

4. Objekt-Orienteret Analyse

Klasser, attributter og associationer
Konceptuel model
Klassediagram
Generalisering
Aggregering
Use Cases
Use Cases og kravspecifikation

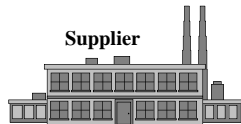
Efter denne lektion (og øvelserne) skal du:

- Kunne analysere en given problemstilling ved hjælp af teknikker til Objekt-Orienteret Analyse
- For en given (mindre) problemstilling kunne tegne et klassediagram
- Være orienteret om hvorledes generalisering og aggregering tegnes i et klassediagram
- For en given (mindre) problemstilling kunne skrive et antal Use Cases
- Kende sammenhængen mellem krav og Use Cases

Classes



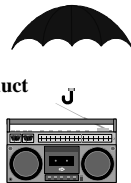
Customer



Supplier



Product



J

Attributes

- An attribute is a property of a class that an organization chooses to record
- Attributes for customer entity:
 - Name
 - Address
 - Contact
 - Phone
 - Fax



Attribute Values

- An *attribute value* is a value assigned to an attribute for an class instance (object)
- An *instance* is a specific collection of assigned values for a class that represents a specific class member
- Attributes of customer *instance* Sally Smith:
 - Name: Smith, Sally
 - Address: 101 Decatur Street, Atlanta, GA 30303
 - Contact: Smith, John
 - Phone: 303.456.7890
 - Fax: 303.987.6543

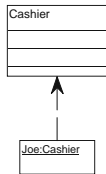


Conceptual Modeling

- Conceptual Model
 - a representation of concepts in a problem domain
 - static structure of concepts in which no operations are defined
 - concepts
 - associations between concepts
 - attributes of concepts
 - a class diagram can be derived from the conceptual model, but a class diagram is not a conceptual model
 - a class diagram includes concepts from the solution domain

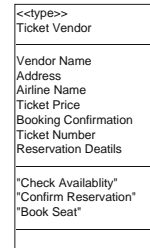
Objects and Classes

- Object Class
 - (a.k.a. Class)
 - Description of objects with:
 - similar properties (attributes)
 - common behavior (operations)
 - common relationships to other objects
- Object Instance
 - (a.k.a. Instance, or Object)



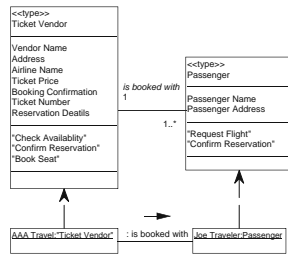
Attributes and Operations

- Attributes
 - A data value held by objects in a class
 - E.g., Name, Price
 - syntax
 - <name> : <type> = <default value>
- Operations
 - A function that may be applied to objects in a class
 - E.g., Check Availability
 - syntax
 - <name> (<args>) : <result types>



Associations and Links

- Association
 - a description of a links with a common structure and semantics
- Link
 - a description of a physical or conceptual connection between object instances

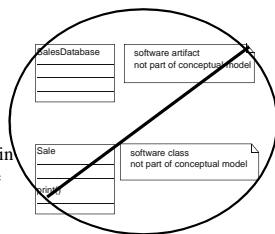


Building a Conceptual Model

- Look for key concepts in
 - Requirements
 - Use Cases
 - Pre-defined category lists
- If in doubt, create a concept rather than an attribute or association
 - Needed especially where multiple object instances can occur
- Add Associations later
- Add Attributes toward the end
 - However, attributes help clarify the meaning (intension) of the concept
- Concepts are the key
 - Devote more time to concept building than attribute or association definition

Avoid Software Descriptions

- Problem Domain
 - Concepts that exist independent of any (software) solution description
- Solution Domain
 - Concepts that exist only in the artificial world of the (software) solution description



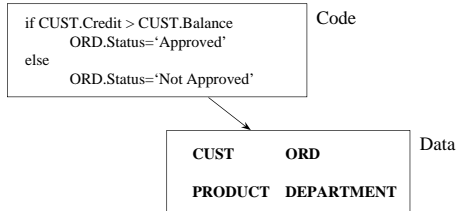
Distribution of Submarine Supplies Requirements



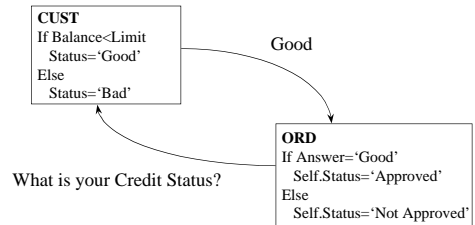
We are losing business due to our poor order processing and delivery. We mainly serve as a distribution center that consolidates produce and meats for submarine shops. However, we are not meeting our business objectives of profit, quality, responsiveness. Moreover, we are not as efficient as we would like, as our works spend an inordinate amount of time trying to use our current system. Things are a bit chaotic now...

