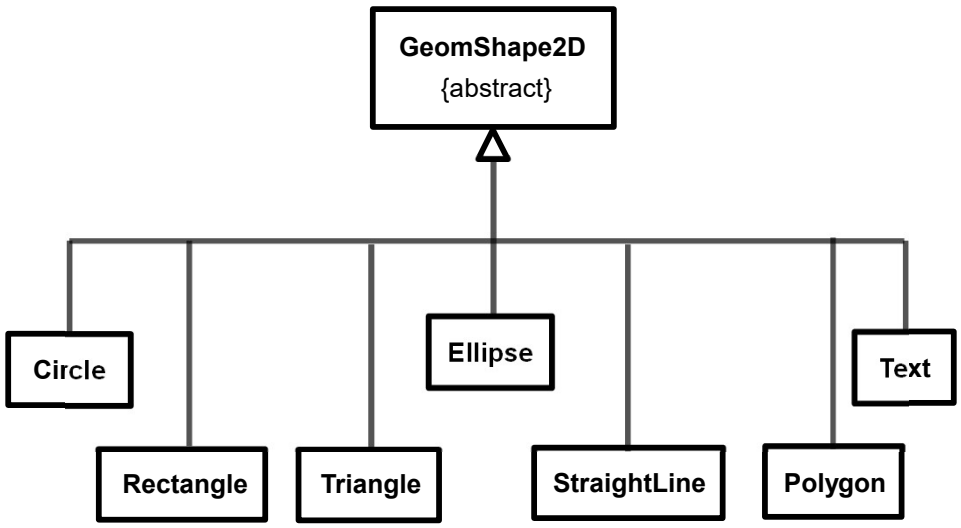
TECHNISCHE UNIVERSITÄT DARMSTADT

Kapitel 02:
Fallbeispiel Graphik

Karsten Weihe

Fallbeispiel Graphik

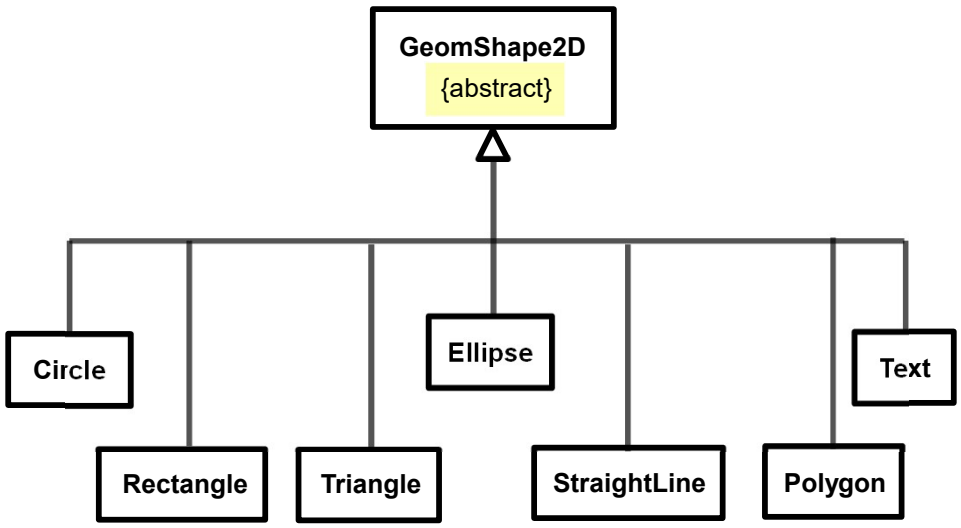
TECHNISCHE UNIVERSITÄT DARMSTADT



Worum geht's in diesem Fallbeispiel inhaltlich: Hierarchien von Entitäten gibt es überall in der Welt. Konkret geht es jetzt um zweidimensionale geometrische Formen. Diese Entität werden wir als eine Klasse GeomShape2D umsetzen, und von der werden wir diverse Klassen ableiten für die verschiedenen speziellen Arten von geometrischen Formen. Zum Beispiel Kreis, Ellipse oder Rechteck, aber auch Text sind geometrische Formen.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT



Das Wort „abstract“ an dieser Stelle in geschweiften Klammern besagt, dass die Klasse geomShape2D eine abstrakte Klasse ist. Was das konkret bedeutet, werden wir gleich sehen.

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorarbeit: Applets

In Kapitel 10 werden wir die Standardwege betrachten, wie GUIs in Java implementiert werden. Hier, in Kapitel 02, reicht uns zur Demonstration die Anbindung an eine typischerweise eher leichtgewichtige Variante von graphikorientierten Programmen: Applets.

Applets

TECHNISCHE UNIVERSITÄT DARMSTADT

```
<object codebase=
     "http://www.algo.informatik.tu-darmstadt.de/fop”
  classid="java:applets.MyApplet.class"
  codetype="application/x-java-applet”
  width="800”
  height="600”
  >
</object>
```

Wir werden ein kleines Programm schreiben, das sich in eine HTML-Seite integrieren lässt. So sieht die Integration in HTML aus. Wir werden hier so wenig wie möglich zu HTML sagen, da dies nicht unser Thema, sondern nur Mittel zum Zweck ist.

Applets

TECHNISCHE UNIVERSITÄT DARMSTADT

```
<object codebase=
      "http://www.algo.informatik.tu-darmstadt.de/fop”
   classid="java:applets.MyApplet.class"
   codetype="application/x-java-applet”
   width="800”
   height="600”
   >
</object>
```

Das Object-Tag ist *ein* geeignetes HTML-Konstrukt.

Applets

TECHNISCHE UNIVERSITÄT DARMSTADT

```
<object codebase=
      "http://www.algo.informatik.tu-darmstadt.de/fop”
  classid="java:applets.MyApplet.class"
  codetype="application/x-java-applet”
  width="800”
  height="600”
  >
</object>
```

Was wir in die HTML-Seite integrieren wollen, ist eben ein Applet, also eine kleine Applikation, eine kleine Anwendung.

Applets

TECHNISCHE UNIVERSITÄT DARMSTADT

```
<object codebase=
      "http://www.algo.informatik.tu-darmstadt.de/fop”
  classid="java:applets.MyApplet.class"
  codetype="application/x-java-applet”
  width="800”
  height="600”
  >
</object>
```

Die Applet-Klasse, die wir in Java schreiben und dann integrieren wollen, nennen wir MyApplet.

Applets

TECHNISCHE UNIVERSITÄT DARMSTADT

```
<object codebase=
    "http://www.algo.informatik.tu-darmstadt.de/fop”
  classid="java:applets.MyApplet.class"
  codetype="application/x-java-applet”
  width="800”
  height="600”
  >
</object>
```

Und das ist die Stelle, an der der Code für MyApplet zu finden ist. Das ist hier allerdings nur ein illustrativer Platzhalter; diese URL gibt es nicht.

