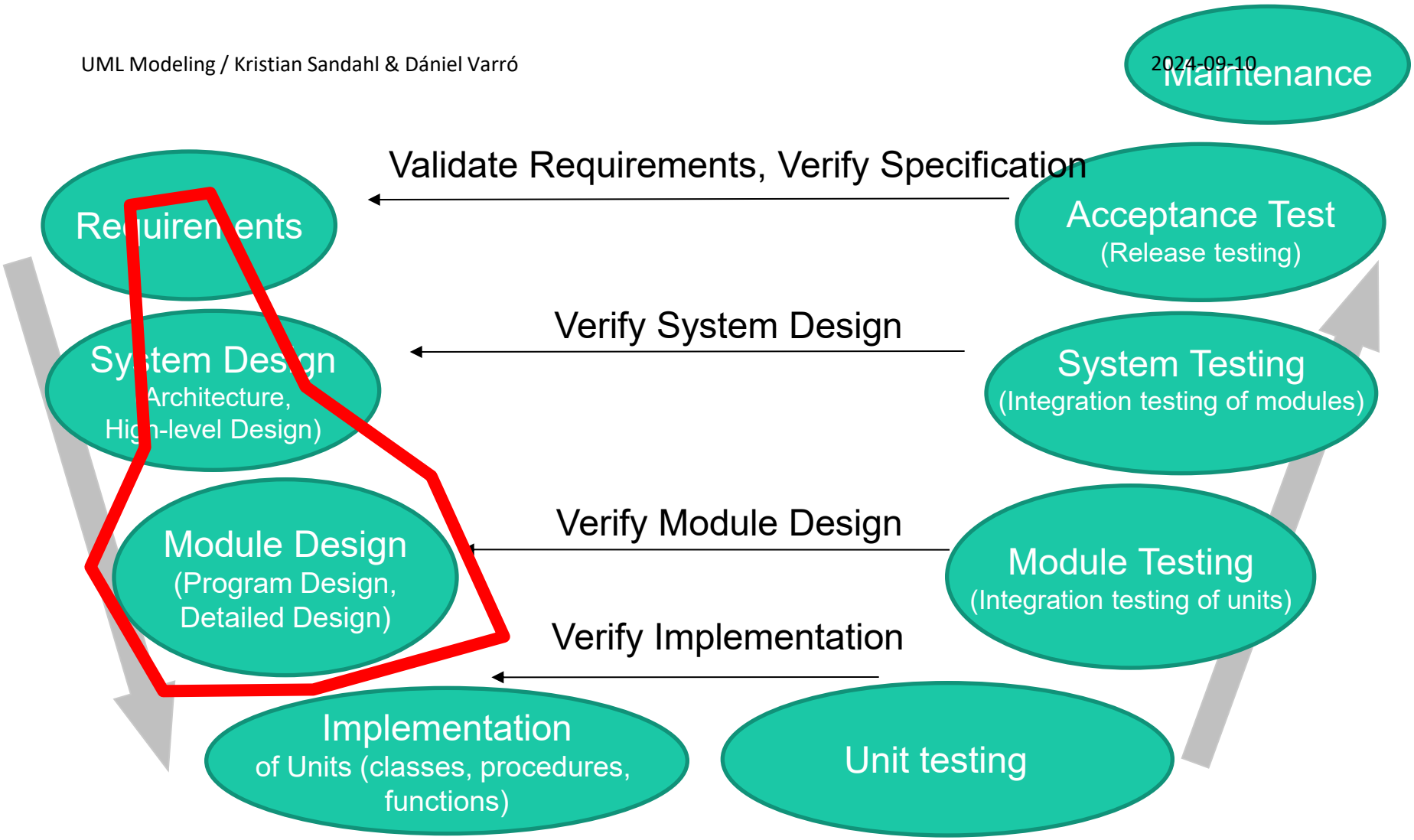
Modeling with UML

Dániel Varró / Kristian Sandahl

LINKÖPING UNIVERSITY

UML in Software Engineering


Maintenance
Validate Requirements, Verify Specification
Requirements
Acceptance Test
(Release testing)
Verify System Design
System Design
(Architecture,
High-level Design)
System Testing
(Integration testing of modules)
Verify Module Design
Module Design
(Program Design,
Detailed Design)
Module Testing
(Integration testing of units)
Verify Implementation
Implementation
of Units (classes, procedures, functions)
Unit testing
Project Management, Software Quality Assurance (SQA), Supporting Tools, Education

The goals of module design

- Provide the expected function
- Prepare for change:
  - Separation of concern
  - Testability
  - Understandability
- Contribute to quality, e.g.:
  - Performance
  - Usability
  - Reliability
  - …
- Map for the implementers, testers, and maintainers
- Provide detailed specs for the internal content and interface of a module

Modelling software

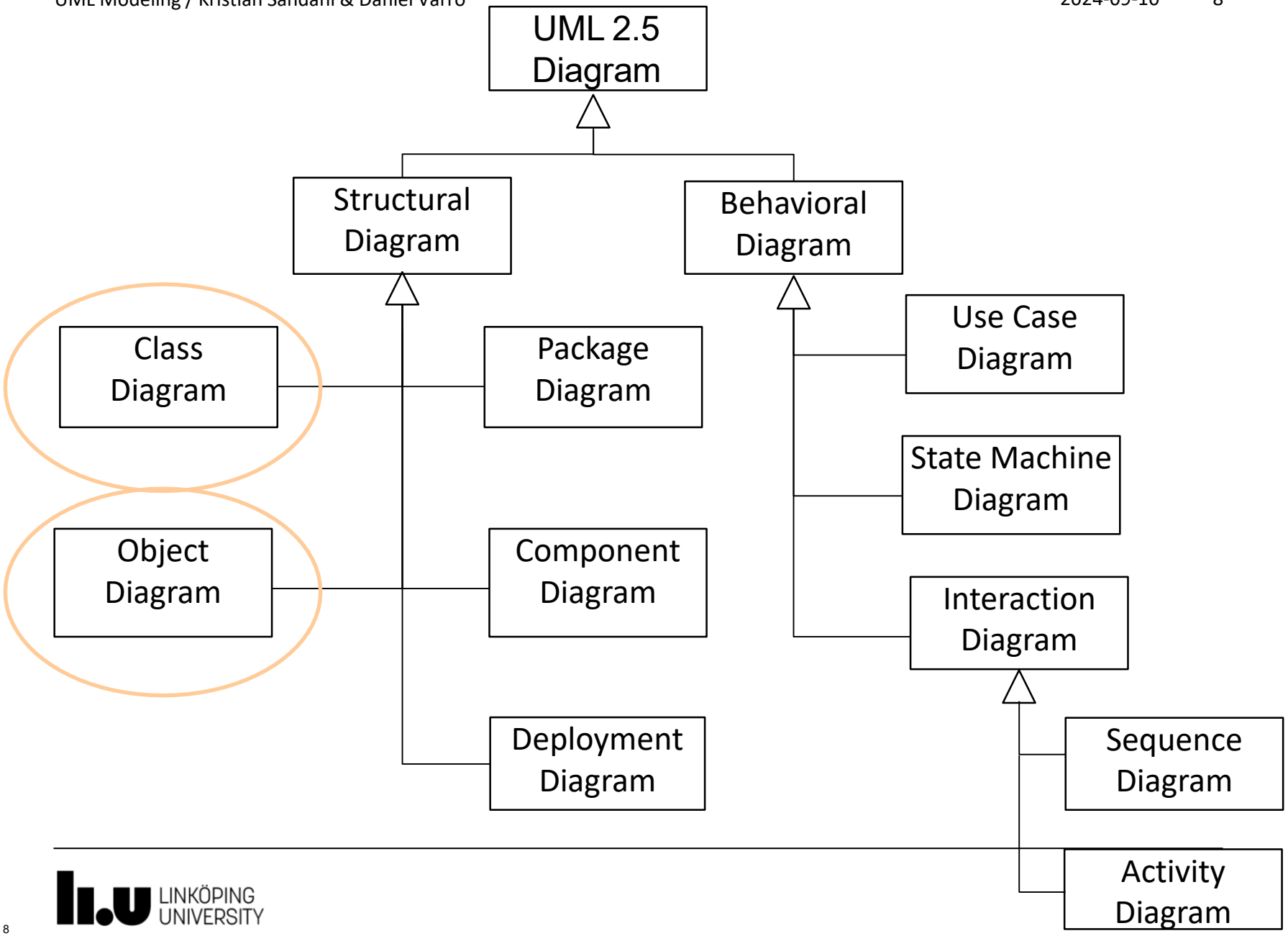
- Models **supplement** natural language
- Models support both elicitation and design
- Models can generate code and test cases
- The boundaries between specification and design have to be decided
- **UML** has become the standard notation
- Industry interest in SysML –
  extends UML (and defined in UML)

Unified Modeling Language

- Wide-spread standard of modeling software and systems
- Several diagrams and perspectives
- Often needs a text of assumptions and intenti
- Many tools tweak the standard, we use UML

UML Class and Object Diagrams

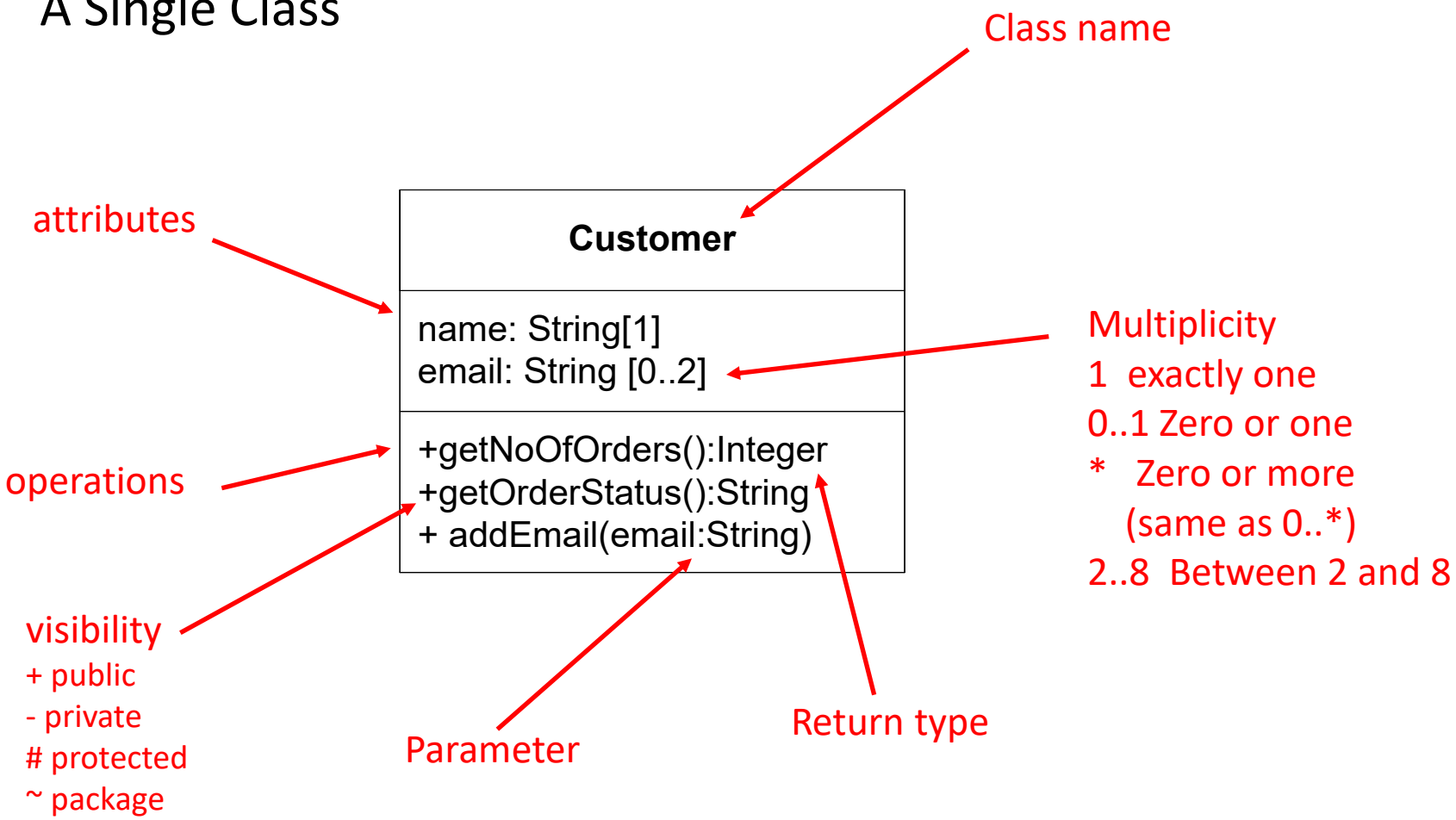
Well-known Diagrams of UML



Where to use Class diagrams?

- **Domain modeling**: Capture key concepts and relations in a domain
  - Ontologies
  - Metamodels
- **Database design**:
  - E.g. used by object-relational mappings (Hibernate)
  - User code manipulates objects → serialized in Rel DB
- **Component / module design**:
  - Internal structure of components / modules
- Defines structure of various serialization formats
  - XMI: XML Metadata Interchange (modeling tools), JSON

A Single Class



Naming of Classes

- Noun
- Singular
- First letter capitalized;
- Use Pascal case (camel case with capital initials) without spaces if needed

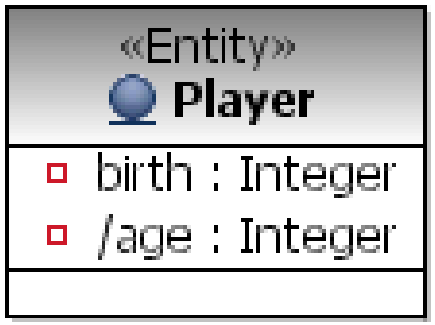
- Not too general, not too specific – at the right level of abstraction
- Avoid software engineering terms (data, record, table, information)

- Good: Hospital, Doctor, PartTimeEmployee
- Bad: register, Hospitals, doctor, PartTimeEmployeeData

Attributes

- Each attribute shall have
  - Name: e.g. birth
  - (Primitive) Type:
    - E.g. String, Integer, Real, Date, …
  - Example:
    - `Integer birth;`
- Each attribute may
  - Specify default value
  - Be derived: e.g. age
    - Calculated from other values

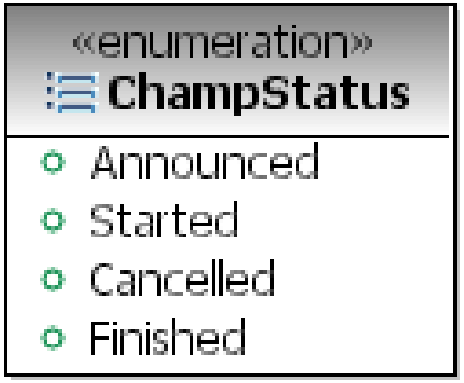
    `age = currYear – birth`



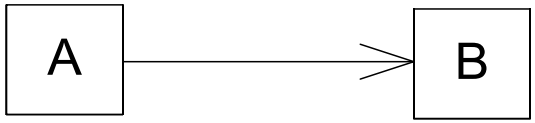
Naming of attributes
- Small initial letter
- Followed by camel case
- Without spaces
- Typically singular

Enumerations

- Enumeration:
  - a fixed set of symbolic values
  - represented as a class with values as attributes
- Usage:
  - Frequently define possible states
  - Use enumerations instead of hard-wired String literals whenever possible

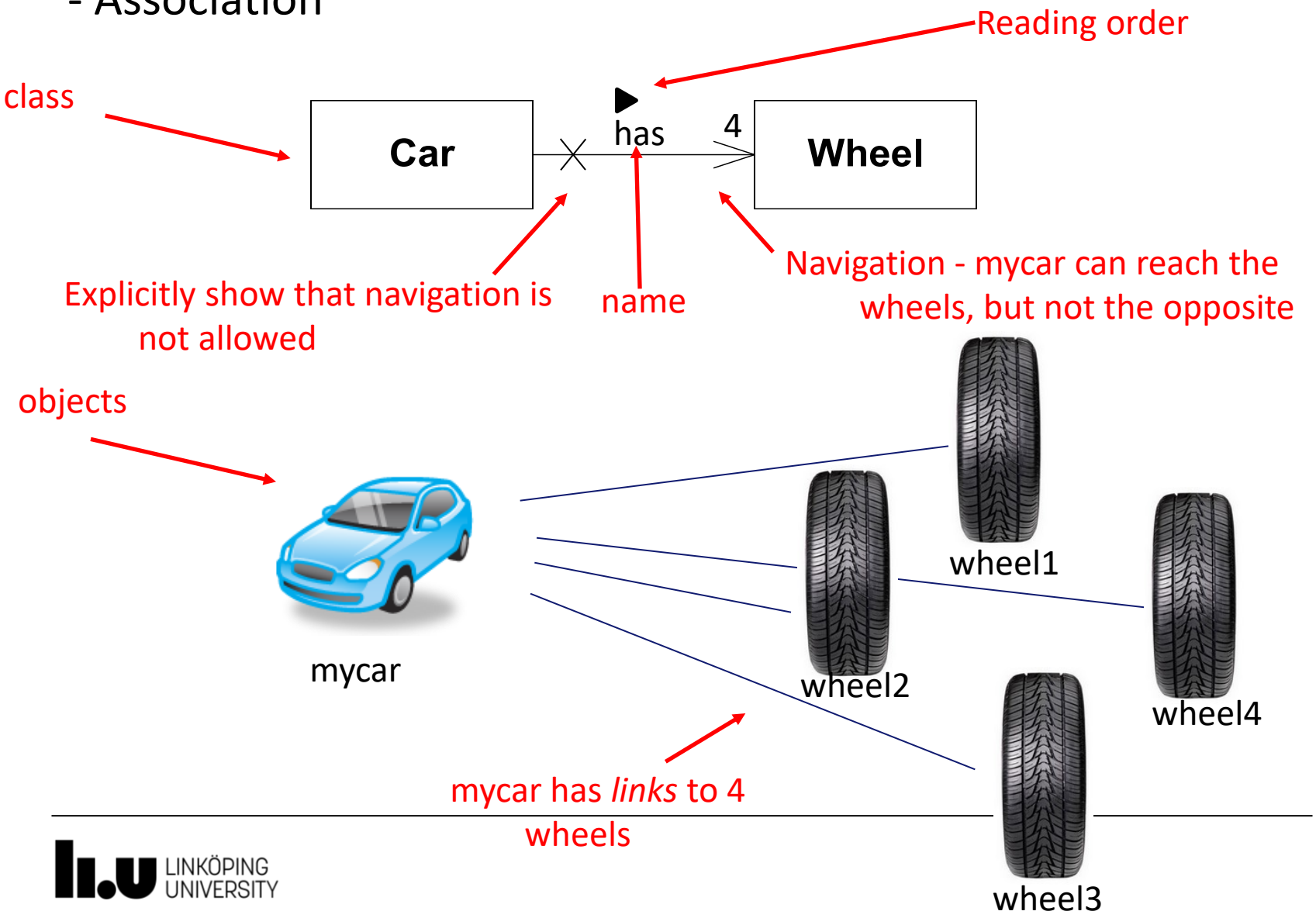


Relationships (1/6) - overview and intuition - Association

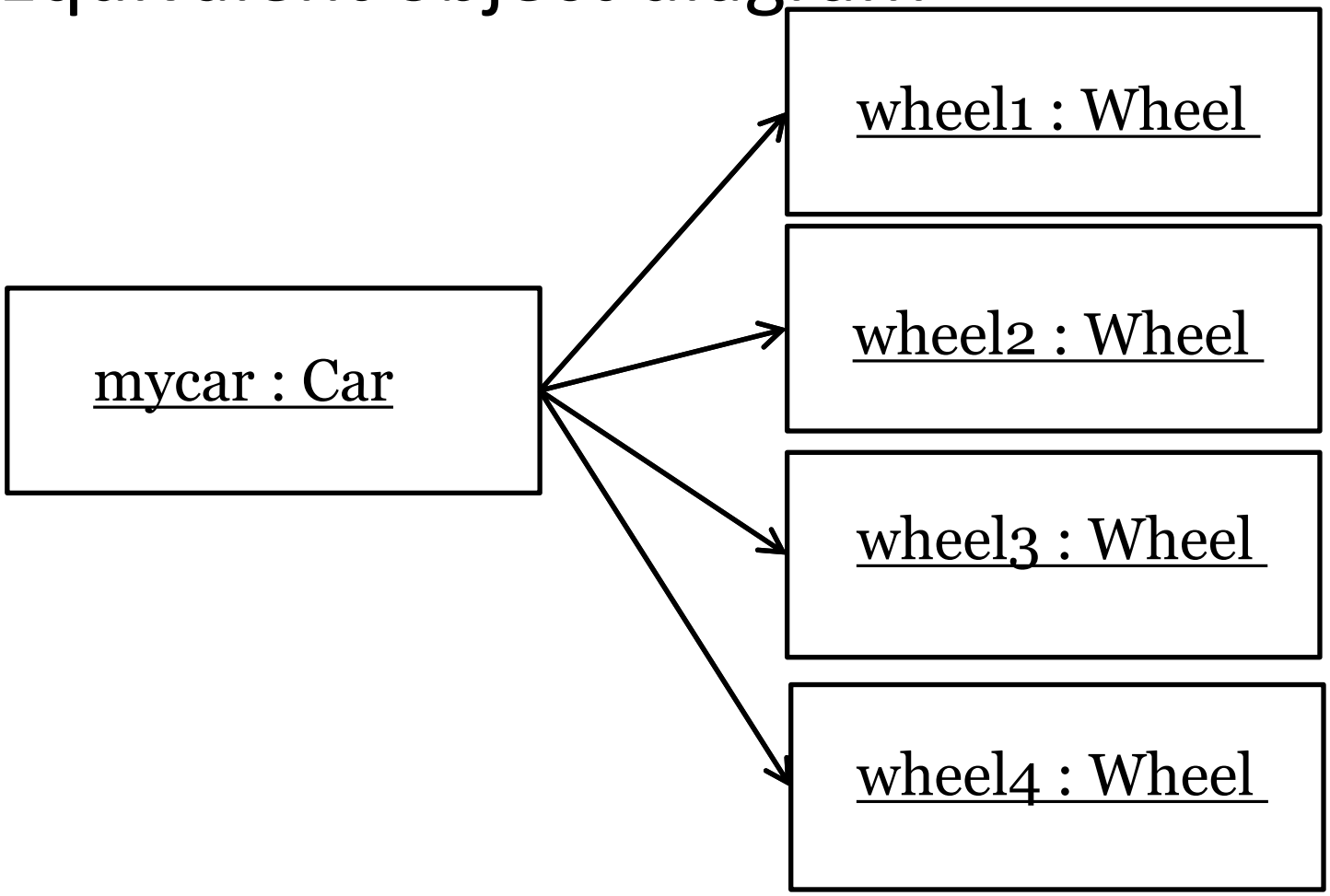


Association
(with navigability)

