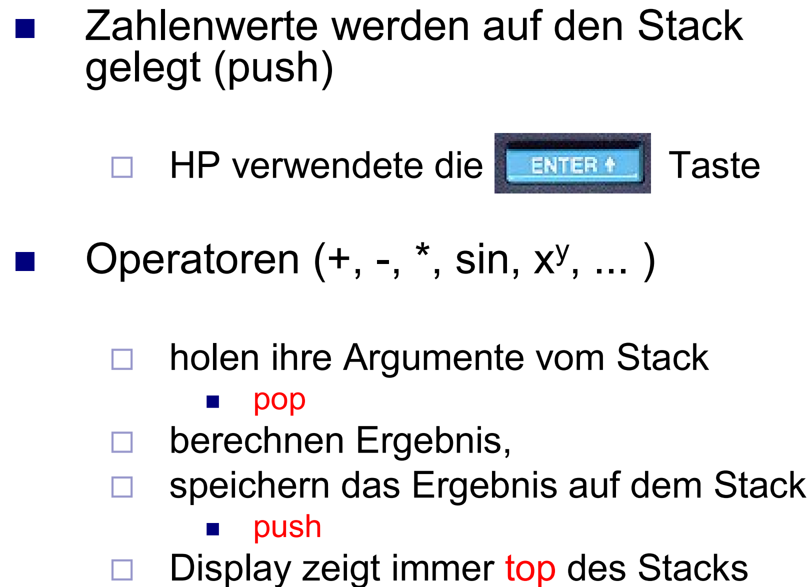


push(6)
push(2) sub sqr push(7)
push(2) neg mul add

Netto: push(Ergebnis des Ausdrucks)

© H. Peter Gumm, Philipps-Universität Marburg

Auswertung von UPN mit Stack



- Zahlenwerte werden auf den Stack gelegt (push)
  - HP verwendete die ENTER ↑ Taste
- Operatoren (+, -, *, sin, x^y, ...)
  - holen ihre Argumente vom Stack
    - pop
  - berechnen Ergebnis,
  - speichern das Ergebnis auf dem Stack
    - push
  - Display zeigt immer top des Stacks

Infix :	1 / sin (ln(17) + 5 * √ 3)
Präfix :	(1/x)(sin(+(ln(17),* (5 , √ (3))))
Postfix:	17 ln 3 √ 5 * + sin 1/x

HP-35 :

© H. Peter Gumm, Philipps-Universität Marburg

Stack Evaluierung: 17 ln 3√5 *+sin1/x

Der Stack

- push(17)
  - 17 auf den Stack legen

17.0			

- ln
  - Einstellige Operation
  - Argument: top(), entfernen mit pop()
  - Resultat berechnen und auf Stack legen

2.8			

- Push(3)

2.8	3.0		

- √
  - Analog zu ln

2.8	1.7		

- Push(5)

2.8	1.7	5.0	

- ×
  - Argumente: Oberste zwei Stackelemente
  - Argumente entfernen pop(), pop()
  - Ergebnis der Operation auf Stack legen

2.8	8.5		

- +
  - Analog zu ×

11.3			

- sin
  - Analog zu ln, √

0.19			

- 1/x
  - Analog zu ln, √, sin

5.1			

© H. Peter Gumm, Philipps-Universität Marburg

FORTH – die Stacksprache

- FORTH ist eine Programmiersprache
  - FORTH besteht aus Kommandos („words“) und Operationen, die einen Stack manipulieren
  - Mit FORTH kann man auch moderne Programme schreiben
    - objektorientiert
    - mit GUI
    - CGI, etc. ..
- FORTH ist sehr maschinennah
  - Forth ist sehr effizient
  - Die Java-Virtual-Machine (JVM) ist eine FORTH-ähnliche Sprache
- FORTH Programme sind für nicht eingeweihte schwer zu lesen
- FORTH-Programmieren macht Spaß

http://wiki.forthfreak.net/jsforth80x25.html

```
\ Drei kleine FORTH-Programme:
\ Euklids Algorithmus
: UMOD ( u1 u2 - remainder)
 0 SWAP UM/MOD DROP ;
: GCD ( u1 u2 -- gcd )
  BEGIN ?DUP WHILE TUCK
  UMOD REPEAT ;
\ das gleiche rekursiv
: GCD-RECURSIVE ( u1 u2 - gcd )
 ?DUP IF TUCK
 UMOD RECURSE THEN ;
\ First 20 Fibonacci numbers
: fibonacci ( -- )
  0 1 20 0 DO DUP .
  TUCK + LOOP 2DROP ;
```