Applets

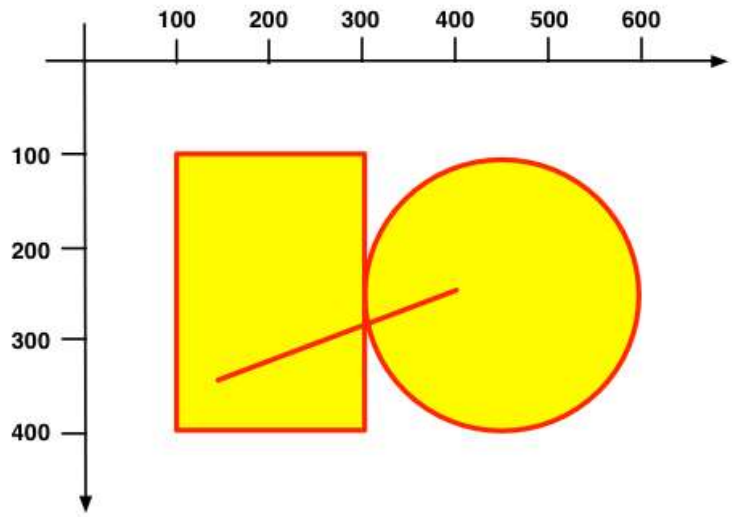
TECHNISCHE UNIVERSITÄT DARMSTADT

```
<object codebase=
      "http://www.algo.informatik.tu-darmstadt.de/fop”
  classid="java:applets.MyApplet.class"
  codetype="application/x-java-applet”
  width="800”
  height="600”
  >
</object>
```

Hier legen wir die Größe der Darstellung in Pixeln fest. Im Zeitalter von Tablets und Smartphones ist es sicher nicht besonders geschickt, die Größe der Darstellung unabhängig von der Größe des Displays festzulegen, aber wir wollen uns in diesem illustrativen Beispiel ja so wenig wie möglich mit HTML befassen, also belassen wir es hier dabei.

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT



Unser Applet soll nicht viel tun, einfach beim Aufruf der HTML-Seite die drei gelbroten Figuren an die Stelle der Seite zeichnen, die für das Applet vorgesehen ist. Es geht ja nur darum zu illustrieren, wie das alles funktioniert. Das Koordinatensystem wird *nicht* mitgezeichnet, das verwenden wir nur zur Orientierung auf den Folien.

Applets

TECHNISCHE UNIVERSITÄT DARMSTADT

```
import java.awt.Applet;
import java.awt.Graphics;

public class MyApplet extends Applet {
   public void paint ( Graphics graphics ) {
      .....
   }
}
```

Als nächstes schreiben wir die Applet-Klasse.

Applets

TECHNISCHE UNIVERSITÄT DARMSTADT

```
import java.awt.Applet;
import java.awt.Graphics;

public class MyApplet extends Applet {
    public void paint ( Graphics graphics ) {
        .....
    }
}
```

Eine Applet-Klasse ist einfach eine Klasse, die von Klasse Applet abgeleitet ist.

Applets

TECHNISCHE UNIVERSITÄT DARMSTADT

```
import java.awt.Applet;
import java.awt.Graphics;

public class MyApplet extends Applet {
  public void paint ( Graphics graphics ) {
      .....
  }
}
```

Wenn der HTML-Interpreter das Applet ausführen will, dann ruft es konkret diese Methode paint von Klasse Applet auf. Diese Methode wird überschrieben, um das eigene Applet zu realisieren.

Applets

TECHNISCHE UNIVERSITÄT DARMSTADT

```
import java.awt.Applet;
import java.awt.Graphics;

public class MyApplet extends Applet {
   public void paint ( Graphics graphics ) {
      ..........
   }
}
```

Die Klasse Graphics verknüpft das Programm mit der Zeichenfläche. Bei Applets ist das die Fläche, die beim Aufruf der HTML-Seite für das Applet reserviert ist. In Kapitel 10, Abschnitt zu Canvas, werden wir sehen, dass Klasse Graphics ein Programm auch mit anderen Arten von Zeichenflächen verknüpfen kann.

Wie wir im Laufe des vorliegenden Kapitels sehen werden, hat Graphics einige Methoden zum Zeichnen von geometrischen Objekten. Diese Zeichenanweisungen werden auf der Zeichenfläche umgesetzt.

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyApplet extends Applet {
   public void paint ( Graphics graphics ) {
      GeomShape2D[ ] picture = new GeomShape2D [ 3 ];
      picture[0] = new Circle ( 450, 250, 100 );
      picture[1] = new Rectangle ( 100, 100, 200, 300 );
      picture[2] = new StraightLine ( 150, 350, 400, 250 );
      for ( int i = 0; i < picture.length; i++ ) {
          picture[i].setBoundaryColor ( Color.RED );
          picture[i].setFillColor ( Color.YELLOW );
          picture[i].paint ( graphics );
      }
   }
}
```

Das ist der gesamte Code von Methode paint selbst. Wie Sie aber sehen, müssen wir offenbar noch ein paar Klassen definieren, die wir eingangs in der Hierarchie gesehen hatten: die Basisklasse GeomShape2D und die davon abzuleitenden Klassen für Kreise, Rechtecke und Strecken.

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyApplet extends Applet {
   public void paint ( Graphics graphics ) {
      GeomShape2D[ ] picture = new GeomShape2D [ 3 ];
      picture[0] = new Circle ( 450, 250, 100 );
      picture[1] = new Rectangle ( 100, 100, 200, 300 );
      picture[2] = new StraightLine ( 150, 350, 400, 250 );
      for ( int i = 0; i < picture.length; i++ ) {
          picture[i].setBoundaryColor ( Color.RED );
          picture[i].setFillColor ( Color.YELLOW );
          picture[i].paint ( graphics );
      }
   }
}
```

Das Bild ist realisiert als ein Array der Basisklasse, und die Objekte sind von den abgeleiteten, formspezifischen Klassen.

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyApplet extends Applet {
   public void paint ( Graphics graphics ) {
      GeomShape2D[ ] picture = new GeomShape2D [ 3 ];
      picture[0] = new Circle ( 450, 250, 100 );
      picture[1] = new Rectangle ( 100, 100, 200, 300 );
      picture[2] = new StraightLine ( 150, 350, 400, 250 );
      for ( int i = 0; i < picture.length; i++ ) {
         picture[i].setBoundaryColor ( Color.RED );
         picture[i].setFillColor ( Color.YELLOW );
         picture[i].paint ( graphics );
      }
   }
}
```

Hier wird das Bild gezeichnet. Die Basisklasse braucht also drei Methoden: zum Setzen der Randfarbe, zum Setzen der Füllfarbe und zum eigentlichen Zeichnen.

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorarbeit:
Abstrakte Methoden und Klassen

Bevor wir mit den Graphikklassen loslegen, schauen wir uns kurz noch ein Konstrukt in Java an, das wir im Fallbeispiel brauchen werden.

Abstrakte Methoden + Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
abstract public class X {
    .....
    public void m1 () { ..... }
    abstract public void m2 ();
    .....
}
```

Dieses Java-Konstrukt ist eigentlich recht einfach, ein einziges, rein illustratives Beispiel sollte reichen.

