Use Cases - Opfølgning på Opgave 2

Michael R. Hansen

mrh@imm.dtu.dk

Informatics and Mathematical Modelling

Technical University of Denmark

Opgaven

Ved svømmestævner måles svømmernes tider i minutter, sekunder og hundrededele af sekunder. Når svømmerne har opnået gode tider, får de en udmærkelse i form af en nål. Kravene for at opnå en given nål afhænger af køn og disciplin.

Lad os antage at der findes 8 forskellige nåle, og at kravene for herrer i disciplinen 100 bryst er givet ved

Nål	Tid
Elite	1.03:90
Guld	1.07:59
Sølvdelfin	1.12:59
Sølv	1.24:00
Bronzedelfin	1.30:00
Bronze	1.39:00
Flipper	2.03:00
Talent	2.21:00

Opgaven – fortsat

Efter hvert løb laves en resultatliste, som viser de forskellige svømmeres tider i det pågældende løb.

Resultatliste: 100 bryst, herrer

Svømmer	Tid
Mads	1.44:25
Finn	1.30:00
Aage	1.24:42
Hans	1.02:59
Sten	2.22:25

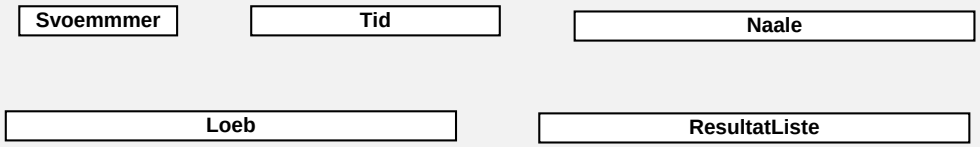
Desuden ønsker vi, at få en udskrift af hvilke nåle svømmerne i et løb kan få for deres præstationer.

```
Mads tildeles flippernaalen
Finn tildeles broncedelfinnaalen
Aage tildeles broncedelfinnaalen
Hans tildeles elitenaalen
```


Trin 1: Væsentlige begreber

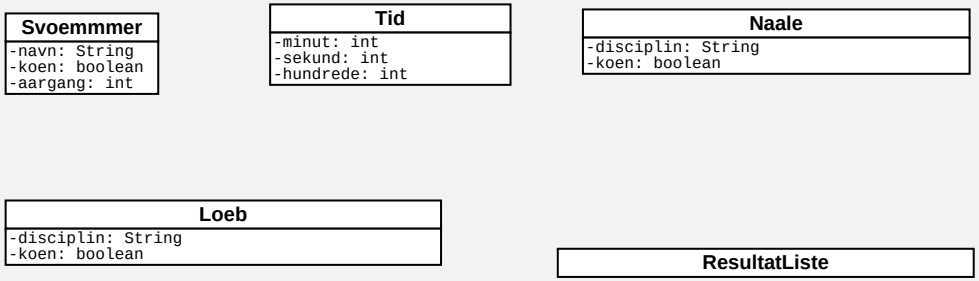
Identificer væsentlige begreber i problemstillingen, som kan være kandidater til klasser.

- "Ryd op" i begreberne, men udvælg hellere en klasse for meget end en for lidt.



