```
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 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:

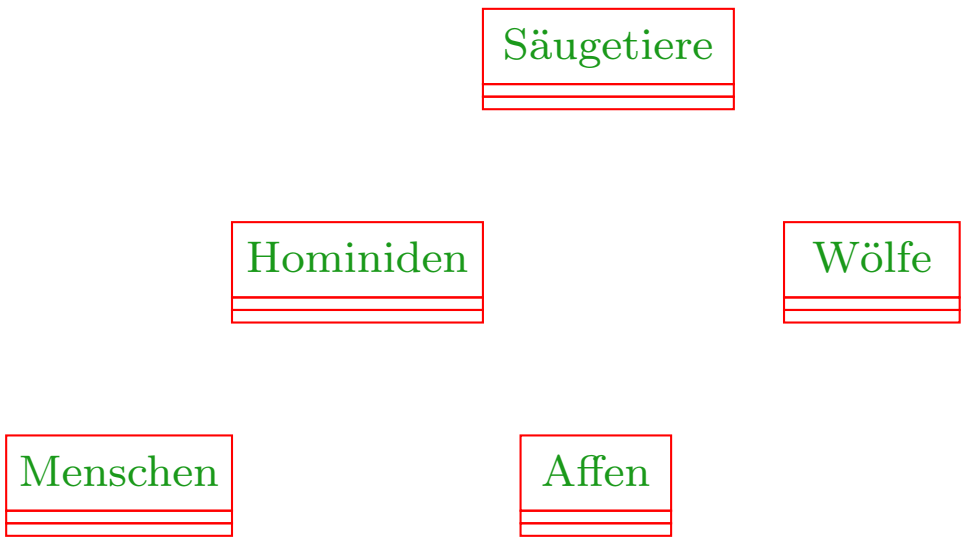
| Technik | Vorteil | Nachteil |
|---|---|---|
| `List` | einfach | nicht-lokal |
| int[] | lokal | etwas komplexer |

- 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.

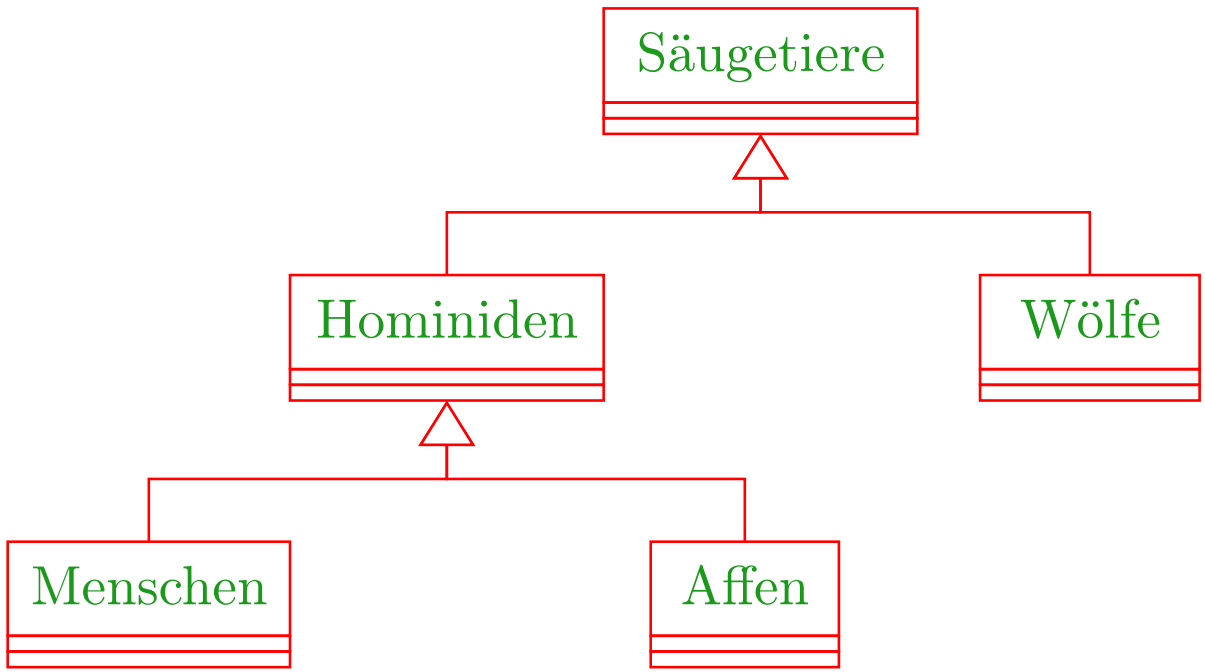


## Idee:

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

$\Longrightarrow$ inkrementelles Programmieren

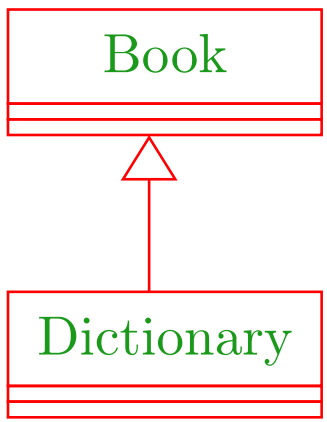
$\Longrightarrow$ Software Reuse


Säugetiere
Hominiden
Wölfe
Menschen
Affen

## 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:



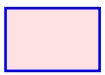
# Implementierung:

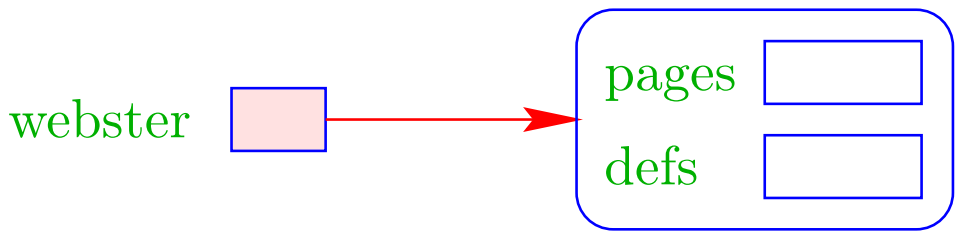
```
public class Book {
   protected int pages;
   public Book() {
      pages = 150;
   }
   public void page_message() {
      System.out.print("Number of pages:\t"+pages+"\n");
      }
   } // end of class Book
   ...
```

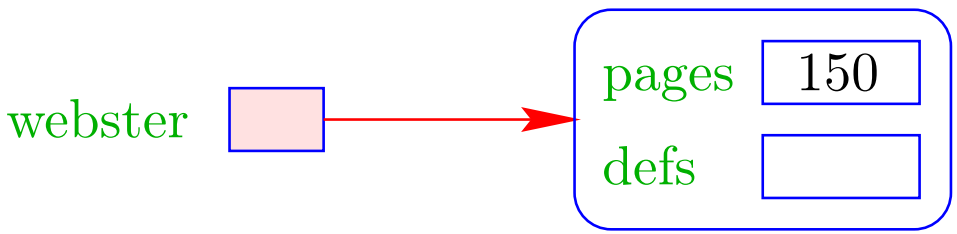
```
public class Dictionary extends Book {
   private int defs;
   public Dictionary(int x) {
      pages = 2*pages;
      defs = x;
      }
   public void defs_message() {
      System.out.print("Number of defs:\t\t"+defs+"\n");
      System.out.print("Defs per page:\t\t"+defs/pages+"\n");
      }
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