© H. Peter Gumm, Philipps-Universität Marburg

Eine Interaktion mit FORTH

```
C:\Dokumente und Einstellungen\Peter\Desktop\P...
GForth 0.5.0, Copyright (C) 1995
GForth comes with ABSOLUTELY NO
Type 'bye' to exit
  ok
23 17 39 5 66   ok
.s <5> 23 17 39 5 66   ok
. 66  ok
.s <4> 23 17 39 5   ok
dup   ok
.s <5> 23 17 39 5 5   ok
drop   ok
.s <4> 23 17 39 5   ok
rot   ok
.s <4> 23 39 5 17   ok
+   ok
.s <3> 23 39 22   ok
-   ok
.s <2> 23 17   ok
*   ok
.s <1> 391   ok
2 /   ok
.s <1> 195   ok
3 mod .s <1> 0   ok
. 0  ok
.s <0>   ok
\ Wir berechnen 2+3*4-5 :     ok
2 3 4 * + 5 - . 9   ok
bye
```

Zahlen und Werte werden auf den Stack gelegt

23 17 39 5 66

Im interaktiven Modus beantwortet FORTH korrekte Eingaben mit ‘ok‘

FORTH-words

.s	zeigt den Stack
.	pop das oberste Element und zeige es an
swap	vertausche oberste Elemente
dup	dupliziere oberstes Element
drop	pop
rot	rotiere obersten drei Elemente
-	subtrahiere stack[top] – stack[top-1] und ersetze sie durch Ergebnis
+, *, /, mod	analog
\	beginnt Kommentarzeile

Mehrere Kommandos auf einer Zeile erlaubt.

© H. Peter Gumm, Philipps-Universität Marburg

Programmieren in FORTH

- Zwischen ':' und ';' können neue *FORTH-Worte* (Programme) definiert werden
  - Das System antwortet mit 'compiled'
- Zuerst üben wir, wie die Funktion berechnet wird:
  - quadratszumme(5,6) = 5*5+6*6
- Dann definieren wir ein Wort für die entsprechenden Aktionen
- Definition beginnt mit '**:**' und endet mit '**;**'

C:\Dokumente und Einstellungen\Peter\Desktop\PraktInf_2\gforth\gf...

```
GForth 0.5.0, Copyright (C) 1995-2000
GForth comes with ABSOLUTELY NO WARRANTY;
Type 'bye' to exit
\                                  ok
\ Auf dem Stack liegen zwei Zahlen   ok
\ Berechne ihre Quadratsumme          ok
\  ok
5 6   ok
.s <2> 5 6   ok
dup *    ok
.s <2> 5 36   ok
swap dup *    ok
.s <2> 36 25   ok
+ . 61   ok
\                                     ok
\ Dafür definieren wir ein Wort       ok
: quadratsumme                      compiled
   dup * swap dup * +              compiled
;  ok
\                                    ok
\ Wir probieren es aus :             ok
2 3 quadratsumme . 13   ok
bye
```


