Innehåll

- OOP snabbintroduktion
- Programvaruutveckling och programmering
- Datatyper
- Uttryck
- Satser
- Arv, komposition, association
- Metodik och klassdesign
- Design (CRC)
- Fält
- Undantag
- In-/utmatning och filer
- Grafik
- GUI:s
- Applets vs applikationer
- Rekursion
- Interfaces och sortering
- Noggrannheten i beräkningar

Program utveckling sker i faser

- Här: starkt förenklat version
- Passar bara mindre projekt

- Fyra faser (delmoment):
  - Fastställa och analysera förutsättningarna/kraven
  - Skapa en design
  - Implementera koden
  - Testa implementationen

Program utveckling ...

- Fastställa och analysera förutsättningarna/kraven
  - Vad ska göras?
  - Hur berörs användarytan?
  - Vilka begränsningar finns?
  - Gör modeller/utkast
  - *Undvik att fundera på implementationen*

Program utveckling ...

- Skapa en design
  - Bestäm klasser, objekt och metoder som behövs
  - Bestäm algoritmer för problemlösningen
  - Algoritmer kan uttryckas i *pseudokod*
  - I princip oberoende av programmeringsspråk

- Designa för återanvändning ?
  - Svårare göra generella klasser
  - Kan löna sig i framtiden
  - Återanvändning en stor anledning till objektorienteringens popularitet

Program utveckling ...

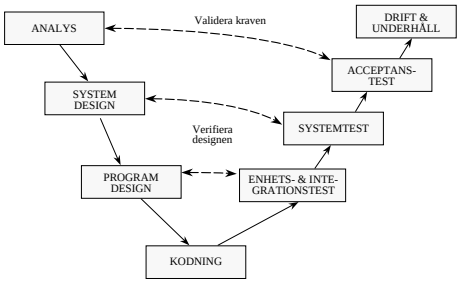
- Implementera koden
  - Översättning av design till källkod
  - Alla viktiga beslut tas vid analys och design
  - Implementationen fokuserar på kod-detaljer

Program utveckling ...

- Testa implementationen
  - Tester måste konstrueras för extremer, svagheter och gränsfall
  - Dokumenteras

- Underhåll av programsystem
  - Står för omkring 80% av tidsåtgången
  - Viktigt med program som är enkla att modifiera, rätta, bygga ut

V modellen



Vad kännetecknar en god klass

- En odelad, väldefinierad abstraktion
- Uppgiften kan beskrivas kort och tydlig
- Namnet är en substantiv eller adjektiv som beskriver abstraktionen på ett adekvat sätt
- Har ett koncist och sammanhörigt gränssnitt
- Har både tillstånd och beteende
- Representerar en mängd möjliga run-time objekt

- Problemet ska delas upp i "lämpliga" klasser
- ➨ Cohesion och Coupling (sammanhörighet och koppling)
  - ➨ Metoderna i varje klass ska ha ***stark sammanhörighet***
  - ➨ Klasserna ska vara ***löst kopplade*** (oberoende av varann)

Cohesion

- Varje metod ska vara "ansvarig" för bara en uppgift
- Cohesion mäter huruvida en metod uppfyller detta krav

- Ju mer en metod fokuserar på en enda uppgift, desto
  - enklare är det att finna ett bra namn
  - enklare och förståeligare blir koden

- Metoder med stark sammanhörighet kan lättare ändras utan att andra metoder påverkas

- ➨ Det ska vara möjligt att beskriva en metod med en enkel mening med ett verb och ett objekt

Cohesion: Exempel 1

- Exempel 1:

```
public void setNameAndAge (String name, int age);
```

Bättre:

```
public void setName (String name);
public void setAge (int age);
```

Exempel 2:

```
/* Anropas en gång om året */
public void calculateHolidays();
{
   holidays += new Holidays();
   age++;
}
```

Bättre:

```
public void calculateHolidays();
public void incrementAge();
```


Cohesion: Exempel 2

- Exempel 3:

```
public void setFirstName (String name)
{
   firstName = name;
}
public void setLastName (String name)
{
   lastName = name;
   fullName = firstName + "" + lastName;
}
```

Bättre:

```
public void setFirstName (String name)
{
   firstName = name;
   fullName = firstName + "" + lastName;
}
public void setLastName (String name)
{
   lastName = name;
   fullName = firstName + "" + lastName;
}
```


Kategorier av metoder

- Konstruktorer
  - Skapa instanser
- Selektor (get-metod)
  - Returnerar information om objektets tillstånd
- Mutator (set-metod)
  - Ändra objektets tillstånd
- Iteratorer
  - Gå igenom en kollektion elementvis
  - Returnerar ett objekt och fortsätter till nästa
- Annat
  - Gör nånting

- ➨ En metod ska bara höra hemma i en kategori

Coupling

- Klasserna ska vara så oberoende som möjligt av varann
- Coupling mäter hur stark klasserna är kopplade

