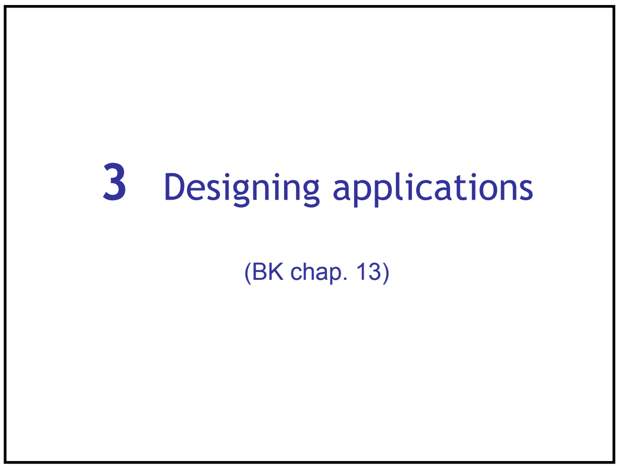
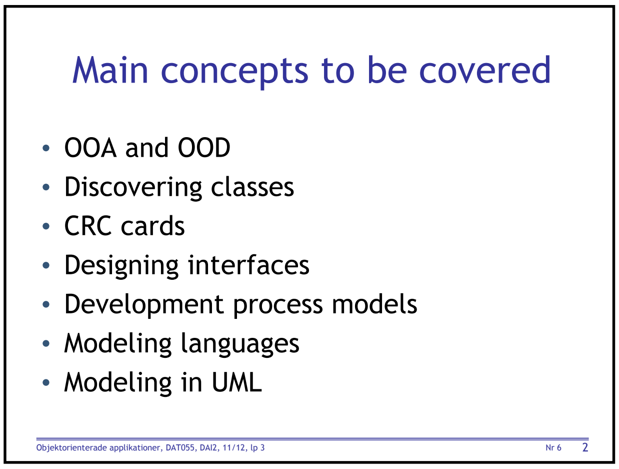


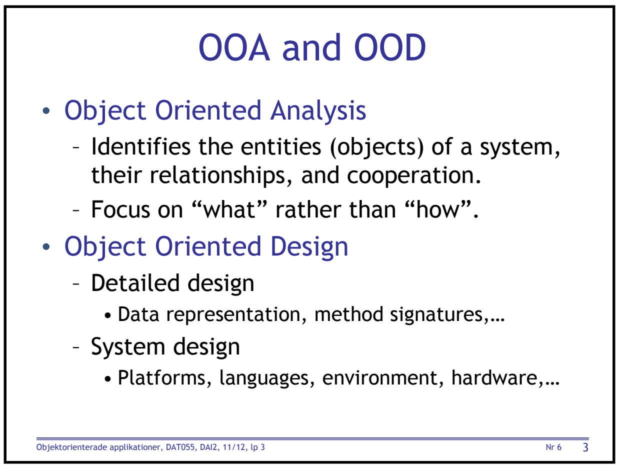
3 Designing applications

(BK chap. 13)



