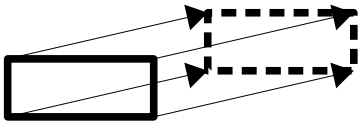
Affine Transformationen

Bisher haben wir diese Transformationen hingenommen, ohne sie besonders zu hinterfragen. Wir wollen uns nun ansehen, was eine Affine Transformation macht. Obwohl in der Mathematik Affine Transformationen für Räume beliebiger Dimension definiert sind, interessieren wir uns hier nur für die Affine Transformation unserer Zeichenebene. Eine Abbildung der Zeichenebene auf sich heißt *Affine Transformation*, wenn dabei Geraden auf Geraden und parallele Geraden wieder auf zu einander parallele Geraden abgebildet werden.Neben den von uns schon verwendeten Transformationen Translation und Rotation gibt es also noch weitere, z.B. die Spiegelung. (s.u.)

Bemerkenswert – und für uns von zentraler Bedeutung – ist die Tatsache, dass die Verkettung, also das Hintereinanderausführen zweier, dreier usw. Affine Transformationen wieder selbst eine Affine Transformation darstellt.

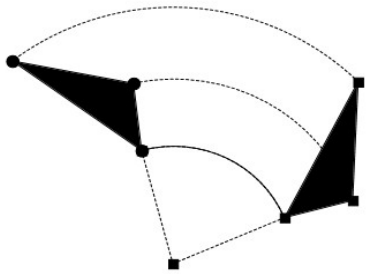
Translationen

Translationen sind Parallelverschiebungen. Zeichnerisch am Beispiel eines Rechtecks erläutert:
Punkt und Bildpunkt lassen sich jeweils mit paralellen und gleich langen Pfeilen verbinden



Rotationen

Rotationen sind Drehungen der Ebene um ein Zentrum. Dabei liegt zwischen Punkt, Zentrum und Bildpunkt jeweils ein gleich großer Winkel.



Darstellung der Bilder

Wir erreichen das gewünschte Bild, wenn wir die beiden Abbildungen miteinander verketten. In der einfachen Variante (unsere ersten Lösungen waren etwas komplizierter als notwendig) arbeiten wir mit einer einzelnen zunächst „leeren“ Transformation – der identischen Abbildung – und fügen ihr zuerst die Rotation und dann die Translation hinzu. Dieses Verketten der beiden Abbildungen (concatenate) erfolgt automatisch beim Aufruf der beiden Methoden rotate bzw. translate. Die Methode lautet dann:

```
protected Shape transformiere(Shape shape)
{
    AffineTransform t = new AffineTransform();
    t.translate(xPosition, yPosition);
    Rectangle2D umriss = shape.getBounds2D();
    t.rotate(Math.toRadians(orientierung),
             umriss.getX()+umriss.getWidth()/2,
             umriss.getY()+umriss.getHeight()/2);
    return  t.createTransformedShape(shape);  }
```

Mathematischer Hintergrund

Wer in der Mathematik in der Mittelstufe mit Vektoren gearbeitet hat, vermutlich werden die meisten sie aber erst in der Oberstufe im Bereich Lineare Algebra /Lineare Geometrie kennen gelernt haben bzw. kennen lernen, kann mit der JAVA – Klassendefinition von AffineTransform aus der JAVADOC ...
The AffineTransform class represents a 2D affine transform that performs a linear mapping from 2D coordinates to other 2D coordinates that preserves the "straightness" and "parallelness" of lines. Affine transformations can be constructed using sequences of translations, scales, flips, rotations, and shears.

Such a coordinate transformation can be represented by a 3 row by 3 column matrix with an implied last row of [0 0 1]. This matrix transforms source coordinates (x, y) into destination coordinates (x', y') by considering them to be a column vector and multiplying the coordinate vector by the matrix according to the following process: ...

