Wie zeichnet das Moebelprojekt ?

Situation: Der Benutzer erzeugt ein Schrankobjekt.

Dazu ruft er den Konstruktor der Klasse Schrank auf.

- Da Schrank von Moebel erbt, ruft der Konstruktor von Schrank (prinzipiell) zunächst den Konstruktor von Moebel auf (hat keinen, da Moebel abstract ist) und Moebel – wie jede Klasse – den von Object.
- Im Konstruktor von Schrank werden die Attributwerte initialisiert. Verwendet der Benutzer nicht den Standardkonstruktor, sondern einen Konstruktor, dem Parameter übergeben werden, dann werden dabei im Konstruktor die Parameterwerte in die Attribute übernommen.

Aufgabe:

Zeichnen Sie die Aktionen ein !

```
public abstract class Moebel
{
  protected int xPosition;
  protected int yPosition;
  protected int orientierung;
  protected String farbe;
  protected boolean istSichtbar;
  protected int breite;
  protected int tiefe;
```

```
public class Schrank extends Moebel
{
    public Schrank(int xPosition,
                   int yPosition,
                   String farbe,
                   int orientierung,
                   int breite,
                   int tiefe) {
    this.xPosition = xPosition;
    this.yPosition = yPosition;
    this.farbe = farbe;
    this.orientierung = orientierung;
    istSichtbar = false;
    this.breite = breite;
    this.tiefe  = tiefe;
  }
```

Situation: Der Benutzer ruft zeige() auf.

Dazu ruft er die Methode der Klasse Moebel Schrank auf.

- Das Schrankobjekt – wir wollen es einmal hier als schrank1 bezeichnen – stellt die Methode zeige() unter seinen von Moebel geerbten Methoden bereit.
- Die Methode zeige() setzt zunächst das Attribut istSichtbar auf true. Nun „weiß“ schrank1, dass er sichtbar sein soll. Anschließend ruft die Methode die Methode zeichne() auf.
- zeichne() ruft u.a. die Methode gibAktuelleFigur() auf. Dies ist die einzige in Schrank implementierte Methode!
- gibAktuelleFigur() erzeugt den GeneralPath für die Figur und übergibt diesen dann der Methode transformiere(...) aus Moebel.

Aufgabe:

Zeichnen Sie die Aktionen ein !

```
public abstract class Moebel
{
-~~~~~~-
  protected boolean istSichtbar;
-~~~~~~-
  public void zeige()
  {
    istSichtbar = true;
    zeichne();
  }
-~~~~~~-
  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);
  }
-~~~~~~-
  private void zeichne()
  {
    if (istSichtbar)
    {
      Shape figur = gibAktuelleFigur();
      Leinwand leinwand = Leinwand.gibLeinwand();
      leinwand.zeichne (this, farbe, figur);
      leinwand.warte(10);
    }
  }
```

```
public class Schrank extends Moebel
{
-~~~~~~-
   protected Shape gibAktuelleFigur()
  {
      GeneralPath schrank = new GeneralPath();
      Shape rahmen = new Rectangle2D.Double(0, 0, breite, tiefe);
      Shape linie1 = new Line2D.Double(0, 0, breite, tiefe);
      Shape linie2 = new Line2D.Double(breite, 0, 0, tiefe);
      schrank.append(rahmen, false);
      schrank.append(linie1, false);
      schrank.append(linie2, false);

      return  transformiere(schrank);
  }
```

Die Methode gibAktuelleFigur setzt die Schrankwand zusammen

Moebel – Objekte haben die Eigenschaft, dass man sie zeichnen kann. Dafür verwenden wir Shapes, also Klassen, die das interface Shape implementieren. Das sind zur Zeit laut Klassenbibliothek

- Arc2D, Arc2D.Double, Arc2D.Float,
- Area,
- BasicTextUI.BasicCaret, DefaultCaret,
- CubicCurve2D, CubicCurve2D.Double, CubicCurve2D.Float,
- Ellipse2D, Ellipse2D.Double, Ellipse2D.Float,
- GeneralPath,
- Line2D, Line2D.Double, Line2D.Float,
- Polygon,
- QuadCurve2D, QuadCurve2D.Double, QuadCurve2D.Float,
- Rectangle, Rectangle2D, Rectangle2D.Double, Rectangle2D.Float, RectangularShape,
- RoundRectangle2D, RoundRectangle2D.Double, RoundRectangle2D.Float

