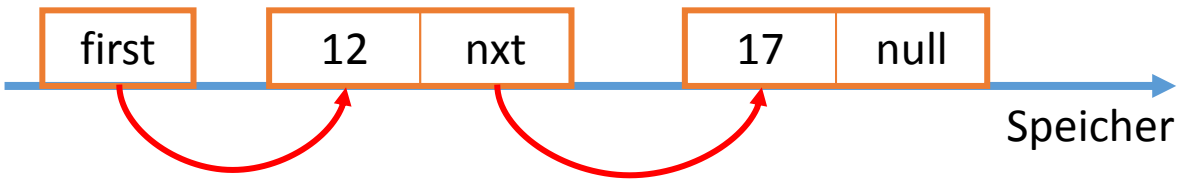
Liste traversieren

```
class Node {
   int value;
   Node next;

   Node (int v; Node n){
      value = v;
      next = n;
   }
}
```

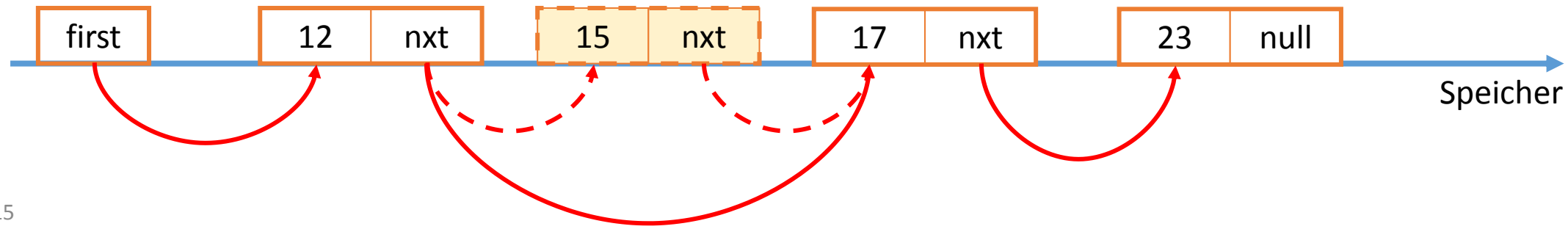



```
void outputI(Node n) {
   while (n != null){
      System.out.println(n.value);
      n = n.next;
   }
}
```

```
void outputR(Node n) {
   if (n != null){
      outputR(n.next);
      System.out.println(n.value);
   }
}
```

Sortierte Liste

```
class SortedList {
      Node first = null;

      // pre: verkettete Liste, erstes Element first
      // post: value in Liste aufsteigend sortiert eingefügt
      public void Insert (int value){..?..}
}
```



Invarianten für Knoten in einer sortierten Liste

Für jeden Knoten n gilt entweder

n == null

oder

n.next == null

oder

n.next != null && n.next.value >= n.value

Fälle : Einfügen von x

1. Fall: leere Liste

2. Fall: nichtleere Liste

a. v <= n.value für alle Knoten n
b. v > n.value für alle Knoten n
c. Es gibt einen Knoten n und Nachfolger m so dass
   x > n.value und x <= m.value

Stepwise Refinement

Einfache Programmiertechnik zum Lösen komplexer Probleme

Top-Down Ansatz

Stepwise Refinement

Formulierung einer groben Lösung mit Hilfe von

- Kommentaren
- fiktiven Funktionen

Wiederholte Verfeinerung

- Kommentare → Programmtext
- Fiktive Funktionen → Funktionsdefinitionen

Beispiel: Einfügen in sortierte Liste

Live Coding in der Vorlesung.

Auf separaten Slides als Präsentation.

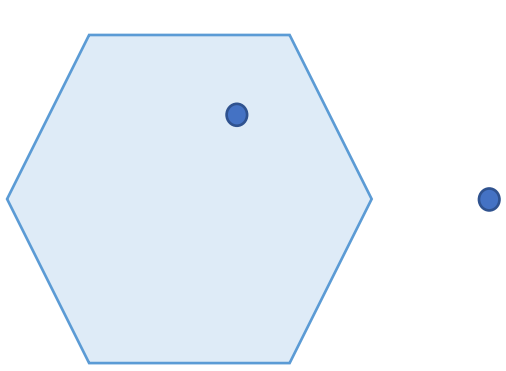
Einfügen

```
public void Insert (int value)
{
    if (first == null || value <= first.value)
        first = new Node(value, first);
    else {
        Node prev = first;
        Node v = prev.next;
        while (v != null && v.value > value){ // suche Nachfolger und Vorgänger
            prev = v;
            v = v.next;
        }
        prev.next = new Node(value, v); // Einfügen
    }
}
```