Abstrakte Methoden + Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
abstract public class X {
    .....
    public void m1 () { ..... }
    abstract public void m2 ();
    .....
}
```

Man kann Methoden einer Klasse abstract definieren, einfach indem man das Schlüsselwort abstract an den Anfang der Methode schreibt. Eine solche Methode nennt man dementsprechend auch abstrakt. In diesem Beispiel haben wir nur eine abstrakte Methode, aber eine Klasse kann beliebig viele abstrakte Methoden haben. Sie kann auch beliebig viele abstrakte und beliebig viele nichtabstrakte Methoden in beliebiger Reihenfolge haben.

Abstrakte Methoden + Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
abstract public class X {
    .....
    public void m1 () { ..... }
    abstract public void m2 ();
    .....
}
```

Der Unterschied zwischen einer abstrakten und einer nichtabstrakten Methode ist, dass die abstrakte Methode keinen Methodenrumpf hat. Statt dessen wird die Definition der abstrakten Methode mit einem Semikolon abgeschlossen.

Alles, was wir bisher gesehen hatten, waren also nichtabstrakte Methoden, und nun sehen wir mit m2 erstmals eine abstrakte Methode.

Abstrakte Methoden + Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
abstract public class X {
    .....
    public void m1 () { ..... }
    abstract public void m2 ();
    .....
}
```

Wenn eine Klasse eine oder mehrere abstrakte Methoden hat, muss das Schlüsselwort abstract zwingend auch an den Beginn der Klassendefinition geschrieben werden.

***Nebenbemerkung*: Bei einer Klasse *ohne* abstrakte Methoden *darf* das Schlüsselwort abstract hier am Beginn der Klassendefinition stehen, *muss* aber nicht. Bisher haben wir abstract nie hingeschrieben, obwohl wir das gedurft hätten; es gibt auch nur seltene Situationen, in denen das sinnvoll ist. In Kapitel 10, Abschnitt „Weitere Listener- und Event-Typen“, werden wir drei Beispiele namens KeyAdapter, MouseAdapter und WindowAdapter dafür sehen, dass das manchmal sinnvoll ist.**

Abstrakte Methoden + Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
abstract public class X {
    .....
    public void m1 () { ..... }
    abstract public void m2 ();
    .....
}
```

Wie schon gesagt: Selbstverständlich darf eine abstrakte Klasse beliebig viele abstrakte und auch beliebig viele *nicht*abstrakte Methoden haben, in beliebiger Reihenfolge.

Abstrakte Methoden + Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class X {
   .........
   public void m1 () { ......... }
   abstract public void m2 ();
   .........
}

public class Y extends X {
   public void m2 () { ......... }
}
```

```
X a = new X();

Y b = new Y();

X c = new Y();
```

Von einer abstrakten Klasse lassen sich keine Objekte mit Operator new erzeugen. Daher wird man abstrakte Klassen immer so verwenden, dass man nichtabstrakte Klassen davon ableitet, also die abstrakten Methoden nichtabstrakt in den abgeleiteten Klassen überschreitet. Das sehen wir uns jetzt an.

Links oben sehen Sie noch einmal dieselbe Definition der Klasse X wie auf der letzten Folie.

Abstrakte Methoden + Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class X {
   .....
   public void m1 () { ..... }
   abstract public void m2 ();
   .....
}

public class Y extends X {
   public void m2 () { ..... }
}
```

```
X a = new X();

Y b = new Y();

X c = new Y();
```

Jetzt leiten wir eine Klasse Y von X ab und implementieren in Y die Methode m2, die in X abstrakt ist. Die abstrakte Methode m2 aus Klasse X wird in Klasse Y also nichtabstrakt überschrieben.

Abstrakte Methoden + Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class X {
   .....
   public void m1 () { ..... }
   abstract public void m2 ();
   .....
}

public class Y extends X {
   public void m2 () { ..... }
}
```

```
X a = new X();

Y b = new Y();

X c = new Y();
```

Von einer abstrakten Klasse dürfen, wie gesagt, keine Objekte erzeugt werden. Versuchen wir es trotzdem wie hier, wird der Compiler die Übersetzung mit einer Fehlermeldung abbrechen.

Dieses Verbot ist sinnvoll, denn wenn eine oder mehrere Methoden für diese Klasse nicht implementiert sind, dann ist es sehr gefährlich, Objekte von dieser Klasse zu haben, denn die Anwendung einer abstrakten Methode kann ja nichts Sinnvolles sein, also von vornherein ausgeschlossen.

Abstrakte Methoden + Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class X {
   .........
   public void m1 () { ......... }
   abstract public void m2 ();
   .........
}

public class Y extends X {
   public void m2 () { ......... }
}
```

```
X a = new X();

Y b = new Y();

X c = new Y();
```

Die von X abgeleitete Klasse Y ist *nicht* abstrakt, alle Methoden sind implementiert, daher ist es erlaubt und wie immer problemlos möglich, Objekte von Y mit Operator new zu erzeugen.

Abstrakte Methoden + Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class X {
    .....
    public void m1 () { ..... }
    abstract public void m2 ();
    .....
}

public class Y extends X {
    public void m2 () { ..... }
}
```

```
X a = new X();

Y b = new Y();

X c = new Y();
```

Die dritte Zeile rechts zeigt, dass es wirklich nur um das Objekt geht. Eine *Referenz* einer abstrakten Klasse wie X ist absolut möglich, warum auch nicht. Da das Objekt, auf das die Referenz verweist, auf jeden Fall von einer nichtabstrakten Klasse ist, sind alle Methoden implementiert, und es kann gar nichts schiefgehen, also erlaubt.

Abstrakte Methoden + Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class X {
    .....
}

public abstract class Z extends X {
    .....
}

public class Y extends Z {
    .....
}
```

```
X a = new X();
Y b = new Y();
X c = new Y();
Z d = new Z();
Z e = new Y();
```

Das alles gilt natürlich analog, wenn Y nicht direkt, sondern indirekt, also über eine oder mehrere Zwischenklassen wie hier Z von X abgeleitet ist. Eine abstrakte Methode kann in einer abgeleiteten Klasse einfach vererbt sein, dann ist sie auch dort abstrakt, und es kann kein Objekt eingerichtet werden, erst von einer nichtabstrakten davon abgeleiteten Klasse.

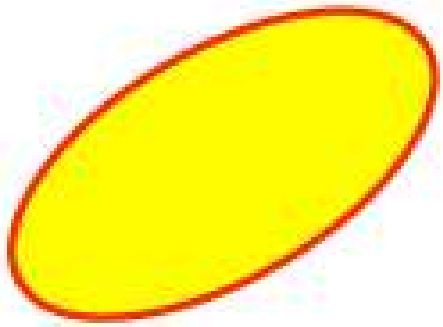
TECHNISCHE UNIVERSITÄT DARMSTADT

Das eigentliche Fallbeispiel Graphik

Nach dieser Vorarbeit können wir jetzt zum eigentlichen Fallbeispiel kommen.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

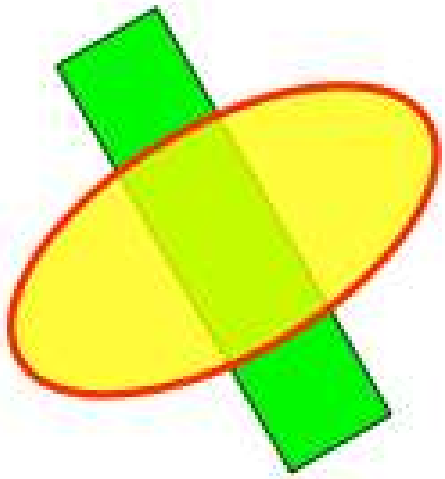


Alle Arten von geometrischen Formen haben einige Attribute gemeinsam, zum Beispiel Randfarbe und Füllfarbe.

