

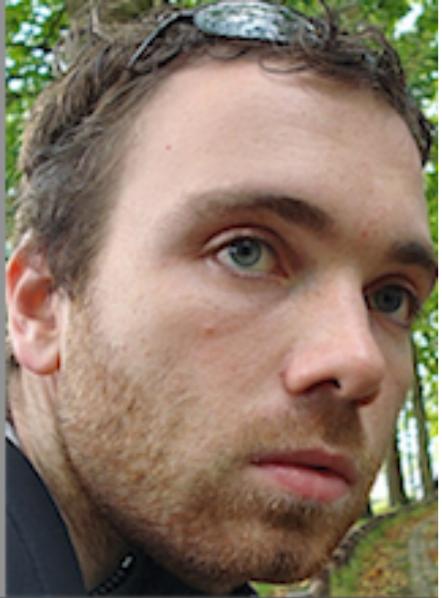


Model-driven Engineering & Domain-Specific Languages

UNIVERSITÉ
CÔTE D'AZUR 

Sébastien Mosser (UCA, I3S)
ENS Lyon, 14.09.2017

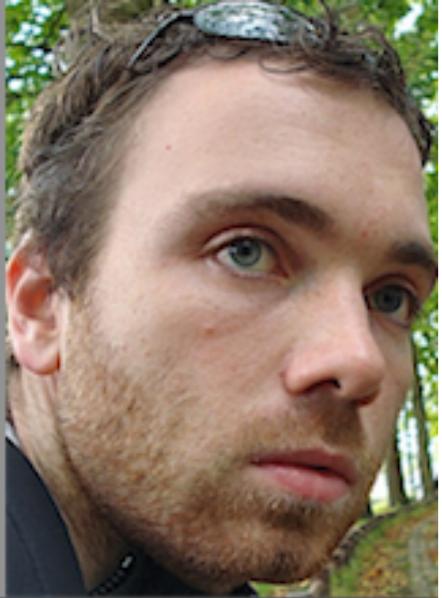




Sébastien Mosser

"geek, snowboarder & composition-driven guy"

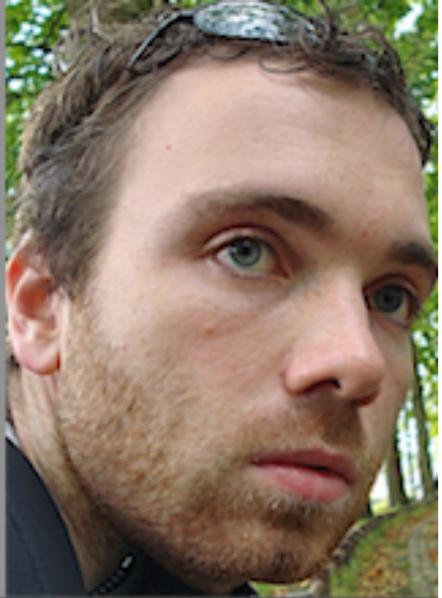
- 12-...: Associate Professor, UNS
- 11-12: Research Scientist, SINTEF (NO)
- 10-11: Postdoc, Inria Lille Nord-Europe
- 2010: PhD Thesis, Software Composition



Sébastien Mosser

"geek, snowboarder & composition-driven guy"

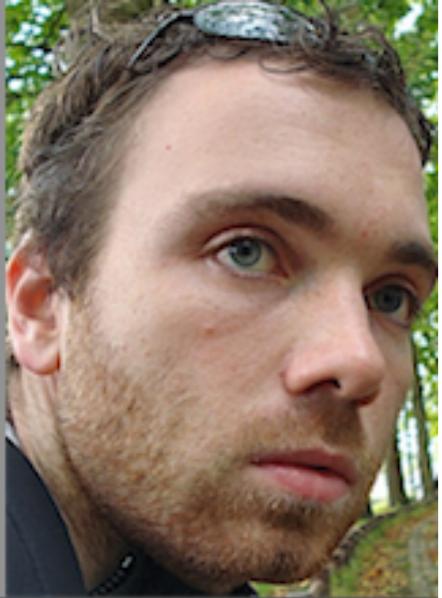
- 12-...: Associate Professor, UNS
- 11-12: Research Scientist, SINTEF (NO)
- 10-11: Postdoc, Inria Lille Nord-Europe
- 2010: PhD Thesis, Software Composition MDE



Sébastien Mosser

"geek, snowboarder & composition-driven guy"