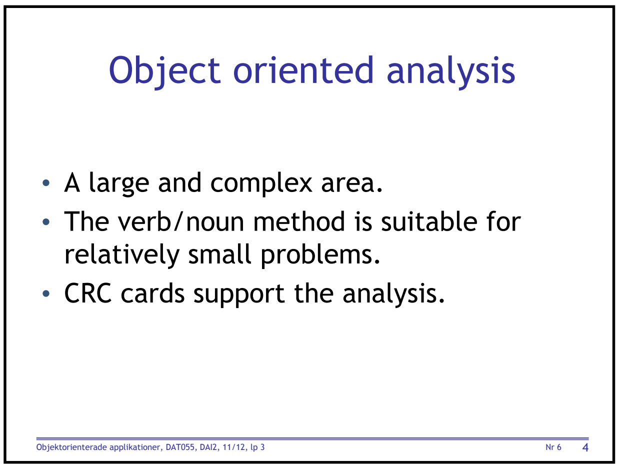
Main concepts to be covered

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



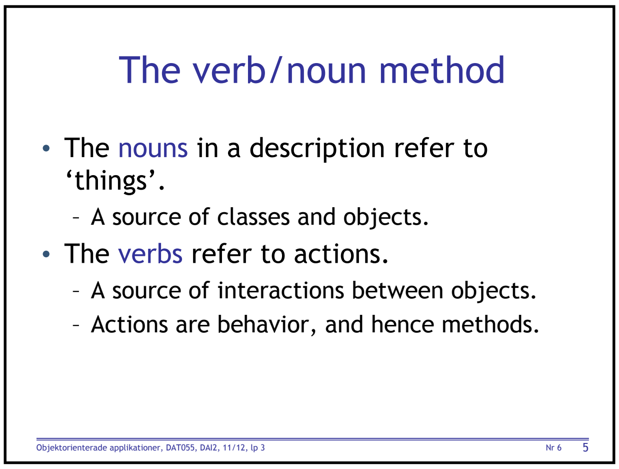
OOA and OOD

- 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,…



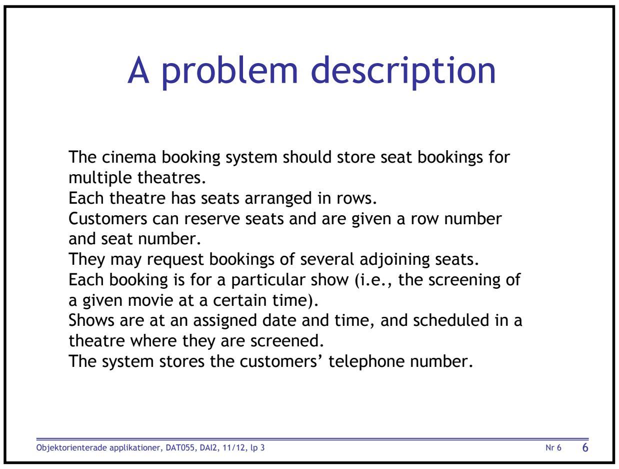
Object oriented analysis

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



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.



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.

Nouns and verbs

Cinema booking system
Stores (seat bookings)
Stores (telephone number)

Theatre
Has (seats)

Movie

Customer
Reserves (seats)
Is given (row number, seat number)
Requests (seat booking)

Time

Date

Seat booking

Show
Is scheduled (in theatre)

Seat

Seat number

Telephone number

Row

Row number

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*.

A CRC card

Class name	**Collaborators**
Responsibilities	

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.

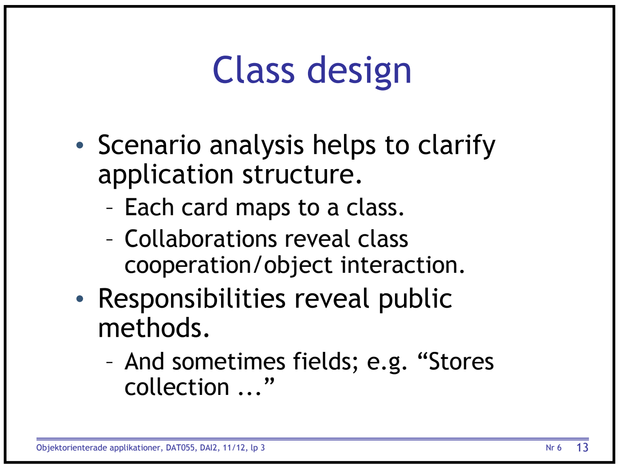
A partial example

CinemaBookingSystem	Collaborators

Can find shows by title and day.
Stores collection of shows.
Retrieves and displays show details.
...
	`Show`
	
	`Collection`

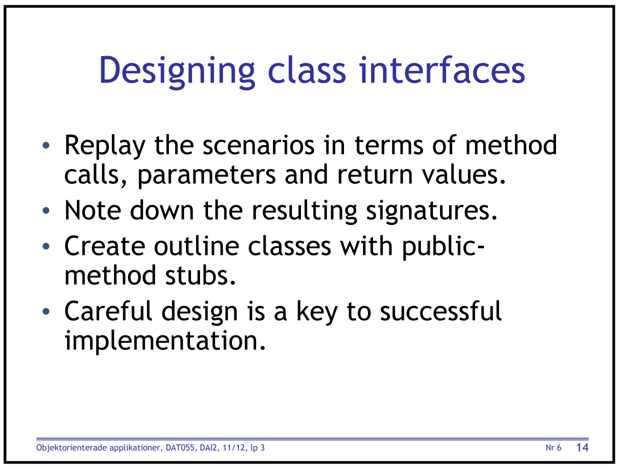
Scenario analysis

- 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.