***Nebenbemerkung*: Bei geometrischen Formen, die nur aus Linien bestehen, also kein Inneres haben, stellt sich die Frage, ob es eine Füllfarbe geben soll. Wir wählen hier den Weg, der beispielsweise auch für viele interaktive Zeichenprogramme gewählt wurde: Auch geometrische Objekte ohne Inneres haben pro forma eine Füllfarbe, die aber beim Zeichnen keinen Effekt hat.**

Fallbeispiel Graphik

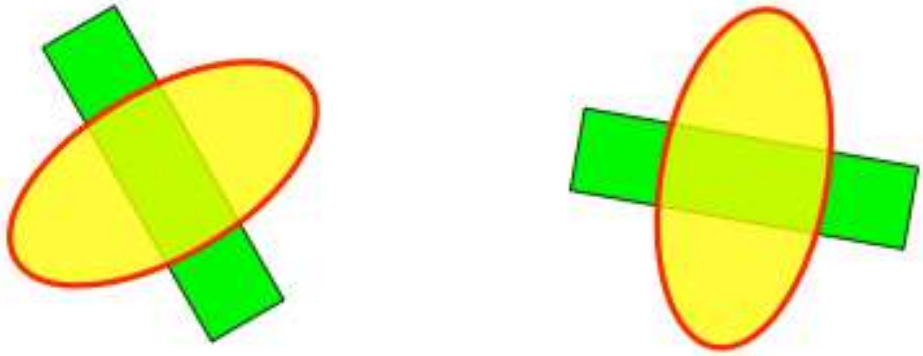
TECHNISCHE UNIVERSITÄT DARMSTADT



Jedes geometrische Objekt hat einen Transparenzwert irgendwo zwischen 0 und 100 Prozent. Die Ellipse ist über das grüne Rechteck drüber gezeichnet, hat aber einen mittleren Transparenzwert, so dass das grüne Rechteck dahinter noch etwas durchscheint.

Fallbeispiel Graphik

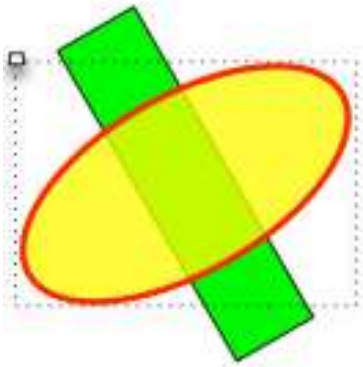
TECHNISCHE UNIVERSITÄT DARMSTADT



Noch ein Attribut, das allen geometrischen Objekten gemeinsam sein soll: Hier sind beide Objekte um denselben Winkel und denselben Punkt rotiert, so dass das Bild als Ganzes rotiert ist.

Fallbeispiel Graphik

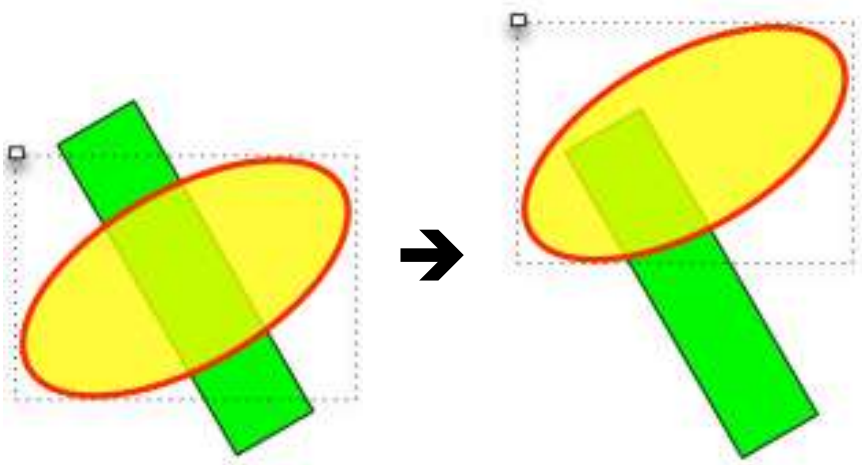
TECHNISCHE UNIVERSITÄT DARMSTADT



Die Position eines geometrischen Objektes wird angegeben durch einen Referenzpunkt. Dessen x- und y-Koordinate sind zwei weitere Attribute jedes geometrischen Objektes. In vielen Programmen wird dafür der linke obere Eckpunkt des umfassenden achsenparallelen Rechtecks genommen. Das machen wir in unserem Fallbeispiel auch. Gezeigt sind das umfassende Rechteck und der Referenzpunkt für die Ellipse.

Fallbeispiel Graphik

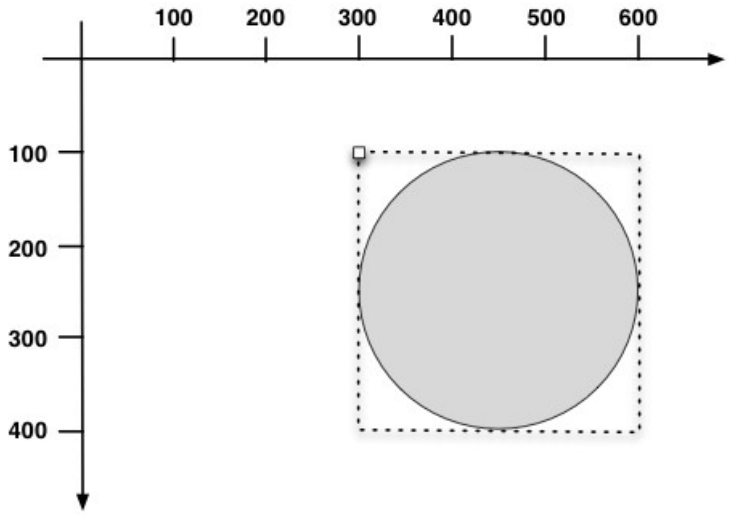
TECHNISCHE UNIVERSITÄT DARMSTADT



Durch Verschieben des Referenzpunktes lässt sich das ganze Objekt verschieben. Die Logik dahinter ist ganz einfach: Wann immer das Objekt gezeichnet wird, wird es relativ zu seinem Referenzpunkt gezeichnet.