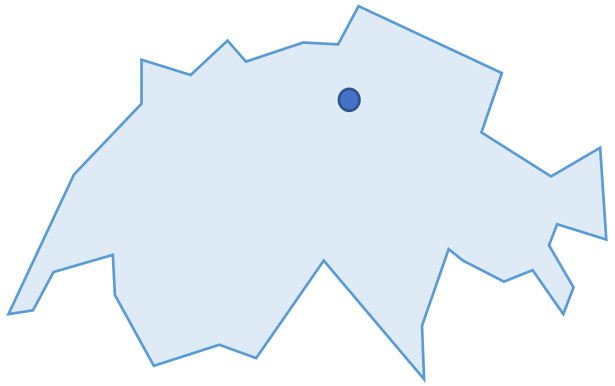
Fallstudie: Point-In-Polygon Algorithmus

Annahme: abgegrenztes Gebiet auf einer Landkarte.

- Repräsentation des Gebietes im Programm?
- Wie entscheiden wir effizient ob ein Punkt «innen» liegt?
- Wie füllen wir das Gebiet mit einer Farbe?



Ein konvexes Gebiet



Ein nicht konvexes Gebiet

Hintergrund: Jordankurven

Schnittfreie Polygone sind *Jordankurven* und zerlegen die Ebene in zwei Gebiete: das Innere und das Äussere

- Das Innere ist beschränkt und
- Das Äussere ist unbeschränkt.

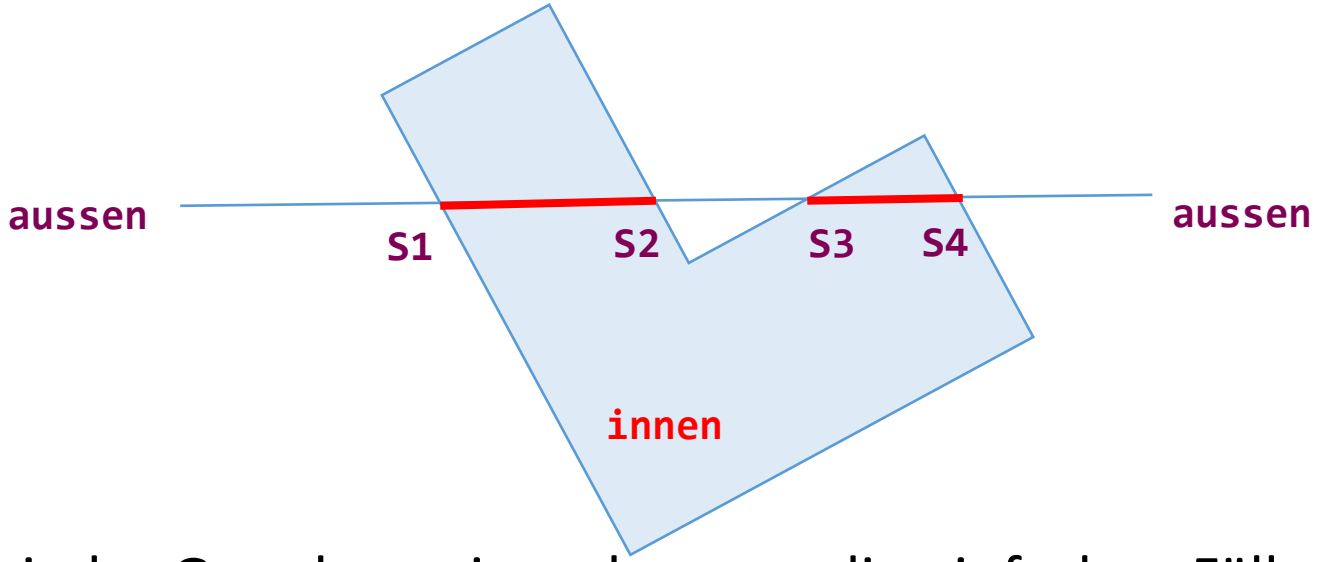
Der allgemeine strikt mathematische Beweis (algebraische Topologie, Abbildung der Einheitskugel) dieser Aussage ist relativ aufwändig und soll uns nicht weiter aufhalten.

