Script generated by TTT

Title: Seidl: Informatik_1 (28.11.2012)

Date: Wed Nov 28 15:16:05 CET 2012

Duration: 91:04 min

Pages: 30

11.3 Schlangen (Queues)

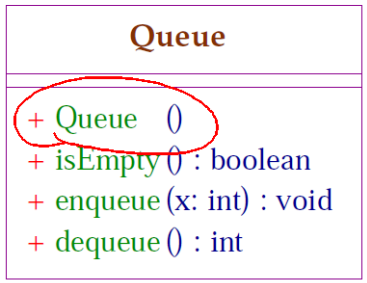
(Warte-) Schlangen verwalten ihre Elemente nach dem FIF0-Prinzip (First-In-First-Out).

Operationen:

`boolean isEmpty()`	:	testet auf Leerheit;
`int dequeue()`	:	liefert erstes Element;
`void enqueue(int x)`	:	reiht x in die Schlange ein;
`String toString()`	:	liefert eine String-Darstellung.

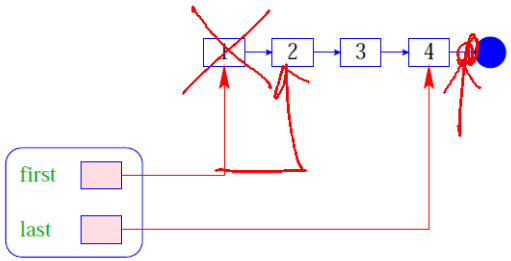
Weiterhin müssen wir eine leere Schlange anlegen können.

Modellierung:



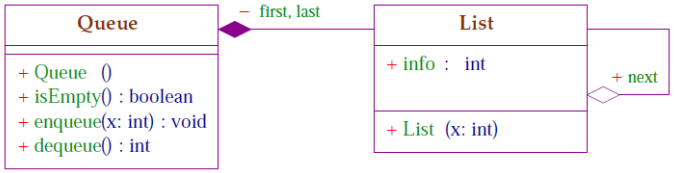
Erste Idee:

- Realisiere Schlange mithilfe einer Liste :



- `first` zeigt auf das nächste zu entnehmende Element;
- `last` zeigt auf das Element, hinter dem eingefügt wird.

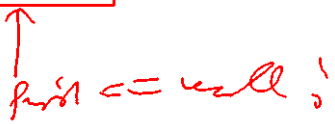
Modellierung:



Objekte der Klasse `Queue` enthalten zwei Verweise auf Objekte der Klasse `List`.

Implementierung:

```
public class Queue {
   private List first, last;
   // Konstruktor:
   public Queue() {
      first = last = null;
      }
   // Objekt-Methoden:
   public boolean isEmpty() {
      return List.isEmpty(first);
      }
   ...
```




```
   public int dequeue() {
      int result = first.info;
      if (last == first) last = null;
      first = first.next;
      return result;
      }
   public void enqueue(int x) {
      if (first == null) first = last = new List(x);
      else { last.insert(x); last = last.next; }
      }
   public String toString() {
      return List.toString(first);
      }
} // end of class Queue
```


- Die Implementierung ist wieder sehr einfach.
- ... nutzt ein paar mehr Features von `List` aus;
- ... die Listen-Elemente sind evt. über den gesamten Speicher verstreut

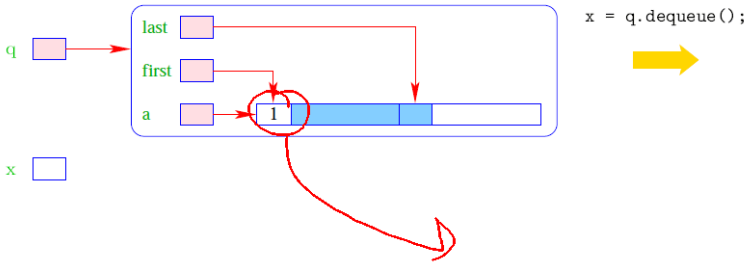
  $\Longrightarrow$ führt zu schlechtem ↑Cache-Verhalten des Programms !

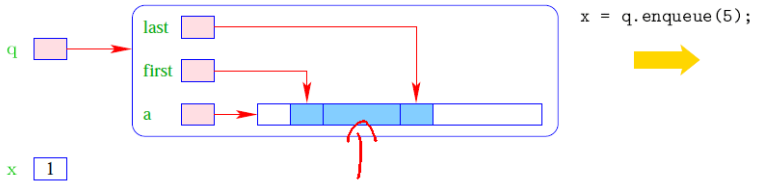
- Die Implementierung ist wieder sehr einfach.
- ... nutzt ein paar mehr Features von `List` aus;
- ... die Listen-Elemente sind evt. über den gesamten Speicher verstreut

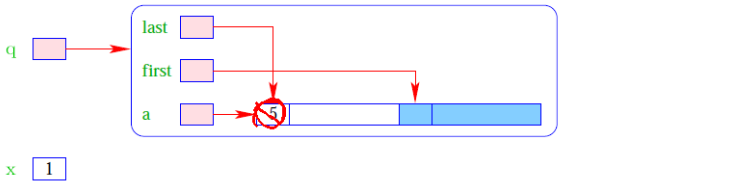
  $\Longrightarrow$ führt zu schlechtem ↑Cache-Verhalten des Programms !

