UNIVERSITÄT PASSAU

Fakultät für Informatik und Mathematik

Mit Objekten arbeiten

Fundamentale Ideen und kleine Beispielexperimente in der 10.Jahrgangsstufe

Ute Heuer

Prolog: Fundamentale Ideen

- In der Informatik ändern sich die Inhalte schnell.
- Die Gefahr ist groß, dass sich der Unterricht in (produktspezifischen) Details verliert und dass das gelehrte Wissen schon bald nicht mehr gültig ist.
- Also gilt es, langlebige von weniger langlebigen Inhalten zu scheiden und so die Bedeutsamkeit eines Sachverhalts zu überprüfen.

nach Hartmann: Informatikunterricht planen und durchführen

Grundlegende Ideen in der 10.Klasse?

Eine Auswahl

- Objekte sind Dienstleister
- Ein Nutzer kann den Zustand eines Dienstanbieters (gewollt oder ungewollt) verändern -> Zustandsmodelle schaffen Überblick
- Autoren einer Klasse verwenden Attribute, Parameter und lokale Variablen passend – Namensräume in Klassendefinitionen
- Über Beziehungen können Objekte Dienste nutzen, die andere Objekte anbieten. Autoren einer Klasse ermöglichen Beziehungen. Welche Beziehungen Objekte dann tatsächlich eingehen beeinflussen Nutzer oft mit.

Auf diese Auswahl konzentriere ich mich im Folgenden.

Ausblick auf weitere grundlegende Ideen

- Klassendefinitionen generalisieren – **Polymorphie**
  - In einem guten Entwurf werden manche Klassen in einer Generalisierungsstruktur definiert.
  - In solchen Fällen können Nutzer sich auf Garantien für Dienstangebote verlassen, die z.B. in einer Oberklasse vereinbart sind. Typischerweise interessiert Nutzer dann gar nicht, von welcher Unterklasse ein Objekt ist.
- Wenn hauptsächlich das Dienstangebot von Objekten interessiert (und nicht interessiert, wie diese Dienste realisiert werden oder welche Attribute ein Objekt hat), dann spricht der Informatiker von **Objekten eines Typs**

Darauf gehe ich in diesem Vortrag nicht näher ein.

Gliederung dieses Vortrag

1. Objektorientierte Konzepte
2. Zustandsmodelle
3. Algorithmen (hierzu nur ein sehr knapper Hinweis auf ein mögliches Werkzeug)
4. Beziehungen zwischen Objekten nutzen
5. Falls Zeit bleibt ein kurzer Ausblick auf: Klassendefinitionen spezialisieren

Abschnitt 1

- Objektorientierte Konzepte

Dienst

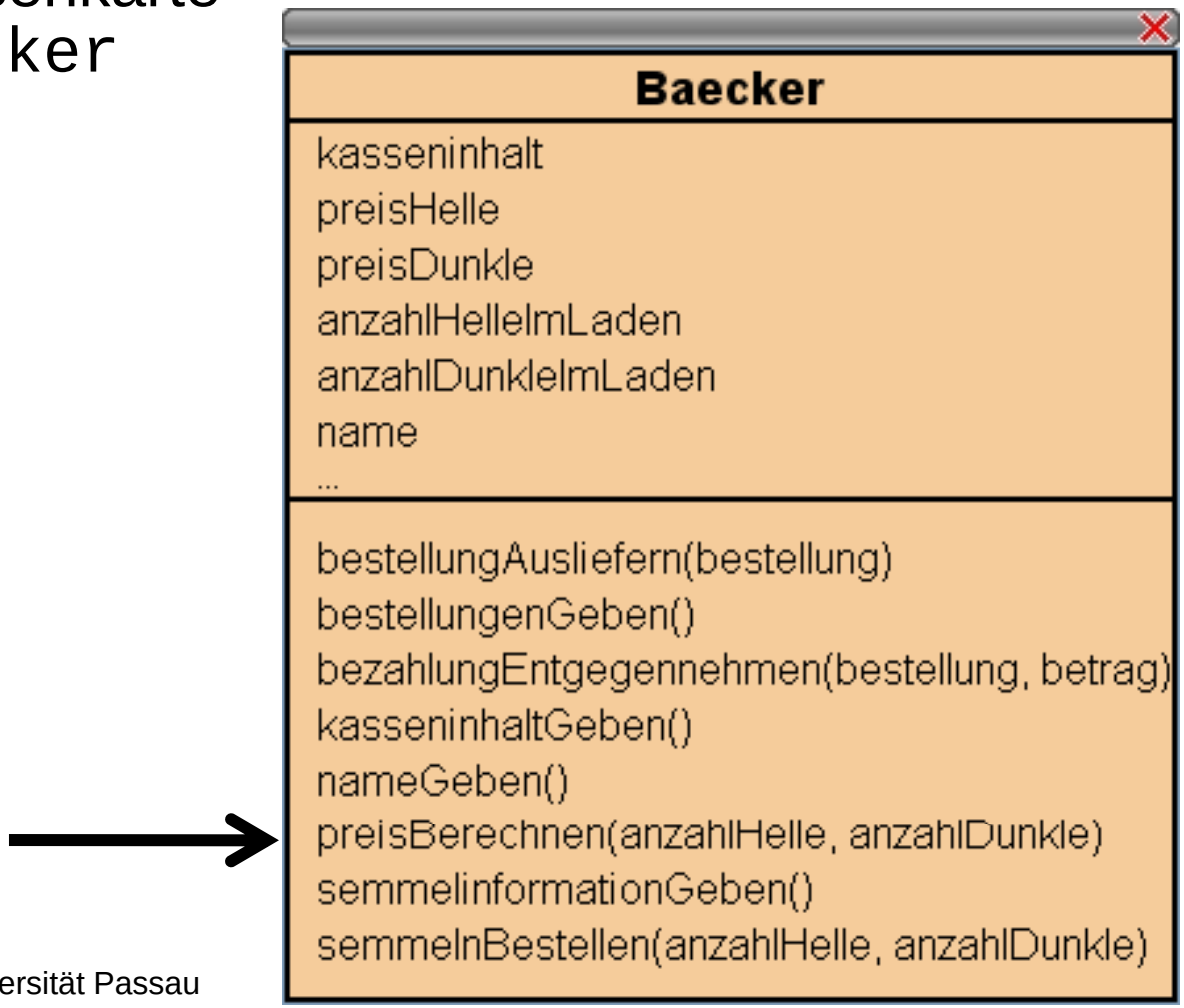
Eine Begriffsklärung

- Manche Objekte verfolgen eine ähnliche Strategie wie der Chef eines Betriebs:
  - Aufgaben delegieren und **Dienste nutzen**
  - Grund: Alles selbst zu erledigen ist nicht wirtschaftlich
- Andere Objekte **bieten** nachgefragte **Dienste an**

Objekte einer Klasse bieten **Dienste**, man sagt auch öffentliche **Methoden** an. Diese werden in der zugehörigen Klassenkarte notiert.

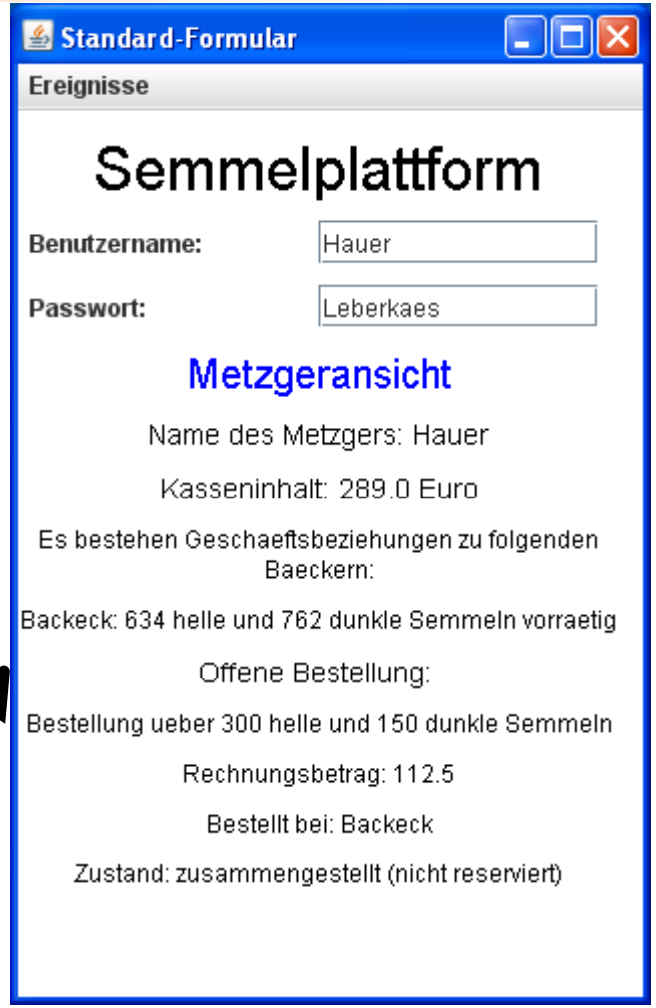
Ein Baecker - Beispiel

- Klassenkarte `Baecker`



Dienste nutzen – Methoden aufrufen

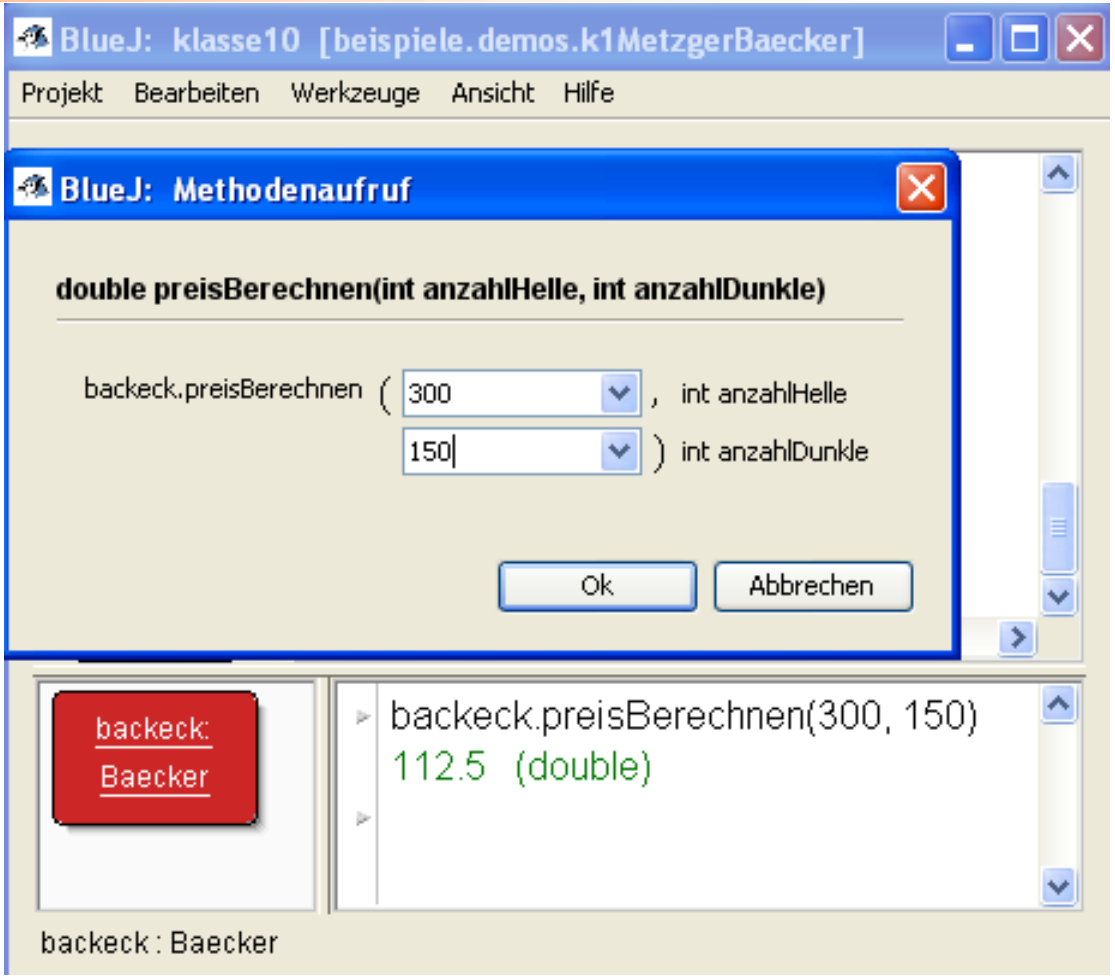
- „Semmelplattform“
- Kunden (z.B. Metzger) können bei einigen Bäckern Semmeln bestellen oder Kostenvoranschläge anfordern. Die teilnehmenden Bäckereien werden auf diesem Portal durch Objekte einer Klasse Baecker repräsentiert.
- Wie geht ein Kunde prinzipiell vor, wenn er an einer Kostenrechnung für 300 Kaisersemmeln und 150 Vollkornsemmeln interessiert ist?



Umsetzung eines Studierenden für interessierte Lehrer:
BlueJ Projekt klasse10 → beispiele.demos.k1MetzgerBaecker

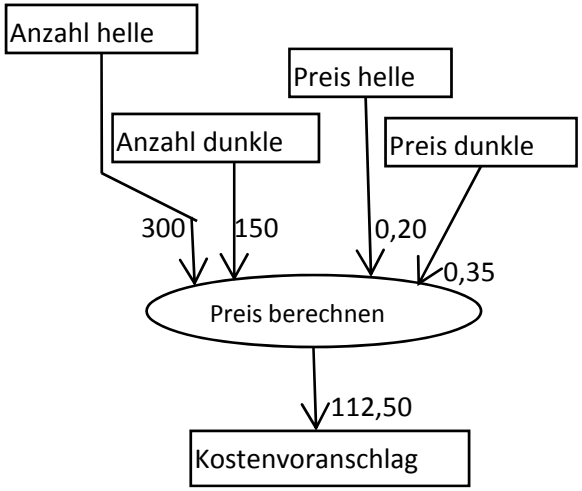
Ein Baecker - Beispiel

- Auswahl des Objekts `backeck`, welches den Dienst anbietet
- Auswahl des Dienstes `preisBerechnen`
- Passende Werte eingeben
- Ausgabewert „interessiert betrachten“



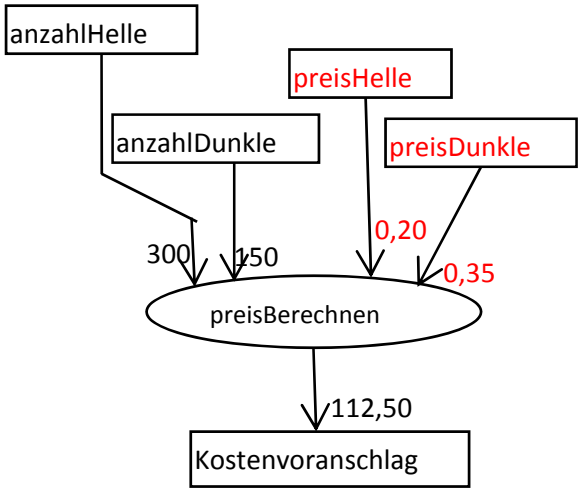
Fakultativ: preisBerechnen in Klasse 9

- Der Nutzer eines Rechenblatts muss sich um vier Eingaben kümmern, wenn er an einem aktuellen Kostenvoranschlag interessiert ist:



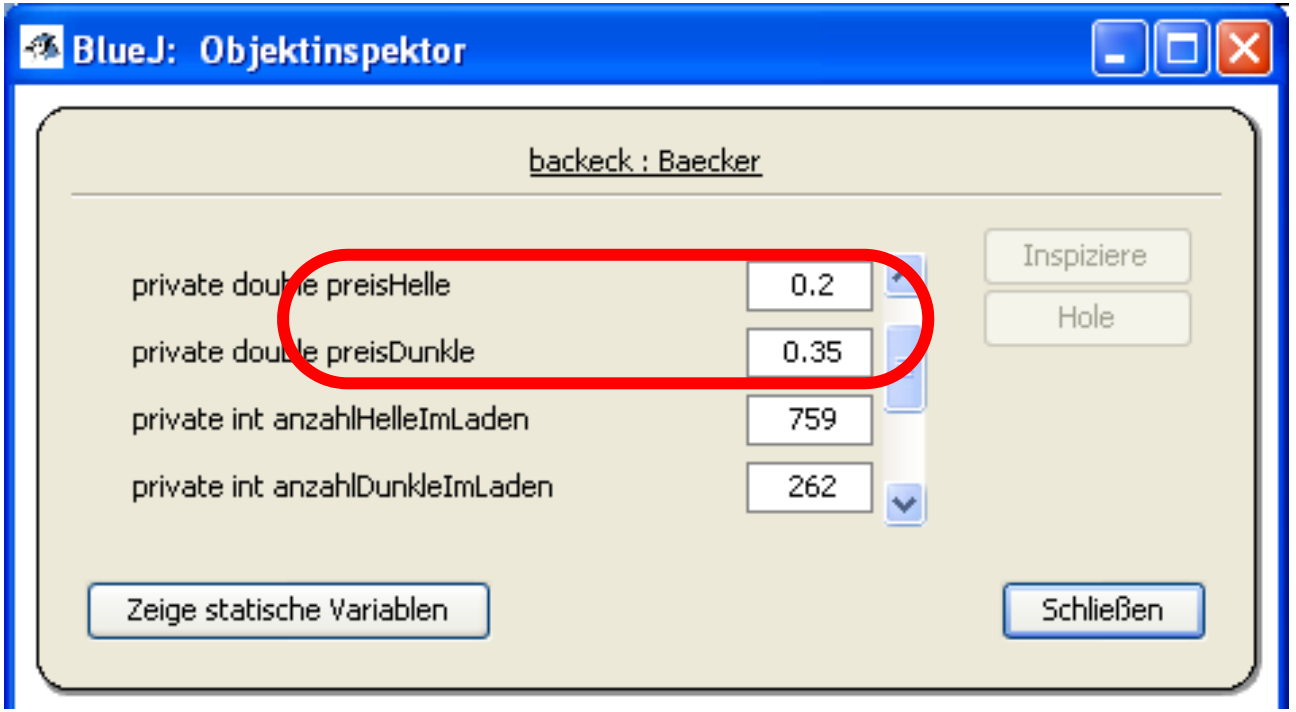
Fakultativ: preisBerechnen in Klasse 10

- Der Nutzer des Dienstes preisBerechnen, den ein Objekt der Klasse `Baecker` anbietet, muss sich nur um zwei Eingaben kümmern.
- Die rot markierten Eingaben werden vom Bäckerobjekt selbst bereitgestellt:



preisBerechnen in Klasse 10

- Die rot markierten Eingaben werden vom Bäckerobjekt selbst bereitgestellt:



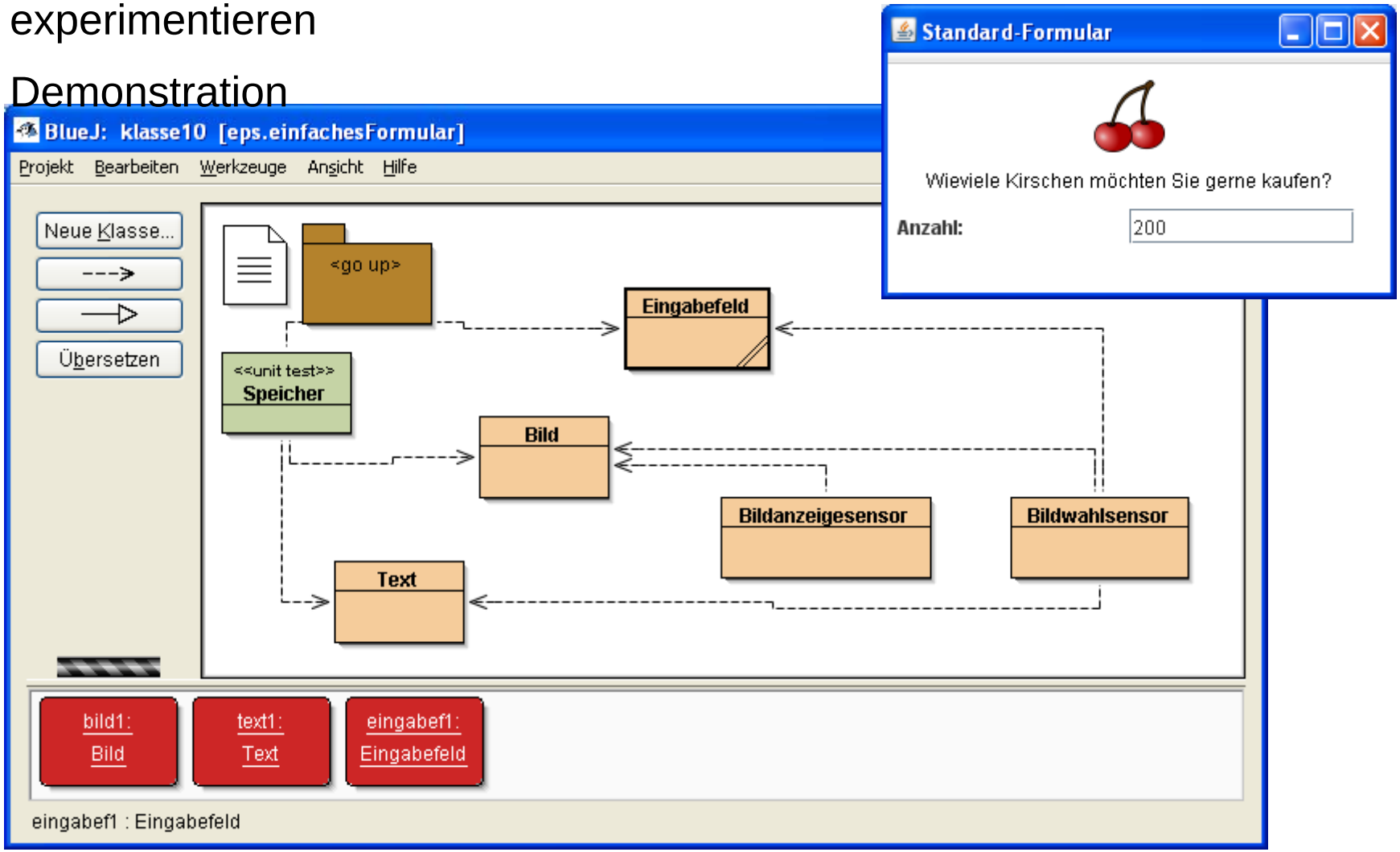
preisBerechnen in Klasse 10

Fazit

- Als Kunde einer Bäckerei erwartet man ja auch nicht, dass man über den Preis der Brötchen frei bestimmen darf. Die Anzahl der Semmeln, die man zu kaufen plant, möchte man hingegen schon selbst festlegen können.
- <-> Sinnvolle Verteilung von Zuständigkeiten