Relationships (1/6) - overview and intuition - Association

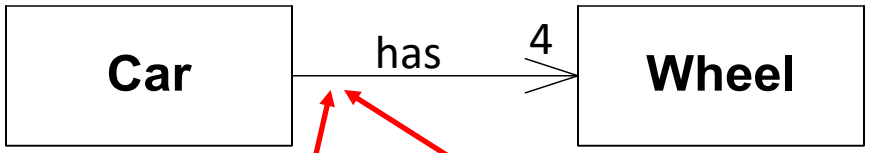


Equivalent object diagram



Relationships (1/6) - overview and intuition

- Association



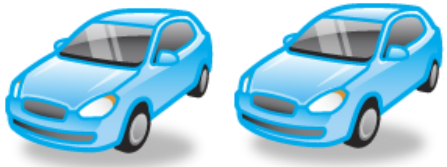
What does it mean to have a * here?

What if we have multiplicity 1 instead?

"*"

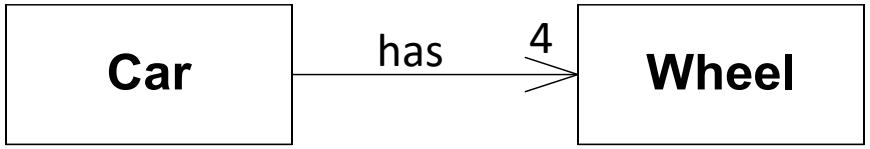
A wheel can be linked to more than one car instance

"1"



A wheel can only be liked to one car instance

Relationships (1/6) - overview and intuition - Association



Associations are the "glue" that ties a system together (by introducing a graph model)

association instance = *link*

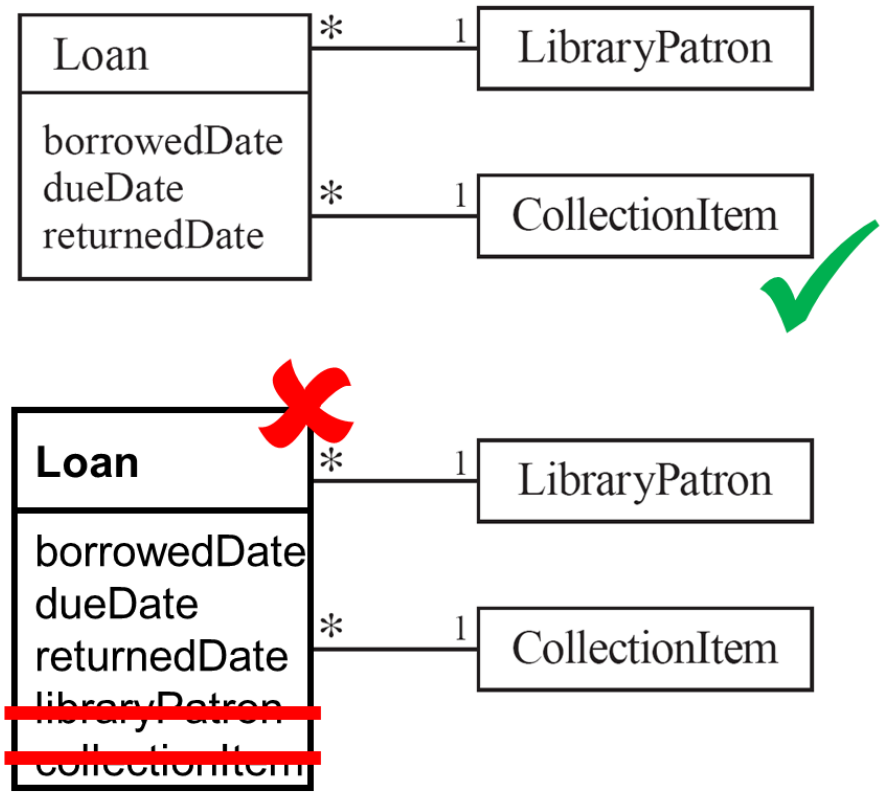
mycar1

A link (association instance) describes a *relation* between objects at run-time.

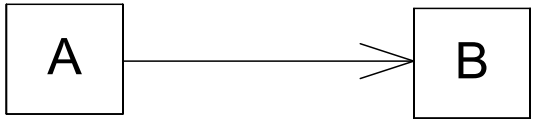
{(mycar1,wheel1),
(mycar1,wheel2),
(mycar1,wheel3),
(mycar1,wheel4)}

Attributes vs. Associations

- **Common mistake**: duplicate an association by adding an **attribute** in the class diagram
- An association is already implemented as a **field**
- A Java class for Loan that corresponds to the above class diagram will have the *borrowedDate*, *dueDate*, and *returnedDate* fields, but also a field for *libraryPatron* and another one for *CollectionItem*
- No need to add it again!
- Attributes have primitive types!



Relationships (2/6) - overview and intuition - Aggregation



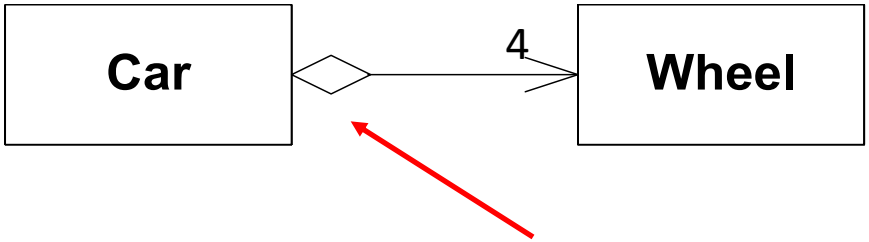
Association
(with navigability)

"A" has a reference(s) to instance(s) of "B". Alternative: attributes

Aggregation

Relationships (2/6) - overview and intuition - Aggregation

Common vague interpretations: "owns a" or "part of"



What does this mean? What is the difference to association?

Vague definitions → Inconsistency and misunderstandings

Aggregation was added to UML with little semantics. Why?

Jim Rumbaugh
"Think of it as a modeling placebo"

Recommendation: - Do not use it in your models.
- If you see it in other's models, ask them what they actually mean.

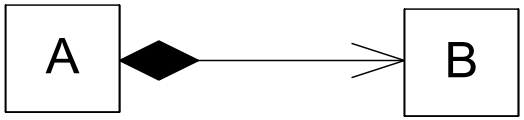
Relationships (3/6) - overview and intuition - Composition

Association
(with navigability)

"A" has a reference(s) to instance(s) of "B". Alternative: attributes

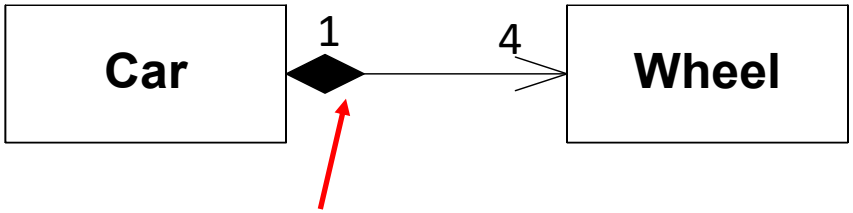
Aggregation

Avoid it to avoid misunderstandings



Composition

Relationships (3/6) - overview and intuition - Composition

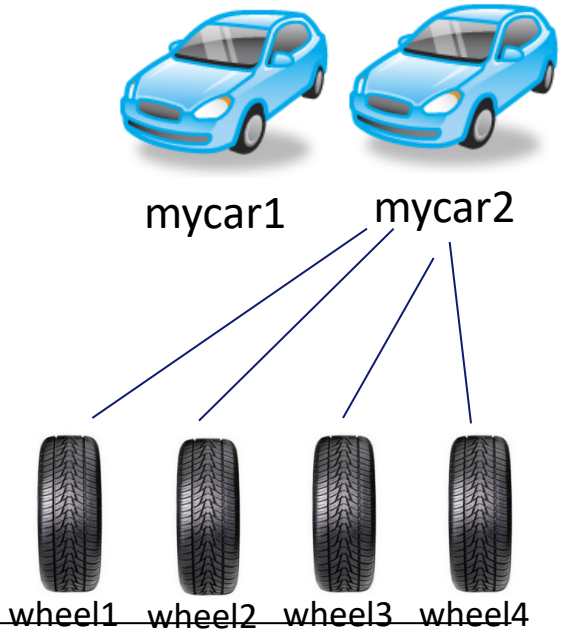


Any difference to association?

Yes! First, multiplicity must be 1 or 0..1. An instance can only have one owner.

But, isn't this equivalent to what we showed with associations?

Well, in this case...

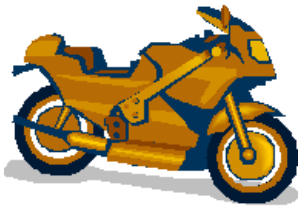


Relationships (3/6) - overview and intuition - Composition

Using composition...

mycar1

Ok for wheels to be part of mycar1 or mybike1



mybike1

wheel1 wheel2 wheel3 wheel4

wheel5 wheel6

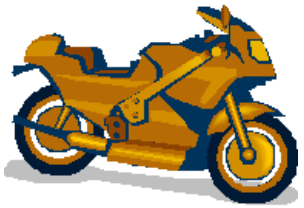
Relationships (3/6) - overview and intuition - Composition

Using composition...

Can mycar1 and mybike1 share the same wheels?

NO!
Not with composition!

mycar1



mybike1

wheel1 wheel2 wheel3 wheel4

Key concepts
- "No sharing" rule
- The owner is responsible for managing its parts, e.g. allocation and deallocation.

Relationships (3/6) - overview and intuition

- Composition

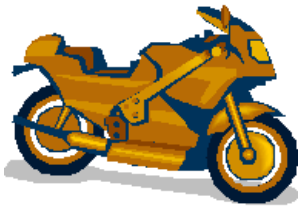
(Note the difference. The diamond is removed.)

Using associations...

Can mycar1 and mybike1 share the same wheels this time?

mycar1

Yes! Associations do not have a "no sharing" rule.



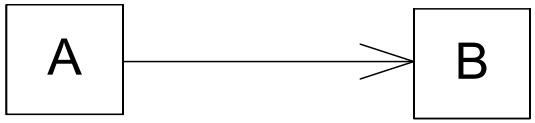
mybike1

However, in this case it is a strange model...

wheel1 wheel2 wheel3 wheel4

wheel5 wheel6

Relationships (4/6) - overview and intuition - Generalization

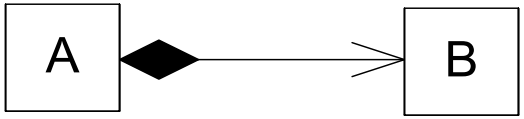


Association
(with navigability)

"A" has a reference(s) to instance(s) of "B". Alternative: attributes

Aggregation

Avoid it to prevent misunderstandings



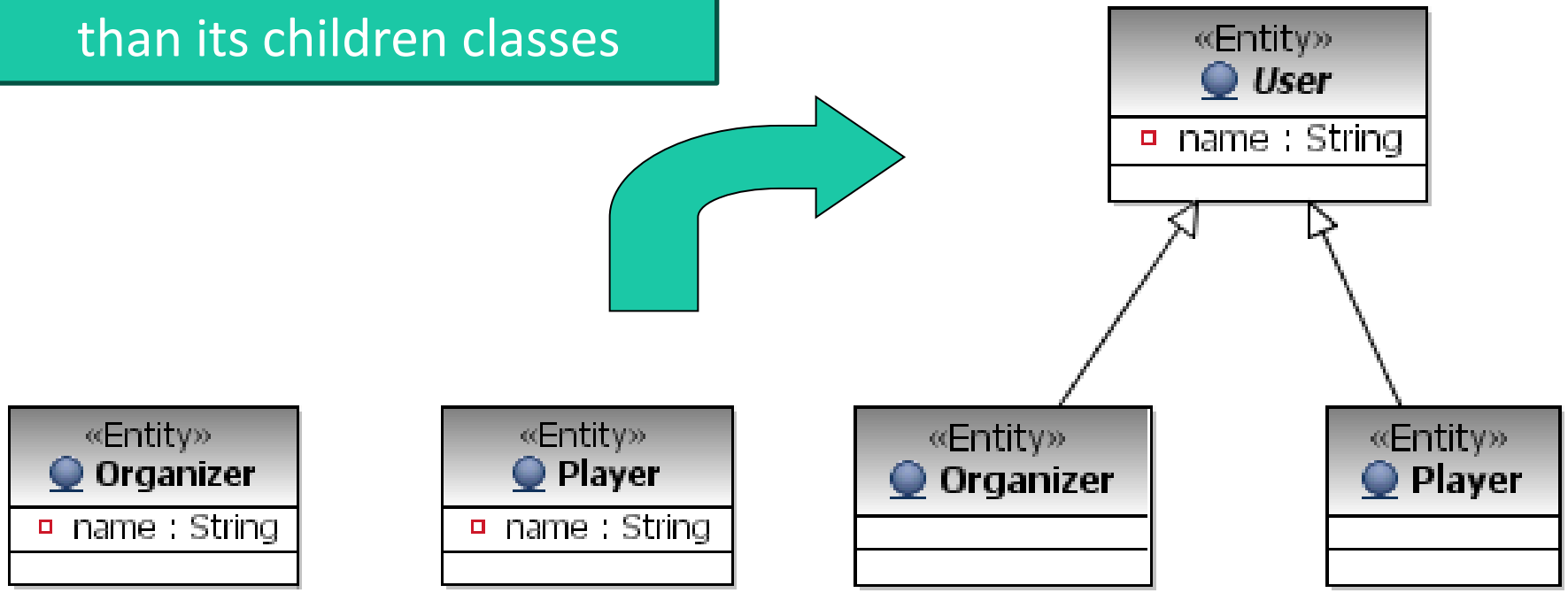
Composition

An instance of "B" is part of an instance of "A", where the former is not allowed to be shared.

Generalization

Typical Use of Generalization

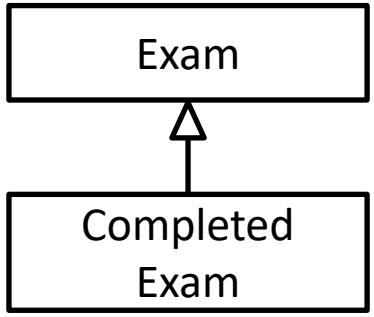
Parent class is more general than its children classes



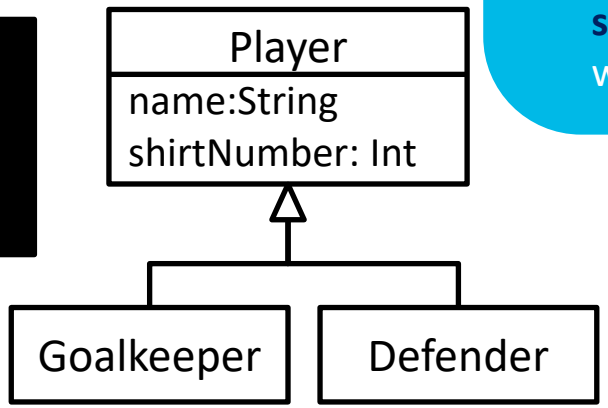
Aim: Lift up common attributes and methods to the superclass

When to Avoid Generalization?

State information should ***NOT*** be a class



Child classes with empty content and no new relationships



A checklist for using generalization

- Check generalizations to ensure they obey the **is-a rule**
- An object **cannot change its type**/class during its lifecycle
- All inherited features must **make sense** in each subclass
- A class must be **different from its superclass and other subclasses** wrt. behavior or structure

Relationships - overview and intuition

Conceptual models, domain models

Association
(with navigability)

"A" has a reference(s) to instance(s) of "B". Alternative: attributes

Aggregation

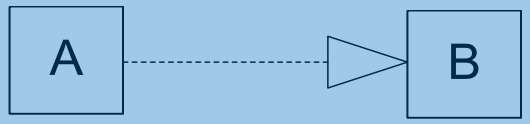
Avoid it to avoid misunderstandings

Composition

An instance of "B" is part of an instance of "A", where the former is not allowed to be shared.

Generalization

1) "A" inherits all properties and operations of "B".
2) An instance of "A" can be used where a instance of "B" is expected.



Realization

"A" provides an implementation of the interface specified by "B".

~~Dependency~~

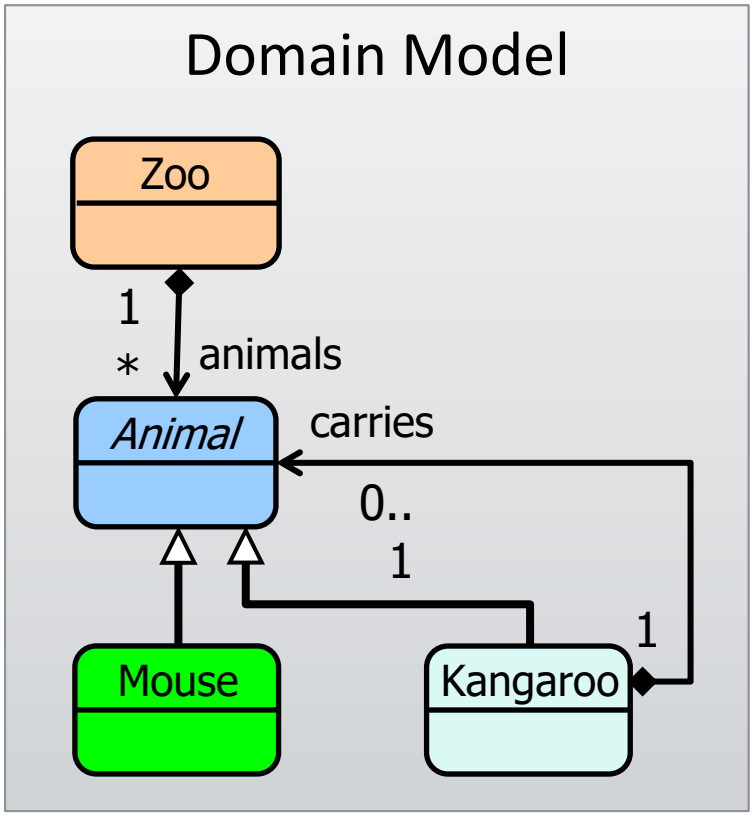
"A" is dependent on "B" if changes in the definition of "B" causes changes of "A".