© H. Peter Gumm, Philipps-Universität Marburg

Postscript

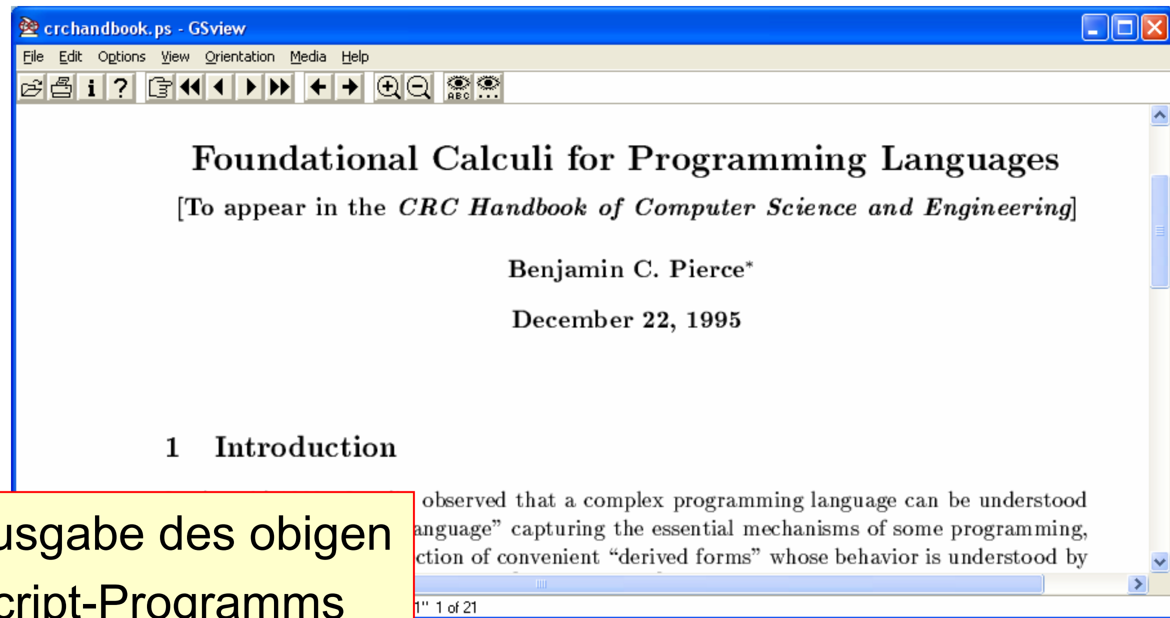
- Seitenbeschreibungs-sprache von Adobe

- Ergebnis
  - Dokumente
    - mit Graphik
    - Mit Farbe
- Beliebige Berechnungen
  - Arithmetik
  - Programme
- Theoretisch kann man damit beliebige Programme schreiben
- Postscript Dokumente sind Programme für den Postscript Interpreter
- Schauen Sie z.B. einmal mit einem Editor in eine Postscript-Datei:

C:\Dokumente und Einstellungen\Peter\Desktop\crchandbook.ps

```
%%Page: 1 1
1 0 bop 55 231 a Fs(F)-7 b(oundational)26 b(Calculi)g(for)g
(Programming)i(Languages)23 323 y Fr([T)-5 b(o)18 b(app)r(ear)h
(the)g Fq(CR)o(C)i(Handb)m(o)m(ok)i(of)f(Computer)g(Scienc)m(e)
(Engine)m(ering)p Fr(])694 449 y(Benjamin)17 b(C.)h(Pierce)1175
431 y Fp(\003)701 551 y Fr(Decem)n(b)r(er)e(22,)i(1995)0
812 y Fo(1)67 b(In)n(tro)r(duction)0 914 y Fn(In)17 b(the)f(mic
observ)o(ed)e(that)f(a)h(complex)h(programmi
)q(e)f(understo)q(o)q(d)0 970 y(in)h(terms)f
g(language")h(capturing)f(the)h(essen)o(tial
e)g(programming,)0 1027 y(st)o(yle)e(togethe
ion)j(of)d(con)o(v)o(enien)o(t)h(\\deriv)o(e
)q(eha)o(vior)h(is)h(understo)q(o)q(d)f(b)o(
j(them)f(in)o(to)h(the)f(core)h(\(cf.)701
```

Ein Postscript Dokument
Ist ein Programm für den
Postscript Interpreter



Die Ausgabe des obigen
Postscript-Programms

© H. Peter Gumm, Philipps-Universität Marburg

Postscript

Ghostscript Image

Hallo, Welt!

Ghostscript

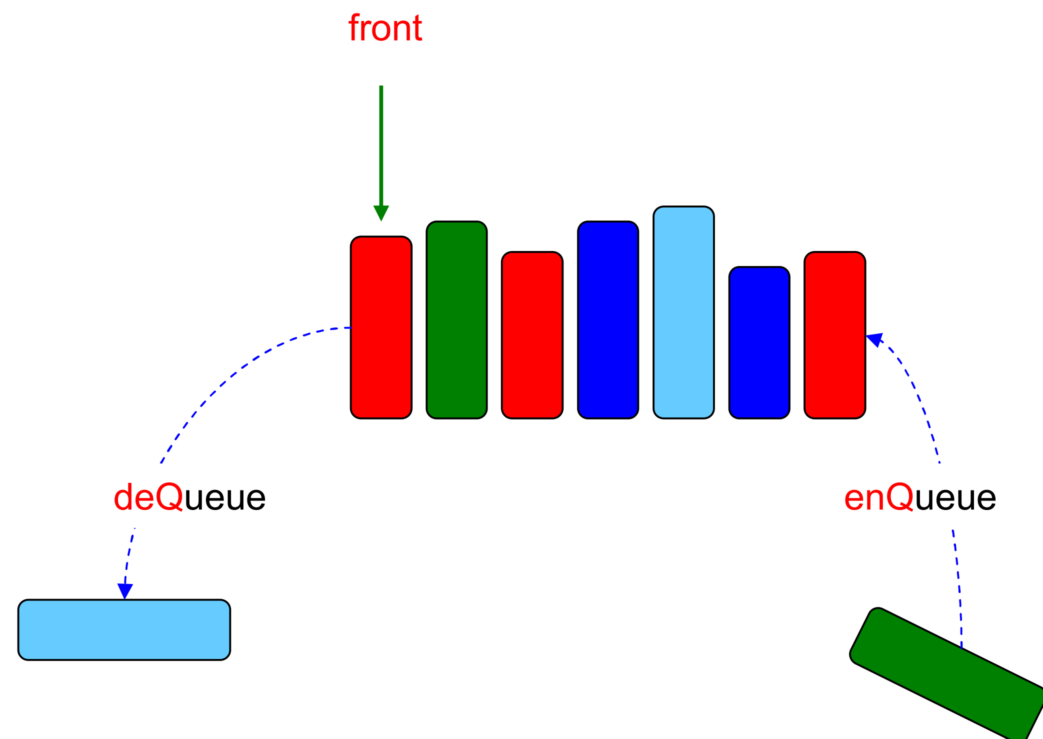
```
GNU Ghostscript 7.05 (2002-04-22)
Copyright (C) 2002 artofcode LLC, Benic
This software comes with NO WARRANTY: s
GS>2 3 add
GS<1>=
5
GS>2 5 7 mul add 6 sub =
31
GS>/quadratsumme {
   dup mul
   exch dup mul
   add } def
GS>2 3 quadratsumme
GS<1>=
13
GS>100 100 moveto
GS>200 300 lineto
GS>stroke
GS>200 200 moveto
GS>0 10 300 {
   200 200 moveto
   100 lineto
   stroke } for
GS>/Times-Roman findfont
Loading NimbusRomNo9L-Regu font fro
20 625828 1622528 330046 0 done.
GS<1>20 scalefont
GS<1>setfont
GS>100 50 moveto
GS>(Hallo, Welt!) show
GS>100 300 moveto
GS>0 15 300 {
   rotate
   (Hallo) show } for
GS>
```

