Programmierzertifikat Objekt-Orientierte Programmierung mit Java

Vorlesung 01: Datenmodellierung

Peter Thiemann

Universität Freiburg, Germany

SS 2008

Inhalt

Zahlen

int ganze Zahlen im Bereich $[-2^{31}, 2^{31} - 1]$,

- exakt, alle Rechenoperationen modulo 2^{32}
- Wertebereich: zwischen -2147483648 und 2147483647
- Literale: `42` `0` `-16384` `+911`

double Gleitkommazahlen mit doppelter Genauigkeit (64 Bit)
Vgl. Standard IEEE 754-1985, Vorlesung Technische Informatik

- inexakt, alle Rechenoperationen werden gerundet
- Wertebereich: etwa $\pm 1.7976931348623157 \times 10^{308}$
- Literale: `3.14159265` `-.14142`
  `+6.02214179E+23` `8.854E-12`

Wahrheitswerte und Zeichenfolgen

boolean Wahrheitswerte

- Wertebereich: nur `true` und `false`
- Literale: `true` `false`

String Zeichenfolgen (Strings)

- Wertebereich: alle endlichen Folgen von Zeichen (bis zu einer unspezifizierten Maximallänge)
- Literale: `""` `"Sushi"` `"küçük"` `"#§$&???"`

Einfache Klassen

- Primitive Datentypen reichen nicht für alle Anwendungen aus
- Beispiel:

  . . . Das Programm soll die Buchhaltung für einen Teegroßhändler unterstützen. Die Quittung für eine Lieferung beinhaltet die Teesorte, den Preis (in Euro pro kg) und das Gewicht der Lieferung (in kg). . . .

- Beispielquittungen
  - 100kg Darjeeling zu 40.10 EUR
  - 150kg Assam zu 27.90 EUR
  - 140kg Ceylon zu 27.90 EUR

Modellierung einer Teelieferung

```
2  // Repräsentation einer Rechnung für eine Teelieferung
3  class Tea {
6      String kind; // Teesorte
7      int price; // in Eurocent pro kg
8      int weight; // in kg
11     Tea(String kind, int price, int weight) {
12         this.kind = kind;
13         this.price = price;
14         this.weight = weight;
15     }
34 }
```

- Vollständige *Klassendefinition*

Grundgerüst einer Klassendefinition

```
2 // Repräsentation einer Rechnung für eine Teelieferung
3 class Tea {
```

- Benennt den Klassentyp Tea
- Rumpf der Klasse spezifiziert
  - die Komponenten der *Objekte* vom Klassentyp
  - den *Konstruktor* des Klassentyps
  - das Verhalten der Objekten (später)

```
34 }
```

Felddeklarationen

```
6     String kind; // Teesorte
7     int price; // in Eurocent pro kg
8     int weight; // in kg
```

- Beschreibt die Komponenten: *Instanzvariable*, *Felder*, *Attribute*
- Beschreibung eines Felds
  - Typ des Feldes (String, int)
  - Name des Feldes (`kind`, `price`, `weight`)
- Kommentare: // bis Zeilenende

Konstruktordeklaration

```
11    Tea(String kind, int price, int weight) {
12        this.kind = kind;
13        this.price = price;
14        this.weight = weight;
15    }
```

- Beschreibt den *Konstruktor*: Funktion, die aus den Werten der Komponenten ein neues Objekt initialisiert
- Argumente des Konstruktors entsprechen den Feldern
- Rumpf des Konstruktors enthält Zuweisungen der Form

$$\textbf{this}.\mathit{feldname} = \mathit{feldname}$$

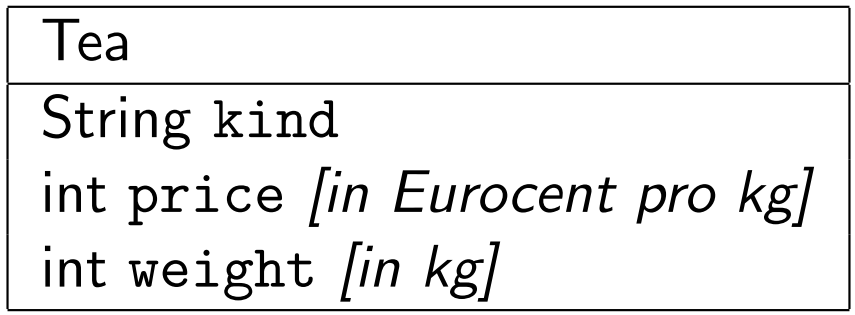
- **this** ist das Objekt, das gerade konstruiert wird
- **this**.*feldname* bezeichnet das entsprechende Feld des Objekts
- *feldname* bezeichnet den Wert des entsprechenden Konstruktorarguments

Beispiel für Teelieferungen

- 100kg Darjeeling zu 40.10 EUR
- 150kg Assam zu 27.90 EUR
- 140kg Ceylon zu 27.90 EUR

```
new Tea("Darjeeling", 4010, 100)
new Tea("Assam", 2790, 150)
new Tea("Ceylon", 2790, 140)
```

