Klassediagrammer (III)

Tabeller og qualified associations

Michael R. Hansen

mrh@imm.dtu.dk

Informatics and Mathematical Modelling

Technical University of Denmark

Oversigt

- Tabelbegrebet
- Anvendelser
- Maps, HashMaps

Tabeller (I)

En *tabel* fra en mængde A til en mængde B, er

- en endelig delmængde A_1 af A og
- en funktion $t\colon A_1 \to B$

En tabel vises ofte på tabular form:

a_1	b_1
a_2	b_2
$\vdots$	$\vdots$
a_n	b_n

- $A_1 = \{a_1, \ldots, a_n\}$
- $t(a_i) = b_i$, for $1 \leq i \leq n$

- A_1 kaldes t's *domæne*
- $a_i \in A_1$ kaldes en *nøgle*
- $b_i = t(a_i)$ kaldes en *værdi*
- (a_i, b_i) kaldes en *indgang*

Tabeller

For mange anvendelser findes naturlige nøgler:

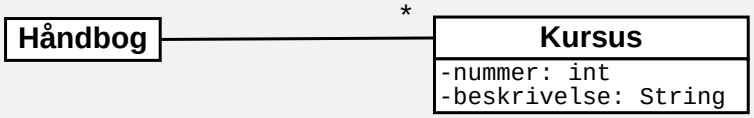
- Studienummer
- Kursusnummer
- CPR nummer
- Varekoder
- Medlemsnumre

En kursushåndbog kan, for eksempel, modelleres som en tabel fra kursusnumre til kursusbeskrivelser.

- Tabeller er nyttige som modelleringsredskab
- Tabeller understøttes af gode programbiblioteker

Eksempel: Studiehåndbog (I)

En studiehåndbog beskriver en samling kurser, hvor hvert kursus beskrives ved et nummer, skemaplacering, formål, forudsætningskurser, osv.



- ikke synligt at et kursus er entydigt bestemt ved et kursusnummer

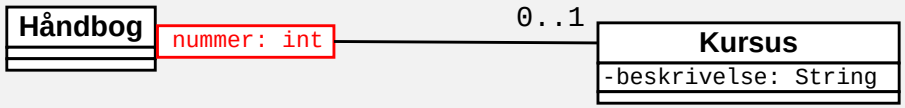
utilfredsstillende modellering

Eksempel: Studiehåndbog (II)

En studiehåndbog beskriver en samling kurser, hvor hvert kursus er entydigt bestemt ved et nummer, og . . . .

Modelleres vha en *kvalificeret* (eng. *qualified*) association

- en håndbog har 0-1 kursus for hvert nummer



Den kvalificerede association kan realiseres ved at `Håndbog` kan indeholde en tabel, hvor

- numre er nøgler, og
- kurser er værdier

Tabel-Interfacet: `Map`

I Java er der forskellige biblioteker for tabeller, der implementerer interfacet `Map`, hvori bl.a. følgende operationer, er specificeret

`int size()` : $|A_1|$

`boolean containsKey(Object k)` : $k \in A_1$

`Object get(Object k)` : $t(k)$, hvis $k \in A_1$, og `null` ellers

`Object put(Object k, Object v)` :

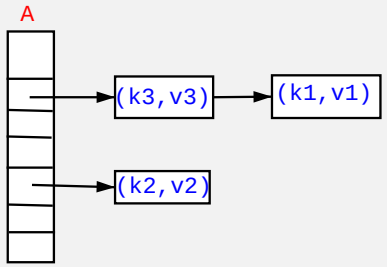
$$A_1' = A_1 \cup \{k\}, \quad t'(k_1) = \begin{cases} v & \text{hvis } k = k_1 \\ t(k_1) & \text{ellers} \end{cases}$$

returnerer $t(k)$, hvis $k \in A_1$, og ellers `null`

hvor $t : A \to B$ og $A_1 \subseteq A$ er tabellen før operationen
og $t' : A \to B$ og $A_1' \subseteq A$ er tabellen efter, hvis der er sidevirkning.

