

Chapter 10



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 10: Information Modeling

DTU course 02264

Agenda

Abstract

- In this chapter we will specify information items, their relationships, and their attributes and behaviors (“object lifecycles”) using UML class, object and state machine models.
- It is important not to confuse class models and information models. Even if they use just the same notation, they refer to completely different things: domain concepts and relationships on the one hand, and implementation elements on the other.

Contents

1. Elements of Information Models
2. Creating Information Models
3. Refactoring Information Models
4. Splitting Large Information Models



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 8.1:

Elements of Information Models

DTU course 02264

Information Model

- **An Information Model describes the information items of a domain, including their attributes and relationships, in a purely logical, domain-oriented way.**
 - The information model is part of the requirements and elaborates the problem space, not the solution space.
- **In a UML-setting, we use analysis-level class models (ACM) to represent the information items and their relationships.**
- **Information Models do not just contain static structure, however: they contain *all* there is to be known from a domain point of view on information items, individually and collectively.**
 - That includes their attributes and states, but also their operations and state transitions, and possibly their interactions. Therefore, an information model may contain class diagrams as well as state machines, and interaction diagrams.

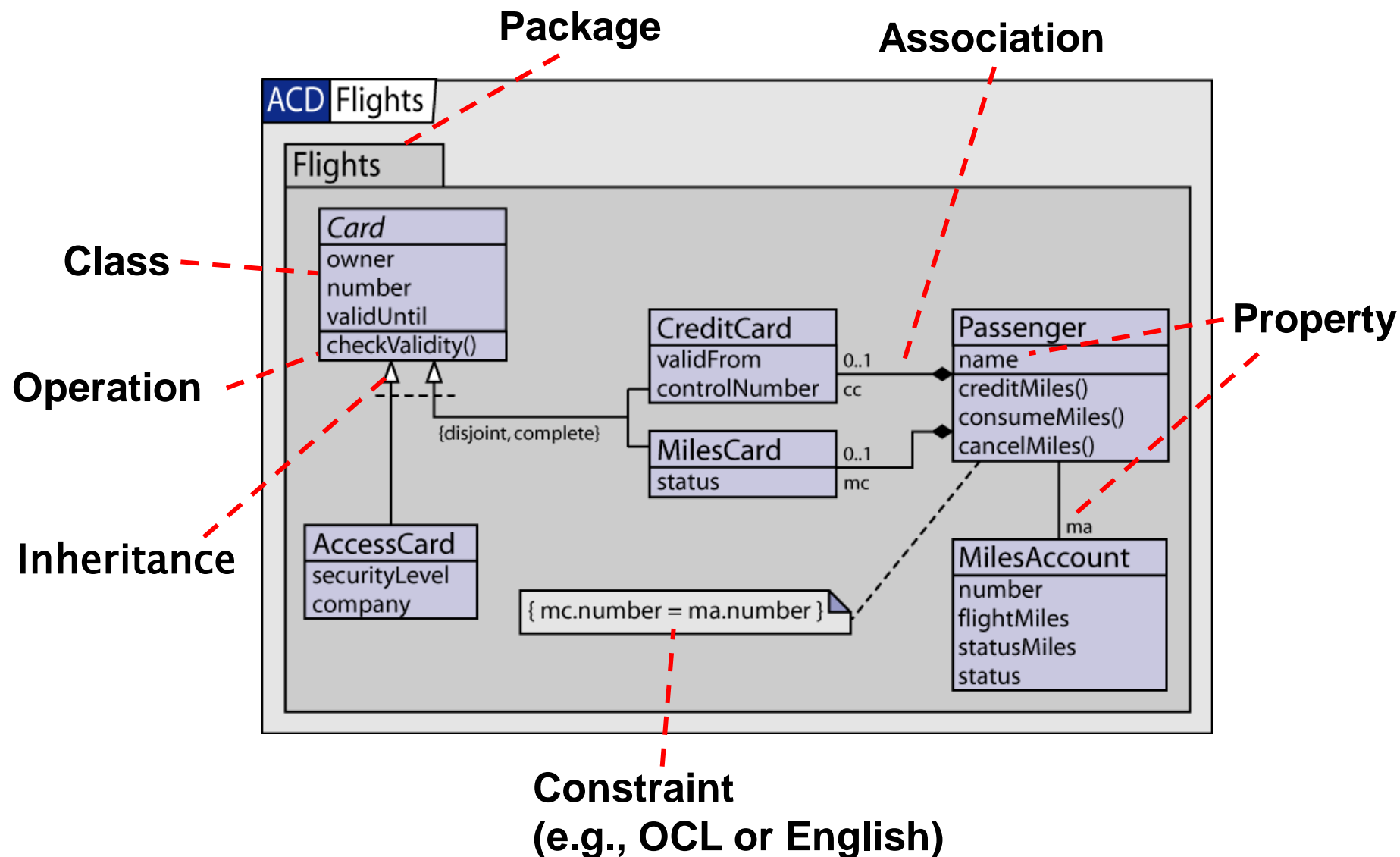
Information Items

- **Sometimes, we have to integrate widely contradicting sources of information for creating class models.**
- **In such situations, it might be a good strategy to create object models first and only later abstract them to class models.**
 - It seems that it is much easier for people to agree on given individual cases, particularly when they are real cases.
- **When we have collected many such individual scenarios, we may abstract recurring elements to the type level and improve the resulting class diagram by refactoring (see below).**

Analysis vs. Design Level Models

- **It is important not to confuse analysis and design (or implementation) level models.**
 - Even if they use just the same notation, they refer to completely different things, namely domain concepts and relationships on the one hand, and design (or implementation) elements on the other.
- **Analysis-level Models describe the problem space, not the solution space. Their purpose is understanding, not construction.**
 - Therefore, they do not include design- or implementation-related information.
 - This includes technological details such as programming-language specific features or constraints.
 - Similarly, whether a given model will lend itself to an efficient implementation (or, in fact, any implementation) is irrelevant.
- **Addressing design and implementation issues would prematurely constrain the set of possible solutions, and thus obstruct the search for the optimal solution.**

Concepts of A-level class models



Concepts of A-level class models

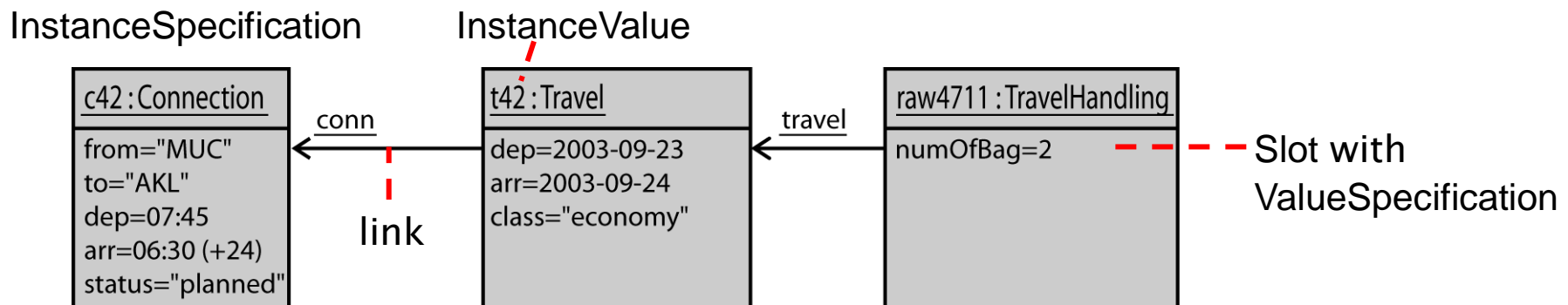
- **Many basic concepts of class models can be used on any level.**
 - E.g. Class, Feature, Multiplicity, Type, Parameter, InstanceSpecification (“Object”)
- **Other concepts are restricted to specific levels, or sets of levels**
 - ACM: AssociationClass, GeneralizationSet, reading direction
 - A/DCM: Association,
 - DCM: Navigation, Realization/Substitution, keys
 - D/ICM: Interface, Uses/Realizes, visibilities, TemplateClass
- **Some concepts even have different meanings and constraints at different levels.**
 - For instance, there may be a difference between single and multiple Generalization („Inheritance“) at the D- and I-levels, but not at the A-level.
 - Similarly, a Package at the D/I-levels refers to a system element, whereas a package at the A-level is only a grouping mechanism for model elements.

Scenarios

- **At times, class models are too complex to create or use**
 - For instance, the model grew too large to comprehend, or the audience lacks the required skill or experience.
- **In such cases, it might be easier to work with specific examples rather than the generic description.**
 - The UML provides Object Models (OM) for this purpose. Technically, they are a restricted form of class diagrams containing only objects of classes, their slots and links.
- **OMs present a certain state of the information model, e.g.**
 - a pre/post-condition of an acceptance test;
 - the formalization a business rule; or
 - an example of the capabilities and constraints of an information model.