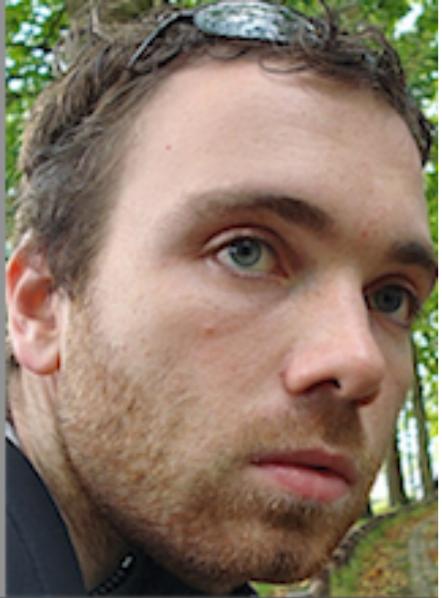
- 12-...: Associate Professor, UNS
- 11-12: Research Scientist, SINTEF (NO)
- 10-11: Postdoc, Inria Lille Nord-Europe MDE
- 2010: PhD Thesis, Software Composition MDE



Sébastien Mosser

"geek, snowboarder & composition-driven guy"

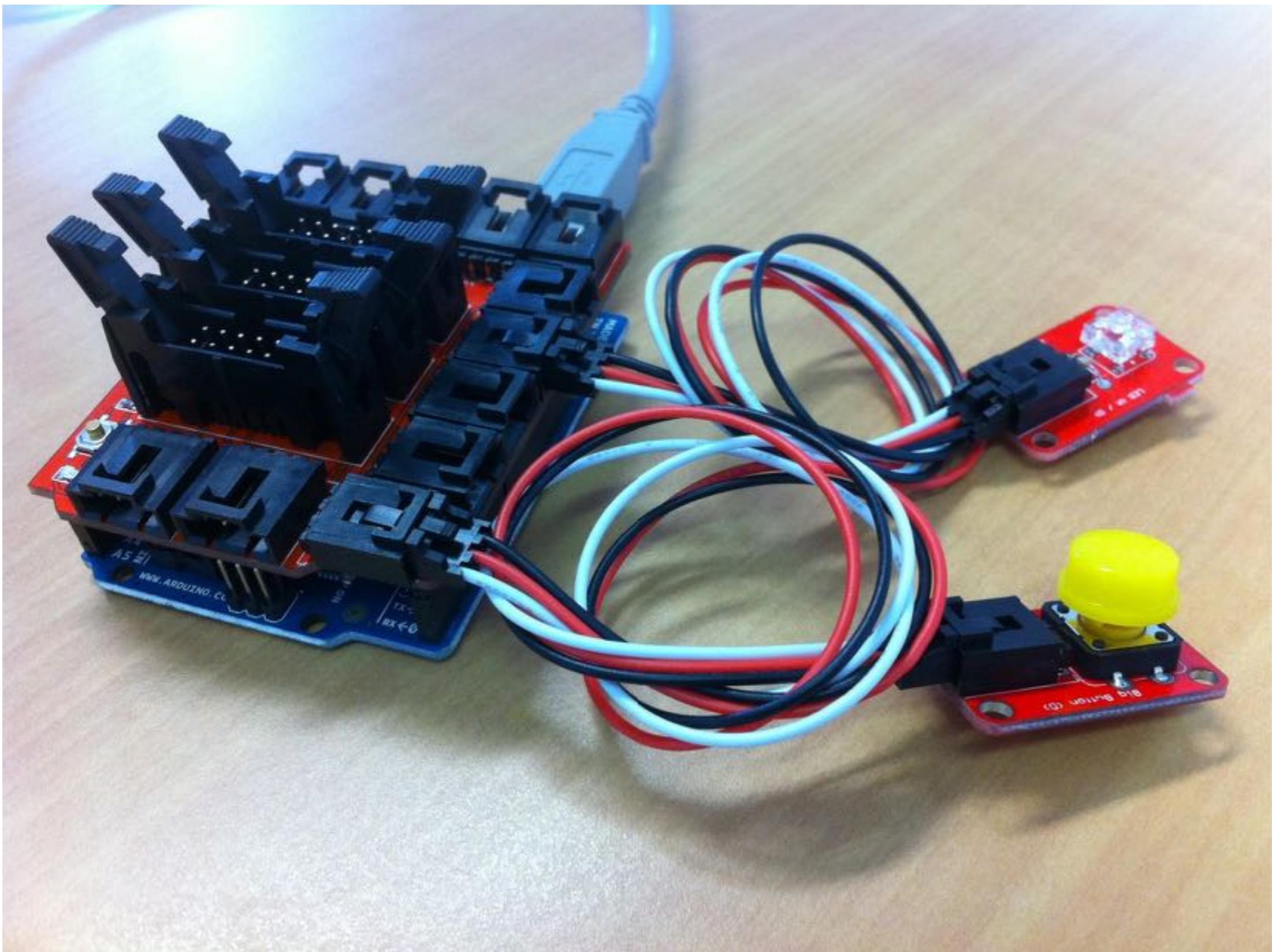
- 12-...: Associate Professor, UNS
- 11-12: Research Scientist, SINTEF (NO) MDE & DSL
- 10-11: Postdoc, Inria Lille Nord-Europe MDE
- 2010: PhD Thesis, Software Composition MDE