Klassendiagramm

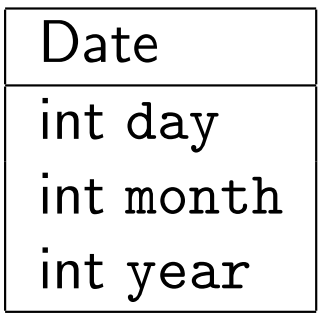


- Die Spezifikation einer Klasse kann auch als *Klassendiagramm* angegeben werden.
- Obere Abteilung: Name der Klasse
- Untere Abteilung: Felddeklarationen
- Anmerkung: Klassendiagramme werden in der Softwaretechnik verwendet. Sie sind im UML (Unified Modeling Language) Standard definiert. Sie sind nützliche Werkzeuge für die Datenmodellierung.

Beispiel: Datumsklasse

Ein Datumswert besteht aus Tag, Monat und Jahr.

- Drei Komponenten
- Jede Komponente kann durch int repräsentiert werden.
- Klassendiagramm dazu



Beispiel: Implementierung der Datumsklasse

```
1  // Ein Datum
2  class Date {
3    int day;
4    int month;
5    int year;
6
7    Date(int day, int month, int year) {
8      this.day = day;
9      this.month = month;
10     this.year = year;
11   }
12 }
```

Beispiel: Verwendung der Datumsklasse

- Korrekte Beispiele

```
new Date (30, 9, 2007) // 30. September 2007
new Date (13, 4, 2003) // 13. April 2003
new Date ( 1, 10, 1999) // 1. Oktober 1999
```

- Aber auch sinnlose Date Objekte sind möglich

```
new Date (33, 88, 1600) // ???
new Date (−1, −4, 0) // ???
```

- Anmerkung: Das wird noch ausgeschlossen.

Beispiel: Billardkugeln

Die Position einer Billardkugel auf dem Tisch wird durch ihre X- und Y-Koordinaten beschrieben. Jede Billardkugel besitzt einen Radius, der durch die Spielregel vorgeschrieben ist.

- Drei Komponenten
- Jede Komponente kann durch int repräsentiert werden.
- Klassendiagramm dazu

Ball
int `x` int `y` int `RADIUS` = 5 *[durch Spielregel vorgegeben]*

Beispiel: Implementierung von Billardkugeln

```
1  // eine Billardkugel
2  class Ball {
3    int x;
4    int y;
5    int RADIUS = 5; // durch Spielregel vorgegeben
6
7    Ball(int x, int y) {
8      this.x = x;
9      this.y = y;
10   }
11 }
```

- Zeile 4 *initialisiert* das Feld `RADIUS` auf den Wert 5.
- Der Konstruktor (Zeile 6-9) nimmt kein `RADIUS`-Argument.
- Der Konstruktor darf das `RADIUS`-Feld nicht setzen.

$\Rightarrow$ Das `RADIUS`-Feld eines jeden Ball-Objekts hat den Wert 5 und kann vom Konstruktor nicht anders gesetzt werden.

Beispiel: Verwendung von Billardkugeln

new Ball (36, 45) // ==> *Ball(x = 36, y = 45, RADIUS = 5)*
new Ball (100, 3) // ==> *Ball(x = 100, y = 3, RADIUS = 5)*

- Auch sinnlose Werte sind möglich:
  - außerhalb des Tisches
  - negative Koordinaten

Zusammenfassung

- Eine Klasse spezifiziert einen zusammengesetzten Datentyp, den *Klassentyp*.
- Die zum Klassentyp gehörigen Werte sind die *Instanzen* bzw. *Objekte* der Klasse.
- Ein Objekt enthält die Werte der Komponenten in den *Instanzvariablen*.
- Werte vom Klassentyp C werden durch den Konstruktoraufruf

$$\textbf{new } \mathrm{C}(v_1, \ldots, v_n)$$

gebildet, wobei $v_1, \ldots, v_n$ die Werte der Instanzvariablen sind.

Erstellen einer Klasse

1. Studiere die Problembeschreibung. Identifiziere die darin beschriebenen Objekte und ihre Attribute und schreibe sie in Form eines Klassendiagramms.
2. Übersetze das Klassendiagramm in eine Klassendefinition. Füge einen Kommentar hinzu, der den Zweck der Klasse erklärt. (Mechanisch, außer für Felder mit fest vorgegebenen Werten)
3. Repräsentiere einige Beispiele durch Objekte. Erstelle Objekte und stelle fest, ob sie Beispielobjekten entsprechen. Notiere auftretende Probleme als Kommentare in der Klassendefinition.

