Überblick

Klassen, Klassen, Klassen

- Die bedeutendste Errungenschaft der oo Programmierung ist also die Möglichkeit, eigene Datentypen durch Klassen zu spezifizieren
- Zusätzlich gibt es noch weitere Aspekte und Feinheiten, von denen wir nun einige diskutieren wollen
- Vorher aber noch einmal Klartext: die Neuheiten des oo Paradigmas sind hauptsächlich neue Möglichkeiten zur Datendarstellung bzw. Modellierung von Daten zu deren Verarbeitung
- Die eigentliche Verarbeitung geschieht in den (Klassen- und Objekt-)Methoden: hier werden die eigentlichen Algorithmen implementiert, bei Java mit Hilfe des imperativen Paradigmas
- oo ist daher eigentlich nicht so sehr ein Programmier- sondern vielmehr ein Modellierungs-Paradigma

Klassen als Mittel der Abstraktion

- Klassen bieten eine weitere Möglichkeit der *Abstraktion*, da es dem Programmierer nun möglich ist, ähnliche Objekte (mit ähnlichen Eigenschaften) zusammen zu fassen
- Statt für alle möglichen Objekte gewisse Eigenschaften (insbes. Methoden) zu programmieren, werden die Unterschiede wegabstrahiert und die Methode einmal zur Verfügung gestellt (iimplementiert)
- Das spart nicht nur Arbeit sondern auch potentielle Fehlerquellen wie z.B. copy-paste von vermeintlich redundantem Programmcode

Klassen als Mittel der Abstraktion

- Ein anschauliches Beispiel:
  - Sie lernen einmal wie ein Auto zu fahren ist und können dann alle möglichen Autos (untersch. Hersteller etc.) fahren
  - Warum? Die Bedienung abstrahiert von den speziellen (für das Fahren irrelevanten) Eigenschaften der unterschiedlichen Fabrikate
  - Sie müssen z.B. nur wissen, wie man das Lenkrad dreht, um zu lenken (wie diese Lenk-Bewegung tatsächlich auf die Straße gebracht wird, müssen Sie nicht wissen)
  - Analoges gilt fürs Gasgeben, Bremsen, Blinken, etc.

Klassen als Mittel der Abstraktion

- Noch ein Beispiel: Lichtschalter in einem Haus
  - Alle Lichtschalter sind gleich zu bedienen
  - Alle Lichtschalter sind gleich konstruiert
  - Dennoch unterscheidet sich der Lichtschalter für die Flurbeleuchtung vom Lichtschalter fürs Badezimmer: Es ist nicht derselbe Lichtschalter
  - Statt jeden einzelnen Lichtschalter neu zu modellieren/programmieren, abstrahiert man eine Klasse Lichtschalter, der alle Funktionalitäten nur einmal (für alle möglichen Lichtschalter) implementiert

Klassen als Mittel der Abstraktion

- In unserem konkreten Beispiel der Klasse `Rechteck`:
  - Sie können z.B. testen, ob ein Punkt in einem beliebigen Rechteck ist, ohne andere (für diesen Test irrelevante) Besonderheiten unterschiedlicher Rechtecke (wie z.B. der Flächeninhalt, die Farbe des Rechtecks (wenn es überhaupt eine hat)) zu kennen
- Fazit: Abstraktion hilft, Details zu ignorieren, und reduziert damit die Komplexität des Problems; dadurch werden komplexe Apparate und Techniken beherrschbar

Klassen als Mittel der Datenkapselung

- Eine weitere wichtige Errungenschaft von Klassen ist die Möglichkeit, die innere Struktur der Objekte einer Klasse (also die Daten-Komponenten/Instanzvariablen) vor dem Benutzer/Programmierer zu *verstecken*
- Man spricht hier von *Datenkapselung* (engl. *information hiding*)
- Konkret geschieht das mit dem schon kurz angesprochenen Sichtbarkeitsattribut **`private`**
- Warum ist das wichtig?