- Your assignment
 - Draw a Conceptual Model for DSS

Traditional Approach



The Object-Oriented View



If you have been thinking of data as inactive try to think about animating it so that it becomes an active object

Concept Qualities

- Named using noun phase
- Based on concepts (terms) found in problem domain
- Exclude solution details
- Avoid redundant concepts
 - User and Customer
- Do not add concepts that are not in the specific problem domain
 - It is tempting to generalize or add related concepts
- No solution domain concepts

Attribute Qualities

- Attributes record the current state of an object
- Keep attribute (types) simple
 - Consider using only common types found in most programming languages
 - Boolean, Number, String (text)
 - Or other simple types
 - Date, Time
- If there are any interesting subparts, associations, or operations, then "promote" an attribute to a concept
 - Price is often promoted to a concept for international currency
- Attribute values cannot be distinguished
 - One address string "Joe Smith" cannot be distinguished from another
- When in doubt, create a concept

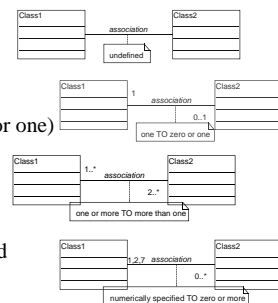
Association Qualities

- Named using verb phrase
- Focus on relationships where an object "needs to know" (or remember) another object
- Associations describe persistent, long-term, static, structural relationships
- Avoid showing an association that can be derived by traversing other associations
 - Unless the association aids in providing the reader a better understanding of the model
- Define multiplicity



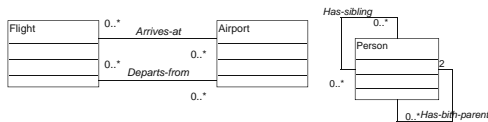
Multiplicity of Associations

- Undefined
- 1 TO optional (zero or one)
- One or more TO > 1
- Numerically specified



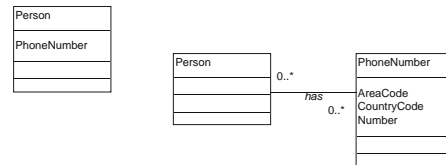
Multiple Associations

- Consider multiple associations
 - between two different class
 - to a single class
- when each association has a distinct meaning



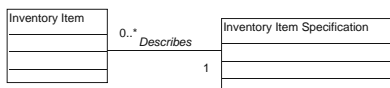
Multiple Items

- Create a separate concept where an object is associated with multiple items
 - A Person has multiple PhoneNumbers



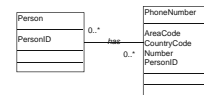
Item and Item Specification

- Separate objects from their description
 - May want to track inventory items, e.g. a Chair
 - However, may need to replenish inventory, so
 - Track inventory item specifications as well as inventory items



Design Creep

- Design Creep
 - The tendency to introduce design and implementation consideration into the analysis phase
 - While object-orientation reduces the distinction between analysis and design (compared to traditional development) it is not an excuse to "rush to code"
- Examples
 - Defining solution concepts in the conceptual model or early in the class diagram construction
 - Defining (or removing) class or associations based on the envisioned implementation
 - Using object identifiers or foreign keys