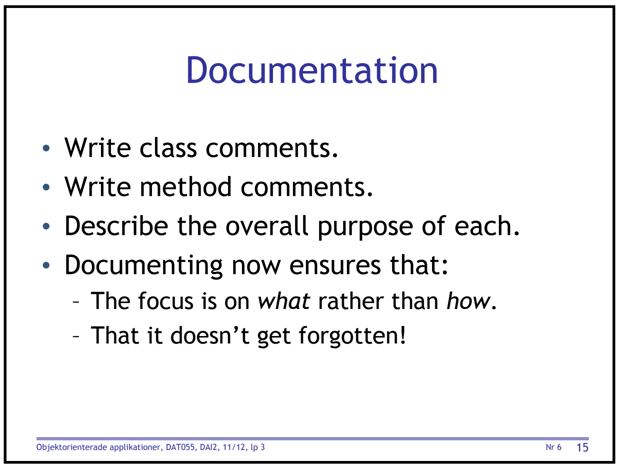
Class design

- 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 ...”



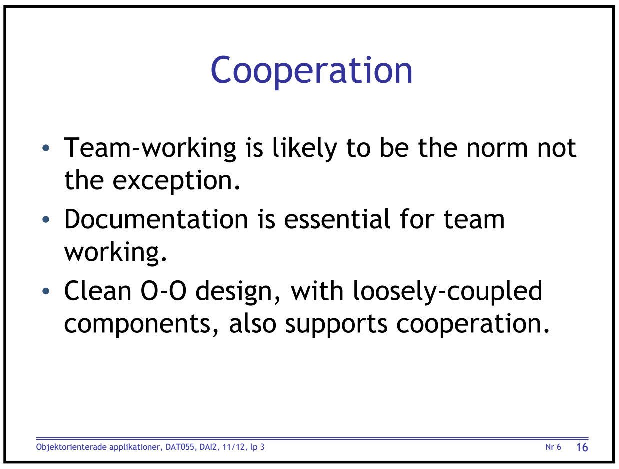
Designing class interfaces

- 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.



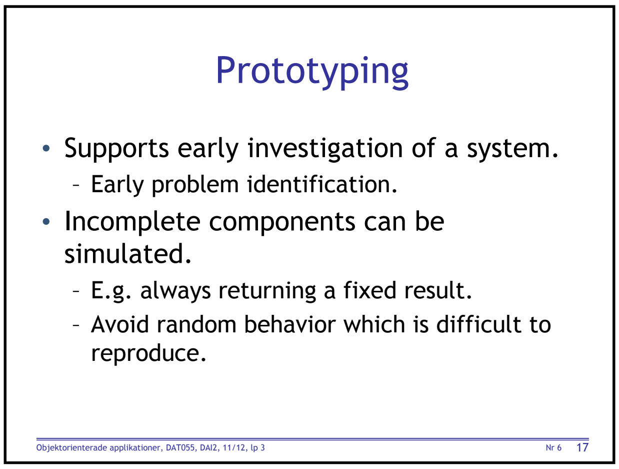
Documentation

- 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!



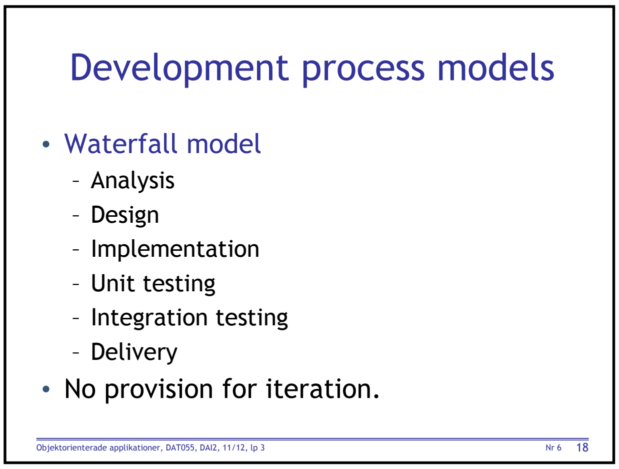
Cooperation

- 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.



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.



Development process models

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

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.

Graphical modeling languages

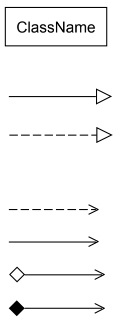
- A modeling language has a *graphical syntax* (and a more or less well defined semantics).
- Graphical modeling focus on *conceptual aspects* of a design.
- OMT = Object Modeling Technique (*Michael Blaha*,Jim *Rumbaugh*, *William Premerlani*)
- Booch (*Grady Booch*)
- UML = Unified Modeling Language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