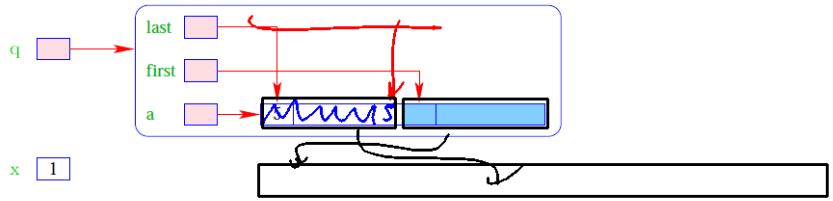
Zweite Idee:

- Realisiere die Schlange mithilfe eines Felds und zweier Pointer, die auf das erste bzw. letzte Element der Schlange zeigen.
- Läuft das Feld über, ersetzen wir es durch ein größeres.

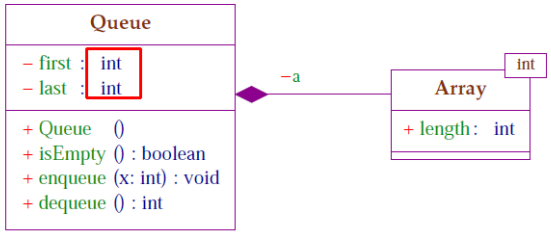








Modellierung:



Implementierung:

```
public class Queue {
   private int first, last;
   private int[] a;
// Konstruktor:
   public Queue() {
      first = last = -1;
      a = new int[4];
      }
// Objekt-Methoden:
   public boolean isEmpty() { return first==-1; }
   public String toString() {...}
   ...
```


Implementierung von `enqueue()`:

- Falls die Schlange leer war, muss `first` und `last` auf `0` gesetzt werden.
- Andernfalls ist das Feld `a` genau dann voll, wenn das Element `x` an der Stelle `first` eingetragen werden sollte.
- In diesem Fall legen wir ein Feld doppelter Größe an.

  Die Elemente `a[first], a[first+1],..., a[a.length-1]`, `a[0], a[1],..., a[first-1]` kopieren wir nach `b[0],..., b[a.length-1]`.
- Dann setzen wir `first = 0;`, `last = a.length` und `a = b;`
- Nun kann `x` an der Stelle `a[last]` abgelegt werden.

Implementierung von `enqueue()`:

- Falls die Schlange leer war, muss `first` und `last` auf `0` gesetzt werden.
- Andernfalls ist das Feld `a` genau dann voll, wenn das Element `x` an der Stelle `first` eingetragen werden sollte.
- In diesem Fall legen wir ein Feld doppelter Größe an.

  Die Elemente `a[first], a[first+1],..., a[a.length-1]`, `a[0], a[1],..., a[first-1]` kopieren wir nach `b[0],..., b[a.length-1]`.
- Dann setzen wir `first = 0;`, `last = a.length` und `a = b;`
- Nun kann `x` an der Stelle `a[last]` abgelegt werden.


```
public void enqueue(int x) {
   if (first==-1) {
      first = last = 0;
   } else {
   int n = a.length;
   last = (last+1)%n;
   if (last == first) {
      b = new int[2*n];
      for (int i=0; i<n; ++i) {
         b[i] = a[(first+i)%n];
         } // end for
      first = 0; last = n; a = b;
      } }// end if and else
   a[last] = x;
   }
```

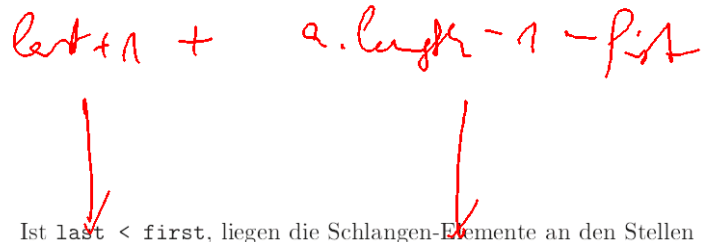

Implementierung von `dequeue()`:

- Falls nach Entfernen von `a[first]` die Schlange leer ist, werden `first` und `last` auf `-1` gesetzt.
- Andernfalls wird `first` um 1 (modulo der Länge von `a`) inkrementiert.
- Für eine evt. Freigabe unterscheiden wir zwei Fälle.
- Ist `first < last`, liegen die Schlangen-Elemente an den Stellen `a[first],...,a[last]`.