- Stackcode
  - Arithm. Operatoren u.a.
  - **add, mul, sub, div, mod**
- Definitionen
  - beginnen mit **/**
  - enden mit **def**
  - Code zwischen **{** und **}**
- Stackmanipulation
  - **dup, exch,**
- Graphische Operationen
  - **moveto** (Bezugspunkt)
  - **lineto**
  - **stroke** (mache sichtbar)
- for-Schleife
  - ***from k to* {**
    Block **} for**
- Text
  - Fontauswahl
  - **scaleto** Skalierung
  - Strings in runden Klammern
  - **show**
  - **rotate**: Textrichtung

© H. Peter Gumm, Philipps-Universität Marburg

Queues

- Behälter Datentyp
  - Zugriff immer auf Objekt, das am längsten im Behälter ist
  - First In First Out – FIFO
- Warteschlange
  - Schlange in der Bäckerei
  - An der Bushaltestelle
- Operationen
  - enQueue – einfügen
  - deQueue – entnehmen
  - isEmpty – ist noch ein Element vorhanden ?
  - front – nächstes Element



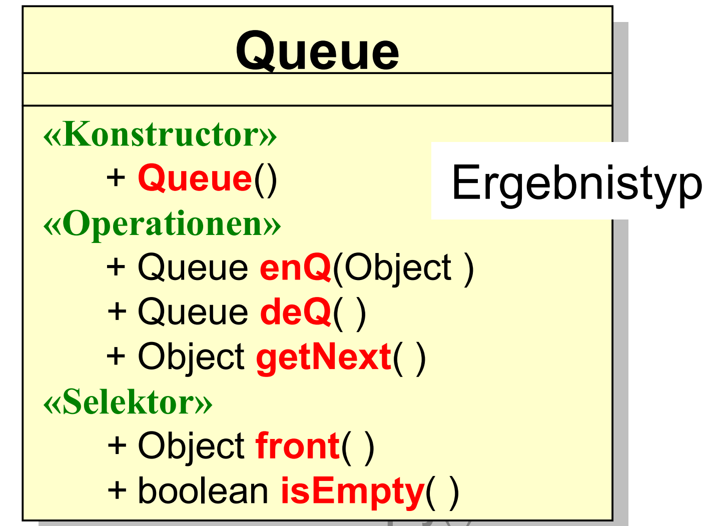
© H. Peter Gumm, Philipps-Universität Marburg

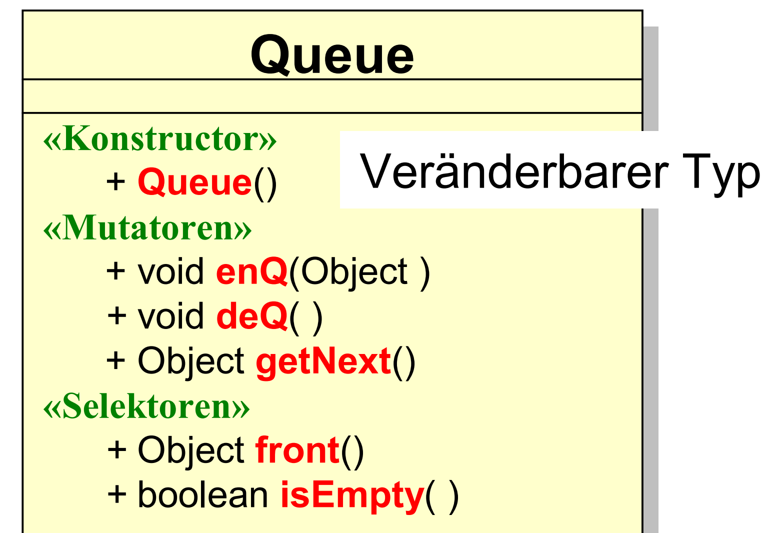
Queues - Warteschlangen

- FIFO Behälter
- Sort – Queue Q von Objekten

- Operationen:
  - enQ : $Queue \times Object \to Queue$
  - deQ : $Queue \to Queue$
  - emptyQueue : $\to Queue$
  - front : $Queue \to Object$
  - isEmpty : $Queue \to boolean$