Klassen als Mittel der Datenkapselung

- Die Instanzvariablen (auch Attribute) werden für jedes konkrete Objekt neu angelegt und entspr. mit Literalen des jeweiligen Typs belegt (ein konkretes Rechteck hat vier konkrete Werte für die Attribute `xLow`, `xUp`, `yLow` und `yUp`)
- Die Instanzvariablen repräsentieren den *Zustand* der Objekte
- Die Methoden sind nur einmal realisiert / definiert und operieren bei jedem Aufruf auf den Daten eines bestimmten konkreten Objekts (sie repräsentieren das Verhalten der Objekte)
- Kapselung bedeutet, dass (von gewollten Ausnahmen abgesehen) die Methoden die einzige Möglichkeit darstellen, mit einem konkreten Objekt zu kommunizieren und so Informationen über dessen Zustand (die aktuellen Werte der Attribute) zu gewinnen oder diesen zu verändern

Klassen als Mittel der Datenkapselung

- Kapselung hilft, die Komplexität der Bedienung eines Objekts zu reduzieren
- Durch Kapselung werden die Implementierungsdetails von Objekten verborgen
- Dadurch können Daten nicht bewusst oder versehentlich verändert werden
- Kapselung ist daher auch ein wichtiger Sicherheitsaspekt: Ein direkter Zugriff auf die Daten wird unterbunden, der Zugriff erfolgt nur über definierte *Schnittstellen* nämlich über die bereitgestellten Methoden
- Beispiel: welche Attribute den Zustand eines Lichtschalters definieren kann uns egal sein, solange wir wissen, wie wir den Lichtschalter bedienen müssen; tatsächlich können wir den Zustand eines Lichtschalters nur über dessen Schnittstelle verändern (indem wir den Schalter drücken oder drehen)

Klassen als Mittel der Datenkapselung

- Die Klasse `Rechteck` sauber gekapselt:

```
public class Rechteck {
      private double xLow;
      private double xUp;
      private double yLow;
      private double yUp;

      public double getXLow() { return xLow; }
      ...
}
```

- Zugriff erfolgt hier nur z.B. über die **`public`** Methode `getXLow`, die als **`private`** deklarierten Elemente sind *nach außen* nicht sichtbar
- Denkbar wäre z.B. nun auch eine Methode

  `public void` `verschiebe(`**`double`** `xOffset,` **`double`** `yOffset)`

  um den Zustand eines konkreten Rechtecks zu ändern, etc.

Klassen als Mittel der Datenkapselung

- Eine (gleichwertige) Alternative wäre i.Ü. ein Rechteck durch den linken unteren Punkt und eine Ausdehnung in x- und eine Ausdehnung in y-Richtung zu definieren
- Die entsprechende Klasse `RechteckAlternativ`

```
public class RechteckAlternativ {
      private double xCoord; // x-Koordinate des Punktes
      private double yCoord; // y-Koordinate des Punktes
      private double xRange; // x-Ausdehnung
      private double yRange; // y-Ausdehnung
      ...
}
```

- Auch hier könnte natürlich eine Methode `verschiebe` implementiert werden
- An diesem Beispiel sieht man auch die Stärken der Kapselung: dem Benutzer der Methode `verschiebe` ist letztlich herzlich egal, wie die Klasse nun intern strukturiert ist (solange die Methode das macht, was sie soll)

Klassen als Mittel der Datenkapselung

- Auch der Typ `String` ist ein perfektes Beispiel für Daten-Kapselung
- Die Implementierungsdetails von Zeichenketten (letztlich als **`char`**`[]` bleiben vor dem Benutzer verborgen
- Schnittstelle dieser Klasse ist aber relativ speziell und ist für die allgemeine Veranschaulichung aller weiteren Aspekte der oo Programmierung nicht ganz perfekt geeignet
- Durch saubere Datenkapselung sind Klassen i.Ü. die konsequente Erweiterung der Idee der *abstrakten Datentypen*
  (Ja, da denken Sie mal drüber nach)

