Java ohne Kara

Ab jetzt: Java ohne Kara

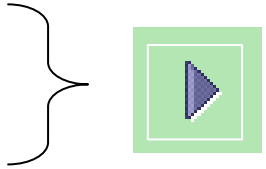
Ziel:

Erfahrungen sammeln mit ersten Java Programmen

JavaKara -> Java

Ablauf in JavaKara:

1. Programm schreiben
2. Kompilieren
3. Programm starten

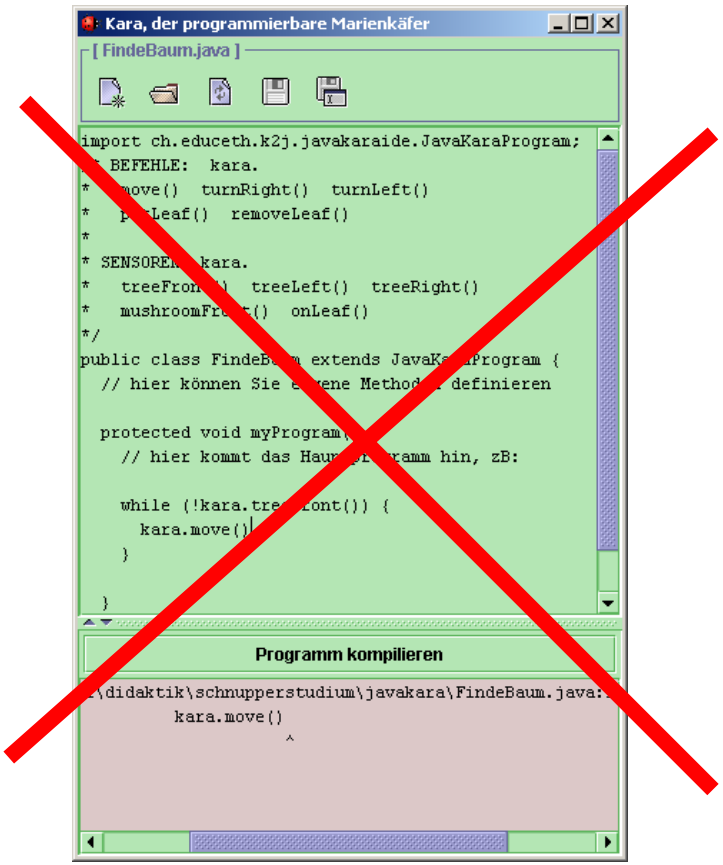


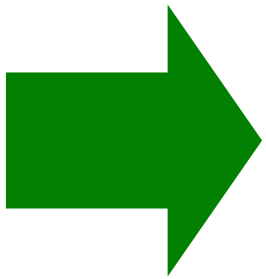
Ablauf in Java gleich

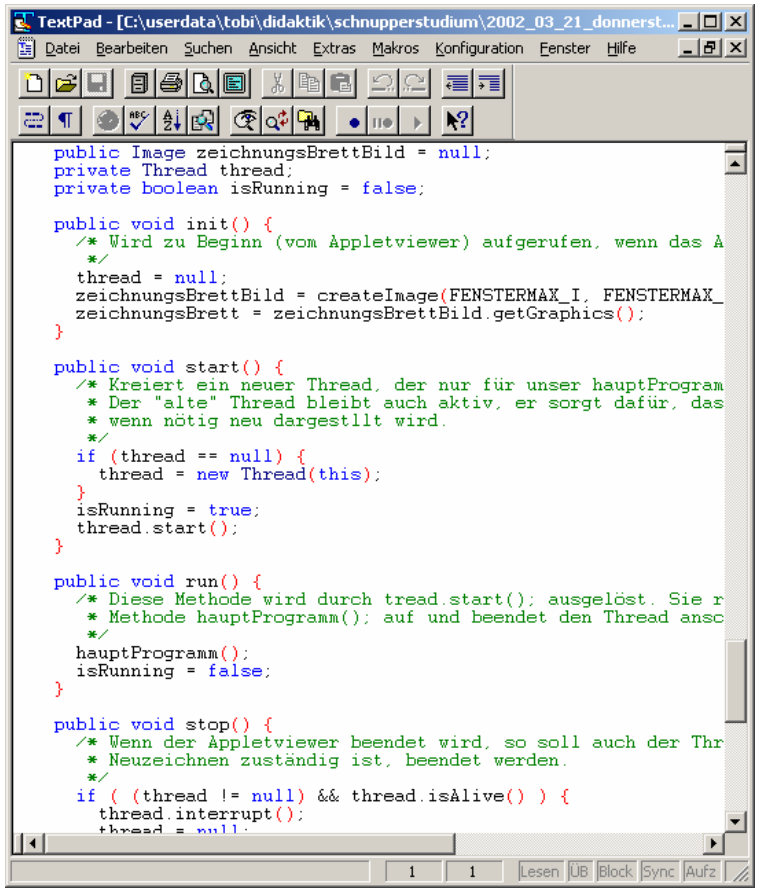
1. Programm schreiben
2. Kompilieren
3. Programm starten

1. Programm schreiben

Texteditor statt JavaKara-Editor

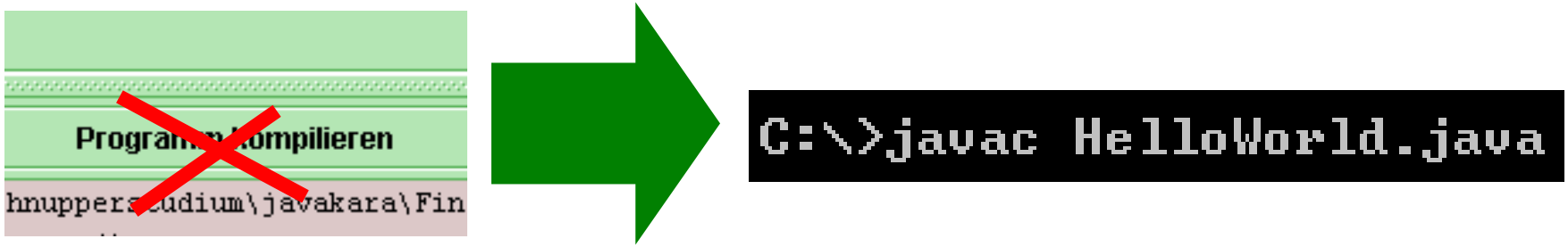






2. Kompilieren

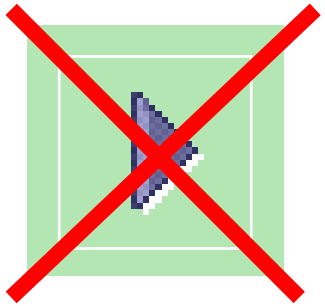
Befehl in MS-DOS-Eingabeaufforderung anstatt bequemer Knopf



`javac` steht für „**java c**ompiler“

3. Programm starten

Befehl in MS-DOS-Eingabeaufforderung anstatt bequemer Knopf



```
C:\>appletviewer HelloWorld.html
```

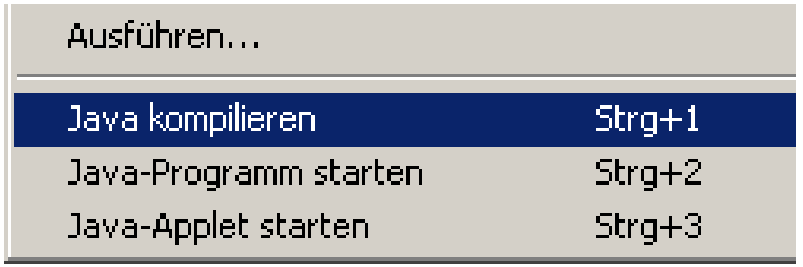
TextPad

Wir verwenden hier TextPad (www.textpad.com)

- **Vorteil 1:**
  Farben erleichtern das Lesen der Programmzeilen

```
int ms = m * 60;
int hs = h * 60 * 60;

// Clear the background.

g.setColor(bgcolor);
g.fillRect(start.x, start.y, size.

// Draw the clock circle.

g.setColor(ccolor);
g.fillOval(start.x, start.y, diam,

// Draw the clock numbers.
```

- **Vorteil 2:**
  Kompilieren und Start des Applets mit je einem Menu-Befehl



TextPad

- **Wir brauchen nur wichtigste Funktionen des TextPad:**
  - **Datei öffnen**
  - **Datei speichern**
  - **Rückgängig-Befehl**
  - **Java kompilieren**
  - **Java-Applet starten**