- (Bedingte) Gleichungen
  - isEmpty(q) $\Rightarrow$ deQ(enQ(q,x)) = emptyQueue
  - $\neg$ isEmpty(q) $\Rightarrow$ deQ(enQ(q,x)) = enQ(deQ(q),x)
  - isEmpty(q) $\Rightarrow$ front(enQ(q,x)) = x
  - $\neg$ isEmpty(q) $\Rightarrow$ front(enQ(q,x)) = front(q)
  - isEmpty(emptyQueue)=true
  - isEmpty(enQ(q,x)) = false





© H. Peter Gumm, Philipps-Universität Marburg

Beispiele von Queues

- Kanal
  - Daten müssen in der richtigen Reihenfolge ankommen
    - Sender schreibt in den Kanal,
    - Empfänger liest aus dem Kanal
- Puffer (engl.: buffer)
  - Lesen aus einer Datei
    - Festplatte schreibt in den Puffer
    - Programm liest aus dem Puffer
    - Vorteil: Zeitliche Entkopplung
  - Schreiben einer Datei
    - analog
- Warteschlange
  - Printerqueue
    - Programm schreibt Druckauftrag in die Printerqueue
    - Drucker arbeitet alle Druckaufträge in der Reihenfolge des Ankommens ab
- Pipe (üblich unter Unix/Linux)
  - Verbindung zweier Programme
    - Programm 1 leitet Ausgabe in pipe
    - Programm 2 entnimmt Eingabe aus der Pipe

© H. Peter Gumm, Philipps-Universität Marburg

Producer - Consumer

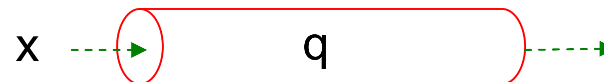
- Produzent
  - produziert Daten
- Konsument
  - Nimmt Daten entgegen
- Entkopplung durch Queue = Puffer
  - Produzent: enQ
  - Konsument: deQ
  - Vorteil: Produzent und Konsument können asynchron arbeiten
    - Keiner verlangsamt den anderen
- Konkret
  - Beim Lesen von der Festplatte:
    - Producer: Festplatte
    - Consumer: Programm
  - Bei einer Pipe
    - Programm1 ist producer
    - Programm2 ist consumer
  - Beim Senden von Daten
    - Producer: Sender
    - Consumer: Empfänger

© H. Peter Gumm, Philipps-Universität Marburg

Producer-Consumer-Protokoll

- Producer
  - produziert
  - wartet ggf. bis Queue nicht voll
    - busy waiting
  - fügt Element in Queue
- Consumer
  - wartet ggf. bis q nicht leer
    - busy waiting
  - entnimmt Element
  - konsumiert es

```
void producer(){
while(!feierabend()){
  produce(x);
  while(q.isFull()) {  }
  q.enQ(x);
  }
}
```




```
void consumer(){
  while(!feierabend()) {
    while(q.isEmpty()){ }
    x = q.getNext();
    consume(x);
  }
}
```

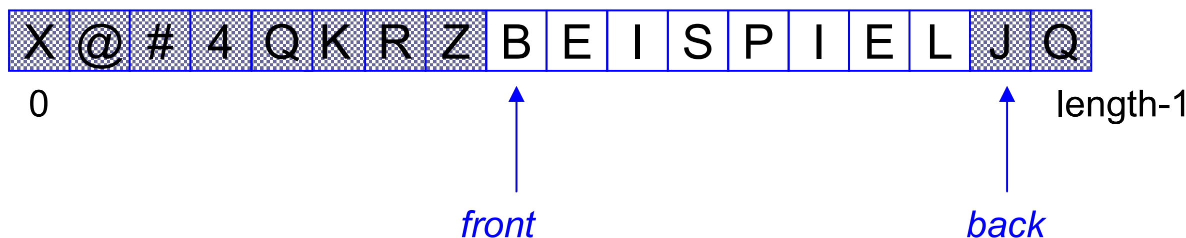
Eine volle Queue beim Producer, bzw. eine leere Queue beim Consumer führt nicht zu einer Exception, sondern veranlasst den Benutzer zum Warten.