Das kann man ausnutzen, indem man das Polygon mit Geraden schneidet. Man weiss, dass der unbeschränkte Teil der Geraden aussen liegt. Daher funktioniert die folgende Abzählmethode.

Abzählmethode

Zähle auf einer beliebigen Geraden (die den fraglichen Punkt enthält), von unendlich fern kommend, die Anzahl ***echter*** Schnittpunkte mit dem Polygon

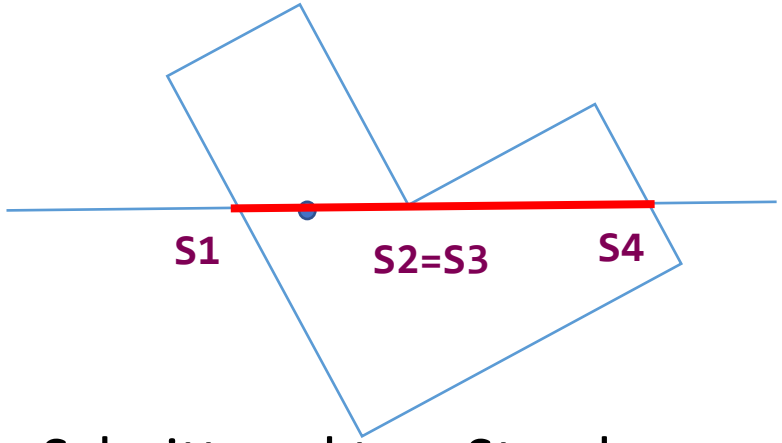
Alle Bereiche der Gerade zwischen ungeradzahligen und geradzahligen Schnittpunkten liegen innen

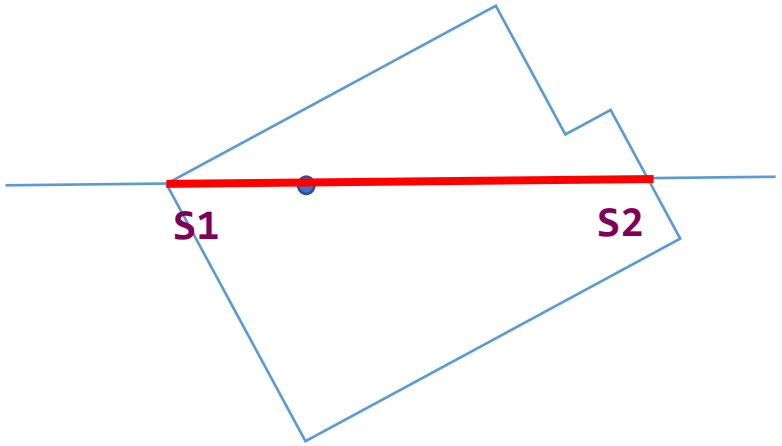


Das funktioniert mit jeder Geraden, wir suchen uns die einfachen Fälle raus (also horizontal oder vertikal)