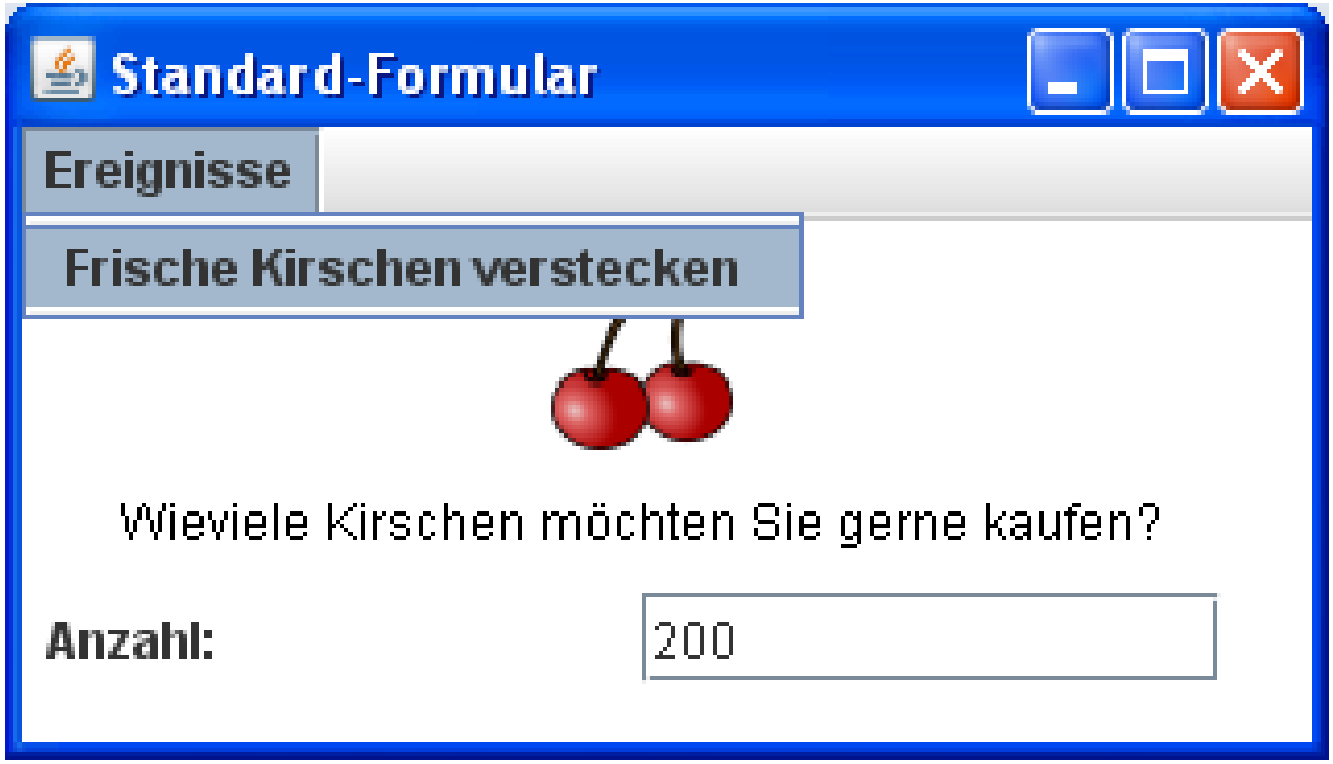
Formularexperimente (zusammen mit Florian Prager entwickelt)

- Die vordefinierten Klassen Eingabefeld, Text, Bild, … sind gegeben
- Schüler können Objekte dieser Klassen erzeugen und mit diesen experimentieren
- Demonstration



Formularexperimente

- Erzeugt man Objekte der Klassen Text, Eingabefeld, Bild und Bildanzeigesensor geeignet, kann das so aussehen:



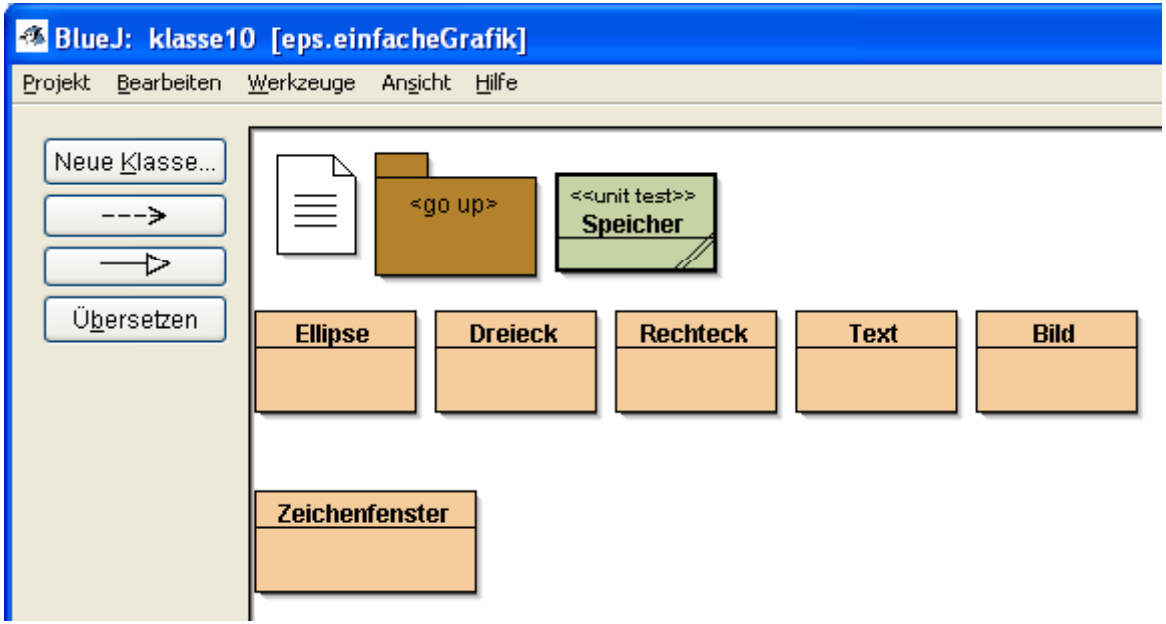
Objekte sind Dienstleister

- Wie findet man heraus, was Objekte zu bieten haben?
  - Schüler experimentieren zu Beginn der Jahrgangsstufe 10 mit vordefinierten Klassen.
  - Sie erzeugen Objekte und testen Methoden.

Weitere Experimente mit vordefinierten Klassen

Eine Auswahl

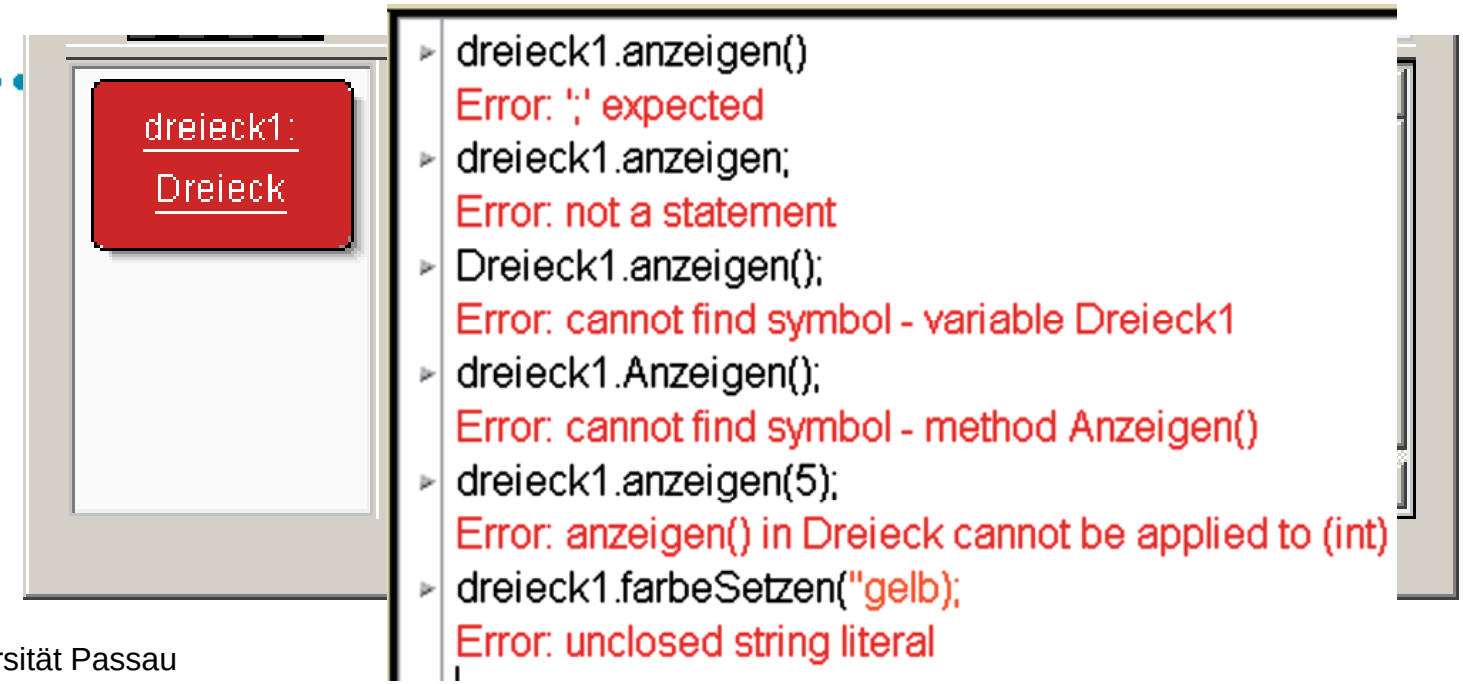
- Schreiber → BlueJ Projekt klasse10: beispiele.k1Schreiber
- Rechtecke, Ellipsen, Dreiecke … erzeugen
- Mit einer Kaffeegrafik experimentieren → beispiele.k1Kaffeegrafik



Dienste nutzen - Fehlermeldungen

1.13 Einfache Grafik IV – Fehlermeldungen (E)

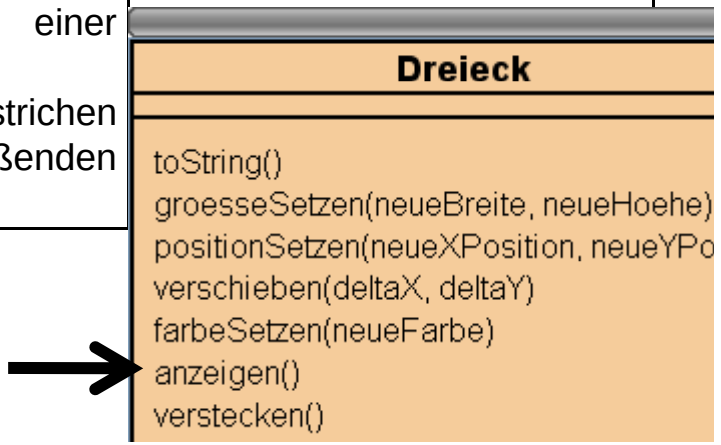
Lernen Sie typische Fehlermeldungen ihrer Entwicklungsumgebung kennen! Ziehen Sie gegebenenfalls ein Wörterbuch zu Rate. Versuchen Sie, die Meldungen zu verstehen, auch wenn Sie nicht treffend ausfallen sollten. Notieren Sie dazu fünf verschiedene Meldungen und formulieren Sie jeweils mit eigenen Worten, was sie bedeuten und wie man den Fehler beheben kann!



Auszug aus unserem Lehrermaterial (Lösungen zu Aufgabe 1.13)

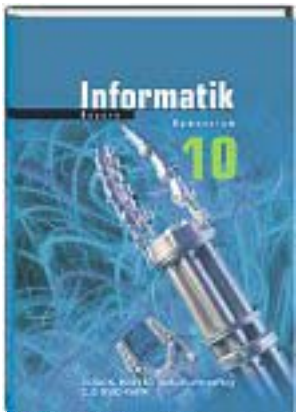
Anweisung	Erläuterung der Fehlermeldung (im Kontext)	mögliche Korrektur
2	Statement, engl. Anweisung (siehe z.B. dict.leo.org) Not a statement heißt frei übersetzt: Diese Zeichenfolge passt in kein „Schreibweisen-Schema“, welches in der Sprache Java festgelegt ist. Objekte bieten Dienste an. Will man diese nutzen, darf (in Java) das Klammerpaar nicht vergessen werden, auch wenn der Dienst gar keine Eingaben entgegennimmt.	dreieck1.anzeigen**()**;
3	**D**`reieck1` gibt es nicht, ein entsprechendes Objekt wurde nicht erzeugt und „benamt“. In Java wird zwischen Groß- und Kleinschreibung unterschieden.	**d**reieck1.anzeigen();
5	Dreieckobjekte bieten den Dienst anzeigen() an. Dieser Dienst verlangt nicht nach Nutzereingaben, dann darf man auch keine machen.	dreieck1.anzeigen(~~5~~);
6	**litera**, lat. Buchstabe. Der Begriff **Literale** bezeichnet Zeichenfolgen, die zur Darstellung der Werte von z.B. Ganzzahlen, Fließkommazahlen oder Strings in einer Programmiersprache zulässig sind. (nach Wikipedia) Strings müssen (in vielen Sprachen) mit Anführungsstrichen beginnen und enden. Hier wurden die schließenden Anführungsstriche vergessen.	dreieck1.farbeSetzen(“gelb**“**);

Alle Dienste, die Dreieckobjekte anbieten, sind übersichtlich in der zugehörigen Klassenkarte notiert:



Vorsicht Werbung

- Schulbuch Informatik 10 Bayern G
- Wir (U. Heuer, F. Fiedler, S. Ritzer - alle Didaktik der Informatik, Uni Passau) haben ein Konzept für das erste Halbjahr der 10.Klasse entwickelt, in Schülerkursen an der Uni erprobt und in dem abgebildeten Schulbuch umgesetzt.



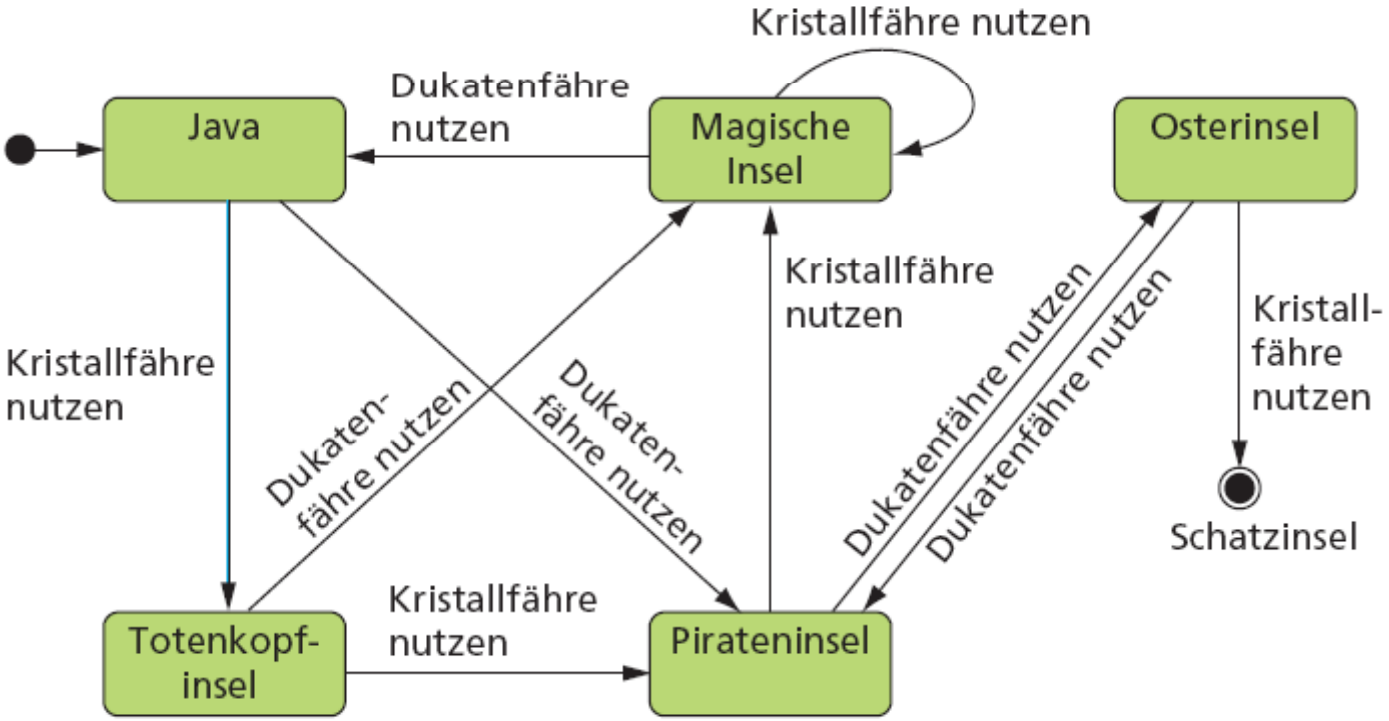
Abschnitt 2

- Zustandsmodelle

Zustandsmodelle

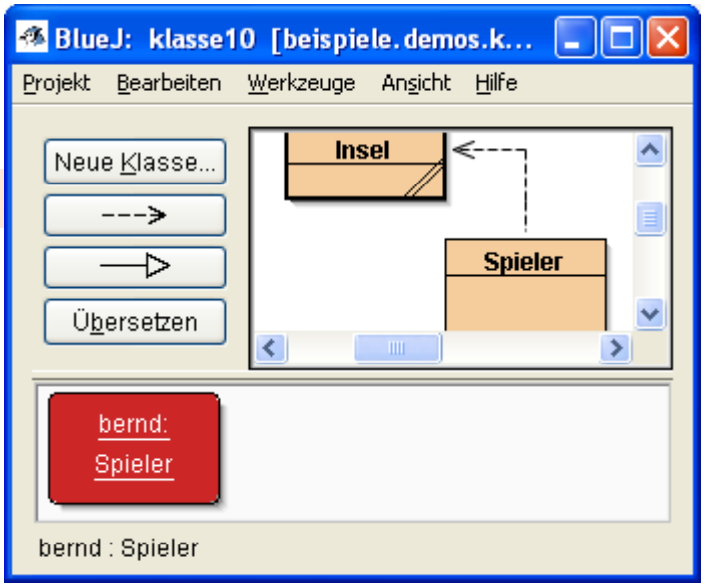
- Kaffeeautomat, MP3-Spieler, Präsentationsprogramm, Eintrittskartenreservierungssystem
- Wenn man von den Diensten dieser Systeme profitieren will, müssen bestimmte Tasten in einer bestimmten Reihenfolge gedrückt werden.
- Schüler analysieren, planen und setzen Abläufe in Programme um.
- Folgende Beispiele sollen Ihnen einen Eindruck von Umsetzungsmöglichkeiten geben:

Schatzsuche



Zustandsmodell des Spiels Schatzsuche

Schatzsuche



Während des Spiels fährt Bernd von einer *Insel* zur anderen, indem er *Fährdienste nutzt.* Informatiker sprechen statt von der Insel, auf der sich der Spieler gerade befindet, vom **Zustand** des Spielers. Statt „ein Menüeintrag wird angewählt“ sagt der Informatiker: ein Ereignis tritt ein. Ein **Ereignis** kann eine Zustandsänderung erzwingen.

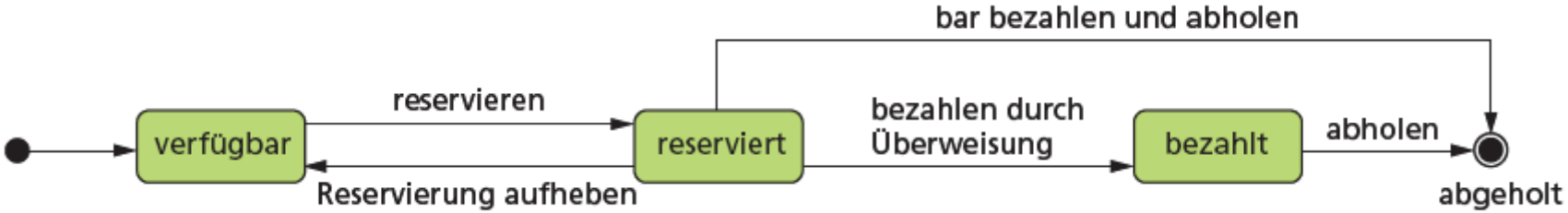
Zwischenspeicher – Aufgabe

Mit Hilfe eines Zwischenspeichers, auch Zwischenablage genannt, können Elemente zwischen Standardprogrammen leicht ausgetauscht werden.

Das Bild zeigt zwei verschiedene Zustände eines Standardprogramms, die für den Problembereich „Einfügen, Kopieren und Ausschneiden von Elementen“ relevant sind.

- Wie viele weitere relevante Zustände gibt es?
- Geben Sie zu dem beschriebenen Problembereich ein Zustandsdiagramm an! Berücksichtigen Sie dabei auch das Ereignis Element markieren in sinnvoller Weise.
- Schreiben Sie einen aussagekräftigen Hilfe-Text, der die Arbeit mit der Zwischenablage für Benutzer von Standardprogrammen kurz erläutert.

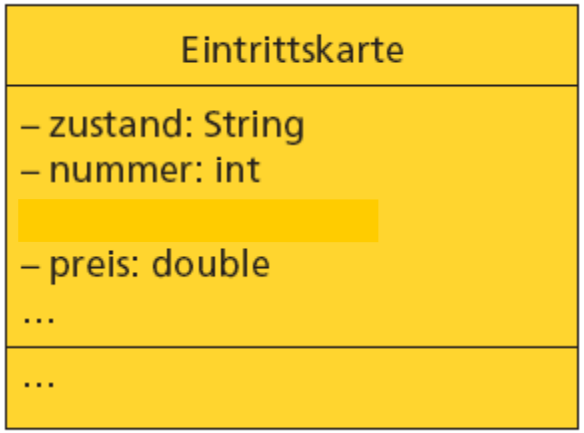
Hintertupfingener Opernfestspiele



Das Zustandsdiagramm zeigt, was in Hintertupfingen möglich ist, und was nicht:
Eine Karte kann beispielsweise mehrfach reserviert und storniert werden. Eine bezahlte oder abgeholte Karte kann in Hintertupfingen jedoch nicht mehr zurückgegeben werden.

Der Informatiker sagt: Das Zustandsdiagramm zeigt alle „im Leben einer Eintrittskarte“ möglichen Abläufe.

Klassendefinition Eintrittskarte




```
public class Eintrittskarte {
    /*---------------Attribute-----*/
    private String zustand;
    private int nummer;
    private double preis;
//     ...
    /*---------------Konstruktor---*/
    public Eintrittskarte(int neueNummer) {
        // Wert zuweisen
        // Der Wert "verfuegbar" wird dem Attribut zustand zugewiesen
        // <---
        // Den Wert "verfuegbar" im Attribut zustand speichern/halten
        zustand = "verfuegbar";
        // Den Wert des Eingangsparameters (spezielle lokale Variable) neueNummer
        // im Attribut nummer speichern
        nummer = neueNummer;
        preis = 24.0;
    }
```

… siehe **`beispiele.k2Oper.Eintrittskarte`**

Eintrittskarte - Schülerfehler

- „Syntaxfehler“ der nicht vom Compiler erkannt wird:
  public void Eintrittskarte() …
- public Eintrittskarte (int neueNummer) …
  - Die nötige Wertzuweisung nummer = neueNummer unterbleibt. Und in einer Methode nummerGeben() wird geschrieben return neueNummer.
  - „Wo“ ist hier der Wert? Wozu diese Zuweisung?
  - Statt neueNumer wird nummer als Name des Eingangsparameters verwendet. Die nötige Zuweisung unterbleibt. Denn eine Zuweisung der Art nummer = nummer scheint wirklich überflüssig…
- Wie würden Sie die Fehler im Gespräch mit Schülern klären?

Eingeführte Begriffe für Variablen

Attribut

Eingangsparameter

Lokale Variable

```
public class Eintrittskarte {
    /*---------------Attribute-----*/
    private String zustand;
    private int nummer;
    private double preis;
//     ...
    /*---------------Konstruktor---*/
    public Eintrittskarte(int neueNummer) {
        // Wert zuweisen
        // Der Wert "verfuegbar" wird dem Attribut zustand zugewiesen
        // <---
        // Den Wert "verfuegbar" im Attribut zustand speichern/halten
        zustand = "verfuegbar";
        // Den Wert des Eingangsparameters (spezielle lokale Variable) neueNummer
        // im Attribut nummer speichern
        nummer = neueNummer;
        preis = 24.0;
    }

    /*---------------Methoden------*/
    public double mehrwertsteuerGeben(int mwstsatz) {
        double mehrwertsteuer;
        mehrwertsteuer = preis * mwstsatz / (100 + mwstsatz);
        return mehrwertsteuer;
    }
    public void reservieren() {
        if (zustand.equals("verfuegbar")) {
            zustand = "reserviert";
        }
    }
//     ...
}
```

Eintrittskarte - Schülerfehler

- Der Wert „verfuegbar“ wird bei der Zuweisung im Konstruktor verwendet. Bei der Umsetzung der Bedingung in der Methode reservieren wird jedoch der Wert „verfügbar“ verwendet. Schülerfrage nachdem das Problem gelöst war: Warum werde ich vom Computer (vom Compiler) nicht gewarnt?
- Wie würden Sie antworten?
- Eine Methode wird folgendermaßen definiert:

```
/*---------------Methoden------*/
public double mehrwertsteuerGeben(int mwstsatz, double mehrwertsteuer) {
    mehrwertsteuer = preis * mwstsatz / (100 + mwstsatz);
    return mehrwertsteuer;
}
```

- Der Schüler findet das ok, denn es funktioniert ja. Er zeigt mir das, indem er eintrittskarte1.mehrwertsteuerGeben(19, 0) aufruft.
- Wie würden Sie reagieren?