Fallbeispiel Graphik

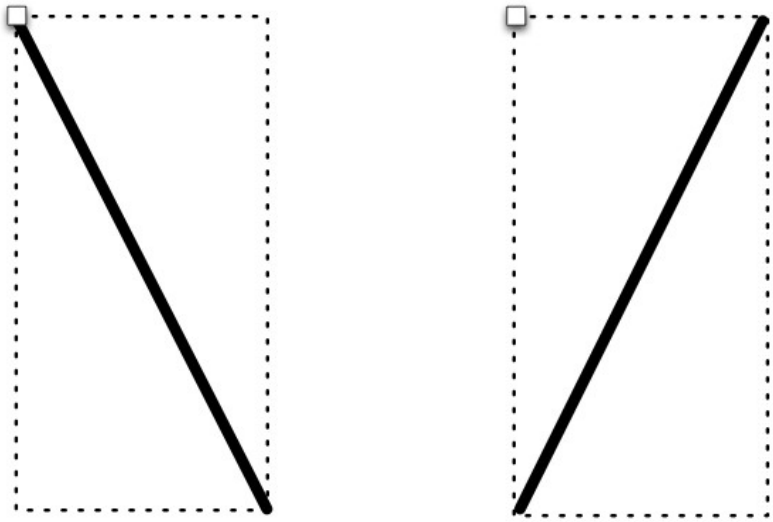
TECHNISCHE UNIVERSITÄT DARMSTADT



Beispielsweise bei diesem Kreis ist der Punkt mit x-Koordinate 300 und y-Koordinate 100 der Referenzpunkt. Der Nullpunkt der Zeichenfläche ist in der Informatik meist links oben wie hier.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT



Noch ein Beispiel: Bei Strecken ist der Referenzpunkt je nach Steigungswinkel jeweils der hier eingezeichnete Punkt, entweder einer der beiden Endpunkte oder eben nicht.

TECHNISCHE UNIVERSITÄT DARMSTADT

Klasse GeomShape2D

Wir gehen jetzt in die Java-Details, als erstes die Basisklasse.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class GeomShape2D {
  protected int positionX;
  protected int positionY;
  protected int rotationAngle;
  protected int transparencyValue;
  protected Color boundaryColor;
  protected Color fillColor;
  ..........
}
```

Hier sehen Sie einen Ausschnitt aus dieser Klasse.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class GeomShape2D {
  protected int positionX;
  protected int positionY;
  protected int rotationAngle;
  protected int transparencyValue;
  protected Color boundaryColor;
  protected Color fillColor;
  ..........
}
```

Etliche Bestandteile der Klasse werden an diesem Punkt erst einmal ausgelassen und später in diesem Kapitel betrachtet.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class GeomShape2D {
  protected int positionX;
  protected int positionY;
  protected int rotationAngle;
  protected int transparencyValue;
  protected Color boundaryColor;
  protected Color fillColor;
  ..........
}
```

Wie gesagt, wird GeomShape2D eine abstrakte Klasse sein. Der Grund ist, dass es Methoden gibt, die man für beliebige geometrische Formen gar nicht sinnvoll implementieren kann, nur für die einzelnen geometrischen Formarten individuell, also in den einzelnen abgeleiteten Klassen jeweils unterschiedlich implementiert. Trotzdem werden wir gleich sehen, dass wir solche Methoden auch in der Basisklasse haben wollen, obwohl sie dort auf den ersten Blick keinen Sinn zu haben scheinen.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class GeomShape2D {
  protected int positionX;
  protected int positionY;
  protected int rotationAngle;
  protected int transparencyValue;
  protected Color boundaryColor;
  protected Color fillColor;
  ..........
}
```

***Erinnerung*: In Kapitel 01f, Abschnitt „Zugriffsrechte und Packages“, haben wir gesehen, dass protected dafür sorgt, dass auf diese Attribute in der eigenen Klasse GeomShape2D, im Package von GeomShape2D und in allen von GeomShape2D direkt oder indirekt abgeleiteten Klassen zugegriffen werden kann.**

Die Attribute von GeomShape2D sind protected definiert, damit die abgeleiteten Klassen, die die verschiedenen geometrischen Formarten realisieren, darauf zugreifen können, denn auch die Methoden in diesen abgeleiteten Klassen müssen mit den hier gezeigten Attributen der Basisklasse arbeiten.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class GeomShape2D {
  protected int positionX;
  protected int positionY;
  protected int rotationAngle;
  protected int transparencyValue;
  protected Color boundaryColor;
  protected Color fillColor;
  ..........
}
```

Das sind die beiden Koordinaten des Referenzpunktes, und zwar als Pixelpunkte, daher vom Typ int.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class GeomShape2D {
  protected int positionX;
  protected int positionY;
  protected int rotationAngle;
  protected int transparencyValue;
  protected Color boundaryColor;
  protected Color fillColor;
  ..........
}
```

Die weiteren vier Attribute, die wir betrachtet hatten.

***Vorgriff*: Die beiden Farbattribute sind von einer Klasse namens Color, die sich in einem Package namens java.awt findet und die wir in Kapitel 10 wiedersehen werden. Ein Objekt vom Typ Color hat drei Attribute: Rot-, Grün- und Blauwert. Aus diesen drei Werten lassen sich bekanntlich alle für uns wahrnehmbaren Farben mischen.**

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
.....

public int getRotationAngle () {
   return rotationAngle;
}

public void setRotationAngle ( int angle ) {
   rotationAngle = angle;
}

.....
```

Jetzt kommen wir zu den Methoden der Klasse GeomShape2D. Wir betrachten hier noch nicht alle Methoden, nur die get- und set-Methoden und den Konstruktor.

Es ist generelle Konvention, jedes Attribut private oder protected zu definieren und eine get-Methode sowie bei variablen Attributen auch eine set-Methode zu diesem Attribut zu definieren, mit denen der Wert des Attributs gelesen beziehungsweise überschrieben wird. Das haben wir auch bei Klasse Robot in FopBot schon gesehen.

Gleich, bei der von GeomShape2D abgeleiteten Klasse Circle, werden wir den Sinn dahinter studieren können.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
.....

public int getRotationAngle () {
    return rotationAngle;
}

public void setRotationAngle ( int angle ) {
    rotationAngle = angle;
}

.....
```

Es ist ebenfalls Konvention, die Zugriffsmethoden genau so zu nennen: get beziehungsweise set vor dem Namen des Attributs. Nach allgemeiner Konvention für Identifier muss dann der erste Buchstabe des Attributnamens groß sein, da das der erste Buchstabe des *zweiten* Wortes im Methodennamen ist.