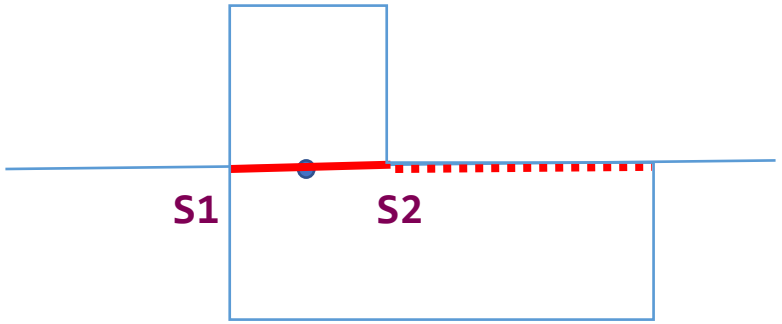
Spezialfälle

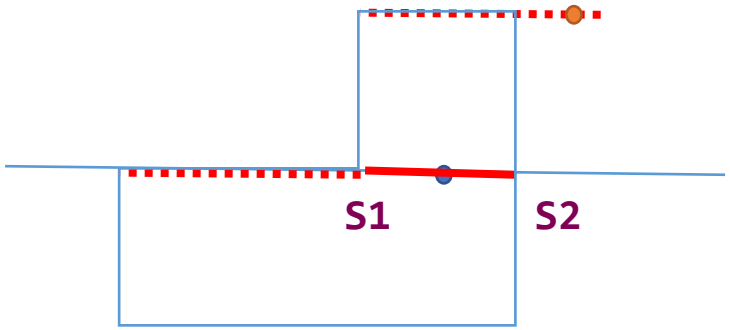
Schnittpunkt = Eckpunkt



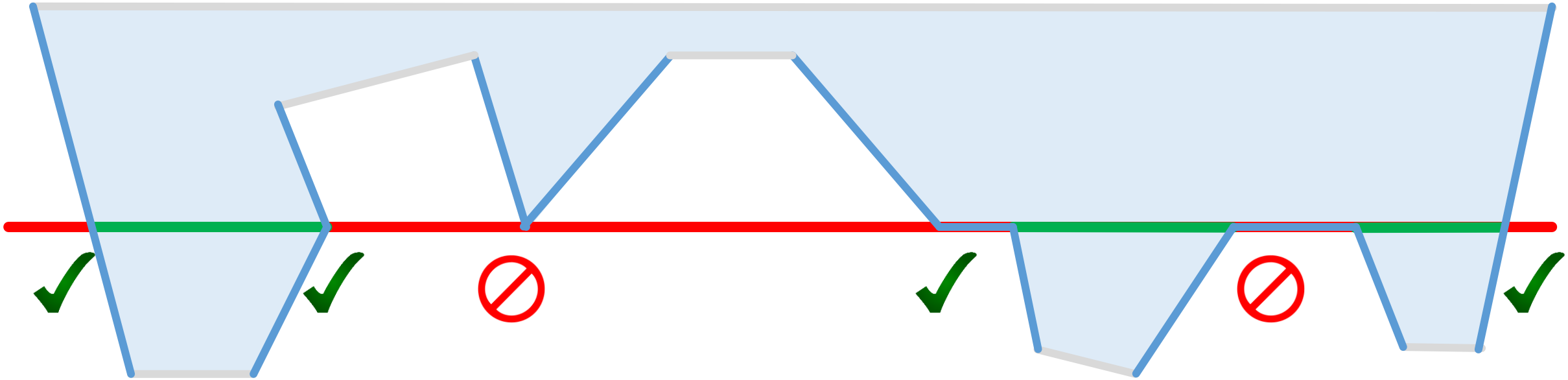


«Schnittpunkt» = Strecke





Echte vs. unechte Schnitte

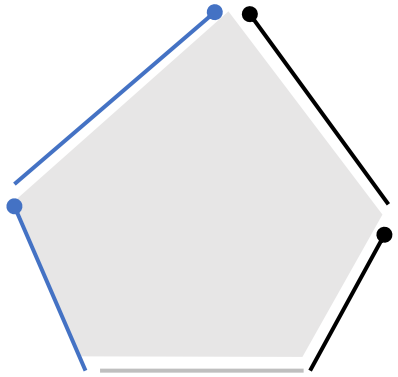


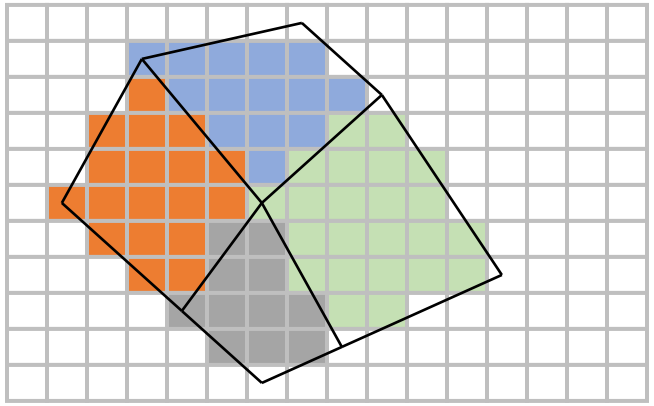
Was charakterisiert echte Schnitte?