LINKÖPING UNIVERSITY

Design models, architecture models, implementation models

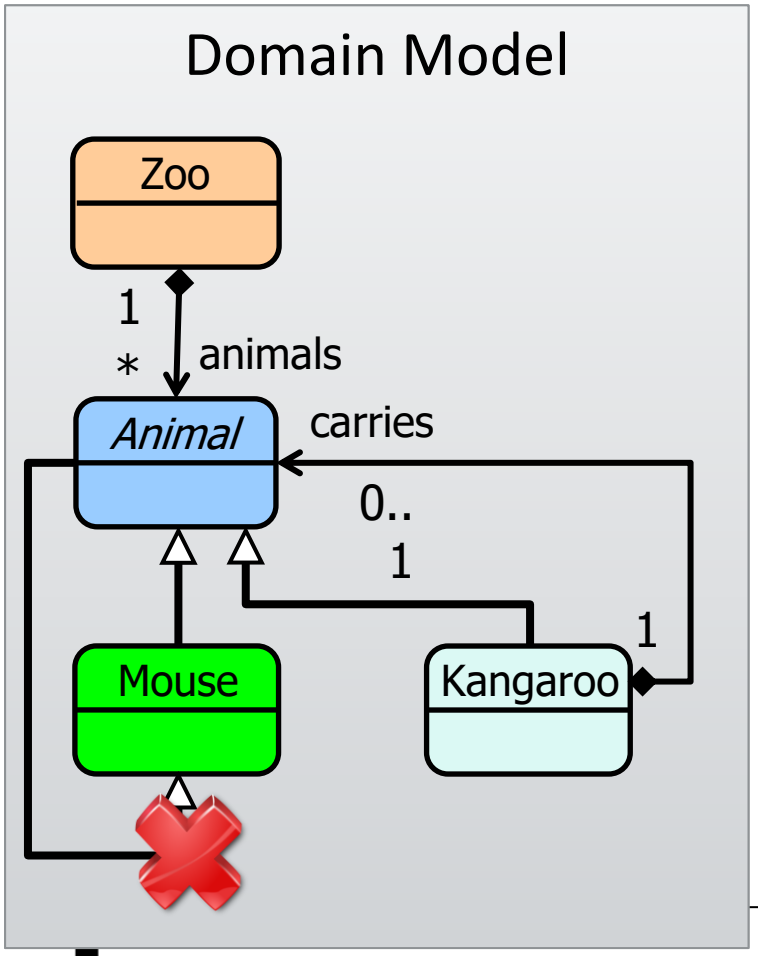
Consistency of UML Class Diagrams by Example

Class Diagram for Animals in Zoo



- There are animals in the zoo
- Animal is an abstract class (it cannot have direct instances)
  - Notation: Slanted (italic) text
- Animal has two subclasses
  - Mouse and Kangaroo
- A kangaroo may carry an arbitrary animal in her pouch

Class Diagram for Animals in Zoo

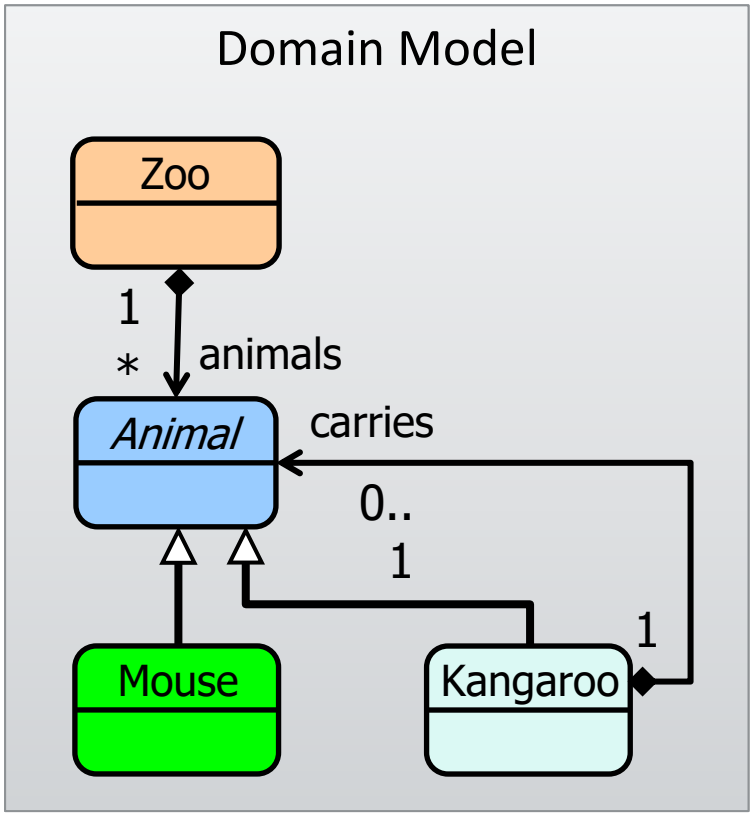


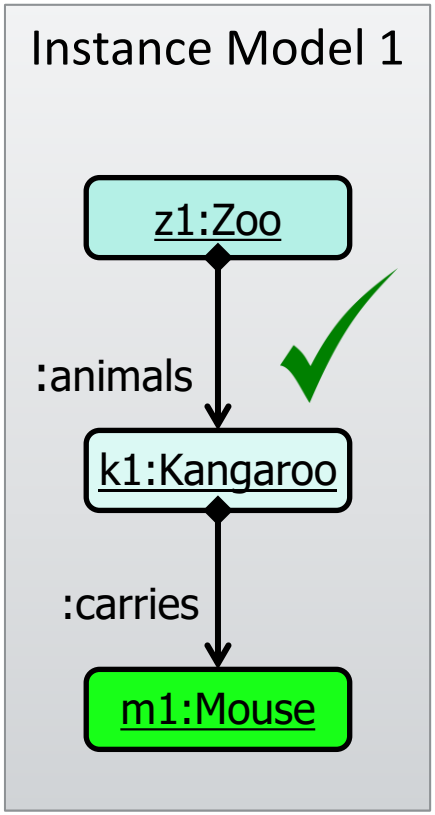
Is this a well-formed domain model?

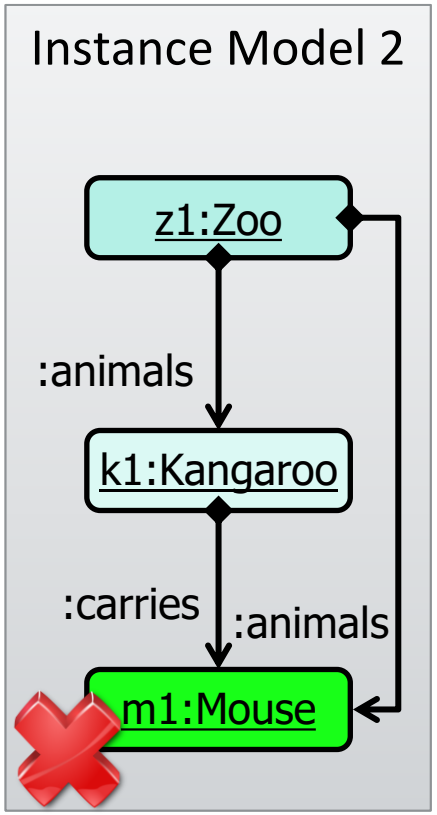
No!

Circularity in the generalization hierarchy is disallowed in a domain model

Instance Models for Animals in Zoo





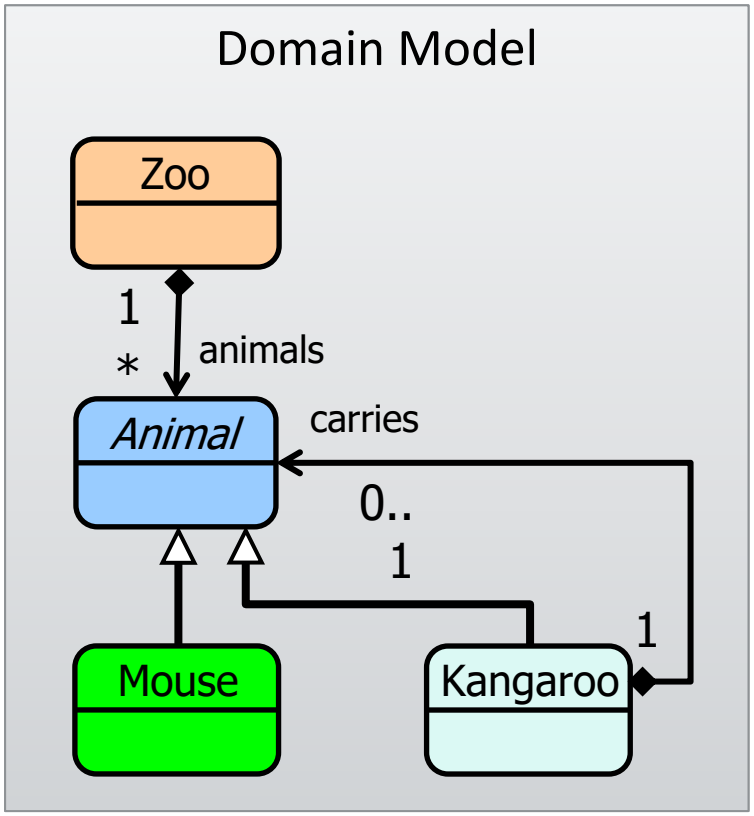


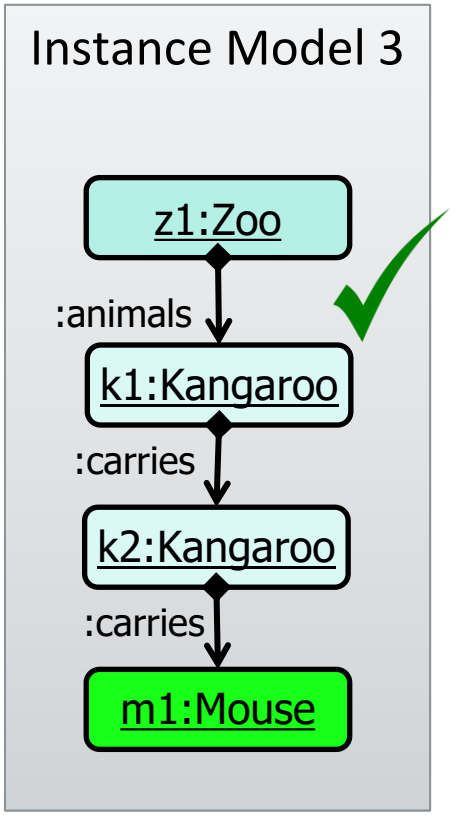
Is this a well-formed instance model?

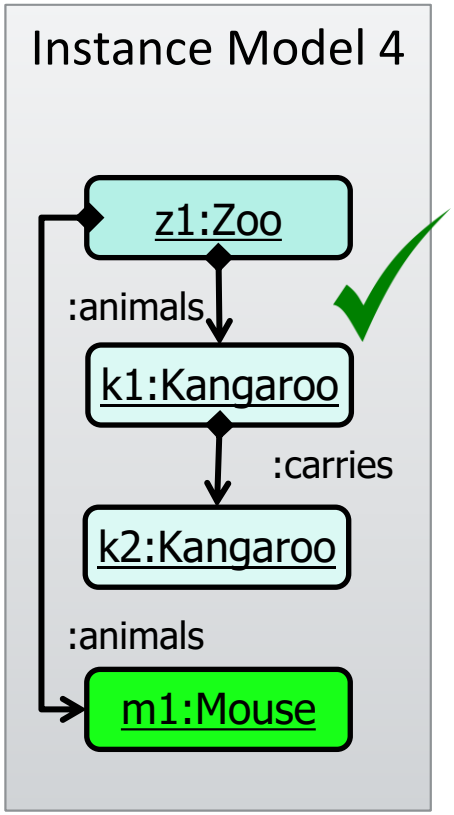
IM1: Yes

IM2: No! → m1 contained by multiple objects

Instance Models for Animals in Zoo





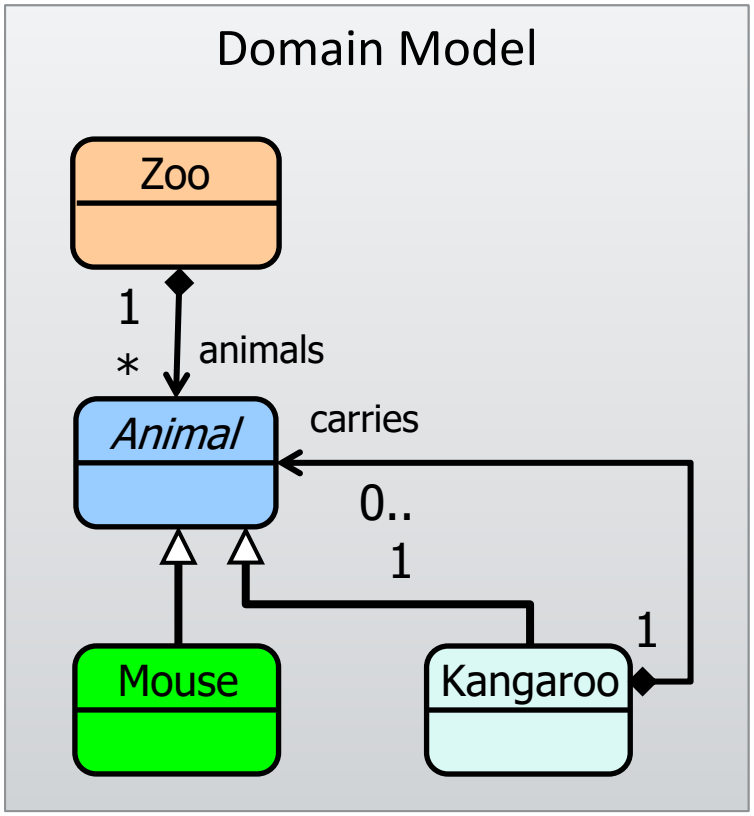


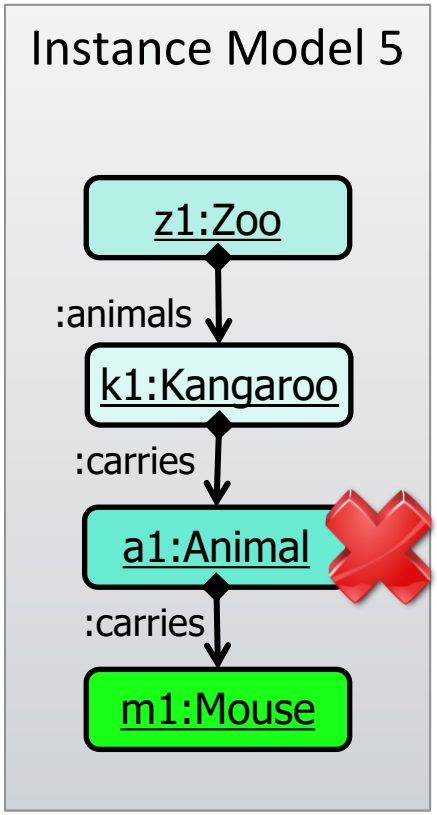
Is this a well-formed instance model?

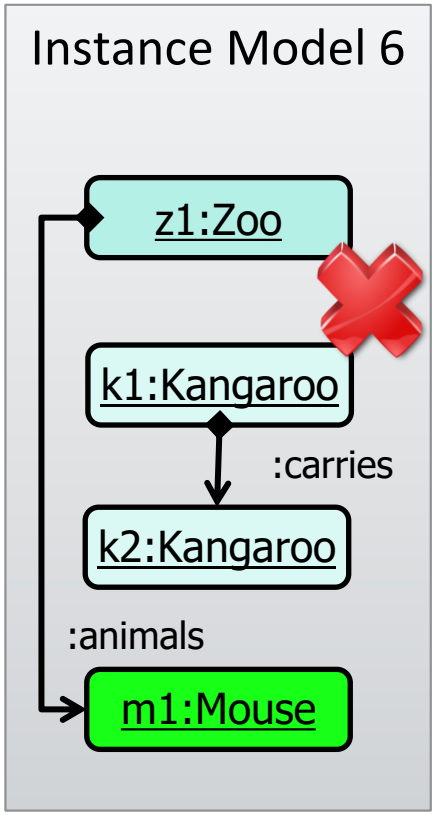
IM3: Yes

IM4: Yes

Instance Models for Animals in Zoo





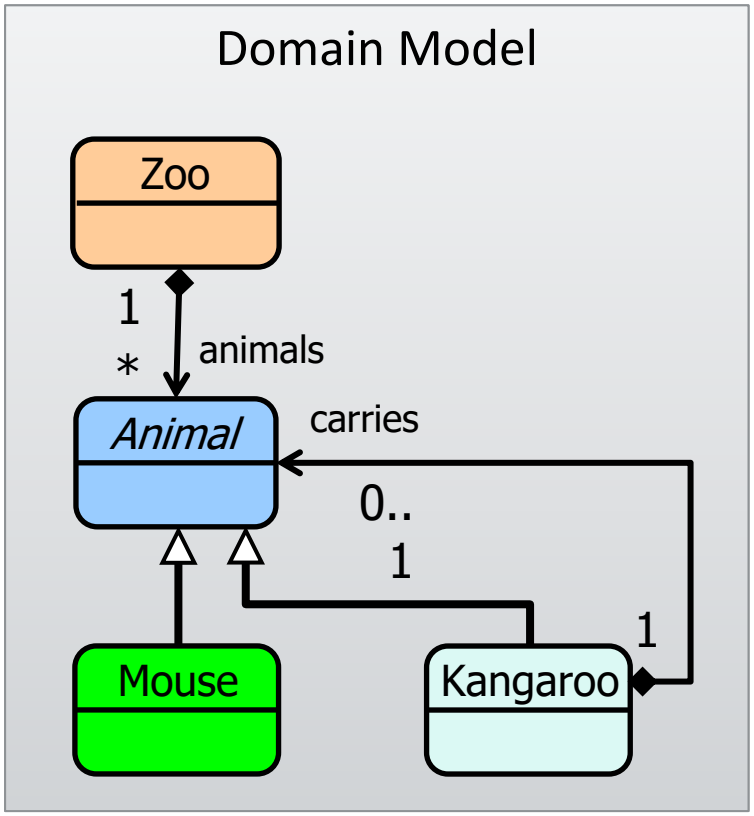


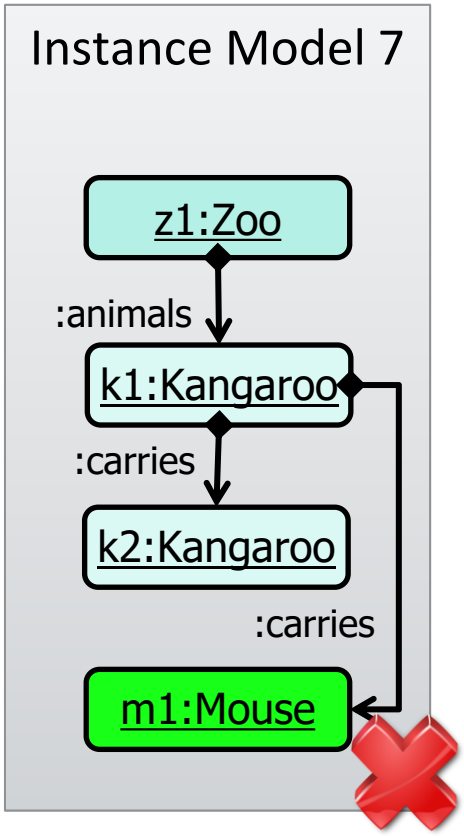
Is this a well-formed instance model?

