Einführung in die objektorientierte Programmierung mit geometrischen Figuren als Komponenten

Workshop am 10. Informatiktag NRW
im Rahmen der *INFOS 2011*
an der Westfälischen Wilhelms-Universität Münster

Christian Wolf

Luisenschule,
Gymnasium der Stadt Mülheim a.d. Ruhr

Dienstag, 13. September 2011

Abstract

- Modellieren und Implementieren ist verbindliche Aufgabe und Ziel des Informatikunterrichts
- Bei Wahl des objektorientierten Paradigmas (Sek II): Konzept *Stifte und Mäuse (SuM)* gern und aus guten Gründen eingesetzt
- Aber: SuM ist problembehaftet!
- Daher: Vorschlag und Erprobung des Konzeptes *Geometric Figures as Swing Components (GeoFaSC)*

Überblick

Vorgaben und Empfehlungen „*en détail*“

Bildungsstandards Informatik für die Sekundarstufe I (2008)

Prozessbereich *Modellieren und Implementieren* [GI 2008, S. 9]:

- „*Bei den Prozessen geht es um die Art, wie mit den Inhalten umgegangen wird. Dabei geht es um Arbeitsweisen, die in der Informatik besonders ausgeprägt sind – etwa das Implementieren eines Modells – [...]. [...]*

  Die Schülerinnen und Schüler aller Jahrgangsstufen

  - *erstellen informatische Modelle zu gegebenen Sachverhalten,*
  - *implementieren Modelle mit geeigneten Werkzeugen,*
  - *reflektieren Modelle und deren Implementierung.*“

Vorgaben und Empfehlungen *„en détail“*

Richtlinien und Lehrpläne für die Sekundarstufe II (1999)

In der *(2) Sequenz „objektorientiert allgemein“* [MfSuW 1999, S. 47] heißt es:

- *„Typisch für Anwendungsprogrammierung nach dem objektorientierten Ansatz ist es, in einer gegebenen Problemsituation Objekte und ihre Beziehungen zu identifizieren (Objektorientierte Analyse), sie in geeigneten Klassen abzubilden (Objektorientiertes Design), und diese schließlich in der gewählten Programmierumgebung zu implementieren (Objektorientierte Programmierung).“*

Vorgaben und Empfehlungen „*en détail*“

Einheitliche Prüfungsanforderungen Informatik (2004)

In [Kmk 2004, S. 4 f.] heißt es:

- „*Objektorientierte Modellierung insbesondere: Objekt, Klasse, Beziehungen zwischen Klassen, Interaktion von Objekten, Klassendiagramm (z. B. mit UML)*“
- „*Die Prüflinge [...], können [...] Klassenbildung einsetzen, sind insbesondere mit dem Modellbildungszyklus vertraut; dazu gehören in problemadäquater Auswahl und Reihenfolge: [...] Umsetzen*[1] *unter Berücksichtigung der zur Verfügung stehenden Werkzeuge und Hilfsmittel, Testen der Lösung, [...].*“

[1] Umsetzen, d.h. Implementieren

Vorgaben und Empfehlungen „*en détail*“

Es lässt sich also konstatieren:

- Modellieren und Implementieren wird als Aufgabe und Ziel des gesamten schulischen Informatikunterrichts empfohlen
- objektorientiertes Modellieren (OOM) und Programmieren (OOP) ist bei Wahl des OO-Paradigmas verbindlich

Das SuM-Konzept

- „Paradekonzept“ mit Software-Bibliothek als Werkzeug zur Einführung in die OOP[2]
- In [MfSuW 1999, S. 47] als *„Ausgangspunkt erster Anwendungsentwicklungen“* empfohlen
- SuM-Bibliothek:
  - Rechnerkomponenten modellierende Klassen
    - `Maus`
    - `Tastatur`
    - `Bildschirm`
  - sowie für universale Zeichenoperationen die Klasse
    - `Stift`

[2] Auch für weiterführende Inhalte und Konzepte eingesetzt, z.B. dynamische Datenstrukturen [Schriek 2006, Band II]

Das SuM-Konzept

Pro-Argumente

+ Aus fachdidaktischer Sicht:
  - Werkzeug für alle einführenden relevanten Inhalte und Konzepte bezüglich der OOP
+ Aus didaktisch-methodischer Sicht:
  - Intuitiver, universaler Zeichenansatz (weil Stift-basiert)
  - Dadurch: Grafikorientierung bei Programmlösungen
  - Dadurch: Anschauung und ,,visuelle Überprüfung“ der Programmlösungen
+ Praktikables Lehr-Lernmaterial existiert:
  - Implementationen für Java und Python
  - Erweiterungen verfügbar (z.B. einfacher GUI-Builder)
  - Schrieks Schulbände [Schriek 2006]