Der interne Name des private- oder protected-definierten Attributs muss nicht wie hier identisch mit dem Attributnamen sein, der im get-/set-Methodenpaar kodiert ist. Manchmal macht es Sinn, nach außen einen anderen Namen zu propagieren als intern. Gleich, bei der abgeleiteten Klasse Circle, werden wir sehen, dass es nicht einmal ein internes Attribut geben muss.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
.....

public int getRotationAngle () {
    return rotationAngle;
}

public void setRotationAngle ( int angle ) {
    rotationAngle = angle;
}

.....
```

Logischerweise hat die get-Methode als Rückgabetyp den Typ des Attributs, und die set-Methode hat einen einzigen Parameter, der ebenfalls vom Typ des Attributs ist.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
.....

public Color getBoundaryColor () {
    return boundaryColor;
}

public void setBoundaryColor ( Color color ) {
    boundaryColor = color;
}

.....
```

Für ein Attribut von einem Referenztyp ist das get-/set-Methodenpaar völlig analog. Wir wissen zwar momentan noch nicht genau, wie die Klasse Color überhaupt aussieht, das ist aber hier auch noch nicht notwendig.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class GeomShape2D {
  ..........
  public GeomShape2D
                ( int posX, int posY, ..... ) {
    positionX = posX;
    positionY = posY;
    ..........
  }
  ..........
}
```

Wir schauen uns ganz kurz noch den Konstruktor an, bevor wir zu den von GeomShape2D abgeleiteten Klassen kommen.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class GeomShape2D {
  ..........
  public GeomShape2D
                ( int posX, int posY, ..... ) {
    positionX = posX;
    positionY = posY;
    ..........
  }
  ..........
}
```

Beispielhaft sehen Sie hier nur die ersten beiden Attribute.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public abstract class GeomShape2D {
  ..........
  public GeomShape2D
                ( int posX, int posY, .......... ) {
    positionX = posX;
    positionY = posY;
    ..........
  }
  ..........
}
```

Die anderen Attribute von GeomShape2D sind auch hier völlig analog.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Circle extends GeomShape2D {
    protected int radius;
    public int getRadius () {
        return radius;
    }
    public void setRadius ( int rad ) {
        radius = rad;
    }
    .....
}
```

Jetzt kommen wir zur angekündigten Klasse Circle, abgeleitet von GeomShape2D. Ein Kreis hat einen Radius. Ein Radius macht für allgemeine geometrische Formen keinen Sinn und war daher in GeomShape2D noch nicht definiert. Aber in Klasse Circle wird dieses Attribut jetzt benötigt, zusammen mit dem get-/set-Methodenpaar für den Radius.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Circle extends GeomShape2D {
    ..........
    public int getCenterX () {
        return positionX + radius;
    }
    public void setCenterX ( int centerX ) {
        positionX = centerX − radius;
    }
    ..........
}
```

Ein Kreis hat auch einen Mittelpunkt, aber da sieht die Situation anders aus. In Klasse GeomShape2D sind ja schon die Koordinaten des Referenzpunktes als Attribute definiert. Referenzpunkt und Mittelpunkt unterscheiden sich in beiden Koordinaten nur um den Radius. Daher braucht der Mittelpunkt *nicht* in einem Kreis zusätzlich definiert werden, sondern kann einfach bei Bedarf berechnet werden. Wir sehen nur beispielhaft die x-Koordinate des Mittelpunktes; die y-Koordinate ist natürlich völlig analog.

Dies ist das soeben angekündigte Beispiel dafür, dass nach außen hin durch das get-/set-Methodenpaar ein Attribut dargestellt wird, das intern ganz anders organisiert sein kann.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Circle extends GeomShape2D {
    ..........
    public int getCenterX () {
        return positionX + radius;
    }
    public void setCenterX ( int centerX ) {
        positionX = centerX − radius;
    }
    ..........
}
```

Zudem ist dies ein Beispiel dafür, wofür protected sinnvoll sein kann. Wenn man bei der Definition einer Basisklasse antizipiert, dass ein Attribut der Basisklasse im private-Bereich der abgeleiteten Klassen nützlich sein könnte, dann bietet sich protected an.

Grundsätzlich sollte jedes Attribut in der Regel private oder protected definiert und der public-Zugriff durch ein get-/set-Methodenpaar moderiert werden (bei konstanten Attributen natürlich nur get, nicht set). Auch wenn ein Attribut erst einmal ganz „normal“ organisiert ist, kann es später passieren, dass man seine interne Organisation anders gestalten möchte. Hat man das Attribut von vornherein durch ein get-/set-Methodenpaar eingekapselt, dann müssen nur die Interna der Klasse modifiziert werden; alles, was auf dieser Klasse aufbaut und im Extremfall vielleicht Millionen von Zeilen umfassen kann, bleibt von dieser Modifikation unberührt.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Circle extends GeomShape2D {
    ..........
    public Circle ( int centerX, int centerY, int radius ) {
        super ( ..... );
        this.radius = radius,
        setCenterX ( centerX );
        setCenterY ( centerY );
    }
}
```

Der Konstruktor benötigt natürlich die Koordinaten und den Radius des neuen Kreises als Parameter.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Circle extends GeomShape2D {
    ..........
    public Circle ( int centerX, int centerY, int radius ) {
       super ( ..... );
       this.radius = radius,
       setCenterX ( centerX );
       setCenterY ( centerY );
    }
}
```

Er muss den Konstruktor der Basisklasse aufrufen, und das muss auch die allererste Anweisung sein. Da die Farben und andere Attribute der Basisklasse *nicht* Parameter dieses Konstruktors der Klasse Circle sind, müssen hier vorab festgelegte Standardwerte für diese Attribute eingesetzt werden. Typische Standardwerte wären etwa Schwarz für die Randfarbe und Weiß für die Füllfarbe.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Circle extends GeomShape2D {
    .....
    public Circle ( int centerX, int centerY, int radius ) {
        super ( ..... );
        this.radius = radius,
        setCenterX ( centerX );
        setCenterY ( centerY );
    }
}
```

Für die Koordinaten des Kreismittelpunktes nutzen wir einfach die beiden eben definierten Methoden. Die Logik, dass der Kreismittelpunkt sich aus dem Referenzpunkt berechnet, ist nur in den vier Methoden getCenterX/Y und setCenterX/Y zu finden. Das verringert die Fehleranfälligkeit, insbesondere wenn man an dieser Logik später etwas ändern will, denn man muss ja nur diese vier Methoden ändern und kann nicht aus Versehen irgendeine andere Stelle, an der diese Logik ebenfalls verwendet wird, übersehen, denn eine solche andere Stelle gibt es dann ja nicht.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Rectangle
                    extends GeomShape2D {
   protected int width;
   protected int height;
   ..........
}
```

Um ein Gefühl für die Vielfalt der von GeomShape2D abgeleiteten Klassen zu bekommen, schauen wir kurz in eine zweite Klasse hinein. Zusammen mit dem Referenzpunkt und dem Rotationswinkel aus GeomShape2D definieren die Breite und die Höhe eindeutig die Form und Platzierung eines Rechtecks. Eine Klasse zur Realisierung von Rechtecken benötigt daher nur diese beiden Attribute zusätzlich zu denen von Klasse GeomShape2D.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class StraightLine
                    extends GeomShape2D {
   protected int x1;
   protected int x2;
   protected int y1;
   protected int y2;

   ..........
}
```

Ein drittes und letztes Beispiel: Eine gerade Strecke wird durch die Koordinaten ihrer beiden Endpunkte eindeutig definiert.

Da der Referenzpunkt schon in GeomShape2D gespeichert wird, bräuchte man eigentlich nicht alle vier Koordinaten zu Attributen machen. Es würde reichen, zwei dieser vier Zahlen zu speichern und zusätzlich die boolesche Information, ob der Referenzpunkt einer der beiden Endpunkte ist oder nicht. Zum Zwecke der Illustration entscheiden wir uns anders als bei Circle, wo wir den redundanten Mittelpunkt „eingespart“ hatten.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
GeomShape2D [ ] picture
      = new GeomShape2D [3];

picture[0] = new Circle ( ..... );
picture[1] = new Rectangle ( ..... );
picture[2] = new StraightLine ( ..... );


for ( int i = 0; i < picture.length; i++ ) {
   picture[i].paint ( g );
}
```