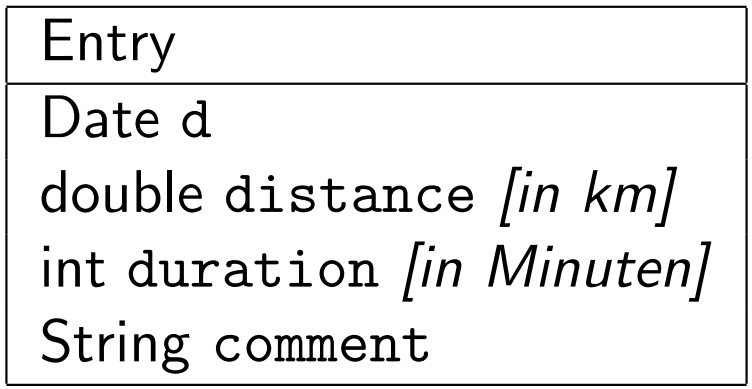
Objekte, die Objekte enthalten

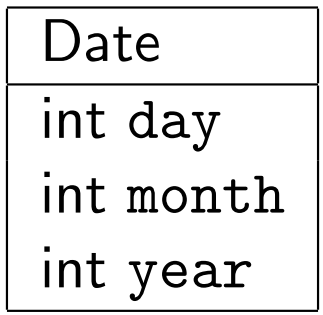
Entwickle ein Programm, das ein Lauftagebuch führt. Es enthält einen Eintrag pro Lauf. Ein Eintrag besteht aus dem Datum, der zurückgelegten Entfernung, der Dauer des Laufs und einem Kommentar zum Zustand des Läufers nach dem Lauf.

- Eintrag besteht logisch aus vier Bestandteilen
- Das Datum hat selbst Bestandteile (Tag, Monat, Jahr), deren Natur aber für das Konzept Eintrag nicht wichtig sind.

Eintrag im Lauftagebuch

Klassendiagramm





Eintrag im Lauftagebuch

Implementierung

```
1  // ein Eintrag in einem Lauftagebuch
2  class Entry {
3    Date d;
4    double distance; // in km
5    int duration; // in Minuten
6    String comment;
7
8    Entry(Date d, double distance, int duration, String comment) {
9      this.d = d;
10     this.distance = distance;
11     this.duration = duration;
12     this.comment = comment;
13   }
14 }
```

Eintrag im Lauftagebuch

Beispielobjekte

- Beispieleinträge
  - am 5. Juni 2003, 8.5 km in 27 Minuten, gut
  - am 6. Juni 2003, 4.5 km in 24 Minuten, müde
  - am 23. Juni 2003, 42.2 km in 150 Minuten, erschöpft
- . . . als Objekte in einem Ausdruck

```
new Entry (new Date (5,6,2003), 8.5, 27, "gut")
new Entry (new Date (6,6,2003), 4.5, 24, "müde")
new Entry (new Date (23,6,2003), 42.2, 150, "erschöpft")
```

- . . . in zwei Schritten mit Hilfsdefinition

```
Date d1 = new Date (5,6,2003);
Entry e1 = new Entry (d1, 8.5, 27, "gut");
```

Eintrag im Lauftagebuch

Organisation der Beispiele in Hilfsklasse

```
1  // Beispiele für die Klasse Entry
2  class EntryExample {
3      Date d1 = new Date (5,6,2003);
4      Entry e1 = new Entry (this.d1, 8.5, 27, "gut");
5
6      Date d2 = new Date (6,6,2003);
7      Entry e2 = new Entry (this.d2, 4.5, 24, "müde");
8
9      Date d3 = new Date (23,6,2003);
10     Entry e3 = new Entry (this.d3, 42.2, 150, "erschöpft");
11
12     EntryExample () {
13     }
14 }
```