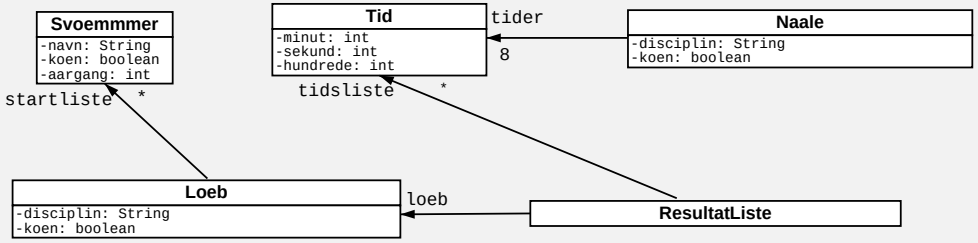
Trin 2: Attributter

Analyser begreberne for at identificere attributter

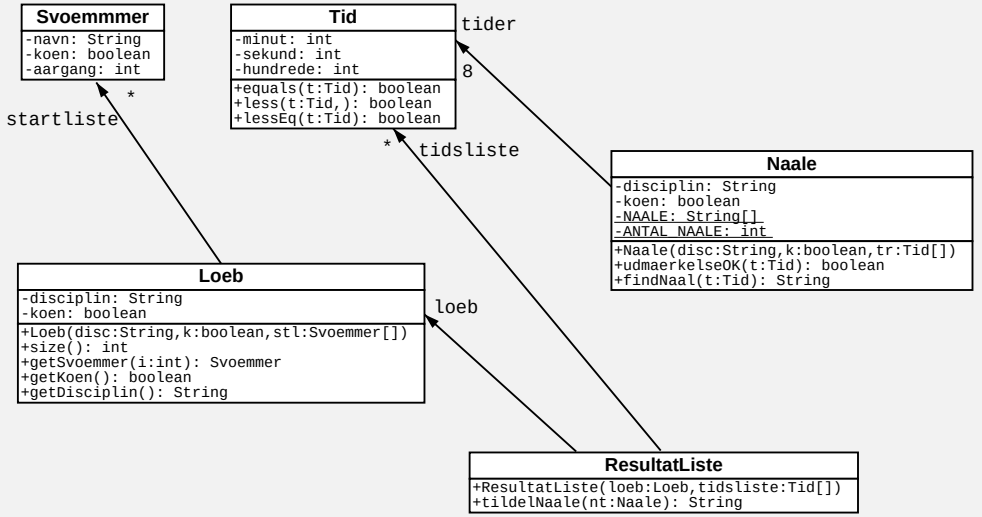


Trin 3: Associationer

Find associationer mellem klasserne



Detaljeret klassediagram



Implementering: Svoemmer

```
public class Svoemmer
{   private String navn;
    private boolean koen;
    private int aargang;

    public Svoemmer(String nv, boolean k, int a)
    {   navn    = nv;
        koen    = k;
        aargang = a;}

    public String  getNavn() { return navn; }

    public boolean getKoen() { return koen; }

    public int     getAargang() { return aargang; }

    public String  toString(){ return navn + "    Aargang: " + aargang; }
}
```

Implementering: Tid

```
public class Tid
{ private int minut;   private int sekund;   private int hundrede;

  public Tid(int m, int s, int h) throws SvoemException
  { if (0 <= m && m <= 99 && 0 <= s && s <= 59 && 0 <= h && h <= 99)
       { minut = m;  sekund = s;  hundrede = h;}
    else throw new SvoemException( "Ulovlig tid: " + m + "." + s + "." + h);}

  public boolean less(Tid t)
  { return  minut < t.getMinut()
          || (minut == t.getMinut() && sekund < t.getSekund())
          || (minut == t.getMinut() && sekund == t.getSekund()
                                    && hundrede < t.getHundrede()); }

  public String toString()
  { return cvt(minut, " ") + "." +  cvt(sekund, "0") + ":" + cvt(hundrede, "0");}

  private static String cvt(int i, String s)
  { return (i>9 ? "" : s) + i; }

...
```


Implementering: Naale

```
public class Naale
{ private String disciplin;  private boolean koen;

  private Tid[] tider;
  private static final String[] NAALE = {"Elite", "Guld", ...};
  private static final int ANTAL_NAALE = NAALE.length;

  public Naale(String disc, boolean k, Tid[] tr) throws SvoemException
   { if (tr.length == ANTAL_NAALE)  ...}

  public boolean udmaerkelseOK(Tid t){ return t.lessEq(tider[ANTAL_NAALE-1]);}

  public String findNaal(Tid t) throws SvoemException
   { if (udmaerkelseOK(t))
       {  int i = 0;
          while (tider[i].less(t))
              i++ ;
          return NAALE[i];}
      else  throw new SvoemException("tiden " + t + " giver ikke en naal"); }

  ...
```


Implementering: Loeb

```
public class Loeb
{  private String disciplin;
   private boolean koen;
   private Svoemmer[] startliste;

   public Loeb(String disc, boolean k, Svoemmer[] stl) throws SvoemException
   {  for (int i=0; i<stl.length; i++)
          if (stl[i].getKoen() != k) throw new SvoemException("svoemmer  ...");

      disciplin  = disc;  koen = k;  startliste = stl; }

   public String toString()
   {  String res = "Startliste i disciplinen: " + disciplin
                 + " for " + (koen ? "damer" : "herrer") + "\n";

      for (int i=0; i<startliste.length; i++)
                res += startliste[i] + "\n";

      return res; }
...
}
```