Das SuM-Konzept

Contra-Argumente

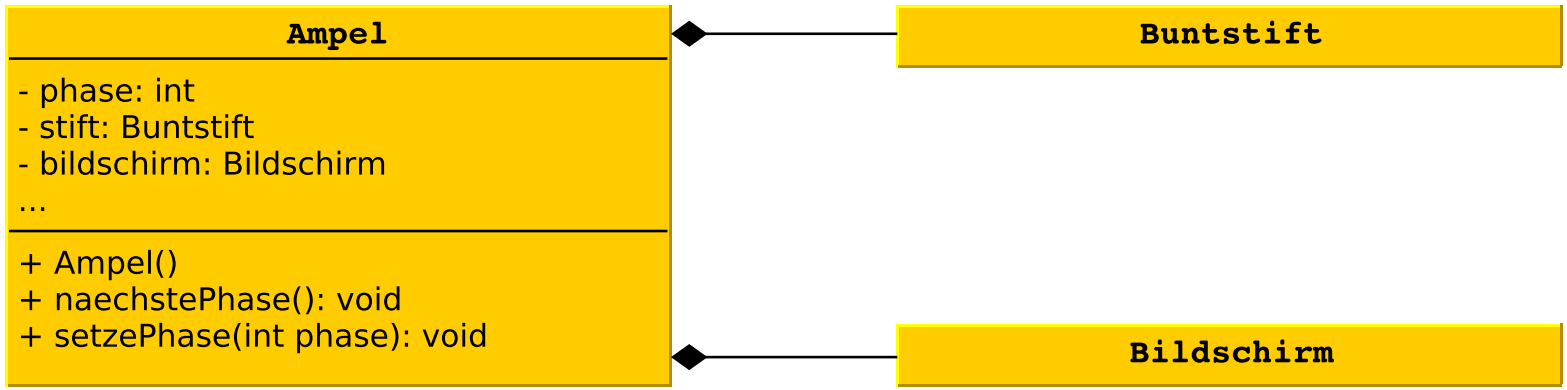
- Problematik 1:
  - Verleitung zu trivialen, inadäquaten OOM
  - Häufig wird nur genau ein Objekt verwendet, z.B. `(Bunt)Stift`
- Problematik 2:
  - Fokussierung von Zeichenprimitiven
  - Teils aufgeblähter Quellcode

$\longrightarrow$ exemplarische Beleuchtung anhand der Programmierung einer grafischen (Verkehrs)Ampel

Das SuM-Konzept

Contra-Argumente - Problematik 1 (Verleitung zu trivialen, inadäquaten OOM)

- Modell 1 (verleitet durch Stift-Ansatz [Spollwig 2004 u. 2006]):



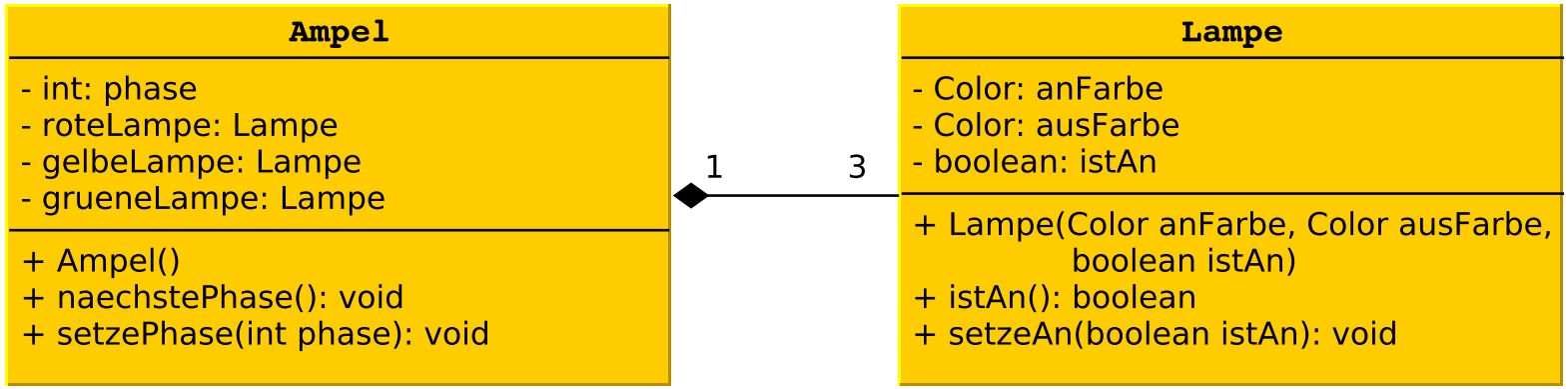
Fragwürdig, weil:

- Erwartete Klasse `Lampe` fehlt
- Warum *hat* eine `Ampel` einen `Bildschirm` und einen `Stift`?

Das SuM-Konzept

Contra-Argumente - Problematik 1 (Verleitung zu inadäquater OOM)

- Modell 2:



Sinnvoller, aber:

- Mit SuM nicht einfach zu realisieren
- Beide Klassen `Ampel` und `Lampe` benötigen zur Darstellung wieder die Klasse `Stift`

Das SuM-Konzept

Contra-Argumente - Problematik 2 (Fokussierung von Zeichenprimitiven)

Beispiel: Programmierung des Phasenwechsels von Grün auf Gelb

- nach Modell 1:

```
buntstift.hoch();
buntstift.bewegeBis(xKreisGruen, yKreisGruen);
buntstift.runter();
buntstift.setzeFuellmuster(Muster.GEFUELLT);
buntstift.setzeFarbe(Farbe.GRAU);
buntstift.zeichneKreis(radiusKreisGruen)
buntstift.hoch();
buntstift.bewegeBis(xKreisGelb, yKreisGelb);
buntstift.runter();
buntstift.setzeFarbe(Farbe.GELB);
buntstift.zeichneKreis(radiusKreisGelb);
```

Das SuM-Konzept

Contra-Argumente - Problematik 2 (Fokussierung von Zeichenprimitiven)