Klassen unterstützen die Wiederverwendbarkeit

- Durch Abstraktion und Kapselung wird die *Wiederverwendbarkeit* von Programmteilen gefördert
- Beispiel: Sog. Collections sind Objekte, die Sammlungen anderer Objekte aufnehmen und verarbeiten können
  - Collections sind meist sehr kompliziert aufgebaut haben dafür aber typischerweise eine einfache Art der Bedienung (Schnittstelle)
  - Werden Collections als Klasse realisiert, werden durch die Kapselung die komplexen Details des Aufbaus wegabstrahiert
  - Dies erleichtert die Wiederverwendung: Wenn ein Programm eine spezielle Collection benötigt, muss ein Objekt der passenden Klasse erzeugt werden, auf das das Programm über die einfache Schnittstelle (Methoden der Klasse) zugreifen kann
- Wiederverwendbarkeit hilft, effizienter und fehlerfreier zu programmieren

Beziehungen zwischen Klassen

- Betrachten wir noch einmal unsere Klasse

```
public class Rechteck {
     private double xLow;
     private double xUp;
     private double yLow;
     private double yUp;

     ...

     public boolean pointInside(double xCoord, double yCoord) { ... }
}
```

- Konsequent weiter gedacht, ist das Argument der Methode `pointInside` ein *Punkt*, bestehend aus zwei Koordinaten $(xCoord, yCoord)$
- Das Rechteck selbst wird ebenfalls von zwei solchen *Punkten* $(xLow, yLow)$ und (xUp, yUp) aufgespannt

Beziehungen zwischen Klassen

- Das riecht nach einer Klasse `Punkt`!

```
public class Punkt {
     private double x;
     private double y;

     public double getX() { return x; }

     public double getY() { return y; }
}
```

- Die Instanzvariablen `x` und `y` sind gekapselt, der einzige Zugriff erfolgt über die Objekt-Methoden `getX` und `getY`
- Damit ist i.Ü. der Zustand eines beliebigen Objekts vom Typ `Punkt` nicht veränderbar

Beziehungen zwischen Klassen

- Damit können wir `Rechteck` neu definieren

```
public class Rechteck {
     private Punkt low;
     private Punkt up;

     public boolean pointInside(Punkt p) { ... }
     ...
}
```

- Wie etwaige Methoden dieser Klasse implementiert werden, schauen wir uns später an
- Was wir mit diesem Beispiel eigentlich erstmal nur sehen sollen ist: Klassen und deren Objekte existieren typischerweise nicht isoliert, sondern stehen in *Beziehung* zueinander

Beziehungen zwischen Klassen

- Grundsätzlich gibt es in der OO Programmierung drei Arten von Beziehungen
- *Generalisierung* und *Spezialisierung* (“is-a”-Beziehungen)
  - Beispiel: ein *Quadrat* und ein *Rechteck* (Spezialisierungen) sind *Vierecke* (Generalisierung)
- *Aggregation* und *Komposition* (“part-of”-Beziehungen)
  - Beschreibt die Zusammensetzung eines Objekts aus anderen Objekten (Aggregation: nicht essentiell, Komposition: essentiell)
  - Beispiel: ein *Punkt* ist Teil eines *Rechtecks* (Komposition, das Rechteck existiert sonst inicht)
- *Verwendungs-* und *Aufrufbeziehungen*
  - Beispiel: ein *Rechteck* verwendet in unserem Beispiel ein *Punkt* als Eingabeparameter in einer Methode

Vererbung

- Eine “is-a”-Beziehung zwischen zwei Klassen `A` und `B` sagt aus, dass `B` alle Eigenschaften von `A` besitzt (und darüberhinaus typischerweise noch ein paar mehr)
- “is-a”-Beziehungen werden in der oo Programmierung durch *Vererbung* modelliert
- Die speziellere Klasse wird dabei nicht komplett neu definiert, sondern von der allgemeineren Klasse *abgeleitet*
- Die speziellere Klasse *erbt* implizit alle Eigenschaften der allgemeineren Klasse (auch: *Vaterklasse* oder *Oberklasse*) ohne, das sie nochmal explizit aufgeführt werden müssen
- Eigene Eigenschaften können nach Belieben hinzugefügt werden

Vererbung

- Vererbungen können mehrstufig sein, d.h. eine abgeleitete Klasse kann wiederum Vaterklasse für andere abgeleitete Klassen sein (dies führt ggfs. zu einer *Vererbungshierarchie*)
- Grundsätzlich kann eine Klasse auch von mehreren Vaterklassen abgeleitet sein (z.B. ist ein Amphibienfahrzeug eine Spezialisierung sowohl eines Wasserfahrzeug als auch eines Landfahrzeug)
- In diesem Fall spricht man von *Mehrfachvererbung*