Implementering: ResultatListe

```
public class ResultatListe
{ private Loeb  loeb;
  private Tid[] tidsliste;
  ...

  public String tildelNaale(Naale nt) throws SvoemException
  { if (loeb.getDisciplin() != nt.getDisciplin()
        || loeb.getKoen() != nt.getKoen())
            throw new SvoemException("disciplin eller koen passer ikke");

    String res = "";

    for (int i=0; i<tidsliste.length; i++)
        if (nt.udmaerkelseOK(tidsliste[i]))
            res += loeb.getSvoemmer(i).getNavn() + " tildeles "
                   +  nt.findNaal(tidsliste[i]) + "naalen\n";

    return res;
  }
...
}
```

Hovedprogram (I)

Indfør konstanter for køn og discipliner:

```
static final boolean DAMER  = true;
static final boolean HERRER = false;

static final String BRYST100 = "100 bryst";
static final String BRYST50  = "50 bryst";
```

Et nåletidsobjekt oprettes på følgende måde:

```
Naale naaleBryst100Herrer =
   new Naale(BRYST100, HERRER,
             new Tid[]
                { new Tid(1,3,90),    new Tid(1,7,59),
                  new Tid(1,12,59), new Tid(1,24,0),
                  new Tid(1,30,0),  new Tid(1,39,0),
                  new Tid(2,3,0),    new Tid(2,21,0)
                }
            );
```


Hovedprogram (II)

Et løb oprettes på følgende måde:

```
  Loeb bryst100Herrer =
      new Loeb(BRYST100, HERRER,
               new Svoemmer[]
                 { new Svoemmer("Mads", HERRER, 87),
                   new Svoemmer("Finn", HERRER, 90),
                   new Svoemmer("Aage", HERRER, 79),
                   new Svoemmer("Hans", HERRER, 82),
                   new Svoemmer("Sten", HERRER, 89)
                 }
              );
```

En resultatliste oprettes på følgende måde:

```
  ResultatListe bryst100HerrerResultater =
      new ResultatListe(bryst100Herrer,
                        new Tid[]
                           { new Tid(1,44,25), new Tid(1,30,0),
                             new Tid(1,24,42), new Tid(1,2,59),
                             new Tid(2,22,25)
                           }
                       );
```


Udskrift

Ved udførelse af sætningen

```
System.out.println( bryst100HerrerResultater.tildelNaale(naaleBryst100Herrer) );
```

udskrives

```
Mads tildeles Flippernaalen
Finn tildeles Broncedelfinnaalen
Aage tildeles Broncedelfinnaalen
Hans tildeles Elitenaalen
```


Use Cases — Hvad skal systemet kunne?

Nogle begreber som ofte benyttes i forbindelse med use cases:

- actors – roller som brugere af systemet kan have.
  Eks.: Svømmere, træner, stævnesekretær, tilskuer, osv.
- stake holders – en betegnelse for de parter som har en interesse i projektet. bestemmer rammerne
  Eks: Klubber, forbund, svømmere, ...
- scenario – en sekvens af trin der beskriver en interaktion mellem en aktør og systemet.
  Eks.: Tilmeld svømmer.
  <find stævne, find løb, angiv svømmer, angiv starttid, OK>
- use cases – en samling af scenarier med et fælles mål.
  Ivar Jacobson 1992
  Eks.: Tilmelding svømmer. Scenarier der tager højde for om stævne, løb og svømmer findes, om svømmerens starttid er god nok, ...

Use Case Beskrivelser – ikke en del af UML

Eks. på ofte benyttes følgende form (med passende udvidelser).

Tilmeld Svømmer

Main Success Scenario:

1. find stævne
2. find løb
3. angiv svømmer
4. angiv starttid
5. systemet bekræfter tilmelding

Extensions:

1a. tilmelding afbrydes, stævnet findes ikke

2a. tilmelding afbrydes, løbet findes ikke

4a. tilmelding afbrydes, starttid ikke god nok

Bemærk brugen af andre use cases: find stævne og find løb.

Use Case Diagrammer

Viser actors, use cases og deres indbyrdes relationer.

Use Cases

- is a technique for capturing functionality $\rightarrow$ base for class modelling
- base for making test cases