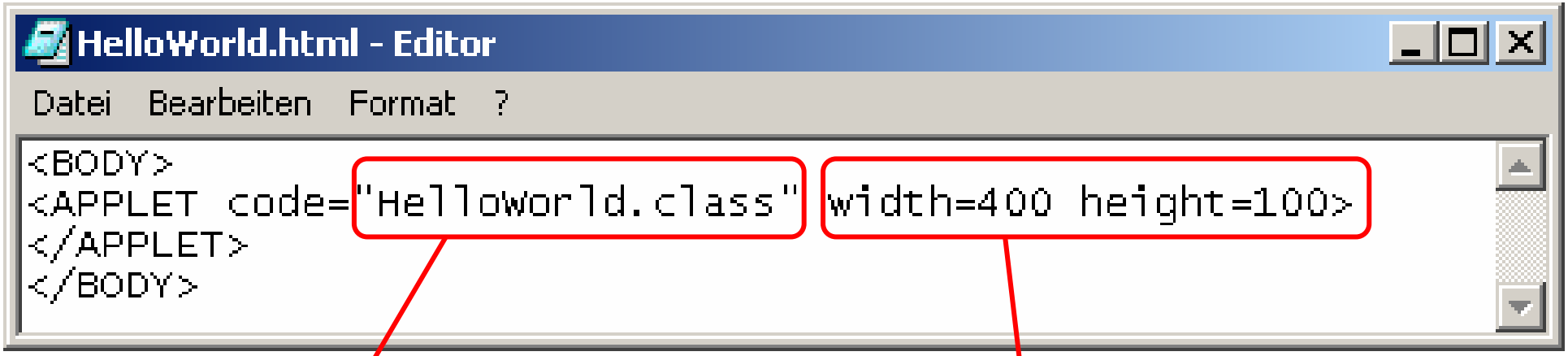
- **Siehe TextPad-Anleitung in den Unterlagen**

Warum HelloWorld.html ?

- **2 Arten Java Programme**
  - **Java Applikationen**
  - **Java Applets**
    - **Kann/muss man in Webseiten einbauen**
    - **Einfachere Programme als bei Applikationen**
- **Applets: Man benötigt Webseite (.html) und ein Betrachtungsprogramm**

```
C:\>appletviewer HelloWorld.html
```

Wie sieht die .html-Datei aus?

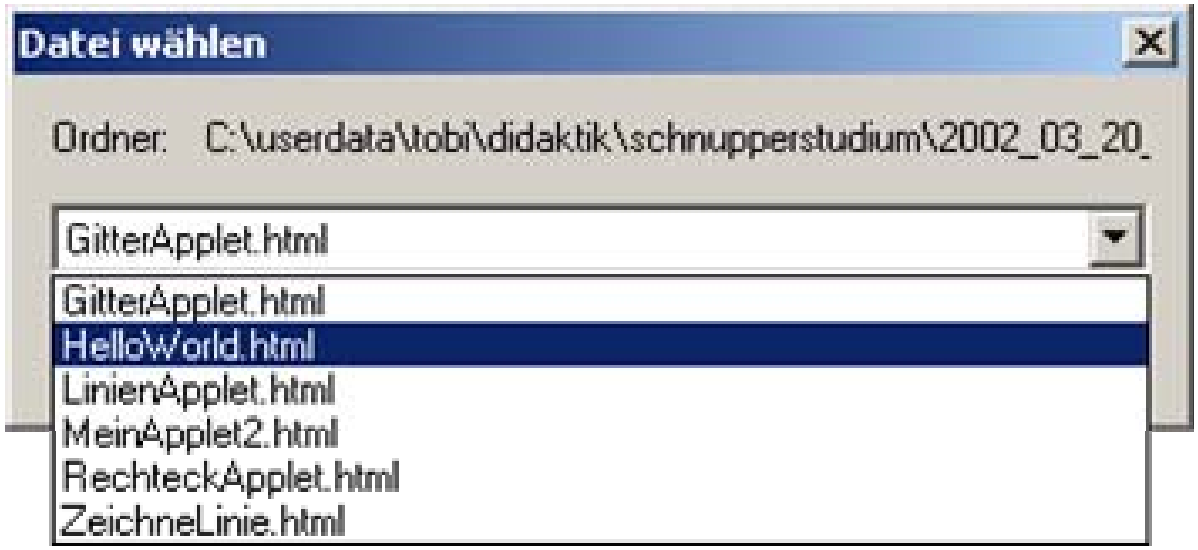


Dateiname der kompilierten Java-Klasse

Grösse des Programm-Fensters

TextPad-Problem

- **Um Java-Applet aus dem TextPad zu starten, muss die .html-Datei bereits existieren**



Also nochmals...

1. **Programm schreiben -> HelloWorld.java**

2. **Kompilieren -> HelloWorld.class**
   (javac HelloWorld.java)

3. **HTML-Datei (z.B. HelloWorld.html)**
   erstellen und Programm starten
   (appletviewer HelloWorld.html)

Ein einfaches Java-Applet

```
import java.awt.*;
import java.applet.*;
public class HelloWorld extends Applet
{
  public void paint(Graphics screen)
  {
    screen.drawString("hello world", 10, 10);
  }
}
```

Gähhhnnn...

Jetzt seid Ihr aber dran!

Auftrag

- **Jede schreibt das HelloWorld-Applet, speichert es, kompiliert es und lässt es laufen.**
- **Zeit: 20 Minuten**
- **Ihr findet alles was Ihr braucht in den Unterlagen**

Los!


ALSO GUT, ICH WILL VERSUCHEN, IHNEN NOCH EINMAL ZU ERKLÄREN, WAS „DATEI LÖSCHEN" BEDEUTET ...
Quelle: Uli Stein

Wichtig bei Java-Programmen

Goldene Regel

- **Dateiname = Klassenname**
  `HelloWorld.java <-> (...) class HelloWorld (...)`

- **Kompilieren nicht vergessen!**
  (Sonst startet man eine alte „HelloWorld.class“-Version)

Das Spezielle an Java

- **Java stellt viele Funktionen zur Verfügung**
- **Durch diese „Fertig-Bauelemente“ ist man schneller am Ziel**
- **Aber: Man muss sich an die Richtlinien halten**

public void paint(...)

- **Wird von Java aufgerufen, sobald ein Bildschirmbereich neu gezeichnet werden muss**
- **Von Java festgelegter Name**
- **Ideal, um Grafik-Sachen zu zeichnen.**

Ablauf bei Java-Programmen

- **In Java programmiert man Ereignisse**
- **Ereignis: „Führe bei Mausklicks die Methode xy aus“**
- **Nicht Ereignis-gesteuert: Viel komplizierter!**

Ablauf bei Java-Programmen

- **Einige vordefinierte Methoden: init, start, paint, stop, destroy**
- **Die restlichen Ereignisse muss man selber definieren:**
  - **Z.B. Neuer Knopf erstellen**
  - **Ereignis-Methode schreiben**
  - **Methode zuordnen „Wenn Mausklick auf diesen Knopf, dann führe Methode xy aus“**

Hä? – Brauchen wir vorläufig gar nicht!

Ablauf bei Java

Von Java definierte Methoden

- **Beim Start der Ausführung:**
  `void init()`
- **Immer, wenn es etwas zu zeichnen gibt:**
  `void paint(Graphics g)`
- **Vor dem Programmende:**
  `void destroy()`
- **Und noch weitere wie**
  `void start(),void stop(),void update()`

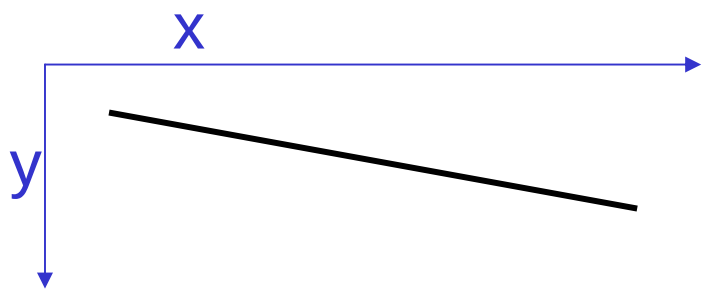
Was muss ich jetzt tun?

- **Methode `paint()` eignet sich ideal für das Zeichnen von Punkten, Linien, etc. auf den Bildschirm.**

- **Java-Applet-Programm muss die Methode enthalten, dann wird sie automatisch aufgerufen.**

Linie zeichnen

```
import java.awt.*;
import java.applet.*;
public class ZeichneLinie extends Applet
{
  public void paint(Graphics screen)
  {
    screen.drawLine(10,10,200,50);
  }
}
```



Graphics...

```
import java.awt.*;
import java.applet.*;
public class ZeichneLinie extends Applet
{
  public void paint(Graphics screen)
  {
    screen.drawLine(10,10,200,50);
  }
}
```

Graphics...