UML diagram types

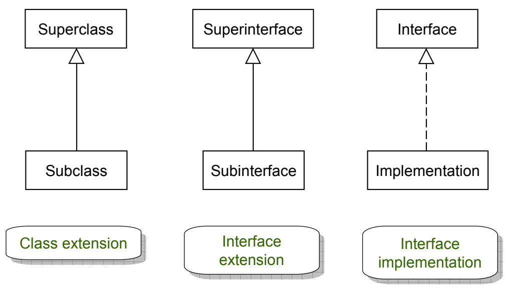
- Static design view
  - Class diagrams (static relations)
  - Component diagrams (modularization)
  - Deployment diagrams (run-time config.)
- Dynamic design view
  - Use case diagrams (user level behavior)
  - Scenario diagrams (object cooperation)
  - State diagrams (individual object behavior)

Class diagrams

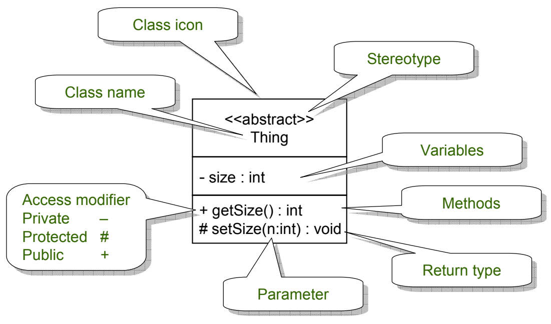
- Class icons
- Type relationships
  - Inheritance ("is a")
  - Implementation
- Object relationships
  - Dependency
  - Association ("knows")
  - Aggregation ("has")
  - Composition ("contains")



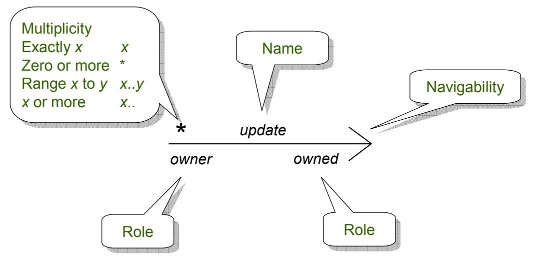
Inheritance relationships



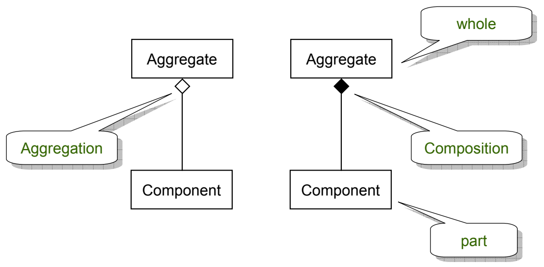
Class icons



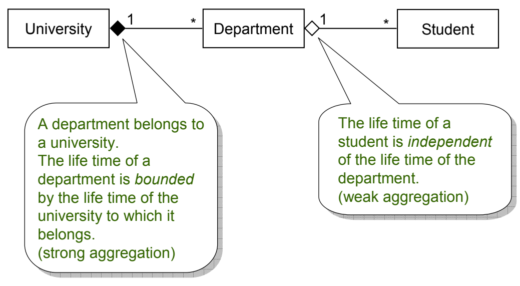
Object relation properties



Aggregation and composition

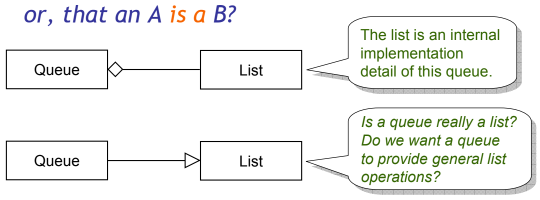


Aggregation and composition (2)

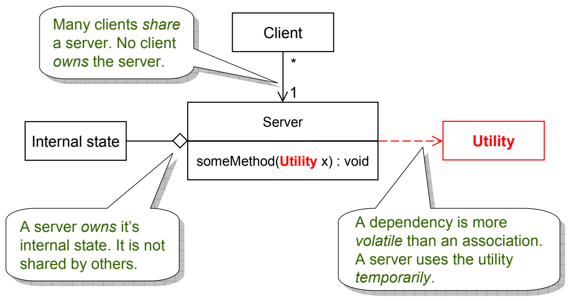

University
1
*
Department
1
*
Student
A department belongs to a university. The life time of a department is *bounded* by the life time of the university to which it belongs. (strong aggregation)
The life time of a student is *independent* of the life time of the department. (weak aggregation)

Aggregation vs inheritance

- Sometimes aggregation is 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?*


Queue
List
The list is an internal implementation detail of this queue.
Queue
List
Is a queue really a list? Do we want a queue to provide general list operations?