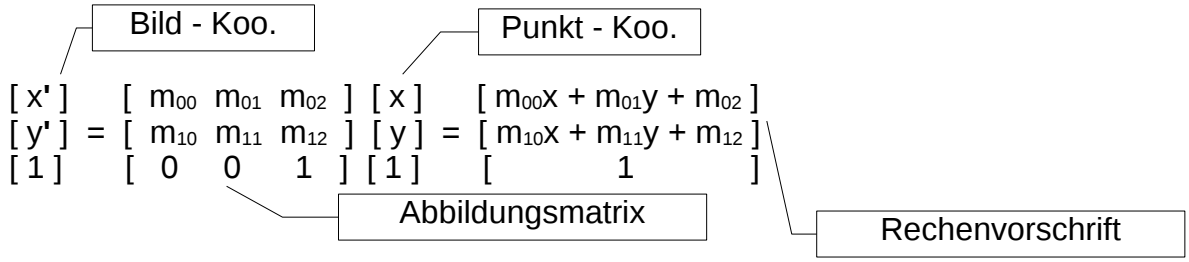
... oder mit dem folgenden Versuch einer Übersetzung der JAVA – Klassendefinition von AffineTransform hoffentlich etwas anfangen:

Die Klasse AffineTransform liefert eine zweidimensionale Affine Transformation der Zeichenebene, die ein lineare Abbildung von zweidimensionalen Koordinaten des Urbildes in zweidimensionale Koordinaten des Bildes ausführt, welche die Geradeneigenschaft und die Parallelität von Geraden bei der Abbildung erhalten. Sie lassen sich aus Translationen (Verschiebungen), Skalierungen (Streckungen), Spiegelungen, Rotationen (Drehungen), und Scherungen zusammensetzen .

Solche eine Koordinatentransformation kann durch eine 3 x 3 – Matrix dargestellt werden ...

$$\begin{vmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ 0 & 0 & 1 \end{vmatrix}$$

Durch die Matrix werden die Koordinaten des Originals in die Koordinaten des Bildes transformiert. ...



Diese Hilfe geht auf zwei Aspekte der Realisierung von Linearen Transformationen in JAVA ein.

1. Welche Abbildungen bezeichnet man als „Lineare Transformationen“ ?
   – Dies ist eigentlich eine mathematische Fragestellung

2. Wie werden diese unter JAVA realisiert ?
Zur ersten Fragestellung:
Als Lineare Transformationen (Linearen Abbildungen) bezeichnet man Abbildungen, bei denen Geraden im Bild immer noch Geraden sind und zwei parallele Geraden nach der Abbildung immer noch parallel sind.
Bekannt sind Ihnen sicher Spiegelungen, Verschiebungen und Drehungen, wohl auch noch die zentrische Streckung. Es gehören aber auch Scherungen und Achsstreckungen hinzu, sowie alle Kombinationen (Verknüpfungen) solcher Abbildungen.
Gemeinsam ist allen, dass die Umrechnungsfunktionen zwischen den Koordinaten vor und nach der Abbildung für alle Punkte gleich sind und *einfache Lineare Funktionen* darstellen.

Zur zweiten Fragestellung:
Sie werden in JAVA mit sogenannten „Homogenen Koordinaten“ realisiert. Obwohl die Punkte in einer Ebene liegen, also zweidimensional sind mit nur zwei Koordinaten, werden sie mit *drei* Koordinaten beschrieben.
Mit diesem „Trick“ ist es möglich, auch die Verschiebung, bei der zu allen Koordinaten Zahlen *addiert* werden, durch eine Matrizen*multiplikation* zu realisieren.