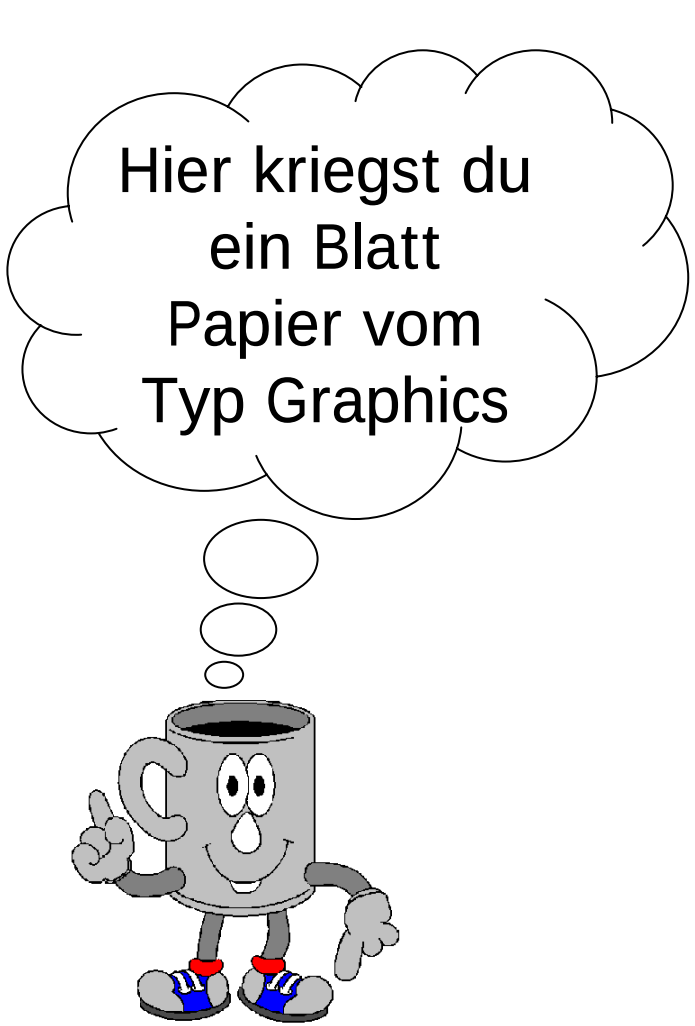
An Programm: starte dein paint(...) !

Java

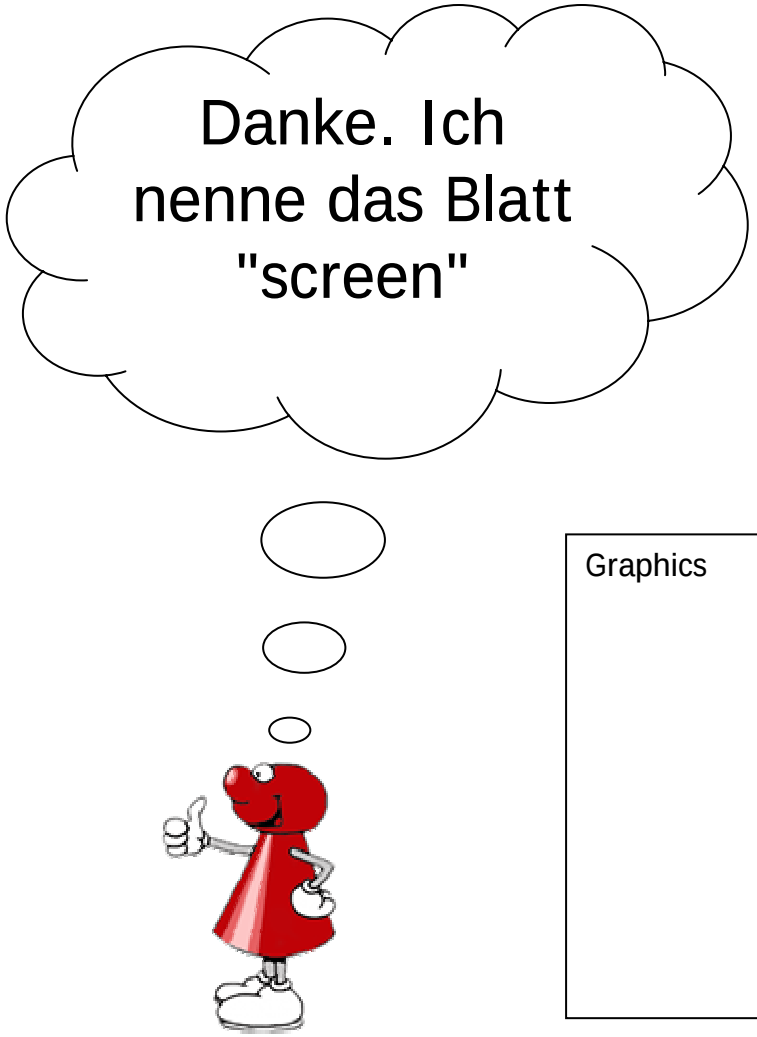
Okay. Aber wo soll ich denn hinzeichnen?

Programm

Graphics...



Java



Programm

Zusammengefasst

- **Der Methode paint(...) wird ein Objekt vom Typ Graphics übergeben:**

```
public void paint (Graphics screen)
{
   screen.drawLine(10,10,200,50);
}
```

- **Dem Objekt geben wir einen Namen**
- **Objekte vom Typ Graphics stellen viele Methoden bereit, z.B. drawLine(...)**

Graphics-Befehle

```
Graphics screen;
```

- **Text ausgeben:**

```
screen.drawString("Java ist cool", 10, 20);
```

- **Linie zeichnen:**

```
screen.drawLine(5, 10, 250, 40);
```

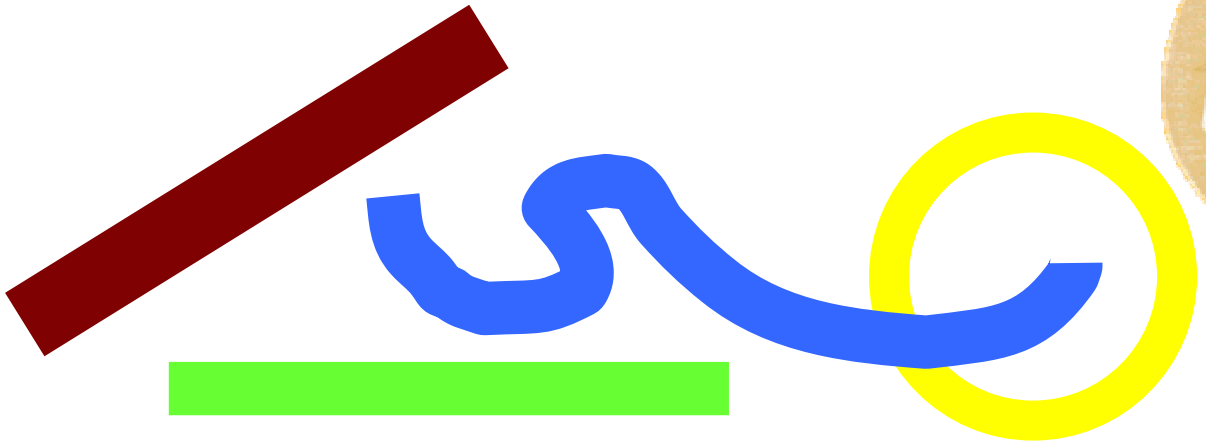
- **Punkt zeichnen:**

```
screen.drawLine(10, 70, 10, 70);
```

Jetzt kombinieren wir alles!

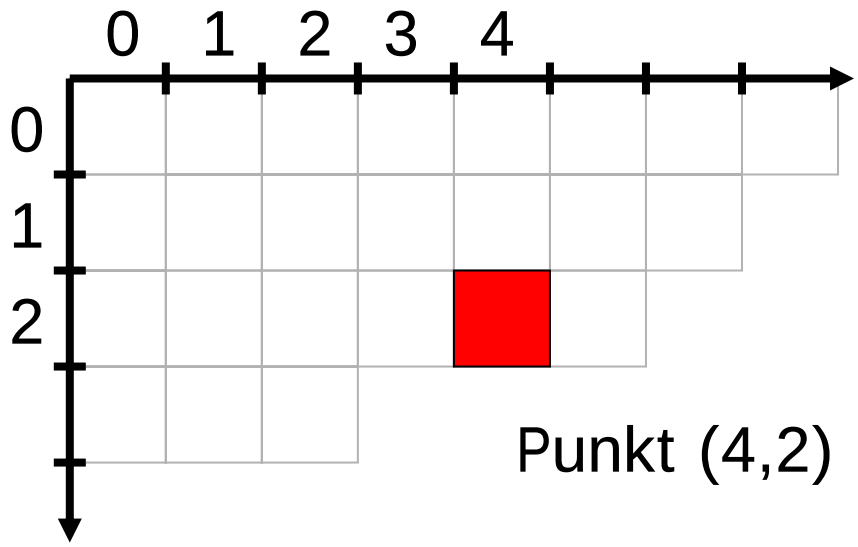
- **Variablen, Schleifen, Methoden, Linien, Kreise, Punkte, Texte, ...**
- **Grafik-Aufgaben im Anhang**

- **Zeit: 45 Minuten**
- **Danach 15 Minuten Pause**



Hinweise zu Grafik

- **Koordinaten beginnen bei (0,0)**
- **Fenster der Grösse 400x200**
  - **Punkt oben-links: (0,0)**
  - **Punkt unten-rechts: (399,199)**



Graphics-Blatt "weitergeben"