Mit diesem Konzept – eine Basisklasse GeomShape2D nebst abgeleiteten Klassen für unterschiedliche konkrete geometrische Formarten – können wir dann zum Beispiel Bilder so malen, wie es auch Zeichenprogramme vorsehen, in denen Bilder aus einzelnen geometrischen Formen zusammengesetzt sind.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
GeomShape2D [ ] picture
      = new GeomShape2D [3];

picture[0] = new Circle ( ..... );
picture[1] = new Rectangle ( ..... );
picture[2] = new StraightLine ( ..... );


for ( int i = 0; i < picture.length; i++ ) {
   picture[i].paint ( g );
}
```

So wie hier wird das Bild aufgebaut – einfach ein Array von einzelnen, beliebig gemischten geometrischen Objekten.

Fallbeispiel Graphik

TECHNISCHE UNIVERSITÄT DARMSTADT

```
GeomShape2D [ ] picture
      = new GeomShape2D [3];

picture[0] = new Circle ( ..... );
picture[1] = new Rectangle ( ..... );
picture[2] = new StraightLine ( ..... );


for ( int i = 0; i < picture.length; i++ ) {
   picture[i].paint ( g );
}
```

Und so wie hier gezeigt wird das Bild auf eine Zeichenfläche gezeichnet. Der Komponententyp des Arrays ist von der Basisklasse. Hier können keine Methoden aufgerufen werden, die in der Basisklasse nicht schon *definiert* sind. Aber sie müssen in der Basisklasse noch nicht *implementiert* sein, denn die tatsächlich aufgerufene Implementation der Zeichenmethode ist die der jeweiligen abgeleiteten Klasse. Das ist auch sinnvoll; natürlich sollte das Zeichnen bei jedem geometrischen Objekt spezifisch für diese Objektart sein, und das ist eben die Implementation in der jeweiligen abgeleiteten Klasse.

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
import java.awt.Graphics;

public abstract class GeomShape2D {

    ..........

    public abstract void paint ( Graphics g );

    ..........

}
```

Wie haben in der Zwischenzeit gesehen, was es bedeutet, dass eine Klasse wie etwa GeomShape2D abstrakt ist: Mindestens eine Methode ist abstrakt. Diese abstrakte Methode namens paint hatten wir bisher ausgelassen.

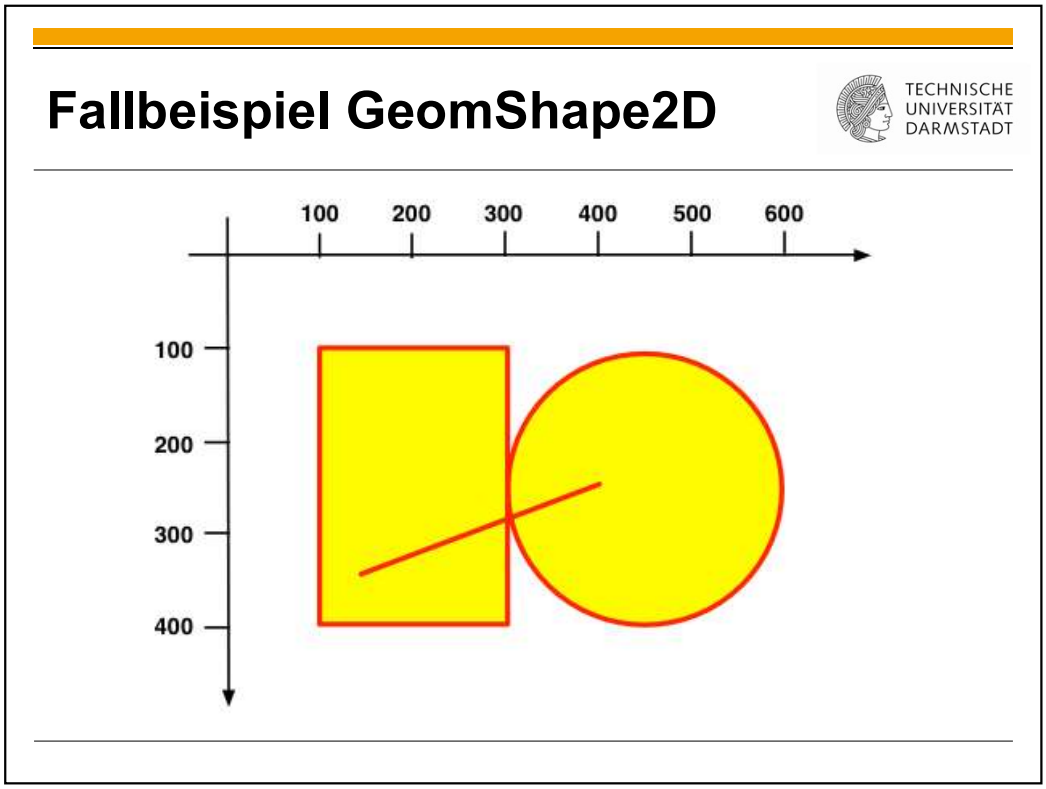
Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
import java.awt.Graphics;

public abstract class GeomShape2D {
    ...
    public abstract void paint ( Graphics g );
    ...
}
```

Unsere Methode paint hat einen einzigen Parameter, und der ist von Klasse java.awt.Graphics. Ein Objekt der Klasse Graphics repräsentiert eine Zeichenfläche und moderiert das Zeichnen auf dieser Zeichenfläche. Wir werden das Zeichnen gleich an den Implementationen von paint in den von GeomShape2D abgeleiteten Klassen sehen.



***Erinnerung*: Die drei gelbroten Formen wollen wir zeichnen.**

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyApplet extends Applet {
   public void paint ( Graphics graphics ) {
      GeomShape2D picture = new GeomShape2D [ 3 ]:
      picture[0] = new Circle ( 450, 250, 100 );
      picture[1] = new Rectangle ( 100, 100, 200 300 );
      picture[2] = new StraightLine ( 150, 350, 400, 250 );
      for ( int i = 0; i < picture.length; i++ ) {
          picture[i].setBoundaryColor ( Color.RED );
          picture[i].setFillColor ( Color.YELLOW );
          picture[i].paint ( graphics );
      }
   }
}
```

***Erinnerung*: Das war der Code für unsere Applet-Klasse, die genau dieses Bild zeichnet.**

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyApplet extends Applet {
   public void paint ( Graphics graphics ) {
      GeomShape2D picture = new GeomShape2D [ 3 ]:
      picture[0] = new Circle ( 450, 250, 100 );
      picture[1] = new Rectangle ( 100, 100, 200 300 );
      picture[2] = new StraightLine ( 150, 350, 400, 250 );
      for ( int i = 0; i < picture.length; i++ ) {
          picture[i].setBoundaryColor ( Color.RED );
          picture[i].setFillColor ( Color.YELLOW );
          picture[i].paint ( graphics );
      }
   }
}
```

Es fehlt nur noch die Methode paint für die Klassen Circle, Rectangle und StraightLine. Die Definitionen schauen wir uns nun zum Schluss an.

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Circle extends GeomShape2D {
    .........
    public void paint ( Graphics g ) {
        g.setColor ( getFillColor() );
        g.fillOval ( getPositionX(), getPositionY(),
                     2*getRadius(), 2*getRadius() );
        g.setColor ( getBoundaryColor() );
        g.drawOval ( getPositionX(), getPositionY(),
                       2*getRadius(), 2*getRadius() );
    }
    .........
}
```

Kommen wir als erstes zur Implementation von paint für die Klasse Circle.