Concepts of Object Models

- **Object Models contain only the following concepts**
 - InstanceSpecification (more commonly known as Object)
 - ~~Link~~
 - Slot
 - Literal



- **Due to the paucity of the notation, there is no difference between different levels of abstraction.**

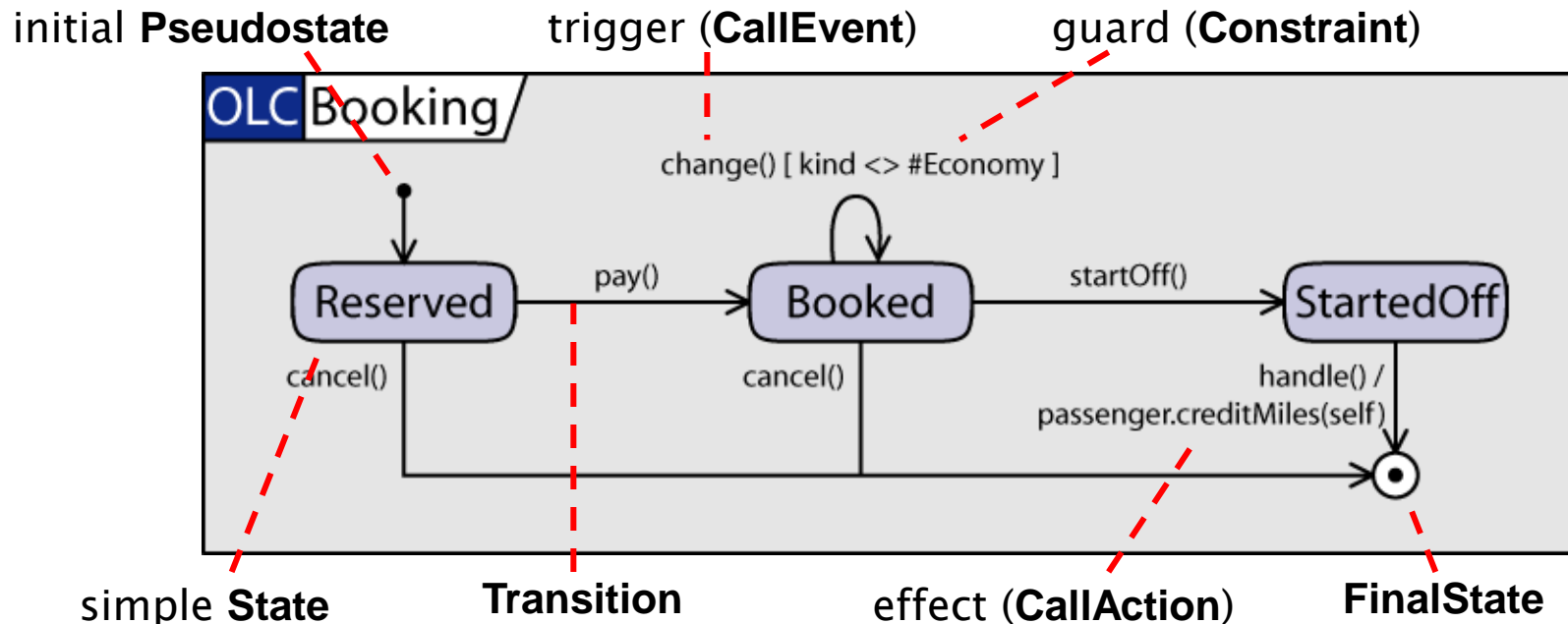
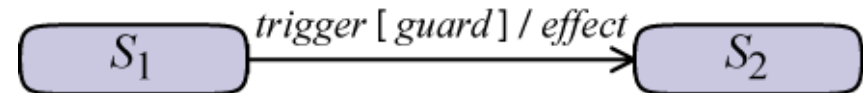
Object Life Cycle

- A scenario may be considered as a snapshot of a trace, i.e., a sequence of scenarios.
- If there are several such traces, they can be abstracted into a more compact description using a state machine: the „Object Lifecycle“.
- The object life cycle represents the states and transitions of all instances of a class:
 - the triggers correspond to the classes' operations;
 - the states are usually represented as a single attribute or small subset of all the attributes of the class.

Concepts of A-level StateMachines

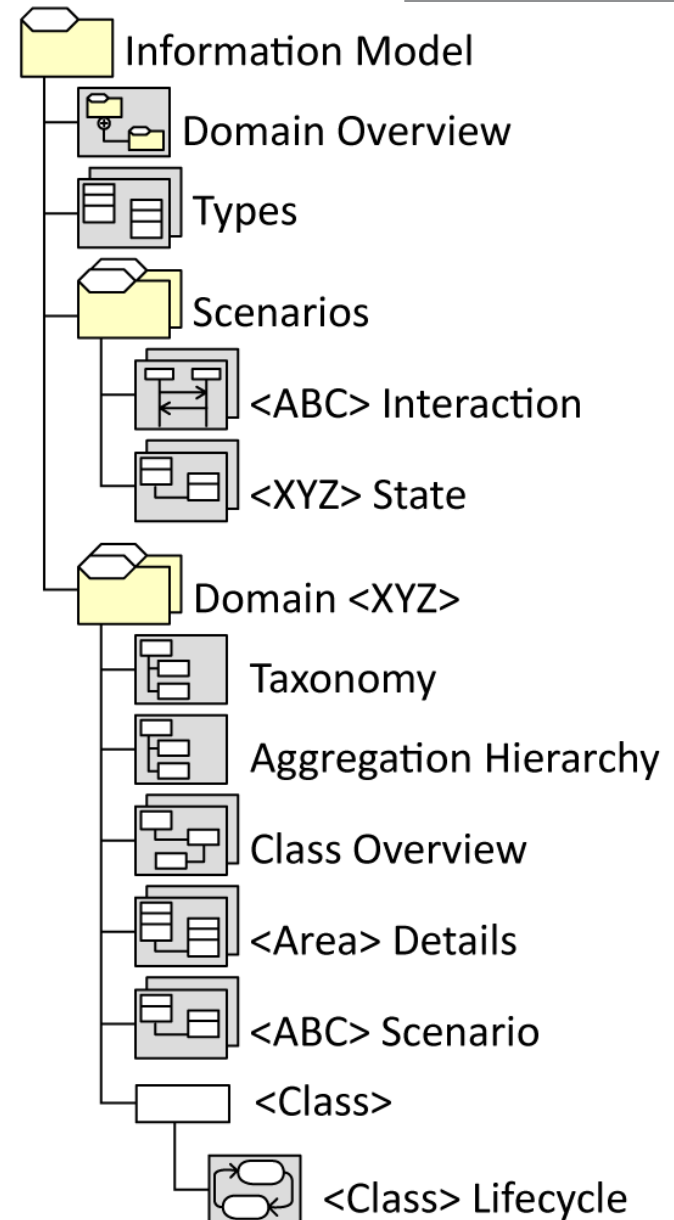
State machines model behavior

- using states interconnected ...
- with transitions triggered ...
- by event occurrences.



Overall Structure of Information Models

- **In MagicDraw, Object Behavior Models may be modeled as a part of a Class, establishing their connection directly.**
 - If not placed there, a hyperlink ought to be established.
 - In other tools, e.g. ADONIS, such a connection must be established explicitly by a hyperlink, whose consistency may be validated automatically.
 - Observe that the placement will have direct influence on the version control regime and the work organisation.
 - For instance, when placing a StateMachine under a Class, they must be versioned together, which means they must be created together.
 - So, team specializations for structural vs. behavioral modeling are not possible any more. Instead, team specialization must follow early structural break downs, which, when changed later on, are likely to give rise to dangerous ripple effects.
- **Either way, the name of the Class and lifecycle (and possibly the package containing all the models, diagrams, and elements belonging to it) should coincide.**
- **An object lifecycle may be illustrated by some scenarios, each of which will consist of a set of traces and some states.**
 - Traces emphasize the triggers and effects occurring during a run.
 - States emphasize the attribute values at some moment in time, usually including more than those attributes that define the states of the object lifecycle.





Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 8.2:

Creating Information Models

DTU course 02264

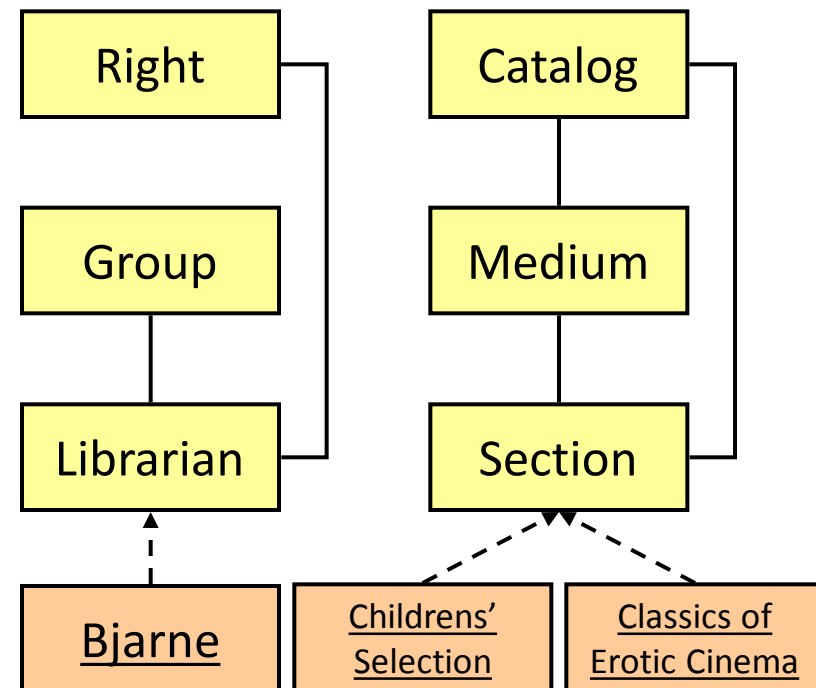
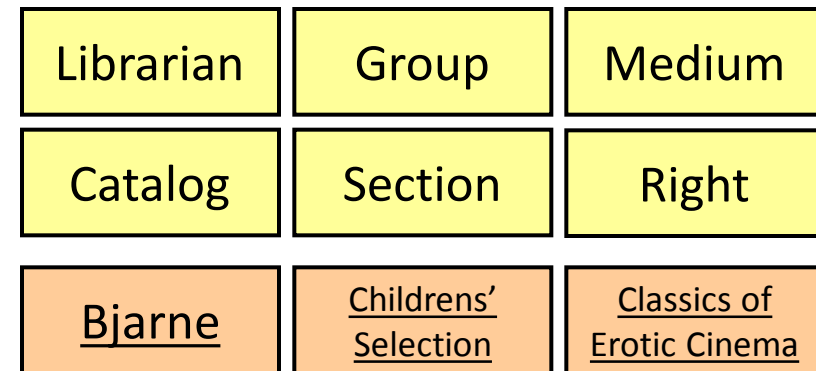
Creating Information Models

- **Information models capture static structure, that is, entities, their attributes and relationships.**
 - In Database design, this type of model is known as logical design.
 - The Abbott-Technique proposes to go through interview notes, and pick nouns to become classes, attributes, and associations, while verbs become operations.
- **Clearly, this only works only under very restricted conditions:**
 - for small scale systems,
 - only for an initial draft, and
 - domain novices with severely restricted abstraction capacity.

In other words: don't try this at work.
- **On the other hand, it is useful as a teaching device, and we have to start at some place.**
 - So we go back once more to the usage scenarios of the Taarbaek Library and extract our first information model from it.

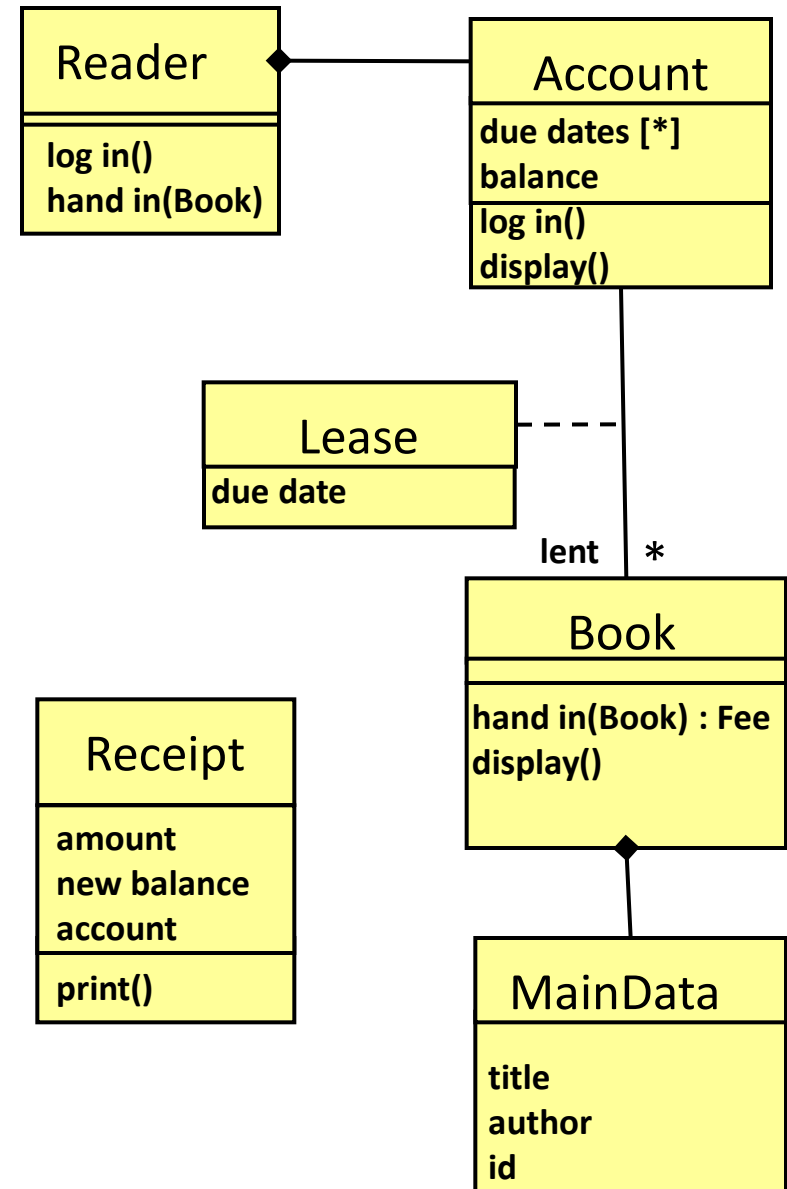
Creating the first classes and objects

- [...] adding [the new **librarian**] to the right **group**, and equipping him with the appropriate **rights** to perform the tasks he will be doing. Later on, **Bjarne** will enter his personal data and upload a photo to complete his account.
- [...] taking [**media**] out of the **catalog** when they have been stolen, misplaced, or badly damaged.
- [...] when people swap media from the "**Childrens' Selection**" movie **section** with those from the libraries well-renowned and popular "**Classics of Erotic Cinema**" section.