Tabelbiblioteket: `HashMap`

En *HashMap* er en datastruktur der består af et array A, hvor A's indgange er en liste af tabelindgange.



- en operation `int hashCode()` på nøgler bestemmer position i A for en indgang (k, v)
- nøglers `equals` operation benyttes ved søgning i listerne
- HashTabeller har en maksimal fyldningsgrad. Default er 75 %. Overskrides max. så udvides A.

- hvis `k1.equals(k2)`, så `k1.hashCode()==k2.hashCode()`
- for en lang række operationer, er den gennemsnitlige køretid for en enkelt operation konstant

godt default valg

Klasse: Haandbog

- operationer på håndbogen implementeres direkte ved HashMap-operationer

```
import java.util.*;
public class Haandbog {

    private Map<Integer,Kursus> katalog;

    public Haandbog(){ katalog = new HashMap<Integer,Kursus>();}

    public void opret(int nr, Kursus k){ katalog.put(nr, k); }

    public Kursus find(int nr){ return katalog.get(nr); }

    public String toString(){   return "" + katalog; }
}
```


Klasse: Kurser

```
public class Kursus
{
    private String navn;
    private String indhold;

    public Kursus(String n, String i)
    {
        navn = n;
        indhold = i;
    }

    public String toString()
    {
        return "Navn: " + navn + "\n" + indhold + "\n";
    }
}
```


Hovedprogram

```
Kursus k3333 = new Kursus("Programmering", "Java, C++, UML");
Kursus k4444 = new Kursus("Matematik", "Funktioner, relationer");
Kursus k5555 = new Kursus("Fysik", "Newtons love");
Kursus k6666 = new Kursus("Kemi", "Gasser");

Haandbog hb = new Haandbog();

hb.opret(3333,k3333);
hb.opret(4444,k4444);
hb.opret(5555,k5555);
hb.opret(6666,k6666);

System.out.println( hb );

System.out.println( hb.find(4444));
```


Udskrift

```
{3333=Navn: Programmering
Java, C++, UML
, 4444=Navn: Matematik
Funktioner, relationer
, 6666=Navn: Kemi
Gasser
, 5555=Navn: Fysik
Newtons love
}
Navn: Matematik
Funktioner, relationer
```

KasseApparat: Problemstilling

Et elektronisk kasseapparat indeholder et katalog, hvor navn og pris for en vare knyttes sammen med en varekode.

Et indkøb består at en række punkter, hvor det angives hvor mange styk af en given vare der købes.

En regning skal beskrive enkeltindkøb af varer og angive en totalpris.

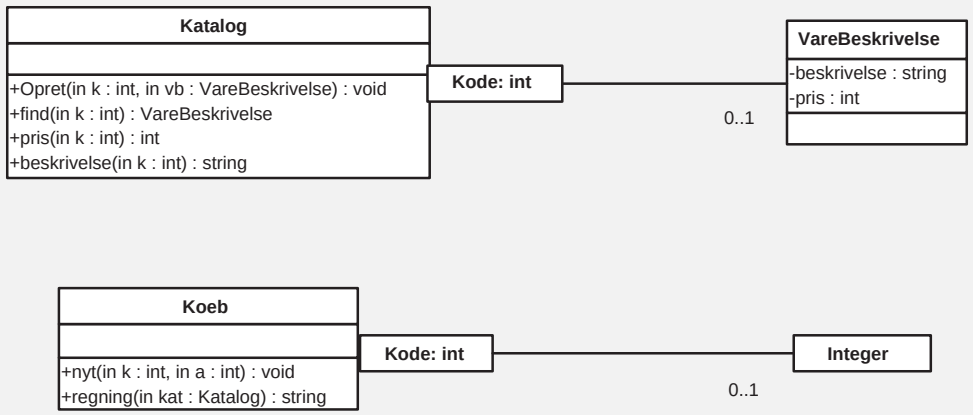
KasseApparat: Problemstilling

Et elektronisk kasseapparat indeholder et katalog, hvor navn og pris for en vare knyttes sammen med en varekode.