Beispiel: Programmierung des Phasenwechsels von Grün auf Gelb

- nach Modell 1:

```
buntstift.hoch();
buntstift.bewegeBis(xKreisGruen, yKreisGruen);
buntstift.runter();
...
```

- nach Modell 2:

```
grueneLampe.setzeAn(true);
gelbeLampe.setzeAn(false);
```

Das GeoFaSC-Konzept

Entwicklung

- Beobachtung 1: Grafische Lösungen erfordern häufig geometrische Figuren und deren Komposition, Beispiele aus [Schriek, 2006]:
  - Pfeil und Dartscheibe → Linie und Kreise
  - Abprallende Kugeln → Kreise
  - Wolf und Rotkäppchen → Polylinie und Kreis
  - Lokomotive und Waggons → Rechtecke, Kreise und Linien
- Beobachtung 2: Swing als Java-basierte, objektorientierte und komponentenbasierte Grafikbibliothek [Loy u.a. 2002]
- Beobachtung 3: Spollwigs Vorschlag einer Grafikbibliothek *uGrafik* für Delphi [Spollwig 2006 und 2008]

Das GeoFaSC-Konzept

Entwicklung

- Beobachtung 1: Grafische Lösungen erfordern häufig geometrische Figuren und deren Komposition, Beispiele aus [Schriek, 2006]:
  - Pfeil und Dartscheibe → Linie und Kreise
  - Abprallende Kugeln → Kreise
  - Wolf und Rotkäppchen → Polylinie und Kreis
  - Lokomotive und Waggons → Rechtecke, Kreise und Linien
- Beobachtung 2: Swing als Java-basierte, objektorientierte und komponentenbasierte Grafikbibliothek [Loy u.a. 2002]
- Beobachtung 3: Spollwigs Vorschlag einer Grafikbibliothek *uGrafik* für Delphi [Spollwig 2006 und 2008]

Das GeoFaSC-Konzept

Entwicklung

- Beobachtung 1: Grafische Lösungen erfordern häufig geometrische Figuren und deren Komposition, Beispiele aus [Schriek, 2006]:
  - Pfeil und Dartscheibe → Linie und Kreise
  - Abprallende Kugeln → Kreise
  - Wolf und Rotkäppchen → Polylinie und Kreis
  - Lokomotive und Waggons → Rechtecke, Kreise und Linien
- Beobachtung 2: Swing als Java-basierte, objektorientierte und komponentenbasierte Grafikbibliothek [Loy u.a. 2002]
- Beobachtung 3: Spollwigs Vorschlag einer Grafikbibliothek *uGrafik* für Delphi [Spollwig 2006 und 2008]

Das GeoFaSC-Konzept

Entwicklung

- Beobachtung 1: Grafische Lösungen erfordern häufig geometrische Figuren und deren Komposition, Beispiele aus [Schriek, 2006]:
  - Pfeil und Dartscheibe → Linie und Kreise
  - Abprallende Kugeln → Kreise
  - Wolf und Rotkäppchen → Polylinie und Kreis
  - Lokomotive und Waggons → Rechtecke, Kreise und Linien
- Beobachtung 2: Swing als Java-basierte, objektorientierte und komponentenbasierte Grafikbibliothek [Loy u.a. 2002]
- Beobachtung 3: Spollwigs Vorschlag einer Grafikbibliothek *uGrafik* für Delphi [Spollwig 2006 und 2008]

Das GeoFaSC-Konzept

Entwicklung

- Beobachtung 1: Grafische Lösungen erfordern häufig geometrische Figuren und deren Komposition, Beispiele aus [Schriek, 2006]:
  - Pfeil und Dartscheibe → Linie und Kreise
  - Abprallende Kugeln → Kreise
  - Wolf und Rotkäppchen → Polylinie und Kreis
  - Lokomotive und Waggons → Rechtecke, Kreise und Linien
- Beobachtung 2: Swing als Java-basierte, objektorientierte und komponentenbasierte Grafikbibliothek [Loy u.a. 2002]
- Beobachtung 3: Spollwigs Vorschlag einer Grafikbibliothek *uGrafik* für Delphi [Spollwig 2006 und 2008]

Das GeoFaSC-Konzept

Entwicklung

- Beobachtung 1: Grafische Lösungen erfordern häufig geometrische Figuren und deren Komposition, Beispiele aus [Schriek, 2006]:
  - Pfeil und Dartscheibe → Linie und Kreise
  - Abprallende Kugeln → Kreise
  - Wolf und Rotkäppchen → Polylinie und Kreis
  - Lokomotive und Waggons → Rechtecke, Kreise und Linien
- Beobachtung 2: Swing als Java-basierte, objektorientierte und komponentenbasierte Grafikbibliothek [Loy u.a. 2002]
- Beobachtung 3: Spollwigs Vorschlag einer Grafikbibliothek *uGrafik* für Delphi [Spollwig 2006 und 2008]

Das GeoFaSC-Konzept

Entwicklung

- Beobachtung 1: Grafische Lösungen erfordern häufig geometrische Figuren und deren Komposition, Beispiele aus [Schriek, 2006]:
  - Pfeil und Dartscheibe → Linie und Kreise
  - Abprallende Kugeln → Kreise
  - Wolf und Rotkäppchen → Polylinie und Kreis
  - Lokomotive und Waggons → Rechtecke, Kreise und Linien
- Beobachtung 2: Swing als Java-basierte, objektorientierte und komponentenbasierte Grafikbibliothek [Loy u.a. 2002]
- Beobachtung 3: Spollwigs Vorschlag einer Grafikbibliothek *uGrafik* für Delphi [Spollwig 2006 und 2008]

