Objektorientierte Programmierung unter der Lupe

Vererbung

```
[Modifizierer] class Klassenname
  [extends Basisklasse]
   [implements Schnittstellen] {
   Attributdeklarationen
   Methodendeklarationen
  }
```

- `extends`: erweitere „Superklasse“ *`Basisklasse`*
- neue Klasse „erbt“ Attribute und Methoden
- Methoden der Superklasse können überschrieben (ersetzt) werden
- Konstruktoren werden **nicht** vererbt
- Zugriff auf Methoden/Konstrukturen der Superklasse: `super`
- Beispiel: Girokonto als spezielles Konto mit zusätzlichem Limit (beim Auszahlen berücksichtigen !)

Interfaces

```
[Modifizierer] class Klassenname
  [extends Basisklasse]
  [implements Schnittstellen] {
  Attributdeklarationen
  Methodendeklarationen
 }
```

- Interfaces sind Schnittstellenbeschreibungen – legen Verhalten fest, aber implementieren dieses nicht
- `implements`: Klasse implementiert eine oder mehrere *`Schnittstellen`*
- die Klasse **muss** dann alle in der Schnittstellen-beschreibung definierten Methoden implementieren

Deklaration eines Interfaces

```
[public] interface Interfacename
  [extends Interface-Liste] {
  Konstanten
  abstrakte Methoden
 }
```

- alle *Konstanten* sind implizit `public static final`
- alle *Methoden* sind implizit `public abstract` (d.h. ohne Anweisungsblock)
- `extends`: Interfaces können im –Gegensatz zu Klassen– von mehreren anderen Interfaces abgeleitet werden

Deklaration eines Interfaces

```
[public] interface Interfacename
  [extends Interface-Liste] {
  Konstanten
  abstrakte Methoden
 }
```

mehr zu Vererbung und Interfaces im Handbuch

- alle *`Konstanten`* sind implizit `public static final`
- alle *`Methoden`* sind implizit `public abstract` (d.h. ohne Anweisungsblock)
- `extends`: Interfaces können im –Gegensatz zu Klassen– von mehreren anderen Interfaces abgeleitet werden

Grafik

Erstes Beispiel in Fenster

- Wir verwenden das Abstract Window Toolkit (awt) – Swing wäre etwas komfortabler
- Wir müssen die Klasse `Frame` erweitern
- Der Konstruktor der Superklasse wird mit `super()` aufgerufen
- `setSize(int x, int y)` legt die Größe des Fensters in Pixeln fest
  Achtung: einschließlich Beiwerk (Rand) des Window-Managers
- `setLocation(int x, int y)` definiert Position des Fensters auf dem Bildschirm
- `setBackground(Color color)` & `setForeground (Color color)` legen Farbe von Vorder- & Hintergrund fest

⇒ Programm `FirstFenster.java`

Erstes Beispiel in Fenster

- `setVisible(true)` zeigt das Fenster auf dem Bildschirm an
- Methode `paint()` wird zum Neuzeichnen des Fensters aufgerufen – insbesondere beim ersten Erscheinen, aber auch beim Freilegen verdeckter Flächen
- `paint(Graphics g)` wird mit einem Objekt der Klasse `Graphics` aufgerufen
  - ★ `setFont(Font font)` legt Schriftart fest
  - ★ `drawString(String s, int x, int y)` zeichnet Zeichenkette `s` an Position `(x, y)`

⇒ Programm `FirstFenster.java`

Klasse `Font`

- `Font(String` *`font`*`, int` *`style`*`, int` *`size`*`)`
  erzeugt neues `Font`-Objekt

  ★ *`font`*: z.B. „`SansSerif`“, „`Serif`“ oder „`Monospaced`“

  ★ *`style`*: „`Font.PLAIN`“, „`Font.BOLD`“,
  „`Font.ITALIC`“ oder „`Font.BOLD | Font.ITALIC`“