  Sind dies höchstens `n/4`, werden sie an die Stellen `b[0],...,b[last-first]` kopiert.


```
public void enqueue(int x) {
   if (first==-1) {
      first = last = 0;
   } else {
   int n = a.length;
   last = (last+1)%n;
   if (last == first) {
      b = new int[2*n];
      for (int i=0; i<n; ++i) {
         b[i] = a[(first+i)%n];
         } // end for
      first = 0; last = n; a = b;
      } }// end if and else
   a[last] = x;
   }
```

```
public int dequeue() {
   int result = a[first];
   if (last == first) {
      first = last = -1;
      return result;
      }
   int n = a.length;
   first = (first+1)%n;
   int diff = last-first;
   if (diff>0 && diff<n/4) {
      int[] b = new int[n/2];
      for(int i=first; i<=last; ++i)
         b[i-first] = a[i];
      last = last-first;
      first = 0; a = b;
   }  else ...
```

- Ist `last < first`, liegen die Schlangen-Elemente an den Stellen `a[0],...,a[last]` und `a[first],...,a[a.length-1]`.
  Sind dies höchstens `n/4`, werden sie an die Stellen `b[0],...,b[last]` sowie `b[first-n/2],...,b[n/2-1]` kopiert.
- `first` und `last` müssen die richtigen neuen Werte erhalten.
- Dann kann `a` durch `b` ersetzt werden.


```
   if (diff<0 && diff+n<n/4) {
      int[] b = new int[n/2];
      for(int i=0; i<=last; ++i)
         b[i] = a[i];
      for(int i=first; i<n; i++)
         b[i-n/2] = a[i];
      first = first-n/2;
      a = b;
      }
   return result;
   }
```


Zusammenfassung:

- Der Datentyp `List` ist nicht sehr abstract, dafür extrem flexibel
  $\Longrightarrow$ gut geeignet für rapid prototyping.
- Für die nützlichen (eher) abstrakten Datentypen `Stack` und `Queue` lieferten wir zwei Implementierungen:

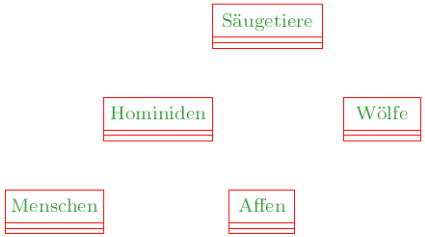
Technik	Vorteil	Nachteil
`List`	einfach	nicht-lokal
int[]	lokal	etwas komplexer

- Achtung: oft werden bei diesen Datentypen noch weitere Operationen zur Verfügung gestellt.

12 Vererbung

Beobachtung:

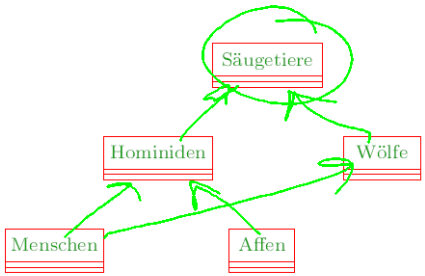
- Oft werden mehrere Klassen von Objekten benötigt, die zwar ähnlich, aber doch verschieden sind.



12 Vererbung

Beobachtung:

- Oft werden mehrere Klassen von Objekten benötigt, die zwar ähnlich, aber doch verschieden sind.

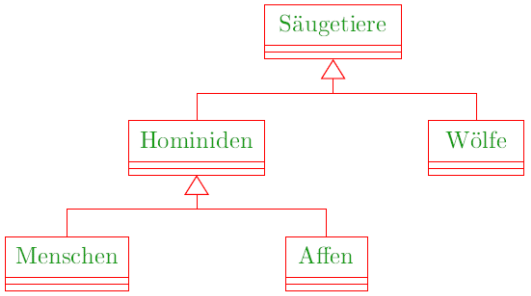


Idee:

- Finde Gemeinsamkeiten heraus!
- Organisiere in einer Hierarchie!
- Implementiere zuerst was allen gemeinsam ist!
- Implementiere dann nur noch den Unterschied!

$\Longrightarrow$ inkrementelles Programmieren

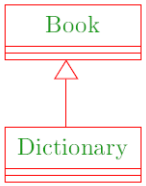
$\Longrightarrow$ Software Reuse



Prinzip:

- Die Unterklasse verfügt über die Members der Oberklasse und eventuell auch noch über weitere.
- Das Übernehmen von Members der Oberklasse in die Unterklasse nennt man Vererbung (oder inheritance).

Beispiel:



Implementierung:

```
public class Book {
   protected int pages;
   public Book() {
      pages = 150;
   }
   public void page_message() {
      System.out.print("Number of pages:\t"+pages+"\n");
      }
   } // end of class Book
   ...
```



```
public class Dictionary extends Book {
   private int defs;
   public Dictionary(int x) {
      pages = 2*pages;
      defs = x;
      }
   public void defs_message() {
      System.out.print("Number of defs:\t\t"+defs+"\n");
      System.out.print("Defs per page:\t\t"+defs/pages+"\n");
      }
   } // end of class Dictionary
```