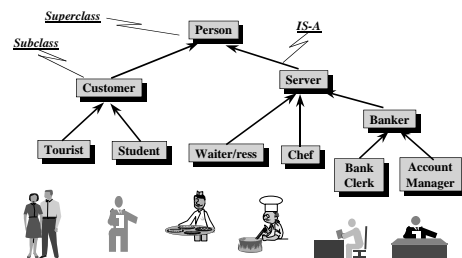


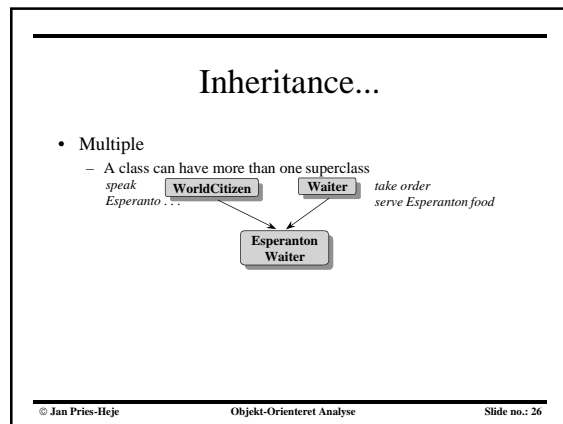
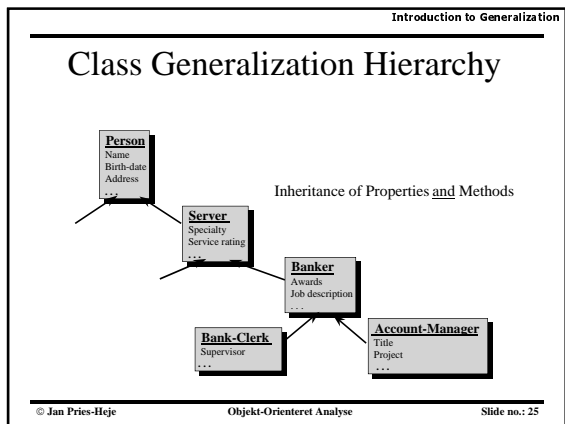
Exercise: In an interview with a user the statements below were recorded. You are now asked to illustrate (draw) the statements as a Conceptual Model



- A person is identified by a Social Security Number, and has a first name and a last name
- A department may have many persons assigned, but need not have any (= can be zero)
- A department is identified by three letter name (like "CIS"), and has a full name, and a phone number
- A person may be assigned to one or more than one departments.
- A person belongs to one and exactly one category. All persons will belong to a category.
- A category is identified by a code, and has a name

Generalization and Inheritance





Inheritance in C++

```

class Person
{
public:
string Name;
string Address;
string PhoneNumber;
void Print();
}

class Customer
{
public:
string Name;
string Address;
string PhoneNumber;
string Contact;
float Balance;
void Print();
}

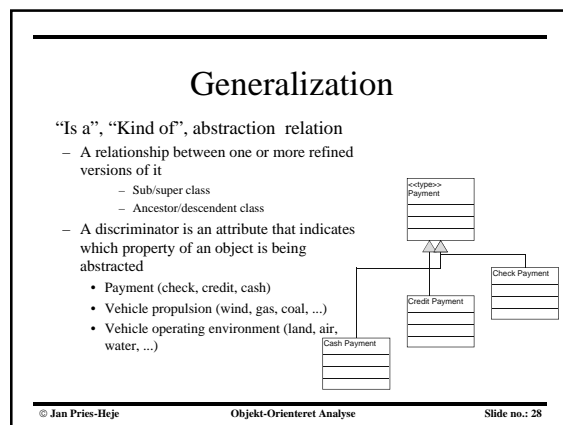
class Person
{
public:
string Name;
string Address;
string PhoneNumber;
void Print();
}

class Customer:
public Person
{
public:
string Contact;
float Balance;
}

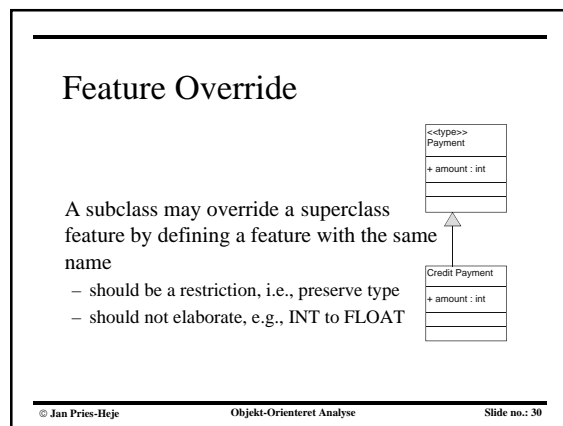
We do this...
  
```

Instead of this...

© Jan Pries-Heje Objekt-Orientet Analyse Slide no.: 27



- ## Applying Generalization
- The following two rule **must** apply
 - Is-A Rule
 - All members of a subtype set must be members of their supertype set
 - It must be said that, “The subtype is a supertype”
 - 100% Rule
 - 100% of the supertype’s definition should be applicable to the subtype. The subtype must conform to 100% of the supertype’s:
 - attributes
 - associations
- © Jan Pries-Heje Objekt-Orientet Analyse Slide no.: 29



Inheritance Guidelines

- Only inherit from objects of the same type
 - Conform to an object typology
 - Do not inherit for convenience
 - Using inheritance to “copy” properties into an object which is not a subtype causes problems
 - Introduces object into hierarchy which does not obey semantics of typology. Lead to confusion.
 - Instead, generalize common aspects and inherit them or use them as elements in aggregation
- Inherit query and update operations
- Do not override the protocol or meaning of operations