Polymorphismus (bei Objekten)

- Beispiel: *Fährschiffe* können *Autos*, *Lastwägen* und *Motorräder* – oder ganz allgemein *Straßenfahrzeuge* (Vaterklasse) – aufnehmen
- Um diese Beziehung zu beschreiben, genügt es, zu definieren, dass *Fährschiffe Straßenfahrzeuge* aufnehmen
- *Polymorphismus* besagt, dass nicht nur Objekte der Klasse *Straßenfahrzeuge* aufgenommen werden können, sondern auch Objekte aller abgeleiteten Klassen, also auch von *Auto*, *Lastwagen* und *Motorräder*
- Die zusätzlichen Eigenschaften der abgeleiteten Klassen sind für das *Fährschiff* irrelevant, wichtig ist, dass sie die Eigenschaften eines *Straßenfahrzeugs* haben (was sie als Objekte einer von *Straßenfahrzeuge* abgeleitete Klasse auch haben)
- Andersherum funktioniert Polymorphismus nicht!

Polymorphismus (bei Methoden)

- Ein weiterer Aspekt ist, dass Objekte unterschiedlicher Klassen die gleiche Funktionalität besitzen können (die allerdings in jeder Klasse unterschiedlich realisiert ist)
- Beispiel:
  - Bei allen Straßenfahrzeugen gibt es die Funktionalität, die Anzahl der Reifen abzufragen (realisiert durch eine Methode “anzahlReifen”)
  - Diese Methode kann in allen drei Klassen (Auto, Lastwagen, Motorrad) denselben Namen haben
  - In allen drei Fällen steckt aber ein unterschiedlicher Algorithmus dahinter
  - Diese Funktionalität kann bereits in der Vaterklasse zur Verfügung stehen und in den abgeleiteten Klassen *überschrieben* (*verfeinert*) werden
  - Wird im Zusammenhang eines Fährschiffes die Anzahl der Reifen eines Straßenfahrzeugs benötigt, das auf diesem Schiff gerade transportiert wird, wird dynamisch ausgewählt, welche Methode ausgeführt wird, ohne dass sich das Fährschiff darum kümmern muss (ist das Straßenfahrzeug z.B. ein Auto, wird “anzahlReifen” der Klasse Auto ausgeführt)

oo Modellierung

- Betrachten wir das oo Paradigma noch aus einer anderen Perspektive (ohne unser bis hierher erarbeitetes Vorwissen)
- Oo Programmierung ist ein weiterer Ansatz, Problemstellungen der realen Welt zu modellieren
- Der oo Ansatz orientiert sich dabei an *Dingen* (Objekte, die gewisse Eigenschaften und ein gewisses Verhalten haben), die modelliert werden müssen
- Die oo Sichtweise stellt sich die Welt als *System von Objekten* vor, die untereinander *Botschaften* austauschen

oo Modellierung

Beispiel

- Susanne in Rosenheim möchte ihrer Freundin Gabi in Buxtehude einen Blumenstrauß schicken
- Susanne geht daher zum Blumenhändler Mark und erteilt ihm den Auftrag
- In der oo Programmierung ist Mark ein Objekt: Susanne sendet an Mark eine Botschaft: *Sende 7 gelbe Rosen an Gabi, Schneestr. 1 in Buxtehude*
- Susanne hat getan, was sie konnte, nun ist es in Marks Verantwortung, den Auftrag zu bearbeiten
- Mark versteht die Botschaft und weiß, was zu tun ist, d.h. er kennt einen Algorithmus für das verschicken von Blumen
- Er versucht einen Blumenhändler in Buxtehude finden: er findet Florist Sascha und schickt ihm eine leicht veränderte Botschaft (z.B. mit Absender)