Sébastien Mosser

"geek, snowboarder & composition-driven guy"

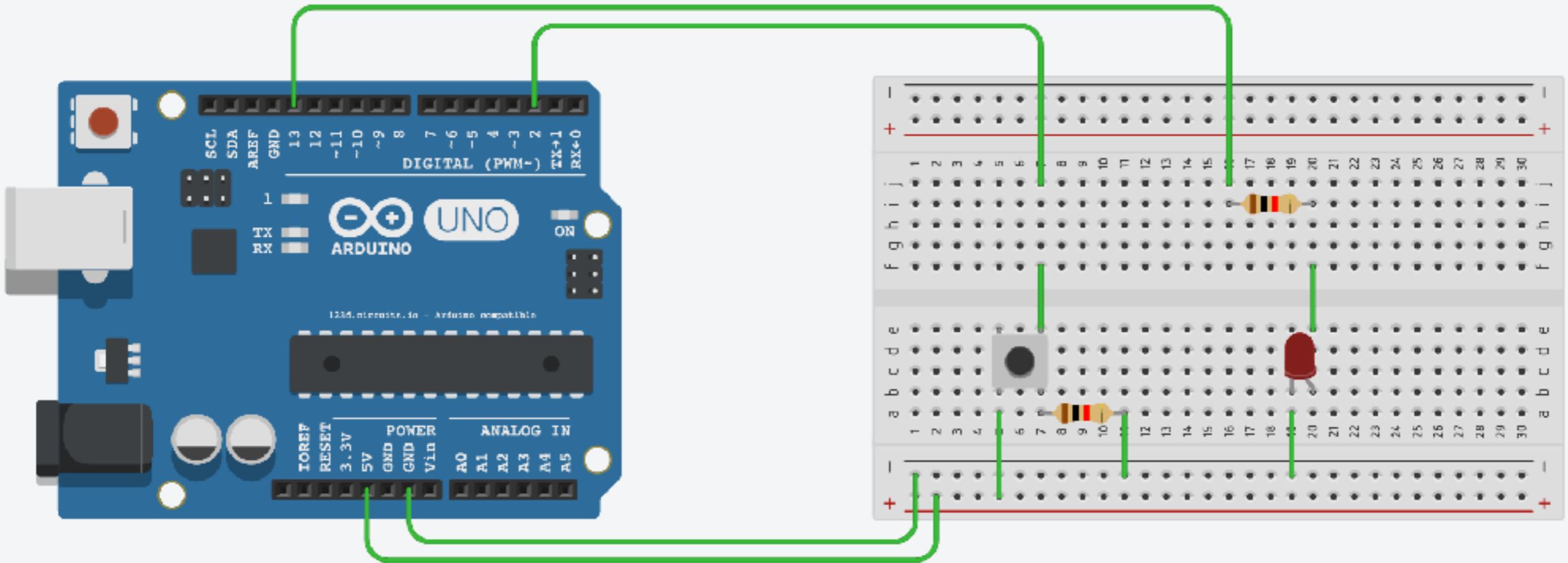
- 12-...: Associate Professor, UNS DSL
- 11-12: Research Scientist, SINTEF (NO) MDE & DSL
- 10-11: Postdoc, Inria Lille Nord-Europe MDE
- 2010: PhD Thesis, Software Composition MDE