Beispiel: Zugfahrplan

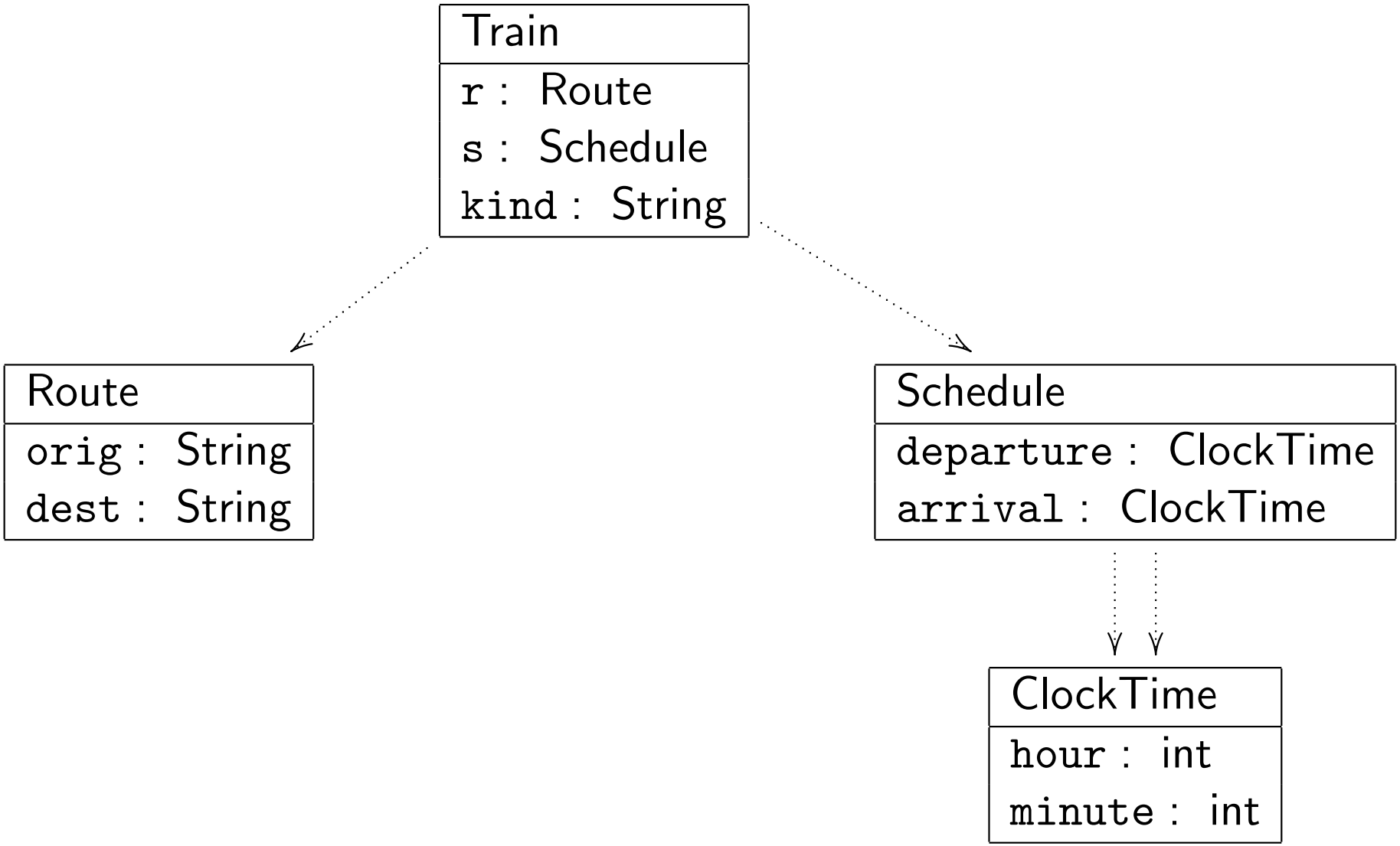
In einem Programm für Reiseauskünfte müssen Informationen über den Zugfahrplan vorgehalten werden. Für jeden Zug vermerkt der Plan die Strecke, die der Zug fährt, die Verkehrszeiten sowie die Information, was für eine Art von Zug es sich handelt (RB, RE, EC, ICE, . . .). Die Strecke wird durch den Start- und den Zielbahnhof bestimmt. Eine Verkehrszeit definiert die Abfahrts- und die Ankunftszeit eines Zuges.

- Ein Zug besteht aus drei Komponenten: Strecke, Verkehrszeit, Schnellzug.
- Strecken und Verkehrszeiten bestehen aus jeweils zwei Komponenten.
- Eine Verkehrszeit enthält zwei Zeitangaben, die selbst aus Stunden und Minuten bestehen.

$\Rightarrow$ Neuigkeit: Schachtelungstiefe von Objekten > 2

Beispiel: Zugfahrplan

Klassendiagramme



Beispiel: Zugfahrplan / Implementierung

```
1  // eine Zugfahrt
2  class Train {
3    Route r;
4    Schedule s;
5    String kind;
6
7    Train(Route r, Schedule s, String kind) {
8      this.r = r;
9      this.s = s;
10     this.kind = kind;
11   }
12 }
```

```
1  // eine Verkehrszeit
2  class Schedule {
3    ClockTime departure;
4    ClockTime arrival;
5
6    Schedule(ClockTime departure,
7             ClockTime arrival) {
8      this.departure = departure;
9      this.arrival = arrival;
10   }
11 }
```

```
1  // eine Bahnstrecke
2  class Route {
3    String orig;
4    String dest;
5
6    Route(String orig, String dest) {
7      this.orig = orig;
8      this.dest = dest;
9    }
10 }
```

```
1  // eine Uhrzeit
2  class ClockTime {
3    int hour;
4    int minute;
5
6    ClockTime(int hour, int minute) {
7      this.hour = hour;
8      this.minute = minute;
9    }
10 }
```

Beispiel: Zugfahrplan

Beispielzüge

```
Route r1 = new Route ("Freiburg", "Dortmund");
Route r2 = new Route ("Basel", "Paris");

ClockTime ct1 = new ClockTime (13,04);
ClockTime ct2 = new ClockTime (18,20);
ClockTime ct3 = new ClockTime (14,57);
ClockTime ct4 = new ClockTime (18,34);

Schedule s1 = new Schedule (ct1, ct2);
Schedule s2 = new Schedule (ct3, ct4);

Train t1 = new Train (r1, s1, "ICE");
Train t2 = new Train (r2, s2, "TGV");
```

Erstellen einer zusammengesetzten Klasse

1. Identifiziere die beteiligten Klassen und erstelle Klassendiagramme. Gehe dabei top-down vor.
2. Übersetze die Klassendiagramme in Klassendefinitionen. Beginne dabei mit den einfachen Klassen, die keine Felder von Klassentyp enthalten.
   (Zusammengesetzte Klassen heißen auch *Aggregate* oder *Kompositionen*)
3. Illustriere **alle** Klassen durch Beispiele. Beginne hierbei mit den einfachen Klassen.

