

19 Designing applications

Main concepts to be covered

- OOA and OOD
- Discovering classes
- CRC cards
- Designing interfaces
- Development process models
- Modeling languages
- Modeling in UML

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 2

OOA and OOD

- **Object Oriented Analysis**
 - Identifies the entities (objects) of a system, their relationships, and cooperation.
 - Focus on "what" rather than "how".
- **Object Oriented Design**
 - Detailed design
 - Data representation, method signatures,...
 - System design
 - Platforms, languages, environment, hardware,...

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 3

Object oriented analysis

- A large and complex area.
- The verb/noun method is suitable for relatively small problems.
- CRC cards support the analysis.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 4

The verb/noun method

- The **nouns** in a description refer to 'things'.
 - A source of classes and objects.
- The **verbs** refer to actions.
 - A source of interactions between objects.
 - Actions are behavior, and hence methods.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 5

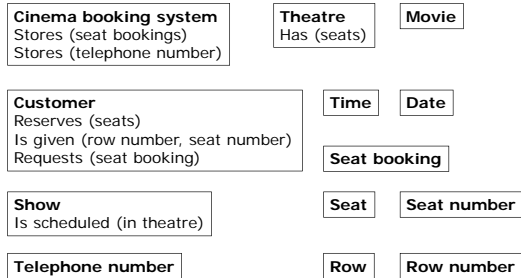
A problem description

The cinema booking system should store seat bookings for multiple theatres.
Each theatre has seats arranged in rows.
Customers can reserve seats and are given a row number and seat number.
They may request bookings of several adjoining seats.
Each booking is for a particular show (i.e., the screening of a given movie at a certain time).
Shows are at an assigned date and time, and scheduled in a theatre where they are screened.
The system stores the customers' telephone number.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 6

Nouns and verbs



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 7

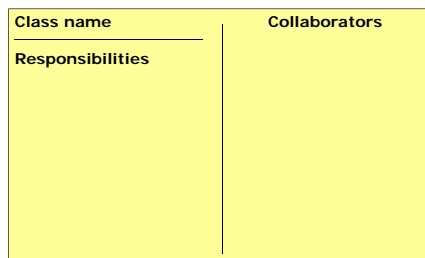
Using CRC cards

- First described by Kent Beck and Ward Cunningham.
- Each index card records:
 - A *class* name.
 - The class's *responsibilities*.
 - The class's *collaborators*.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 8

A CRC card



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 9

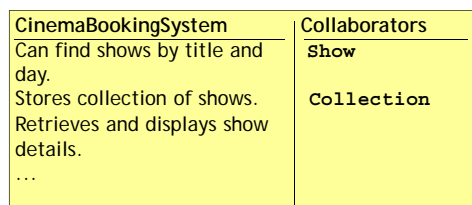
Scenarios

- An activity that the system has to carry out or support.
 - Sometimes known as *use cases*.
- Used to discover and record object interactions (collaborations).
- Can be performed as a group activity.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 10

A partial example



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 11

Scenario analysis

- Scenarios serve to check the problem description is clear and complete.
- Sufficient time should be taken over the analysis.
- The analysis will lead into design.
 - Spotting errors or omissions here will save considerable wasted effort later.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 12

Class design

- Scenario analysis helps to clarify application structure.
 - Each card maps to a class.
 - Collaborations reveal class cooperation/object interaction.
- Responsibilities reveal public methods.
 - And sometimes fields; e.g. "Stores collection ..."

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 13

Designing class interfaces

- Replay the scenarios in terms of method calls, parameters and return values.
- Note down the resulting signatures.
- Create outline classes with public-method stubs.
- Careful design is a key to successful implementation.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 14

Documentation

- Write class comments.
- Write method comments.
- Describe the overall purpose of each.
- Documenting now ensures that:
 - The focus is on *what* rather than *how*.
 - That it doesn't get forgotten!

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 15

Cooperation

- Team-working is likely to be the norm not the exception.
- Documentation is essential for team working.
- Clean O-O design, with loosely-coupled components, also supports cooperation.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 16

Prototyping

- Supports early investigation of a system.
 - Early problem identification.
- Incomplete components can be simulated.
 - E.g. always returning a fixed result.
 - Avoid random behavior which is difficult to reproduce.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 17

Development process models

- **Waterfall model**
 - Analysis
 - Design
 - Implementation
 - Unit testing
 - Integration testing
 - Delivery
- No provision for iteration.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 18

Development process models (2)

- Iterative incremental development
 - Use early prototyping.
 - Frequent client interaction.
 - Iteration over:
 - Analysis
 - Design
 - Prototype
 - Client feedback
- A growth model is the most realistic.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 19