Gegenüberstellung – was kennzeichnet …

Attribut	Lokale Variable	Eingangsparameter
Name ist in der ganzen Klassendefinition bekannt	Name ist nur lokal in einer Methodendefinition bekannt	Name ist in der ganzen Methodendefinition bekannt
Autoren sorgen dafür, dass in Methodendefinitionen Attributwerte sinnvoll verändert werden	Autoren benutzen lokale Variablen um Ergebnisse lokal zwischen zu speichern	Autoren benutzen Parameter, um Werte von Nutzern entgegennehmen zu können und nicht zum Zwischen-speichern von Ergebnissen
Attributwertänderungen beschreiben Zustandsänderungen des zugehörigen Objekts, sie sind „von Dauer“	Änderungen des Werts einer lokalen Variablen sind „nicht von Dauer“ – spätestens nach Abar-beitung des Methoden-aufrufs ist der Wert „weg“	Setzt ein Nutzer den Wert eines Eingangsparameters, so ist dies ebenfalls „nicht von Dauer“ – nach Abarbeitung des Methodenaufrufs ist der Wert „weg“
Nutzer interessieren sich mehr für Dienste und weniger (direkt) für Attribute	Nutzer interessieren sich nicht für lokale Variablen	Nutzer interessieren sich für Eingangsparameter, da sie diese mit passenden Werten „befüllen“ müssen

Sondierende und verändernde Methoden

Festlegung

- Alle Methoden, die bei Aufruf den Zustand eines Objekts niemals verändern, nennen wir **sondierende Methoden**. Nutze ich sondierende Methoden bekomme ich **immer einen Wert zurückgegeben**.
- Alle restlichen Methoden nennen wir **verändernde Methoden**. Nutze ich verändernde Methoden, bekomme ich **meistens keinen Wert zurückgegeben**. Entsprechend wird der Platzhalter **void** in der Methodendefinition vor dem Methodennamen verwendet.
- Anmerkung – nicht für Schüler: Es gibt unter Informatikern vollkommen akzeptierte verändernde Methoden, die Werte zurückgeben.

Sondierende und verändernde Methoden

- Warum ist es meines Erachtens hilfreich, diese beiden Begriffe in der 10.Klasse einzuführen?

Aus Nutzersicht

- **Sondierend**e Dienste „hinterlassen keine Spuren“, also auch keine ungewollten. Man kann Sie in dieser Hinsicht unbesorgt nutzen.
- Nutzt man hingegen **verändernd**e Dienste, muss man sich gewissenhaft fragen, ob die angestossenen Änderungen tatsächlich beabsichtigt waren.

Sondierende und verändernde Methoden

- Bei der Eintrittskarte - Nutzersicht:
  - Die Mehrwertsteuer können Sie sich so oft berechnen lassen, wie Sie möchten. Der Zustand ihrer Karte verändert sich nicht.
  - Haben Sie Ihre Karte aber erstmal erfolgreich durch Überweisung bezahlt, sind Sie in Hintertupfingen dran. Ihr Geld sehen Sie nie wieder zurück (aber dafür sehen Sie vielleicht eine schöne Oper, wenn Sie ihre Karte abholen und hingehen).

Sondierende und verändernde Methoden

Aus Autorensicht

- Habe ich wirklich eine **sondierend**e Methode geschrieben oder habe ich „versehentlich“ Attributwerte verändert?
  Habe ich meine sondierende Methode geeignet benannt? xxxGeben, xxxBerechnen o.ä. (bei Methoden mit Rückgabetyp boolean auch istReserviert, hatGegriffen, o.ä. „Kurzform einer Ja/Nein-Frage“)
- Sorgt meine **verändernd**e Methode für eine „vollständige“ Zustandsänderung oder habe ich vergessen, Attributwerte anzupassen?
  Habe ich meine verändernde Methode geeignet benannt? xxxSetzen oder xxxVerb, welches die zugehörige Zustandsänderung treffend beschreibt (sicherlich nicht geben, berechnen, istReserviert o.ä.).

Sondierende und verändernde Methoden

Aus Autorensicht

- Habe ich wirklich eine **sondierend**e Methode geschrieben oder habe ich „versehentlich“ Attributwerte verändert?
  Habe ich meine sondierende Methode geeignet benannt? xxxGeben, xxxBerechnen o.ä. (bei Methoden mit Rückgabetyp boolean auch istReserviert, hatGegriffen, o.ä. „Kurzform einer Ja/Nein-Frage“)
- Sorgt meine **verändernd**e Methode für eine „vollständige“ Zustandsänderung oder habe ich vergessen, Attributwerte anzupassen?
  Habe ich meine verändernde Methode geeignet benannt? xxxSetzen oder xxxVerb, welches die zugehörige Zustandsänderung treffend beschreibt (sicherlich nicht geben, berechnen, istReserviert o.ä.).

Beispiel Eintrittskarte - Autorensicht

- Die sondierende Methode mehrwertsteuerGeben darf die Mehrwertsteuer nicht in einem Attribut speichern.

sondierende Methode

```
public double mehrwertsteuerGeben(int mwstsatz) {
    double mehrwertsteuer;
    mehrwertsteuer = preis * mwstsatz / (100 + mwstsatz);
    return mehrwertsteuer;
}
```

```
public class Eintrittskarte {
    /*---------------Attribute-----*/
    private String zustand;
    private int nummer;
    private double preis;
    private double mehrwertsteuer;
... Konstruktor ...
    /*---------------Methoden------*/
    public double mehrwertsteuerGeben(int mwstsatz) {

        mehrwertsteuer = preis * mwstsatz / (100 + mwstsatz);
        return mehrwertsteuer;
    }
...
```

eine kleine Änderung:

Lokale Variable „wird zu“ Attribut!

keine sondierende Methode mehr da zustandsverändernd!

falscher Name, verständnisblockierend!

Beispiel Eintrittskarte - Autorensicht

Falsch auch deshalb, weil es mehr Arbeit macht als nötig. Methoden, die den Preis anpassen dürfen bspw. nicht vergessen, auch die Mehrwertsteuer anzupassen.
Fazit:
Wenn solche Abhängigkeiten leicht vermieden werden können, vermeide diese!!

```
public class Eintrittskarte {
    /*---------------Attribute-----*/
    private String zustand;
    private int nummer;
    private double preis;
    private double mehrwertsteuer;
... Konstruktor ...
    /*---------------Methoden------*/
    public double mehrwertsteuerGeben(int mwstsatz) {

        mehrwertsteuer = preis * mwstsatz / (100 + mwstsatz);
        return mehrwertsteuer;
    }
...
```

falscher Name, verständnisblockierend

Beispiel Eintrittskarte - Autorensicht

sondierende Methode

```
public double mehrwertsteuerGeben(int mwstsatz) {
    double mehrwertsteuer;
    mehrwertsteuer = preis * mwstsatz / (100 + mwstsatz);
    return mehrwertsteuer;
}
```

Möglich und richtig ist folgende Variation:

```
public class Eintrittskarte {
    /*---------------Attribute-----*/
    private String zustand;
    private int nummer;
    private double preis;
    private double mwstsatz;
```

Keine Abhängigkeiten zwischen Attributen

… im Konstruktor mwstsatz setzen …

Eingangsparameter „wird zu“ Attribut!

```
    public double mehrwertsteuerGeben(                ) {
        double mehrwertsteuer;
        mehrwertsteuer = preis * mwstsatz / (100 + mwstsatz);
        return mehrwertsteuer;
    }
```

möglicher, passender Name

…

Sondierende und verändernde Methoden

Aus Autorensicht

- Habe ich wirklich eine **sondierend**e Methode geschrieben oder habe ich „versehentlich“ Attributwerte verändert?
  Habe ich meine sondierende Methode geeignet benannt? xxxGeben, xxxBerechnen o.ä. (bei Methoden mit Rückgabetyp boolean auch istReserviert, hatGegriffen, o.ä. „Kurzform einer Ja/Nein-Frage“)

- Sorgt meine **verändernd**e Methode für eine „vollständige“ Zustandsänderung oder habe ich vergessen, Attributwerte anzupassen?
  Habe ich meine verändernde Methode geeignet benannt? xxxSetzen oder xxxVerb, welches die zugehörige Zustandsänderung treffend beschreibt (sicherlich nicht geben, berechnen, istReserviert o.ä.).

Beispiel „Eintrittskarte mit Darstellung“

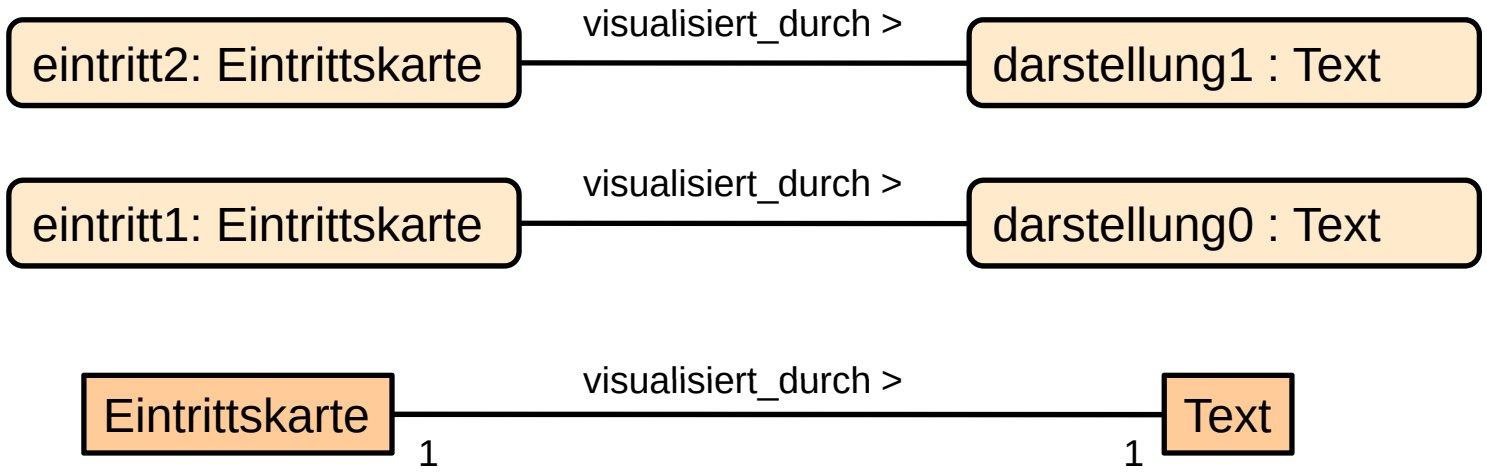
- Der verändernde Dienst abholen() hat nicht nur das Attribut zustand zu aktualisieren sondern auch das Attribut darstellung.

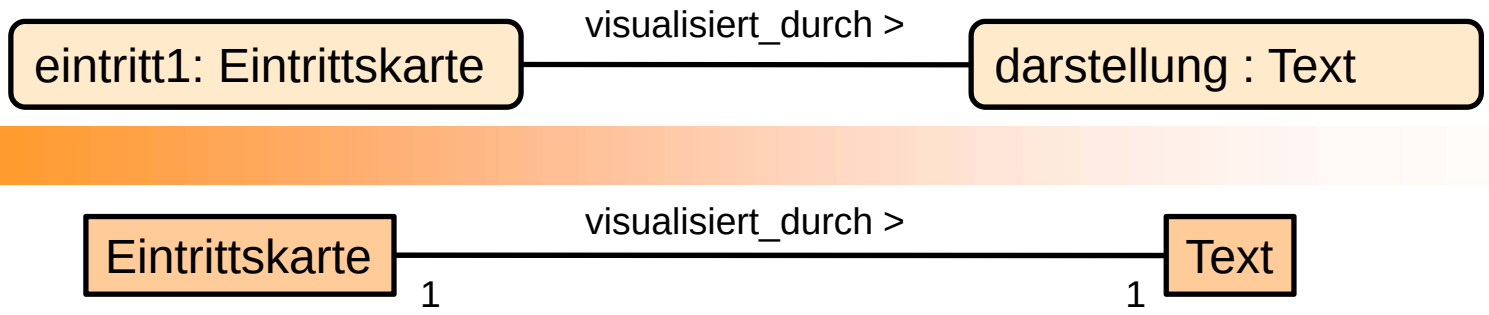

```
public void abholen() {
    if (zustand.equals("bezahlt")) {
        zustand = "abgeholt";
        aktualisieren();
    }
}
//Text-Darstellung aktualisieren
private void aktualisieren() {
    darstellung.inhaltSetzen(infoGeben());
}
//Sondierende Methode
public String infoGeben() {
    return "Diese Eintrittskarte ist " + zustand;
}
```

BlueJ-Projekt klasse10 → beispiele.k4Oper

Einschub: Umsetzung einer 1:1 Beziehung

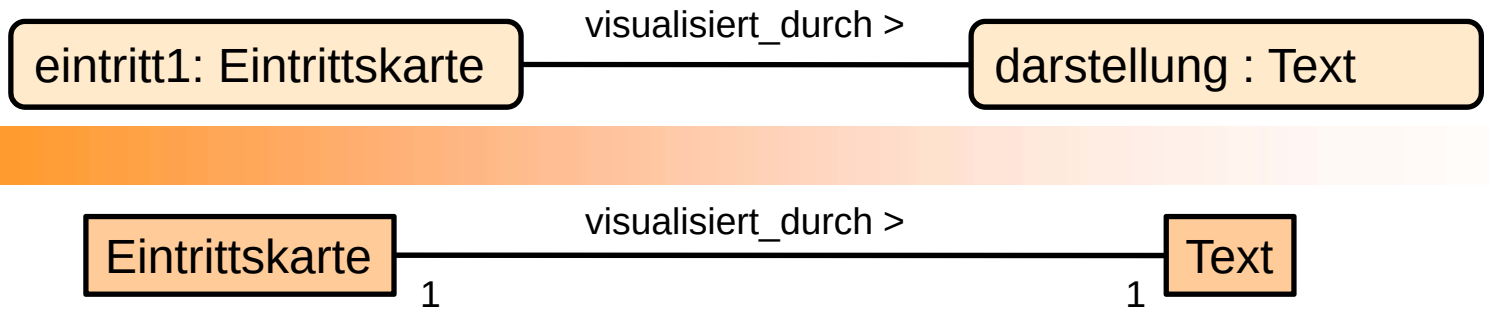
- Kleine grafische Darstellungen motivieren.
- Mit vordefinierten Klassen wie beispielsweise den einfachesFormular-Klassen können solche von Schülern schon im ersten Halbjahr geplant und implementiert werden.
- Möchte man dies, empfiehlt sich eine Information zur Umsetzung von 1:1 Beziehungen an einem Beispiel. (Dies knüpft an die Wiederholung des Begriffs Beziehungen aus der 9.Klasse an.)
- Ich skizziere im Folgenden in drei Schritten, wie man mit Schüler vorgehen kann (kann noch vereinfacht werden, aktualisieren zunächst auflösen…)





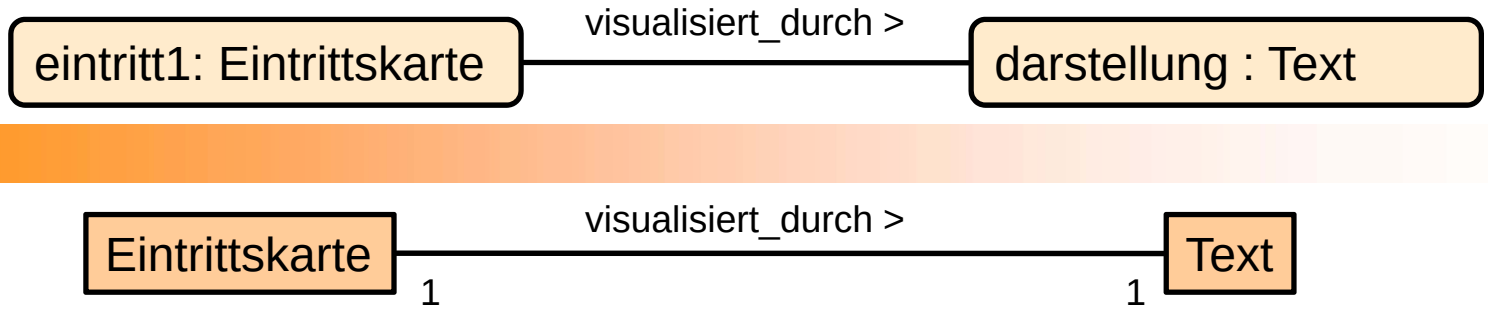

```
public class Eintrittskarte {
    /*---------------Attribute-----*/
    private String zustand;
    // 1:1 Beziehung zwischen Eintrittskarte und Text
    // Wir wollen Dienste des Objekts darstellung nutzen.
    private eps.einfachesFormular.Text darstellung;
```

1.Schritt: Attribut deklarieren

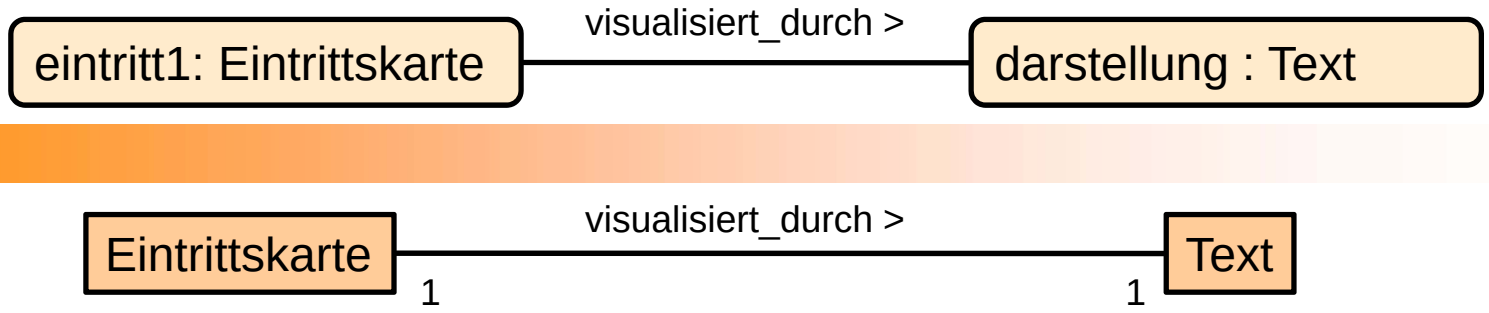



```
public class Eintrittskarte {
    /*---------------Attribute-----*/
    private String zustand;
    // 1:1 Beziehung zwischen Eintrittskarte und Text
    // Wir wollen Dienste des Objekts darstellung nutzen.
    private eps.einfachesFormular.Text darstellung;

    /*---------------Konstruktor---*/
    public Eintrittskarte() {
        zustand = "verfuegbar";
        // Wir erzeugen und initialisieren das Objekt darstellung,
        // es "gehört uns allein".
        darstellung = new eps.einfachesFormular.Text();
        ...
    }
```

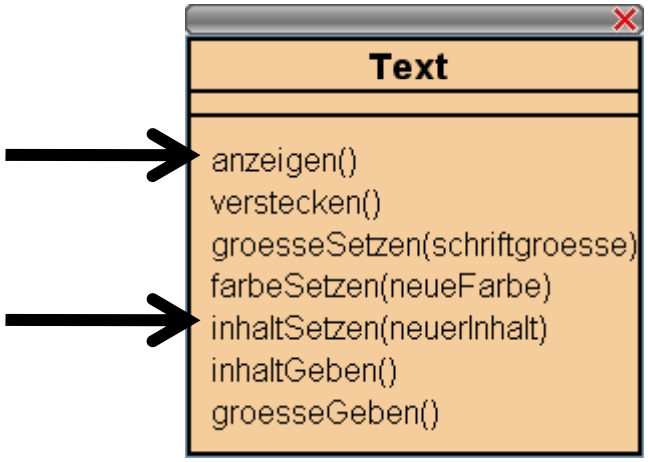
2.Schritt: Objekt erzeugen (2a)



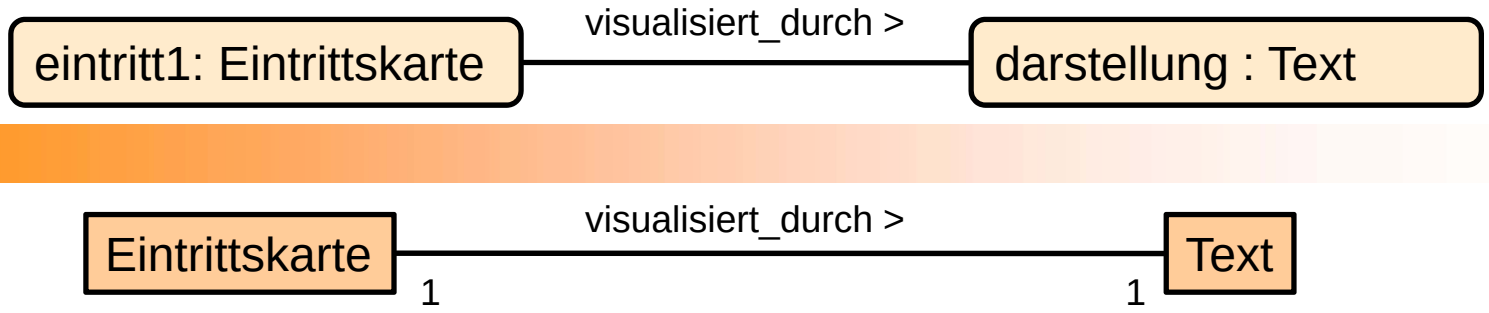

```
import eps.einfachesFormular.Text;
public class Eintrittskarte {
    /*---------------Attribute-----*/
    private String zustand;
    // 1:1 Beziehung zwischen Eintrittskarte und Text
    // Wir wollen Dienste des Objekts darstellung nutzen.
    private eps.einfachesFormular.Text darstellung;

    /*---------------Konstruktor---*/
    public Eintrittskarte() {
        zustand = "verfuegbar";
        // Wir erzeugen und initialisieren das Objekt darstellung,
        // es "gehört uns allein".
        darstellung = new eps.einfachesFormular.Text();
        aktualisieren();
        darstellung.anzeigen();
    }
```




```
/*---------------Konstruktor---*/
public Eintrittskarte() {
    zustand = "verfuegbar";
    // Wir erzeugen und initialisieren das Objekt darstellung,
    // es "gehört uns allein".
    darstellung = new Text();
    aktualisieren();
    darstellung.anzeigen();
}

/*---------------Methoden------*/
 //Darstellung aktualisieren
 private void aktualisieren() {
     darstellung.inhaltSetzen(infoGeben());
 }
 //Sondierende Methode
 public String infoGeben() {
     return "Diese Eintrittskarte ist " + zustand;
 }
```



2.Schritt: Objekt erzeugen (2a) und stimmig initialisieren (2b)




```
/*---------------Methoden------*/
public void reservieren() {
    if (zustand.equals("verfuegbar")) {
        zustand = "reserviert";
        aktualisieren();
    }
}

public void reservierungAufheben() {
    if (zustand.equals("reserviert")) {
        zustand = "verfuegbar";
        aktualisieren();
    }
}

public void bezahlenDurchUeberweisung() {
    if (zustand.equals("reserviert")) {
        zustand = "bezahlt";
        aktualisieren();
    }
}
```

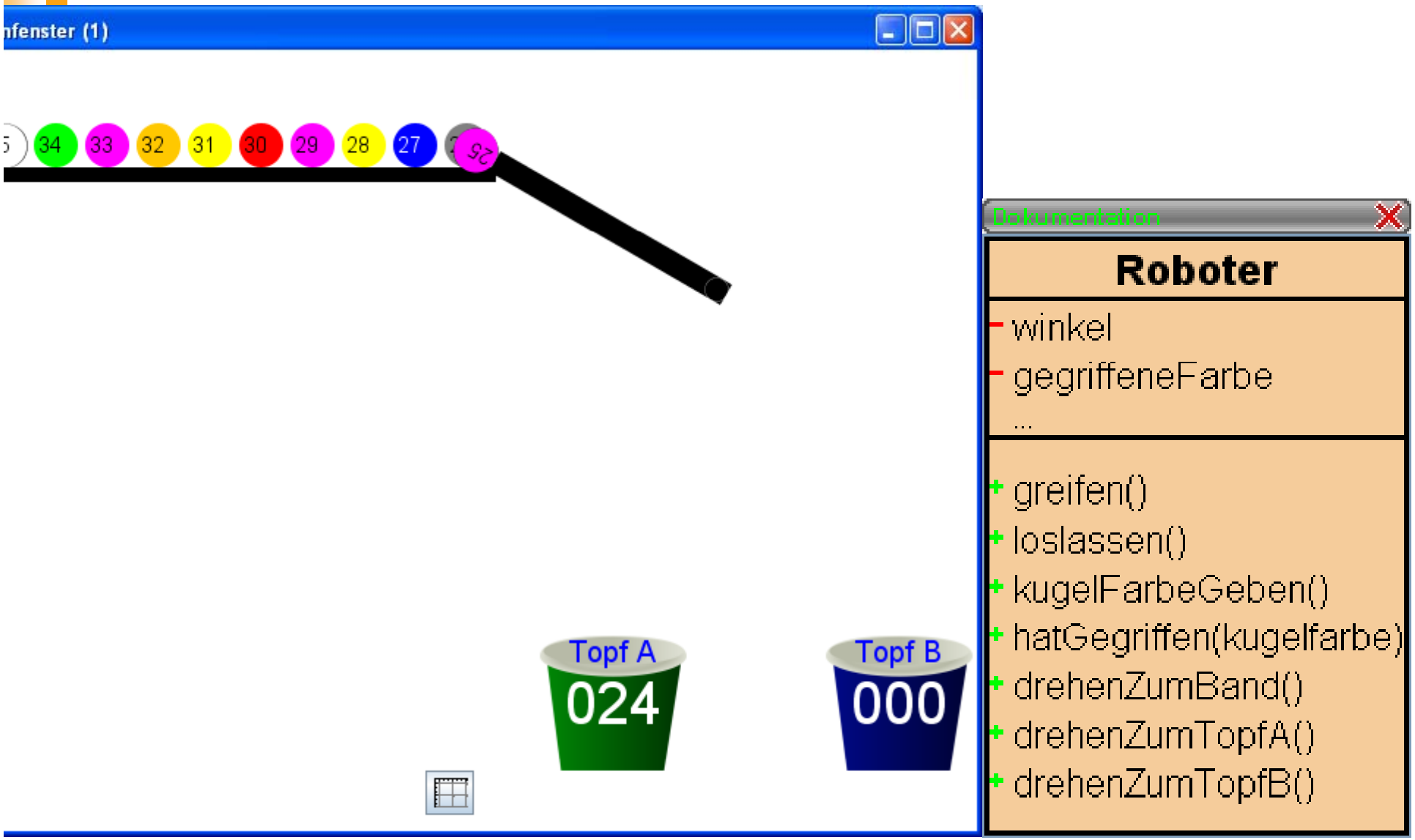
3.Schritt: Überprüfen aller verändernden Methoden.
Ggf. passende Zustandsänderung des Objekts darstellung anstoßen.

u.s.w.

