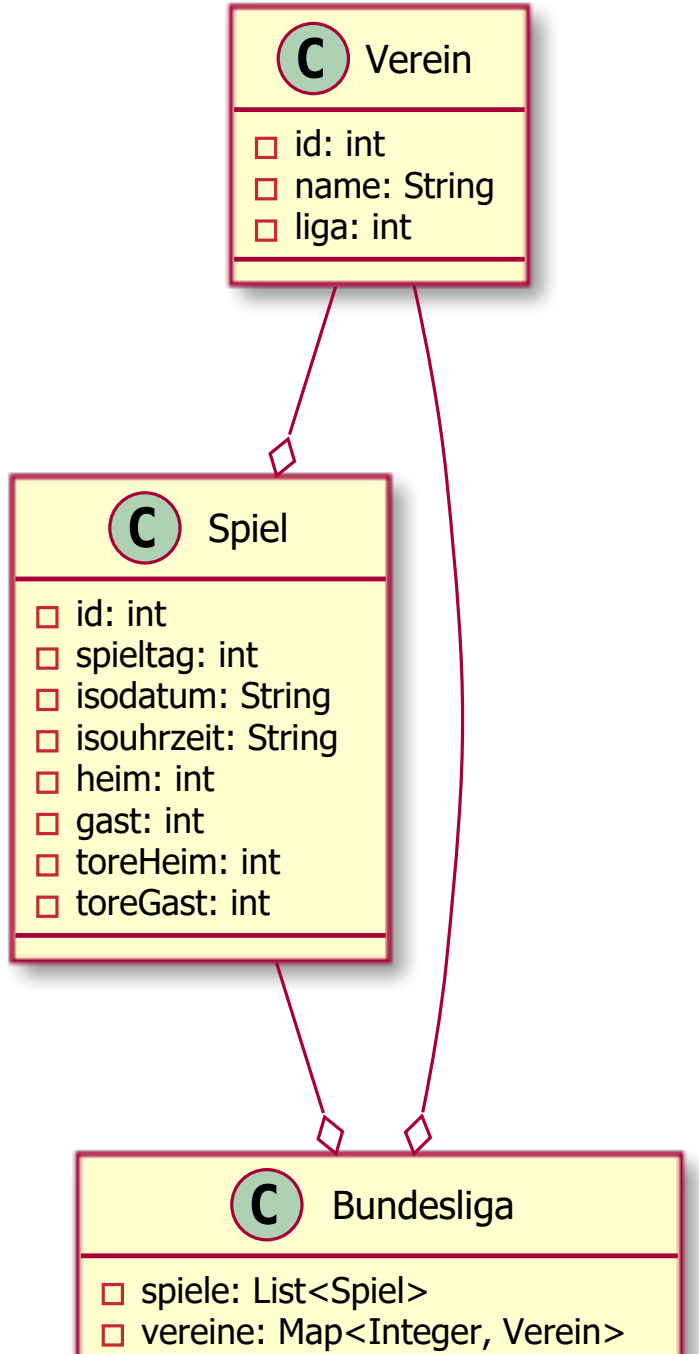
Objektorientiertes Programmieren (OOP)

10-Datenverarbeitung

Dr. Marcel Tilly

Bachelor Wirtschaftsinformatik, Fakultät Informatik

Datenmodell



Datenmodell

Verein

V_ID	Name	Liga
1	FC Bayern München	1
2	FC Schalke 04	1

Spiel

Spiel_ID	Spieltag	Datum	Uhrzeit	Heim	Gast	Tore_Heim
1	1	2017-08-18	20:30:00	1	5	3
2	1	2017-08-19	15:30:00	7	12	1

Grundoperationen

Sortieren

- Vereinsliste aufsteigend nach Vereinsname?
- Zuerst nach Liga, dann nach Vereinsname?

Grundoperationen

Filtern

- **Gegeben**: Liste aller Vereine
- **Gesucht**: Liste aller 2.-Liga-Vereine

...und sortiert nach Vereinsname?

Grundoperationen

Abbilden

- **Gegeben**: Liste der Spiele
- **Gesucht**: Liste der Spielpaarungen (Datum, Heim, Gast)

Grundoperationen

Reduzieren

- **Gegeben**: Liste der Spiele
- **Gesucht**: Wie viele Tore wurden insgesamt geschossen?

Aufgaben

Torstatistiken

1. Wie viele Tore fallen durchschnittlich in jedem Spiel?
2. Wie viele Tore fallen durchschnittlich in einem Spiel der 1. Liga?
3. Wie viele Tore fallen durchschnittlich an einem Spieltag der 2. Liga?
4. Stimmt es, dass in den Nachmittagsspielen (15:30:00) im Schnitt mehr Tore fallen, wie in den Abendspielen?
5. Stimmt es, dass Vereine der 3. Liga zuhause im Schnitt mehr Tore schießen als auswärts?

Aufgaben

Vereine

1. Wie viele Tore hat der FC Bayern München (Verein 1) erzielt?
2. Wie viele Tore hat der FC Schalke 04 (Verein 2) kassiert?
3. Wie viele Punkte hat der 1. FC Nürnberg (Verein 20)? Ein Sieg zählt 3 Punkte, ein Unentschieden 1, eine Niederlage 0 Punkte.
4. Was ist das Torverhältnis des VfL Bochum (Verein 26), also die Rate von erzielten zu kassierten Toren?
5. Welche drei Vereine haben die meisten Tore zuhause geschossen, und wie viele?
6. Welcher Verein hat die wenigsten Tore auswärts geschossen, und wie viele?