Graphical modeling languages

- A modeling language has a *graphical syntax* (and a more or less well defined semantics).
- Graphical modeling focus on *conceptual aspects* of a design.
- OMT = Object Modeling Technique (*Michael Blaha, Jim Rumbaugh, William Premerlani*)
- Booch (*Grady Booch*)
- UML = Unified Modeling Language (*Jacobson, ...*)

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 20

UML diagram types

- Static design view
 - Class diagrams (static relations)
 - Component diagrams (modularization)
 - Deployment diagrams (run-time config.)
- Dynamic design view
 - Use case diagrams (user level behavior)
 - Scenario diagrams (object cooperation)
 - State diagrams (individual object behavior)

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 21

Class diagrams

- Class icons
- Type relationships
 - Inheritance ("is a")
 - Implementation
- Object relationships
 - Dependency
 - Association ("knows")
 - Aggregation ("has")
 - Composition ("contains")

ClassName

→

---→

-----→

→

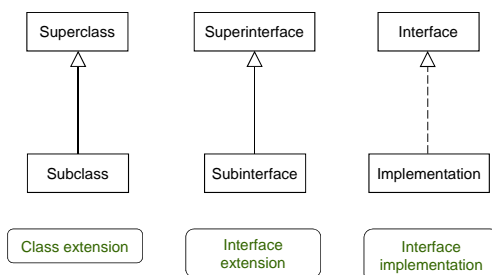
◇→

◆→

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 22

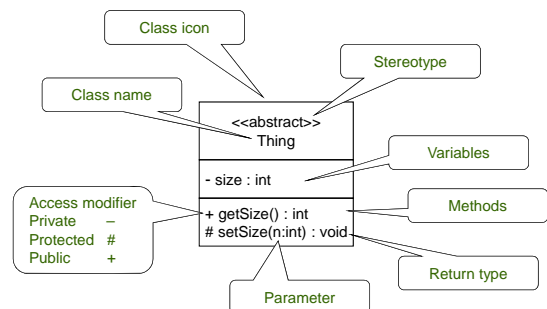
Inheritance relationships



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 23

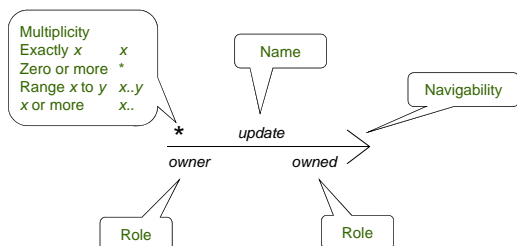
Class icons



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 24

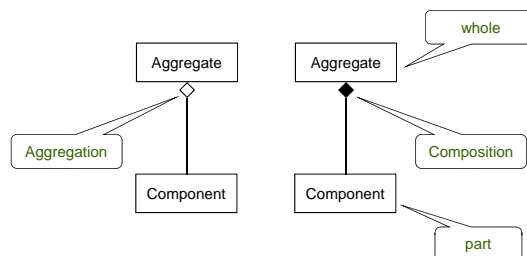
Object relation properties



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 25

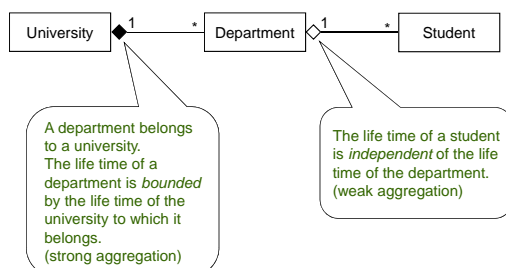
Aggregation and composition



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 26

Aggregation and composition (2)

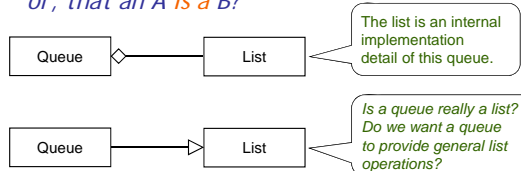


Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 27

Aggregation vs inheritance

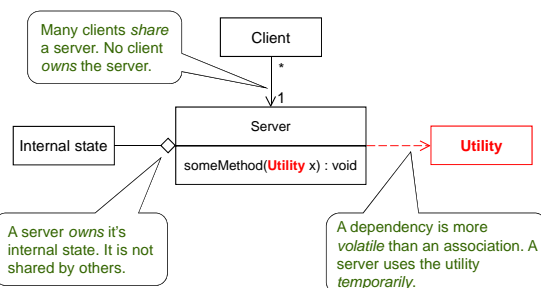
- Aggregation is often a natural alternative to inheritance.
- Ask the question:
 - Which is most natural to say, that an A *has* a B or, that an A *is* a B?



