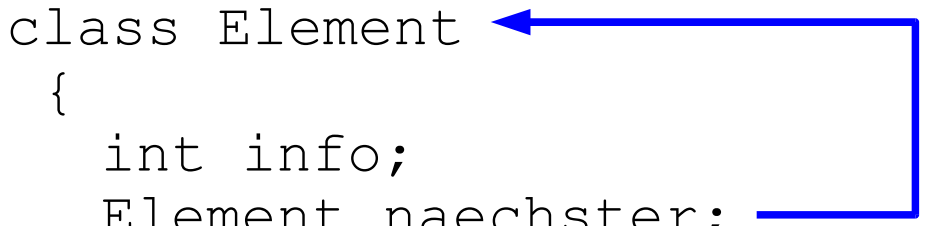
3.3. Rekursive Datentypen



```
class Element
 {
   int info;
   Element naechster;
 }

...

Element element = new Element();

element.info = 1;
element.naechster = new Element();
element.naechster.info = 2;
```

Erläuterung:

- Objekte einer Klasse können auch Verweise auf Objekte derselben Klasse als Variable enthalten.
- Eine solche Klasse nennt man *rekursiv*.

Beispiel: Definition einer Menge

```
public class Menge
 {
 private Element erstesElement;

  public int groesse           ( )          { ... }
  public boolean istEnthalten  ( int info ) { ... }
  public boolean fuegeEin      ( int info ) { ... }
  public boolean entferne      ( int info ) { ... }
 }
```

→ Was die einzelnen Methoden ungefähr tun sollen, sollte eigentlich selbsterklärend sein (kommt auf den nächsten Folien)

Vorgriff:

- Das `private` anstelle des gewohnten `public` sorgt dafür, dass `erstesElement` nur von den Methoden von `Menge` angesprochen werden darf.

Methoden für die Klasse Menge

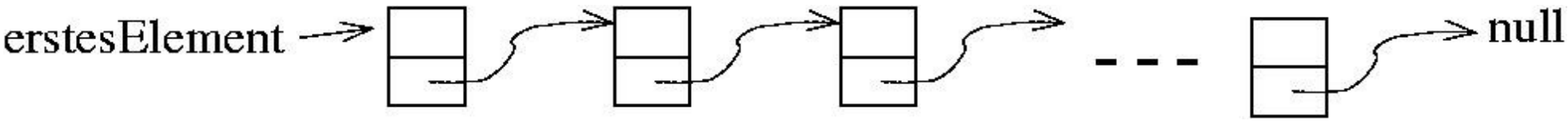
`groesse`:
- Retourniere die Anzahl der Elemente in der Menge

`istEnthalten`:
- Überprüfe, ob das Element `info` in der Menge enthalten ist

`fuegeEin`:
- Füge das Element `info` in die Menge ein
  - Wenn `info` schon in der Menge enthalten ist, wird es nicht noch einmal eingefügt.
  - Der Rückgabewert ist `true`, wenn `info` noch nicht in der Menge enthalten ist (und somit eingefügt wird).

`entferne`:
- Entferne `info` aus der Menge
  - Wenn `info` nicht in der Menge enthalten ist, wird es natürlich nicht entfernt.
  - Der Rückgabewert ist `true`, wenn `info` in der Menge enthalten war (und somit entfernt wurde).

Schematische Darstellung



- Die einzelnen Elemente der Menge bilden eine Art Kette, die durch Verweis `naechster` zusammengehalten wird.
- Das Ende dieser Kette wird sinnvollerweise durch `naechster==null` angezeigt.

  → Da für den letzten Wert der Kette noch kein Nachfolge-Objekt angelegt worden ist
- Eine Kette dieser Art heißt in der Informatik *Liste*.