Das GeoFaSC-Konzept

Entwicklung

- Folgerung: Jede Figur sollte nicht Artefakt eines (Zeichen)Werkzeugs, sondern selbst ein Werkzeug sein!
- Anforderungen:
  1. Eine Figur ist ein Objekt.
  2. Eine Figur ist ohne Zeichenaufwand erzeugbar und verwaltbar.
  3. Eine Figur besitzt Attribute, die sie eindeutig in einem Bezugssystem beschreiben.
  4. Eine Figur besitzt Abfrage- und Änderungsdienste, mit denen man ihren Zustand lesen und schreiben kann.
  5. Eine Figur ist eine (Swing) Komponente, d.h. sie kann andere Figuren und Komponenten inkludieren sowie auch selbst inkludiert werden.

$\longrightarrow$ das GeoFaSC-Konzept

Das GeoFaSC-Konzept

Entwicklung

- Folgerung: Jede Figur sollte nicht Artefakt eines (Zeichen)Werkzeugs, sondern selbst ein Werkzeug sein!
- Anforderungen:
  1. Eine Figur ist ein Objekt.
  2. Eine Figur ist ohne Zeichenaufwand erzeugbar und verwaltbar.
  3. Eine Figur besitzt Attribute, die sie eindeutig in einem Bezugssystem beschreiben.
  4. Eine Figur besitzt Abfrage- und Änderungsdienste, mit denen man ihren Zustand lesen und schreiben kann.
  5. Eine Figur ist eine (Swing) Komponente, d.h. sie kann andere Figuren und Komponenten inkludieren sowie auch selbst inkludiert werden.

$\longrightarrow$ das GeoFaSC-Konzept

Das GeoFaSC-Konzept

Entwicklung

- Folgerung: Jede Figur sollte nicht Artefakt eines (Zeichen)Werkzeugs, sondern selbst ein Werkzeug sein!
- Anforderungen:
  1. Eine Figur ist ein Objekt.
  2. Eine Figur ist ohne Zeichenaufwand erzeugbar und verwaltbar.
  3. Eine Figur besitzt Attribute, die sie eindeutig in einem Bezugssystem beschreiben.
  4. Eine Figur besitzt Abfrage- und Änderungsdienste, mit denen man ihren Zustand lesen und schreiben kann.
  5. Eine Figur ist eine (Swing) Komponente, d.h. sie kann andere Figuren und Komponenten inkludieren sowie auch selbst inkludiert werden.

$\longrightarrow$ das GeoFaSC-Konzept

Das GeoFaSC-Konzept

Entwicklung

- Folgerung: Jede Figur sollte nicht Artefakt eines (Zeichen)Werkzeugs, sondern selbst ein Werkzeug sein!
- Anforderungen:
  1. Eine Figur ist ein Objekt.
  2. Eine Figur ist ohne Zeichenaufwand erzeugbar und verwaltbar.
  3. Eine Figur besitzt Attribute, die sie eindeutig in einem Bezugssystem beschreiben.
  4. Eine Figur besitzt Abfrage- und Änderungsdienste, mit denen man ihren Zustand lesen und schreiben kann.
  5. Eine Figur ist eine (Swing) Komponente, d.h. sie kann andere Figuren und Komponenten inkludieren sowie auch selbst inkludiert werden.

$\longrightarrow$ das GeoFaSC-Konzept

Das GeoFaSC-Konzept

Entwicklung

- Folgerung: Jede Figur sollte nicht Artefakt eines (Zeichen)Werkzeugs, sondern selbst ein Werkzeug sein!
- Anforderungen:
  1. Eine Figur ist ein Objekt.
  2. Eine Figur ist ohne Zeichenaufwand erzeugbar und verwaltbar.
  3. Eine Figur besitzt Attribute, die sie eindeutig in einem Bezugssystem beschreiben.
  4. Eine Figur besitzt Abfrage- und Änderungsdienste, mit denen man ihren Zustand lesen und schreiben kann.
  5. Eine Figur ist eine (Swing) Komponente, d.h. sie kann andere Figuren und Komponenten inkludieren sowie auch selbst inkludiert werden.

$\longrightarrow$ das GeoFaSC-Konzept

Das GeoFaSC-Konzept

Entwicklung

- Folgerung: Jede Figur sollte nicht Artefakt eines (Zeichen)Werkzeugs, sondern selbst ein Werkzeug sein!
- Anforderungen:
  1. Eine Figur ist ein Objekt.
  2. Eine Figur ist ohne Zeichenaufwand erzeugbar und verwaltbar.
  3. Eine Figur besitzt Attribute, die sie eindeutig in einem Bezugssystem beschreiben.
  4. Eine Figur besitzt Abfrage- und Änderungsdienste, mit denen man ihren Zustand lesen und schreiben kann.
  5. Eine Figur ist eine (Swing) Komponente, d.h. sie kann andere Figuren und Komponenten inkludieren sowie auch selbst inkludiert werden.