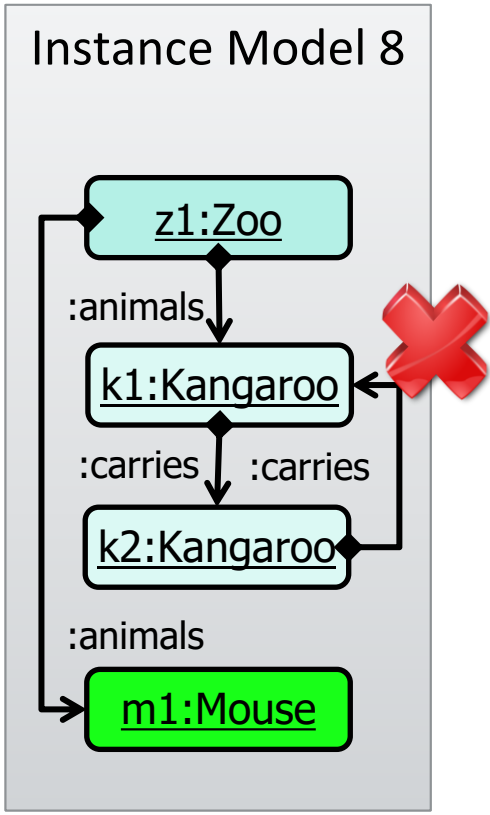
IM5: No! → Animals cannot have direct instances (like a1)

IM6: No! → Kangaroo k1 is not contained by any objects

Instance Models for Animals in Zoo





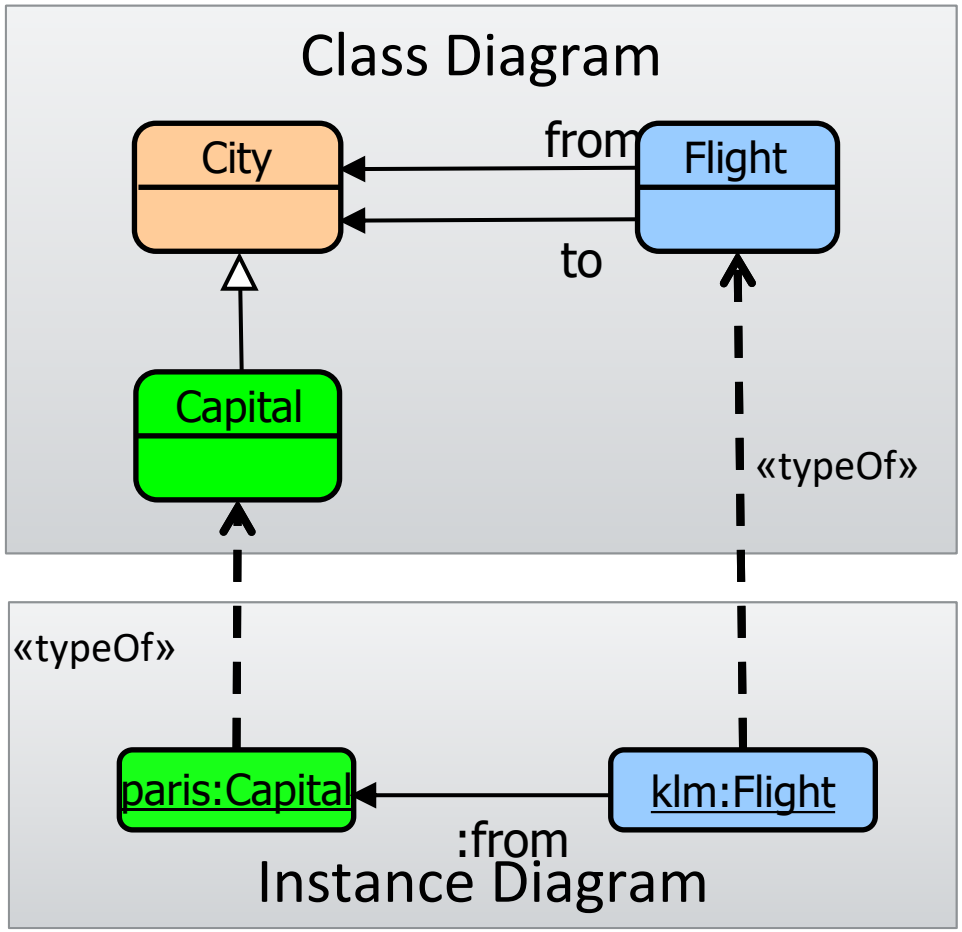


Is this a well-formed instance model?

IM7: No! → Multiplicity of ***carries*** association end is violated (0..1)

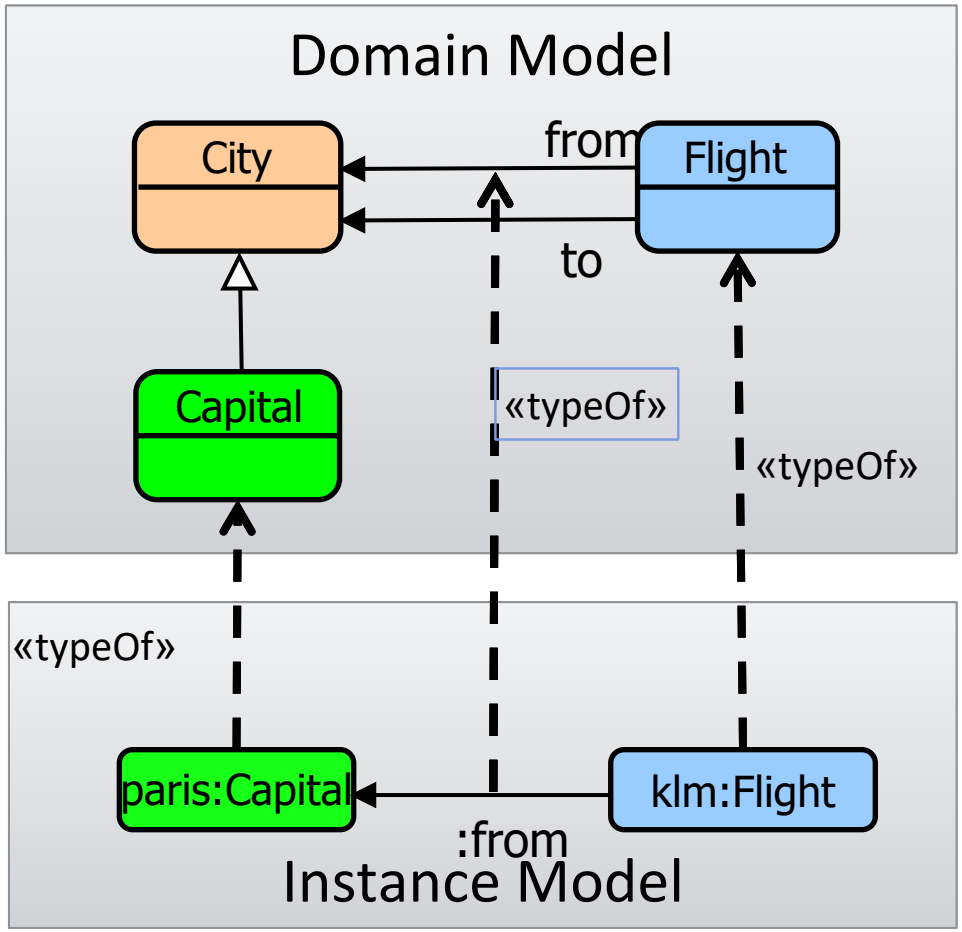
IM8: No! → No circularity in the containment hierarchy in instance models

Classification (aka. Instantiation)



- Each object is ***an instance of*** a class
- **Direct type**:
  No other types exist lower in the class hierarchy
  - paris:Capital, klm:Flight
- **Indirect type**:
  Superclass of the direct type
  - paris:City

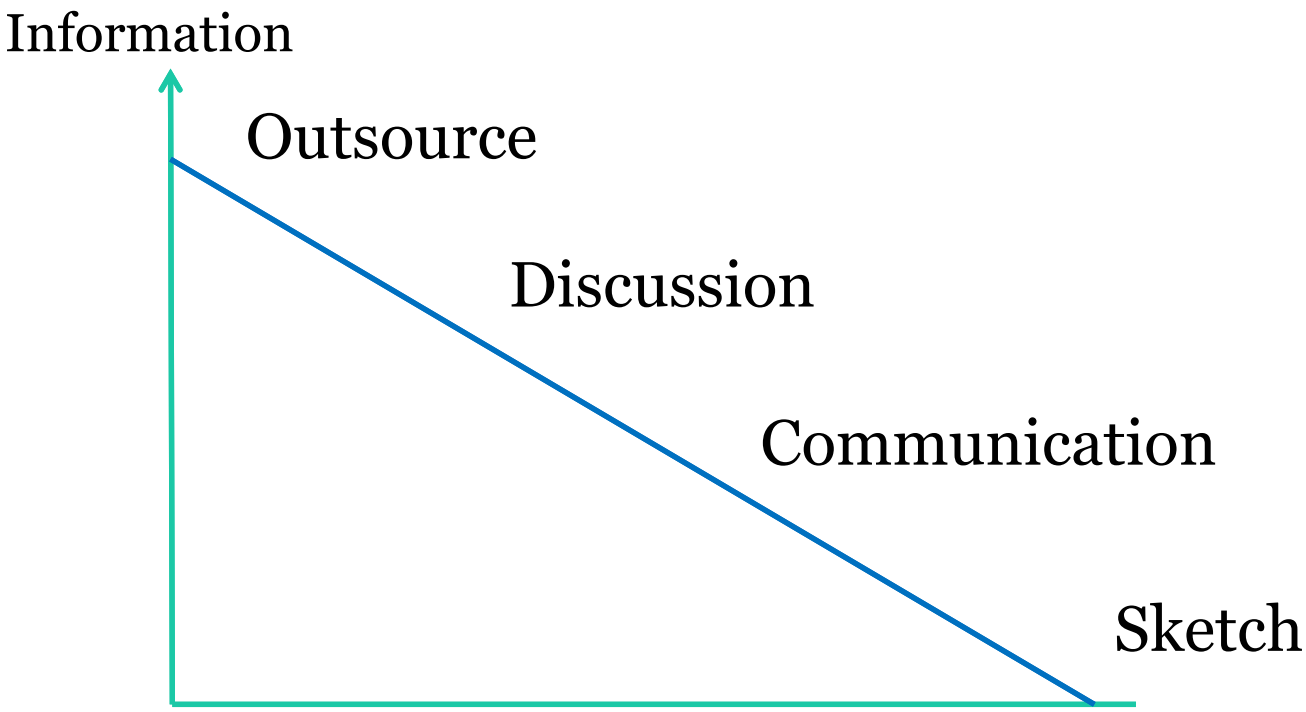
Type conformance



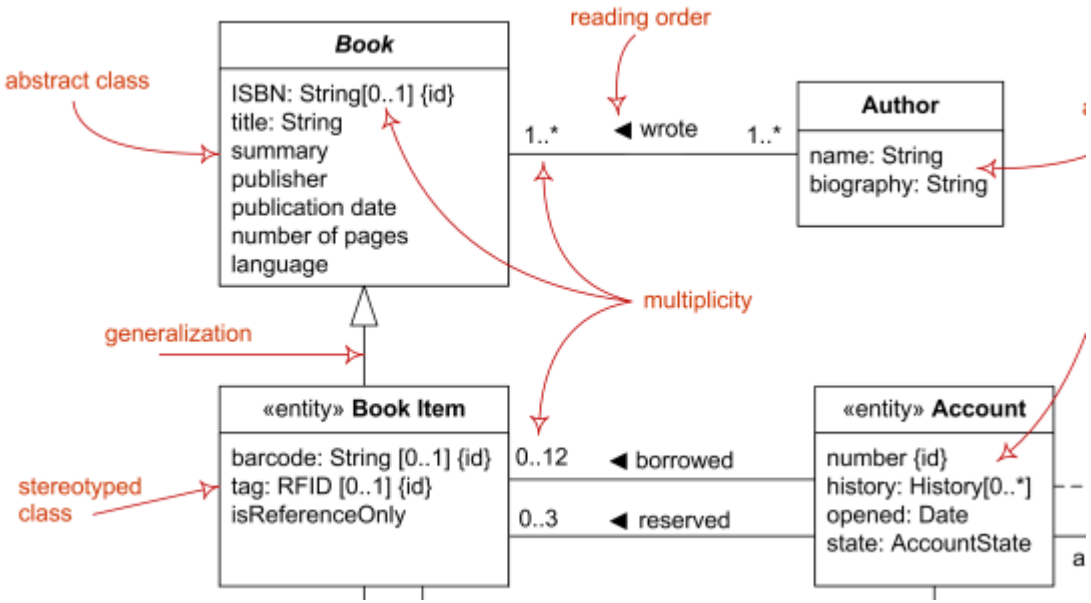
- A link in a model is **type conformant** if
  - ***type(src(link))*** is subtype of ***src(type(link))*** **AND**
  - ***type(trg(link))*** is subtype of ***trg(type(link))***
  - Informally:
    - The type of the source object is a subtype of the source class of the link's type.
    - The type of the target object is a subtype of the target class of the link's type.

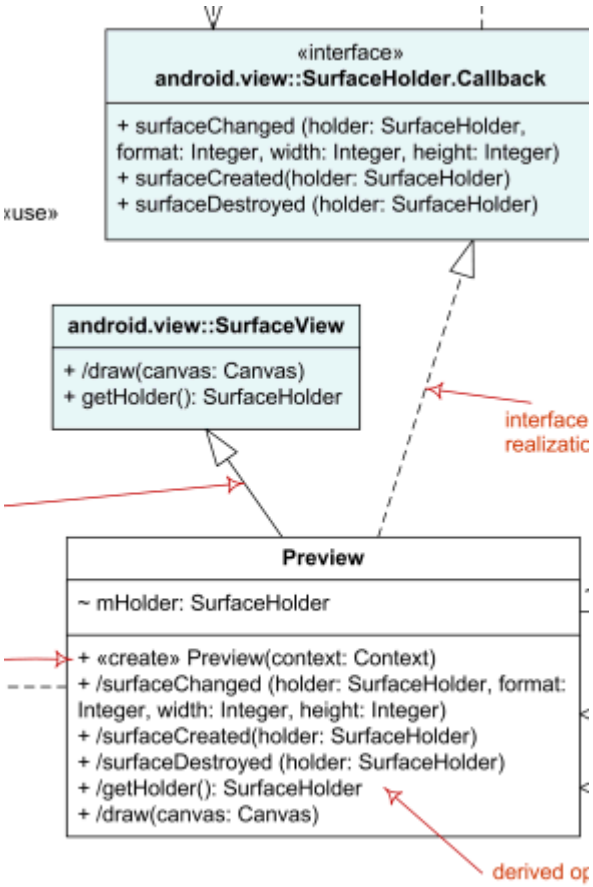
Domain Models
vs
Implementation Models

What you model depends on the recipient and the perspective

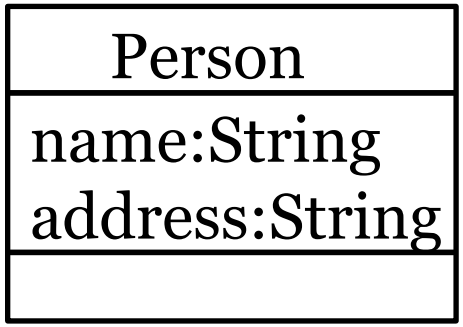


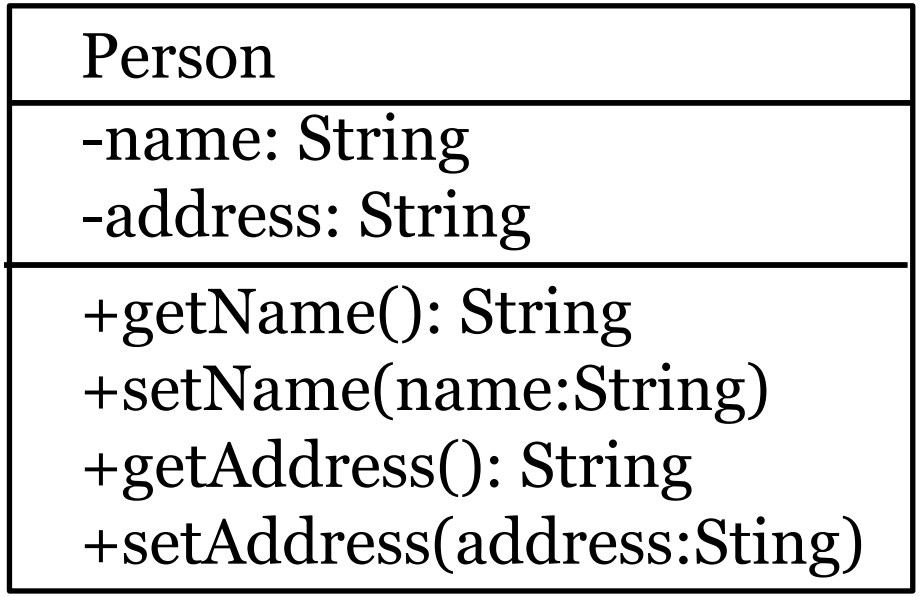
Perspectives: Domain modeling vs. Implementation





Domain model vs. Implementation model





In this course: domain model = conceptual model

Identifying classes: noun analysis

A graduate student application management system (GRADS) help to collect and review applications from prospective graduate students from all around the world to graduate programs offered on different levels (MEng vs. PhD), in different curricula (e.g. Software Engineering) and at a given starting time (E.g. Fall 2017). Prospective graduate students (MEng or PhD students) create a personal profile (with their name and citizenship) and then upload their application, which must contain their language exam score as well as their (undergraduate and graduate) degrees together with their CGPA score. When submitting their application, students specify their preferred supervisors. With the help of GRADS, the Graduate Program Administrator checks if all minimum criteria of graduate admission are fulfilled, which is 3.0 CGPA score and 100 language exam score, and may contact prospective students to complete missing information. Professors then review applications of those students in GRADS who meet the minimum criteria and who selected them as a preferred supervisor by assigning a numeric score. A student may be admitted to the graduate program if he or she is scored over 4 by at least one professor, otherwise his/her application is rejected.

Identifying classes: noun analysis

A graduate student application management system (GRADS) help to collect and review applications from prospective graduate students from all around the world to graduate programs offered on different levels (MEng vs. PhD), in different curricula (e.g. Software Engineering) and at a given starting time (E.g. Fall 2017). Prospective graduate students (MEng or PhD students) create a personal profile (with their name and citizenship) and then upload their application, which must contain their language exam score as well as their (undergraduate and graduate) degrees together with their CGPA score. When submitting their application, students specify their preferred supervisors. With the help of GRADS, the Graduate Program Administrator checks if all minimum criteria of graduate admission are fulfilled, which is 3.0 CGPA score and 100 language exam score, and may contact prospective students to complete missing information. Professors then review applications of those students in GRADS who meet the minimum criteria and who selected them as a preferred supervisor by assigning a numeric score. A student may be admitted to the graduate program if he or she is scored over 4 by at least one professor, otherwise his/her application is rejected.

Key Decisions during Noun Analysis

- Class vs. Actor vs. Actor + Class?
- Class vs. Attribute?
- Class vs. Enumeration?
- Class vs. AssociationEnd?
- Omit from domain model?

- GRADS
- Application
- Student
- StudentProfile
- GraduateProgram
- Curriculum
- StartingTime
- LangExamScore
- Degree
- CGPAScore
- Administrator
- MinimumCriteria
- Professor
- Score

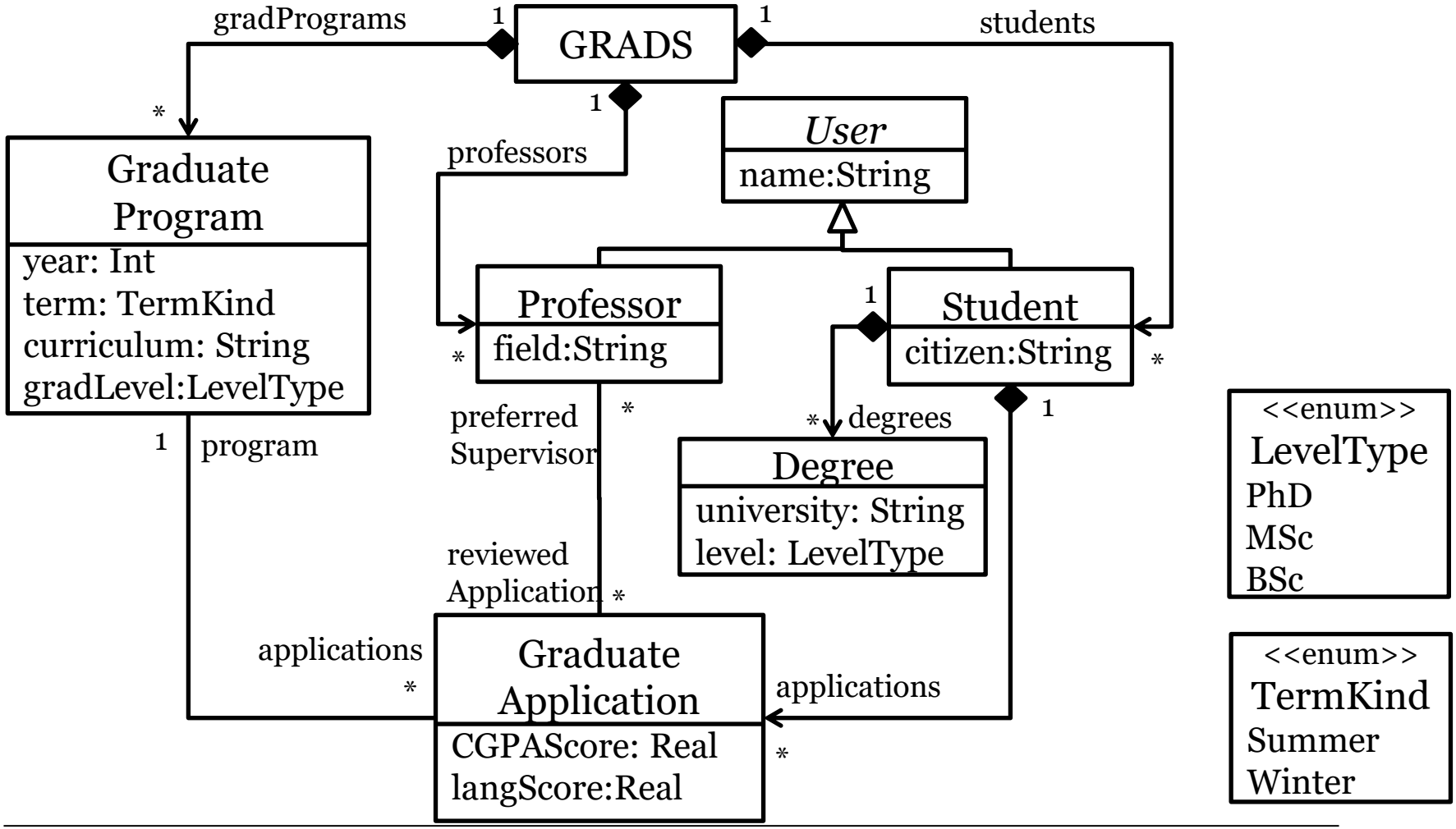
Make your decision!