© H. Peter Gumm, Philipps-Universität Marburg

Queue Implementierung mit Array

- Behälter: Ein Array *theQueue*
  - Zeiger: *front*, *back*
    - *front* zeigt auf erstes Objekt
    - *back* auf den nächsten freien Platz
  - *enQ*(Object e)
    - theQueue[back] = e; back++;
  - *deQ*()
    - if (! isEmpty()) front++;
  - *isEmpty*()
    - front == back
- Problem
  - Bereich zwischen *front* und *back* wandert durch den Array
  - Array kann verlassen werden, auch wenn nur wenige Elemente gespeichert sind

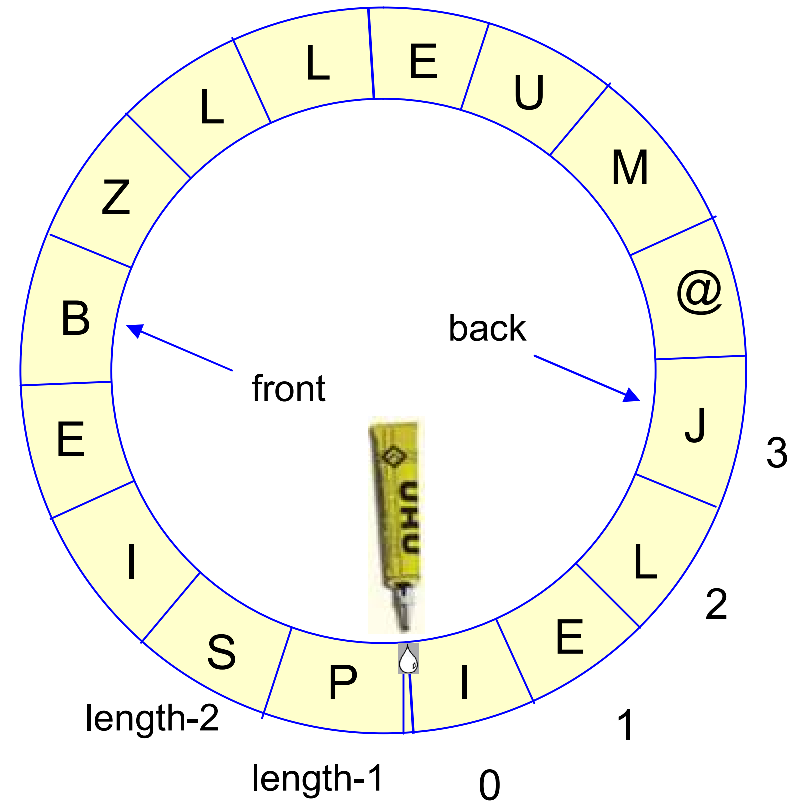
Nur wenige Elemente in der Queue, aber schon gefährlich nahe am Abgrund

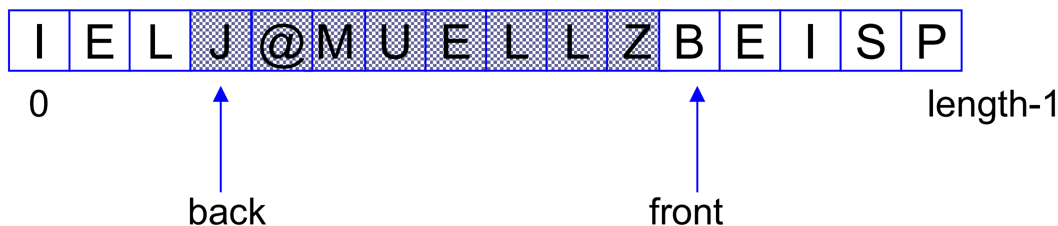


© H. Peter Gumm, Philipps-Universität Marburg

Zirkuläre Arrays

- Gedanklich: Verklebe Ende des Arrays mit seinem Anfang
- Mathematisch Berechne Array-Indizes modulo seiner Länge
- statt
  - front++
- rechne
  - front = (front+1)%length
- Aber wann ist der Array leer
  - front == back
  - kann bedeuten
    - voll
    - leer
  - Zusätzliche Boolesche Variable
    - empty
  - Wird von enQ/deQ ggf. gesetzt





empty: false

© H. Peter Gumm, Philipps-Universität Marburg

Implementierung der Queue

- Array mit Zeigern …
  - front : erstes Element
  - back: Position für das nächste zu speichernde Element
- … und
  - boolesche Variable : full
- Klasseninvariante …
  - front==back $\Leftrightarrow$ leer $\oplus$ voll
  - length $\geq$ 0
- … diese muss von
  - jedem Konstruktor
  - jeder Operation