Aufgaben

1. Liga
   1. Wie ist der aktuelle Tabellenstand? Die Tabelle wird als Vereinsname, gefolgt von Punkten und Torverhältnis definiert.
   2. Wie ist der Tabellenstand nach dem 10. Spieltag?
   3. Wie ist der Tabellenplatzverlauf des Hamburger SV (Verein 18) über alle 32 Spieltage?
   4. Wer hat die Rote Laterne in jeweils der 1., 2. und 3. Liga?
   5. Wie hat sich die Tabellenführung bis zum 32. Spieltag entwickelt? Die Liste soll nicht einen Verein pro Spieltag haben, sondern immer nur wenn ein neuer Verein die Führung übernimmt (d.h. jeder Eintrag ist von dem vorherigen verschieden).

Verallgemeinerungen

Wie kann man die Grundoperationen (Iterieren, Filtern, Sortieren, Abbilden, Reduzieren) verallgemeinern?

$\rightarrow$ Trennung von Iterations- und Verarbeitungscode!

Iterieren (zum Verbrauch)

```
1 interface Consumer<T> {
2       void accept(T t);
3 }
4
5 static <T> void fuerJedes(Collection<T> coll, Consumer<T> cons) {
6       for (T t : coll)
7             cons.accept(t);
8 }
```

Beispiel: Jedes Element auf System.out ausgeben.

```
1 List<Integer> li = Arrays.asList(3, 1, 3, 3, 7);
2 fuerJedes(li, new Consumer<Integer>() {
3       public void accept(Integer i) {
4             System.out.println(i);
5       }
6 });
```

Filtern

```
1  interface Predicate<T> {
2      boolean test(T t);
3  }
4
5  static <T> List<T> filtern(Collection<T> liste, Predicate<T> pred)
       {
6      // neue Liste erstellen
7      List<T> gefiltert = new LinkedList<>();
8
9      // iterieren...
10     for (T v : liste) {
11         // Bedingung prüfen...
12         if (pred.test(v))
13             gefiltert.add(v);
14     }
15
16     return gefiltert;
17 }
```

Abbilden

```
1  interface Function<T, R> {
2      R apply(T t);
3  }
4
5
6  static <T, R> List<R> abbilden(List<T> liste, Function<T, R>
       func) {
7      List<R> abgebildet = new LinkedList<>();
8
9      for (T v : liste)
10         abgebildet.add(func.apply(v));
11
12     return abgebildet;
13 }
```

Reduzieren (Version 1)

```
 1 interface BinaryOperator<T> {
 2       T apply(T a, T b);
 3 }
 4
 5
 6 static <T> T reduzieren(Collection<T> liste, T identity,
       BinaryOperator<T> op) {
 7       T a = identity;
 8       for (T t : liste)
 9             a = op.apply(a, t);
10       return a;
11 }
```

Reduzieren (Version 2)

```
 1 interface BiFunction<A, B, C> {
 2       C apply(A a, B b);
 3 }
 4
 5
 6 static <T, U> U reduzieren(Collection<T> liste, U identity,
       BiFunction<U, T, U> op) {
 7       U a = identity;
 8
 9       for (T t : liste)
10             a = op.apply(a, t);
11
12       return a;
13
14 }
```

Lambdaausdrücke

```
1 Predicate<Integer> istGerade = new Predicate<Integer> () {
2       @Override
3       public boolean test(Integer i) {
4             return i % 2 == 0;
5       }
6 };
```

Bedingung:

- Interface mit genau einer Methode
- Interface annotiert mit @FunctionalInterface

```
1 Predicate<Integer> istGerade = (i) -> { return i % 2 == 0; };
2
3 // wenn 1 Argument, 1 Anweisung: noch kompakter!
4 Predicate<Integer> istGerade = i -> i % 2 == 0;
```

Lambdaausdrücke

- drastische **Reduktion** von Code!
- drastische **Steigerung** der Lesbarkeit!

Was macht folgender Code?

```
1 List<Integer> li = Arrays.asList(3, 2, 4, 1, 9, 7, 6);
2
3 for (Integer i) {
4       if (i < 8) {
5             double c = Math.sqrt(i);
6             if (c > 2)
7                   System.out.println(c);
8       }
9 }
```

Lambdaausdrücke

Mit Grundoperationen

```
1 List<Integer> li = Arrays.asList(3, 2, 4, 1, 9, 7, 6);
2
3 li = filtern(li, i -> i < 5);
4 List<Double> ld = abbilden(li, i -> Math.sqrt(i));
5 fuerJedes(li, i -> System.out.println(i));
```

Oder als Stream

```
1 Arrays.asList(3, 2, 4, 1, 9, 7, 6).stream()
2       .filter(i -> i < 5)
3       .map(Math::sqrt)
4       .filter(d -> d > 2)
5       .forEach(System.out::println);
```

Ist das nicht wirklich kurz und prägnant?!