Key Decisions during Noun Analysis

- Class vs. Actor vs. Actor + Class?
- Class vs. Attribute?
- Class vs. Enumeration?
- Class vs. AssociationEnd?
- Omit from domain model?

- GRADS → Class
- Application → Class
- Student → Actor? Class?
- StudentProfile → Class? Omit?
- GraduateProgram → Class
- Curriculum → Attribute
- StartingTime → Attribute
- LangExamScore → Attribute
- Degree → Class
- CGPAScore → Attribute
- Administrator → Actor
- MinimumCriteria → Omit? Attr?
- Professor → Class+Actor
- Score → Attribute

A Sample Solution (Not the Only One!)

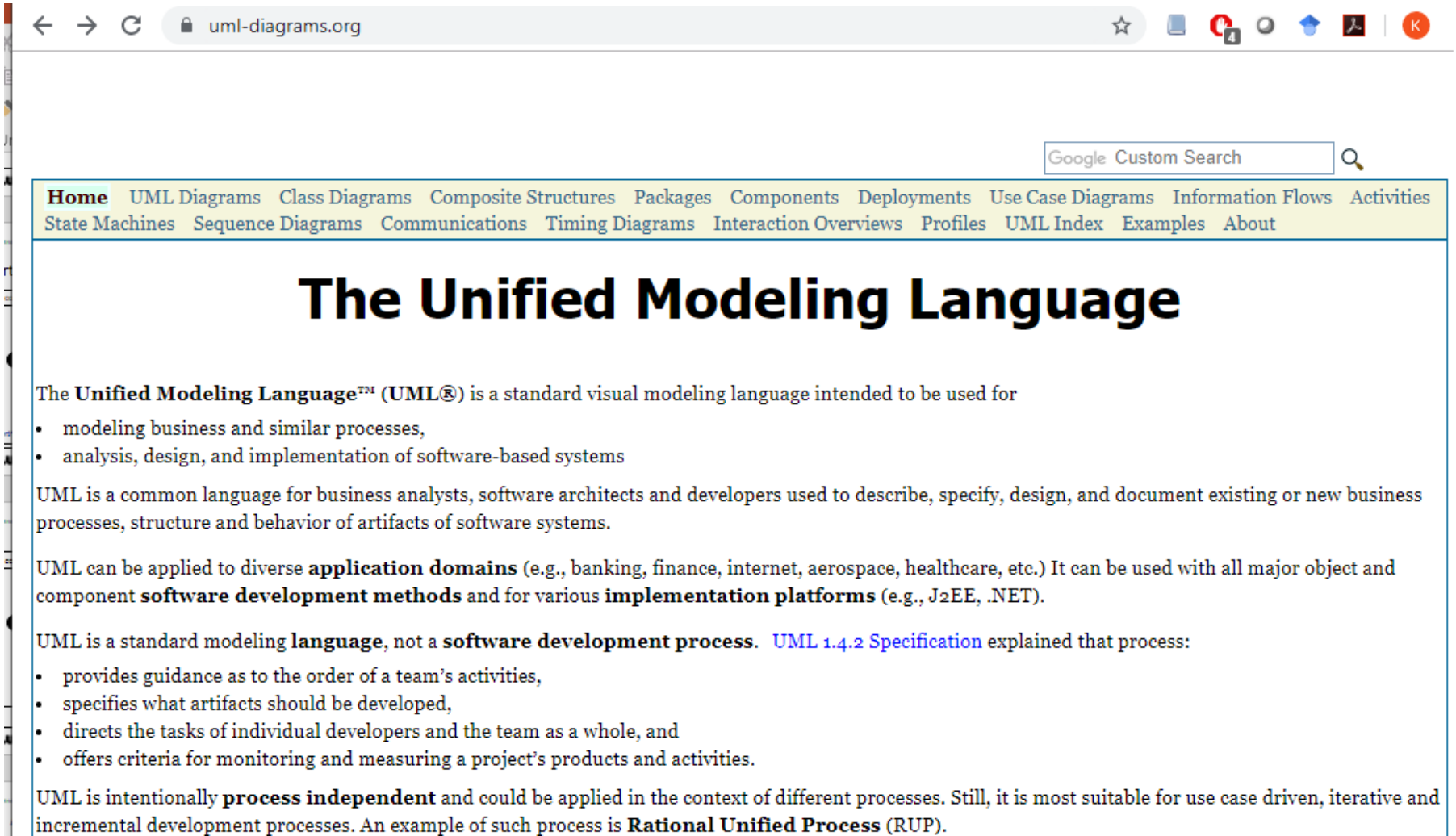


Your Checklist for Domain Modeling

- Use **only domain classes**!
  - No classes like List, HashMap, Collection
- **Attributes**: type should be built-in (primitive)
  - If the type is other domain class → association!
- **Generalization**: class vs. kind/type attribute?
- **Names** for all associations
- Specify **multiplicities** for all association ends (roles)
- **Containment hierarchy**:
  Who is the container of an instance of a class?

- No circular containment in the instance model
- Aggregation may be circular on the class level!

A complete, comprehensive guide to UML 2.5



UML Behavior Modeling
(State Machine Diagrams)

For defining reactive behavior of objects
by executing state transitions and actions
in response to events

State-based Behavior Modeling

- **State partition** (AKA **state space**)
  - A set of distinguished **system states**
  - Examples
    - Days of Week: {Mon, Tue, Wed, Thu, Fri, Sat, Sun}
    - States of microwave oven: {full power, defrost, off}
  - **DEF**: A state partition is a set, exactly one element of which characterises the system at any time.
- **Current state**
  - E.g. today is Wed, the microwave is on defrost, etc.
  - **DEF**: At any given moment, the current state is the element of the partition which is currently valid.

Example: Abstract & Concrete States of a Stack

- **Concrete states** of a stack
  - $Stack_1$
    - Length = 2
    - Element[0] = String(„Winter 2023”)
    - Element[1] = String(„Fall 2023”)
  - $Stack_2$
    - Length = 2
    - Element[0] = String(„Winter 2024”)
    - Element[1] = String(„Fall 2024”)
- **Abstract states** of a stack:
  - **empty** : boolean isEmpty() {return length==0;}
  - **full** : boolean isFull() {return length==MAX;}
  - **hasContent**: boolean hasContent()
    {return length > 0 && length < MAX;}

Are these stacks in a different concrete state? **YES!**

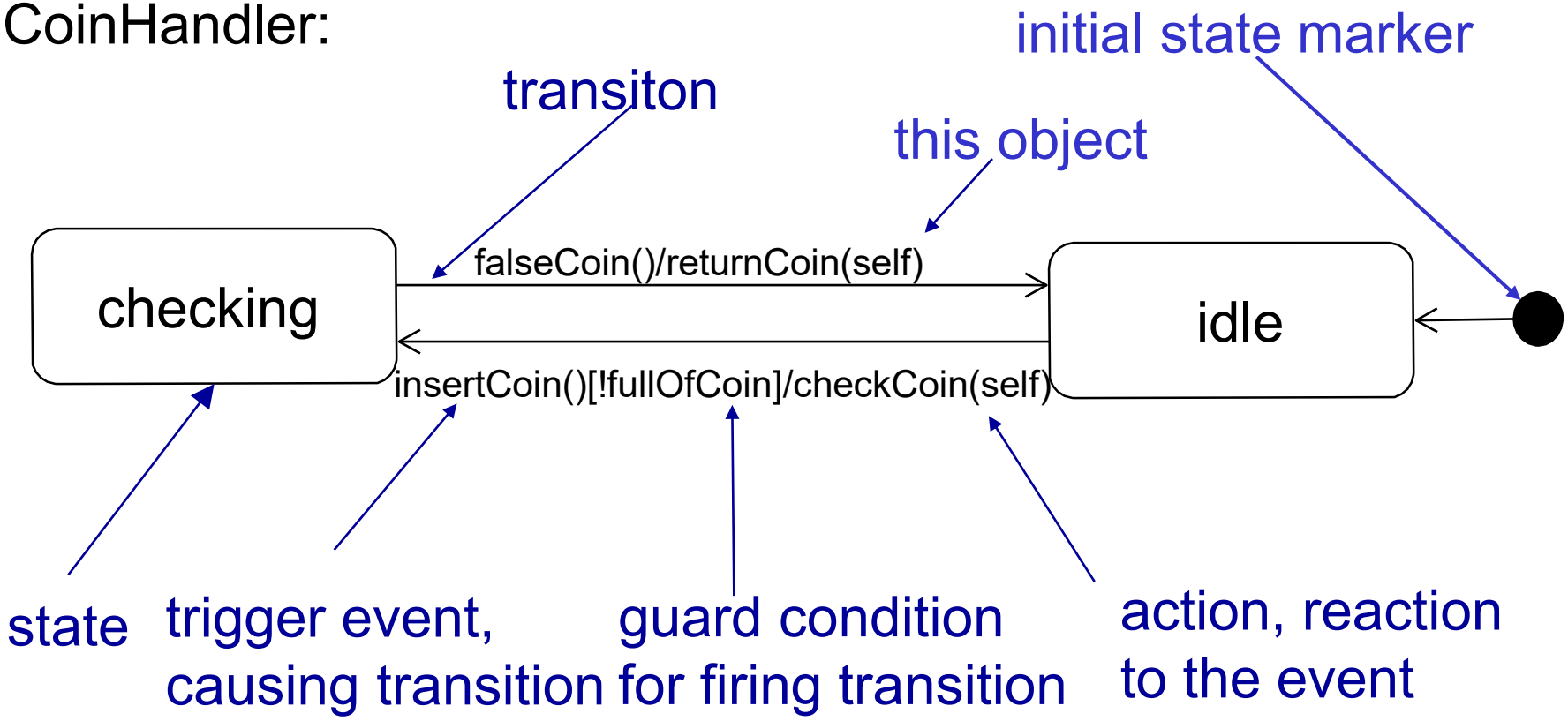
Are these stacks in a different abstract state? **NO!**

Abstract State vs. Concrete State

- **Concrete state** of an object:
  - Current value of each of its attributes
  - Concrete state space:
    - Combination of possible values of attributes
    - May be infinite
- **Abstract states** of an object:
  - Predicates over concrete states
  - One abstract state ← many concrete states
  - Potentally: state hierarchies

UML State machine diagram

For class CoinHandler:



Specification

- Kristian's alarm clock starts sounding at 6.00 with a nasty signal. He can now do either of three things:

  a) Turn the alarm off;

  b) Press the snooze button; or

  c) Do nothing.
- If the snooze button is pressed the signal will turn off and start sounding after 5 minutes again.
- After that the alarm sound starts, the signal will last for 2 minutes.
- If no action has been taken during these 2 minutes, the absence of action will have the same effect as if the snooze button were pressed exactly when the alarm stopped to sound

- Task: design a UML state machine of the class *AlarmClock*

Step 1: Define all Events and Actions

Attributes (Key object attributes used in predicates)

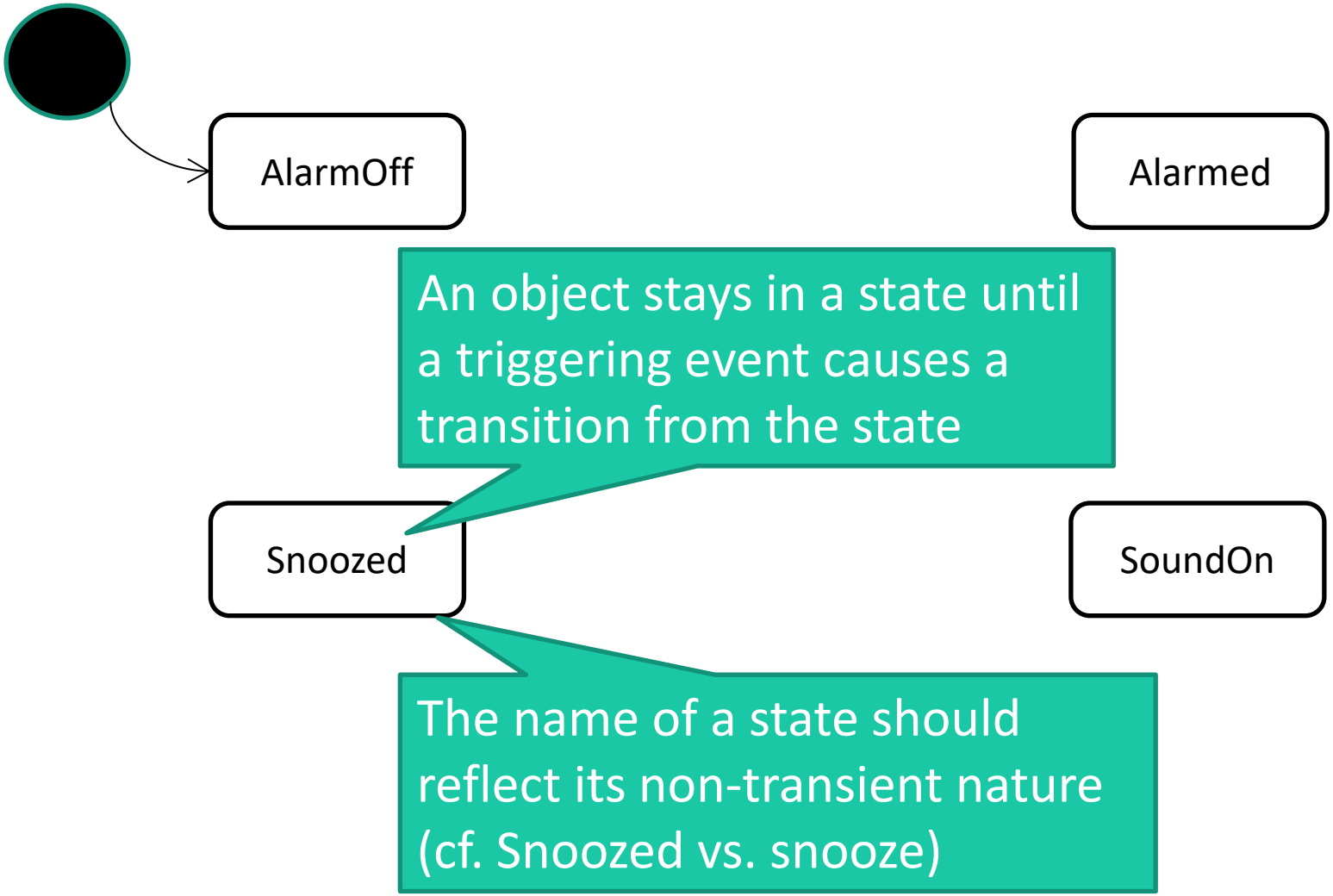
- *t:* current time
- *ta*: time set for alarm
- *ts*: time of snooze button pressed

Events (What can Kristian do with the AlarmClock?)

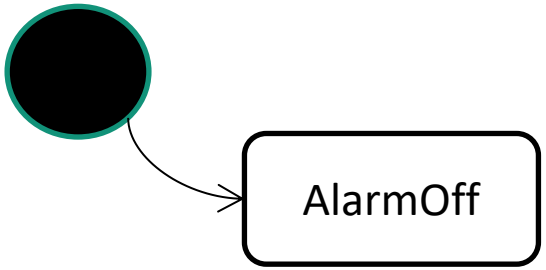
- **setAlarm(tw)**: set alarm time *ta* to the wake-up time and turn on the alarm
- **snooze**: press the snooze button
- **turnOff**: turn off alarm
- **timeout(t)**: timeout event

Actions (What can the AlarmClock do?)