Sébastien Mosser, Philippe Collet, Mireille Blay-Fornarino. "Exploiting the Internet of Things to Teach Domain Specific Languages and Modeling" in Proceedings of the 10th Educators' Symposium at MODELS 2014 (EduSymp'14), ACM, IEEE, pages 1-10, Springer LNCS, Valencia, Spain, 29 september 2014



Arduino Simulator



<https://circuits.io/circuits/2380885-big-red-button>

```
int state = LOW; int prev = HIGH;
long t = 0; long debounce = 200;
void setup() {
    pinMode(8, INPUT);
    pinMode(11, OUTPUT);
}
void loop() {
    int reading = digitalRead(8);
    if (reading == HIGH && prev == LOW
        && millis() - t > debounce) {
        if (state == HIGH) {
            state = LOW;
        } else { state = HIGH; }
        time = millis();
    }
    digitalWrite(11, state);
    prev = reading;
}
```

```
int state = LOW; int prev = HIGH;
long t = 0; long debounce = 200;
void setup() {
    pinMode(8, INPUT);
    pinMode(11, OUTPUT);
}
void loop() {
    int reading = digitalRead(8)
    if (reading == HIGH && prev == LOW
        && millis() - t > debounce) {
        if (state == HIGH) {
            state = LOW;
        } else { state = HIGH; }
        t = millis();
    }
    digitalWrite(11, state);
    prev = reading;
}
```

serious?



```
app RedButton init_state : off {  
    bricks :  
        actuator red_led : 12  
        sensor button : 9  
    states :  
        off {  
            red_led <= low  
            button is high => on  
        }  
        on {  
            red_led <= high  
            button is high => off  
        }  
}
```





Tom Guillermin @TomGuillermin · 9 déc.

@petitroll Je vois déjà un DSL pour définir les musiciens d'un groupe qui feraient une Jam session !



...