$\longrightarrow$ das GeoFaSC-Konzept

Das GeoFaSC-Konzept

Entwicklung

- Folgerung: Jede Figur sollte nicht Artefakt eines (Zeichen)Werkzeugs, sondern selbst ein Werkzeug sein!
- Anforderungen:
  1. Eine Figur ist ein Objekt.
  2. Eine Figur ist ohne Zeichenaufwand erzeugbar und verwaltbar.
  3. Eine Figur besitzt Attribute, die sie eindeutig in einem Bezugssystem beschreiben.
  4. Eine Figur besitzt Abfrage- und Änderungsdienste, mit denen man ihren Zustand lesen und schreiben kann.
  5. Eine Figur ist eine (Swing) Komponente, d.h. sie kann andere Figuren und Komponenten inkludieren sowie auch selbst inkludiert werden.

$\longrightarrow$ das GeoFaSC-Konzept

Die GeoFaSC-Bibliothek

Programmierwerkzeug als Umsetzung des Konzeptes

- Implementierung relevanter Figuren gemäß Anforderungen:
  - `Circle`,
  - `Ellipse`,
  - `LineSegment`,
  - `Point`,
  - `Polyline` (Polygon wenn geschlossen),
  - `Rectangle` und
  - `Square`.
- Als Open-Source-Software unter der GPLv3 mit umfangreicher Dokumentation und Sample-Code unter `http://geofasc.de`

Die GeoFaSC-Bibliothek

Wichtige Features

- Figuren sind basierend auf Swing [Loy u.a. 2002] wie Elemente grafischer Benutzeroberflächen implementiert
- Unterstützung des in Swing üblichen *Model-UI-Component*-Entwurfsmuster[3] [Fowler]
  - (Neu)Zeichnen ist gekapselt und funktioniert automatisch
  - Model und View können vor und zur Laufzeit ausgetauscht werden
- Unterstützung des *Composite*-Entwurfsmusters
  - Erzeugung zusammengesetzter Figuren/ Komponenten in beliebiger Tiefe durch Schachtelung

[3] Eine Abwandlung des *Model-View-Controller*-Entwurfsmusters

Die GeoFaSC-Bibliothek

Wichtige Features

- Figuren sind basierend auf Swing [Loy u.a. 2002] wie Elemente grafischer Benutzeroberflächen implementiert
- Unterstützung des in Swing üblichen *Model-UI-Component*-Entwurfsmuster[3] [Fowler]
  - (Neu)Zeichnen ist gekapselt und funktioniert automatisch
  - Model und View können vor und zur Laufzeit ausgetauscht werden
- Unterstützung des *Composite*-Entwurfsmusters
  - Erzeugung zusammengesetzter Figuren/ Komponenten in beliebiger Tiefe durch Schachtelung

[3] Eine Abwandlung des *Model-View-Controller*-Entwurfsmusters

Die GeoFaSC-Bibliothek

Wichtige Features

- Figuren sind basierend auf Swing [Loy u.a. 2002] wie Elemente grafischer Benutzeroberflächen implementiert
- Unterstützung des in Swing üblichen *Model-UI-Component*-Entwurfsmuster[3] [Fowler]
  - (Neu)Zeichnen ist gekapselt und funktioniert automatisch
  - Model und View können vor und zur Laufzeit ausgetauscht werden
- Unterstützung des *Composite*-Entwurfsmusters
  - Erzeugung zusammengesetzter Figuren/ Komponenten in beliebiger Tiefe durch Schachtelung

[3] Eine Abwandlung des *Model-View-Controller*-Entwurfsmusters

Die GeoFaSC-Bibliothek

Code-Snippet: Rechteck mit Kreis und Button

```
Rectangle rectangle = new Rectangle(70, 190); // Breite x Höhe in Pixel
rectangle.setRoundedCorners(true);
rectangle.setArcSize(arcWidth, arcHeight);
rectangle.setDirection(45);
...

Circle circle = new Circle();                 // Leerer Kreis
CircleModel model = circle.getModel();        // Zugriff übers Modell
model.setRadius(60);
model.setFigureLocation(20, 20);
...

JButton button = new JButton("Click");
button.setBounds(20, 80, 60, 20) // x, y, Breite, Höhe
...

rectangle.add(circle); // rectangle inkludiert circle
rectangle.add(button); // rectangle inkludiert button
```

Die GeoFaSC-Bibliothek

Code-Snippet: Figuren sichtbar machen

- Beliebige Top-Level Container dürfen verwendet werden:

```
import javax.swing.*;
...
JPanel canvas = new JPanel();
JFrame frame = new JFrame("Some figures...");
frame.setContentPane(canvas);
frame.setSize(640, 480);
frame.setVisible(true);
canvas.add(rectangle); // rectangle mit circle u. button sichtbar machen
```