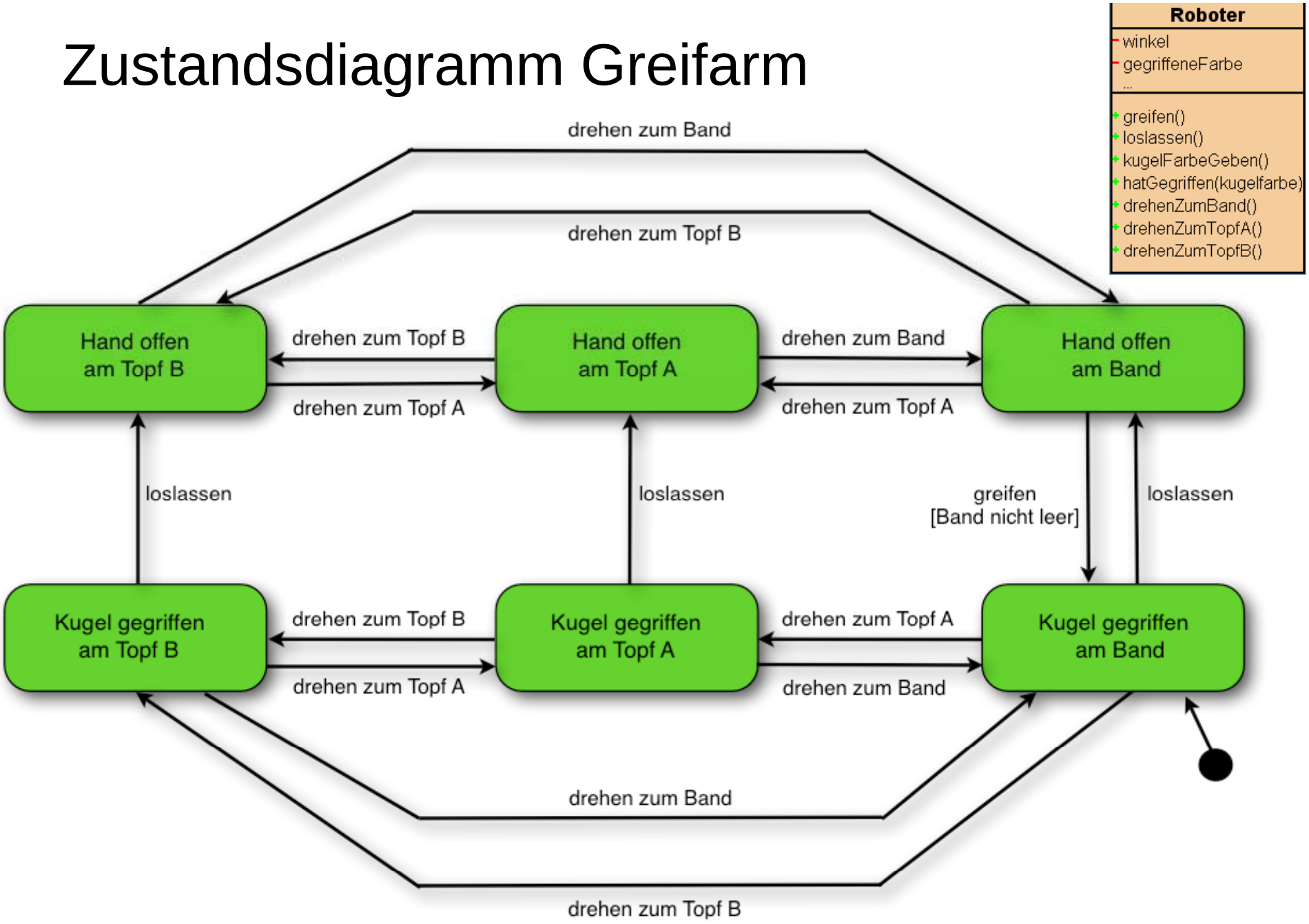
Abschnitt 3

- Algorithmen (nur ein kurzer Blick auf ein mögliches Werkzeug)

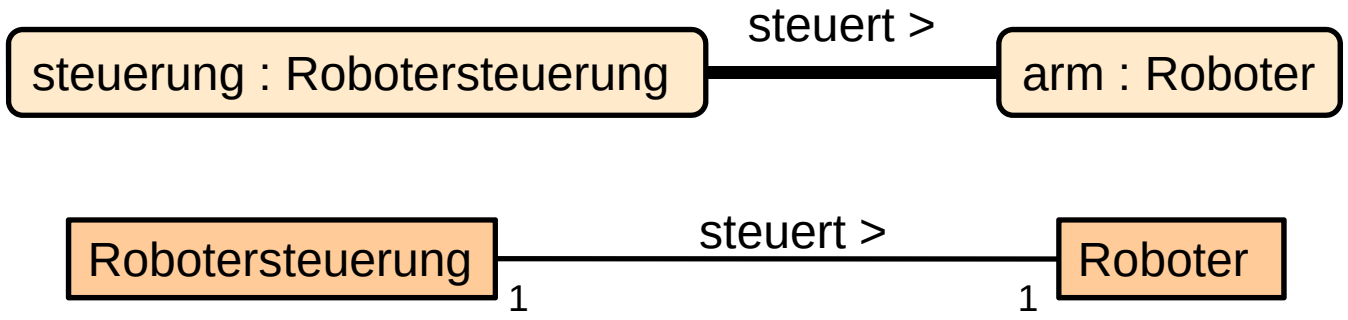
Beispiel Steuerung eines Greifarmroboters



Zustandsdiagramm Greifarm



Steuerung des Greifarms



- Entwerfen und Programmieren einer Robotersteuerung

  Lösungsvorschläge siehe
  `beispiele.k1k3Roboter.Robotersteuerung`

Abschnitt 4

- Beziehungen zwischen Objekten

Beziehungen

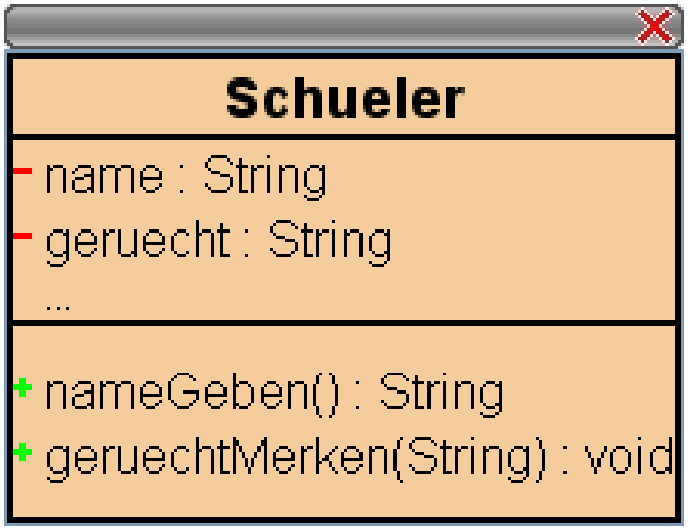
- Aufgabe Schüler und Gerüchte
- I) Viele Schüler, Gerüchte und eine „Strichliste“, die jeder Schüler kennt.
  Dort trägt sich jeder Schüler ein.
  Variante:
  Es gibt einen „Schiedsrichter“ der die Strichliste führt und immer aktualisiert?
  Wann muss aktualisiert werden?
- II) …jeder Schüler darf **höchstens** zwei Mitschüler auswählen, mit denen er eine fiktive Gerüchtebeziehung eingeht um ihnen bei Bedarf ein Gerücht weiterzuerzählen (die Richtung der Gerüchtebeziehung soll eine Rolle spielen). …

Anzahl der Schüler im Spiel

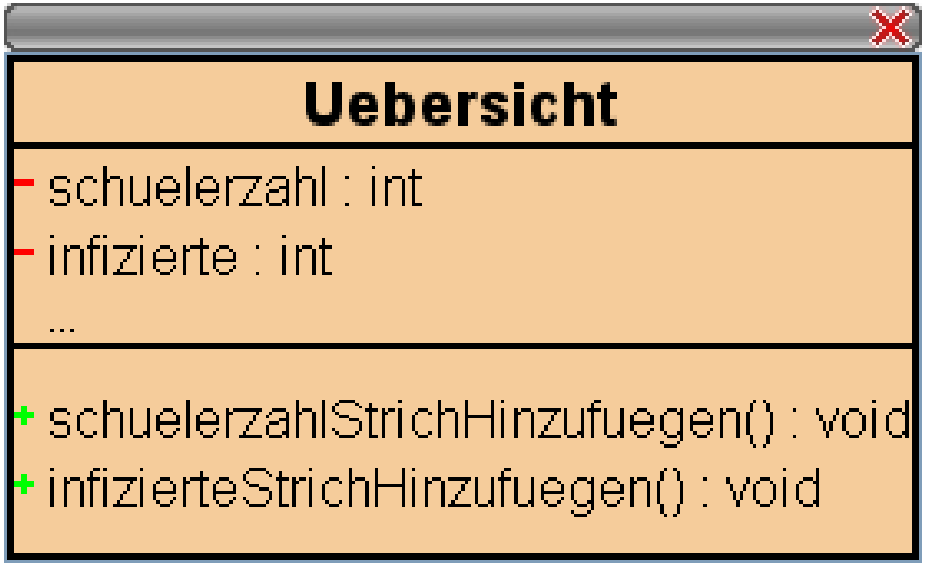
momentane Anzahl der Schüler, die ein Gerücht kennen

Klasse Schüler, Klasse Uebersicht

- Welche Dienste sollten angeboten werden? Welche sind interessant?



typischerweise auch
geruechtGeben() : String

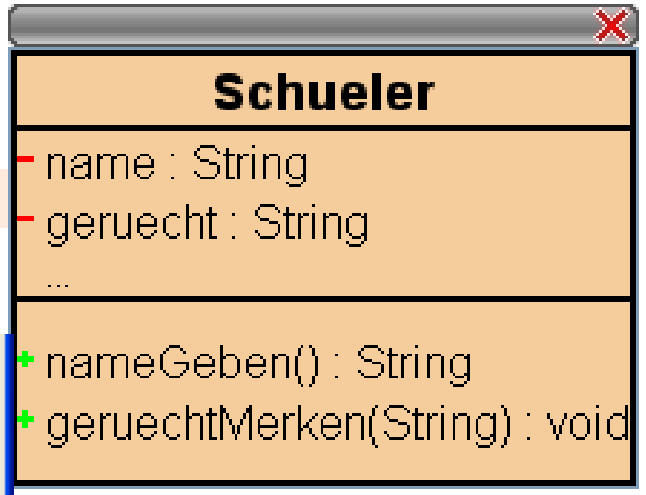


typischerweise auch
schuelerzahlGeben() : int
infizierteGeben() : int

im Folgenden nicht gezeigt / abgebildet

Klasse Schueler umsetzen

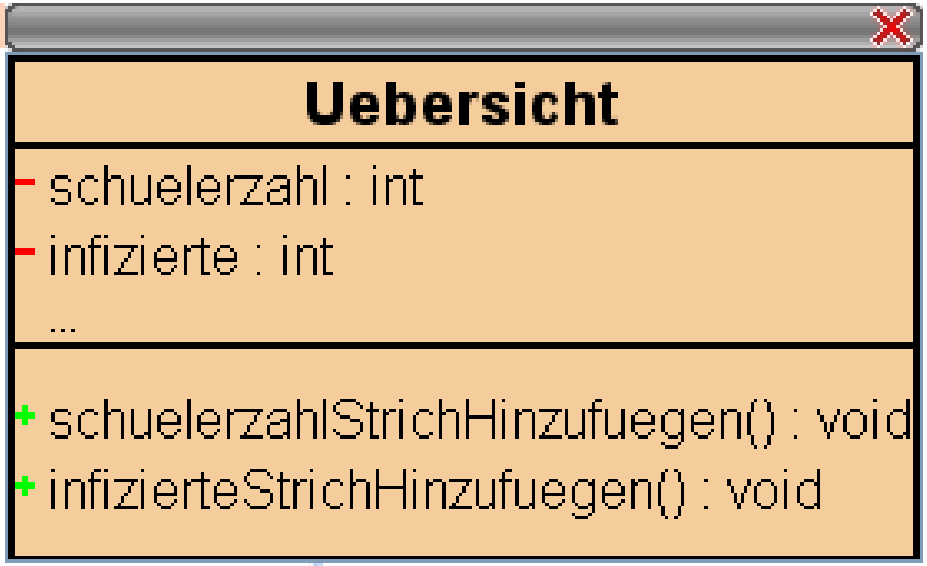
```
public class Schueler {
    /*---------------Attribute-----*/
    private String name;
    private String geruecht;

    /*---------------Konstruktor---*/
    public Schueler(String neuerName) {
        name = neuerName;
    }

    /*---------------Methoden------*/
    public void geruechtMerken(String neuesGeruecht) {
        geruecht = neuesGeruecht;
    }

    public String nameGeben() {
        return name;
    }
```



…altbewährt…

Klasse Uebersicht umsetzen

```
public class Uebersicht {
    /*---------------Attribute-----*/
    private int schuelerzahl;
    private int infizierte;

    /*---------------Konstruktor---*/
    public Uebersicht() {
    }

    /*---------------Methoden------*/
    public void schuelerzahlStrichHinzufuegen() {
        schuelerzahl++;
    }

    public void infizierteStrichHinzufuegen() {
        infizierte++;
    }
}
```

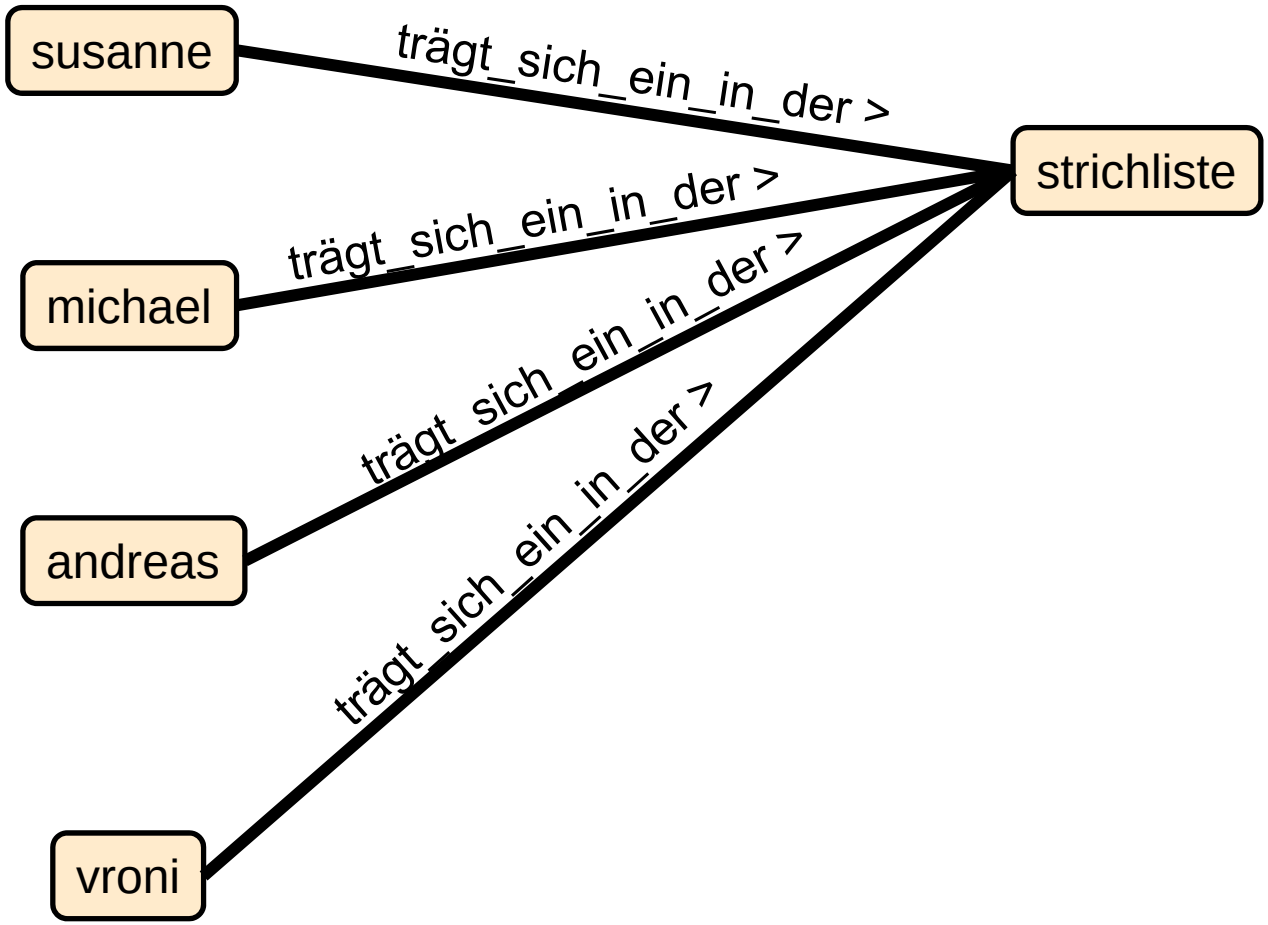


…altbewährt…

…und anschließend mit Objekten dieser selbstdefinierten Klassen experimentieren, Dienste nutzen …

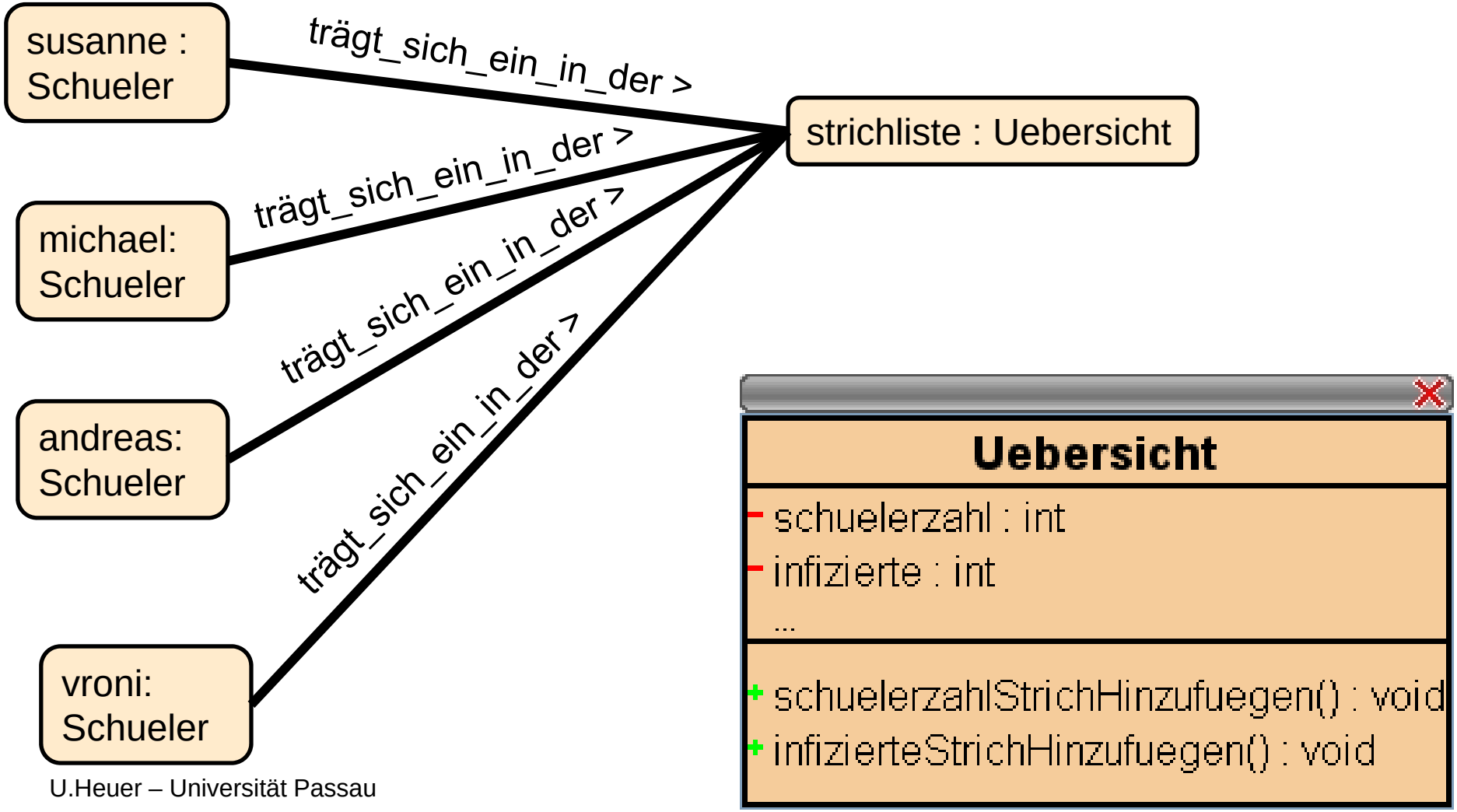
Wiederholung/Vertiefung aus Klasse 9

- Objektdiagramme stellen bestehende Beziehungen zwischen Objekten in einer bestimmten Situation dar



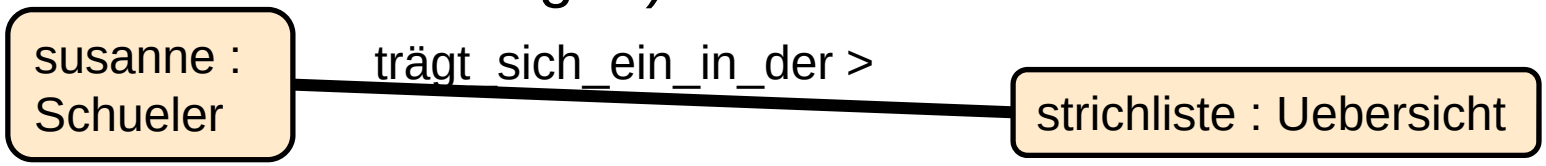
Wiederholung/Vertiefung aus Klasse 9

- Der Pfeil an der Verbindungslinie deutet an, welches Objekt Dienste des anderen Objekts nutzen kann.