- **startSound**: turn on alarm sound
- **endSound**: turn off alarm sound



Step 2: Define all States

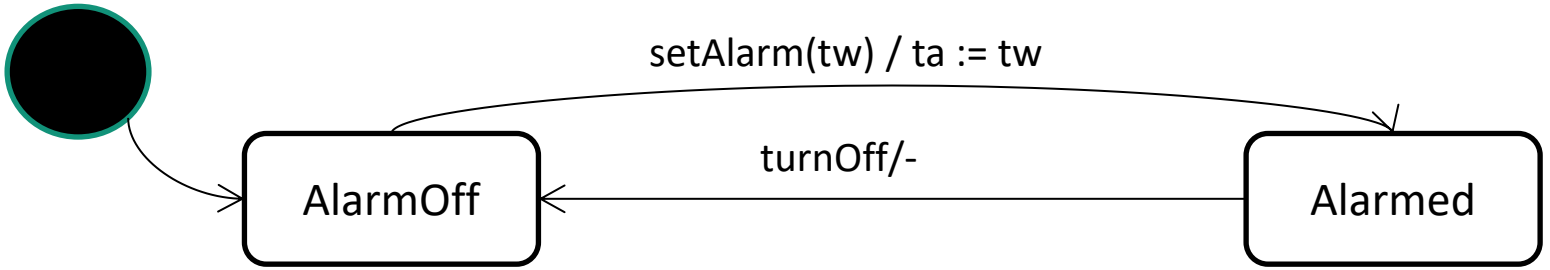


Attributes:
t
ta
ts

Events:
setAlarm(tw)
snooze
turnOff

Actions:
startSound
endSound

Step 3: Define all Transitions



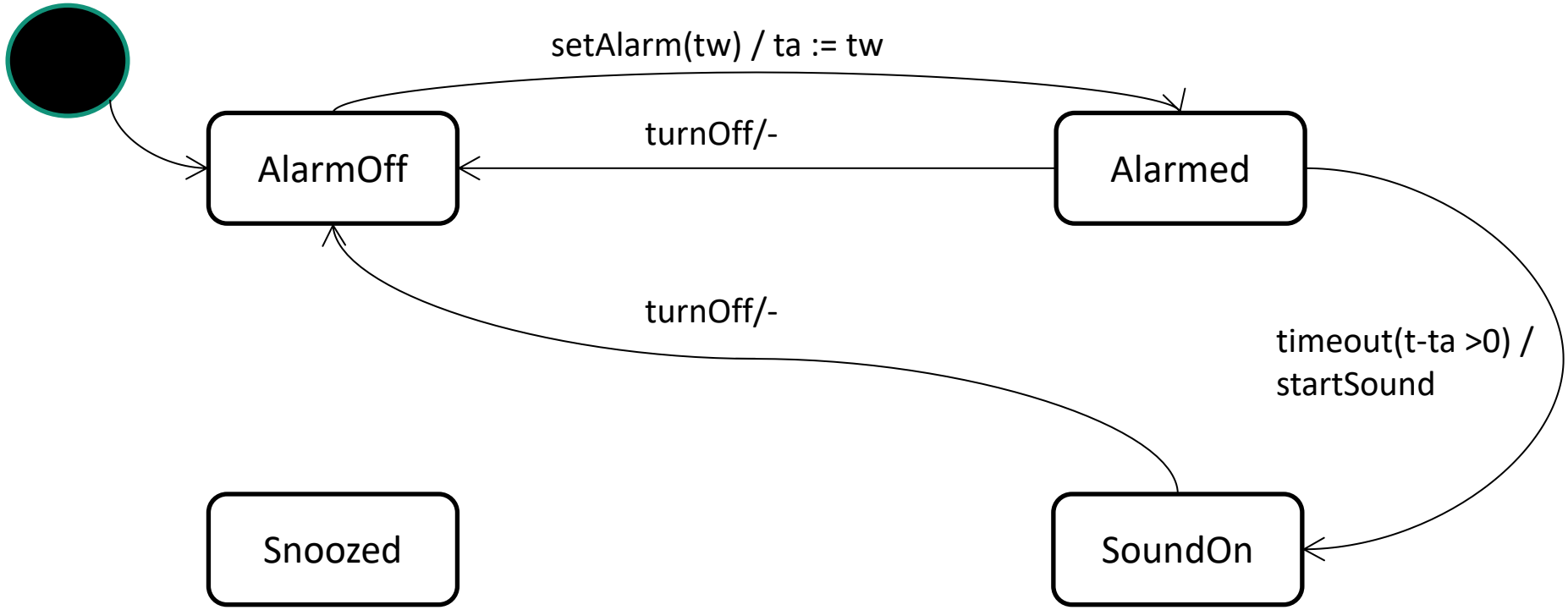
Snoozed

SoundOn

Attributes:
t
ta
ts

Events:
setAlarm(tw)
snooze
turnOff

Actions:
startSound
endSound



Attributes:
t
ta
ts

Events:
setAlarm(tw)
snooze
turnOff

Actions:
startSound
endSound

Specification

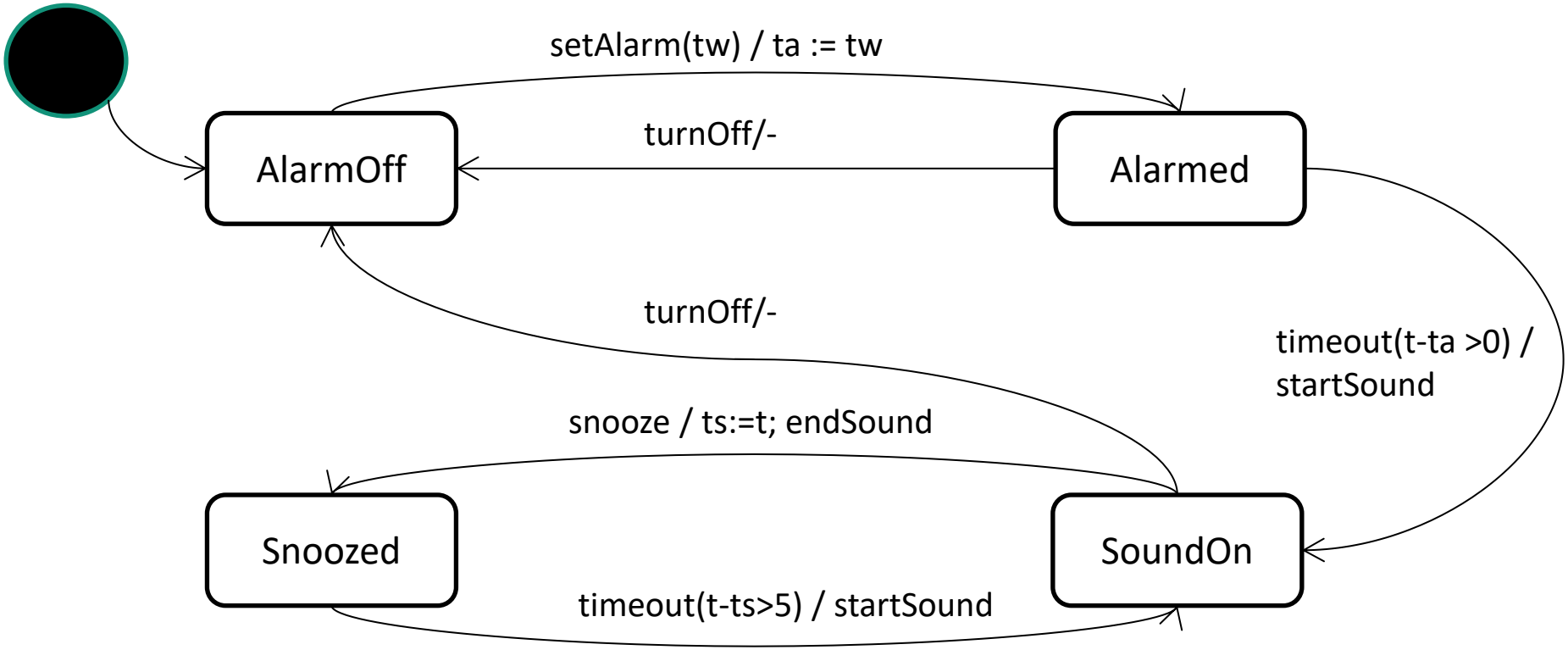
- Kristian's alarm clock starts sounding at 6.00 with a nasty signal. He can now do either of three things:

  a) Turn the alarm off;

  b) Press the snooze button; or

  c) Do nothing.

- If the snooze button is pressed the signal will turn off and start sounding after 5 minutes again.
- After that the alarm sound starts, the signal will last for 2 minutes.
- If no action has been taken during these 2 minutes, the absence of action will have the same effect as if the snooze button were pressed exactly when the alarm stopped to sound



Attributes:
t
ta
ts

Events:
setAlarm(tw)
snooze
turnOff

Actions:
startSound
endSound

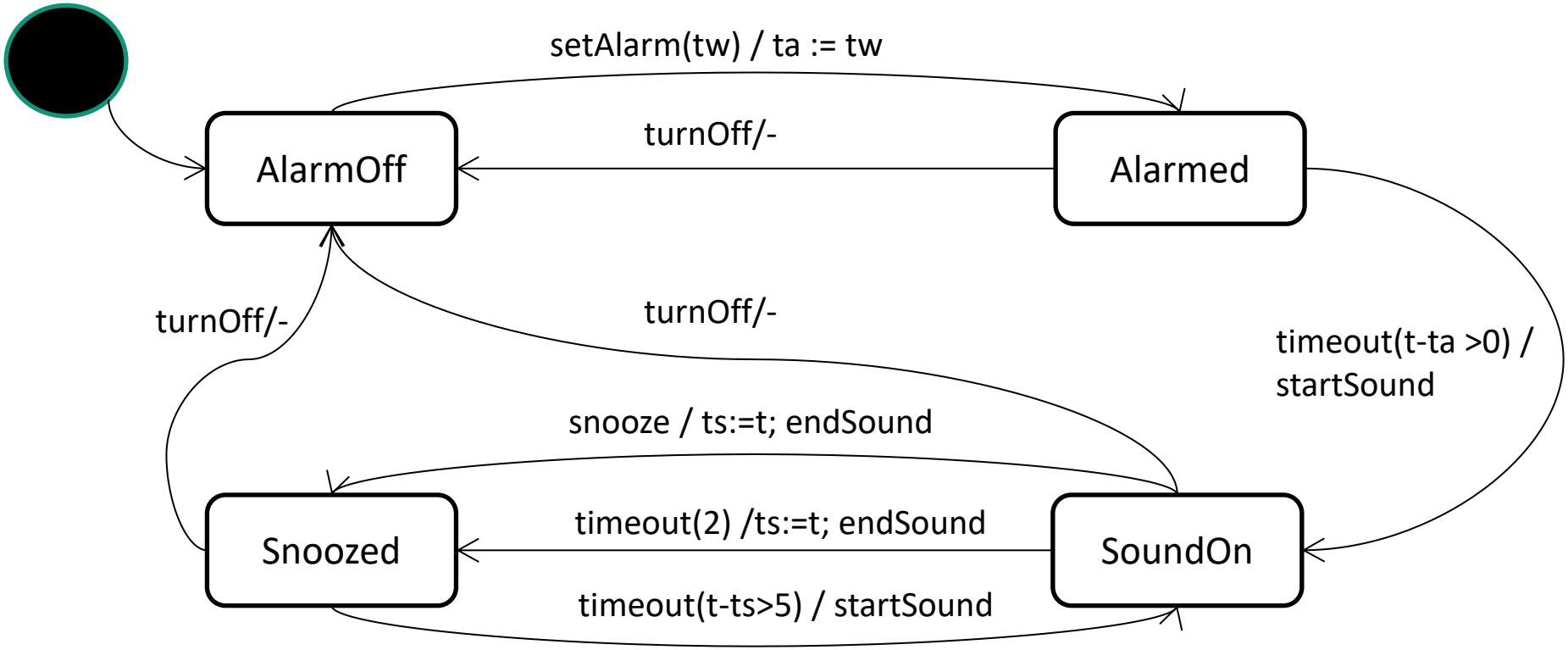
Specification

- Kristian's alarm clock starts sounding at 6.00 with a nasty signal. He can now do either of three things:

  a) Turn the alarm off;

  b) Press the snooze button; or

  c) Do nothing.
- If the snooze button is pressed the signal will turn off and start sounding after 5 minutes again.
- After that the alarm sound starts, the signal will last for 2 minutes.
- If no action has been taken during these 2 minutes, the absence of action will have the same effect as if the snooze button were pressed exactly when the alarm stopped to sound