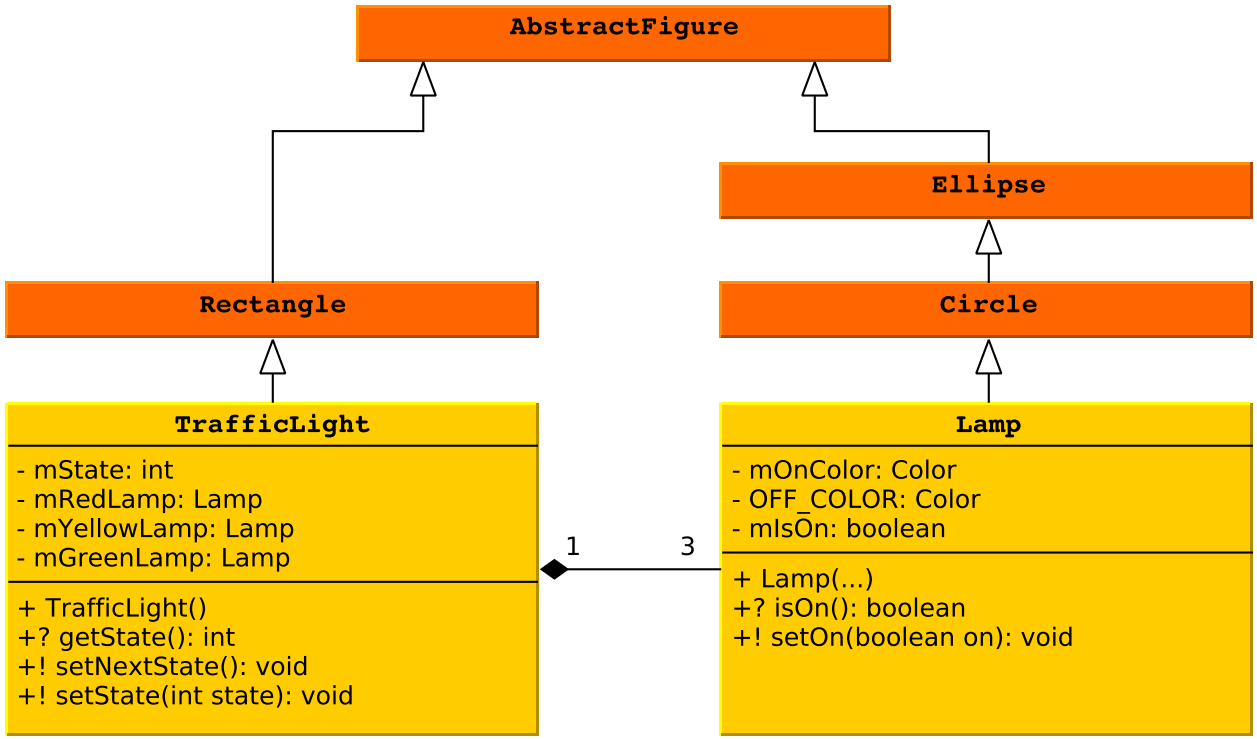
- Alternativ:

```
import geofasc.swing.tool.*;
...
Frame frame = new Frame("Some figures...");
frame.setVisible(true);
frame.getCanvas().add(rectangle);
```

Die GeoFaSC-Bibliothek

Code-Snippet: Lampe und Ampel

- Modell 2 „GeoFaSC-like“:



Die GeoFaSC-Bibliothek

Code-Snippet: Lampe und Ampel

```
import java.awt.Color;
import geofasc.swing.Circle

public class Lamp extends Circle {

  private static Color OFF_COLOR =
                  Color.DARK_GRAY;
  private Color mOnColor;
  private boolean mIsOn;

  public Lamp(int x, int y, int radius,
              Color onColor, boolean isOn) {
    super(x, y, radius);
    mOnColor = onColor;
    setFigureFilled(true);
    setOn(isOn);
  }

  public boolean isOn() {
    return mIsOn;
  }


public void setOn(boolean on) {
    mIsOn = on;
    if (mIsOn)
      setFigureFillColor(mOnColor);
    else
      setFigureFillColor(OFF_COLOR);
  }

} // class Lamp
```

Die GeoFaSC-Bibliothek

Code-Snippet: Lampe und Ampel

```
import java.awt.Color;
import geofasc.swing.Rectangle;

public class TrafficLight extends Rectangle {

  private Lamp mRedLamp, mYellowLamp, mGreenLamp;
  // 1 = RED, 2 = RED_YELLOW, 3 = GREEN, 4 = YELLOW
  private int mState;

  public TrafficLight() {
    super(70, 190);
    setFigureFillColor(Color.BLACK);
    setFigureFilled(true);
    mState = 1;

    mRedLamp = new Lamp(10, 10, 50, Color.RED, true);
    mYellowLamp = new Lamp(10, 70, 50, Color.YELLOW, false);
    mGreenLamp = new Lamp(10, 130, 50, Color.GREEN, false);

    add(mRedLamp);
    add(mYellowLamp);
    add(mGreenLamp);
  }

  public int getState() {
    return mState;
  }

  public void setNextState() {
    setState(mState + 1);
  }

  public void setState(int state) {
    mState = state;
    if (mState > 4)
      mState = 1;

    switch (mState) {
      case 1: // RED
        mRedLamp.setOn(true);
        mYellowLamp.setOn(false);
        mGreenLamp.setOn(false);
        break;
      case 2: // RED_YELLOW
        ...
    }
  }

} // class TrafficLight
```

Kurze Demo...

Das GeoFaSC-Konzept

Wie vermeidet GeoFaSC die Problematiken?