Wiederholung/Vertiefung aus Klasse 9

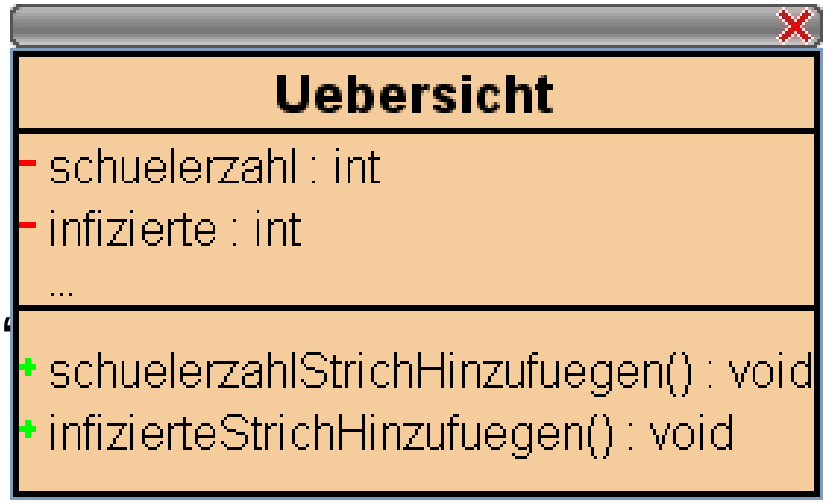
- Die Schülerin Susanne kann beispielsweise den Zustand der Strichliste verändern (einen Strich in einer passenden Rubrik hinzufügen).



Susanne kennt die Strichliste (sie pflegt eine Beziehung zur Strichliste).

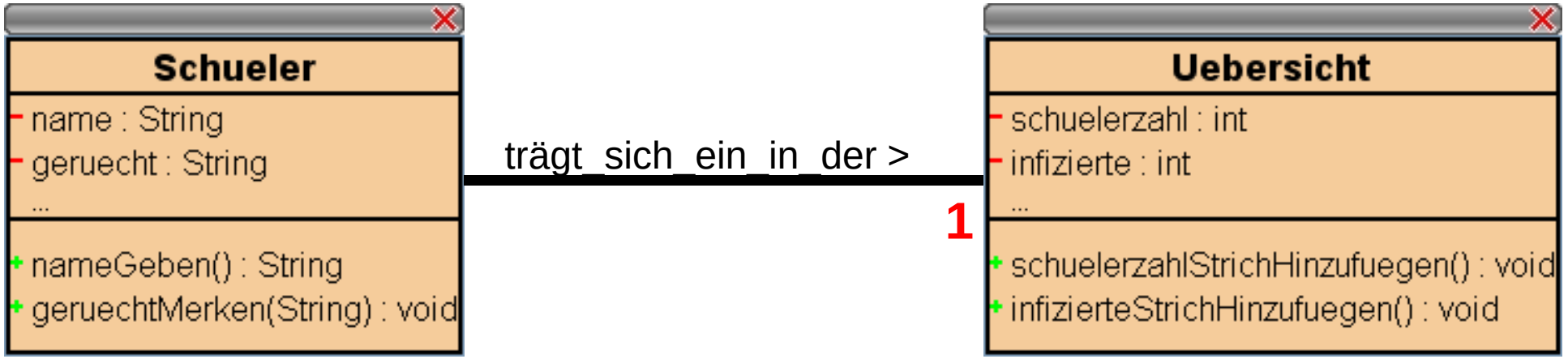
Die Strichliste bietet die Dienste xxxStrichHinzufuegen() an.

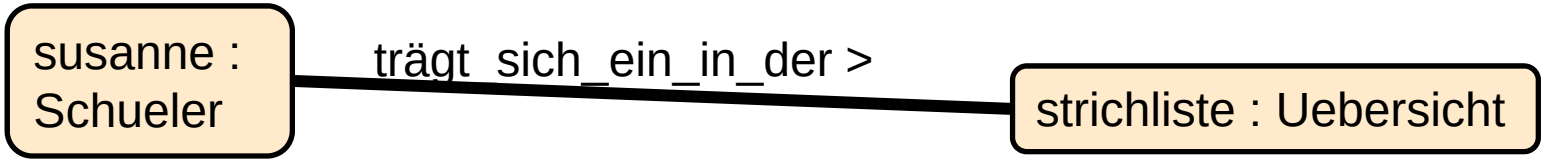
Susanne wird diese Dienste nutzen, um ihren Teil dazu beizutragen, dass die Strichliste „immer aktuell“ ist.



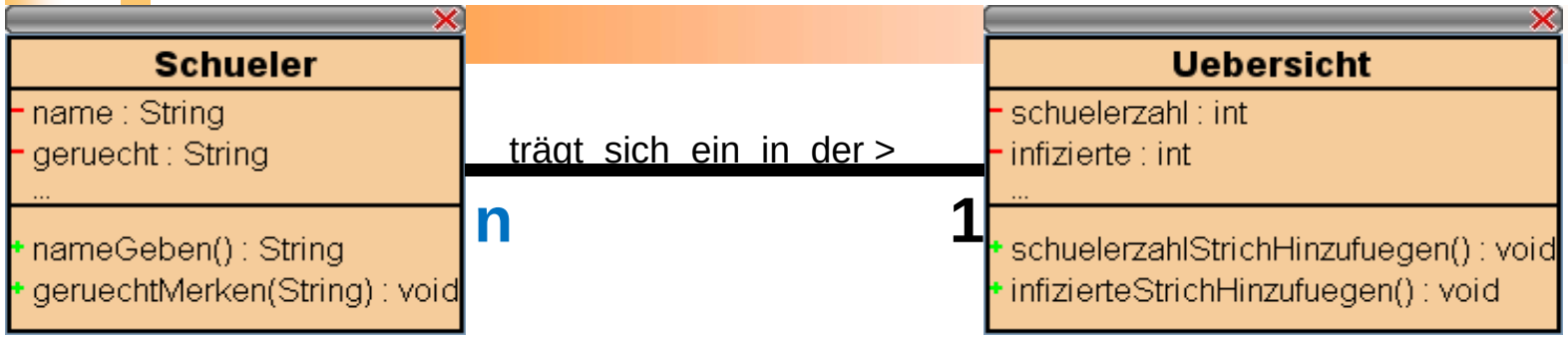
Wiederholung/Vertiefung aus Klasse 9

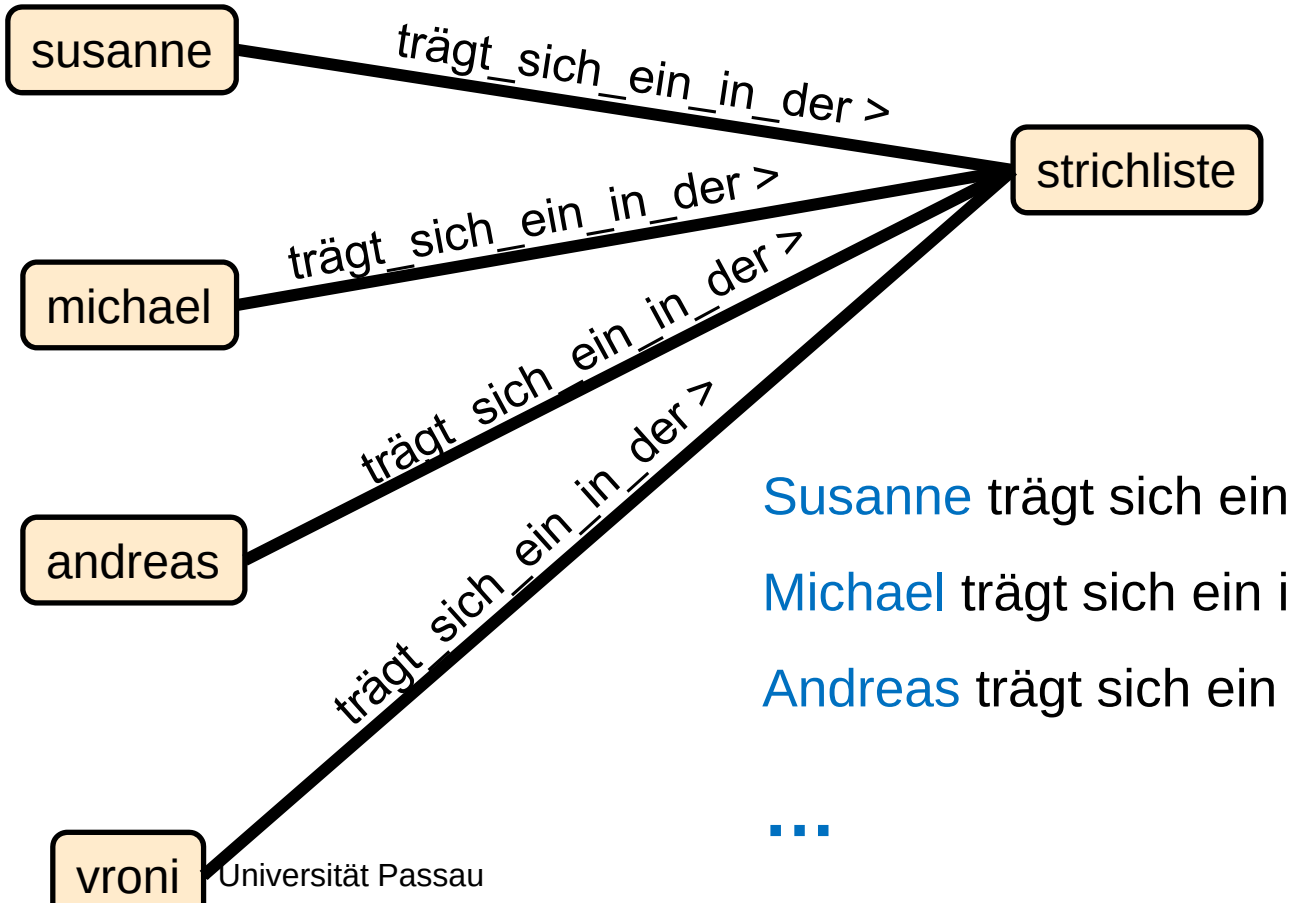
Ein Schülerobjekt trägt sich bei Bedarf in einem Uebersichtsobjekt ein.





Wiederholung/Vertiefung aus Klasse 9





Susanne trägt sich ein in der Strichliste

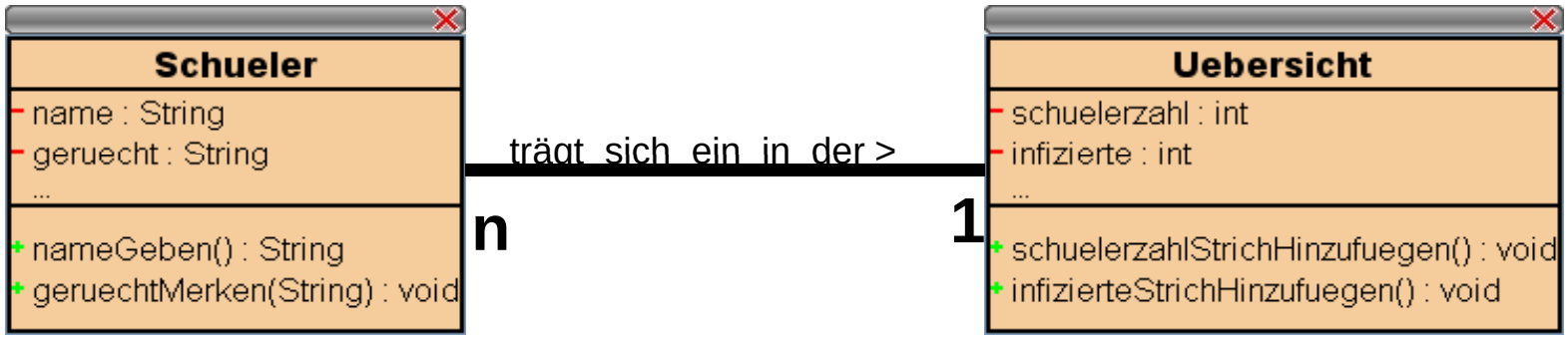
Michael trägt sich ein in der Strichliste

Andreas trägt sich ein in der Strichliste

...

Beziehungen zwischen Klassen

- Im Klassendiagramm geht es darum, zu zeigen, welche Beziehungen die Objekte der Klassen prinzipiell pflegen können.
- Das Klassendiagramm besteht aus Klassenkarten und deren Beziehungen.
  Die Beziehungen werden mit Namen, Pfeil in Nutzungsrichtung und Kardinalität versehen.



Referenz

Aus Klasse 7

- Beziehungen zwischen Begriffen im Lexikon (Verweis auf Begriffe, die an anderer Stelle erklärt werden.)
- Beziehungen zwischen Dokumenten (Hyperstrukturen)

In Klasse 10

- Beziehungen zwischen Objekten

Eine Referenz ist ein Verweis von einem Objekt auf ein anderes (Ziel-)objekt.

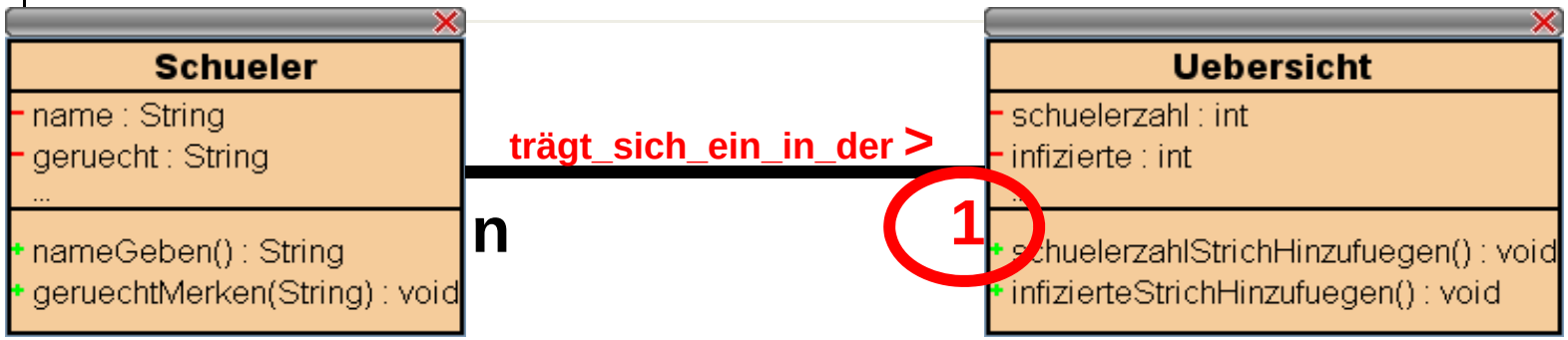
nettes Beispiel: Preisliste eines Computerhändlers im Netz – Kunden speichern sich einen Verweis auf diese Liste und nicht die Liste selbst. Warum?

→ Umsetzung einer Referenz in der Klasse Schueler in drei Schritten

Schritt 1 : ein Referenzattribut deklarieren

„Jeder Schüler muss die Übersicht kennen“

```
public class Schueler {
    /*---------------Attribute-----*/
    private String name;
    private String geruecht;
    private Uebersicht strichliste;
```



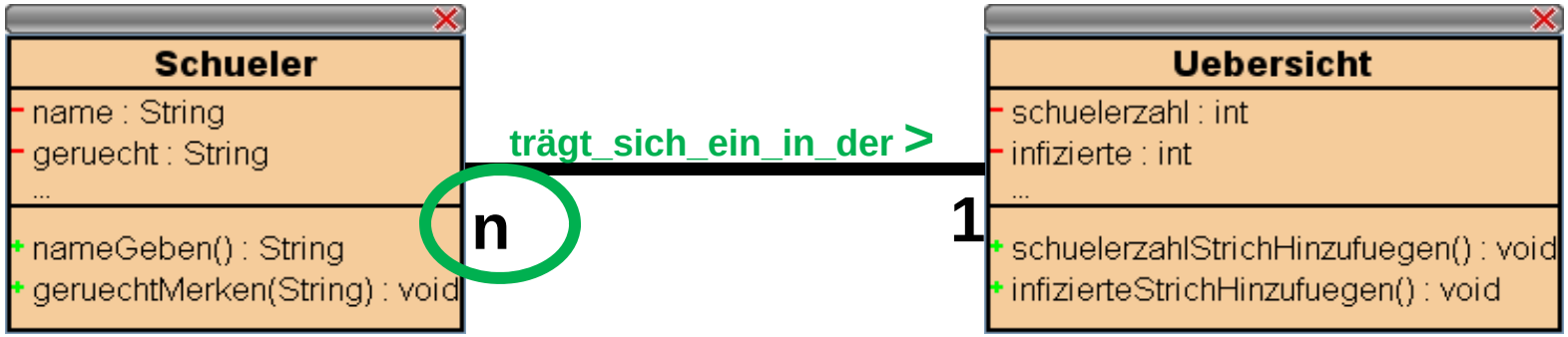
Demonstration: … Attribute inspizieren… „statt einem Pfeil ist bei der strichliste nur null eingetragen!“ …

Schritt 2 : Zuweisung eines (Ziel-)Objekts

```
public class Schueler {
    /*---------------Attribute-----*/
    private String name;
    private String geruecht;
    private Uebersicht strichliste;

    /*---------------Konstruktor---*/
    public Schueler(String neuerName, Uebersicht strichlistefuerAlle) {
        name = neuerName;
        strichliste = strichlistefuerAlle;
        strichliste.schuelerzahlStrichHinzufuegen();
    }
```

Alle Schülerobjekte können eine Referenz auf das eine Uebersichtsobjekt erhalten.

...



Schritt 2 : Zuweisung eines (Ziel-)Objekts

```
public class Schueler {
    /*---------------Attribute-----*/
    private String name;
    private String geruecht;
    private Uebersicht strichliste;

    /*---------------Konstruktor---*/
    public Schueler(String neuerName, Uebersicht strichlistefuerAlle) {
        name = neuerName;
        strichliste = strichlistefuerAlle;
        strichliste.schuelerzahlStrichHinzufuegen();
    }
```

...

Der Konstruktor kann genutzt werden, um die Beziehung herzustellen.

Die Zuweisung des (Ziel-)Objekts erfolgt syntaktisch wie die Zuweisung eines Attributwerts.

Das Referenzattribut beinhaltet nicht das Zielobjekt selbst sondern lediglich einen Verweis darauf. Ändert sich der Zustand des Zielobjekts, haben alle anderen Objekte, die eine Referenz darauf besitzen, sofort Zugang zum veränderten Zustand.

Schritt 3: Prüfung der Definition des Konstruktors und…

```
public class Schueler {
    /*---------------Attribute-----*/
    private String name;
    private String geruecht;
    private Uebersicht strichliste;

    /*---------------Konstruktor---*/
    public Schueler(String neuerName, Uebersicht strichlistefuerAlle) {
        name = neuerName;
        strichliste = strichlistefuerAlle;
        strichliste.schuelerzahlStrichHinzufuegen();
    }
```

Prüfung:

…

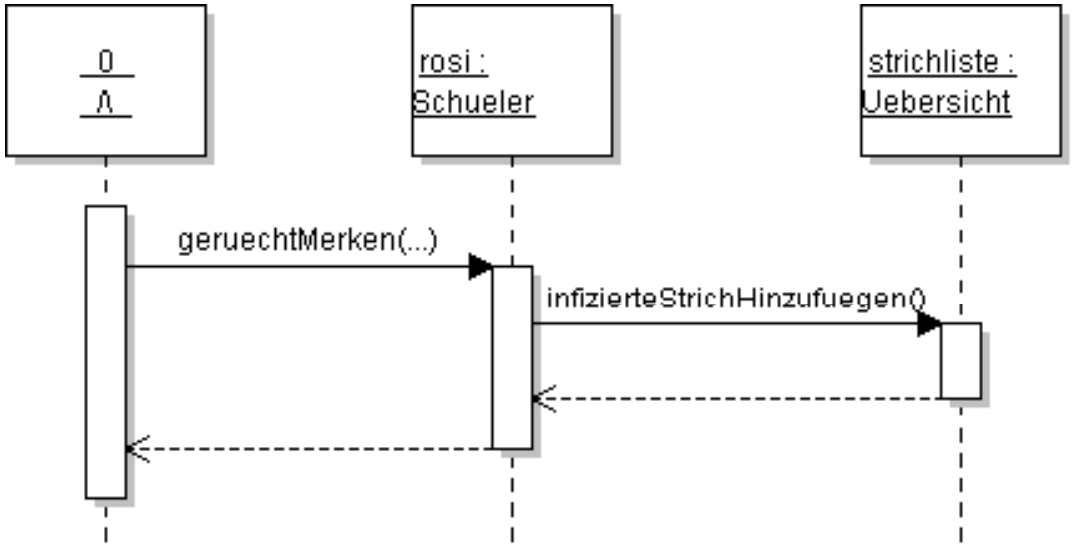
Sorgt der Konstruktor für eine stimmige Initialisierung von Schuelerobjekten?

Wenn ein neues Schülerobjekt erzeugt wird, wird „gleich“ auch dafür gesorgt, dass der Zustand der Strichliste aktualisiert wird.

Wurde nichts vergessen?

Ein Gerücht „hat sich der neue Schüler ja noch nicht gemerkt“ im Konstruktor bleibt also nichts mehr zu tun. Aber…

Schritt 3: …und Überprüfung der Definition aller Methoden der Klasse Schueler



Vgl. mit dem Einstiegsspiel

```
/*---------------Methoden------*/
public void geruechtMerken(String neuesGeruecht) {
    if (geruecht == null) {
        strichliste.infizierteStrichHinzufuegen();
    }
    geruecht = neuesGeruecht;
}
...
```

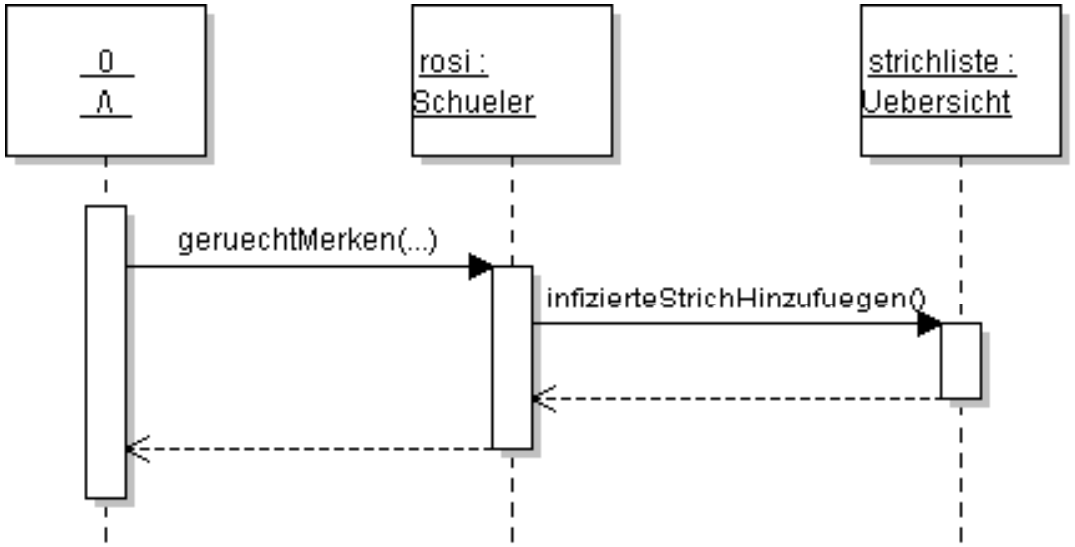
neu

„Wir waren bisher ahnungslos, sind also nun neu wissend und müssen das in der passenden Rubrik der Strichliste vermerken…“

Die Methode geruechtMerken war (und ist) eine verändernde Methode. Sie muss nun zusätzlich ggf. eine Zustandsänderung der Strichliste anstossen.