  erhalten werden.

```
public class BoundedQueue{
// Der Behälter
  private Object[] theQueue;
  private int maxSize=100;

// Das vorderste Element:
  private int front=0;
// Die Position für das nächste Element:
  private int back=0;

// Falls front==back ist Queue voll oder leer
  private boolean full=false;

// Eine Klassen-Invariante
  private boolean invariante(){
    return
      ((front==back) == (isEmpty() ^ isFull()))
         && length() >= 0 ;
  }
/** Konstruktor für Queue mit Kapazität size */
  public BoundedQueue(int size){
    maxSize=size;
    theQueue = new Object[maxSize];
    assert invariante();
  }
```

enQ, deQ, getNext

- next()
  - „biegt“ den Array zu einem Ring
  - natürlich nur, falls wir exklusiv damit im Array navigieren
- enQ()
  - prüft full
  - setzt es evtl.
- deQ()
  - setzt: full=false
- length()
  - benötigt die mathematisch korrekte Version von „modulo“

```
private int next(int n){
     return (n+1)%maxSize;
}
public void enQ(Object o) throws Exception{
  if (full) throw new Exception("Queue Full");
  else{
    theQueue[back] = o;
    back = next(back);
    full = (front == back);
   }
  assert invariante();
}
public void deQ() throws Exception{
  if (isEmpty()) throw new Exception("Queue Empty");
  else{
    front = next(front);
    full = false;
  }
  assert invariante();
}
public boolean isEmpty(){
  return !full && front==back;
}
public int length(){
  return full? maxSize : ((back-front+maxSize)% maxSize);
}
```


© H. Peter Gumm, Philipps-Universität Marburg

Anwendung von Queues

- Eine einfache Simulation
  - Ein Laden hat k Kassen
  - Zu zufälligen Zeiten kommen Kunden
    - im Schnitt n proStunde
    - aber mal mehr – mal weniger
  - Sie laden ihre Einkaufswagen
    - mit 1 – 45 Artikeln
    - zufällig verteilt
    - gehen dann zur Kasse mit der kürzesten Schlange
  - Kassiererin braucht
    - 1 sec pro Artikel
    - 9 sec zum Kassieren
  - Wieviele Kassen müssen besetzt sein, damit
    - 90 % der Kunden nicht länger als 5 min warten müssen ?

BlueJ: Simulation

Project Edit Tools View Help

New Class...

Compile

Kunde

BoundedQueue

Supermarkt

tegut: Supermarkt

Creating object... D

BlueJ: Create Object

// Konstruktor

Supermarkt(int anzahlKassen, int kundenProStunde)

Name of Instance: tegut

new Supermarkt(3 , int anzahlKassen

250) int kundenProStunde

Ok Cancel

BlueJ: Method Call

void simulate(int n)

supermar_1.simulate (2*3600)

Ok Cancel

© H. Peter Gumm, Philipps-Universität Marburg

Codefragment – und Ausgabe

```
public void simulate(int n){
  while( zeit < n){
    if ( Math.random()*3600 < kundenProStunde ){
       // kommt ein Kunde
       kundenzaehler++;
       int wasKauftEr = (int)(Math.random()*(maxArt-minArt)) + minArt;
       Kunde kunde = new Kunde(kundenzaehler,zeit,wasKauftEr);
       System.out.print(uhrZeit()+": Neuer Kunde"+kunde.kdNr
                        +" mit " + kunde.noOfItems + " Artikeln ");
       // schicke ihn an die kürzeste Schlange
       int i = kuerzesteSchlange();
       try { kassen[i].enQ(kunde);
           System.out.println(" an Kasse "+i);
     }catch (Exception e) {};
    }
    // Jede Kassiererin arbeitet eine Sekunde
    for(int k=0; k < anzahlKassen; k++){
        kassieren(kassen[k]);
    }
    zeit++;
  }
```