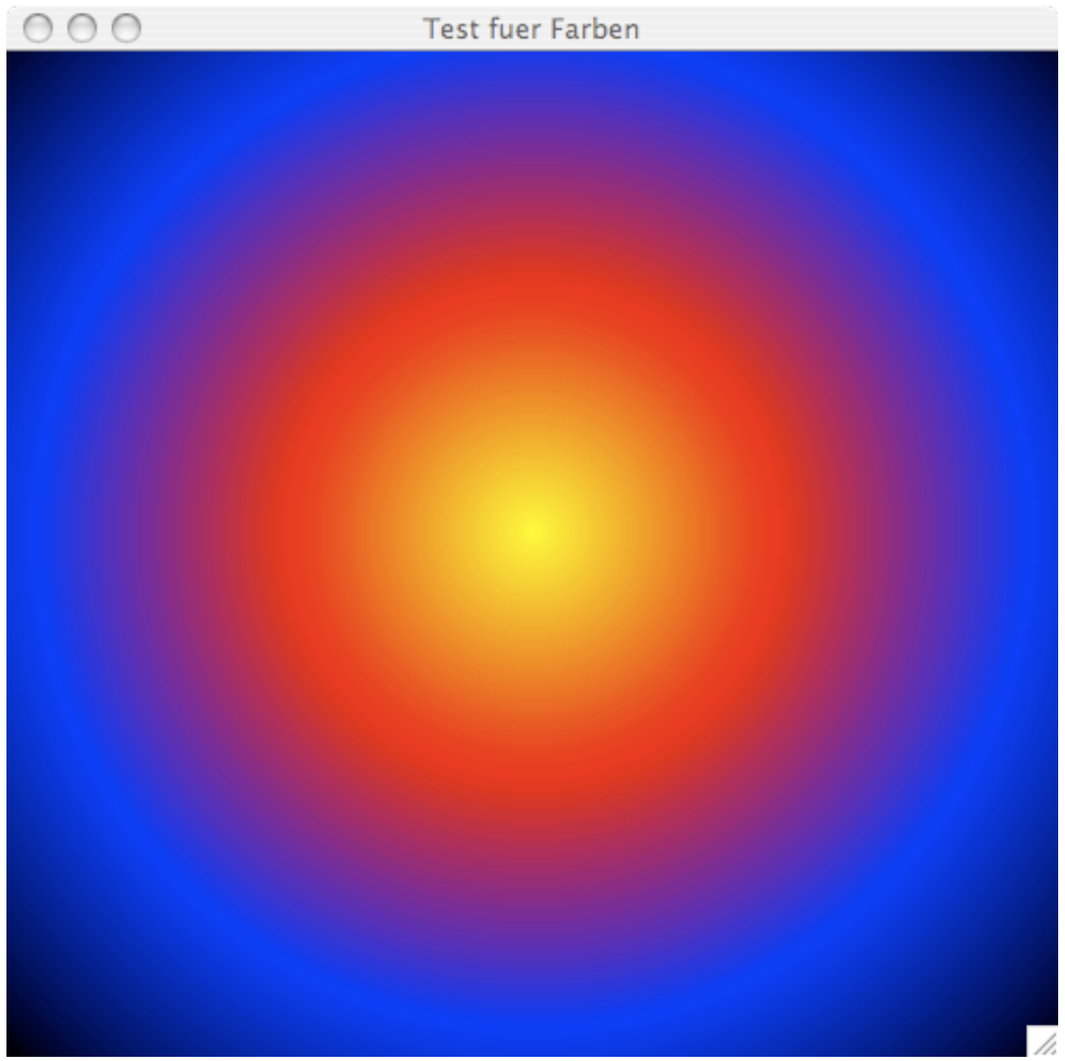
  ★ *`size`*: Schriftgröße in Pixel

⇒ Programm `Schriften.java`

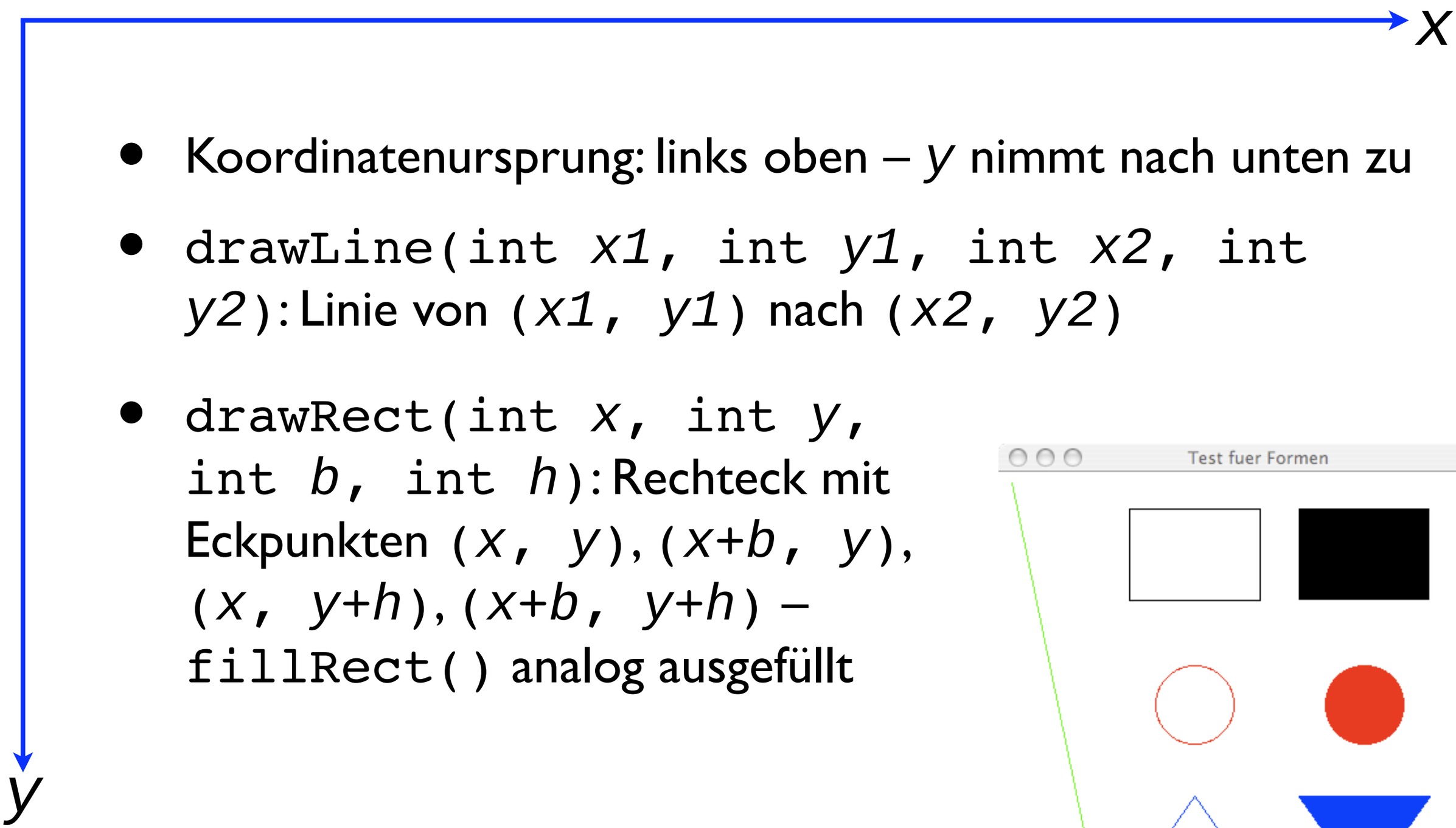
Klasse `Color`

- `setColor(Color c)` legt Farbe für nachfolgende Operationen fest
- Klasse `Color` enthält vordefinierte Farben, z.B.: `black`, `blue`, `cyan`, `gray`, `green`, `magenta`, `orange`, `red`, `white`, `yellow`
- `Color(int r, int g, int b)` erzeugt neue Farbe mit Rot-Anteil $0 \leq r \leq 255$, Grün-Anteil $0 \leq g \leq 255$, Blau-Anteil $0 \leq b \leq 255$