- **Methode zeichneRechteck(...) soll man mit `screen.drawLine(x1, y1, x2, y2);` eine Linie zeichnen können**
- **Wir geben das Blatt an die Methode weiter**

```
public void paint (Graphics blatt)
{
  zeichneRechteck(10, 10, 200, 50, blatt);
}
```

```
void zeichneRechteck ( ... , Graphics screen)
{
  screen.drawLine(x1, y1, x2, y2);
}
```

RechteckApplet.java

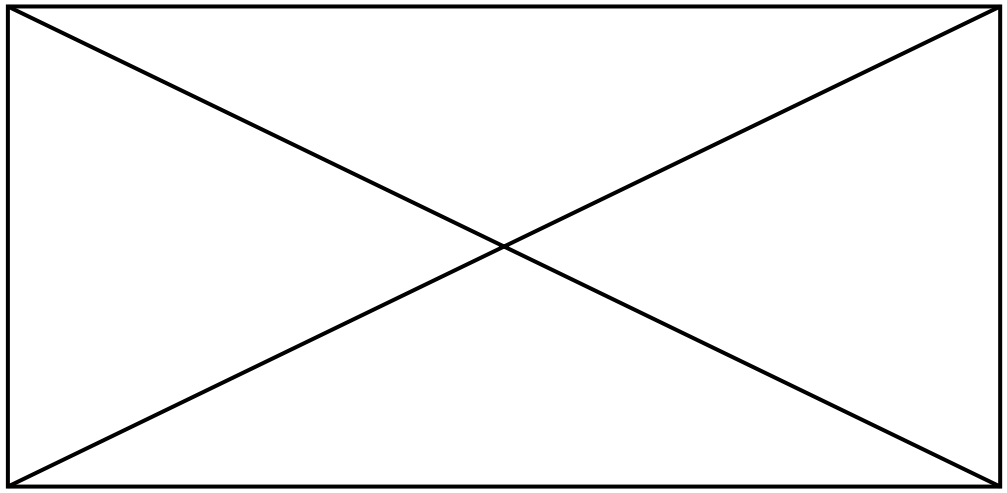
```
void zeichneRechteck(int x, int y,
                     int width, int height,
                     Graphics screen) {
  (...)
  screen.drawLine(obenlinks_x  , obenlinks_y  ,
                  obenrechts_x , obenrechts_y );
  screen.drawLine(obenrechts_x , obenrechts_y ,
                  untenrechts_x, untenrechts_y);
  screen.drawLine(untenrechts_x, untenrechts_y,
                  untenlinks_x , untenlinks_y );
  screen.drawLine(untenlinks_x , untenlinks_y ,
                  obenlinks_x  , obenlinks_y  );
}
```

RechteckApplet.java

```
int obenlinks_x = x;
int obenlinks_y = y;

int obenrechts_x = x+width-1;
int obenrechts_y = y;

int untenlinks_x = x;
int untenlinks_y = y+height-1;

int untenrechts_x = x+width-1;
int untenrechts_y = y+height-1;
```

RechteckApplet.java

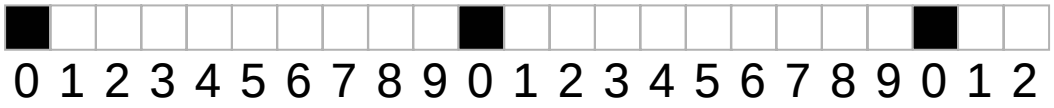
```
public void paint(Graphics g) {
   zeichneRechteck(100,50,200,100,g);
   g.drawLine(100,50,299,149);
   g.drawLine(299,50,100,149);
}
```



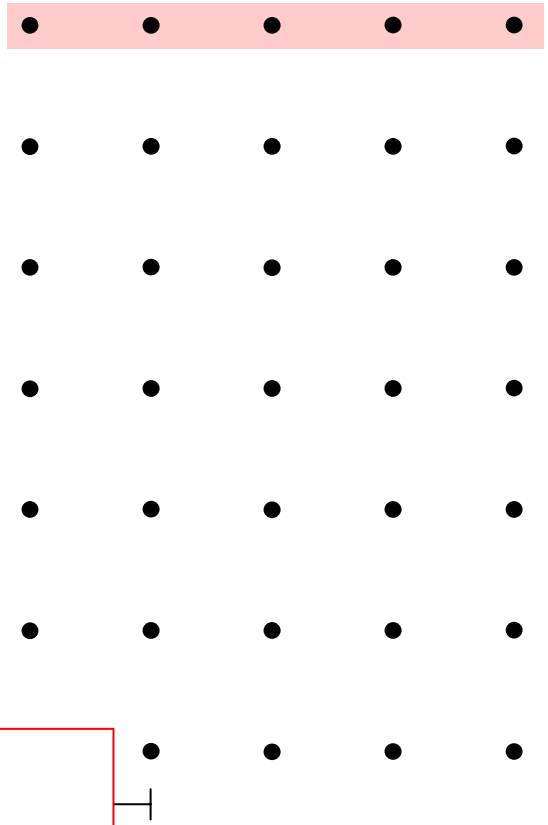
Punktmuster

- **Punktmuster im 10er-Raster**
- **Ideen?**

Erstelle mal die erste Zeile:



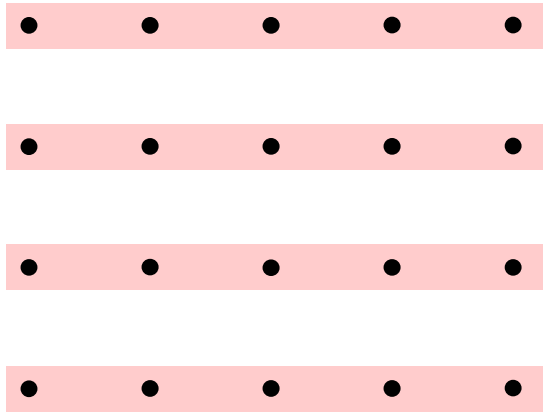

```
for (int i=0; i<400; i++) {
  screen.drawLine(i*10,0,i*10,0);
}
```



Punktmuster

Und jetzt?

Die restlichen Zeilen: Wiederhole erste Zeile „x-Mal“



```
for (int j=0; j<400; j++) {
  for (int i=0; i<400; i++) {
    screen.drawLine(i*10, j*10, i*10, j*10);
  }
}
```

te

Auf den ersten Blick verwirrend

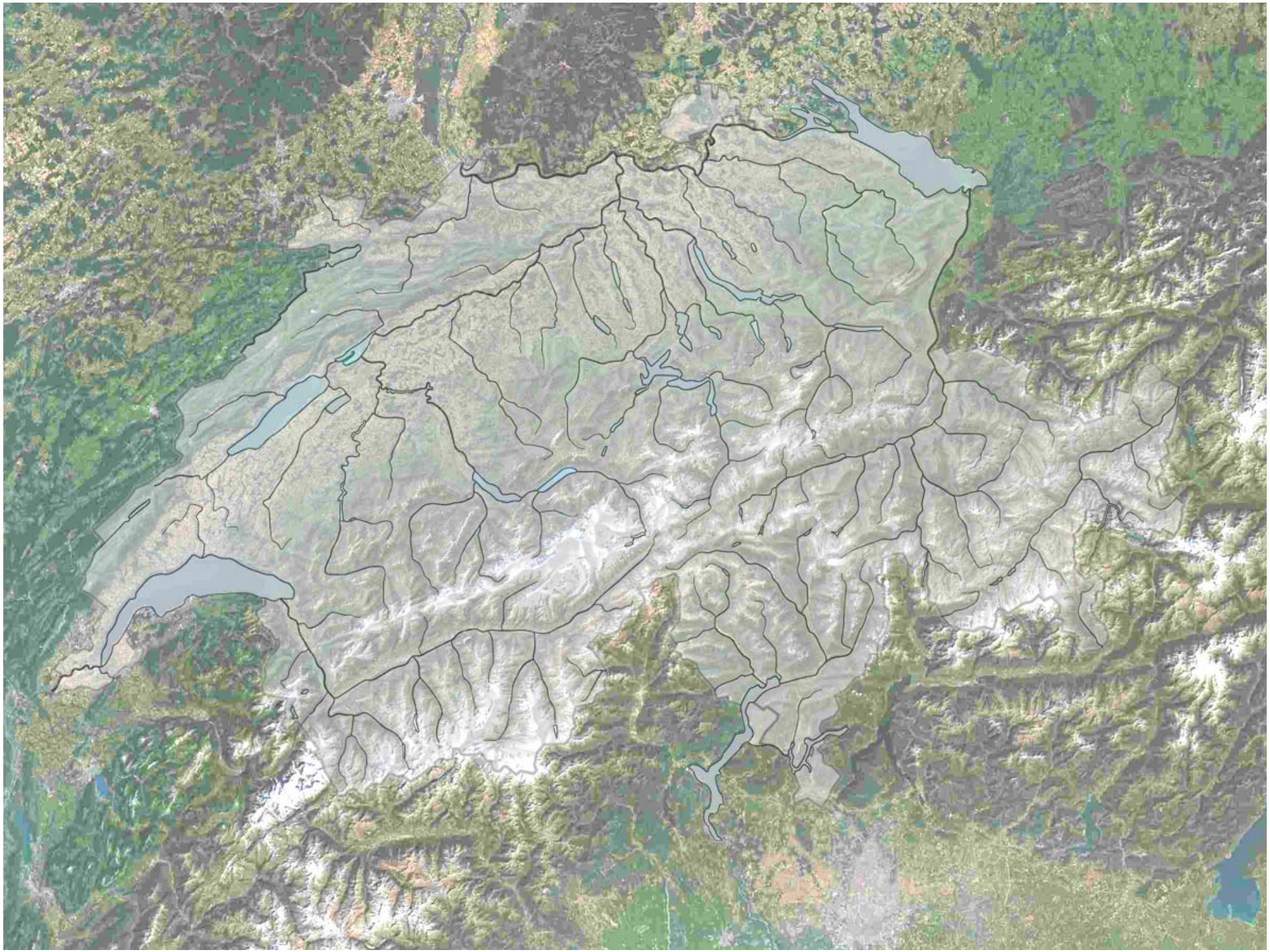
- **Innere Schleife wird "zuerst abgearbeitet" (einzelne Zeile)**

```
for (int j=0; j<400; j++) {
  for (int i=0; i<400; i++) {
    screen.drawLine(i*10, j*10, i*10, j*10);
  }
}
```