Objekte mit unterschiedlichen Ausprägungen

In einem Zeichenprogramm sollen verschiedene geometrische Figuren in einem Koordinatensystem (Einheit: ein Pixel) dargestellt werden. Zunächst geht es um drei Art von Figuren

- *Quadrate mit Referenzpunkt links oben und gegebener Seitenlänge,*
- *Kreise mit dem Mittelpunkt als Referenzpunkt und gegebenem Radius und*
- *Punkte, die nur durch den Referenzpunkt gegeben sind und als Scheibe mit einem Radius von 3 Pixeln wiedergegeben werden.*

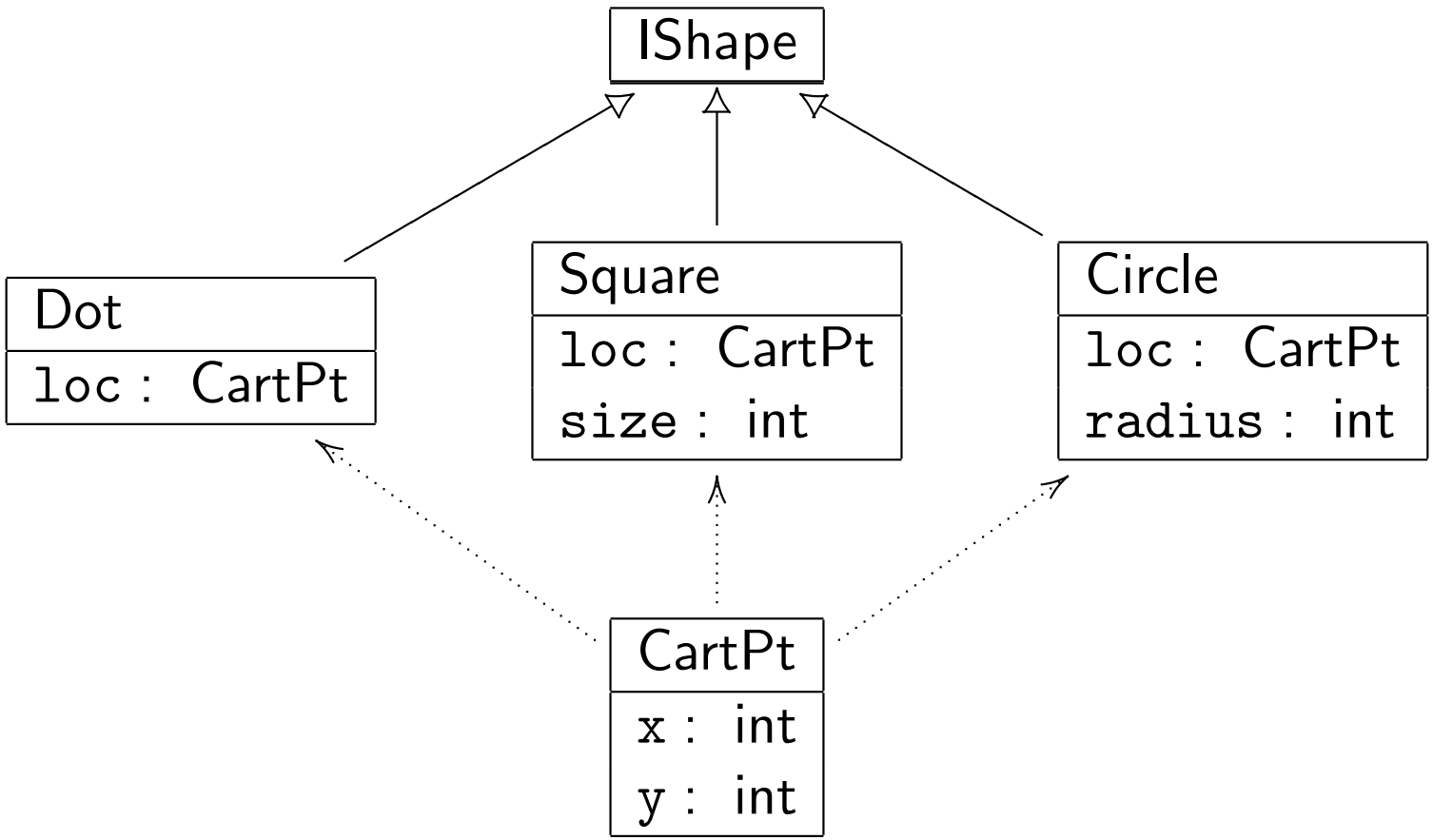
Vereinigung von Klassen

Klar Jede Art Figur kann durch eine zusammengesetzte Klasse repräsentiert werden. Der Referenzpunkt wird jeweils durch ein separates Punktobjekt dargestellt.

$\Rightarrow$ drei unterschiedliche Klassen, deren Objekte nicht miteinander verträglich sind

Gesucht Ein Typ IShape, der Objekte aller Figurenklassen umfasst. D.h., die *Vereinigung* der Klassentypen.