Afficher la conversation



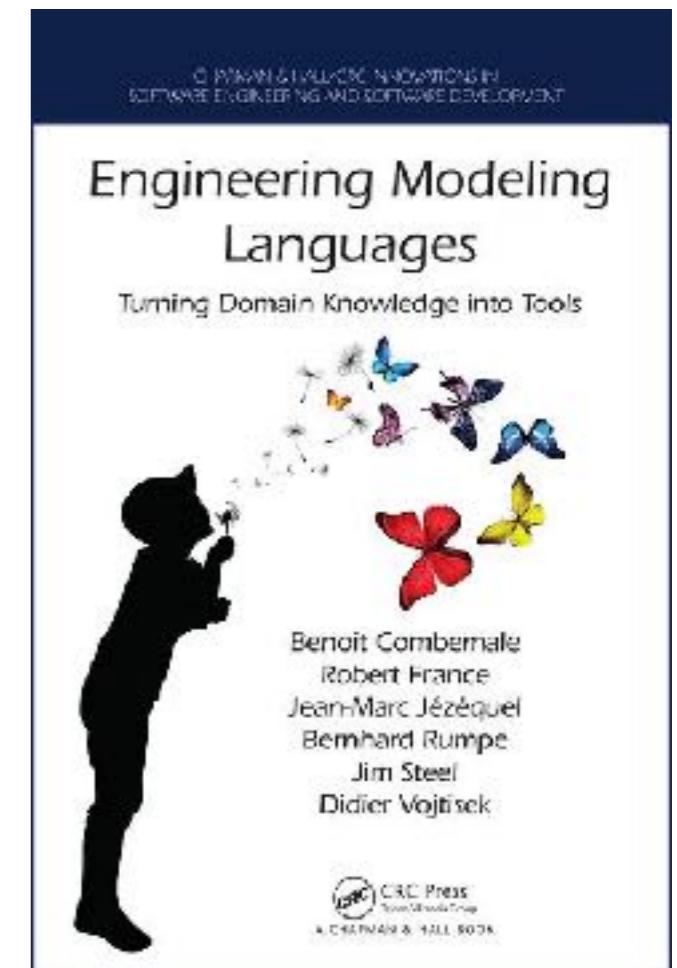
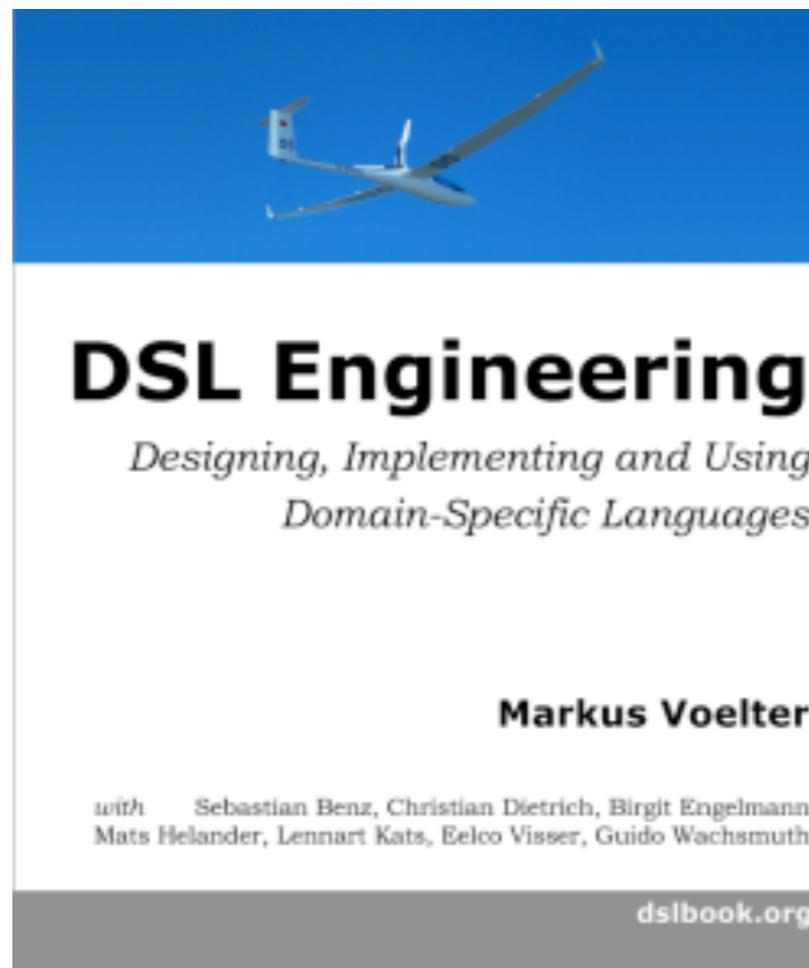
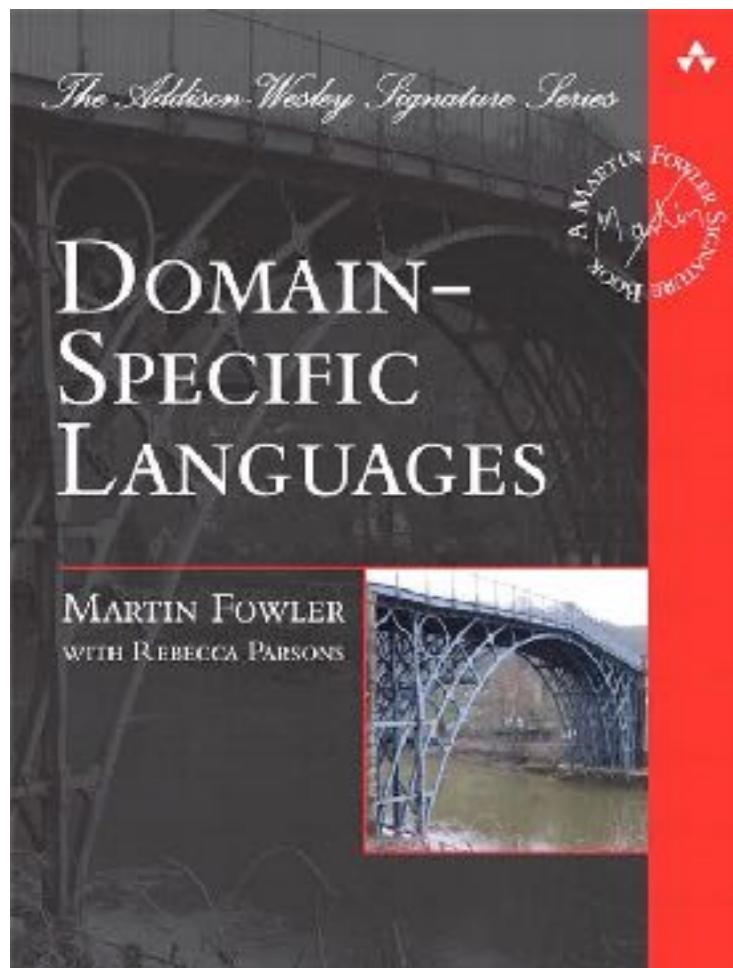
http://www.dailymotion.com/video/x3b6q8g_toa-mata-band-le-premier-groupe-de-robots-lego_music?start=40

«I see a **great future** for
very **systematic** and
very modest
programming languages.»