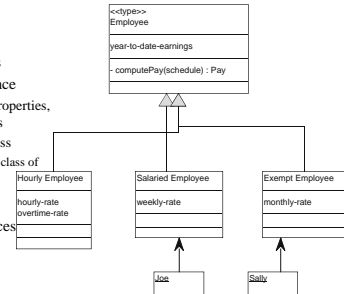
Abstract Classes

Abstract class

- has no direct instances
- only used for inheritance
 - to define common properties, especially operations
 - often is an origin class
 - topmost defining class of property

Concrete class

- can have direct instances

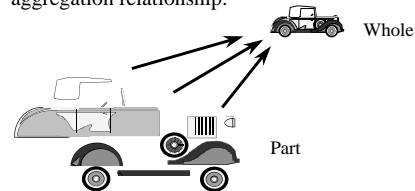


Exercise in Teams:
Draw a Class Diagram
where you apply
Generalization



Aggregation

- The whole-part relationship is called the aggregation relationship.

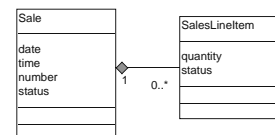


What really is ‘Aggregation’?

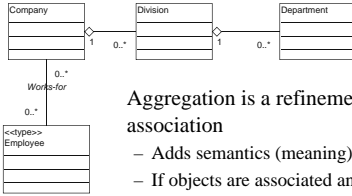
- The notion of ‘Emergent Meaning’
- Physical Containment
 - Vehicle, Engine
- Logical Aggregation
 - Diagrams, Symbols
- Ask:
 - Can I say: IS-A?
 - Can I say: IS-PART-OF?

Aggregation

- “Part-whole”, “a-part-of” relation
 - components objects are associated with the entire assembly object
 - Example
 - SalesLineItem is “part of” a transaction



Aggregation vs Association



Aggregation is a refinement of association

- Adds semantics (meaning)
- If objects are associated and...
 - one is "part of" another
 - the parts derive some attribute values from the whole
 - the parts are subordinate to the whole

Aggregation vs Generalization

- Relates instances
 - Association relates "run-time" classes
- One object instance part of another object instance
 - Hierarchical part composition
 - partonomy
- Multiplicity
 - Can associate multiple objects as part of another
- Relates classes
 - Association relates "design-time" classes
 - "copy" macro of class properties
 - Multiple classes combined to describe a single class
- One class is "a kind of" another class
 - Hierarchical type definition
 - typology
- Multiplicity
 - Does not provide for definition of multiple associations through inheritance

Exercise in Teams:
Draw a Class Diagram
where you apply
Aggregation



What to model?

Consider that you have been given the job of modeling DSB Gods. What do you model? Would you include the new purchase value of the trucks... the actual purchase value... the resale value... the make... the color of the interior... the number of speeding tickets issued against it... the number and destinations of its trips? If you try to include everything, you will never finish. Where do you start, when do you stop, what do you include, exclude, and how much detail do you want?

Inspiration: <http://members.aol.com/mcc4849n/papers/newsover.htm>

The answer is Use Cases

- An answer is provided by looking at Use Cases. You need to have sufficient information to deliver all the questions posed by the possible Use Cases.

Short definition of Use Case:

- Users of a system is called Actors.
- Use Cases are what happens when Actors interact with the system.
- A Use Case is a collection of possible sequences of interactions between the system under discussion and its Users (or Actors), relating to a particular goal.

Use Case Definition (1 of 2)

- Actor
 - Actors are basically users of the system. They are actually user types or categories. Actors are external entities (people or other systems) who interact with the system to achieve a desired goal.
- Use Case
 - Use Cases are what happens when actors interact with the system. An actor uses the system to achieve a desired goal. By recording all the ways our system is used ("cases of use" or Use Cases) we accumulate all the goals or requirements of our system.

Notation



Use Case Definition (2 of 2)

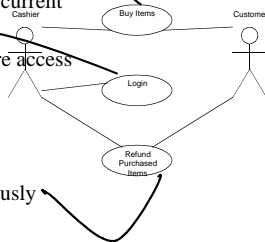
- Use Cases continued
 - A use case is a collection of possible sequences of interactions between the system under discussion and its Users (or Actors), relating to a particular goal or rationale.
 - The collection of Use Cases should define all system behavior relevant to the actors to assure them that their goals (or the rationale) will be carried out properly. Any system behavior that is irrelevant to the actors should not be included in the use cases.
- For more information see:
 - <http://members.aol.com/humansand/papers/completelist.htm>

Requirements to Use Cases

- Traceability from Requirements to Behaviors
 - For each (active) requirement, there should be at least one use case that “implements” it
 - For each use case, there should be at least one requirement that it “implements”

Use Case Diagram

- POST shall compute a running total of sales item for the current transaction
- POST shall provide secure access to its functions
 - *This is a non-functional requirement*
- POST shall allow for the itemized refund of previously purchased items



Documenting a Use Case *simplified*

- Use Case: Buy Items
- Actors: Customer, Cashier
- Type: Primary
- Purpose: Assist and record a sale
- Description: A customer arrives at a checkout with items to purchase. The Cashier records the purchased items and collects a payment. On completion the customer leaves with the items.
- Typical Course of Events
 - Actor Action System Response
 - X does y system does z
 - ...