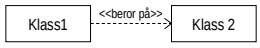
- Ju lösare klasserna är kopplade, desto
  - enklare är det att förstå en enstaka klass
  - enklare och förståeligare blir systemet som helhet

- Klasserna med lös koppling kan lättare ändras utan att andra klasser påverkas
- ➨ Systemet blir lättare att ändra
- ➨ Mera flexibilitet

PROBLEM: Koppling och återanvändning går inte ihop

Klassrelationer igen

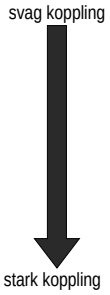
- Ju starkare relation desto starkare koppling (➜ sämre)
- Dependency



- Association

- Komposition

- Arv



`protected`

- **Problem:** Omdefinition av icke publika metoder
- `Private` metoder ärvs inte
- … men `public` ska de inte vara

- ➨ Modifieraren `protected` tillåter arv men ger bättre skydd än `public`
  - ➨ Subklasser kommer åt `public` och `protected` attribut/ metoder
  - ➨ … medan andra klasser bara kommer åt `public`

Innehåll

- OOP snabbintroduktion
- Programvaruutveckling och programmering
- Datatyper
- Uttryck
- Satser
- Arv, komposition, association
- Metodik och klassdesign
- Design (CRC)
- Fält
- Undantag
- In-/utmatning och filer
- Grafik
- GUI:s
- Applets vs applikationer
- Rekursion
- Interfaces och sortering
- Noggrannheten i beräkningar

Analysis vs. Design

- Analysis
  - Models the problem (i.e. the real world)
  - Driven by the functional requirements
  - Makes use of problem domain knowledge
  - ➨ ***Generic model***

- Design
  - Models the solution (i.e. the software)
  - Driven by the non-functional requirements
  - Makes use of solution domain knowledge
  - Refines the analysis results
  - ➨ ***Specific model***

Goals of (OO)Analysis

- Understand the problem(s) to solve
- Ask questions about the problem and the system
- Provide a basis for answering questions
- Decide what the system should/ should not do
- Make sure that the system will satisfy the needs of its users
- Define acceptance criteria
- Provide a basis for system development

Adapted from [Meyer 97], sect. 27.1.

Analysis Activities

- Scenario Planning
- Identify relevant objects
- Allocate responsibilities to objects
- Classify objects (➜ classes)
- Find relationships between classes
- Analyse class dynamics

➨ Static aspects are described in the object model (***class diagrams***)

➨ Dynamic aspects are described in ***scenarios***

Finding the Objects/ Classes

- Checklist approach
- Domain analysis
- Linguistic analysis
- Role plays (CRC cards)
- Heuristics
- Reuse of analysis models
- Experience

Linguistic Analysis

- [Abbot 83], popularised by Booch (for Ada!); later refined in [SHE 89]
- Initial list of candidates using textual analysis:
  - Noun phrases => objects / classes
  - Verb phrases => methods
  - Adjectives => attributes
  - ...
- Review analysis results
- Keep only candidates which are meaningful and useful in the problem domain
- Document results in class diagrams
- Often criticised, but works well for small documents

Linguistic Analysis: Review Candidates

- Eliminate synonyms
- Discard things that are outside the system
- Discard things that describe implementation details
- Most proper names or keys are instances of classes
- Complex noun phrases can describe hidden objects/ classes
- Look for verbs used as nouns
- Check being (is a) and having verbs (has a)
- Check property nouns, mass nouns and units of measure

➨ ***Initial class diagram***

The GSS Case Study

- Gymnastics Scoring System
- Adapted from I. White, Using the Booch Method, A Rational Approach, Benjamin/Cummings, 1994.
- Identify classes and objects
  - Get initial sources of requirements
  - Problem statement (➜ linguistic analysis)
  - Initial list of use cases (➜ define details)
  - Existing system (➜ check documents)
  - Class diagram
- Analyse and review (➜ role play, CRC cards)

The Problem Statement

A gymnastics scoring system will be developed to automate the definition, registration, scoring, and record keeping of a gymnastics season.

A gymnastics league is a group of teams that compete against each other. Each team recruits members to participate in the contests.

A typical meet consists of several contests held in the course of one day. For example, there may be a women's all-around, a women's individual, a men's all-around, and so on. There may also be junior and senior competitions. When a team enters a meet, it enters all the competitions. For each contest, each team enters the same number of members, who must compete in all parts of the competition.

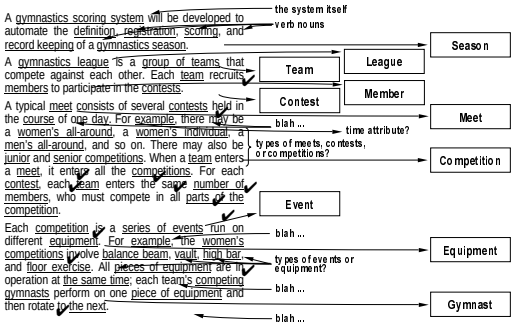
Each competition is a series of events run on different equipment. For example, the women's competitions involve balance beam, vault, high bar, and floor exercise. All pieces of equipment are in operation at the same time; each team's competing gymnasts perform on one piece of equipment and then rotate to the next.