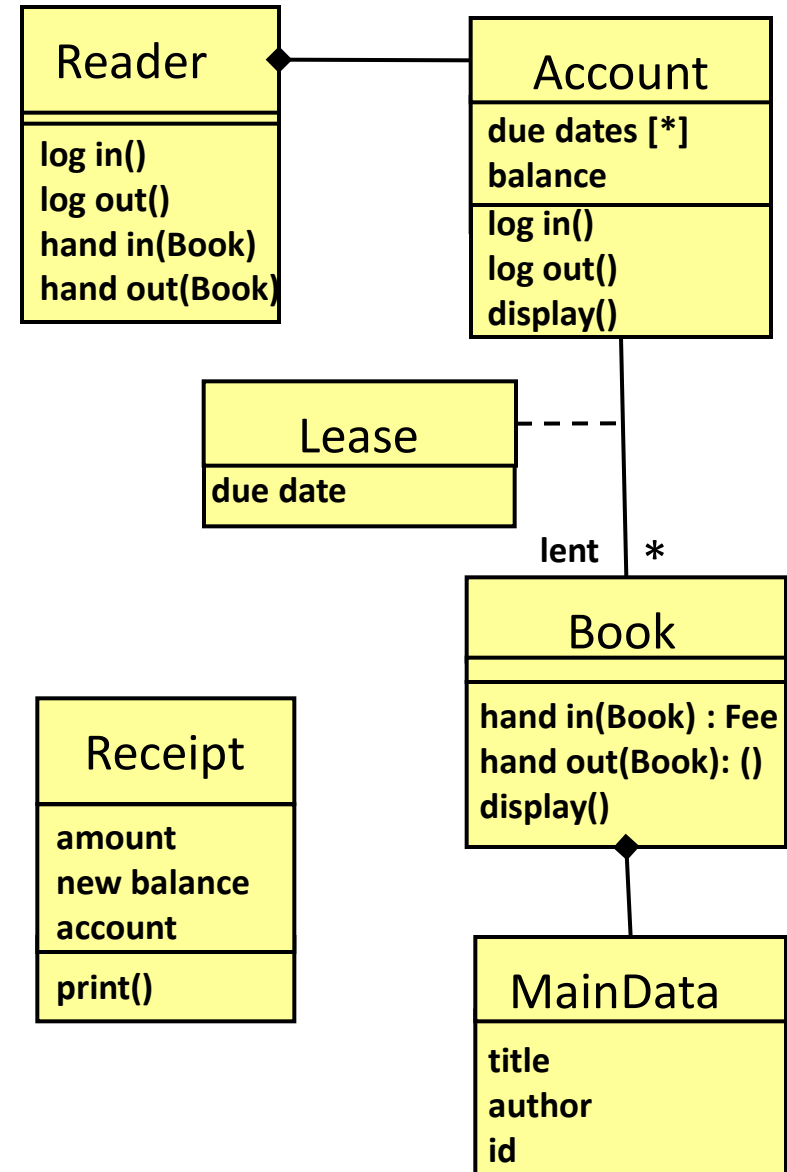
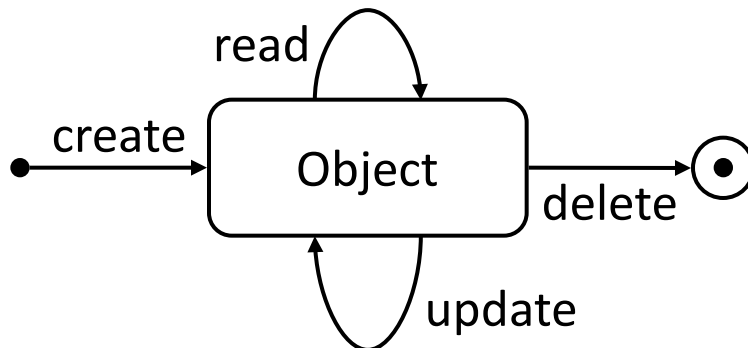
Creating Classes and Actions

- The first **reader** [...] has just one **book**. Anders **logs in**, takes the book, points its ID badge to the work place's scanner and the system recognizes that the **book has been checked in** again, while Anders puts it on the "returned books" shelf just behind him. When he turns round again, the system still **displays** the books' **main data (title, author, id)**, plus an indication that the book was **handed in** late.
- It also displays the **reader's account** indicating the list of **books** she has **lent** right now, when they are **due**, and the amount of **fees** accumulated.
- [...] the **slip** stating that he has received the money and that her **balance** is now positive.



Completing Actions by Lifecycle

- By experience we know, that many operations also have an inverse operation, adding more actions to our class model.
 - In fact, many operations come as CRUD-groups (Create, Read, Update, Delete), representing a minimum lifecycle.
 - Creators and destructors are inherent in OO languages, so we may ignore them here.

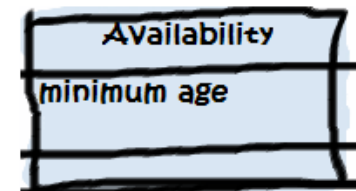
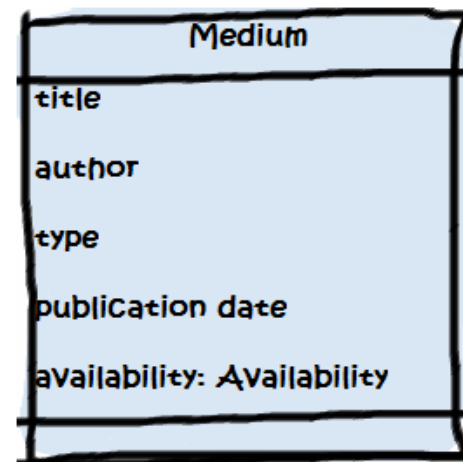
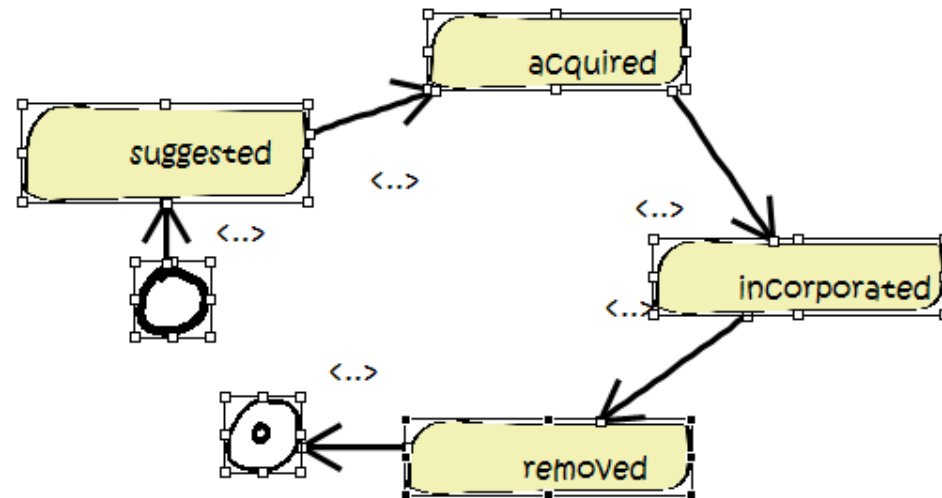


Extracting Information Model Fragments from Features

- **Another way of starting with models is to start at textual descriptions of features and qualities of a system and extract a fragment of a UML model directly from them.**
 - If a certain information item, property, or relationship is mentioned, this can be captured as part of a class model.
 - Likewise, mentioning a state or transition may be incorporated into a state machine.
 - Similarly, a function that is being mentioned may be turned into an operation of a class, a use case, or an action in an activity.
- **There are several benefits to this approach.**
 - It is often straightforward and fast to derive these fragments.
 - It ties the fragment directly to the requirements, thus allowing tracing and justification of the model.
 - It is easy to explain each individual fragment to a person without UML expertise, allowing to justify (parts of) the model.

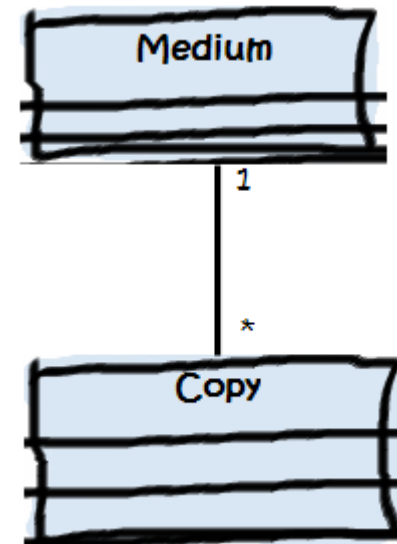
Example: Extraction from MLC1

- **MLC1: Media follow a defined lifecycle from suggested, via acquired, incorporated, to removed.**
 - The availability of incorporated media may be restricted, e.g. in terms of age restrictions, access restrictions for valuable copies and highly demanded media and so on.
 - The status of incorporated media is regularly updated to reflect damages and lending status.
 - Creating new media requires information such as title, author, type, publication date, etc.



Example: Extraction from MLC11

- **MLC11: The corpus may contain several copies to a medium.**
 - The catalog shall provide readers with access to media rather than individual copies.
 - Readers shall be able to find out, whether there is a copy of a given medium available for lending.





Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 8.3:

Refactoring Information Models

DTU course 02264

Restructuring Large Information Models

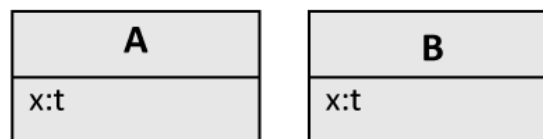
- **After working on an information model for a while, it may get large and complex, presenting more detail than can easily be handled. Such a model does not serve its purpose any more:**
 - It does not provide the overview that we need.
 - It will be difficult to create a clear layout easily.
 - It does not easily fit into a printed report or a slide presentation.
 - Complexity makes it difficult to detect errors, so we may run into quality issues later.