Methode `groesse`

```
public int groesse ()
  {
    Element x = erstesElement;
    int rueckgabe  = 0;

    while ( x != null )
     {
       x = x.naechster;
       rueckgabe++;
     }
    return rueckgabe;
 }
```

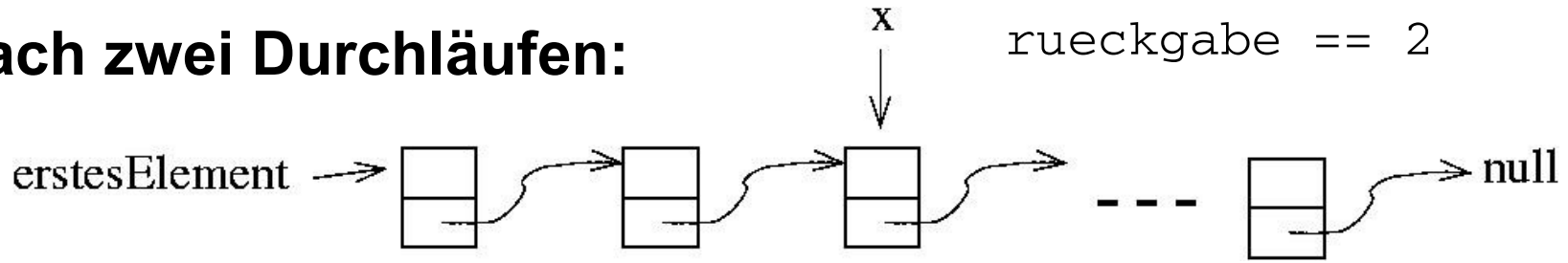
bis `x` das letzte `Element` erreicht hat

`x` hüpft von einem `Element` zum nächsten

Bei jedem Hüpfer wird `rueckgabe` um 1 erhöht

Nach zwei Durchläufen:

`rueckgabe == 2`



Methode `istEnthalten`

```
public boolean istEnthalten ( int info )
 {
   Element x = erstesElement;
   while ( x != null )
    {
      if ( x.info == info )
         return true;
      x = x.naechster;
    }
   return false;
 }
```

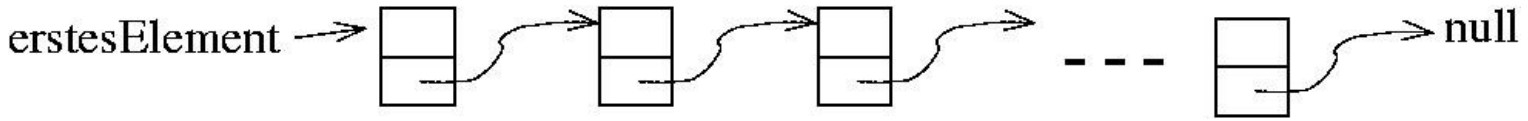
bis `x` den gesuchten Wert `info` enthält

`x` hüpft von einem `Element` zum nächsten

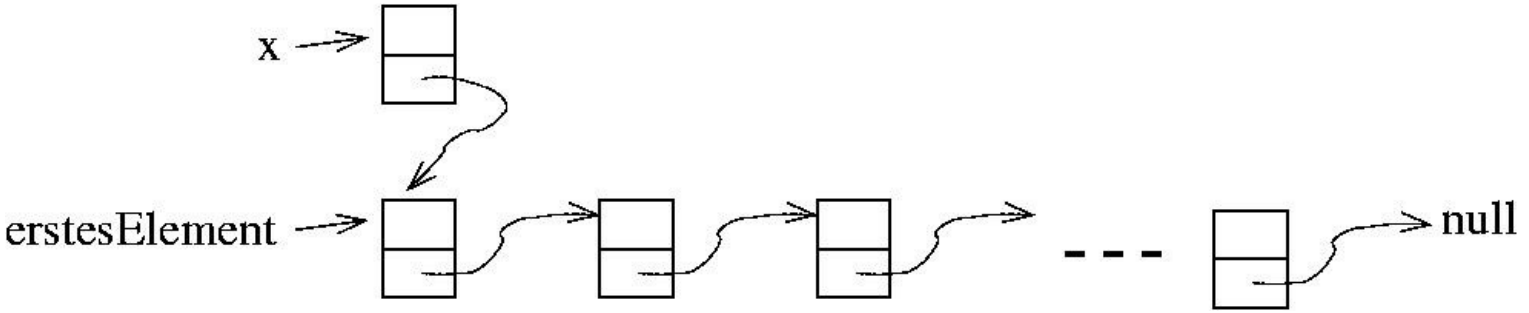
oder der gesuchte Wert `info` nicht gefunden wurde (Ende der Liste)