- **Die Zeile muss bei jedem neuen `j` wiederholt gezeichnet werden**

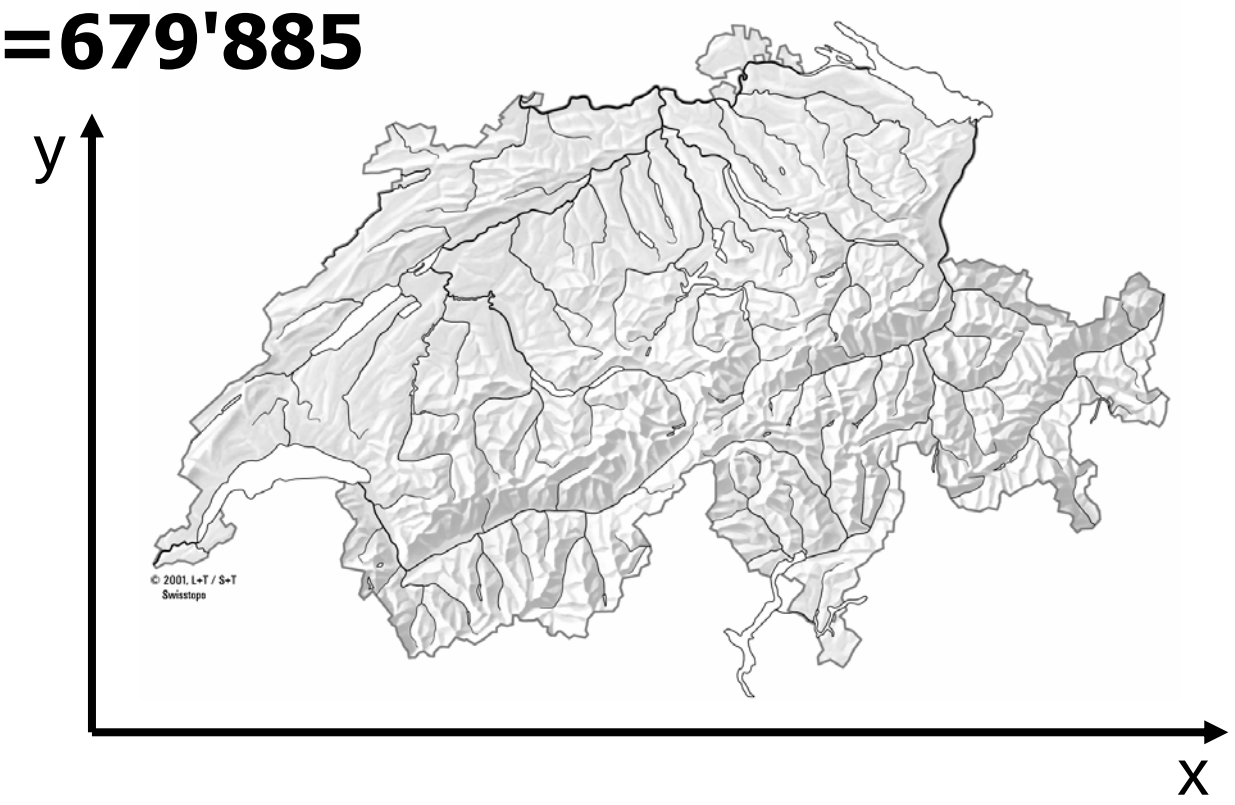
Schweizerkarte

- **Schweizerkarte auf dem Bildschirm anzeigen**
- **Stadt Bern und Stadt Zürich einzeichnen**

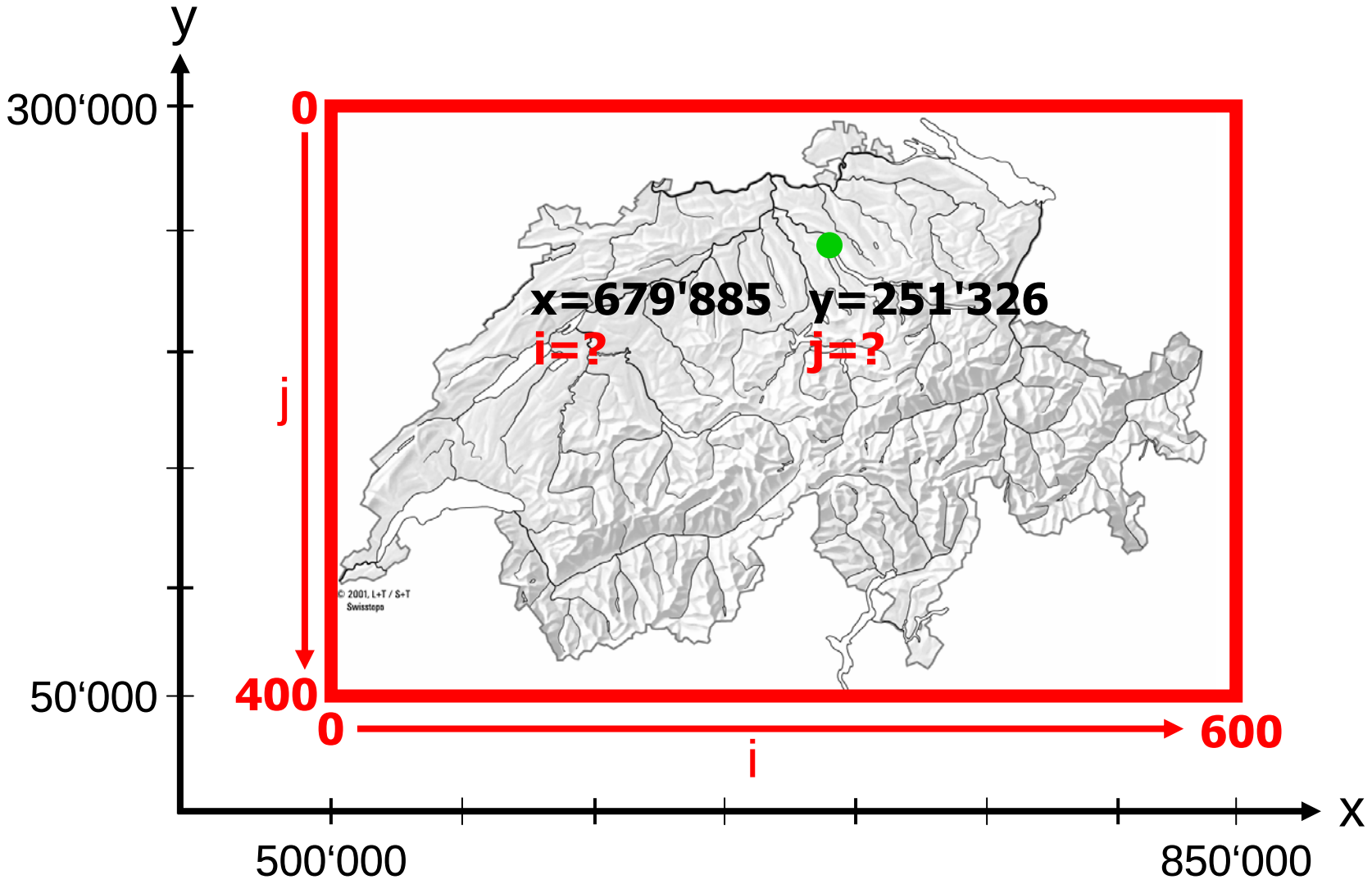


Wo einzeichnen?