Schritt 3: …und Überprüfung der Definition aller Methoden der Klasse Schueler

Anmerkung:
Eine **Bedingung** könnte man ins Sequenzdiagramm mit aufnehmen - machen wir nicht.




```
/*---------------Methoden------*/
public void geruechtMerken(String neuesGeruecht) {
    if (geruecht == null) {
        strichliste.infizierteStrichHinzufuegen();
    }
    geruecht = neuesGeruecht;
}
```

…

Die Methode geruechtMerken war (und ist) eine verändernde Methode. Sie muss nun zusätzlich **ggf.** eine Zustandsänderung der Strichliste anstossen.

Schritt 3: …und Überprüfung der Definition aller Methoden der Klasse Schueler

DEMO

...

```
/*---------------Methoden------*/
public void geruechtMerken(String neuesGeruecht) {
    if (geruecht == null) {
        strichliste.infizierteStrichHinzufuegen();
    }
    geruecht = neuesGeruecht;
}

public String nameGeben() {
    return name;
}
```

...

Ein sondierender Dienst. Wie praktisch, da kann alles beim Alten bleiben.

Hinweis:

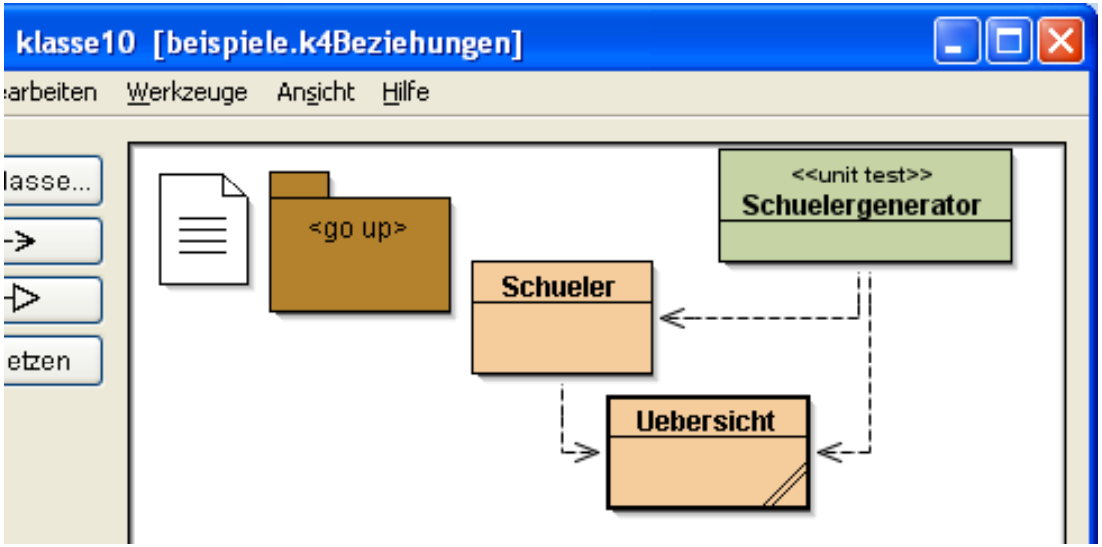
Bei der Umsetzung der Beziehung mit Nutzungsrichtung vom Schüler zur Übersicht fällt auf:

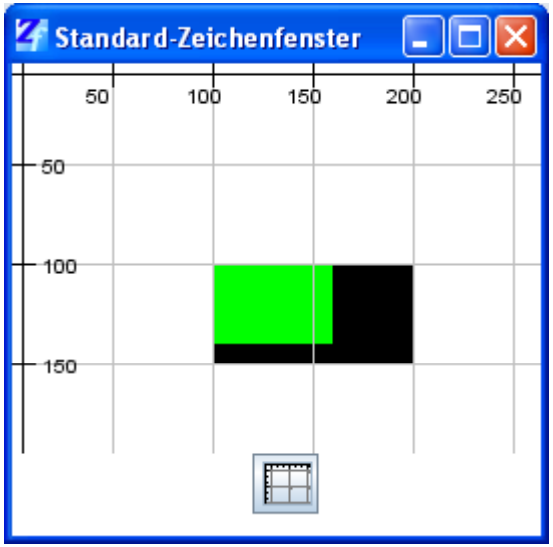
Die Klassendefinition der Klasse Uebersicht bleibt vollkommen unangetastet! Übersichtsobjekte stellen in alt bewährter Weise ihre Dienste zur Verfügung. Nun können auch Schülerobjekte diese Dienste nutzen…

Optional

DEMO

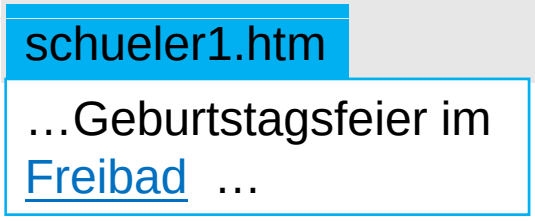
- Übersicht kann ein grünes Rechteckobjekt nutzen, um die momentane Zahl der Wissenden darzustellen. Und ein schwarzes für die Zahl der am Spiel beteiligten Schüler. (1:2 Beziehung zwischen den Klassen Uebersicht und Rechteck).
- Weiter ggf. ein Textobjekt, um den Anteil der wissenden darzustellen…





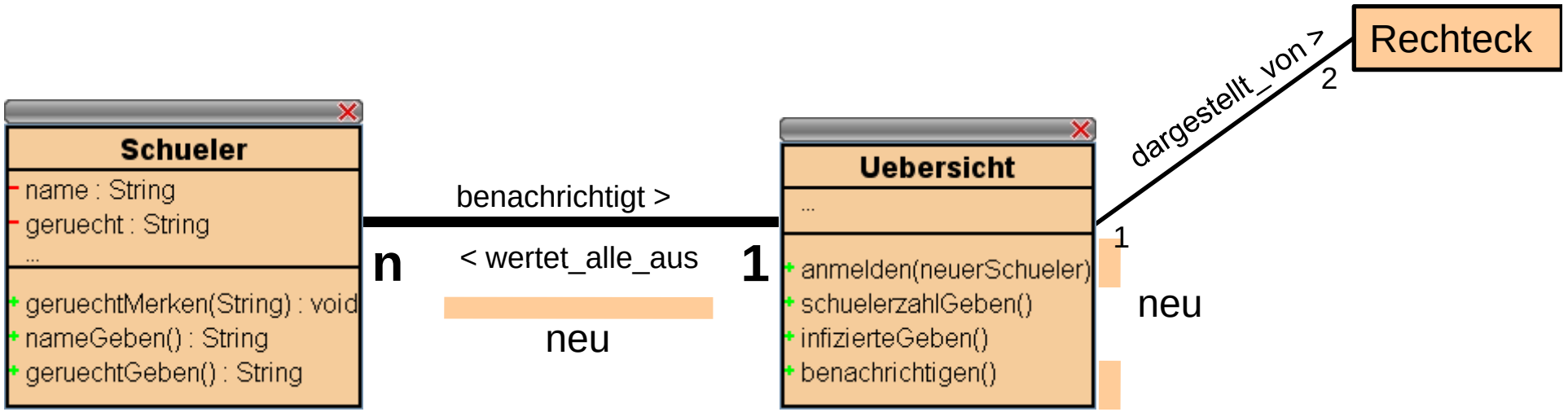
Vergleich: Eine Beziehung herstellen zwischen zwei

	Dokumenten - Inf 7	Objekten – Inf 10
	Im **Ziel**dokument befindet sich ein Objekt der Klasse Verweisziel (auch Anker genannt), dieses hat einen eindeutigen Namen im Namensraum.	Ein **Ziel**objekt wurde erzeugt (im Bsp. ein Objekt der Klasse Uebersicht) und dieses hat einen eindeutigen Namen im Namensraum . (Dass nicht immer ein Name nötig ist thematisieren wir nicht.)
	In das **aktuelle** Dokument einen Verweis (auch Link genannt) einfügen.	Dem **aktuellen** Objekt (im Bsp. ein Schülerobjekt) eine Referenz auf die Strichliste übergeben. (Im Bsp. schon während der Erzeugung des Schülerobjekts)



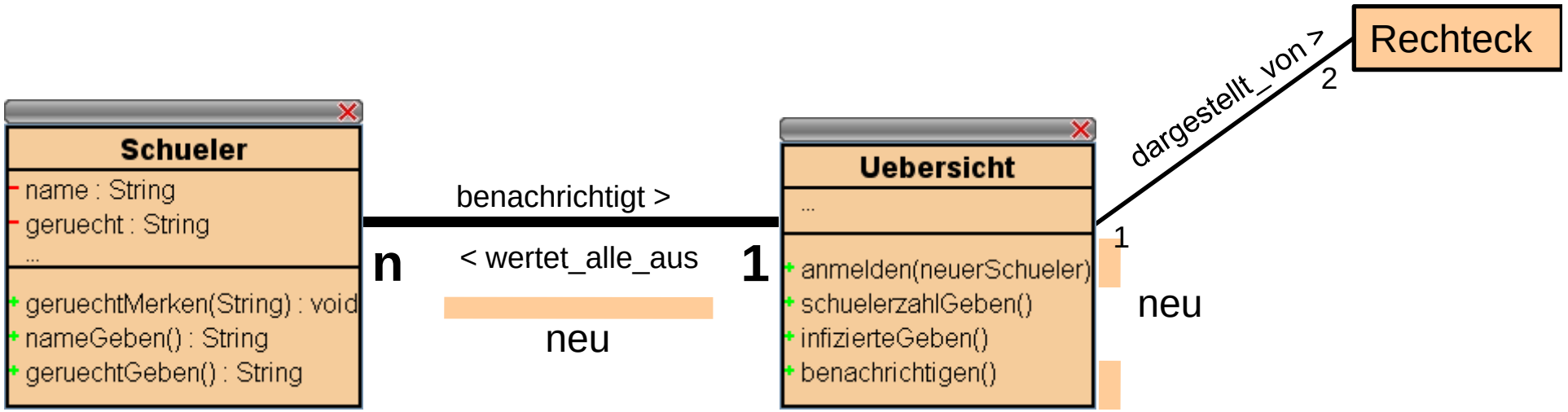
Hinweis

- Umsetzung einer 1:n Beziehung mit Nutzungsrichtung zur n-Seite zeige ich aus Zeitgründen nicht. Eine mögliche Umsetzung finden Sie unter **`beispiele.k4BeziehungenPlus`**.
- Anmerkung: Wir lassen Schueler vor solchen Umsetzung mit Behaelterobjekten experimentieren, der Art wie es z.B. im entsprechenden Lehrbuch auf S.16 in der Aufgabe 1.17 beschrieben ist.



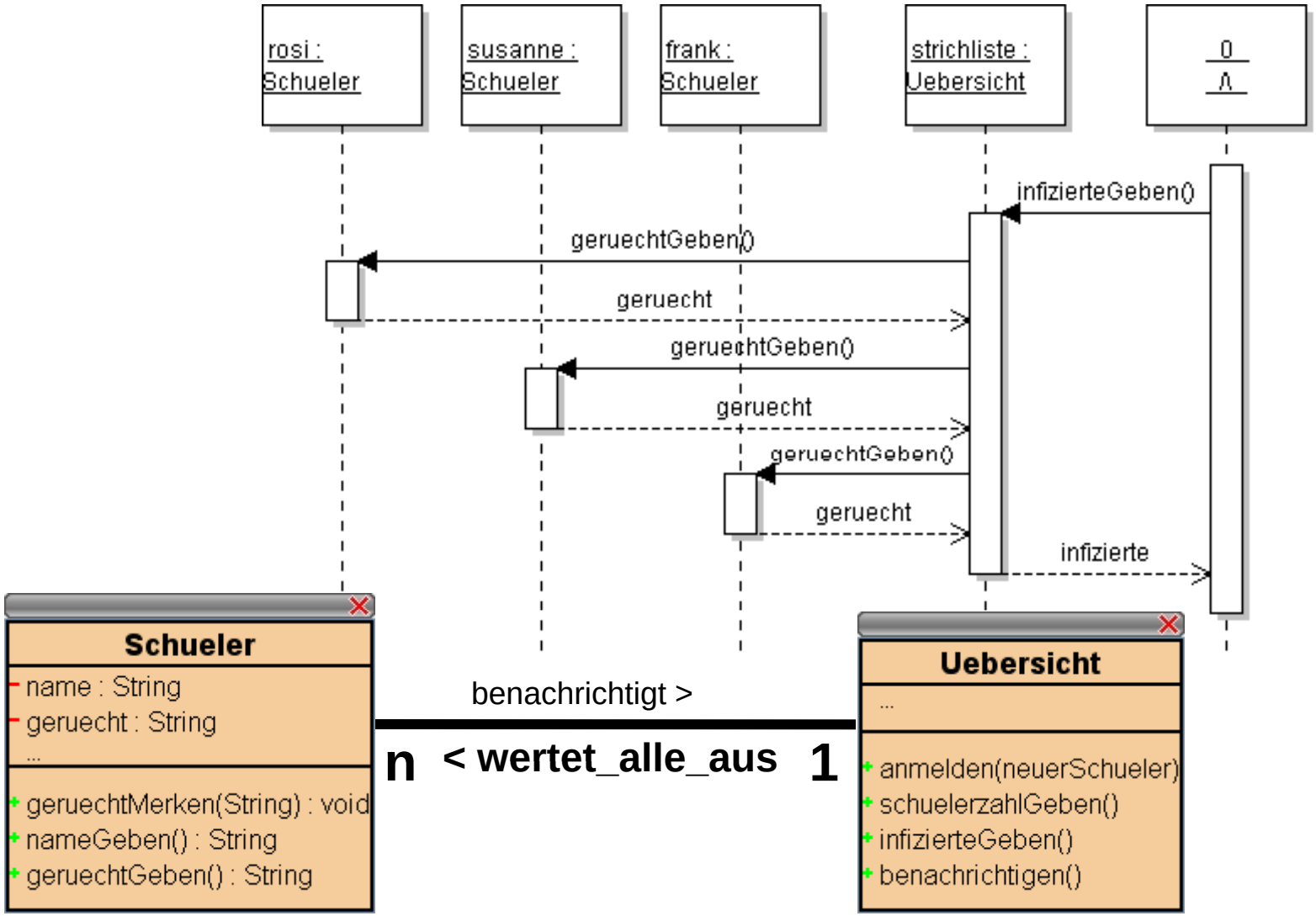
beispiele.k4BeziehungenPlus

- Die Funktionalität hat sich im Vergleich zu k4Beziehungen **nicht** geändert!
- Beim Entwurf von k4BeziehungenPlus wurde jedoch schon daran gedacht, dass später vielleicht noch weitere "Auswertungsmöglichkeiten" gewünscht werden. Die Auswertung ist gut entkoppelt von der Klasse Schueler, die Klasse Schueler soll möglichst bleiben wie sie ist, trotz neuer „Auswertungswünsche“.
- Mögliche neue Anforderungen wären beispielsweise: Es soll ermittelt werden, wieviele Schüler dieses, wieviele jenes Gerücht kennen. Gewünscht werden könnte weiter eine alphabetische Sortierung der Schülernamen…



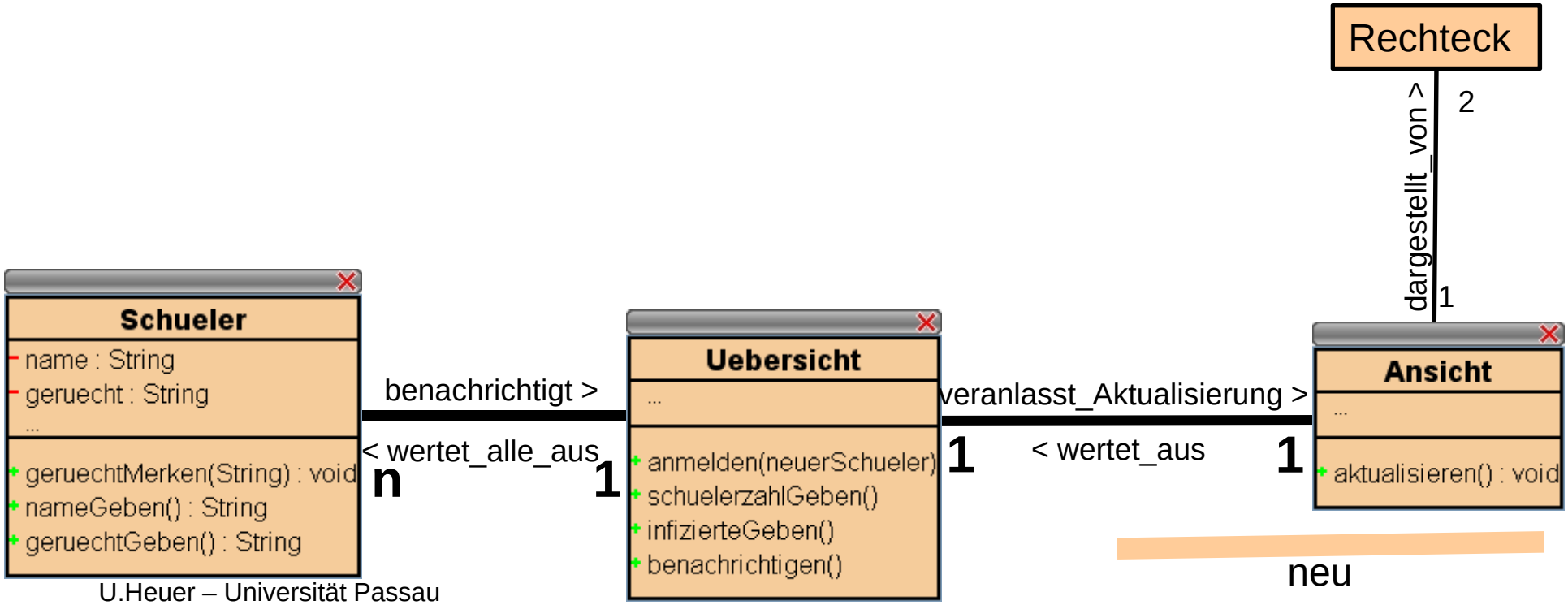
beispiele.k4BeziehungenPlus

Einblick in eine typische Kommunikation entlang der markierten Nutzungsrichtung am Beispiel des Dienstes infizierteGeben()



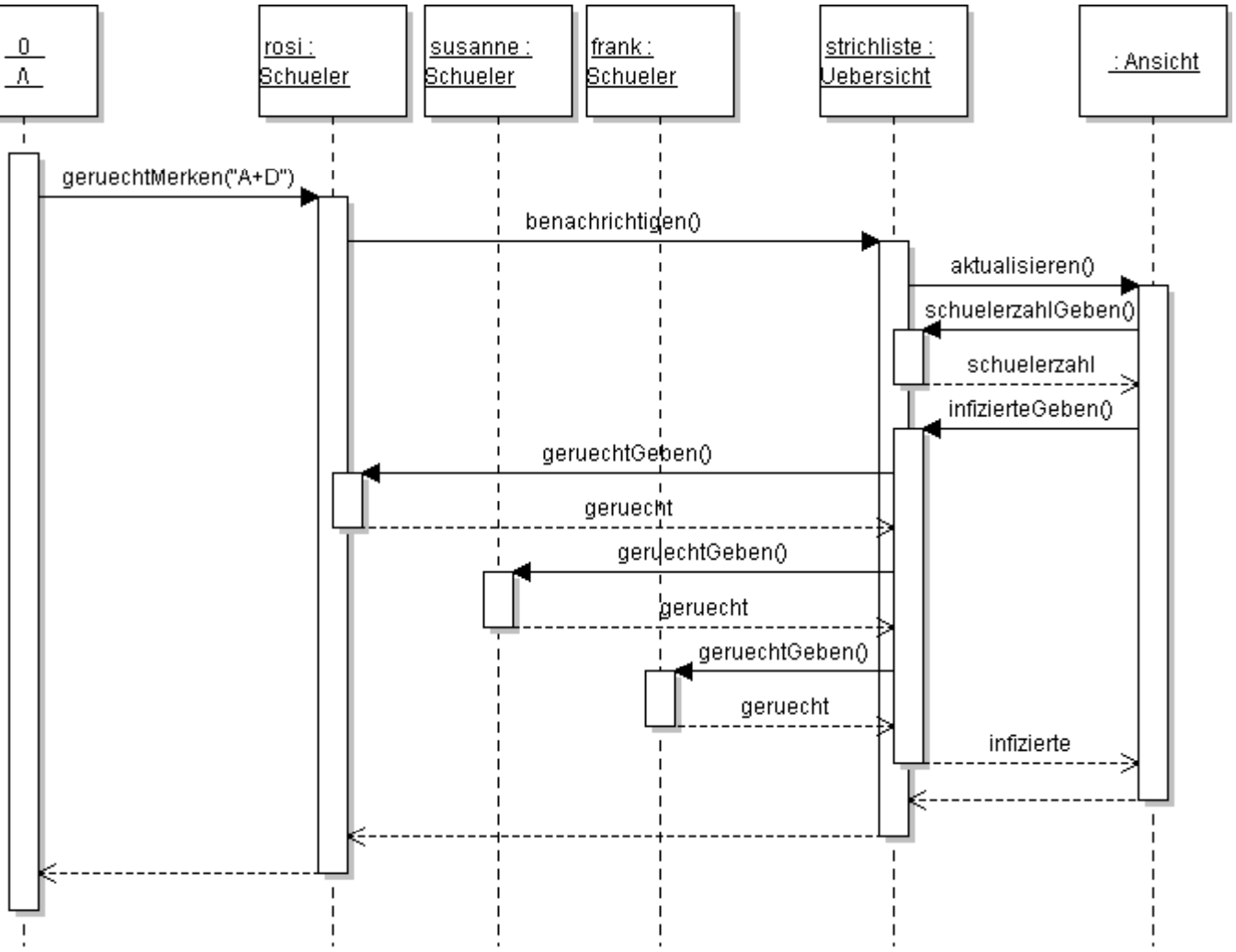
Hinweis: beispiele.k4BeziehungenPlus**Plus**

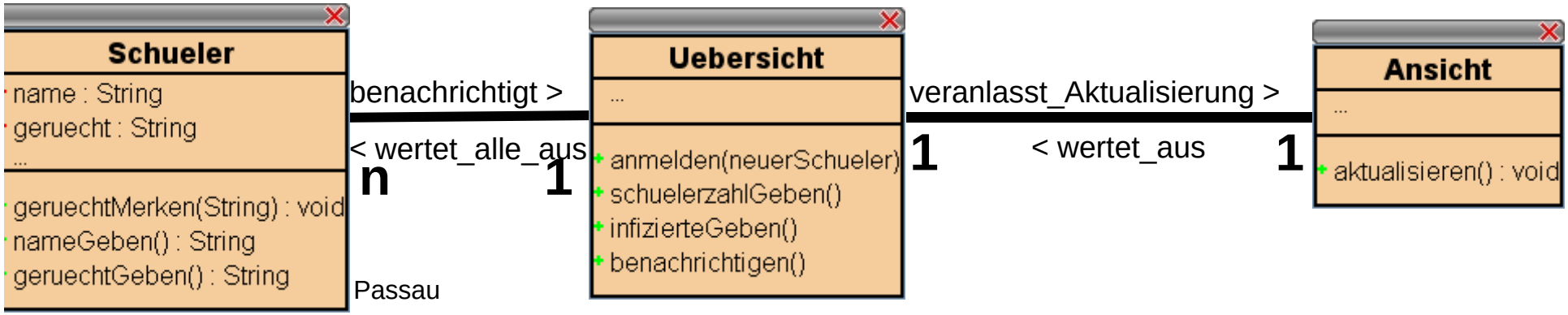
- Wiederum hat sich die Funktionalität der Umsetzung **nicht** geändert!
- Beim Entwurf von k4BeziehungenPlusPlus wurde daran gedacht, Ansicht und Modell gut zu entkoppeln um später leichter Änderungen an der Ansicht vornehmen zu können.



k4BeziehungenPlus**Plus**

Einblick in eine typische Kommunikation entlang der angegebenen Nutzungsrichtungen am Beispiel des Dienstes geruechtMerken





Weitere Beziehungen zwischen Objekten

- Aufgabe Schüler und Gerüchte – Phasen A, B, C
- I) Viele Schüler, Gerüchte und eine „Strichliste“, die jeder Schüler kennt. Dort trägt sich jeder Schüler ein.