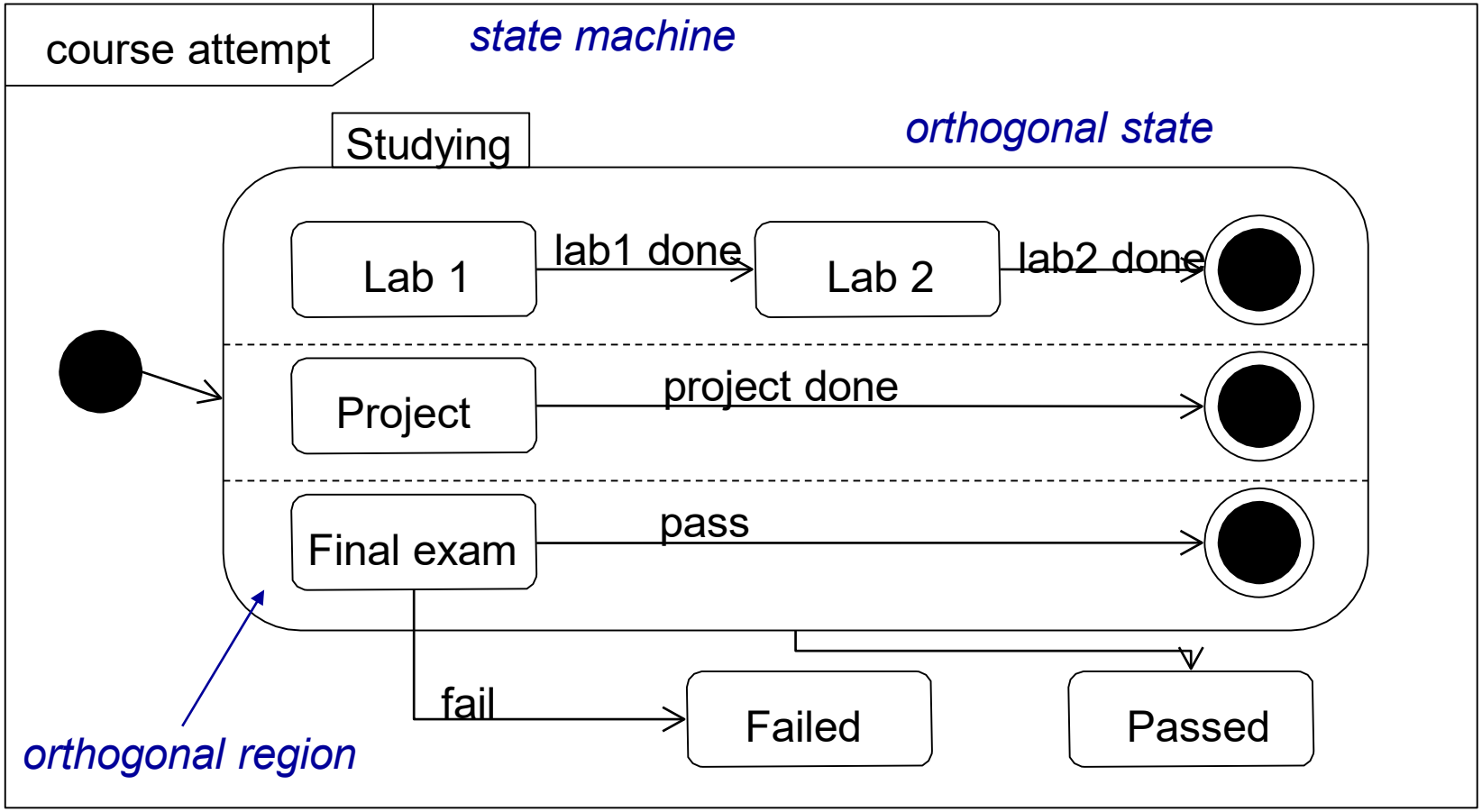


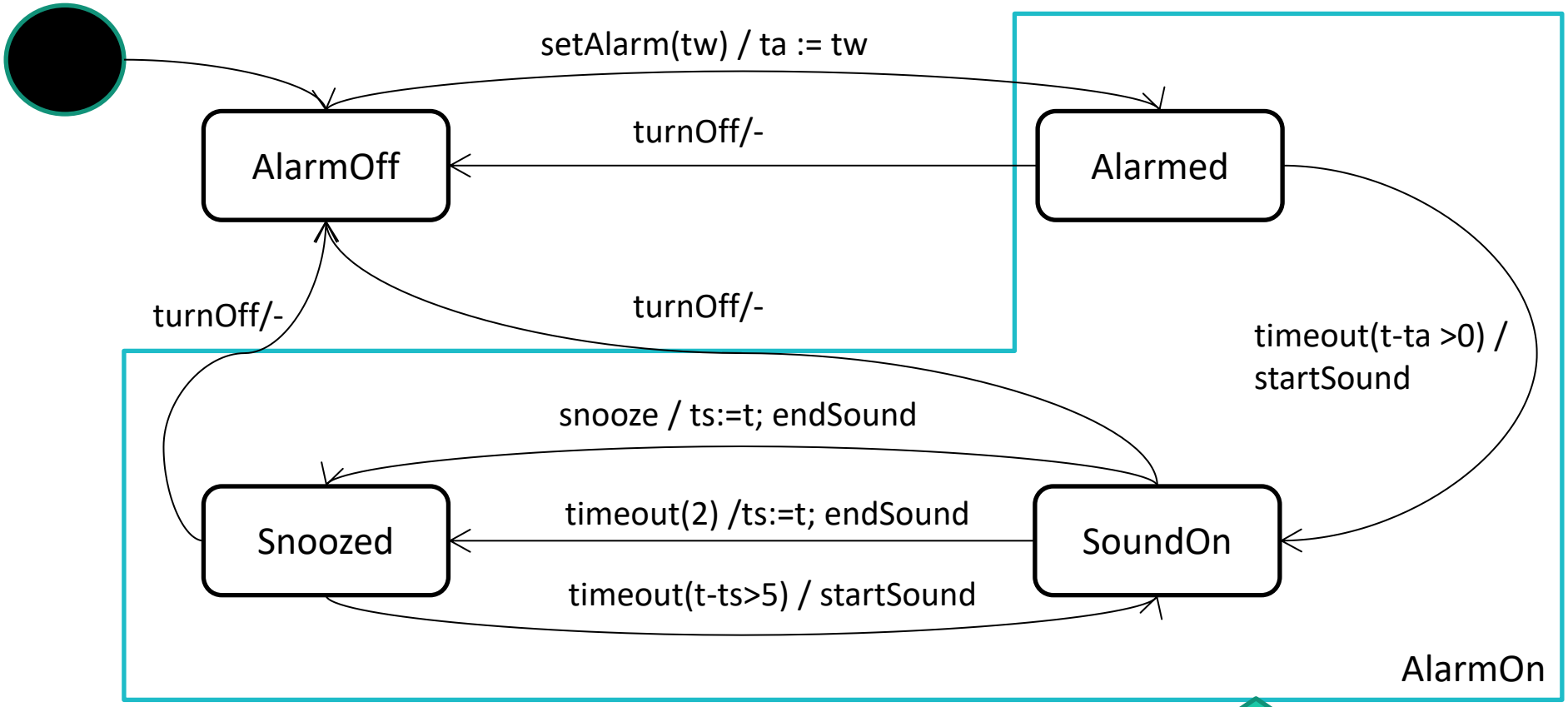
Attributes:
t
ta
ts

Events:
setAlarm(tw)
snooze
turnOff

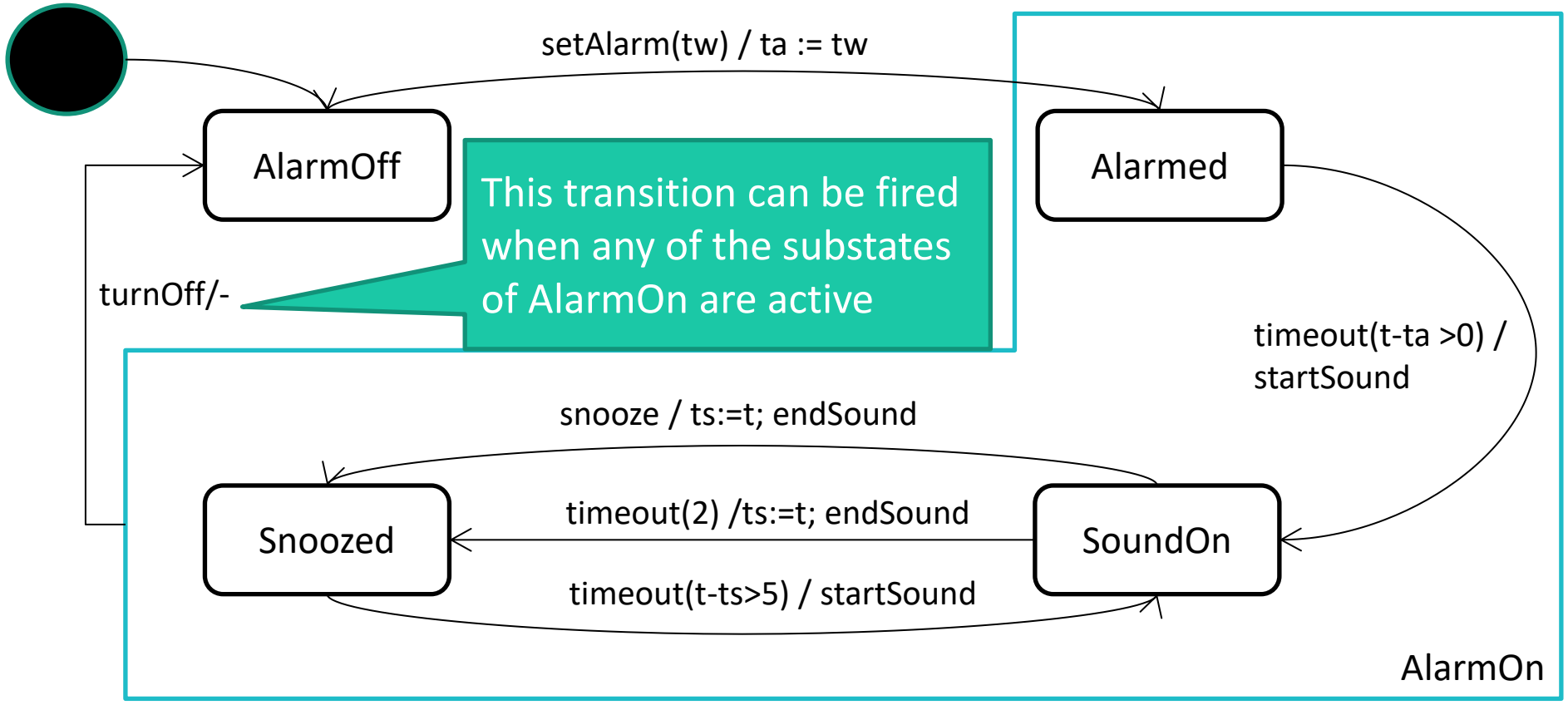
Actions:
startSound
endSound

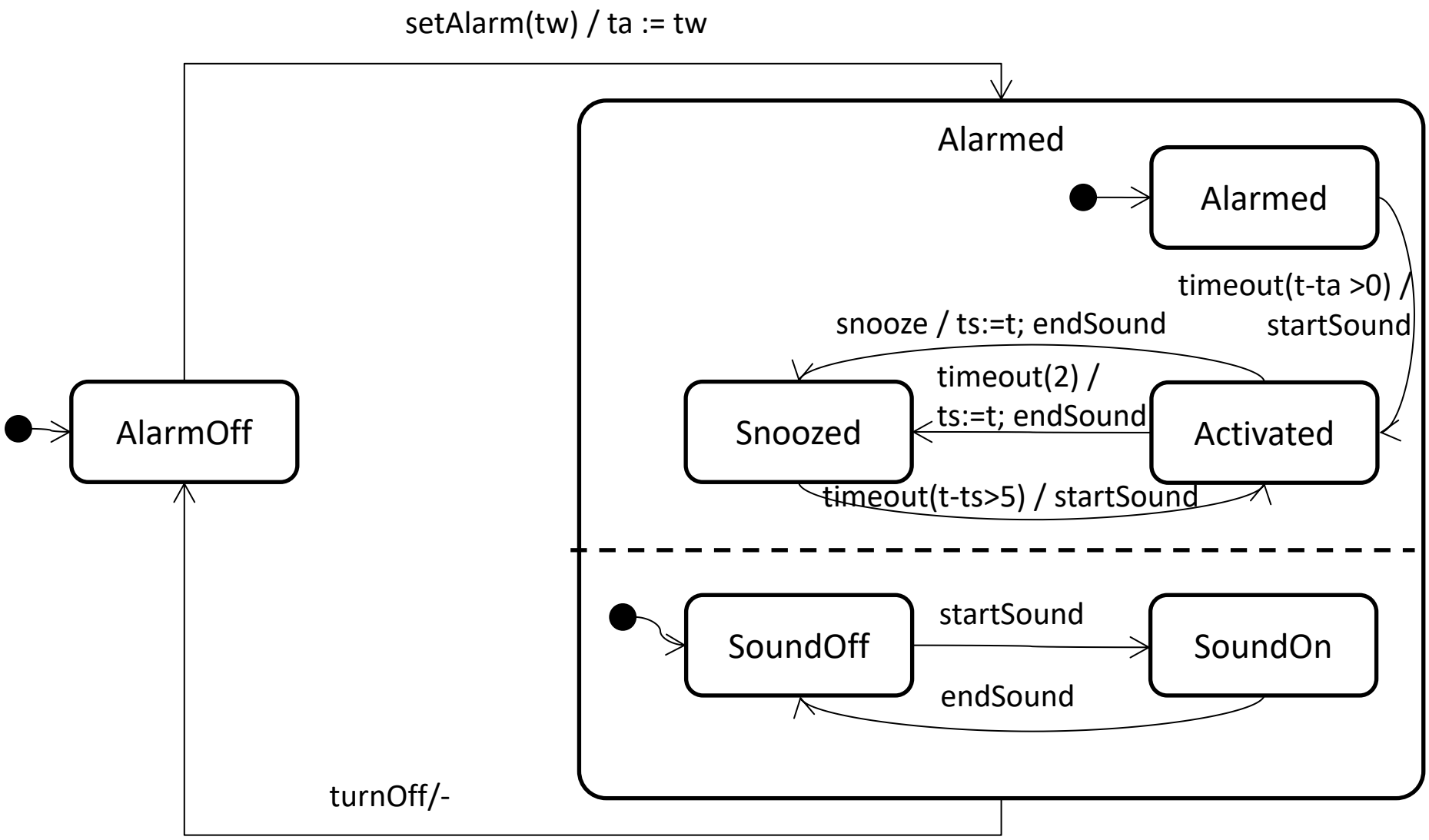
Orthogonal, composite states





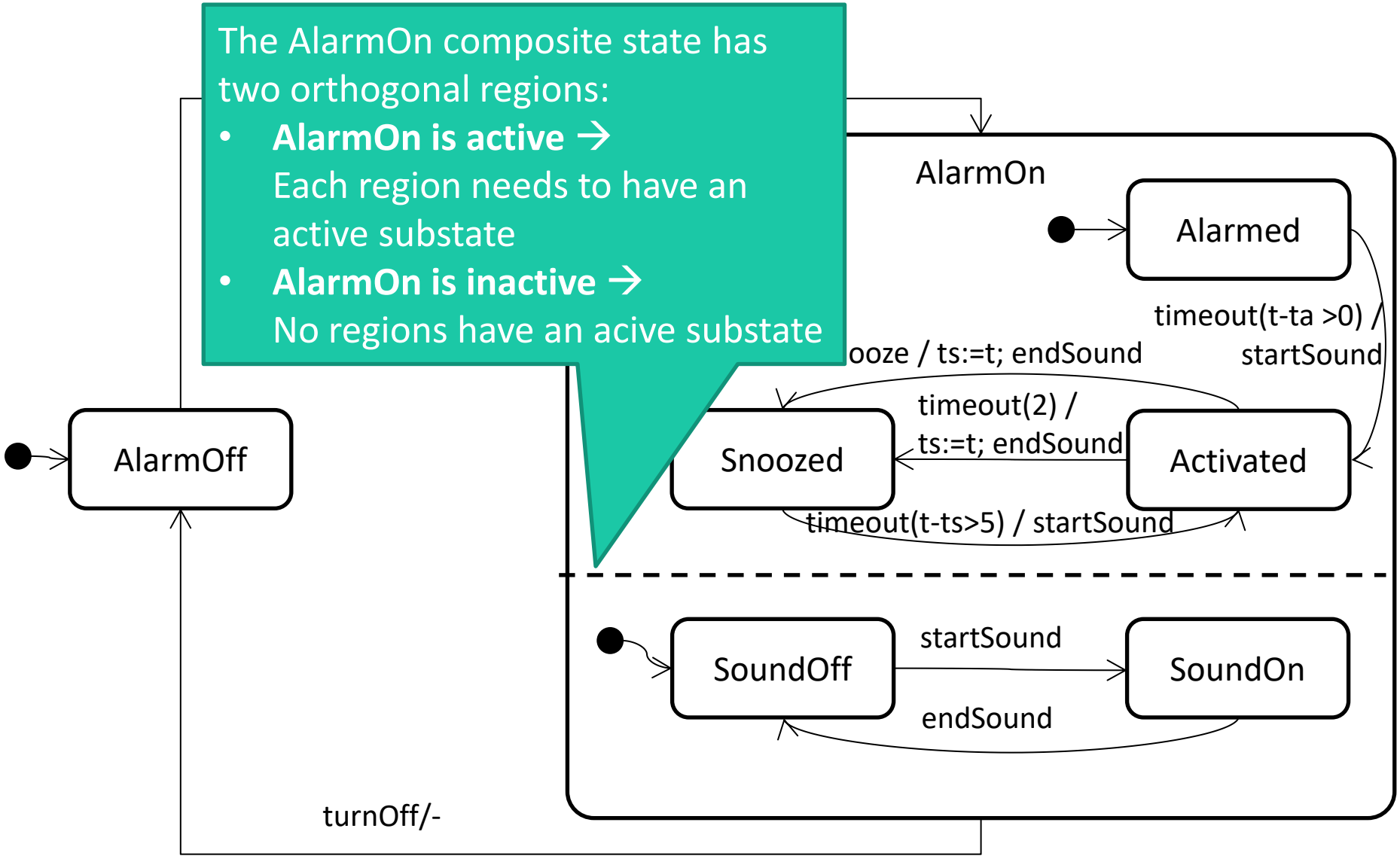
Composite state:
When it is active, exactly one substate needs to be active (in each region)


setAlarm(tw) / ta := tw
AlarmOff
Alarmed
This transition can be fired when any of the substates of AlarmOn are active
turnOff/-
timeout(t-ta >0) / startSound
snooze / ts:=t; endSound
Snoozed
timeout(2) /ts:=t; endSound
SoundOn
timeout(t-ts>5) / startSound
AlarmOn

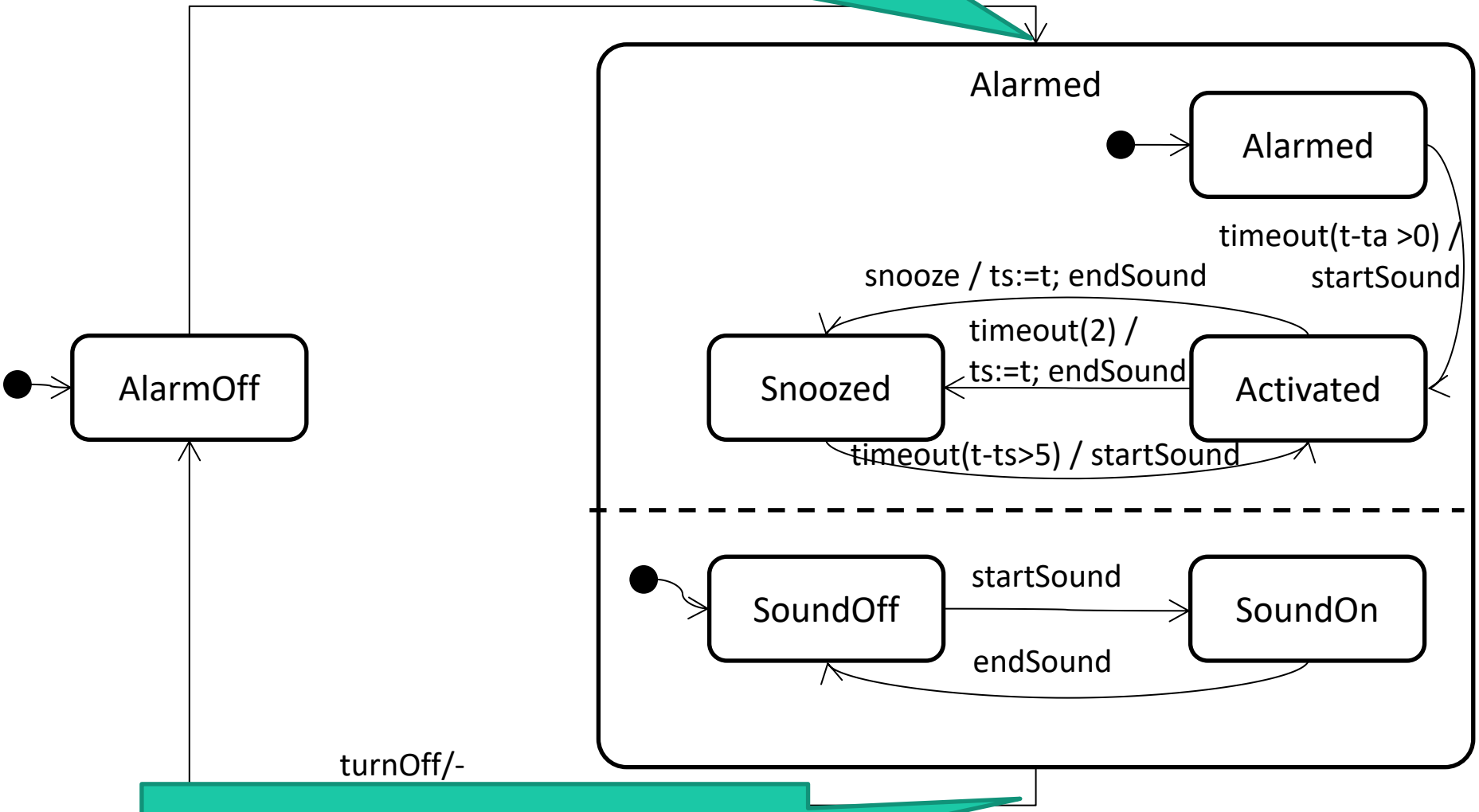

setAlarm(tw) / ta := tw
Alarmed
Alarmed
timeout(t-ta >0) /
startSound
snooze / ts:=t; endSound
timeout(2) /
ts:=t; endSound
AlarmOff
Snoozed
Activated
timeout(t-ts>5) / startSound
SoundOff
startSound
SoundOn
endSound
turnOff/-

The AlarmOn composite state has two orthogonal regions:

- **AlarmOn is active →**
  Each region needs to have an active substate
- **AlarmOn is inactive →**
  No regions have an acive substate

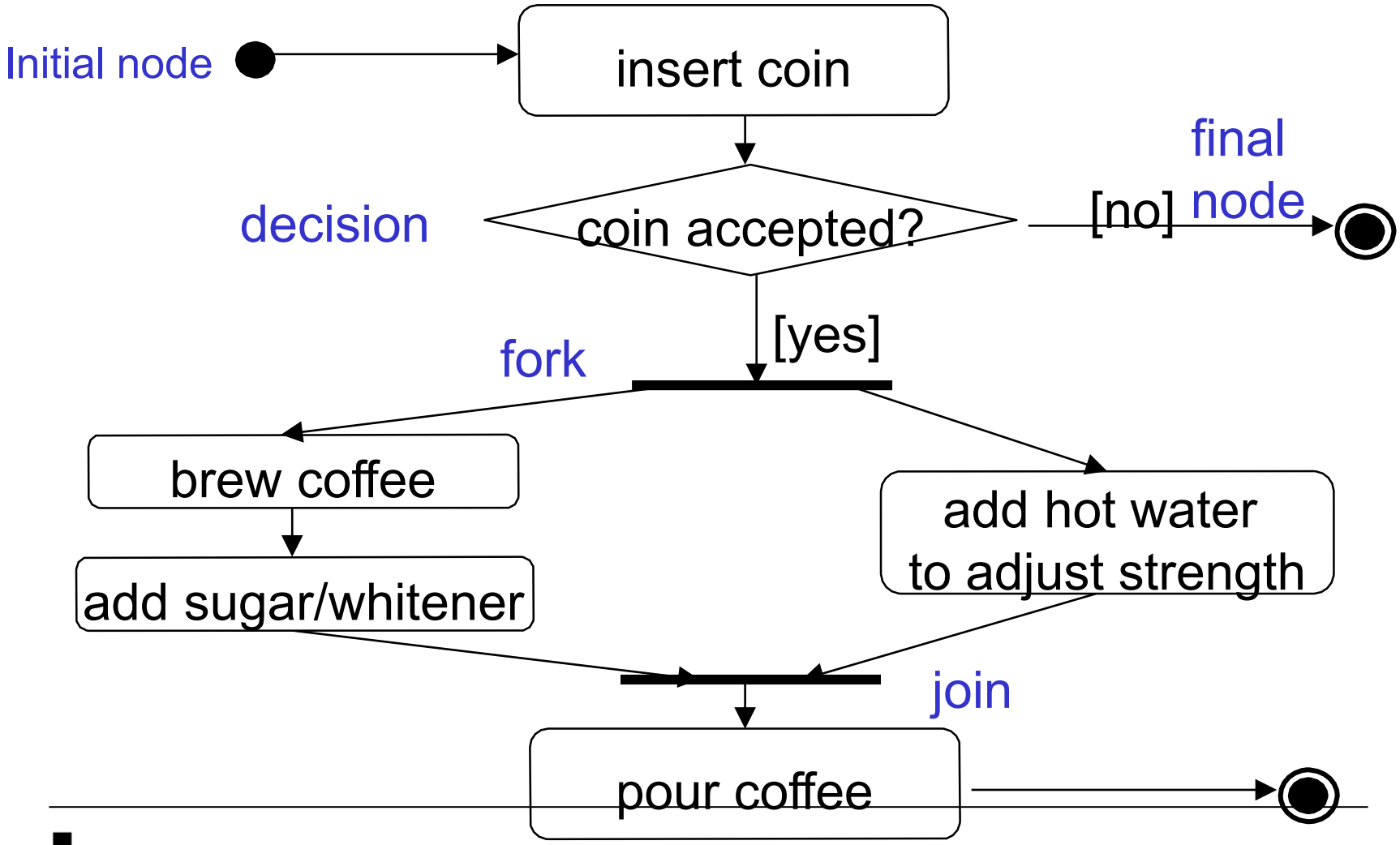


When a composite state is entered at its border → all regions enter their initial state



When a composite state is exited → all regions become inactive

Activity diagram ≠ State diagram

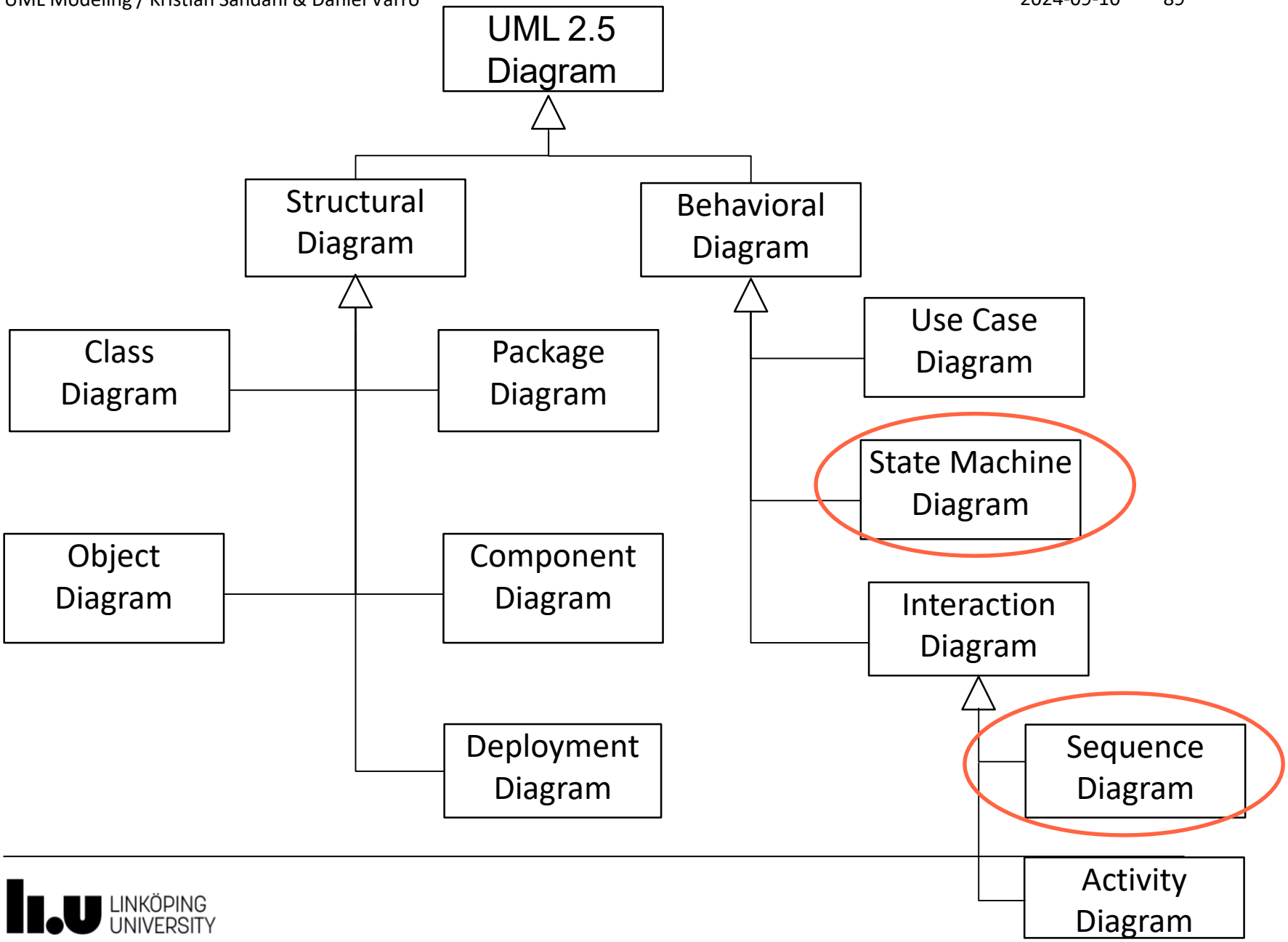


UML Behavior Modeling
(Sequence Diagrams)