Beispiele:
1. Verschiebung:
Eine Verschiebung des Punktes P(100;50) um 20 in x-Richtung und 10 in y-Richtung hätte folgende Matrix[1]:

1	0	20
0	1	10
0	0	1

(100;50) → (120;60)

$$\begin{bmatrix} 120 \\ 60 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 20 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 100 \\ 50 \\ 1 \end{bmatrix} = \begin{bmatrix} 1*100 + 0*50 + 20*1 \\ 0*100 + 1*50 + 10*1 \\ 0*100 + 0*50 + 1*1 \end{bmatrix}$$

2. Drehung
Eine 90° - Linkssdrehung dieses Punktes um den Punkt (0;0) hätte folgende Matrix:

0	1	0
- 1	0	0
0	0	1

(100;50) → (50; - 100)

$$\begin{bmatrix} 50 \\ -100 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 100 \\ 50 \\ 1 \end{bmatrix} = \begin{bmatrix} 0*100 + 1*50 + 0*1 \\ -1*100 + 0*50 + 0*1 \\ 0*100 + 0*50 + 1*1 \end{bmatrix}$$

1 Die zweidimensionale Teilmatrix in der linken oberen Ecke entscheidet mit ihrer Determinante darüber, ob die Abbildung größenerhaltend ist: Ist ihr Betrag = 1, bleibt die Größe erhalten.

3. Zentrische Streckung vom Ursprung aus:
→ selbst herausfinden !

???	???	???
???	???	???
???	???	???

(100;50) → (???;???)

$$\begin{bmatrix} ??? \\ ??? \\ 1 \end{bmatrix} = \begin{bmatrix} ??? & ??? & ??? \\ ??? & ??? & ??? \\ ??? & ??? & ??? \end{bmatrix} \begin{bmatrix} 100 \\ 50 \\ 1 \end{bmatrix} = \begin{bmatrix} ??? & ??? & ??? \\ ??? & ??? & ??? \\ 0*100 + 0*50 + 1*1 \end{bmatrix}$$

Dieser „Trick“ hilft JAVA, die Verknüpfungen von *mehreren* dieser Abbildungen auf einfache Art durch Matrizenmultiplikation – und das ist eine einfache Standardberechnung für Computer – durchzuführen[1].

Wichtig für die Informatik ist: Wir brauchen das Grafikobjekt nur an eine Standardposition zu setzen und vor dem tatsächlichen Zeichnen einer Linearen Transformation zu unterwerfen. Diese liefert uns JAVA in der Klasse AffineTransform !

Aufgabe:

- Was sind jeweils *translations, scales, flips, rotations, shears* ?
- Untersuchen Sie die Typen Affiner Transformationen an einem Polygon.

1 Sollte Interesse bestehen, das im Kurs zu besprechen, dann können wir das gern tun, aber auch im LK Informatik ist das nicht zwingend notwendig.

Untersuchung der verschiedenen Typen von Transformationen

Es geht bei der Aufgabe zur Untersuchung der verschiedenen Transformationen um ein Ausprobieren. In der Beschreibung der Klasse **`AffineTransform`** in der JAVA Klassenbibliothek sind die folgenden Typen angegeben:
Affine transformations can be constructed using sequences of

- *translations*, Verschiebungen
- *scales*, Streckungen
- *flips*, Spiegelungen
- *rotations, and* Drehungen
- *shears* Scherungen

In der Lösung der Übungsaufgabe ist zu berücksichtigen, dass diese Abbildungen jeweils verschiedene Parameter benötigen:

- Die Verschiebung benötigt den Verschiebungsvektor, der hier mit den beiden Werten für die x – und die y – Richtung beschrieben wird.
- Die Streckung benötigt eigentlich vier Parameter, nämlich neben den Skalierungsfaktoren der Achsen auch diese Achsen selbst. Bei einer zentrischen Streckung wird durch den Achsenschnittpunkt das Zentrum definiert. Hier verwendet JAVA offensichtlich das Bildschirmkoordinatensystem des Objektes. Will man andere Zentren haben muss man diese durch eine Verkettung mit einer Verschiebung realisieren.
- Ausgerechnet zur Spiegelung gibt es laut JAVA Klassenbibliothek keine Methode. Ob man sie durch die Klassenvariable TYPE_IDENTITY mit dem Wert TYPE_FLIP realisieren kann, habe ich nicht untersucht.
- Die Drehung benötigt man natürlich das Drehzentrum und einen Winkel.
- Bei den Scherungen erhielte man mit den Parametern (0, 0) die identische Abbildung, die Parameter geben also an, wie weit in der Richtung der Achse verschoben werden soll.