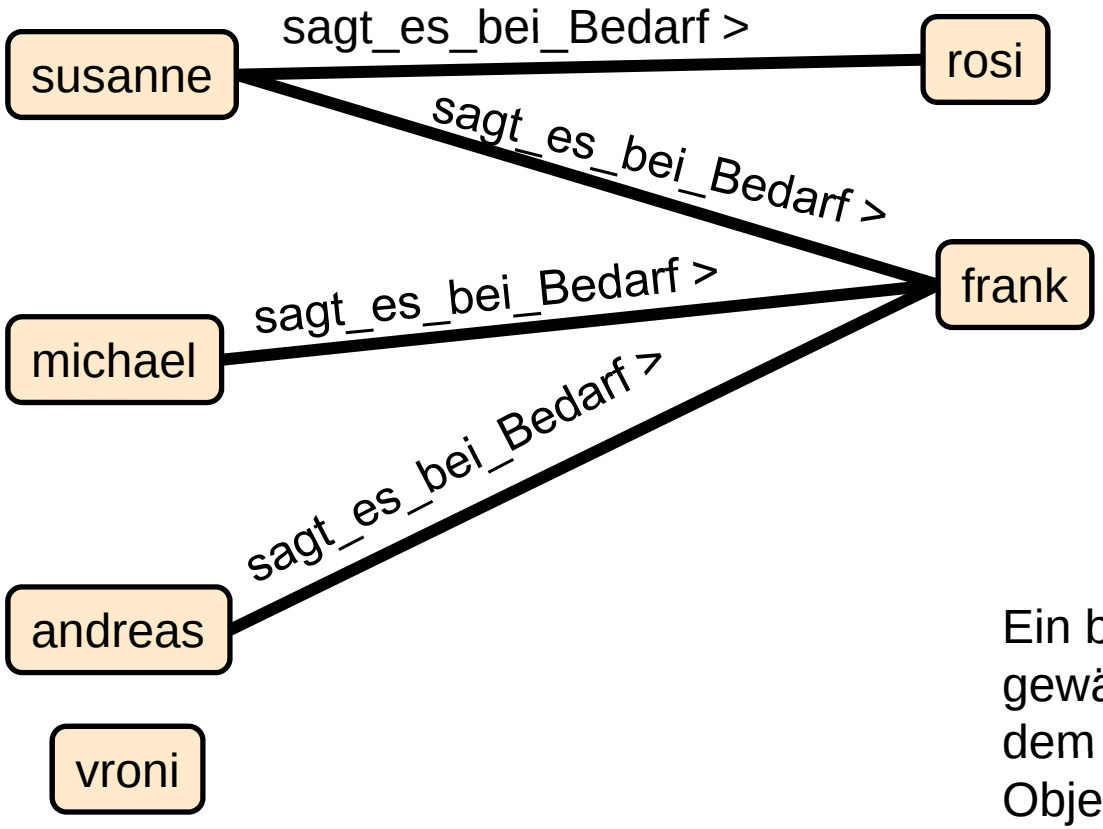
Anzahl der Schüler im Spiel

momentane Anzahl der Schüler, die ein Gerücht kennen

- II) …jeder Schüler darf **höchstens** zwei Mitschüler auswählen, mit denen er eine fiktive Gerüchtebeziehung eingeht um ihnen bei Bedarf ein Gerücht weiterzuerzählen (die Richtung der Gerüchtebeziehung soll eine Rolle spielen). …

Ausschnitt aus dem Objektdiagramm

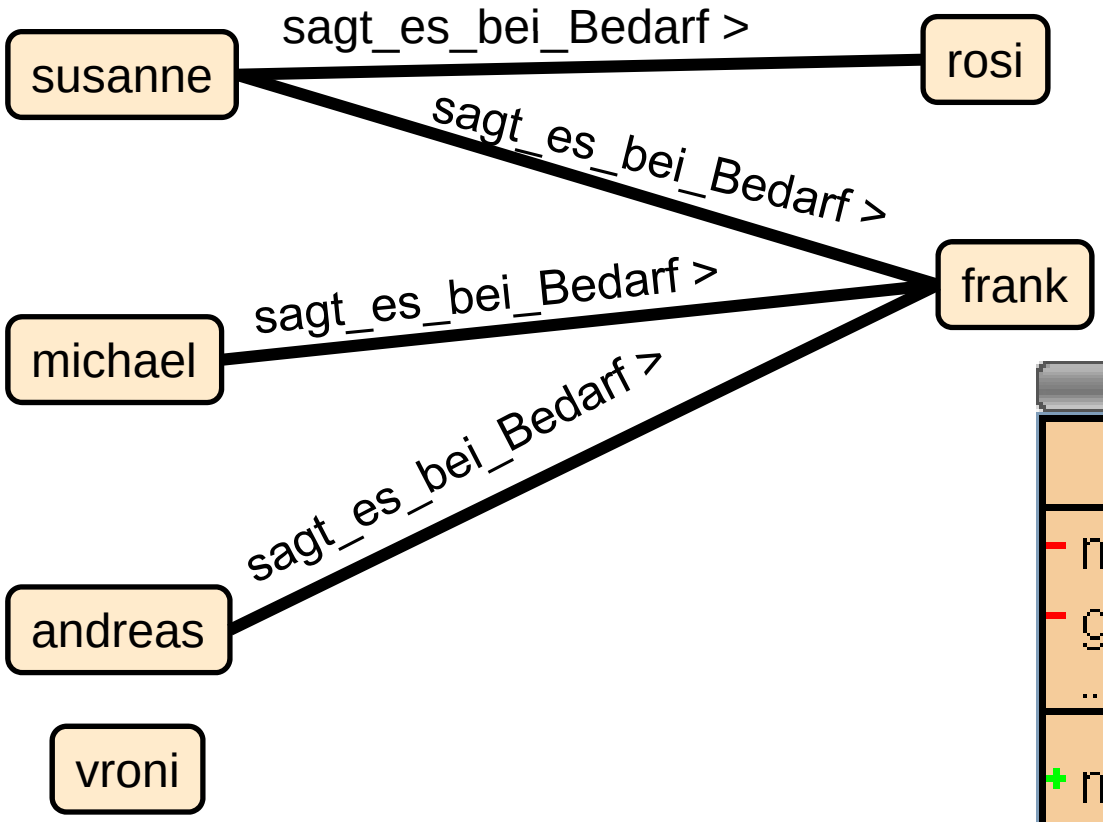
- Objektdiagramme stellen bestehende Beziehungen zwischen Objekten in einer bestimmten Situation dar

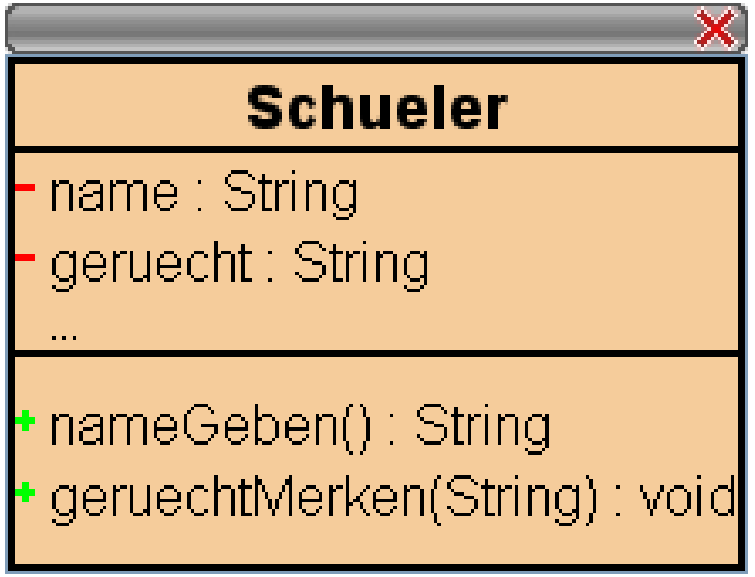


Ein bewusst einfach und geschickt gewählter kleiner Ausschnitt aus dem großen Gerüchteküche-Objektdiagramm.

Ausschnitt aus dem Objektdiagramm

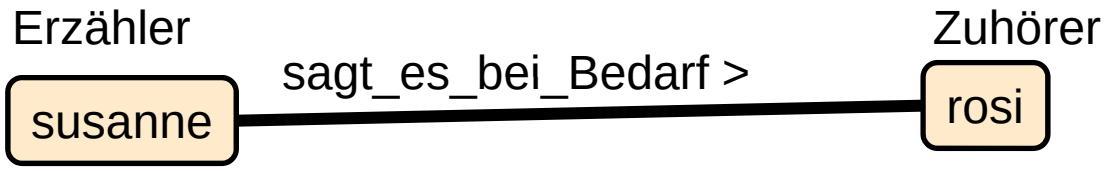
- Der Pfeil an der Verbindungslinie deutet an, welches Objekt Dienste des anderen Objekts nutzen kann.





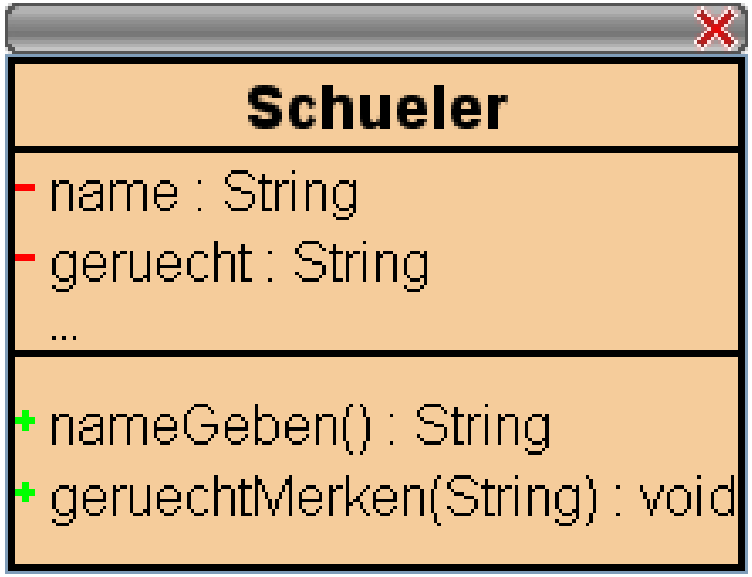
Rosis Zustand verändern

Susanne kann als Erzählerin beispielsweise Rosis Zustand verändern („von ahnungslos nach wissend“) in dem sie ihr das Gerücht erzählt.



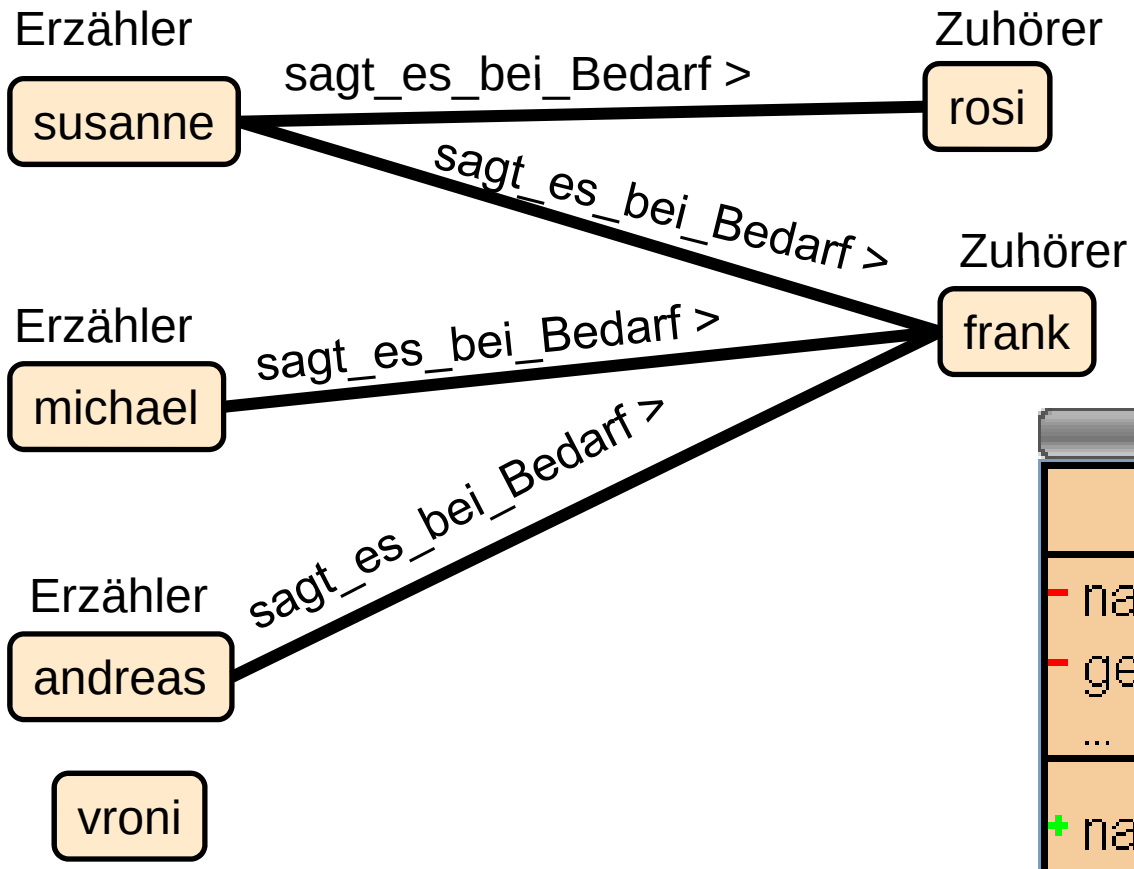
Genauer:
Rosi erlaubt als Zuhörerin der Erzählerin Susanne den Zugriff auf ihr „Gerüchtegedächtnis“. Sie bietet den Dienst geruechtMerken(…) an.

Susanne nutzt diesen Dienst.

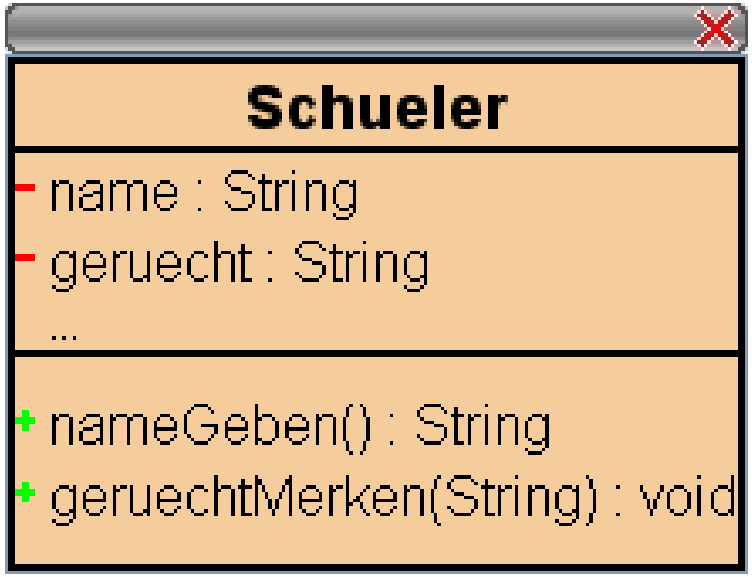


Objekte schlüpfen in Rollen

- Im Diagramm können Rollen angegeben werden, in denen sich die Objekte gerade befinden.



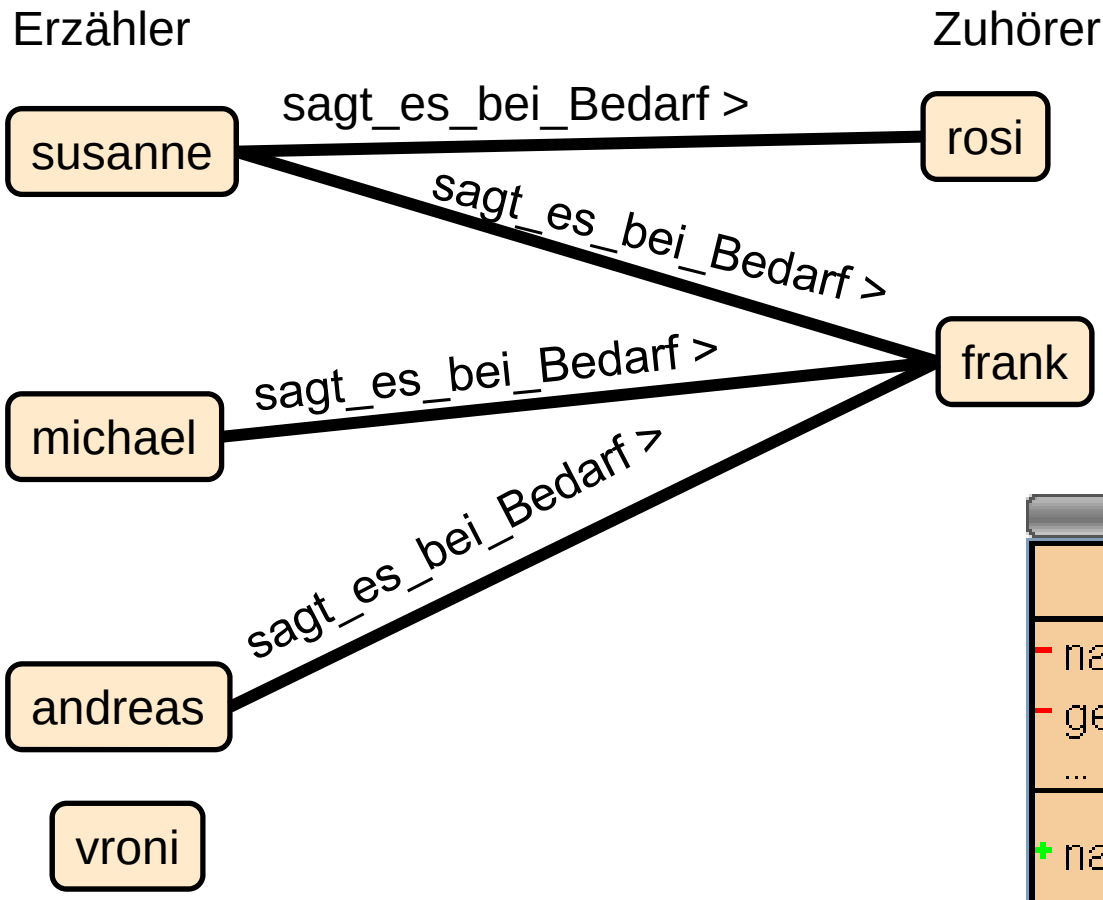
Ein Objekt kann in verschiedene Rollen „schlüpfen“, das ist hier der Übersicht halber nicht gezeichnet.

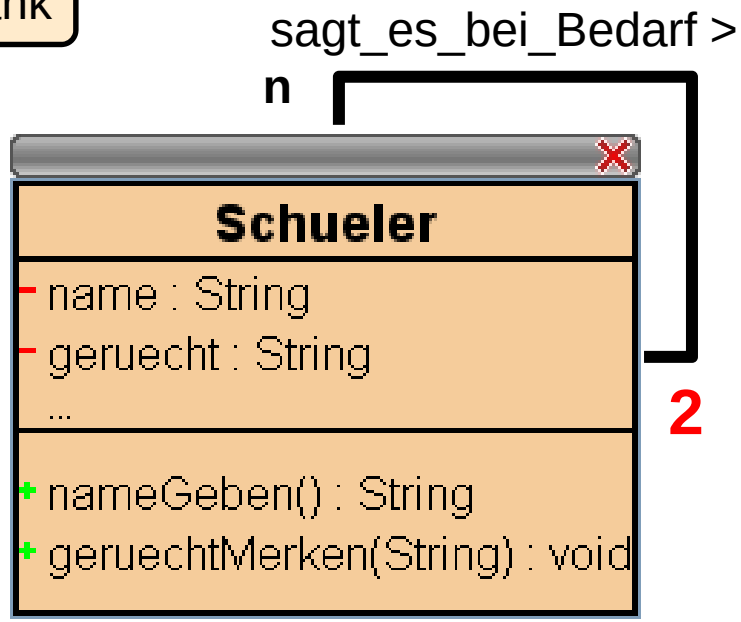


Auf dem Weg zum Klassendiagramm

Ein Schüler sagt es bei Bedarf keinem, einem oder zwei Freunden

(laut Aufgabe)