Each event has a judging panel assigned to it. These people are qualified scorers for this event. Each judge rates each gymnast on the event and reports the score to a scorekeeper. The scorekeeper throws out the high and low scores and averages the rest. This is the gymnast's score for the event. The team score is the sum of all gymnast's scores.

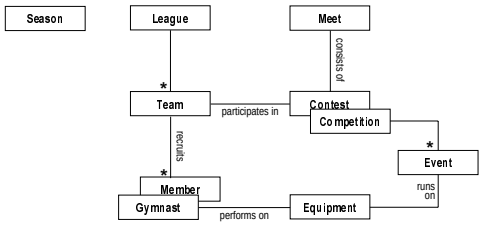
Competition scores are the sum of the scores for each of the events. Meet scores are the sum of the competition scores, and so on.

In addition to running the individual meets, the league prepares the schedule of meets for the season, ensures that qualified judges are assigned, registers teams and gymnasts, and publishes seasonal standings.

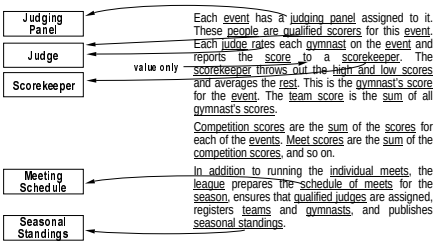
Linguistic Analysis



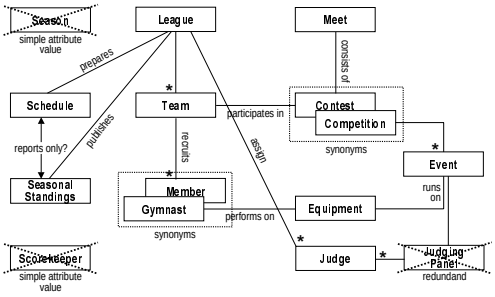
Resulting Key Classes



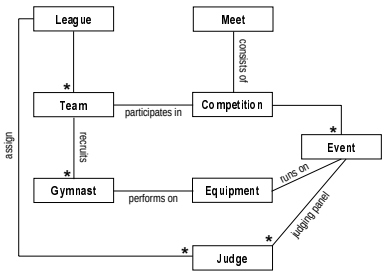
Linguistic Analysis (cont.)



Resulting Key Classes



Resulting Key Classes (cont.)



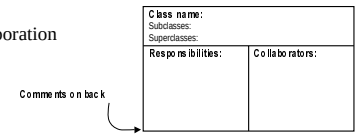
CRC Cards

- ***C***lass–***R***esponsibility–***C***ollaboration
- A ***class*** describes the behaviour of a set of objects of the same kind
- Each class takes ***responsibility*** for particular parts of the overall system behaviour
  - Knowing something (➜ state/ attribute)
  - Doing something (➜ operation/ method)
- Responsibilities are discharged by ***collaboration*** between objects

➨ Informal tool to support analysis and design "verification"

Usage of CRC Cards

- Analysis and “verification” through role-playing
  - Work through system functions
  - Consider responsibilities and collaborations
  - Exploration of alternatives
  - Easy to use
  - Portable
  - Supports team collaboration



- **OOPS!** Working model is needed as input

What is a “Good” Class

- A single, well-defined abstraction
  - Responsibilities can be described briefly and clearly
  - Named by a noun or adjective adequately describing the abstraction
  - Data abstraction or abstract machine
  - Concise and cohesive interface
- Represents a set of possible run-time objects
- Has state and behaviour

CRC Road Map (~[BeSi 97])

- Team Based Approach (3-6 persons)
- Use linguistic analysis as an input

- Brainstorm for initial designs
  - Give every voice a turn (round-robin)
  - Collect **all** ideas, discuss later (á la post-it method)

- Role-play scenarios
  - Create lists of scenarios (“test cases”)
  - Assign roles (distribute CRC cards)
  - Rehearse scenarios (“must do” first, exceptions last)
  - Correct CRC cards and revise scenarios

References

[Abbot 83] R.J. Abbot, Program Design by Informal English Descriptions, *Communications of the ACM* **26** (11), Nov 1983, 882-894.

[BeSi 97] D. Bellin, S. Simone, *The CRC Card Book*, Addison-Wesley, 1997.

[Meyer 97] B. Meyer, *Object-Oriented Software Construction*, Prentice Hall, 1997.

[SHE 89] M. Saeki, H. Horai, H. Enomoto, Software Development Process from Natural Language Specification, *Proceedings ICSE-11*, Pittsburgh, PE, USA, Mar 1989, 64-73.