Aggregation, association and dependency relationships

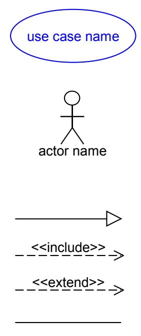

Many clients *share* a server. No client *owns* the server.
Client
*
1
Server
Internal state
someMethod(Utility x) : void
Utility
A server *owns* it's internal state. It is not shared by others.
A dependency is more *volatile* than an association. A server uses the utility *temporarily*.

Use case modeling

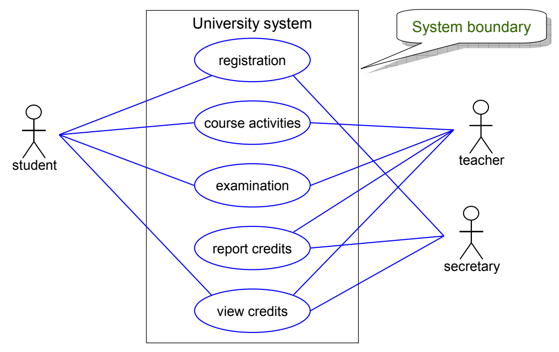
- Use case view
  - Captures the behavior of a system as it appears to a user outside the *system boundary*.
  - Main inventor - *Ivar Jacobson*
- Actor
  - External part that interacts with the system.
  - Idealized user: human, other system, process, ...
- Use case
  - External system behavior, meaningful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

Use case diagrams

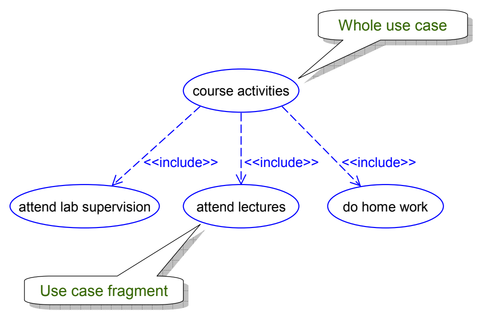
- Use case icons
- Actor icons
- Use case relationships
  - Generalization
  - Inclusion
  - Extension
  - Participation



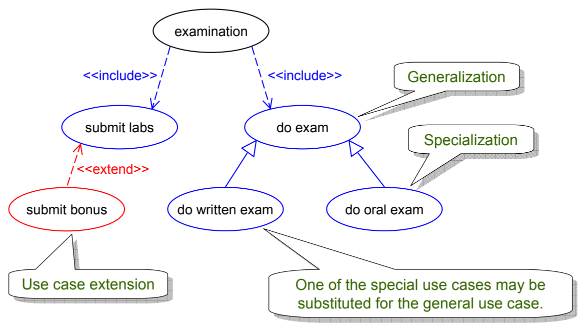
Use case diagram for a university



Use case parts



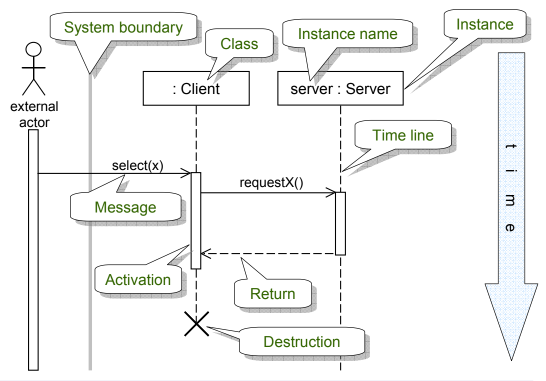
Generalization – specialization and extension



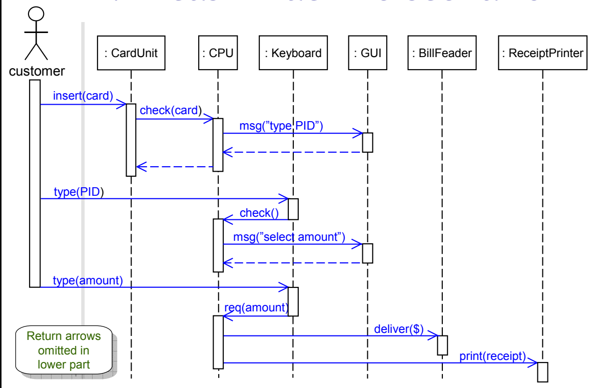
Scenario diagrams

- 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.

Sequence diagrams



Ex. A cash machine scenario



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.

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.