⇒ Programm `Farben.java`



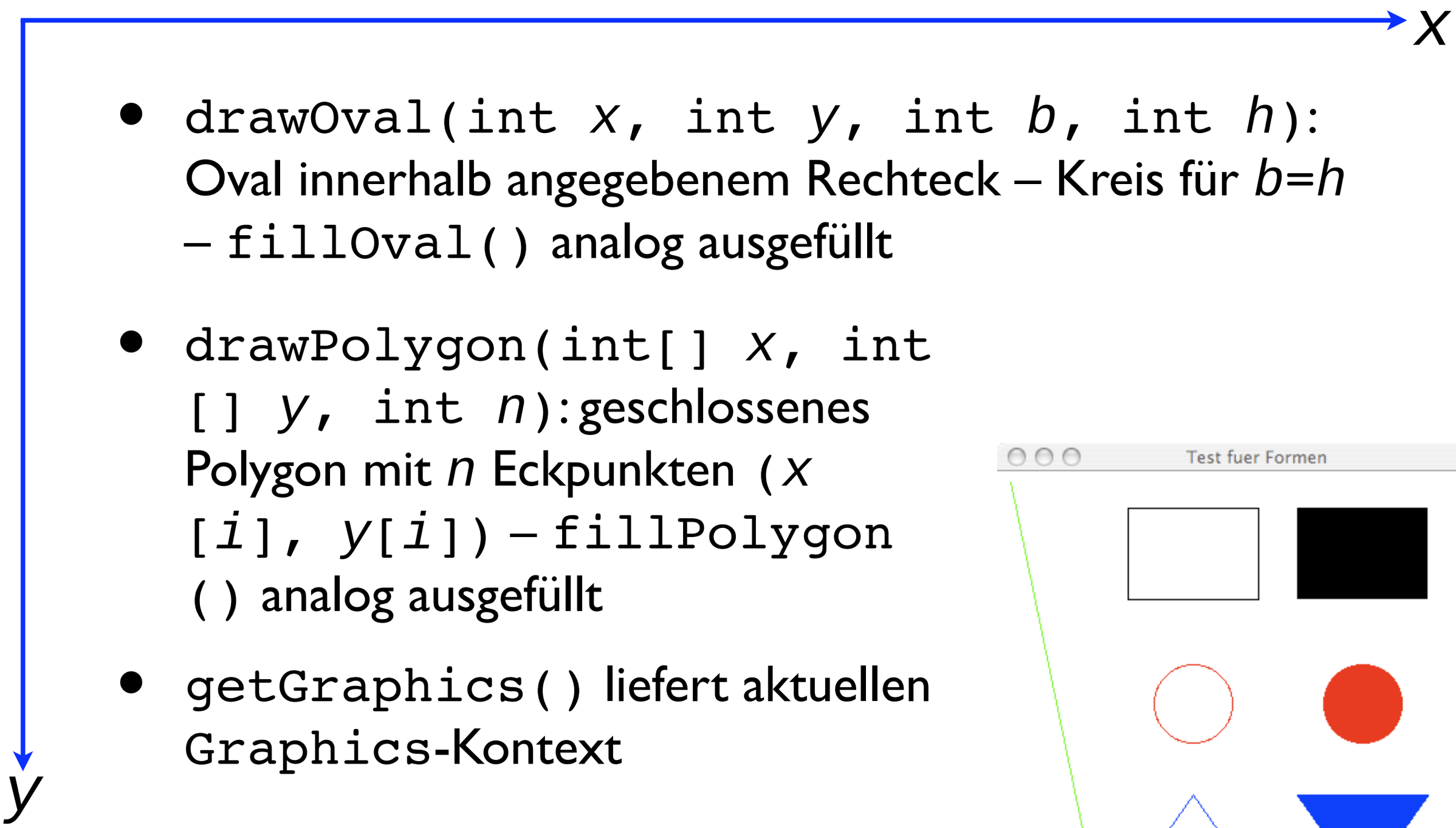
Zeichenmethoden



- Koordinatenursprung: links oben – *y* nimmt nach unten zu
- `drawLine(int x1, int y1, int x2, int y2)`: Linie von `(x1, y1)` nach `(x2, y2)`
- `drawRect(int x, int y, int b, int h)`: Rechteck mit Eckpunkten `(x, y)`, `(x+b, y)`, `(x, y+h)`, `(x+b, y+h)` – `fillRect()` analog ausgefüllt

⇒ Programm `Formen.java`

Zeichenmethoden



- `drawOval(int x, int y, int b, int h)`: Oval innerhalb angegebenem Rechteck – Kreis für *b*=*h* – `fillOval()` analog ausgefüllt
- `drawPolygon(int[] x, int [] y, int n)`: geschlossenes Polygon mit *n* Eckpunkten (`x[i], y[i]`) – `fillPolygon()` analog ausgefüllt
- `getGraphics()` liefert aktuellen `Graphics`-Kontext

⇒ Programm `Formen.java`