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 28

Aggregation, association and dependency relationships



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 29

Use case modeling

- Use case view**
 - Captures the behavior of a system as it appears to a user outside the *system boundary*.
 - Main inventor - Ivar Jacobson
- Actor**
 - External part that interacts with the system.
 - Idealized user: human, other system, process, ...
- Use case**
 - External system behavior, meaningful to an actor.
 - A piece of *interactive functionality* as a sequence of messages between an actor and a system.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 30

Use case diagrams

- Use case icons

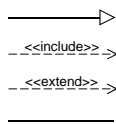


- Actor icons



- Use case relationships

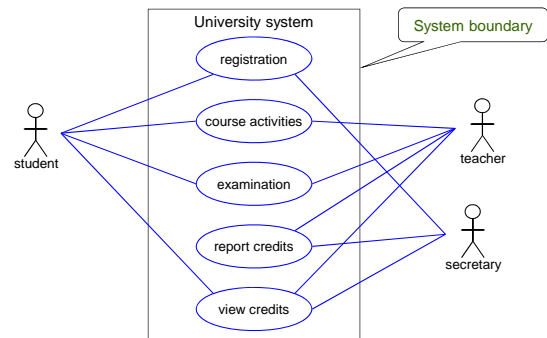
- Generalization
- Inclusion
- Extension
- Participation



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 31

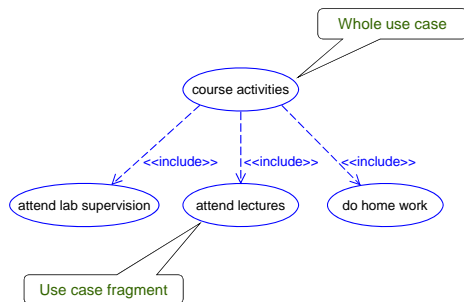
Use case diagram for a university



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 32

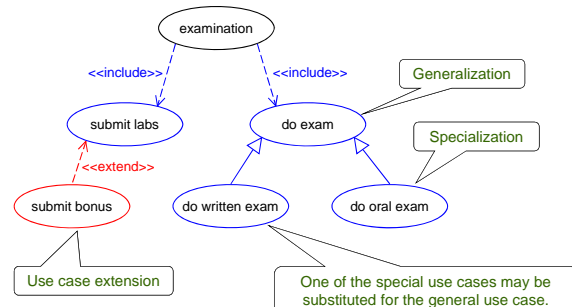
Use case parts



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 33

Generalization - specialization and extension



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 34

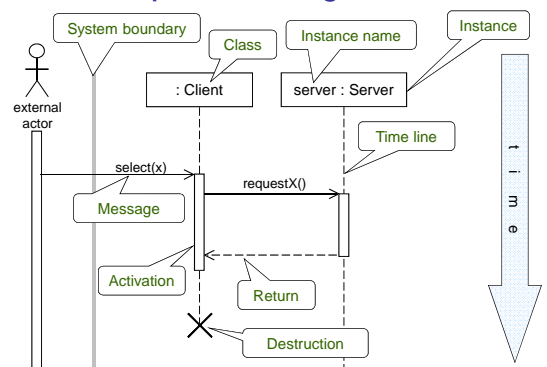
Scenario diagrams

- A scenario diagram visualizes how *cooperating objects* implement a use case, or part of a use case.
- There are two main types of scenario diagrams
 - Cooperation diagrams
 - Focus on object cooperation aspects.
 - Sequence diagrams
 - Visualize the *temporal orderings* of messages sent between cooperating objects.

Object oriented programming, DAT042, D2, 12/13, lp 1

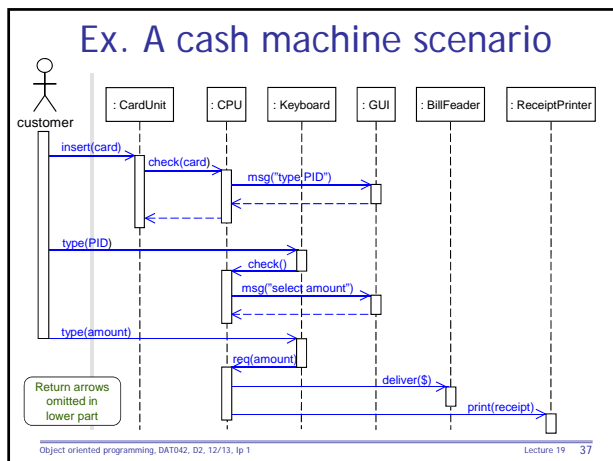
Lecture 19 35

Sequence diagrams



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 36



Review

- Class collaborations and object interactions must be identified.
 - CRC analysis supports this.
- An iterative approach to design, analysis and implementation can be beneficial.
 - Regard software systems as entities that will grow and evolve over time.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 38

Review

- Work in a way that facilitates collaboration with others.
- Design flexible, extendible class structures.
 - Being aware of existing design patterns will help you to do this.
- Continue to learn from your own and others' experiences.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 19 39