BlueJ: Terminal Window

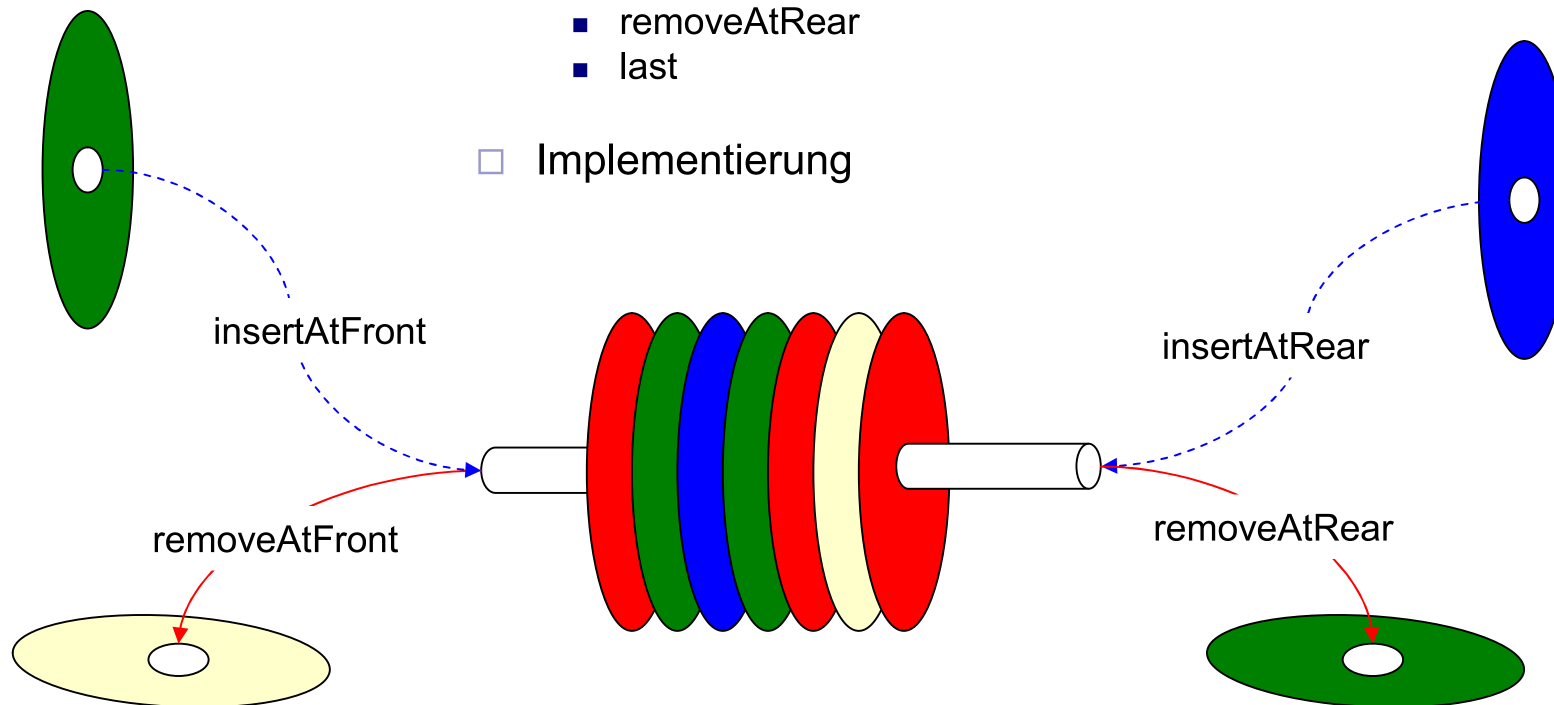
Options

```
01:57.38 Uhr Kunde 496 fertig. Wartezeit : 19
01:57.55 Uhr : Neuer Kunde499 mit 60 Artikeln  an Kasse 1
01:57.59 Uhr : Neuer Kunde500 mit 56 Artikeln  an Kasse 1
01:57.59 Uhr Kunde 495 fertig. Wartezeit : 73
01:58.16 Uhr Kunde 498 fertig. Wartezeit : 40
01:58.18 Uhr : Neuer Kunde501 mit 47 Artikeln  an Kasse 0
01:58.23 Uhr Kunde 497 fertig. Wartezeit : 62
01:58.38 Uhr : Neuer Kunde502 mit 35 Artikeln  an Kasse 2
01:58.41 Uhr : Neuer Kunde503 mit 39 Artikeln  an Kasse 0
01:58.55 Uhr Kunde 499 fertig. Wartezeit : 60
01:59.00 Uhr : Neuer Kunde504 mit 34 Artikeln  an Kasse 1
01:59.05 Uhr Kunde 501 fertig. Wartezeit : 47
01:59.13 Uhr Kunde 502 fertig. Wartezeit : 35
01:59.15 Uhr : Neuer Kunde505 mit 52 Artikeln  an Kasse 2
01:59.21 Uhr : Neuer Kunde506 mit 14 Artikeln  an Kasse 0
01:59.45 Uhr Kunde 503 fertig. Wartezeit : 64
01:59.52 Uhr Kunde 500 fertig. Wartezeit : 113
Anzahl der Kunden 506
Maximale WarteZeit 241
Maximale Länge einer Schlange 6
```


© H. Peter Gumm, Philipps-Universität Marburg

DeQue

- Double ended Queue
  - kombiniert Stack und Queue
    - insertAtFront = push
    - removeAtFront = pop = deQ
    - first = top = front
    - insertAtRear = enQ
  - Zusätzlich
    - removeAtRear
    - last
  - Implementierung



© H. Peter Gumm, Philipps-Universität Marburg