Eine Lösung

In meiner Lösung wird in jedem Fall das Originalobjekt – hier ein Polygon – gezeichnet und zusammen mit ihm das Bild der Transformation. Das lässt sich leicht durch einen **`GeneralPath`** erreichen.

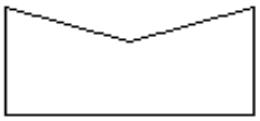
```
  /**
   * Berechnet das zu zeichnende Shape anhand der gegebenen Daten
   */
  protected Shape gibAktuelleFigur()
  {
      GeneralPath originalUndBild = new GeneralPath();
      int[] xKoordinaten = { 0, breite/2, breite, breite, 0 ,0} ;
      int[] yKoordinaten = { 0, tiefe/3, 0, tiefe, tiefe ,0} ;
      polygon = new Polygon(xKoordinaten, yKoordinaten, 6);
      originalUndBild.append(transformiere(polygon), false);
      originalUndBild.append(transformiere(polygon, aktuelleTransformation),
false);
      return originalUndBild;
  }

  /**
   * Fuehre eine Translation durch
   * Parameter: umX - in x-Richtung
   *            umY - in y-Richtung
   */
    public void verschiebe(int umX, int umY){
        verberge();
        aktuelleTransformation = new AffineTransform();
        aktuelleTransformation.translate(umX, umY);
        zeige();
    }
```

nur das kommt hinzu!

Die Methoden zu jeder Affinen Transformation – hier ist nur die Verschiebung angegeben – verbergen zunächst das bisher gezeichnete Objekt, um mehrere Darstellungen nacheinander zu ermöglichen, definieren dann die zugehörige **`aktuelleTransformation`** und rufen **`zeige()`** auf. Diese greift auf die Methode **`gibAktuelleFigur()`** zu, in der die Gesamtfigur gebildet wird.

Arrays



Unabhängig von dem speziellen Polygon, das man zeichnen will – hier hat es die rechts dargestellte Form – ist zu klären, wie JAVA die Daten der Eckpunkte verwaltet. Im Programmtext finden wir das in den Zeilen:

```
int[] xKoordinaten = { 0, breite/2, breite, breite, 0} ;
int[] yKoordinaten = { 0, tiefe/3, 0, tiefe, tiefe} ;
```

Der Aufruf des Konstruktors **`new Polygon(xKoordinaten, yKoordinaten, 5)`** und die Benennung der Variablen in den o.a. Zeilen hilft uns zu verstehen, wie es geht. Die Klasse Polygon verwaltet die Koordinaten der Eckpunkte in zwei integer – Arrays, getrennt für die x – Koordinaten und die y – Koordinaten. Im Konstruktor wird die Zahl der Eckpunkte definiert und die Figur wird zum ersten Punkt hin geschlossen.

Was sind integer – Arrays?

Array ist ein struktureller Datentyp, bei dem mehrere Variablenwerte gleichen Typs unter einem Variablennamen zusammengefasst werden. Der Zugriff, sowohl schreibend als auch lesend, erfolgt über diesen gemeinsamen Variablennamen zusammen mit der Angabe des Indexes, der dahinter in eckigen Klammern angegeben wird. Der Index beschreibt die Position innerhalb dieses Arrays, den der konkrete Wert hat, auf den man zugreifen will. Bei **`n`** vorhandenen Werten gehen die zulässigen Indizes von **`0 .. n – 1`**. Im o.a. Beispiel haben die fünf Eckpunkte daher die Koordinaten

```
(  0       ;  0        )
(  breite/2;  tiefe/3   )
(  breite  ;  0        )
(  breite  ;  tiefe    )
(  0       ;  tiefe    )
```