- **Thus, we may have to spend some effort on consolidating the information model before we move on. There are several maneuvers we may apply to this end:**
 - Apply the model structure template
 - Refactor class models
 - Split Aspects, Tiles, and Abstraction Levels
 - Split the over all model into domain modules

Refactoring class models

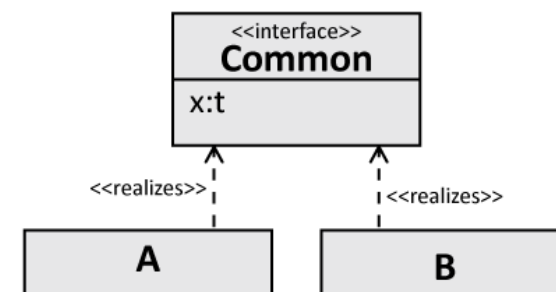
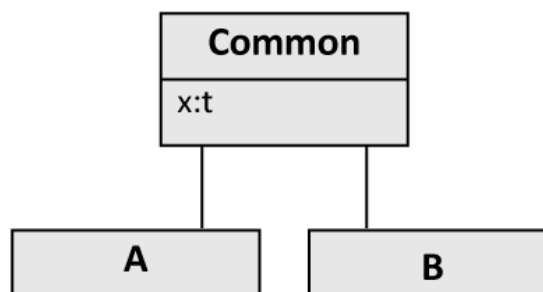
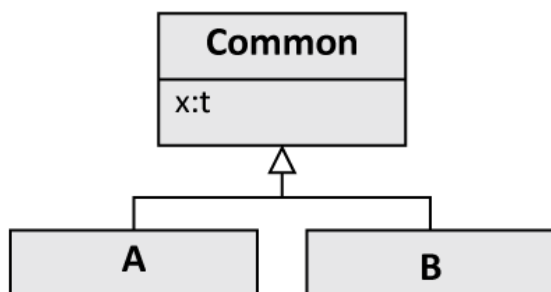
- There are several possible ways to refactor overlapping sets of features.

Move common features to...	...in order to...
super class	highlight the commonality of the domain concepts
interface	highlight behavioral similarities or expected technical differences (D/I-level only!)
associated class	create a reusable/domain class from the factored out set of features



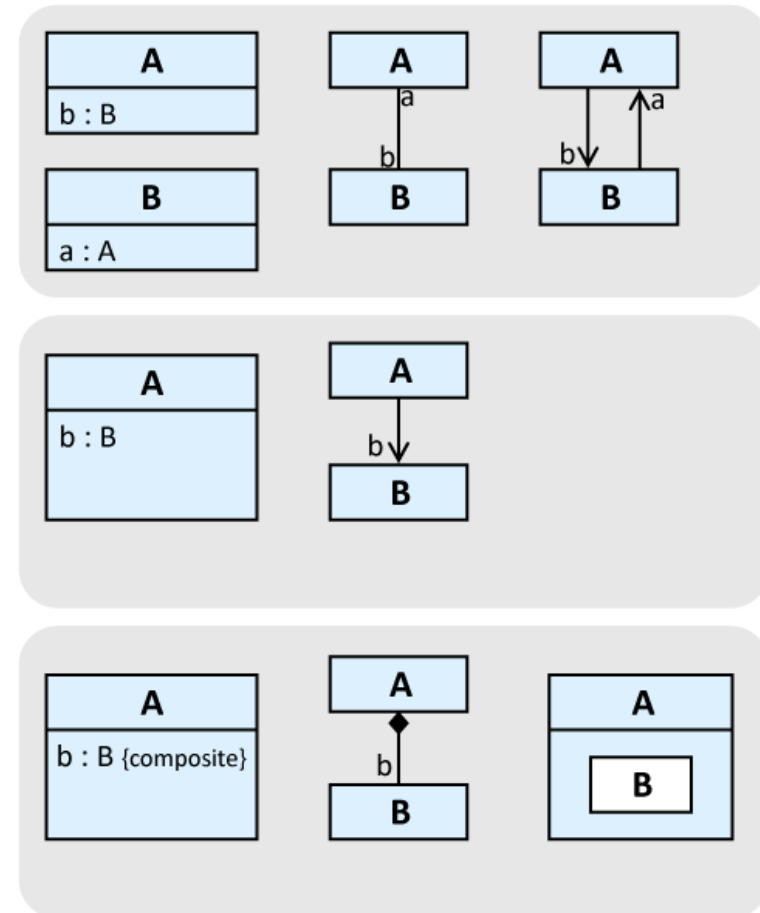
Note:

Some programming languages do not allow static elements in interfaces (e.g., Java).



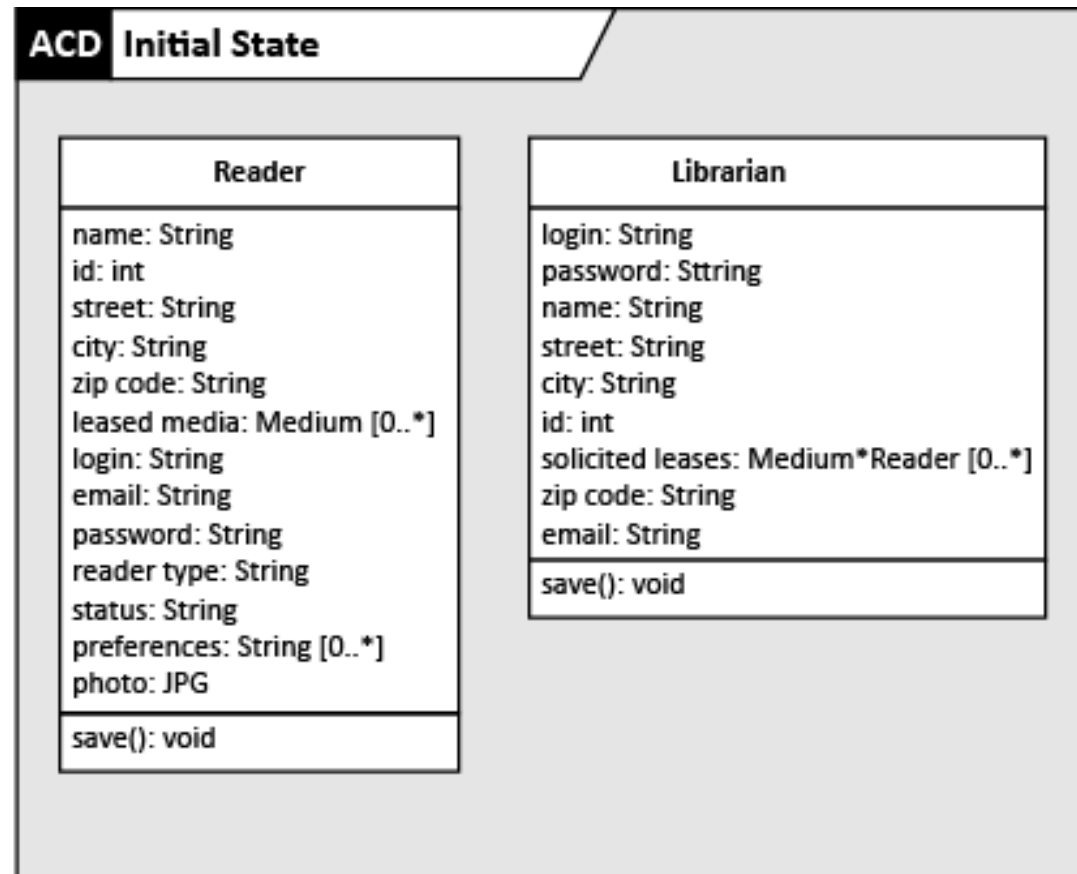
Feature or Association?

- In UML, association-ends are represented as properties.
 - To be precise, the structures on the right are each semantically identical in the sense that they give rise to the same set of object models.
- This leaves us with a choice – how do we know when to use which?
 - As a rule of thumb, use Associations for more important properties: their visual prominence sets them apart.



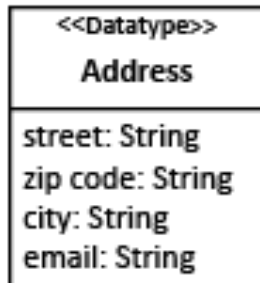
Refactoring class models (Example)

- Assume we have the following class model.
 - It is not very clear, and contains redundancies.
 - We may refactor it following some simple steps.