Die Klasse Graphics bietet alles, was man braucht, um einen Kreis zu zeichnen. Wie das dann technisch auf die Zeichenfläche kommt, die hinter dem Parameter g steht, braucht uns nicht zu interessieren, und das können wir auch nicht sehen.

Der Einfachheit halber ignorieren wir hier und im Folgenden den Rotationswinkel und den Transparenzwert.

***Vorgriff*: Im Kapitel 10 sehen wir die Klasse Graphics in sinnvollen Kontexten in den Abschnitten „Canvas“ und „Applets“.**

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Circle extends GeomShape2D {
    .........
    public void paint ( Graphics g ) {
        g.setColor ( getFillColor() );
        g.fillOval ( getPositionX(), getPositionY(),
                     2*getRadius(), 2*getRadius() );
        g.setColor ( getBoundaryColor() );
        g.drawOval ( getPositionX(), getPositionY(),
                       2*getRadius(), 2*getRadius() );
    }
    .........
}
```

Wir füllen zuerst das Innere mit der Füllfarbe, erst danach zeichnen wir den Rand mit der Randfarbe, damit die Füllfarbe nicht den Rand teilweise oder ganz überdeckt. Mit Methode setColor von Klasse Graphics wird die Farbe intern in g für die nächsten Zeichenoperationen gesetzt, und zwar bis zum nächsten Aufruf einer farbändernden Methode. In unserem Fallbeispiel wird g bis zum nächsten setColor die Farbe Gelb verwenden, denn Gelb hatten wir als Füllfarbe bei jedem der drei geometrischen Objekte gesetzt, und Gelb ist daher die Rückgabe der Methode getFillColor von GeomShape2D, die ja in jeder abgeleiteten Klasse ererbt wird.

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Circle extends GeomShape2D {
    .........
    public void paint ( Graphics g ) {
        g.setColor ( getFillColor() );
        g.fillOval ( getPositionX(), getPositionY(),
                     2*getRadius(), 2*getRadius() );
        g.setColor ( getBoundaryColor() );
        g.drawOval ( getPositionX(), getPositionY(),
                     2*getRadius(), 2*getRadius() );
    }
    .........
}
```

Die Klasse Graphics hat zwar keine eigene Methode, um Kreise zu zeichnen, aber eine für Ellipsen, und jeder Kreis ist ja auch eine Ellipse. Das „fill“ im Namen fillOval besagt, dass die *Fläche* der Ellipse mit der momentanen Farbe, also mit Gelb voll ausgefüllt wird.

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Circle extends GeomShape2D {
    .........
    public void paint ( Graphics g ) {
        g.setColor ( getFillColor() );
        g.fillOval ( getPositionX(), getPositionY(),
                     2*getRadius(), 2*getRadius() );
        g.setColor ( getBoundaryColor() );
        g.drawOval ( getPositionX(), getPositionY(),
                       2*getRadius(), 2*getRadius() );
    }
    .........
}
```

Die Methode fillOval bekommt als erstes die x- und die y-Koordinate des Punktes, den auch wir in GeomShape2D als Referenzpunkt verwenden, die linke obere Ecke des umschließenden achsenparallelen Rechtecks.

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Circle extends GeomShape2D {
    .........
    public void paint ( Graphics g ) {
        g.setColor ( getFillColor() );
        g.fillOval ( getPositionX(), getPositionY(),
                     2*getRadius(), 2*getRadius() );
        g.setColor ( getBoundaryColor() );
        g.drawOval ( getPositionX(), getPositionY(),
                     2*getRadius(), 2*getRadius() );
    }
    .........
}
```

Eine Ellipse hat zwei Radien, den großen und den kleinen. Bei einem Kreis sind beide Radien gleich. Die Methode fillOval bekommt aber nicht die Radien, sondern die Durchmesser, also beide Male Multiplikation mit 2.

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Circle extends GeomShape2D {
    .........
    public void paint ( Graphics g ) {
        g.setColor ( getFillColor() );
        g.fillOval ( getPositionX(), getPositionY(),
                     2*getRadius(), 2*getRadius() );
        g.setColor ( getBoundaryColor() );
        g.drawOval ( getPositionX(), getPositionY(),
                     2*getRadius(), 2*getRadius() );
    }
    .........
}
```

Als nächstes zeichnen wir den Rand des Objektes, und dafür setzen wir die interne Farbe von g auf die Randfarbe, in unserem Fallbeispiel also auf Rot, denn in allen drei geometrischen Objekten hatten wir die Randfarbe auf Rot gesetzt.

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Circle extends GeomShape2D {
    .........
    public void paint ( Graphics g ) {
        g.setColor ( getFillColor() );
        g.fillOval ( getPositionX(), getPositionY(),
                     2*getRadius(), 2*getRadius() );
        g.setColor ( getBoundaryColor() );
        g.drawOval ( getPositionX(), getPositionY(),
                     2*getRadius(), 2*getRadius() );
    }
    .........
}
```

Der Methodenaufruf ist praktisch derselbe, nur dass die Methode nicht fillOval, sondern drawOval heißt, weil sie nicht das Innere des Objektes füllt, sondern nur seinen Rand zeichnet.

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Rectangle extends GeomShape2D {
    .....
    public void paint ( Graphics g ) {
        g.setColor ( getFillColor() );
        g.fillRect ( getPositionX(), getPositionY(),
                     width, height );
        g.setColor ( getBoundaryColor() );
        g.drawRect ( getPositionX(), getPositionY(),
                       width, height );
    }
    .....
}
```

Die paint-Methode von Rectangle ist sehr ähnlich zu der von Circle. Die dazu passenden Methoden von Klasse Graphics heißen fillRect beziehungsweise drawRect. Anstelle der beiden Durchmesser erwarten die beiden Methoden die Breite und die Höhe als dritten und vierten Parameter.

Fallbeispiel GeomShape2D

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class StraightLine extends GeomShape2D {
    .........
    public void paint ( Graphics g ) {
        g.setColor ( getBoundaryColor() );
        g.drawLine ( x1, y1, x2, y2 );
    }

    .........
}
```

Die paint-Methode von StraightLine ist extrem einfach: Randfarbe setzen und die passende Methode von Klasse Graphics mit den Koordinaten der Eckpunkte in der passenden Reihenfolge aufrufen, fertig.