Der Trick

Behandle Eckpunkte einer begrenzenden Kante lageabhängig

- Es werden z.B. nur die oberen Eckpunkte einer Kante mit vertikaler Komponente berücksichtigt
- horizontale Kanten werden ignoriert
- Für eindeutige Segmentierung bei Kachelung: Entscheidung für die Hinzunahme der einen oder anderen Richtung oben/unten, rechts/links (optional: behandle Kanten separat)





Implementation

Polygon

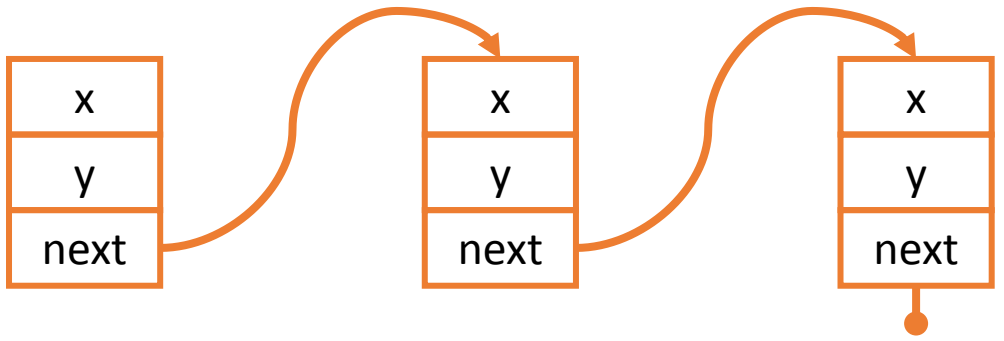
- Folge von Eckpunkten

Benötigte Methoden auf dem Polygon

- Hinzufügen von Eckpunkten
- Abzählen von Schnittpunkten des Polygons mit einer Geraden, dargestellt durch Punkt und Richtung (für die Entscheidung ob Punkt innen liegt)
- Aufzählen von inneren Punkten (für das Füllen)

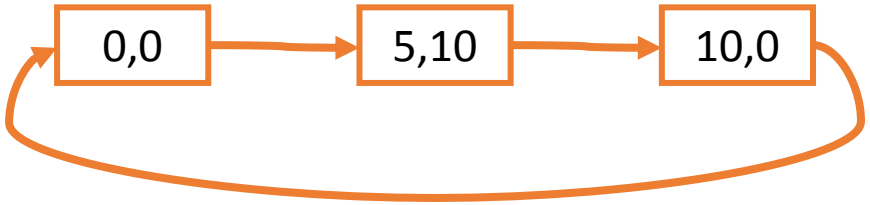
Verkettete Liste von Eckpunkten

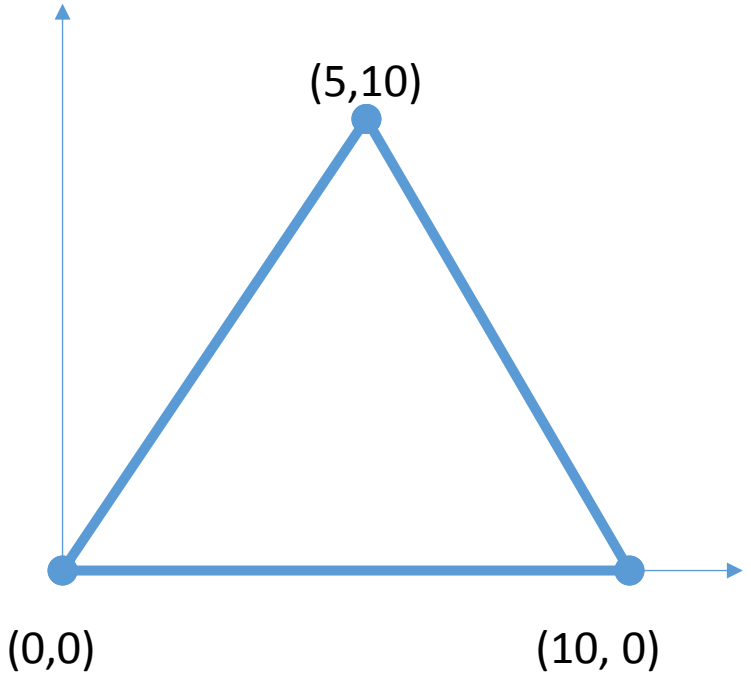
```
public class Vertex {
  public int x, y;
  public Vertex next;

  Vertex(int x0, int y0, Vertex nxt){
    next = nxt;
    x = x0;
    y = y0;
  }
}
```



Polygon: zirkuläre Liste von Eckpunkten

```
public class Polygon {
  Vertex first, last;
  Polygon() {
    first = null; last = null;
  }
}
```