Bei allen diesen Klassen können wir darauf vertrauen[1], dass sie alle vom Interface Shape geforderten Methoden sinnvoll implementieren. Das sind:

- Rectangle getBounds()
- Rectangle2D getBounds2D()

ein Beispiel für overload

- boolean contains(double x, double y)
- boolean contains(Point2D p)
- boolean intersects(double x, double y, double w, double h)
- boolean intersects(Rectangle2D r)
- boolean contains(double x, double y, double w, double h)
- boolean contains(Rectangle2D r)
- PathIterator getPathIterator(AffineTransform at)
- PathIterator getPathIterator(AffineTransform at, double flatness)

Wir benötigen für unser Projekt (zunächst) aber nur wenige dieser Methoden. Sie selbst zu schreiben ist aufwändig und fehlerträchtig. Deshalb greifen wir nur über die oben angegebenen, schon von JAVA bereit gestellten Klassen auf diese Methoden zu.

In vielen Fällen wird die darzustellende Figur aber nicht einfach sein, sondern sich aus mehreren dieser einfachen Figuren zusammensetzen. In diesem Fall verwenden wir einen GeneralPath, der (zusätzlich zu den oben vorgeschriebenen Methoden, die alle Shapes implementieren müssen) die Methode

- public void append(Shape s, boolean connect)

bereit stellt. Ihr können wir beliebige Shapes übergeben und über den zweiten Parameter connect (dort kann nur true oder false übergeben werden) mitteilen, ob eine zusätzliche Verbindungslinie gezeichnet werden soll.

1 Aber, wie immer in der Informatik: alles kann fehlerhaft sein!

Aufgabe:

Füllen Sie ohne nachzusehen die Lücken aus und begründe Sie ihre Wahl!

```
protected ?      ? gibAktuelleFigur(){
   ?      ? schrank = new ?      ?;
   ?      ? rahmen = new Rectangle2D.Double(0, 0, breite, tiefe);
   ?      ? linie1 = new Line2D.Double(0, 0, breite, tiefe);
   ?      ? linie2 = new Line2D.Double(breite, 0, 0, tiefe);
   schrank.?      ?(rahmen, false);
   schrank.?      ?(linie1, false);
   schrank.?      ?(linie2, false);
   return  transformiere(?      ?);
  }
```

Aufgabe:

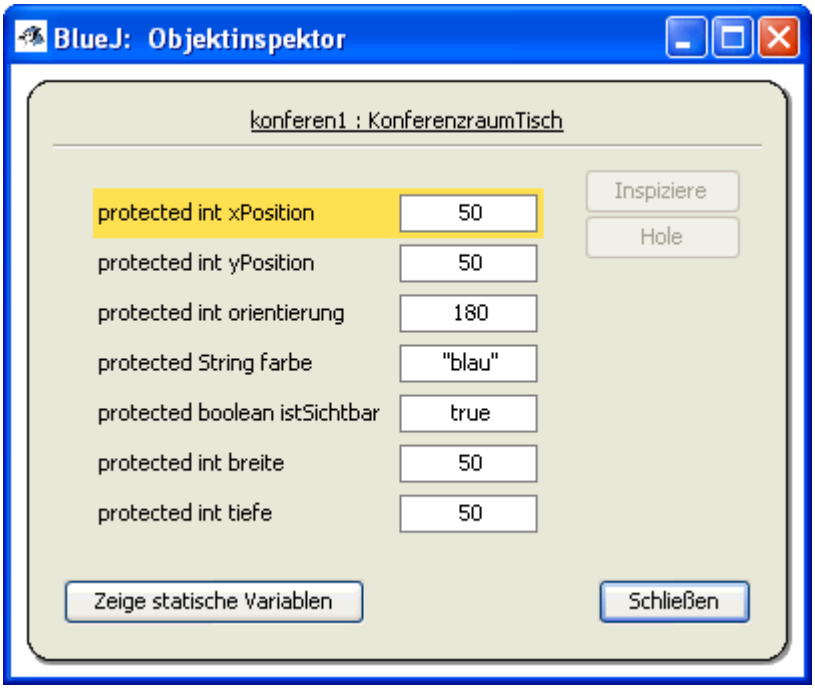
Füllen Sie ohne nachzusehen die Lücken aus und begründe Sie ihre Wahl!

```
ArrayList schraenke = new ArrayList();

protected ?      ? gibAktuelleFigur(){
   ?      ? schrankwand = new ?      ?;
   for (int i=0;i<anzahl;i++)
   schrankwand.?      ?(((?      ?) schraenke.get(i))
                              .?      ?, false);
   return  transformiere(?      ?);
}
```