Factor out Redundancies

extract datatype



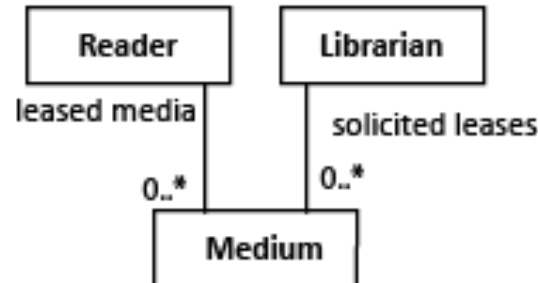
What to do

- 1) Create a datatype with common properties
- 2) Delete properties from original classes
- 3) add new property with type of datatype

Benefit:

- new reusable datatype
- smaller domain classes

extract common component



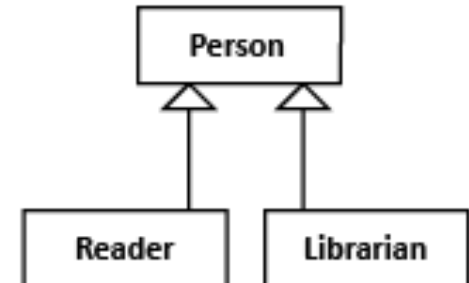
What to do

- 1) Create a common class
- 2) Delete properties from original classes that refer to class
- 3) add new associations between original classes and common class

Benefit:

- new domain class
- connection between existing domain classes

factoring out common properties into superclass



What to do

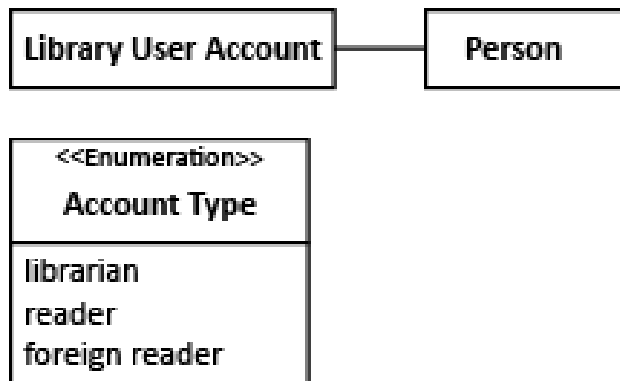
- 1) Create a common superclass
- 2) Delete factored out properties from original classes
- 3) add new generalizations from all original classes to super class

Benefit:

- new reusable domain class
- smaller domain classes
- connection between existing domain classes

Structure and Elaborate

extract domain component and add enumeration



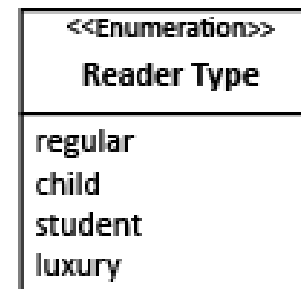
What to do

- 1) Create a new domain class
- 2) Move properties from original class
- 3) add association between classes
- 4) in order to maintain specific information,
add type property with enumeration type

Benefit:

- new reusable domain class
- smaller domain classes
- connection between existing
domain classes

elaborate enumerations



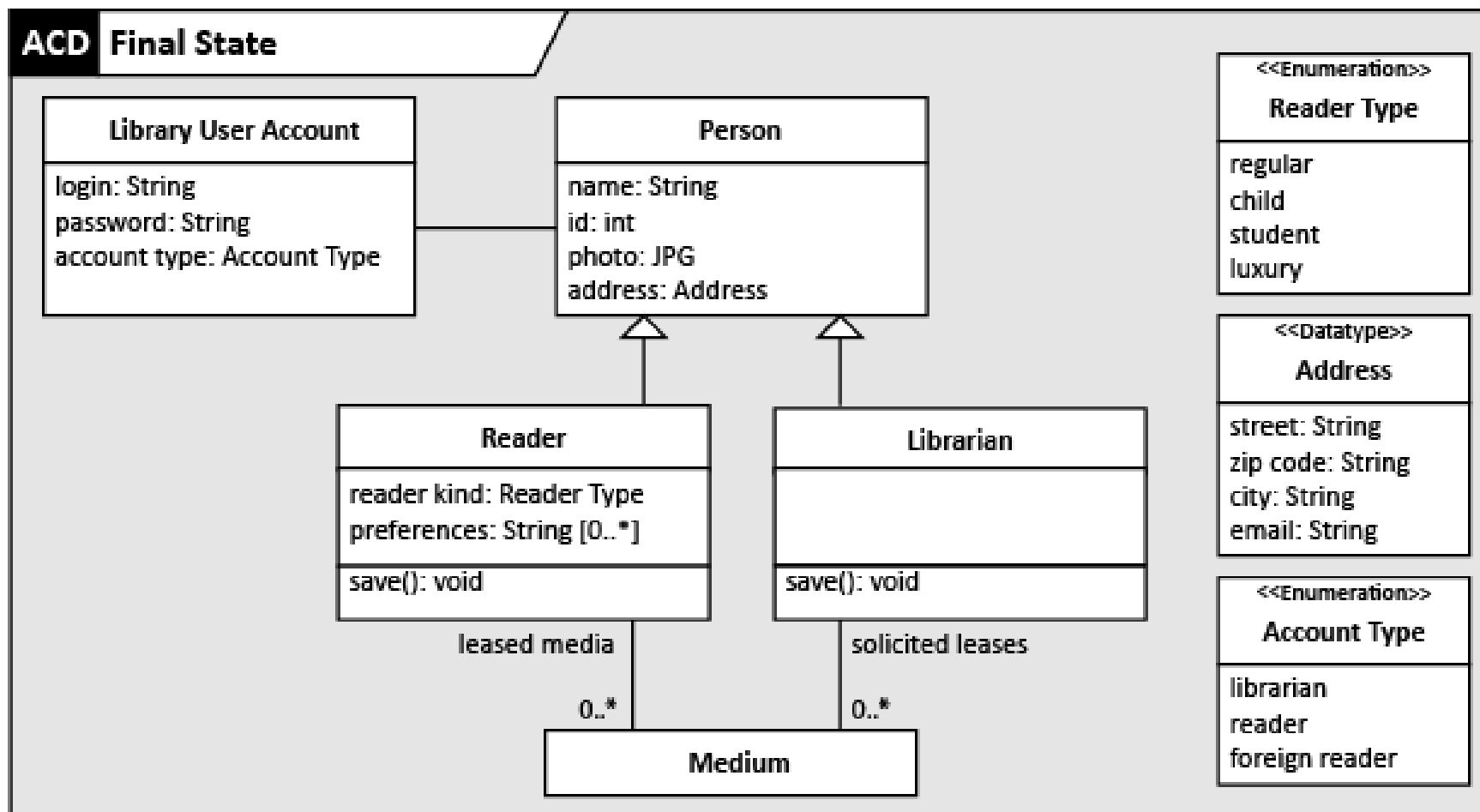
What to do

- 1) Create a new enumeration
- 2) replace the String with new
enumeration

Benefit:

- more specific type#
- more information

Refactoring class models (Example)



Quality Criteria for Information Models

- **After the consolidation effort, the improved information model ought to possess following qualities.**
- **Focus**
 - a focus on the specific entities and their properties and relationship for the particular domain.
- **Balance / Right Size**
 - In terms of size, we typically want to achieve a good balance between the class number (<10#), size (<10 features), and density of associations (<10).
- **Correctness**
 - There is a set of detailed correctness criteria for the elements of information models, see the review criteria in Document QA3 (in version 3.3, Appendices I, J, M, and N, and sections A and E of Appendix L apply).



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 8.4:

Splitting Large Information Models

DTU course 02264

Extract Aspects and Abstraction Levels

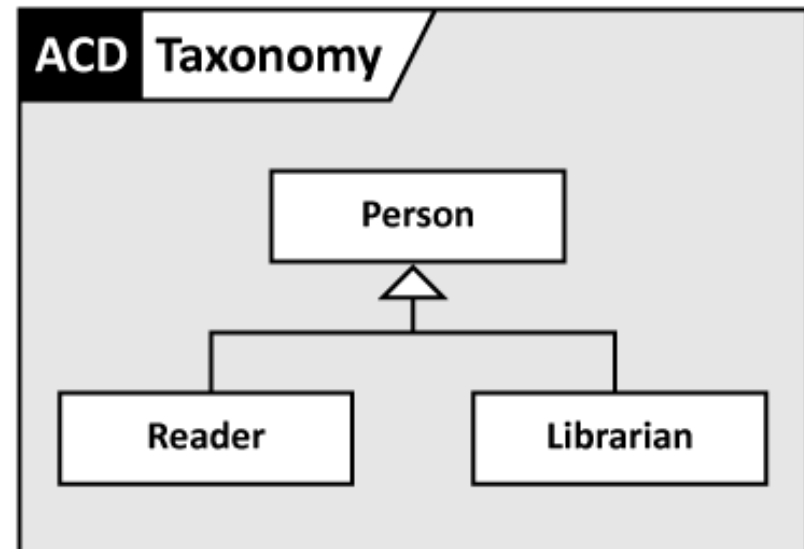
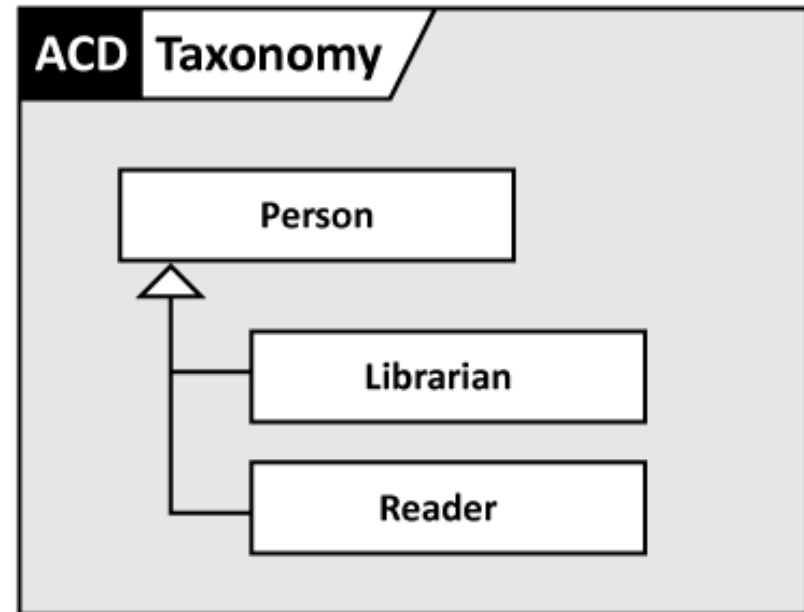
- **After refactoring a class model, there is often the opportunity to extract specific aspects into separate diagrams.**
- **Frequently used aspects include**
 - types and enumerations, and
 - taxonomies and aggregation hierarchies.
- **It is usually sufficient to distinguish between one outline and one detailed view.**
 - In the abstract view, all features (properties and operations) are hidden; most tools allow this as a standard filter. For instance, in MagicDraw, select all diagram elements, and use ?? from the context menu.

Splitting Models into Aspects

- **Another contribution to the same goal is to concentrate individual aspects in special models (and diagrams).**
 - Typical examples are generalization or composition hierarchies.
- **While tiles are notationally symmetric, model aspects are asymmetric, and only a few of them have practical value.**
 - Typically, aspects can only complement and support other structuring means.
- **Tree structured views like taxonomies or composition trees are particularly easy to understand.**
 - For instance, creating a taxonomy of a large and confusing class model or indeed a system written in an OO language is almost inevitable as a start.

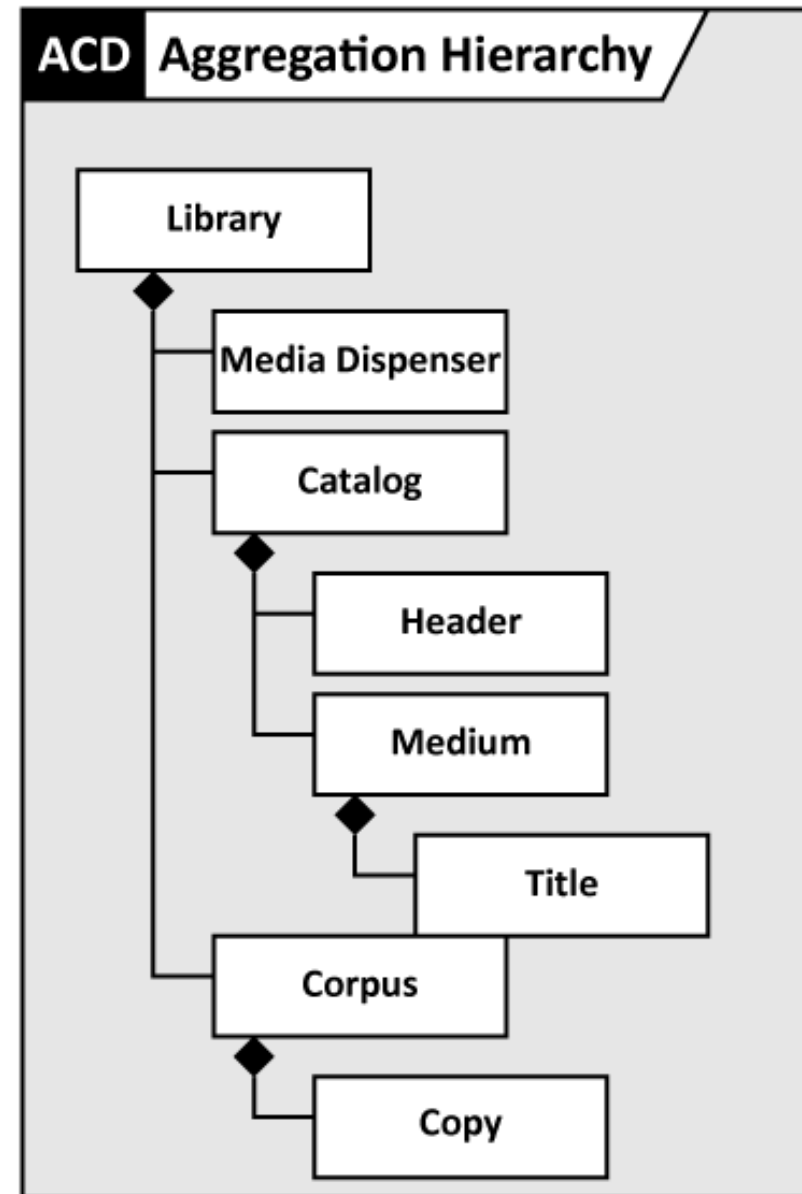
Taxonomy

- A taxonomy groups elements into a specialization tree of classes.
 - The concept has been developed in biology as a tool to help grouping living beings into classes (“taxa”, sing. taxon).
 - A taxonomy contains only classes without features and Generalizations.
 - It is always presented in a tree layout.



Aggregation Hierarchy

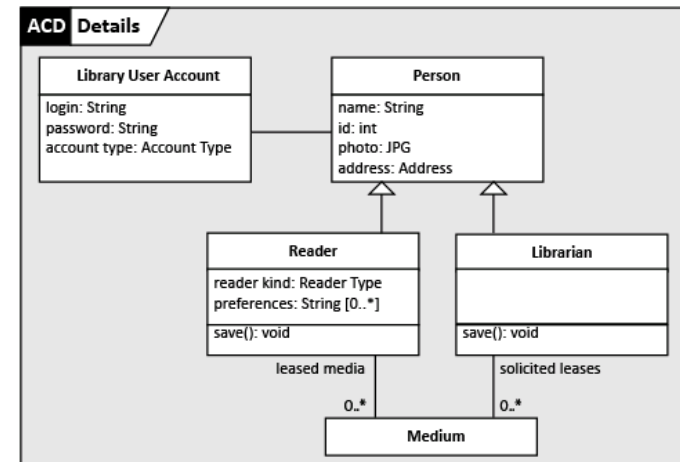
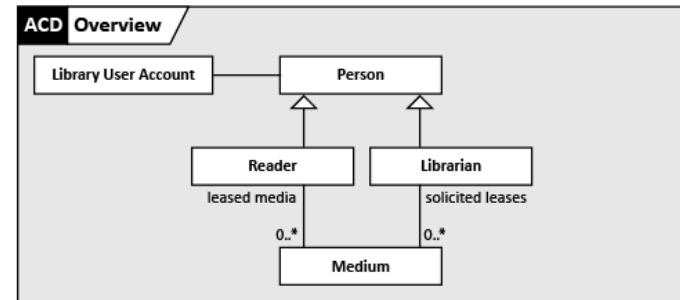
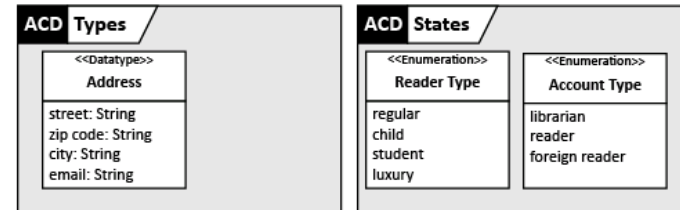
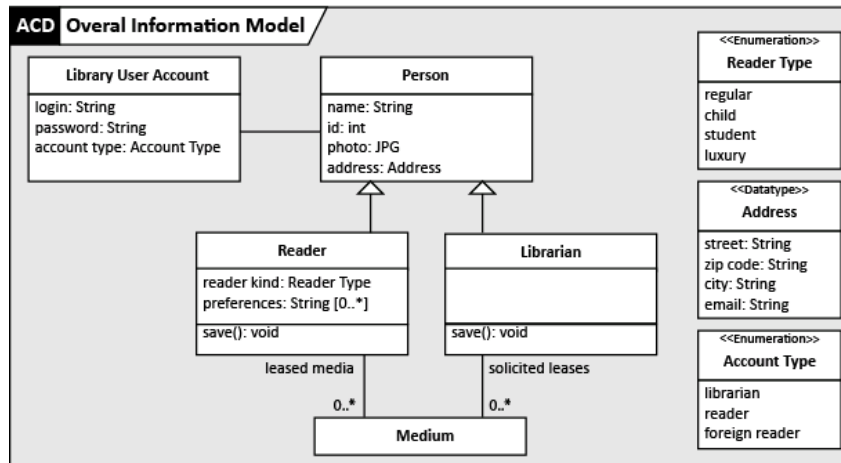
- **An Aggregation Hierarchy composes wholes from Parts.**
 - Aggregation hierarchies are very common in mechanical engineering, where they are also known as part-whole- or composition hierarchies.
 - An Aggregation hierarchy contains only classes without features and composition associations.
 - It is always presented in a tree layout (similar to a taxonomy).