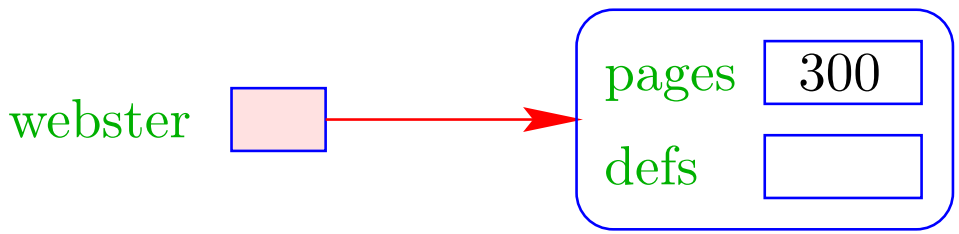
- `class` A `extends` B `{ ... }` deklariert die Klasse A als Unterklasse der Klasse B.
- Alle Members von B stehen damit automatisch auch der Klasse A zur Verfügung.
- Als `protected` klassifizierte Members sind auch in der Unterklasse sichtbar.
- Als `private` deklarierte Members können dagegen in der Unterklasse nicht direkt aufgerufen werden, da sie dort nicht sichtbar sind.
- Wenn ein Konstruktor der Unterklasse A aufgerufen wird, wird implizit zuerst der Konstruktor B`()` der Oberklasse aufgerufen.

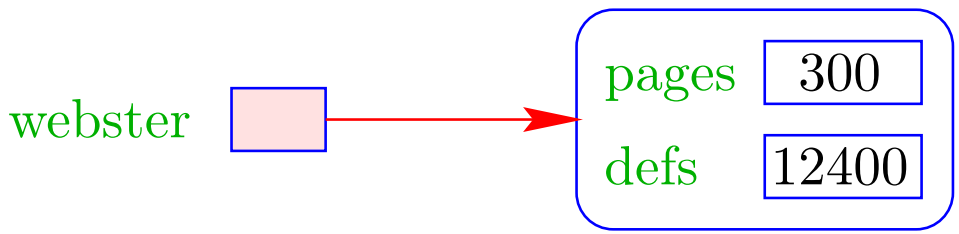
```
Dictionary webster = new Dictionary(12400);
```
liefert:

webster 


webster
pages
defs


webster
pages
150
defs


webster
pages
300
defs


webster
pages
300
defs
12400

```
public class Words {
   public static void main(String[] args) {
      Dictionary webster = new Dictionary(12400);
      webster.page_message();
      webster.defs_message();
   } // end of main
} // end of class Words
```

- Das neue Objekt `webster` enthält die Attribute `pages` und `defs`, sowie die Objekt-Methoden `page_message()` und `defs_message()`.
- Kommen in der Unterklasse nur weitere Members hinzu, spricht man von einer is_a-Beziehung. (Oft müssen aber Objekt-Methoden der Oberklasse in der Unterklasse umdefiniert werden.)

- Die Programm-Ausführung liefert:

```
Number of pages:        300
Number of defs:         12400
Defs per page:          41
```