Buy Items with Cash: Typical Events

- 1 This use case begins when a Customer arrives at a POST checkout with items to purchase
- 2 The Cashier records the identifier for each item. If there is more than one of the same item, the Cashier can enter the quantity as well
- 3 Determines the item price and adds the item information to the running sales transaction. The description and price of the current item are presented
- 4 On completion of the item entry, the Cashier indicates to the POST that item entry is complete
- 5 Calculates and presents the sale total
- 6 The Cashier tell the Customer the total
- 7 The Customer gives a cash payment—the “cash tendered” possibly greater than the sale total
- 8 The Cashier records the cash received amount
- 9 Shows the balance due back to the Customer. Generates a receipt
- 10 The Cashier deposits the cash received and extracts the balance owing. The Cashier gives the balance owing & printed receipt to Customer
- 11 Logs the completed sale
- 12 The Customer leaves

Alternative Courses	
	Line 2: Invalid identifier entered. Indicate error. Line 7: Customer didn't have enough cash. Cancel sales transaction.

Exercise: Rescue Station



The rescue station, located in Copenhagen, controls emergency assistance dispatch for one main station and several local stations. The main station and the local stations have a total of 100 vehicles for all types of assistance. The main station employs 100 rescue workers. The local stations employ 100 rescue workers in total.

The rescue station receives calls for assistance from people in the community, or from a central alarm station (if help is needed in other communities). Depending on which type of assistance is needed, the relevant vehicles are dispatched with an appropriate crew.

Exercise: Rescue Station (continued)

The dispatcher continuously monitors all ongoing assistance activities to determine when vehicles and crews are available for other duties. To succeed in this monitoring task, the dispatcher needs information about assignments, vehicle releases, and vehicle positions throughout assistance calls.

Emergency assistance's is the most visible task, but other types of assistance activities – such as transporting the sick and assisting motorists – have a significant higher volume.

When assistance calls come in they are registered and categorized. Then they are passed on to the dispatcher. The dispatcher takes care of organizing and sending out the necessary vehicles and rescue workers. The dispatcher will also monitor ongoing assistance activities so they can predict when active vehicles and rescue workers will be available.

Rescue Station – Your Assignment

Write a **USE CASE** for the dispatcher
Use template below.

Use Case:

Actors:

Type:

Purpose:

Description:



Actor Action

System Response

1. This Use Case begins ...

Alternate courses:

Use Case Elements

- Transaction
 - an atomic set of activities that is performed either fully or not at all, invoked by a stimulus... , consists of actions, decisions and transmission of stimuli
- A measurable value
 - performance has a visible, quantifiable and/or qualifiedly impact on things outside the system, and in particular, the actor who initiated the task

Documenting Use Cases

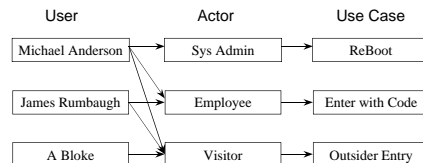
- A flow of events document is created for each use cases
 - Written from an actor point of view
- Details what the system must provide to the actor when the use cases is executed
- Typical contents
 - How the use case starts and ends
 - Normal flow of events
 - Alternate flow of events
 - Exceptional flow of events

Use Case Structure

- Dozen's
 - Not hundreds
- Contains
 - Brief statement of purpose
 - 1 sunny day event (main course, primary scenario)
 - N rainy day(s) (alternate courses, secondary scenarios)
- Complete "end to end" business process
 - From initiation to completion
 - Initiated at an external interface
 - Common error
 - Use Case for an individual step in a process, rather than a complete end-end process that starts at the interface

Users, Actors and Use Cases

Actors do not need to be described in detail. Users that use the system in the same way are modeled by one actor. A physical user may be 'several actors', in the worst case at the same time.



Qualities of a Good Use Case

- Derived from a Requirement
- Preferably relevant to one Actor
- Good Granularity
 - not too small or too large
 - not too trivial or too complex