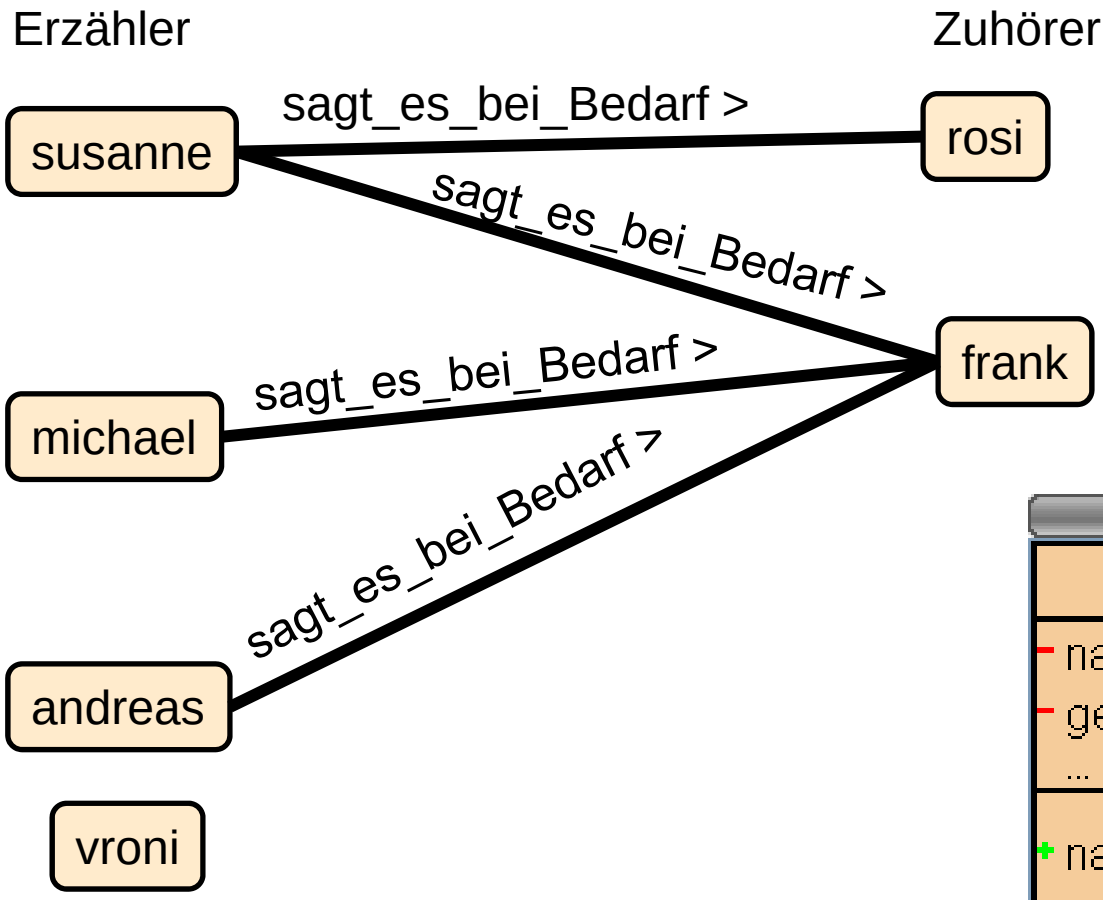
Auf dem Weg zum Klassendiagramm

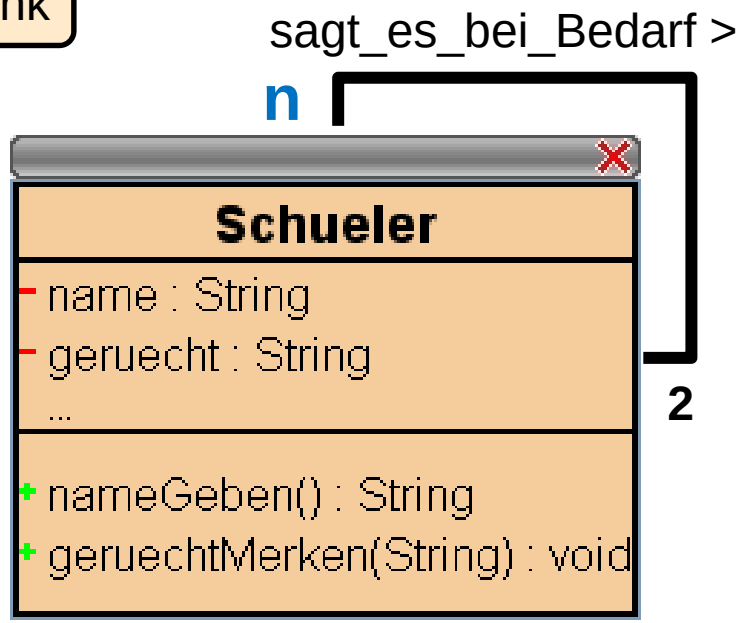
Susanne sagt es Frank

Michael sagt es Frank

Andreas sagt es Frank

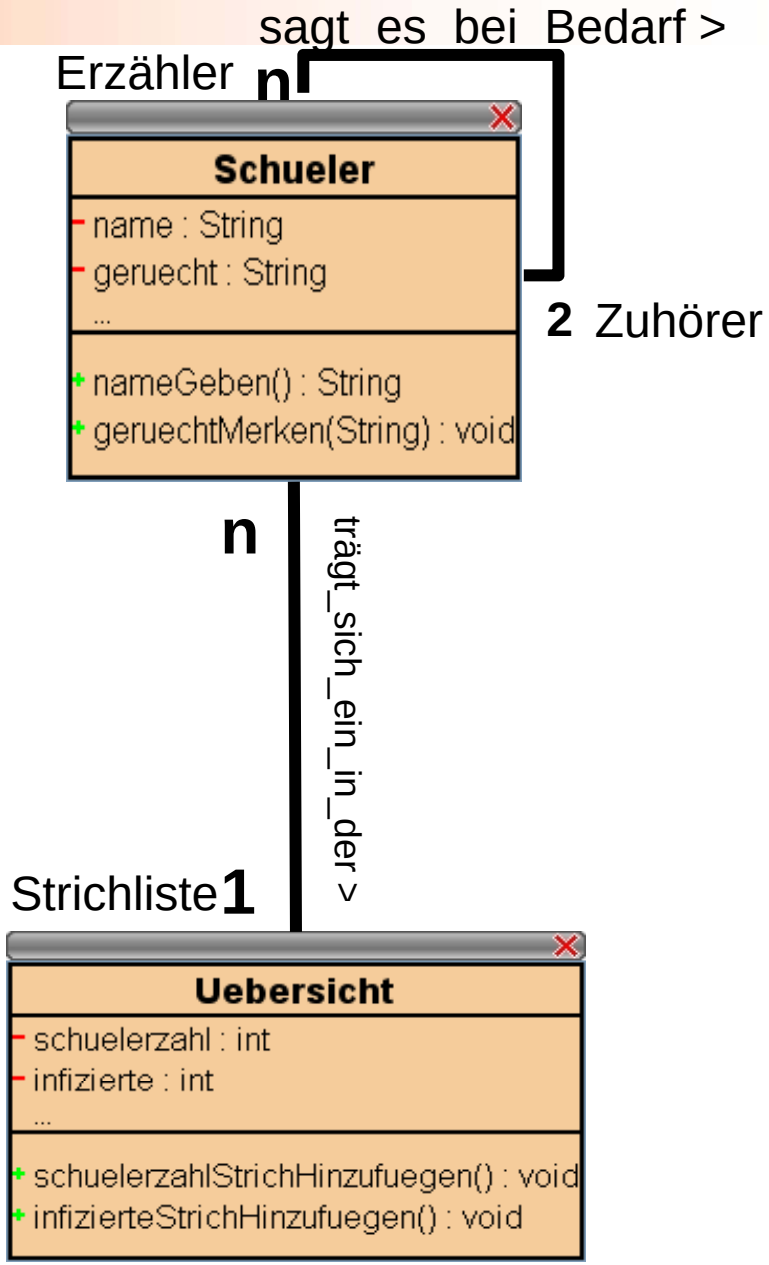
...



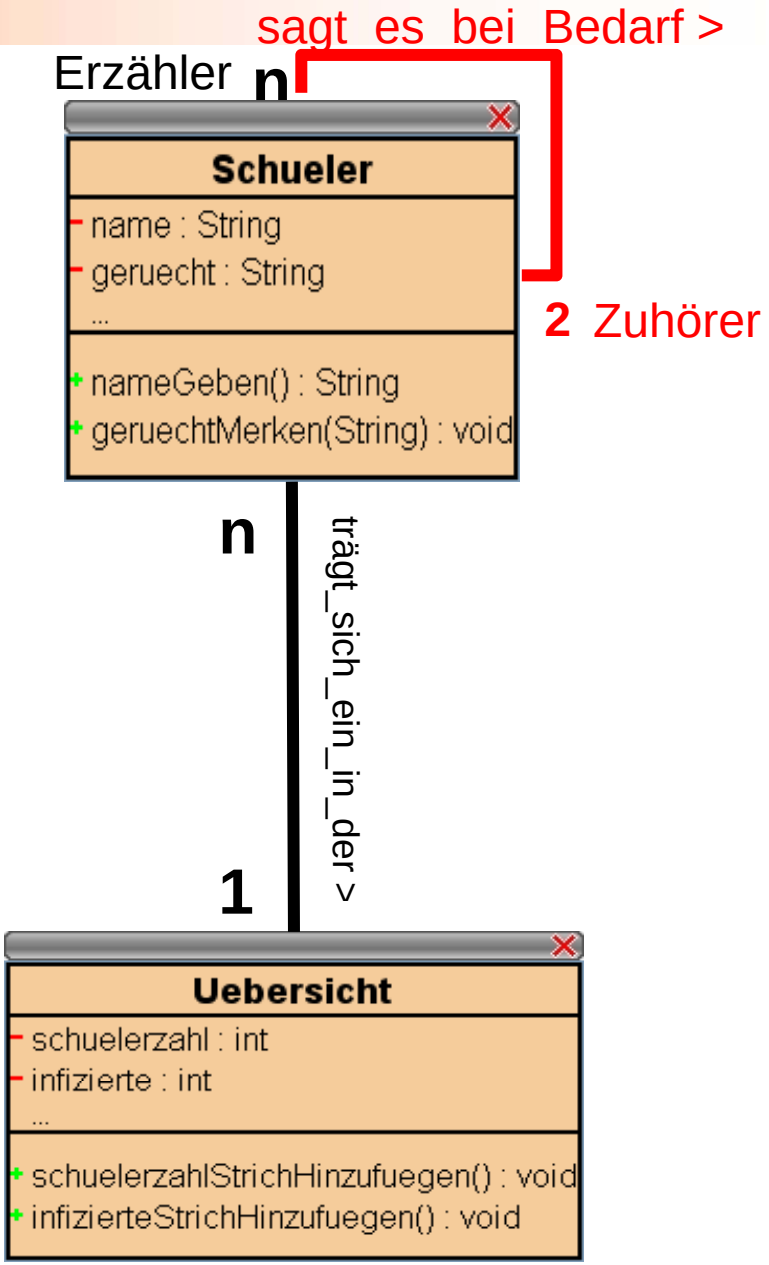


Klassendiagramm

- Zusätzlich können im Klassendiagramm Rollen angegeben werden, um die Nutzungsrichtung einer Beziehung zu veranschaulichen.



Umsetzung der Gerüchtebeziehungen zwischen Schülern



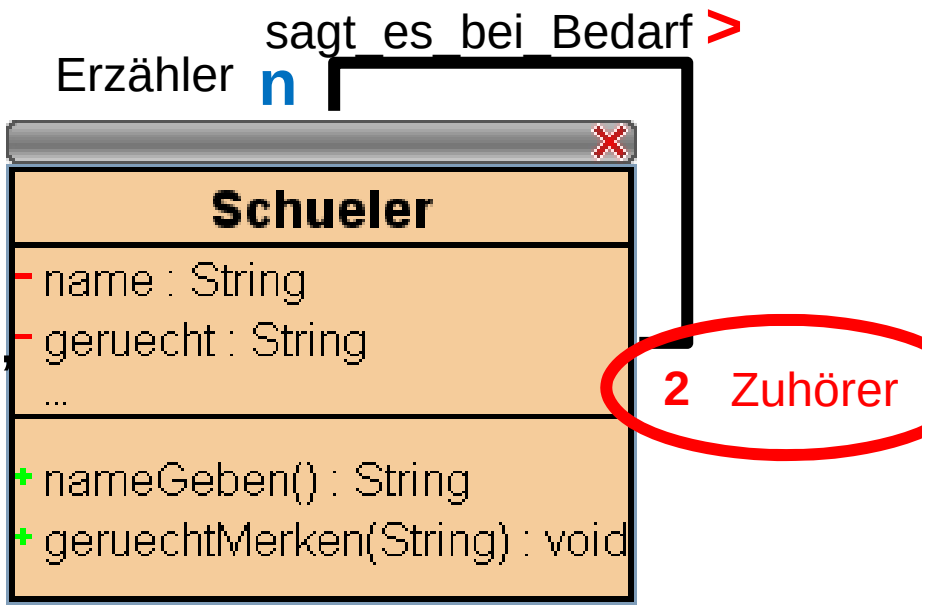
Schritt 1 : Referenzattribute deklarieren

Laut Aufgabe: Ein Schüler kann maximal zwei Zuhörer haben

```
public class Schueler {
    /*---------------Attribute-----*
    private String name;
    private String geruecht;
    private Uebersicht strichliste;
    private Schueler zuhoerer1;
    private Schueler zuhoerer2;
```

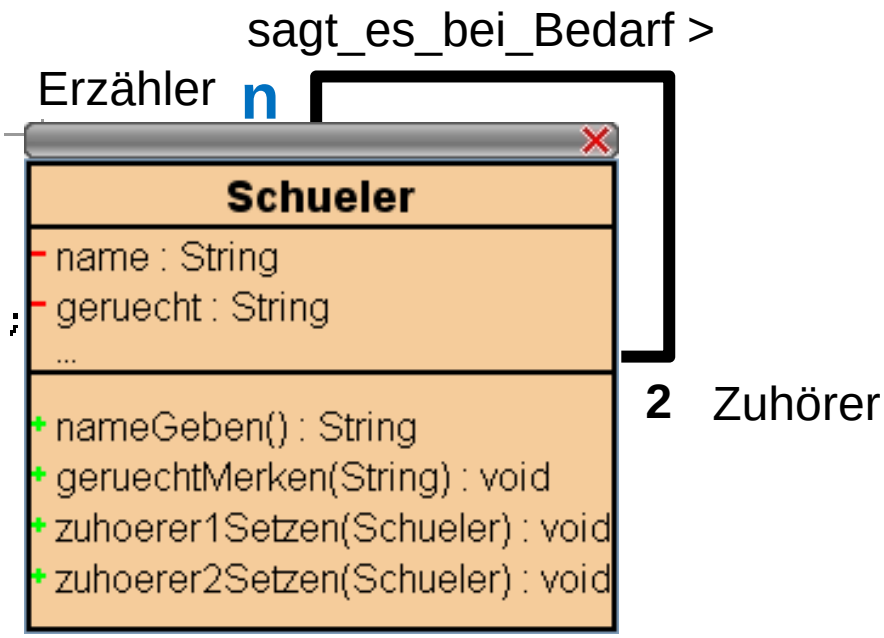
Deklaration der Referenzattribute zuhoerer1 und zuhoerer2.

Experiment : Von dieser Klasse zwei Objekte erzeugen, Attribute inspizieren…



Schritt 2 : Zuweisung eines (Ziel-)Objekts ermöglichen

```
public class Schueler {
    /*----------------Attribute-----
    private String name;
    private String geruecht;
    private Uebersicht strichliste;
    private Schueler zuhoerer1;
    private Schueler zuhoerer2;

        ...

    /*---------------Methoden------*/
    public void zuhoerer1Setzen(Schueler neuerZuhoerer) {
        zuhoerer1 = neuerZuhoerer;
    }
    public void zuhoerer2Setzen(Schueler neuerZuhoerer) {
        zuhoerer2 = neuerZuhoerer;
    }

        ...
```

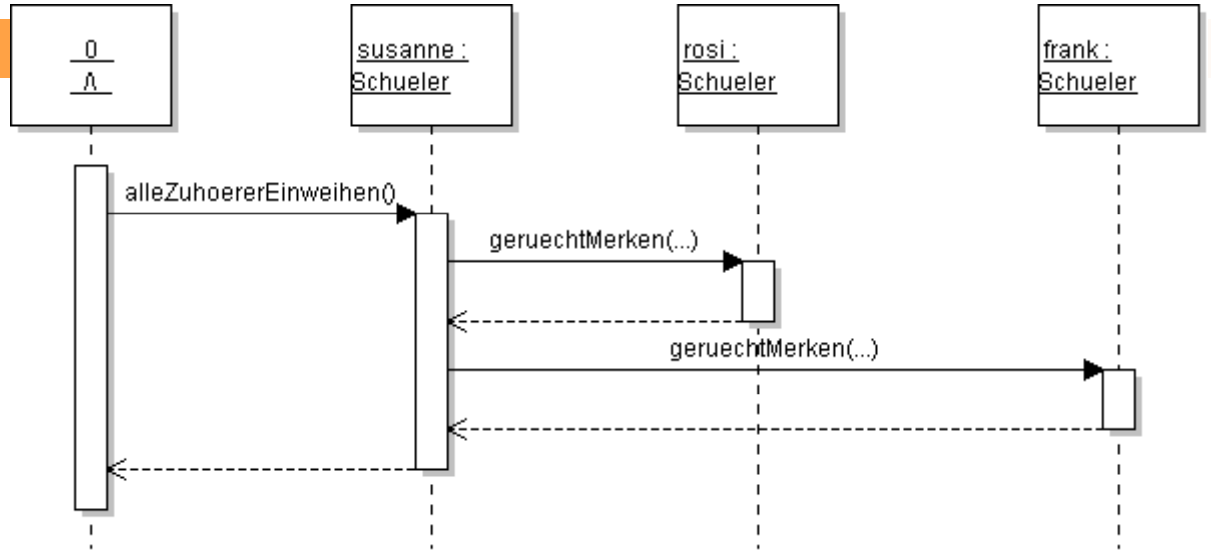


Diese Dienste nutzt man, um Beziehungen herzustellen

Definition der Klasse Schueler erweitern

- Experimente mit der Klasse Schueler (Objekte erzeugen, Dienste nutzen) :
  ein Beziehungsgeflecht kann aufgebaut werden
- Aber Susanne kann Rosi noch nichts erzählen!
- Schön wäre dazu eine Methode „alleZuhoererEinweihen()“

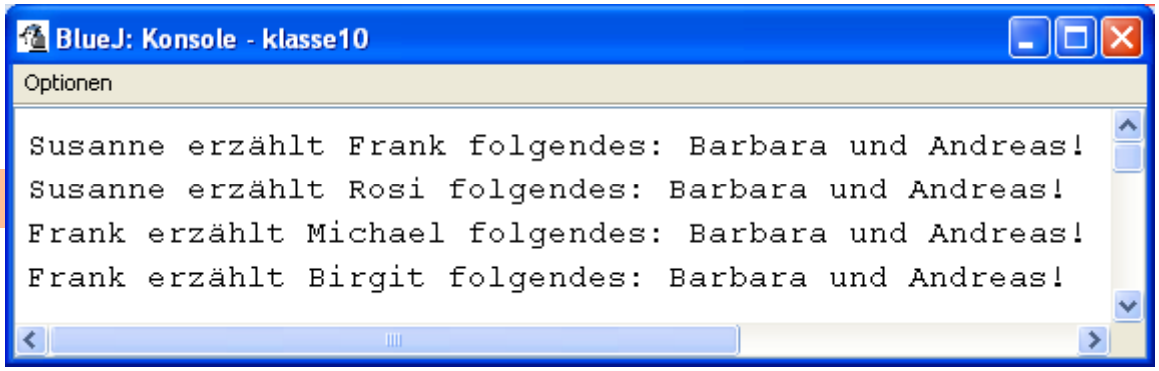
Methode „alleZuhoererEinweihen()“

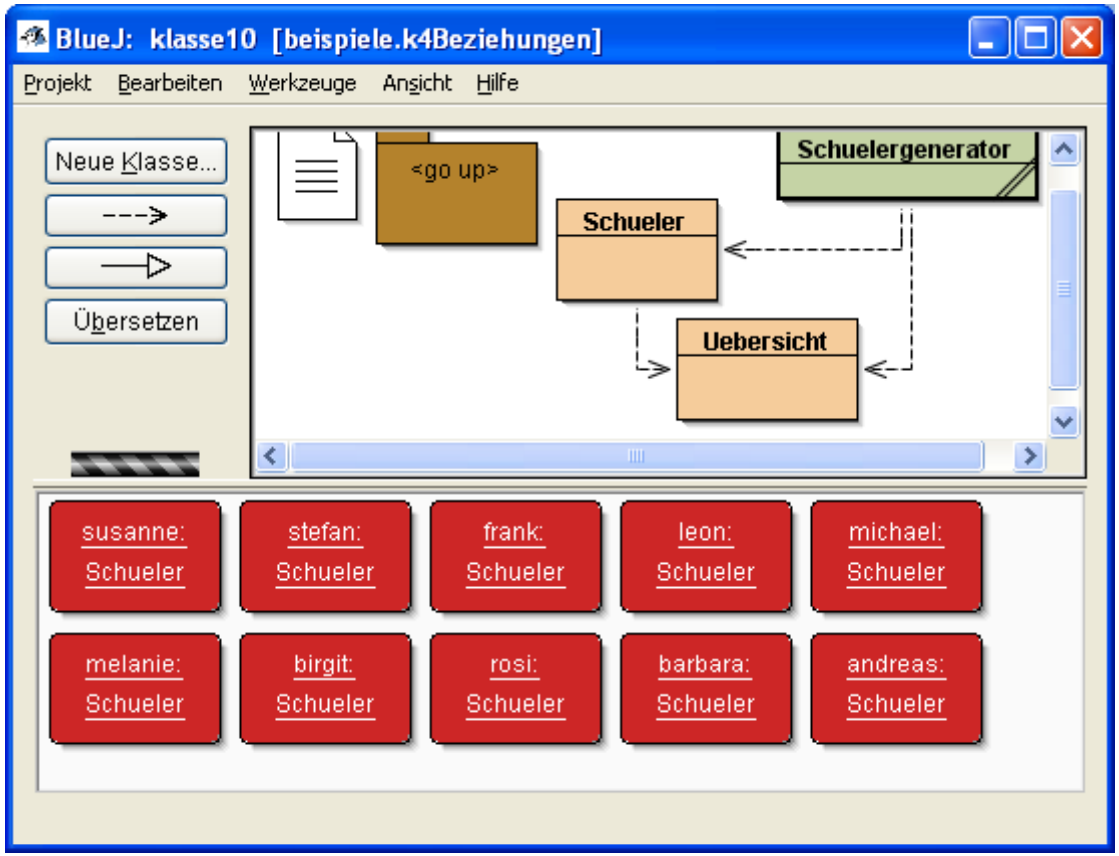


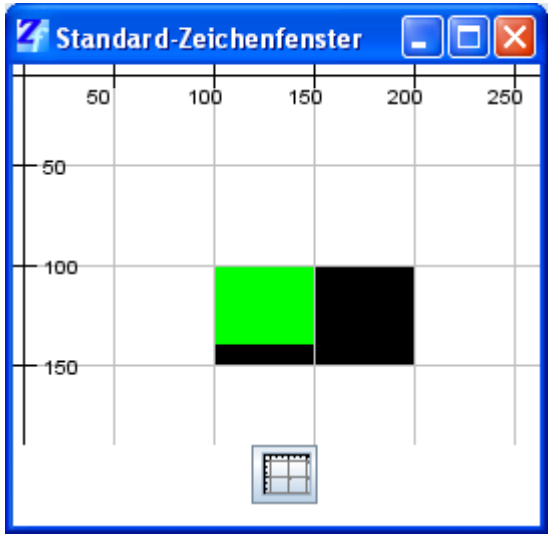

```
public void alleZuhoererEinweihen() {
    if (zuhoerer1 != null) {
        zuhoerer1.geruechtMerken(geruecht);
        System.out.println(name + " erzählt " + zuhoerer1.nameGeben()
            + " folgendes: " + geruecht);
    }
    if (zuhoerer2 != null) {
        zuhoerer2.geruechtMerken(geruecht);
        System.out.println(name + " erzählt " + zuhoerer2.nameGeben()
            + " folgendes: " + geruecht);
    }
}
```

Später ergänzen

Beziehungen







Vielen Dank

für Ihre Aufmerksamkeit!

Material finden Sie unter

www.fim.uni-passau.de

-> schulen

Ausblick

1. Objektorientierte Konzepte
2. Zustandsmodelle
3. Algorithmen (sehr knapp)
4. Beziehungen zwischen Objekten nutzen
5. Klassendefinitionen spezialisieren

Abschnitt 5 Klassendefinitionen spezialisieren

- **Bisher: Objekte nutzen Dienste**, die andere Objekte anbieten. Dazu pflegen Objekte **Beziehungen** zu anderen Objekten.
- eine Klassendefinitionen „mehrfach ausnutzen“ hieß bisher: Objekte einer Klasse erzeugen und arbeiten lassen.
- Analogie: Autofabrik (<-> Klasse) produziert fünf Autos (<-> Objekte) für den Fuhrpark einer Autovermietung (<-> auch ein Objekt), weil die Autovermietung das bestellt hat. Die Autovermietung nutzt diese Autos.

Klassendefinitionen spezialisieren

- **Jetzt** kommt eine ganz andere Idee ins Spiel:
  Manchmal ist es sinnvoll **Fabriken nicht neu zu planen**, (<-> zu definieren), sondern existierende Pläne „zu übernehmen und geeignet zu variieren“.
- Analogie: Kleinwagenfabrik (<-> Klasse) stellt eine „Spezialisierung“ einer Autofabrik (<-> Klasse) dar.
  Anmerkung für Lehrkräfte: Ähnlich wie im realen Leben bei der Kleinwagen- bzw. Autofabrik muss man jedoch die Klasse, von der spezialisiert werden soll, recht gut kennen, um sinnvoll spezialisieren zu können. Oft liest man sich sogar die Klassendefinition der Oberklasse und nicht nur die Dokumentation durch.

Ausblick: Schüler und Gerüchte mit mehreren Ansichten

- → **`beispiele.k5Beziehungen`**