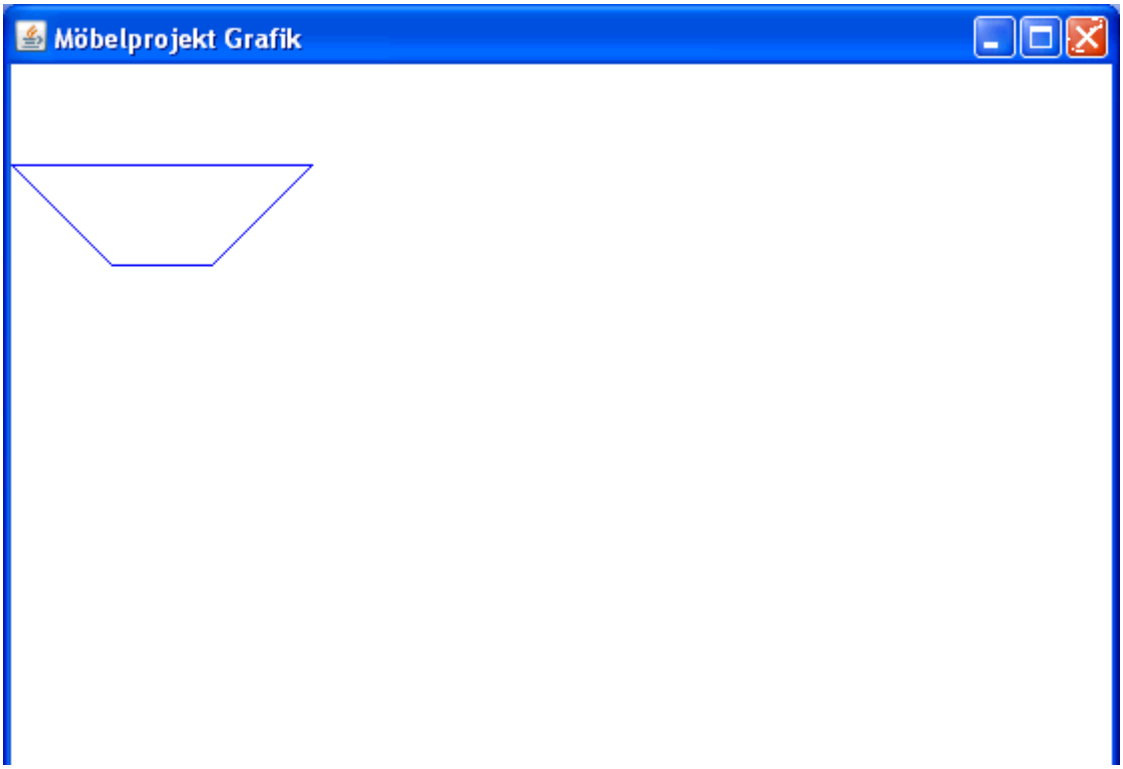
Konferenzraumtisch

Die Attributwerte des Konferenzraumtisches sind nicht so ganz einfach vorzugeben. Eine mögliche Entscheidung gibt als Breite und Tiefe nur den rechteckigen Abschnitt im Inneren vor. Sehr sinnvoll wäre sicher auch, das Rechteck zu verwenden, dass von der Methode getBounds2D() geliefert wird. Hier also zur ersten Variante ein Tisch und seine im Inspektor angezeigten Attributwerte:



Aufgabe:

Zeichnen Sie die nächsten Konferenzraumtische ein, die mit dem vorgegebenen zusammen ein U bilden und geben Sie die Attributwerte an, die dazu von diesen eingenommen werden müssen !



Formulieren Sie die Schleifen !