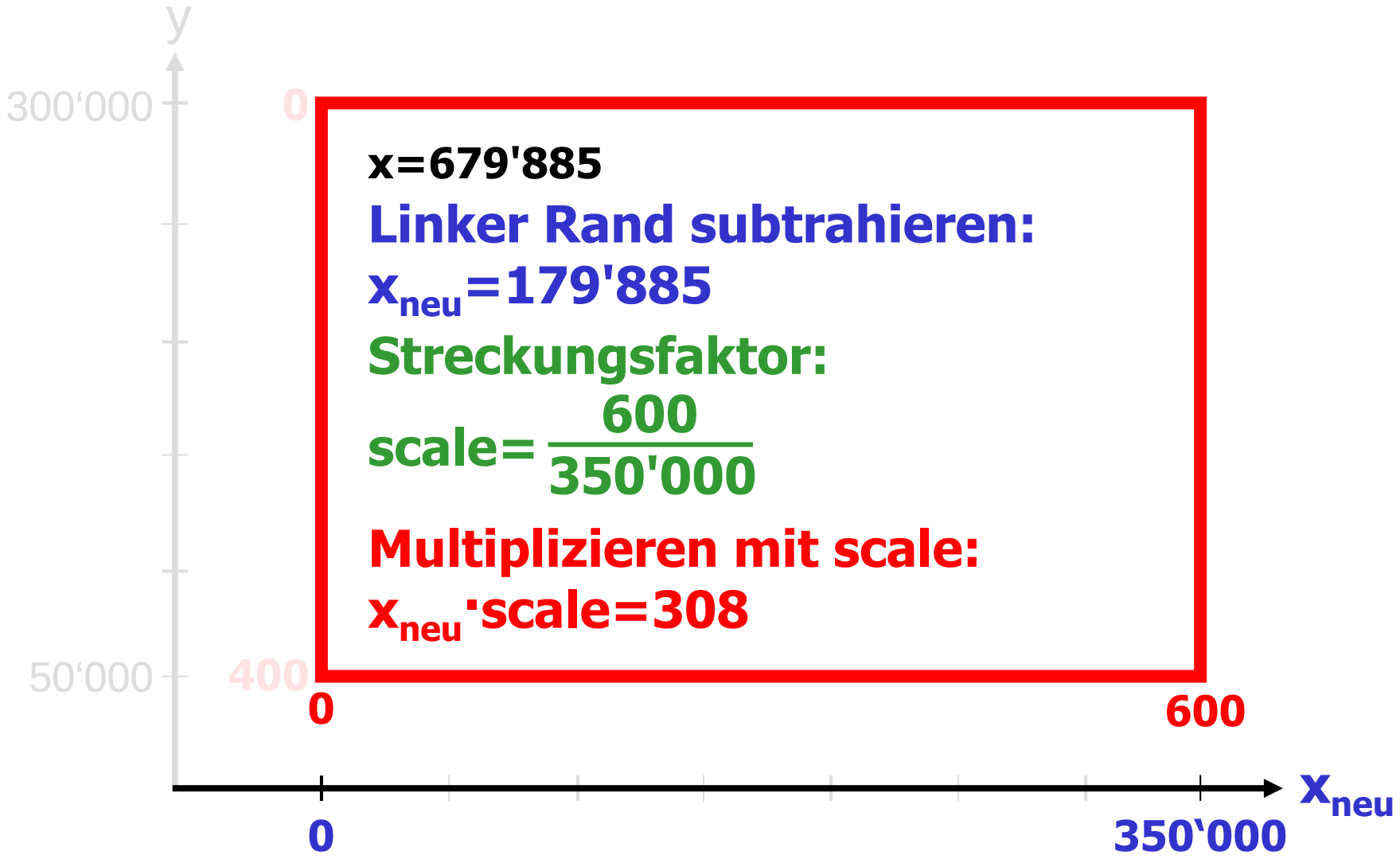
- **Koordinaten von Bern: y=200'000, x=600'000**
- **ETH Zürich, Hönggerberg: y=251'326, x=679'885**



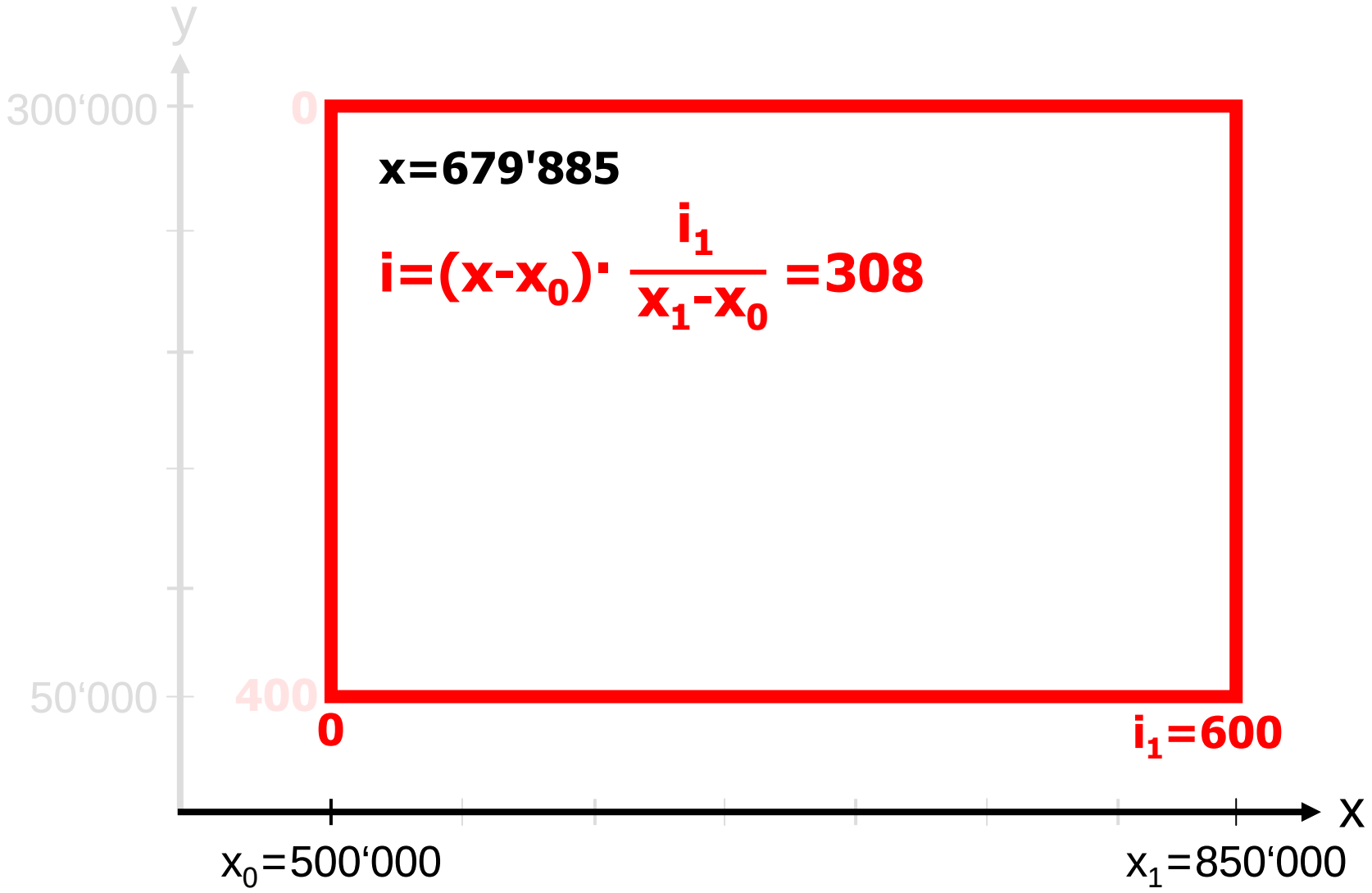
Wo ist Zürich?