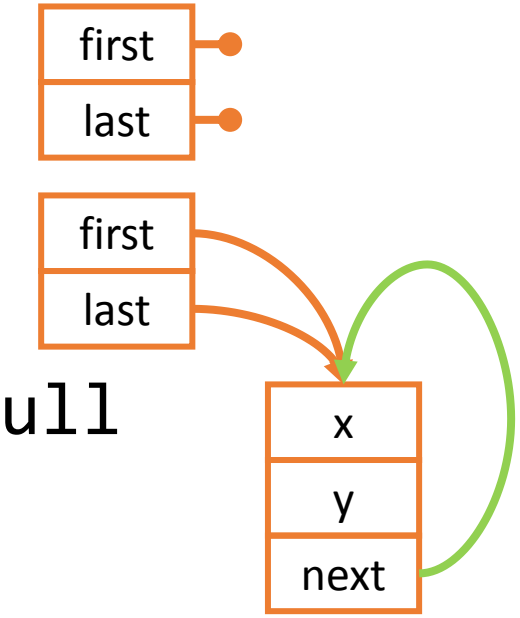
Invarianten der zirkulären Liste

Entweder: **keine Ecke**

```
first == last == null
```

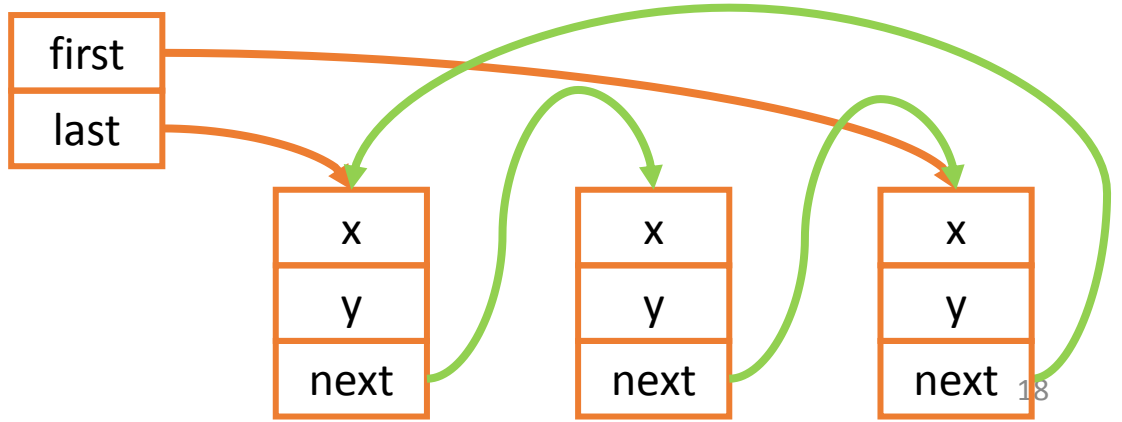
oder: **eine Ecke**

```
first == last && first != null
 && first.next == last
 && last.next = first
```



oder: **mehr als eine Ecke**

```
first != last
&& last.next == first
```



Polygon = Zirkuläre Liste von Eckpunkten

```
public class Polygon {
   Vertex first, last;
   Polygon() {... }
   // add point to the linked circular list
   public void AddPoint(int x, int y) {
      Vertex v = new Vertex(x,y,first);
      if (first == null){
         first = v;
         last = v;
      }
      last.next = v;
      last = v;
   }
}
```

Point in Polygon Algorithmus

Live Coding in der Vorlesung.

Auf separaten Slides als Präsentation.

PointInPolygon

```
public boolean PointInPolygon(int px, int py) {
   if (first == last) // keine oder eine Ecke
      return false;
   Vertex v = first;  // Invariante: first.next != first
   boolean b = false; // "Zähl"variable (false -> true -> false ... )
   do {
      if (IntersectHorizontalLeft(px, py, v, v.next))
         b = !b;
      v= v.next;
   } while (v != first);
   return b;
}
```

Schnitt

```
boolean IntersectHorizontalLeft(int x, int y, Vertex a, Vertex b) {
  if (b.y < y && y <= a.y) {
    return (x-a.x) * (b.y - a.y) <= (b.x-a.x)*(y-a.y);
  }
  else if(a.y < y && y <= b.y) {
    return (x-a.x) * (b.y - a.y) >= (b.x-a.x)*(y-a.y);
  }
  else
    return false;
}
```