Figuren im Klassendiagramm



Interface und Implementierung

- Die Klassentypen Dot, Square, Circle werden zu einem gemeinsamen *Interfacetyp* zusammengefasst, angedeutet durch den offenen *Generalisierungspfeil* im Diagramm.
- Er wird durch eine *Interfacedefinition* angegeben:

```
1 // geometrische Figuren
2 interface IShape { }
```

- Die Klassendefinition gibt an, ob eine Klasse zu einem Interface gehört oder nicht. Dies geschieht durch eine `implements`-Klausel.

```
1 // ein Punkt
2 class Dot implements IShape {
3   CartPt loc;
4
5   Dot(CartPt loc) {
6     this.loc = loc;
7   }
8 }
```

Weitere Implementierungen

- Ein Interface kann beliebig viele implementierende Klassen haben.

```
2  // ein Quadrat
3  class Square implements IShape {
4     CartPt loc;
5     int size;
12 }
```

```
2  // ein Kreis
3  class Circle implements IShape {
4     CartPt loc;
5     int radius;
12 }
```

Verwendung

- Square, Circle und Dot Objekte besitzen jeweils ihren Klassentyp.

```
CartPt p0 = new CartPt (0,0);
CartPt p1 = new CartPt (50,50);
CartPt p2 = new CartPt (80,80);

Square s = new Square (p0, 50);
Circle c = new Circle (p1, 30);
Dot d = new Dot (p2);
```

- Durch die `implements`-Klausel besitzen sie **zusätzlich** noch den Typ IShape.

```
IShape sh1 = new Square (p0, 50);
IShape sh2 = new Circle (p1, 30);
IShape sh3 = new Dot (p2);

IShape sh4 = s;
IShape sh5 = c;
IShape sh6 = d;
```

Typfehler

- Eine Zuweisung

$$\text{Ty var} = \textbf{new}\ \text{Cls}(\ldots)$$

ist *typkorrekt*, falls Cls ein *Subtyp* von Ty ist. Das heißt:

  - Ty ist identisch zu Cls oder
  - Cls ist definiert mit "Cls **implements** Ty"

Ty heißt dann auch *Supertyp* von Cls.
Anderenfalls liegt ein *Typfehler* vor, den Java zurückweist.

- Typkorrekte Zuweisungen

```
Square good1 = new Square (p0, 50);
IShape good2 = new Square (p1, 30);
```

- Zuweisungen mit Typfehlern

```
Square bad1 = new Circle (p0, 50);
IShape bad2 = new CartPt (20, 30);
```

Erstellen einer Vereinigung von Klassen

1. Wenn ein Datenbereich auftritt, in dem Objekte mit unterschiedlichen Attributen auftreten, so ist das ein Indiz, dass eine Vereinigung von Klassen vorliegt.
2. Erstelle zunächst das Klassendiagramm. Richte das Augenmerk zunächst auf den Entwurf der Vereinigung und verfeinere zusammengesetzte Klassen später.
3. Übersetze das Klassendiagramm in Code. Aus dem Interfacekasten wird ein Interface; die darunterliegenden Klassenkästen werden Klassen, die jeweils das Interface implementieren.
   Versehe jede Klasse mit einer kurzen Erklärung.

Objekte mit Referenzen zu Objekten der gleichen Klasse

Erstelle ein Programm, das ein Lauftagebuch führt. Der Läufer erstellt jeden Tag einen Eintrag, der den Lauf des Tages dokumentiert. Ein Eintrag besteht aus dem Datum, der zurückgelegten Entfernung, der Dauer des Laufs und einem Kommentar zum Zustand des Läufers nach dem Lauf.