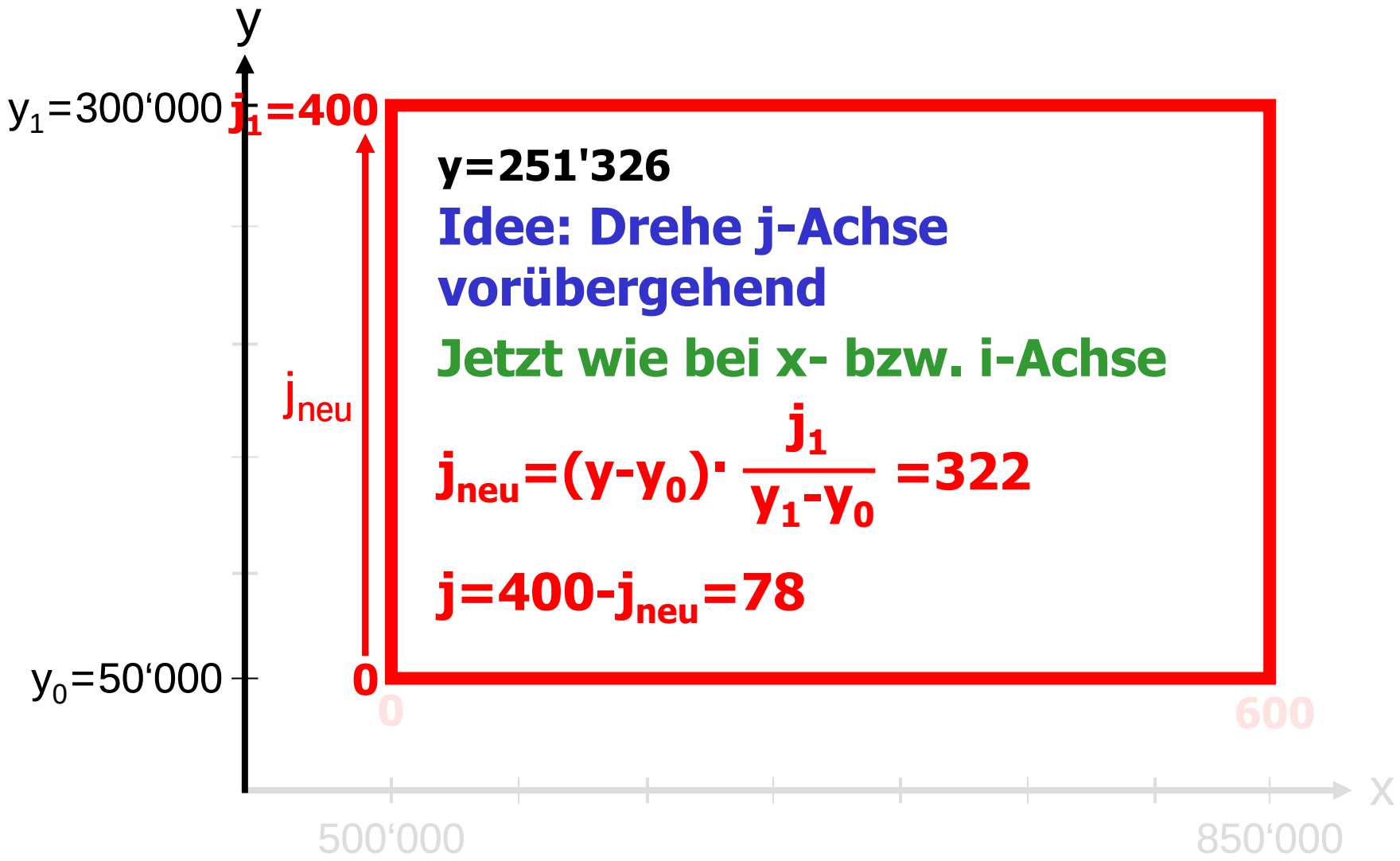
Für x bzw. i


y
300‘000
50‘000
0
400
x=679'885
Linker Rand subtrahieren:
x_{neu}=179'885
Streckungsfaktor:
scale= 600 / 350'000
Multiplizieren mit scale:
x_{neu}'scale=308
0
600
0
350‘000
x_{neu}

Auf einen Blick



Für y bzw. j



y=251'326

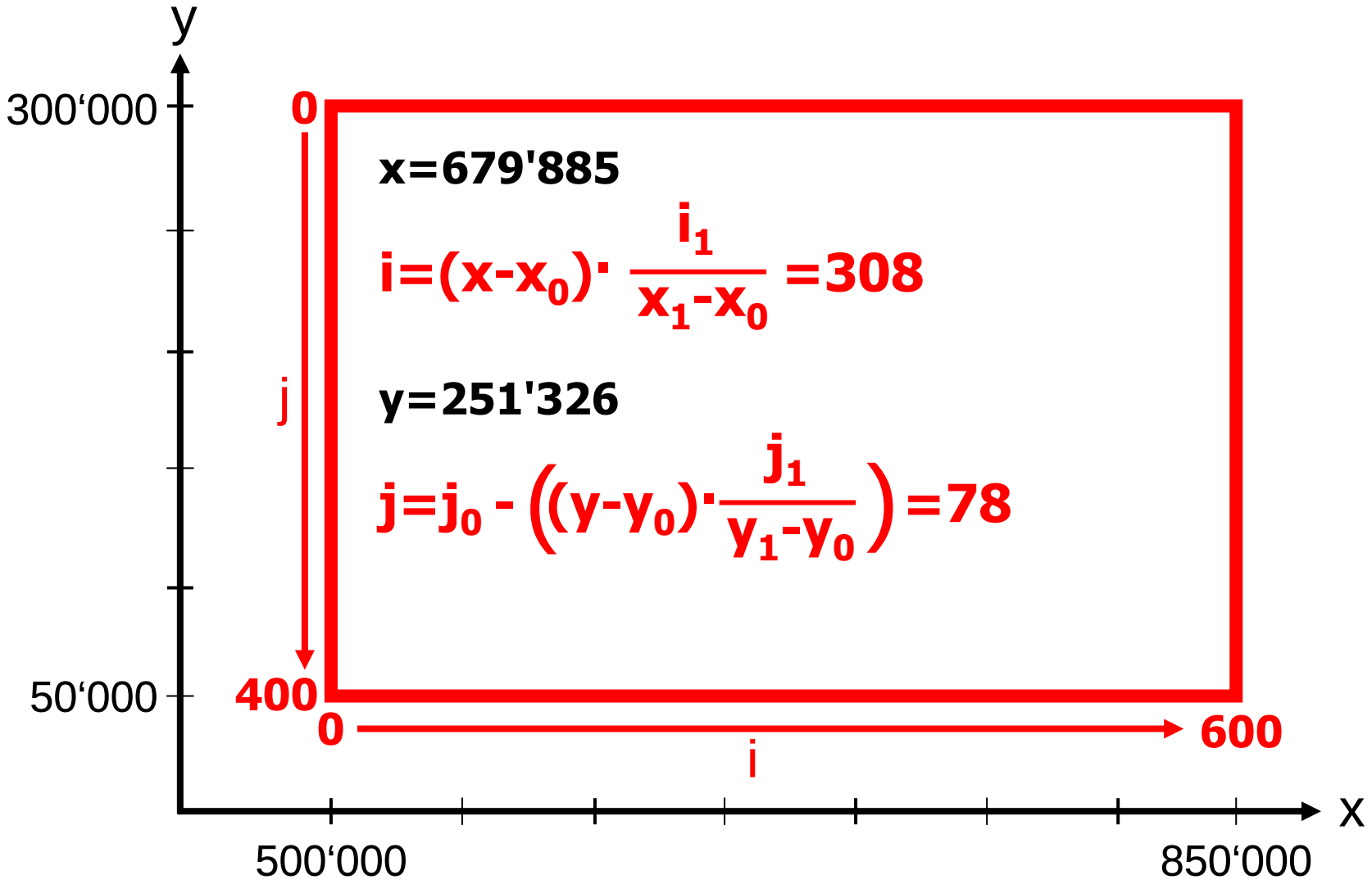
Idee: Drehe j-Achse vorübergehend

Jetzt wie bei x- bzw. i-Achse

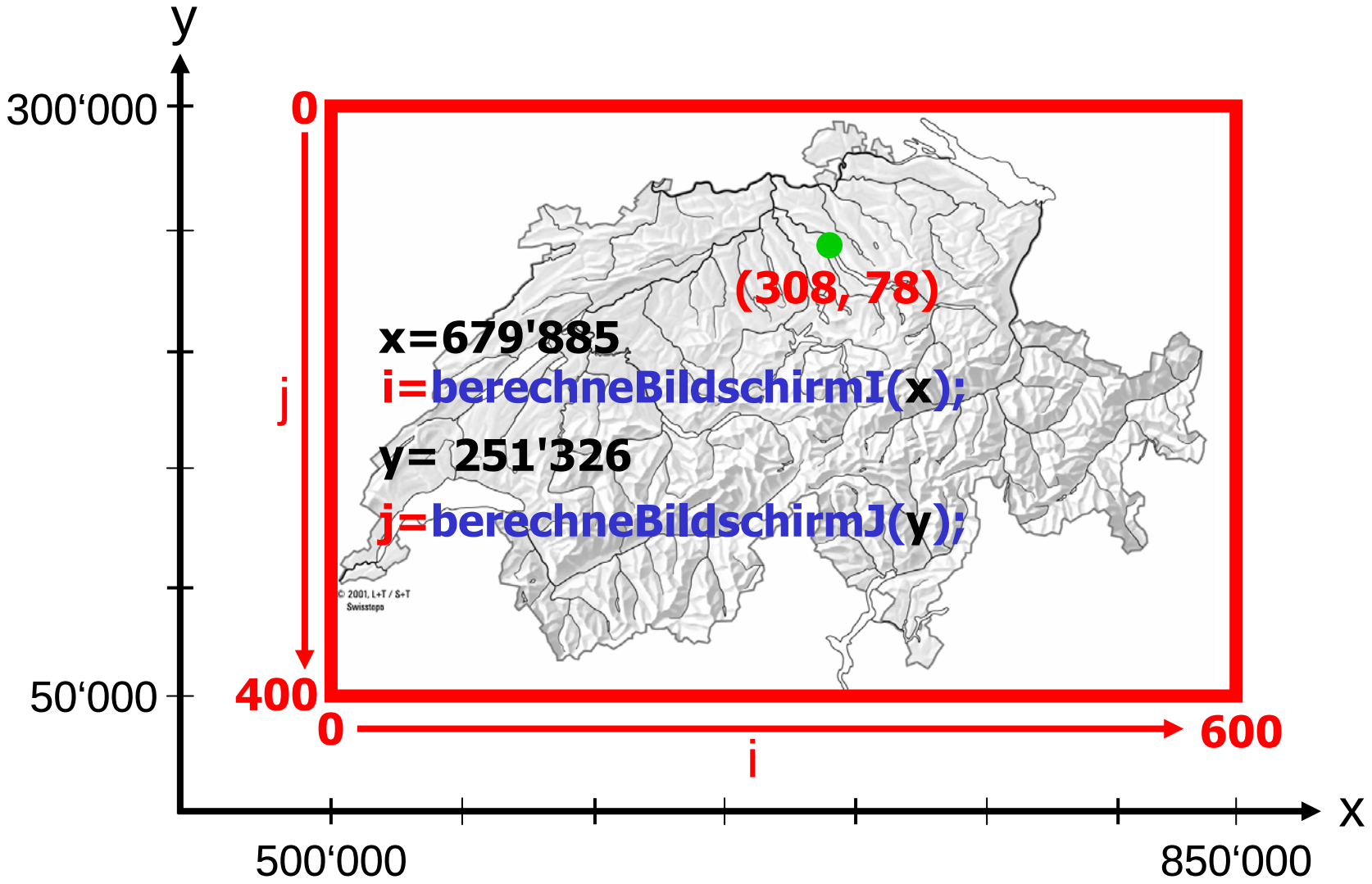
$$j_{neu}=(y-y_0)\cdot\frac{j_1}{y_1-y_0}=322$$