- **Vs.** Problematik 1 (Verleitung zu trivialen, inadäquaten OOM):
  - Fehlendes, universales Zeichenwerkzeug und automatisches (Neu)Zeichnen von Figuren
  - Defokussierung des Zeichnens und ,,Blick für Wesentliches“
  - Ermutigung zu nicht-trivialen, wirklichkeitsnäheren Modellen
- **Vs.** Problematik 2 (Fokussierung von Zeichenprimitiven):
  - Trennung der Zuständigkeiten, insbesondere automatisches (Neu)Zeichnen der Figuren
  - Programmierer muss nicht zuständig sein für das Zeichnen (er darf jedoch bei Bedarf)
  - Einfachere Umsetzung der nicht-trivialen, wirklichkeitsnäheren Modelle

Das GeoFaSC-Konzept

Wie vermeidet GeoFaSC die Problematiken?

- **Vs.** Problematik 1 (Verleitung zu trivialen, inadäquaten OOM):
  - Fehlendes, universales Zeichenwerkzeug und automatisches (Neu)Zeichnen von Figuren
  - Defokussierung des Zeichnens und „Blick für Wesentliches“
  - Ermutigung zu nicht-trivialen, wirklichkeitsnäheren Modellen
- **Vs.** Problematik 2 (Fokussierung von Zeichenprimitiven):
  - Trennung der Zuständigkeiten, insbesondere automatisches (Neu)Zeichnen der Figuren
  - Programmierer muss nicht zuständig sein für das Zeichnen (er darf jedoch bei Bedarf)
  - Einfachere Umsetzung der nicht-trivialen, wirklichkeitsnäheren Modelle

Das GeoFaSC-Konzept

Wie vermeidet GeoFaSC die Problematiken?

- **Vs.** Problematik 1 (Verleitung zu trivialen, inadäquaten OOM):
  - Fehlendes, universales Zeichenwerkzeug und automatisches (Neu)Zeichnen von Figuren
  - Defokussierung des Zeichnens und ,,Blick für Wesentliches“
  - Ermutigung zu nicht-trivialen, wirklichkeitsnäheren Modellen
- **Vs.** Problematik 2 (Fokussierung von Zeichenprimitiven):
  - Trennung der Zuständigkeiten, insbesondere automatisches (Neu)Zeichnen der Figuren
  - Programmierer muss nicht zuständig sein für das Zeichnen (er darf jedoch bei Bedarf)
  - Einfachere Umsetzung der nicht-trivialen, wirklichkeitsnäheren Modelle

Das GeoFaSC-Konzept

Wie vermeidet GeoFaSC die Problematiken?

- **Vs.** Problematik 1 (Verleitung zu trivialen, inadäquaten OOM):
  - Fehlendes, universales Zeichenwerkzeug und automatisches (Neu)Zeichnen von Figuren
  - Defokussierung des Zeichnens und ,,Blick für Wesentliches“
  - Ermutigung zu nicht-trivialen, wirklichkeitsnäheren Modellen
- **Vs.** Problematik 2 (Fokussierung von Zeichenprimitiven):
  - Trennung der Zuständigkeiten, insbesondere automatisches (Neu)Zeichnen der Figuren
  - Programmierer muss nicht zuständig sein für das Zeichnen (er darf jedoch bei Bedarf)
  - Einfachere Umsetzung der nicht-trivialen, wirklichkeitsnäheren Modelle

Das GeoFaSC-Konzept

Wie vermeidet GeoFaSC die Problematiken?

- **Vs.** Problematik 1 (Verleitung zu trivialen, inadäquaten OOM):
  - Fehlendes, universales Zeichenwerkzeug und automatisches (Neu)Zeichnen von Figuren
  - Defokussierung des Zeichnens und ,,Blick für Wesentliches“
  - Ermutigung zu nicht-trivialen, wirklichkeitsnäheren Modellen
- **Vs.** Problematik 2 (Fokussierung von Zeichenprimitiven):
  - Trennung der Zuständigkeiten, insbesondere automatisches (Neu)Zeichnen der Figuren
  - Programmierer muss nicht zuständig sein für das Zeichnen (er darf jedoch bei Bedarf)
  - Einfachere Umsetzung der nicht-trivialen, wirklichkeitsnäheren Modelle

Das GeoFaSC-Konzept

Wie vermeidet GeoFaSC die Problematiken?

- **Vs.** Problematik 1 (Verleitung zu trivialen, inadäquaten OOM):
  - Fehlendes, universales Zeichenwerkzeug und automatisches (Neu)Zeichnen von Figuren
  - Defokussierung des Zeichnens und ,,Blick für Wesentliches“
  - Ermutigung zu nicht-trivialen, wirklichkeitsnäheren Modellen
- **Vs.** Problematik 2 (Fokussierung von Zeichenprimitiven):
  - Trennung der Zuständigkeiten, insbesondere automatisches (Neu)Zeichnen der Figuren
  - Programmierer muss nicht zuständig sein für das Zeichnen (er darf jedoch bei Bedarf)
  - Einfachere Umsetzung der nicht-trivialen, wirklichkeitsnäheren Modelle

Das GeoFaSC-Konzept

Wie vermeidet GeoFaSC die Problematiken?

- **Vs.** Problematik 1 (Verleitung zu trivialen, inadäquaten OOM):
  - Fehlendes, universales Zeichenwerkzeug und automatisches (Neu)Zeichnen von Figuren
  - Defokussierung des Zeichnens und ,,Blick für Wesentliches“
  - Ermutigung zu nicht-trivialen, wirklichkeitsnäheren Modellen
- **Vs.** Problematik 2 (Fokussierung von Zeichenprimitiven):
  - Trennung der Zuständigkeiten, insbesondere automatisches (Neu)Zeichnen der Figuren
  - Programmierer muss nicht zuständig sein für das Zeichnen (er darf jedoch bei Bedarf)
  - Einfachere Umsetzung der nicht-trivialen, wirklichkeitsnäheren Modelle

Das GeoFaSC-Konzept

Wie vermeidet GeoFaSC die Problematiken?