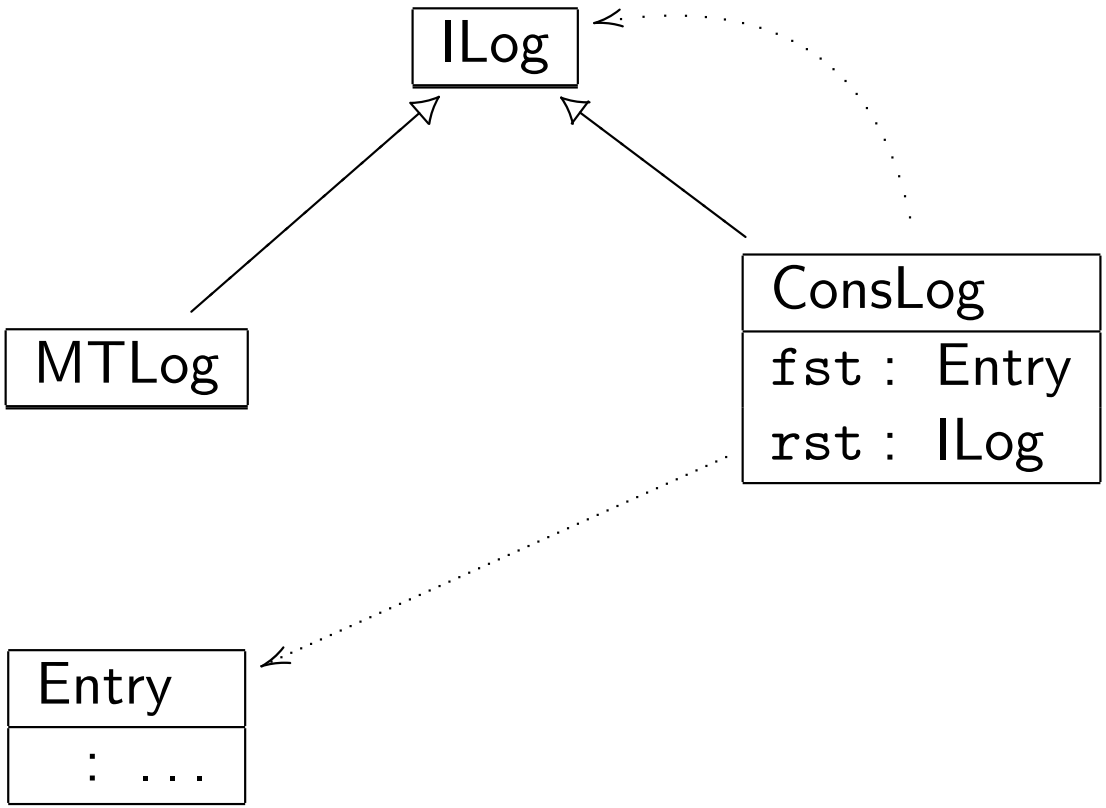
- Bereits erledigt: Klasse Entry für einzelne Einträge
- Noch zu tun: Ein Tagebuch enthält eine **beliebige Anzahl** von Entry-Objekten

$\Rightarrow$ Wunsch: repräsentiere das Tagebuch durch eine **Liste** von Einträgen

Entwurf einer Liste von Entry

- Eine Liste von Einträgen ist entweder
  - leer **oder**
  - besteht aus einem Eintrag und einer restlichen Liste von Einträgen
- Das Wort "**oder**" weist auf eine Vereinigung von Klassen hin
- Repräsentiere also eine Liste von Einträgen durch ein Interface ILog, das als Vereinigung zweier Klassen dient, die je für die leere bzw nicht-leere Liste stehen.
  - leer $\longrightarrow$ Klasse MTLog
  - nicht-leer $\longrightarrow$ Klasse ConsLog

Klassendiagramm zur Liste von Entry



Implementierung von ILog

```
1 // Lauftagebuch
2 interface ILog {}
```

```
1 // leeres Tagebuch
2 class MTLog implements ILog {
3   MTLog() {}
4 }
```

```
1  // Listenglied im Lauftagebuch
2  class ConsLog implements ILog {
3    Entry fst;
4    ILog rst;
5
6    ConsLog(Entry fst, ILog rst) {
7      this.fst = fst;
8      this.rst = rst;
9    }
10 }
```

Beispiel: ein Tagebuch

- Beispieltagebuch
  - am 5. Juni 2003, 8.5 km in 27 Minuten, gut
  - am 6. Juni 2003, 4.5 km in 24 Minuten, müde
  - am 23. Juni 2003, 42.2 km in 150 Minuten, erschöpft
- . . . zunächst die einzelnen Einträge als Objekte

```
Entry e1 = new Entry (new Date (5,6,2003), 8.5, 27, "gut");
Entry e2 = new Entry (new Date (6,6,2003), 4.5, 24, "müde");
Entry e3 = new Entry (new Date (23,6,2003), 42.2, 150, "erschöpft");
```

- . . . Aufbau der Liste. Ergebnis in `i4`.

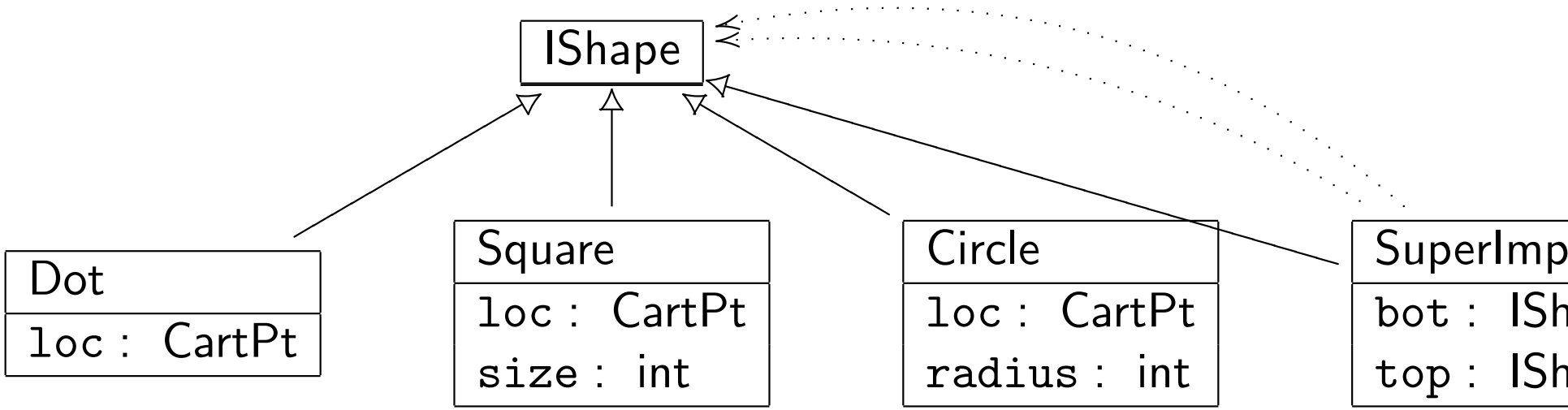
```
ILog i1 = new MTLog ();
ILog i2 = new ConsLog (e1, i1);
ILog i3 = new ConsLog (e2, i2);
ILog i4 = new ConsLog (e3, i3);
```