Split large model in to tiles

- **Even after refactoring and extracting aspects and abstraction levels, the model and/or the diagrams may still be to large.**
- **In order to reduce the model to a size easily handled, we may have to split the diagram into several ones focusing on different topics.**
 - Select a small number of very important classes, each of which can be moved to a different “tile” together with immediately adjacent classes.
 - Tiles should correspond to elements of the domain architecture.
 - Each class should be defined on exactly one tile, but might have to be referred to from several other tiles – those references should be distinguished visually.
- **This tiling can be deepened further by creating separate packages for each tile.**
 - This also allows to split up previously extracted aspects into tile-specific subsets, which should be included in the tile’s package.
 - This should only be done when these aspects themselves are too large to be kept together.
 - Aspects that cut across tiles should be placed one level above the tiles in the package structure.

Splitting up class models



Restructuring Large Information Models

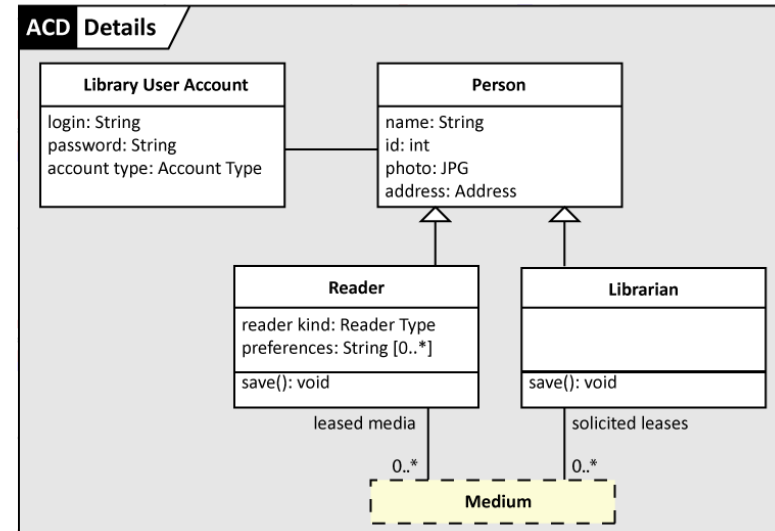
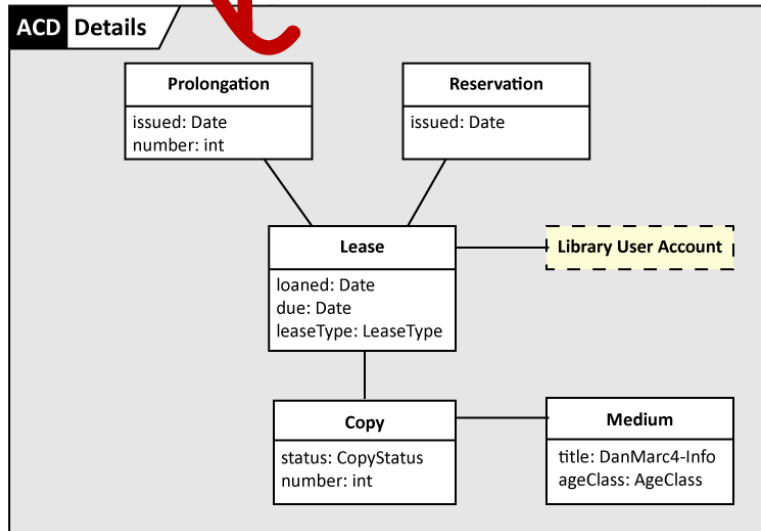
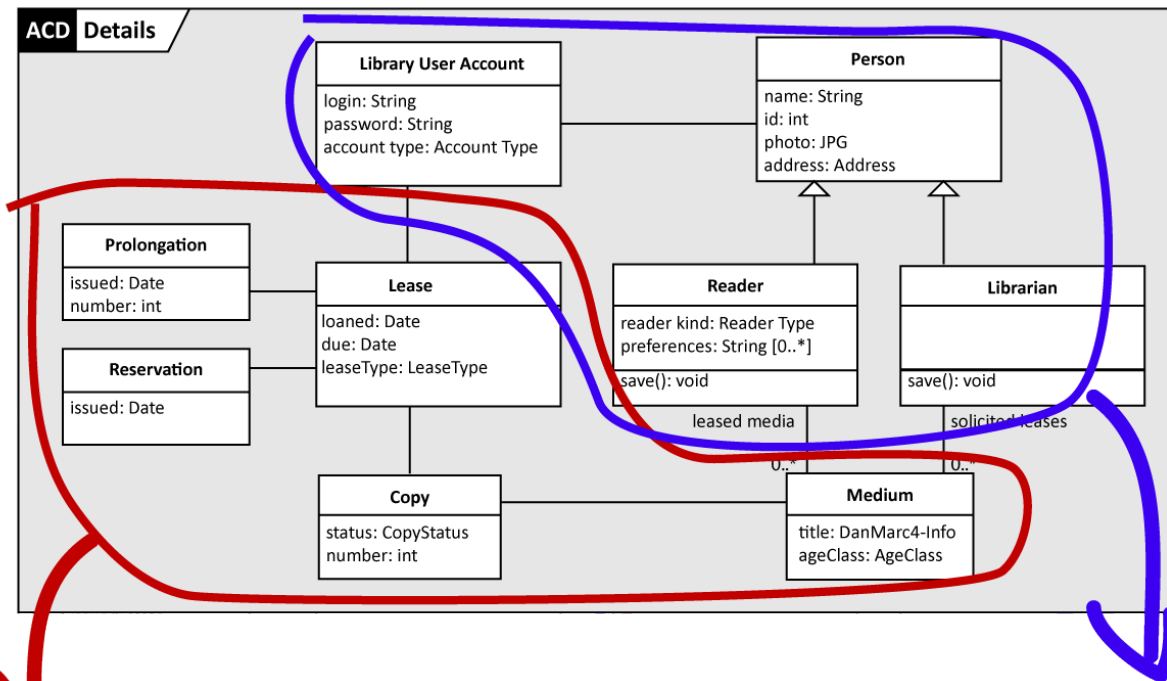
- **Sometimes, information models become rather large over time.**
 - We could create larger and larger print outs to stick up at our office's wall to handle the complexity, but that is clearly no sustainable solution.
- **A more methodical approach for splitting up information models is needed. There are basically four ways of doing this.**
 1. Split one large model into several independent modules using UML packages and import-relationships between them.
 2. Split one diagram into several partially overlapping diagrams that collectively cover the whole model.
 3. Move isolated information model fragments like individual concerns, or enumerations and data types to separate diagrams ("tiles").
 4. Concentrate some aspects like generalization or composition in specialized diagrams.
- **Each of these approaches has their own benefits and drawbacks, and they should be combined where appropriate.**
 - To some degree, these techniques also apply to other model types.

Splitting Information Models into Tiles

- **A lightweight way of splitting up large information models is to simply use a set of overlapping diagrams to present separate portions of the class model individually („tiles“).**
 - All the tiles are equal in generality, expressive means used, and importance.
- **Of course, this will not work well for large numbers of tiles.**
 - But tiles can easily be created such that they fit well on printed pages.
- **Again, the UML standard itself provides good examples of this technique.**
 - There, the meta model concerned with the abstract syntax of static structure models (i.e., class models) is spread out over 16 diagrams.
 - Looking for all relationships of one class is rather difficult to do using just these diagrams.
 - Creating a complete taxonomy of part or all of UML is a very tedious task.

Split Information Models into tiles

create one
maybe spl





Prof. Dr. Harald Störrle

Software Engineering Section
Department of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads
Building 322, Room 024
DK-2800 Kgs. Lyngby

hsto@imm.dtu.dk
www.imm.dtu.dk/~hsto