- **Vs.** Problematik 1 (Verleitung zu trivialen, inadäquaten OOM):
  - Fehlendes, universales Zeichenwerkzeug und automatisches (Neu)Zeichnen von Figuren
  - Defokussierung des Zeichnens und ,,Blick für Wesentliches“
  - Ermutigung zu nicht-trivialen, wirklichkeitsnäheren Modellen
- **Vs.** Problematik 2 (Fokussierung von Zeichenprimitiven):
  - Trennung der Zuständigkeiten, insbesondere automatisches (Neu)Zeichnen der Figuren
  - Programmierer muss nicht zuständig sein für das Zeichnen (er darf jedoch bei Bedarf)
  - Einfachere Umsetzung der nicht-trivialen, wirklichkeitsnäheren Modelle

Anwendungsmöglichkeiten

- Allgemein: Einführung in die OOP mit Java und Fokus Grafikorientierung sowie Vermeidung der SuM-Problematiken
- Unterrichtspraktische Beispiele (z.T. adaptiert aus [Schriek 2006]):
  - `Lamp`, `TrafficLight` und `DiscoLights` (Beziehungen, Komponentenansatz)
  - `Bullet` und `FlyingBullets` (Beziehungen, Animation)
  - `Pencil`, `Wolf` und `Rotkaeppchen` (Eigene Klassen)
  - `Windradpark`, `Windrad` und `TwinFluegel` (Kontrollstrukturen)
  - `Dictatorship` (1- und 2-dim. Arrays)

Kurze Demo...

Voraussetzungen

- Didaktische:
  - Lerngruppen der Sek II
  - Mathematische Kenntnis der grundlegenden Figuren
  - Koordinatenbegriff und Koordinatensystem (auch verschachtelte)
  - Grundlegende Englisch-Kenntnisse (da Implementierung in Englisch)
  - KEINE Kenntnisse in Swing und des MVC-Entwurfsmusters nötig
- Technische:
  - Java Development Kit (ab Version 1.5.20)
  - Entwicklungsumgebung (z.B. BlueJ, Eclipse, Netbeans)
  - Zugang zur Dokumentation/ Referenz

Literatur

[Fowler] Fowler, Amy:
A Swing Architecture Overview.
Bericht, Oracle Sun Developer Network (SDN).
http://java.sun.com/products/jfc/tsc/articles/architecture/ (abgerufen: 4. März 2011).

[GI 2008] Gesellschaft für Informatik:
Grundsätze und Standards für die Informatik in der Schule. Bildungsstandards Informatik für die Sekundarstufe I. Empfehlungen der Gesellschaft für Informatik e.V. erarbeitet vom Arbeitskreis "Bildungsstandards".
http://www.gi-ev.de/fileadmin/gliederungen/fb-iad/fa-ibs/Empfehlungen/bildungsstandards_2008.pdf (abgerufen: 24. August 2011).

[Kmk 1989] Kultusministerkonferenz:
Einheitliche Prüfungsanforderungen Informatik.
http://www.kmk.org/fileadmin/veroeffentlichungen_beschluesse/1989/1989_12_01-EPA-Informatik.pdf (abgerufen: 31. März 2011), 1989.
i.d.F. vom 05.02.2004.

[Loy u.a. 2002] Loy, Marc; Eckstein, Robert und Wood, David:
Java Swing.
O'Reilly Media, 2002.

[MfSuW 1999] Ministerium für Schule und Weiterbildung:
Richtlinien und Lehrpläne für die Sekundarstufe II - Gymnasium/Gesamtschule in Nordrhein-Westfalen. Informatik.
Ritterbach Verlag, 1999.

Literatur

[Schriek 2006] Schriek, Bernhard:
Informatik mit Java - Eine Einführung mit BlueJ und der Bibliothek Stifte und Mäuse, Bände I-III.
Nili-Verlag, 2005–2007.

[Spollwig 2004] Spollwig, Siegfried:
Kritisches zu Stiften und Mäusen - Was ist objektorientierte Modellierung?
LOG IN, 130, 2004.

[Spollwig 2006] Spollwig, Siegfried:
Von Stiften und Mäusen oder ,,Was heisst objektorientierte Modellierung?"
`http://oszhdl.be.schule.de/gymnasium/faecher/informatik/didaktik/sum/sum-kritik.htm` (abgerufen: 4. März 2011), 2006.

[Spollwig 2008] Spollwig, Siegfried:
delphi class in a box.
Cornelsen, 2008.

[Wolf 2010] Wolf, Christian:
GeoFaSC - Geometric Figures as Swing Components.
`http://www.geofasc.de` (abgerufen: 4. März 2011), 2010.

[Wolf 2011] Wolf, Christian:
Praxisbericht: Objektorientierte Programmierung mit geometrischen Figuren als Komponenten. In: Weigend, M.; Thomas, M.; Otte, F. (Hrsg.): Informatik mit Kopf, Herz und Hand (INFOS 2011 14. GI-Fachtagung Informatik und Schule, Münster). Bd. 1 Münster, S. im Druck.
ZfL-Verlag, 2011.