Allgemeine Baumstruktur

Ein Zeichenprogramm kennt mindestens drei Arten von Figuren: Punkte, Quadrate und Kreise. Darüber hinaus kann es auch mit Kombinationen von Figuren arbeiten: aus je zwei Figuren kann durch Übereinanderlegen eine neue Figur erzeugt werden.

- Neue Alternative für Figuren wird durch neue Implementierungsklasse von IShape definiert.
- Die neue Klasse heißt SuperImp für superimposition (Überlagerung).

Erweiterte Figuren im Klassendiagramm



Implementierung

```
1  // Überlagerung zweier Figuren
2  class SuperImp implements IShape {
3     IShape bot;
4     IShape top;
5
6     SuperImp(IShape bot, IShape top) {
7        this.bot = bot;
8        this.top = top;
9     }
10 }
```

Einige Beispiele

```
1   CartPt cp1 = new CartPt (100,200);
2   CartPt cp2 = new CartPt (20, 50);
3   CartPt cp3 = new CartPt (0,0);
4
5   Square s1 = new Square (cp1, 40);
6   IShape s2 = new Square (cp2, 30);
7   Circle c1 = new Circle (cp3, 20);
8
9   IShape sh1 = new SuperImp (c1, s1);
10  IShape sh2 = new SuperImp (s2, new Square (cp1, 300));
11  IShape sh3 = new SuperImp (s1, sh2);
```

- Z6 weist ein Square-Objekt einer IShape-Variable zu
- Z9 **übergibt Objekte von einem Subtyp anstelle des erwarteten Typs**. Der Konstruktor von SuperImp erwartet Argumente von Typ IShape und erhält stattdessen Objekte der Subtypen Square bzw. Circle.
- Z10, zweites Argument, gleiches Phänomen
- **alle Zuweisungen sind typkorrekt!**

Entwurf von Klassenhierarchien

Vorgehensweise in vier Schritten

- Problemanalyse
- Klassendiagramme
- Klassendefinitionen
- Beispiele

Problemanalyse

- Problemstellung liegt vor (Auftraggeber)
- Welche Informationen werden benötigt? Suche z.B. nach Hauptwörtern in der Problemstellung
- Gruppiere Informationen, die gemeinsam oder ähnlich verarbeitet werden müssen
- Ergebnis:
  - Liste von Klassennamen mit kurzer Beschreibung
  - informelle Beispiele

Klassendiagramme

- Erstelle zu jedem Klassennamen aus der Beschreibung einen entsprechenden Kasten im Klassendiagramm
- Folgende Auswahlmöglichkeiten ergeben sich dabei
  - *Primitiver Datentyp*, falls offensichtliche Verbindung zwischen Information und einem vorgegebenen Datentyp besteht.
  - *Neue Klasse*, falls eine zusammengesetzte Information repräsentiert werden soll. Jede Teilinformation wird zu einem *Attribut* dieser Klasse.
  - Verweis auf eine andere Klasse, falls eine Teilinformation selbst zusammengesetzt ist.
  - *Vereinigung von Klassen*, falls eine Information in disjunkte Kategorien aufgeteilt werden kann.
  - *Rekursive Klasse* (d.h. eine Klasse durch deren Attribute Objekte derselben Klasse erreichbar sind), falls eine Information aus einer unbekannten und unbeschränkten Anzahl von Teilinformationen besteht.
- Dieser Prozess muss ggf. wiederholt werden bis das Ergebnis zufriedenstellend ist.

Klassendefinitionen

- Die Übertragung von Klassendiagrammen in Klassendefinition erfolgt schematisch:
  - Für jede Art von Kasten im Diagramm gibt eine Regel die passende Definitionsform an.
  - Für jeden Pfeil im Diagramm gibt eine Regel die notwendigen Anpassungen an.
    Bsp: der Generalisierungspfeil entspricht einer `implements`-Klausel.
- Jede Klassendefinition benötigt eine kurze Erklärung.

Beispiele

- Übersetze die informellen Beispiele in Objektstrukturen.
- Definiere Objektstrukturen und interpretiere sie als Information.
- Sammeln Sie aussagekräftige Objektstrukturen, die **nicht** sinnvoll als Information interpretiert werden können.