Et indkøb består at en række punkter, hvor det angives hvor mange styk af en given vare der købes.

En regning skal beskrive enkeltindkøb af varer og angive en totalpris.

Klassediagram



KasseApparat: Hovedprogram

Opret varekatalog og indkøb, og udskriv regning

```
Katalog katalog = new Katalog();
katalog.opret(1, new VareBeskrivelse("is",   10));
katalog.opret(2, new VareBeskrivelse("fisk", 20));

Koeb koeb = new Koeb(1, 3);
koeb.nyt(2, 4);
koeb.nyt(1, 3);

System.out.println(koeb.regning(katalog));
```

Udskrift:

```
4       fisk     :       80
6       is       :       60
Total = 140
```

Klasse: Katalog

```
public class Katalog {
    private Map<Integer,VareBeskrivelse> map;

    public Katalog() { map = new HashMap<Integer,VareBeskrivelse>();}

    public void opret(int k, VareBeskrivelse vb) { map.put(k, vb);}

    public VareBeskrivelse find(int k) throws KatalogException
      { if (! map.containsKey(k))
             throw new KatalogException(" findes ikke");
        return map.get(k);
      }

      public int pris(int k)
      { return (map.get(k)).getPris();}

      public String beskrivelse(int k)
      { return (map.get(k)).getBeskrivelse();}

      public String toString() { return "" + map;}
}
```


Klasse: VareBeskrivelse

```
public class VareBeskrivelse
{ private String beskrivelse;
  private int    pris;

  public VareBeskrivelse(String b, int p)
  {
    beskrivelse = b;
    pris = p;
  }

  public int getPris() { return pris; }

  public String getBeskrivelse() { return beskrivelse; }

  public String toString()
  {
    return beskrivelse + " koster " + pris + " kr.";
  }
}
```


Klasse: Koeb (I)

```
public class Koeb {

  private Map<Integer,Integer> map;

  public Koeb() { map = new HashMap<Integer,Integer>(); }

  public Koeb(int k, int a)
      { map = new HashMap<Integer,Integer>();
        nyt(k, a);
      }

  public void nyt(int k, int a)
     { int i = a;
       if (map.containsKey(k))
         i += (map.get(k));

       map.put(k, i);
      }

  public String toString() { return map + "";}
}
```


Samlinger

For at lave en regning skal alle indgange i et indkøb behandles.
Hertil benyttes samlinger og evt. iteratorer.

- der laves en *samling* (eng. *collection*) af tabelindgange

  ```
  Set map.entrySet()
  ```

  hvor `Set` er en type for samlingen: mængde, og `entrySet` returnerer samlingen af tabelindgange.

- Elementerne har type `Map.Entry`.

  På denne type findes to operationer:

  ```
  Object getKey()

  Object getValue()
  ```

Klasse: Koeb (II)

```
public String regning(Katalog kat) {
   int total = 0;  int a = 0; int p = 0;
   int k;
   String res = "";


   for (Map.Entry<Integer,Integer> entry : map.entrySet())
        {
          k      = entry.getKey();
          a      = entry.getValue();

          p      = a * kat.pris(k);
          total += p;
          res   += a + "\t" + kat.beskrivelse(k) + "\t : \t " + p + "\n";
        }

        return res + "Total = " + total;
   }
```


Opsummering

- Modellering vha tabeller
- Implementering vha Maps
- mange operationer i en model kan ofte direkte implementeres ved operationer i tabelbiblioteker
- der findes mange effektive implementeringer af tabeller

  kurser i Algoritmer og Datastrukturer