oo Modellierung

Das Beispiel geht weiter

- Mark ist damit fertig: er hat die Verantwortung für den Prozess an Sascha weitergegeben
- Auch Sascha hat einen zur Botschaft passenden Algorithmus parat: er stellt die 7 Rosen zusammen und beauftragt seinen Boten Daniel, sie auszuliefern
- Daniel muss den Weg zur Zieladresse finden und fragt sein Navi, das ihm entsprechend antwortet
- Daniel findet den Weg, überreicht Gabi die Blumen und teilt ihr in einer Botschaft den Absender mit
- Damit ist der von Susanne angestoßene Vorgang, an dem mehrere Objekte beteiligt waren, beendet

oo Modellierung

- Objekte besitzen Eigenschaften (die Attribute/Instanzvariablen)
  z.B. Blumenhändler: Ort an dem er sein Geschäft hat, Name, Telefonnummer, Öffnungszeiten, Warenbestand, etc.
- Objekte können bestimmte Operationen (die (Objekt-)Methoden) ausführen
  z.B. Blumenhändler: Lieferauftrag für Blumen entgegennehmen, Sträuße binden, Boten schicken, Blumen beim Großhandel einkaufen, etc.
- Wenn ein Objekt eine geeignete Botschaft empfängt, wird eine zur Botschaft passende Operation gestartet (Methode aufgerufen)
- Der Umwelt (d.h. den anderen Objekten) ist bekannt welche Methoden ein Objekt beherrscht
- Allerdings weiß die Umwelt von den Methoden nur, was sie bewirken und welche Daten sie als Eingabe benötigen

oo Modellierung

- Die Umwelt weiß aber nicht, wie das Objekt funktioniert, d.h. nach welchen Algorithmen die Botschaften verarbeitet werden
- Das bleibt „privates“ Geheimnis des Objekts
  - Z.B. hat Susanne keine Ahnung, wie Mark den Blumentransport bewerkstelligt (es interessiert sie vermutlich auch gar nicht)
  - Susannes Aufgabe war einzig und allein, ein für ihr Problem *geeignetes* Objekt zu finden und ihm eine geeignete Botschaft zu senden (ungeeignete Objekte wären z.B. Kati die Zahnärztin oder Markus der Immobilienmakler gewesen, denn diese Objekte hätten Susannes Nachricht gar nicht verstanden)
  - Für Susanne war zudem wichtig, zu wissen, *wie* sie die Botschaft für Mark formulieren muss
- Eine Methode ist die Implementierung eines Algorithmus, in Java kommt hier bekanntermaßen das imperative Paradigma ins Spiel

oo Modellierung

- Die Objekte in diesem Beispiel kann man in Gruppen (Klassen) einteilen z.B. sind Sascha und Mark *Blumenhändler*:
  - Sie beherrschen *die selben Methoden* und besitzen *die selben Attribute* (z.B. Ort oder Öffnungszeiten)
  - Sie sind aber unterschiedliche Objekte: die Attribute haben *unterschiedliche Werte*

  Man sagt: Sascha und Mark sind Objekte (Instanzen) der Klasse *Blumenhändler*
- Eine Klasse ist eine Definition eines bestimmten Typs von Objekten, ein Bauplan indem die Methoden und Attribute beschrieben werden
- Nach diesem Schema können Objekte (Instanzen) einer Klasse erzeugt werden
- Ein Objekt ist eine Konkretisierung (Inkarnation) einer Klasse

oo Modellierung

- Alle Instanzen einer Klasse sind von der Struktur (Methoden/Attribute) her gleich, sie unterscheiden sich allein in der Belegung ihrer Attribute mit Werten
- Beispiel: Sascha und Mark haben beide das Attribut Ort, aber bei Sascha hat es den Wert „Buxtehude“ und bei Mark den Wert „Rosenheim“

Zusammenfassung

- Eine Klasse definiert die Attribute und Methoden ihrer Objekte
- Der Zustand eines konkreten Objekts wird durch seine Attributwerte und Verbindungen (Links) zu anderen konkreten Objekten bestimmt
- Das mögliche Verhalten eines Objekts wird durch die Menge von Methoden beschrieben
- Die wichtigsten Konzepte der oo Programmierung sind:
  - Abstraktion
  - Kapselung
  - Wiederverwendung
  - Beziehungen
  - Polymorphismus

  die wir uns bereits erarbeitet haben