`fuegeEin`: Veranschaulichung

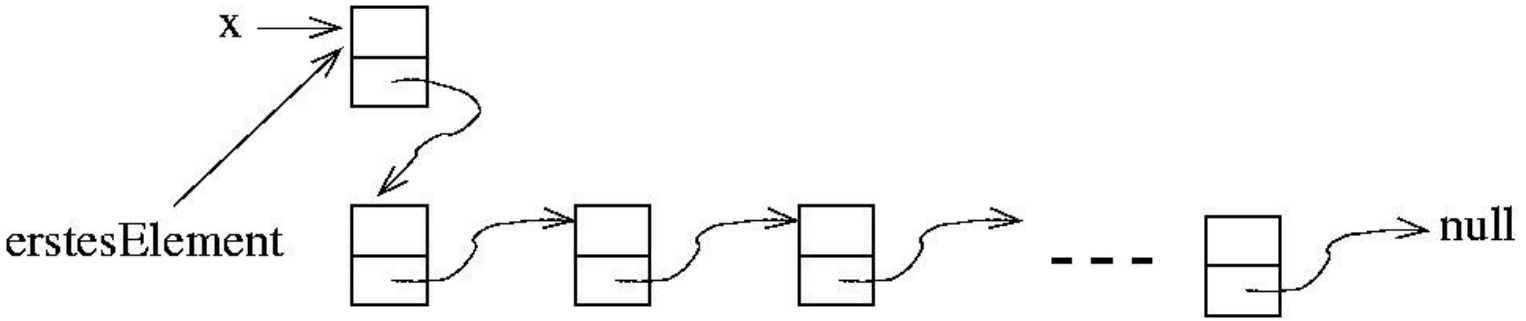
- **Ausgangsliste:**



- **Nach** `x.naechster = erstesElement`**:**



- **Nach** `erstesElement = x`**:**



Methode **fuegeEin**

```
public boolean fuegeEin (int info)
 {
   if ( istEnthalten(info) )
      return false;

   Element x = new Element();
   x.info = info;
   x.naechster = erstesElement;

   erstesElement = x;
   return true;
 }
```

Wenn `info` schon enthalten ist, wird nichts getan

ansonsten wird ein neues Element `x` angelegt

das `info` enthält

und auf das alte erste Element zeigt

`x` wird dann zum neuen ersten Element

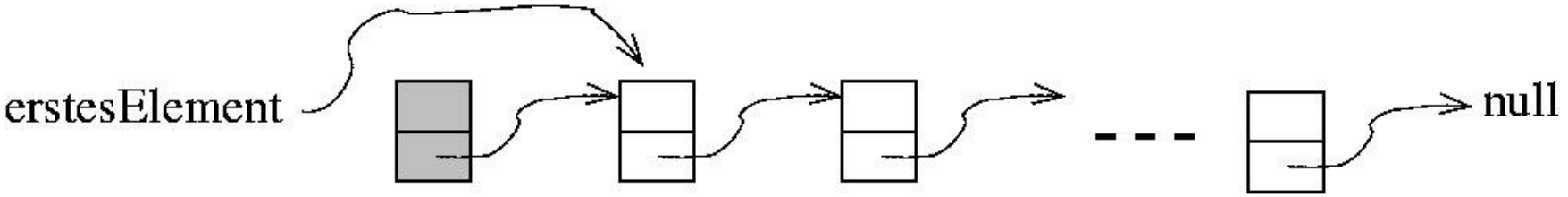
Methode `entferne`

- Die Idee ist, das Element mit der gesuchten `info` einfach aus der Liste auszukoppeln.
  - Hinterher gibt es dann keinen Verweis mehr auf dieses Element.
  - *Erinnerung*: Ein solches Element wird früher oder später vom Garbage Collector weggeräumt.
- *Methodisches Problem*:
  - ◊ Um ein Element aus der Liste zu entkoppeln, muss man Komponente ”`naechster` seines Vorgängers” ändern.
  - ◊ Beim Durchlauf durch die Liste, um das Element zu finden, muss man also immer um ein Element zurückbleiben.
  - ◊ Falls das zu löschende Element das allererste ist, geht die Entkopplung ganz einfach:

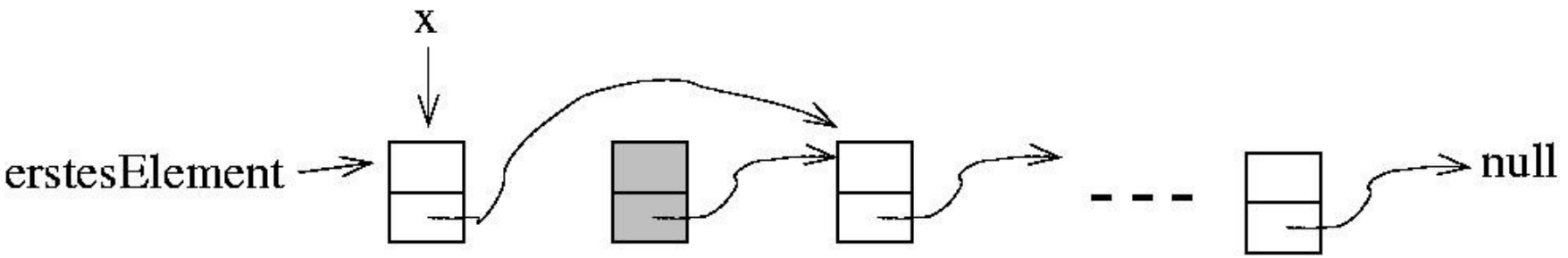
```
erstesElement = erstesElement.naechster;
```

`entferne`: Veranschaulichung

- **Nach** `erstesElement = erstesElement.naechster`
  im Fall `erstesElement.info == info:`



- **Nach** `x.naechster = x.naechster.naechster`
  im Fall `x.naechster.info == info`
  - hier schon im ersten Durchlauf der `while`–Schleife



Methode `entferne`

```
public boolean entferne ( int info )
 {
   if ( erstesElement == null )
      return false;

   if ( erstesElement.info == info ) {
      erstesElement = erstesElement.naechster;
      return true;
   }

   Element x = erstesElement;
   while ( x.naechster != null ) {
      if ( x.naechster.info == info ) {
         x.naechster = x.naechster.naechster;
         return true;
      }
      x = x.naechster;
   }
   return false;
 }
```

aus einer leeren Liste kann man nichts entfernen

erstes Element entfernen

iteriere über alle nächsten Elemente

wenn das nächste Element das gesuchte ist

wird das übernächste Element zum nächsten gemacht

Andere mögliche Methoden

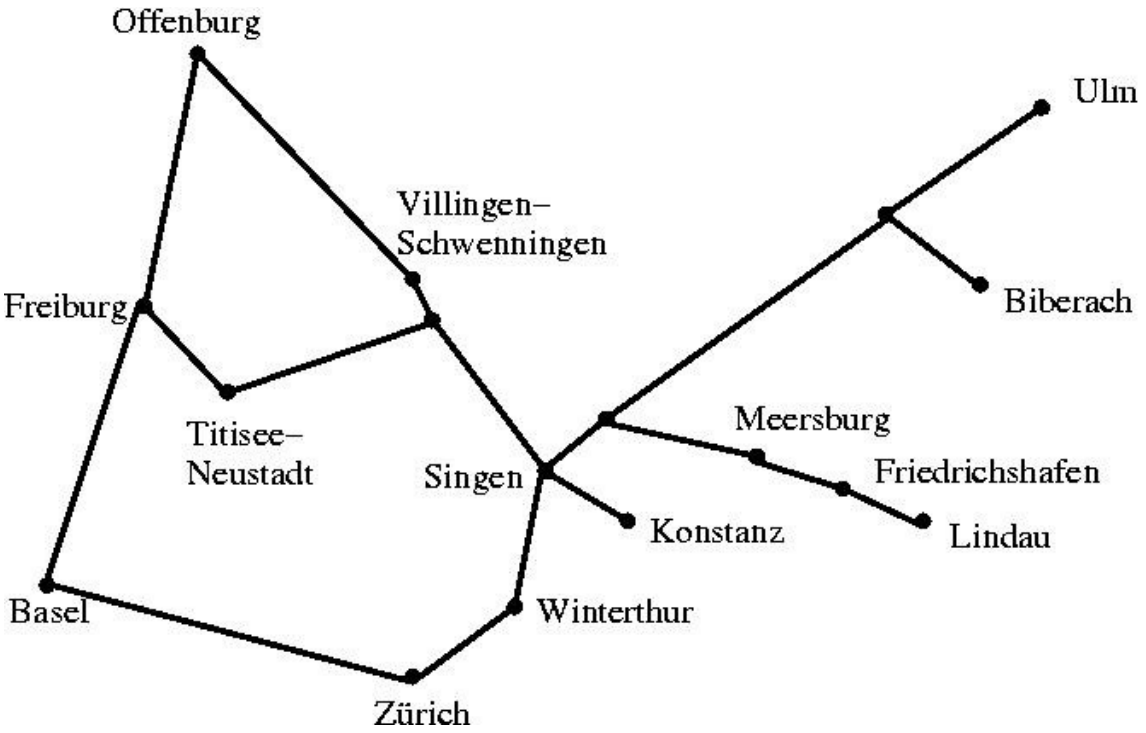
Es sind noch weitere Methoden denkbar, die auf Listen oder Mengen operieren:

- Vermischen zweier Listen: `merge` bzw. `union`
- Schnittmenge zweier Mengen: `intersect`
- Eine Liste an eine andere hängen: `append`
- Eine Liste sortieren: `sort`
- Eine Liste ausgeben: `print`

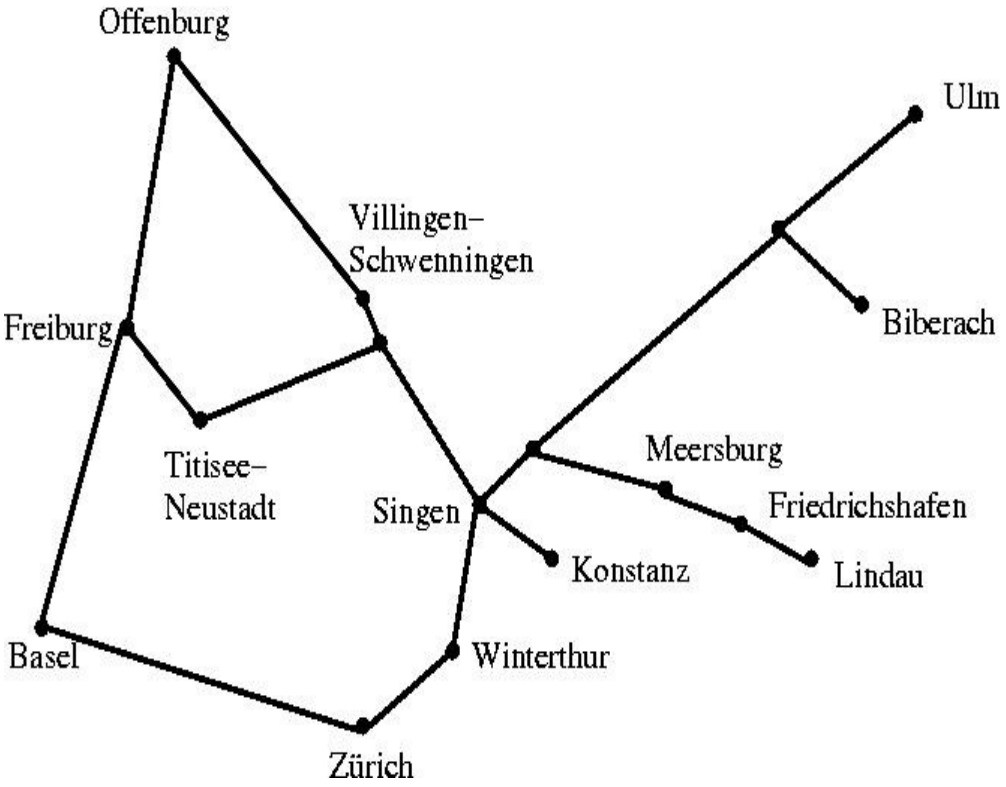
Komplexere Strukturen

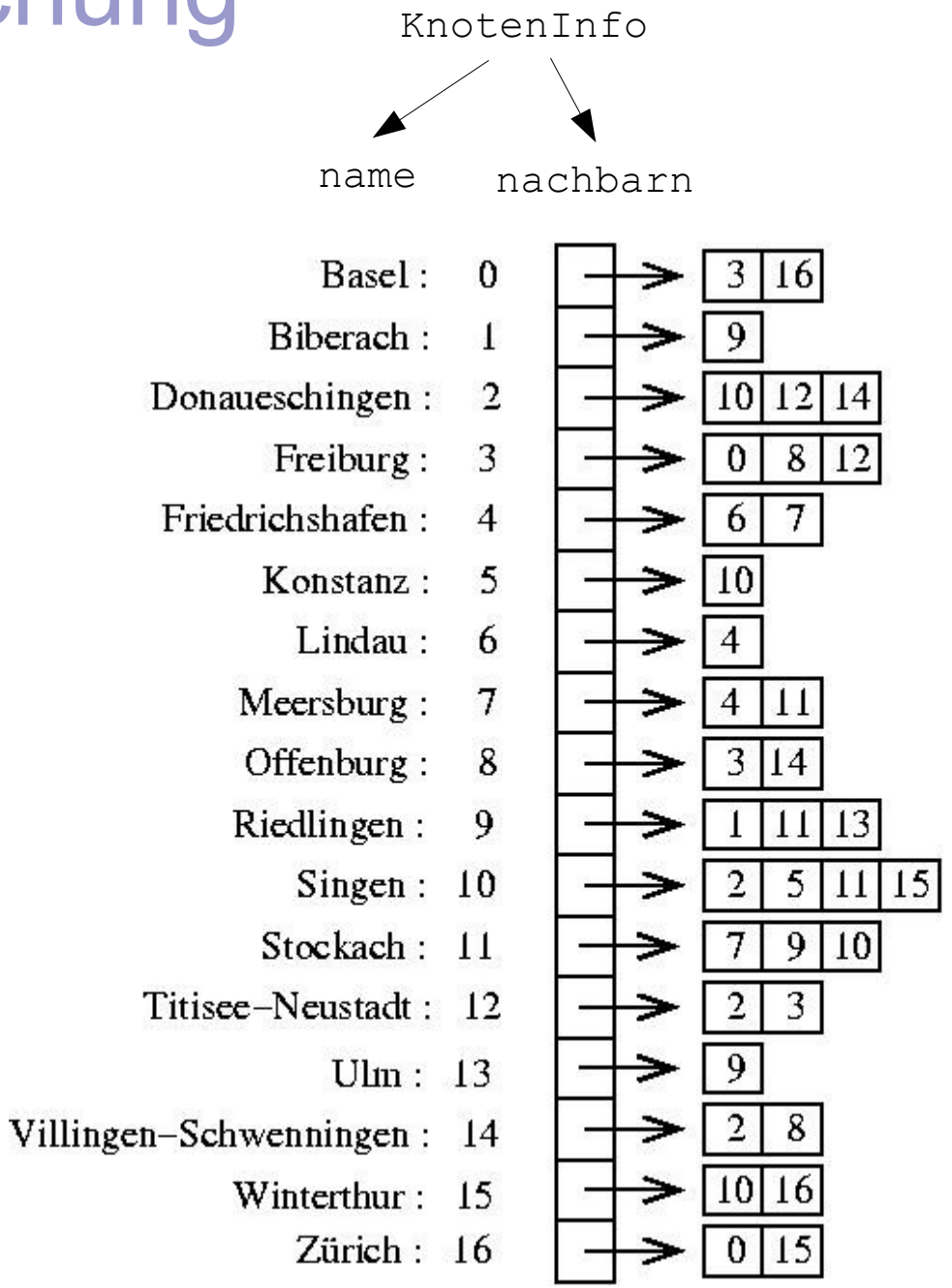
- **Beispiel:** Autobahnnetz

- *Idee*:
  - Ein Array von Knotenpunkten
  - Für jeden Knotenpunkt eine Liste von Nachbarknotenpunkten.



Veranschaulichung





Realisierung

```
public class KnotenInfo
 {
   public String name;
   public Menge nachbarn;
 }

...

KnotenInfo[] autobahnnetz = new KnotenInfo[17];

autobahnnetz[0] = new KnotenInfo();
autobahnnetz[0].name = "Basel";
autobahnnetz[0].nachbarn.fuegeEin(3);
autobahnnetz[0].nachbarn.fuegeEin(16);

autobahnnetz[1] = new KnotenInfo();
autobahnnetz[1].name = "Biberach";
autobahnnetz[1].nachbarn.fuegeEin(9);
...
```

Garbage Collector überlisten

- Wir haben gesagt, dass man beliebig viel Speicherplatz erzeugen kann, ohne dass auch nur ein Stück davon unerreichbar wird.
- Bisher nicht klar, wie das möglich sein soll.
- Das geht mit Listen jetzt ganz einfach:

```
public class Element
 {
   Element naechstes;
 }

...

Element x; // == null

while ( true )
 {
   // Neues Element vorne in die Liste einfuegen
   Element y = new Element();
   y.naechstes = x;
   x = y;
 }
```