$$j=400-j_{neu}=78$$

Auf einen Blick



Wo ist nun Zürich?



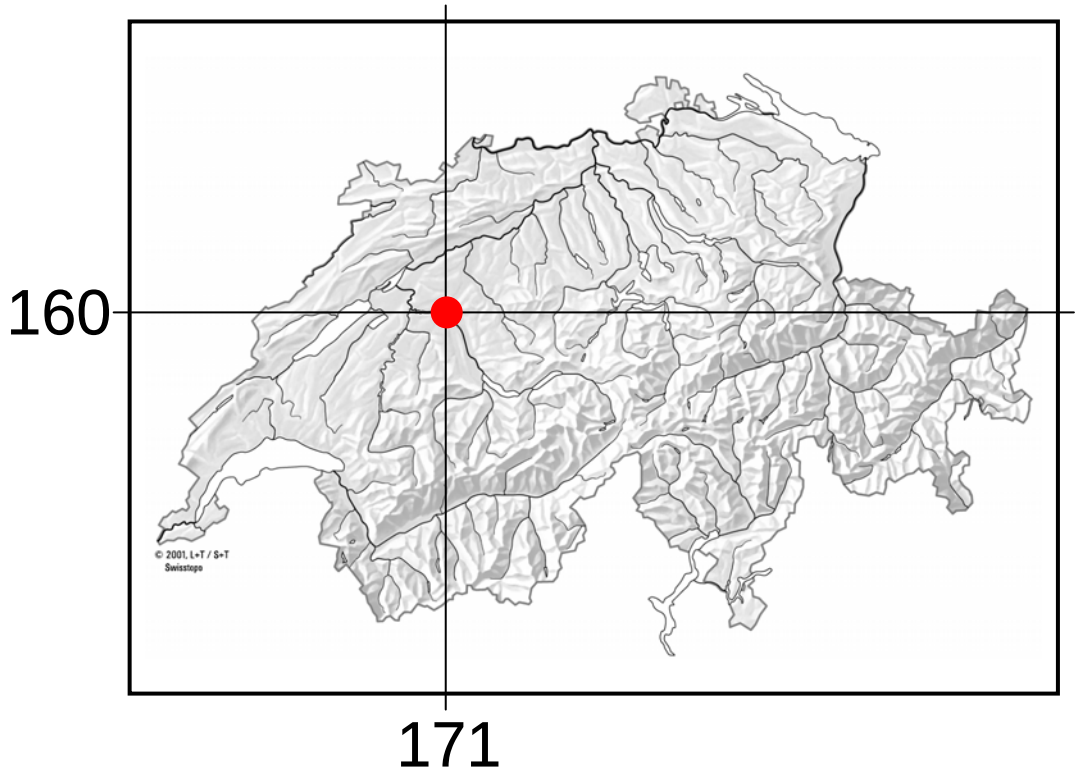
Nun in Java...

```
int berechneBildschirmI (double x) {
  double bildschirmI;
  bildschirmI = (x-XMIN) * FENSTERMAX_I / (XMAX-XMIN);
  return (int) bildschirmI;
}

int berechneBildschirmJ (double y) {
  double bildschirmJ;
  double jverkehrt;
  jverkehrt = (y-YMIN) * FENSTERMAX_J / (YMAX-YMIN);
  bildschirmJ = FENSTERMAX_J - ( jverkehrt );
  return (int) bildschirmJ;
}
```

Bern

i=berechneBildschirmI(600000);

j=berechneBildschirmJ(200000);



(Zusatzfolien)

Nicht dein Traum-Typ?

- **Typen kann man in Java umwandeln!**
- **Gleitkommazahl in eine Ganzzahl umwandeln**

```
int i;
double  pi = 3.14159;
i = (int) pi;
```

- **Ganzzahl in Zeichenkette umwandeln**

```
int jahr = 2002;
String text = "Wir haben das Jahr" + jahr;
```

Java-Gerüst

```
import java.awt.*;
import java.applet.*;
public class ZeichneLinie extends Applet
{
  public void paint(Graphics screen)
  {
    /* ... */
  }
}
```

awt: Advanced Windowing Toolkit

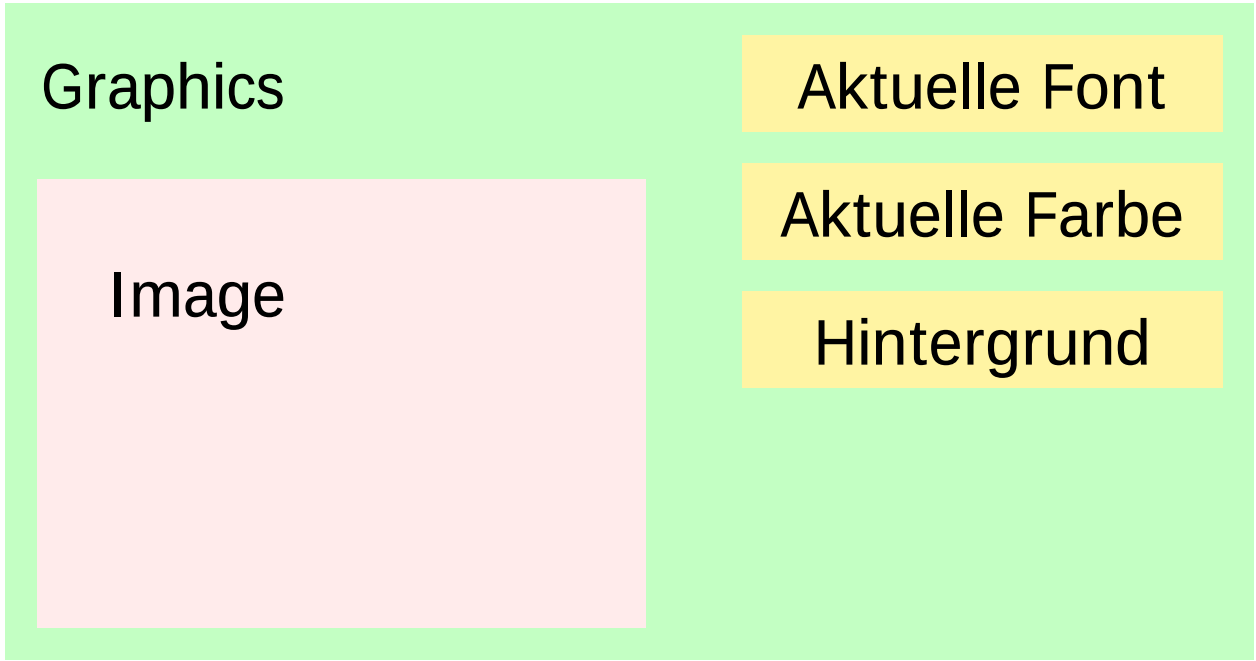
Java-Gerüst

```
import java.awt.*;
import java.applet.*;
public class ZeichneLinie extends Applet
{
  public void paint(Graphics screen)
  {
    /* ... */
  }
}
```

„Erweitert“ Applet, d.h. ZeichneLinie gehört zu den Applets

Graphics

- **Neben Graphics gibt es in Java auch noch eine Klasse Image.**
- **Graphics beinhaltet immer ein Image, aber auch noch weitere Objekte**



Graphics

- **Wir zeichnen die Linien in das Graphics-Objekt.**
- **Dort werden die Linien auf das Image-Objekt übertragen**