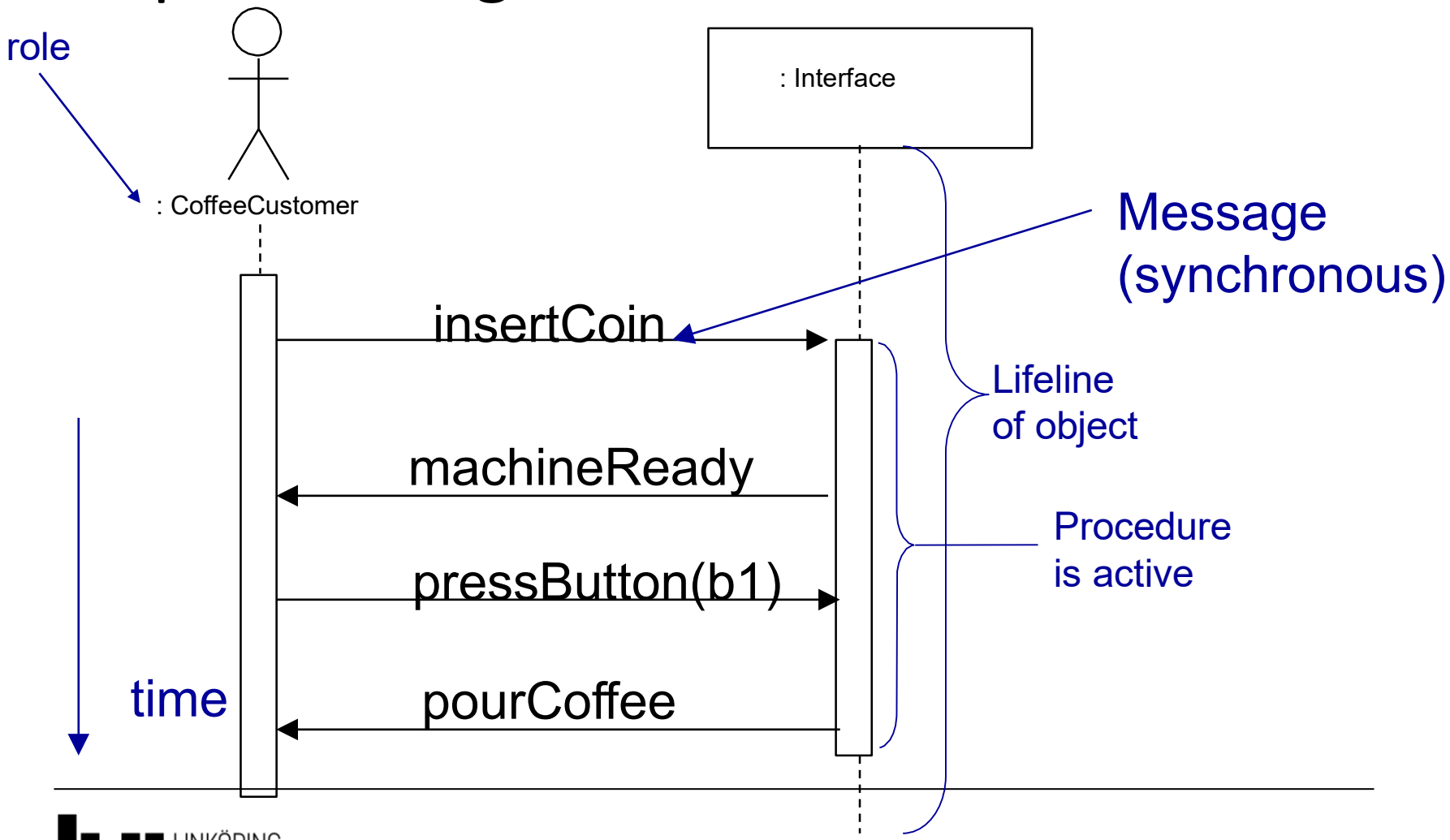
Provide a description of the dynamic behavior as interactions

- between actors and the system and
- between objects within the system

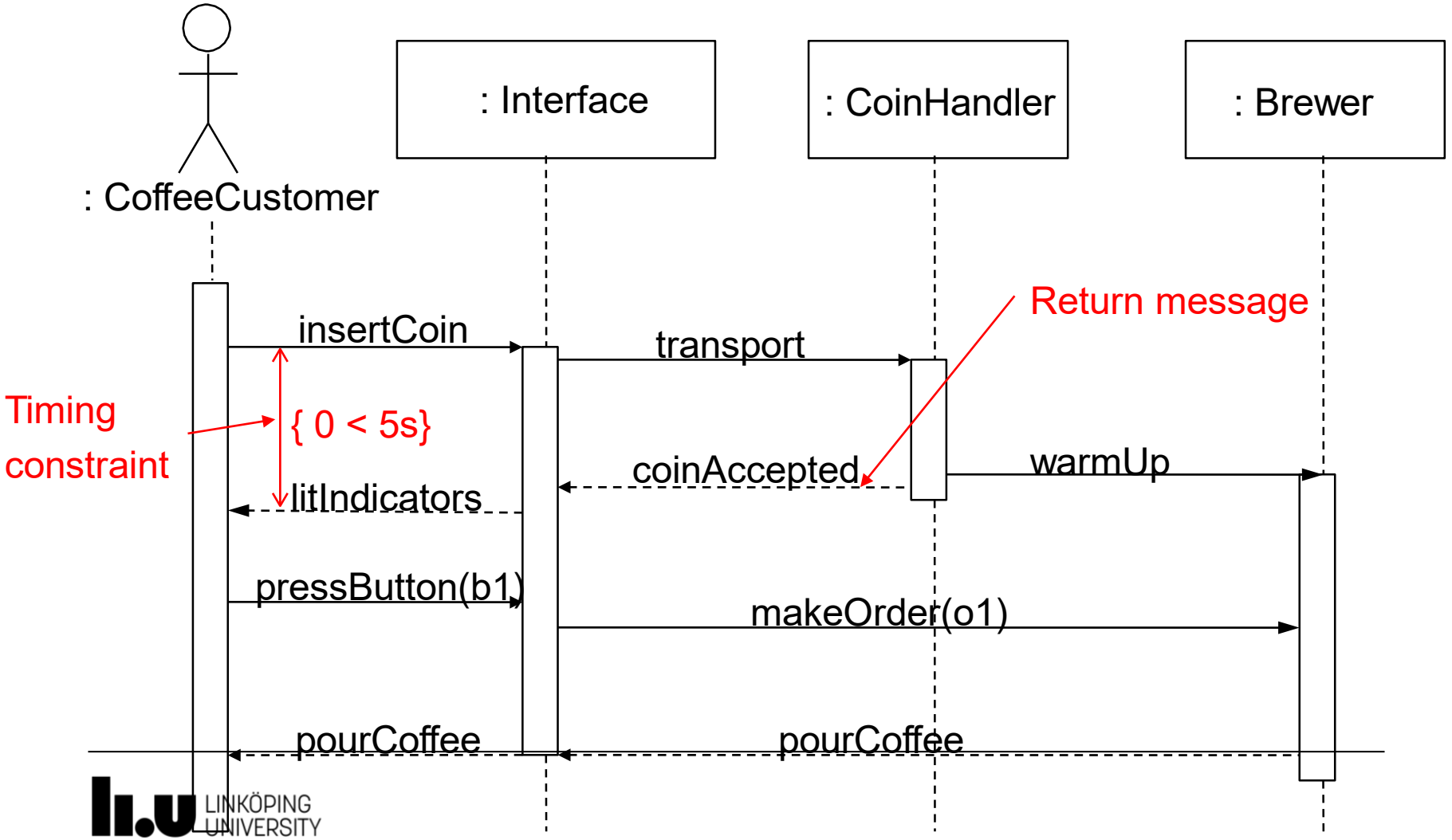
Well-known Diagrams of UML



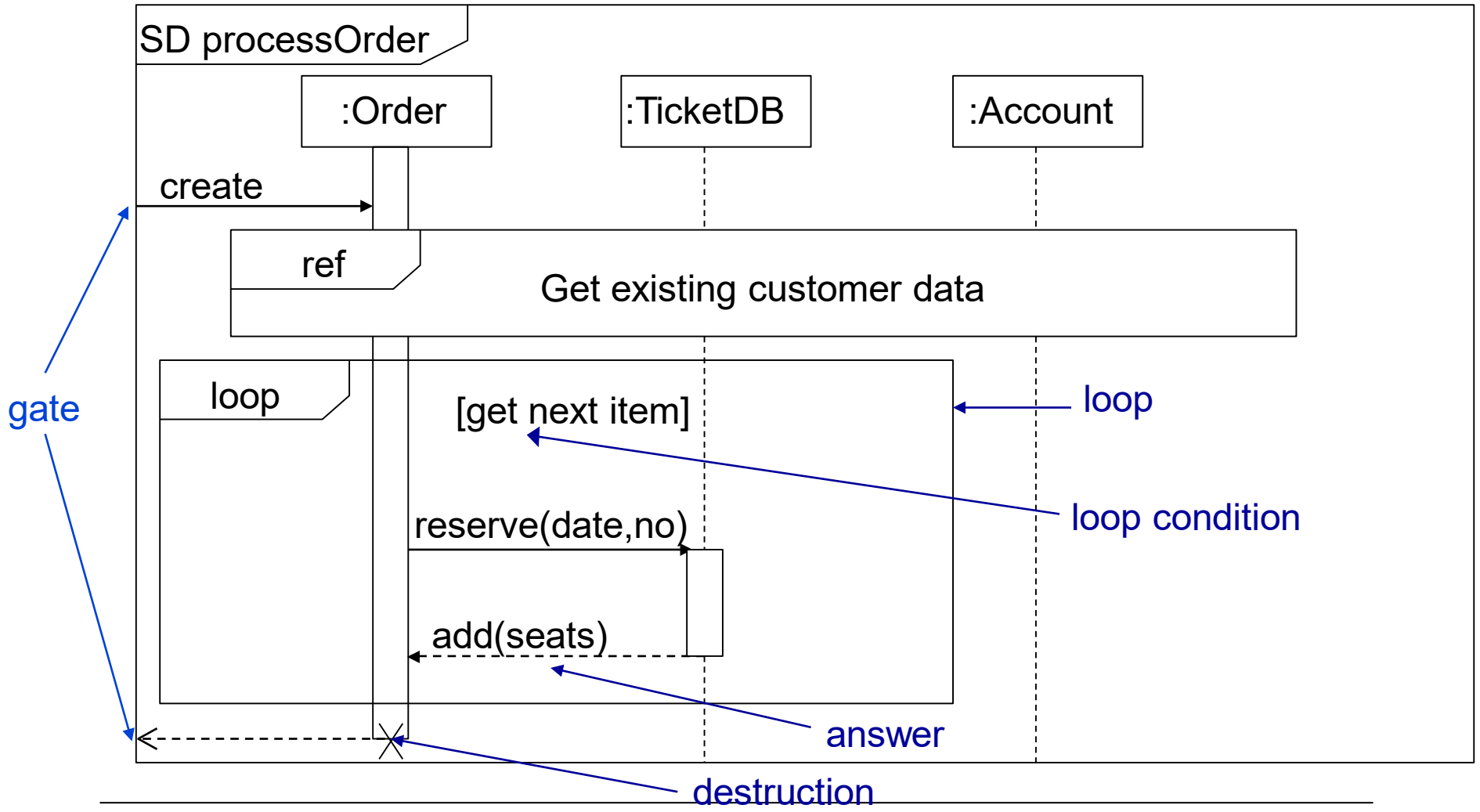
Sequence diagram



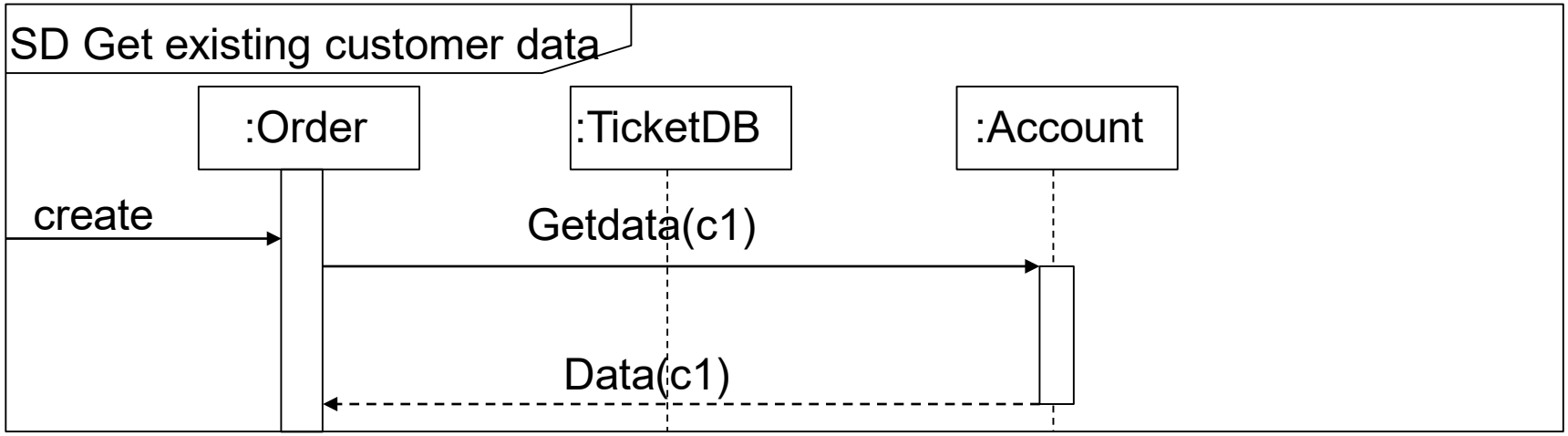
Sequence diagram with several roles



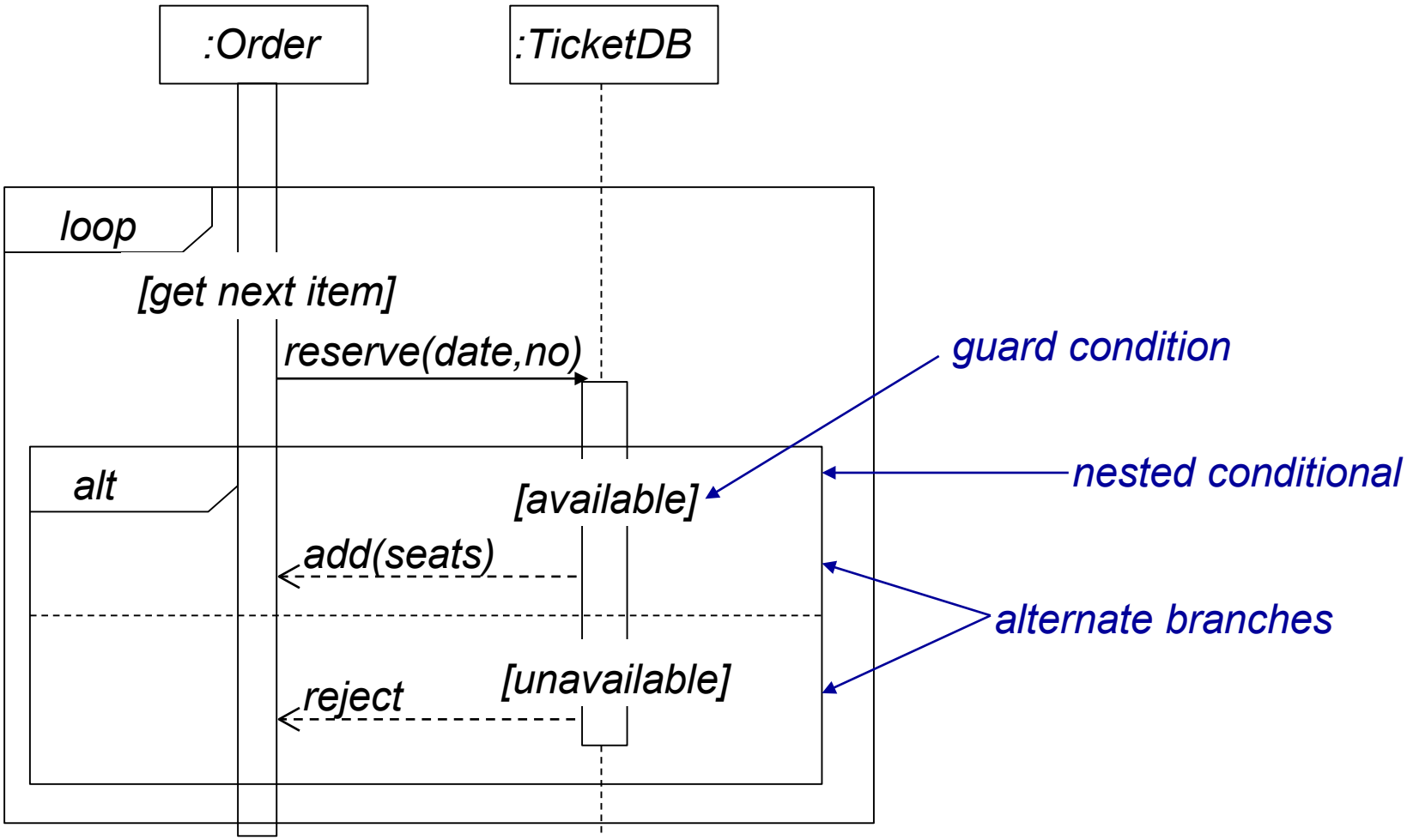
Combining fragments of sequence diagrams



Combining fragments of sequence diagrams

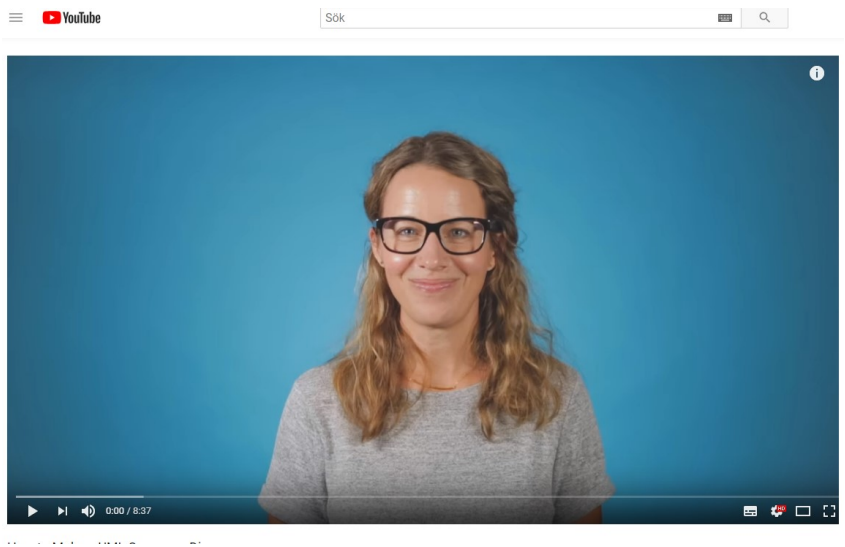


More fragments of sequence diagrams



Rehearsal and a little example

https://www.youtube.com/watch?v=pCK6prSq8aw&t=7s



Two flaws:
Objects preceded with ":"
Eject card after invalid card or invalid PIN shall terminate transaction.

Summary

- Structural diagrams
  - Class vs. Objects, Attributes, Relationships
- Behavioral diagrams
  - Sequence diagram
  - State machine diagram
- Domain analysis vs implementation

Preparation for Friday (Modeling Practice)

A review management system (REMS) help the review of scientific journal papers submitted by researchers. Authors submit a paper by using a form to specify a title, an abstract, a list of keywords and a first version as PDF document. They may also suggest names for excluded reviewers. When a new submission is received, REMS assigns a qualified editor to manage its review process by matching the keywords of the paper with editors' expertise. An editor sends invitation to several reviewers (not excluded by the authors) who either accept or decline this invitation. When two reviewers agree to review the paper, no further reviewers will be invited. A reviewer needs to complete a review which includes a textual critic and a recommendation: accept, minor revision, major revision or reject. Based upon the recommendations of the reviews, the editor makes a decision on the paper (which is also one of accept, minor revision, major revision and reject). If the decision is major revision, the authors need to resubmit a revised version of the paper, and the editor initiates a 2nd round of review, which is identical with the 1st round, except for excluding major revision as a possible outcome.

Write a **functional requirement** to capture that *only qualified editors will handle any paper*.
Write an **non-functional requirement** on *the availability* of the REMS system.

Draw a ***use case diagram*** for the REMS system highlighting key actors, use cases and their relations.

Draw a UML Class Diagram as domain model for the REMS system showing the domain concepts, their relationships and potential generalizations. Specify multiplicities for your associations and arrange all objects into a containment hierarchy by appropriate composition relations between classes.

Describe the **high-level workflow** of the *paper review process* using a UML Activity Diagram. You may assume that the successful invitation of a reviewer is separated into an activity called *Invite-and-Accept-Review* which you may use in your diagram. Your actions should have direct traceability to use cases!

Describe the **state-based behavior** of the "*Paper*" class by a UML Statechart Diagram. Use operations derived from use cases as triggering events of transitions. (The Paper class represents a submission that is handled by REMS for review.)

Modeling with UML /
Dániel Varró & Kristian Sandahl

www.liu.se

LINKÖPING UNIVERSITY