«I see a **great future** for
very **systematic** and
very modest
programming languages.»

- E.W. Dijkstra (1972)

Bibliography



[Fowler]

[Voelter]

[Combemale et al]

Planning

- #37: Course introduction
 - #38: Verification & Validation
 - #39: Unsupervised lab
-
- #40: Adv. code Generation
 - #41: Variability modelling
 - #42: Software Composition
 - #48: Bibliography defence

ArduinoML

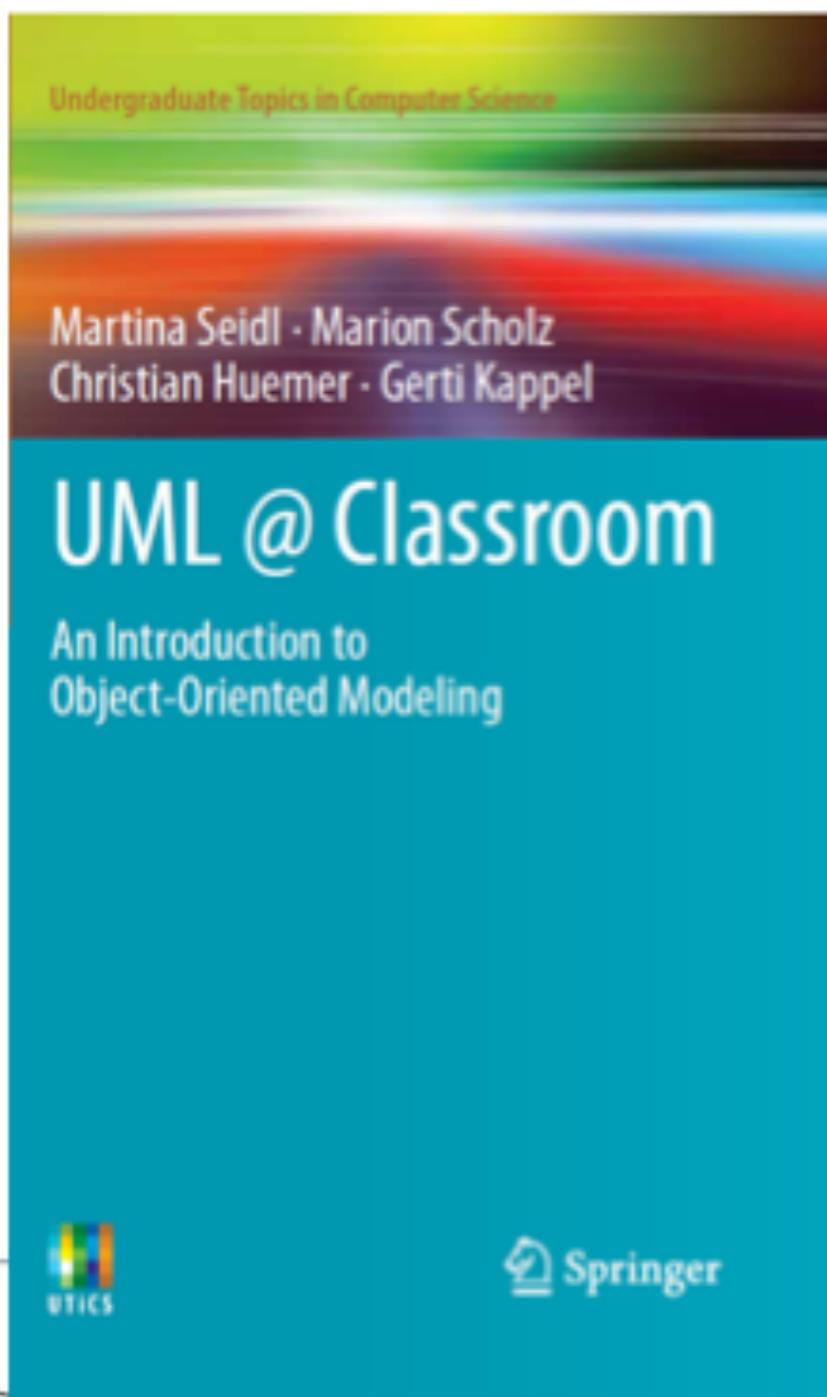
**Scientific
Workflows**

(Object-oriented) Models



Literature

- The lecture is based on the following book:



**UML @ Classroom:
An Introduction to Object-Oriented
Modeling**

Martina Seidl, Marion Scholz, Christian
Huemer and Gerti Kappel

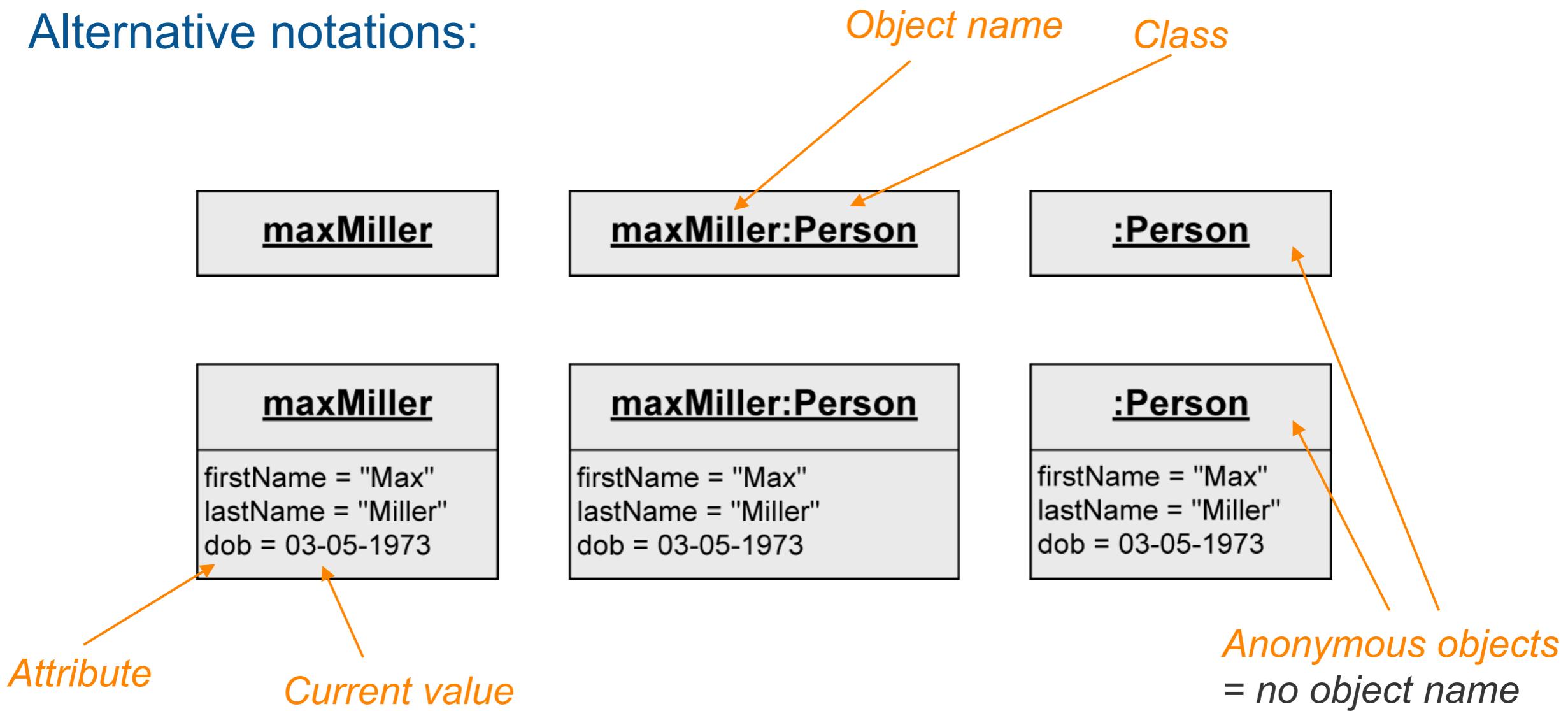
Springer Publishing, 2015

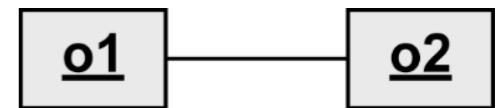
ISBN 3319127411

- Use Case Diagram
- **Structure Modeling**
- State Machine Diagram
- Sequence Diagram
- Activity Diagram

Object

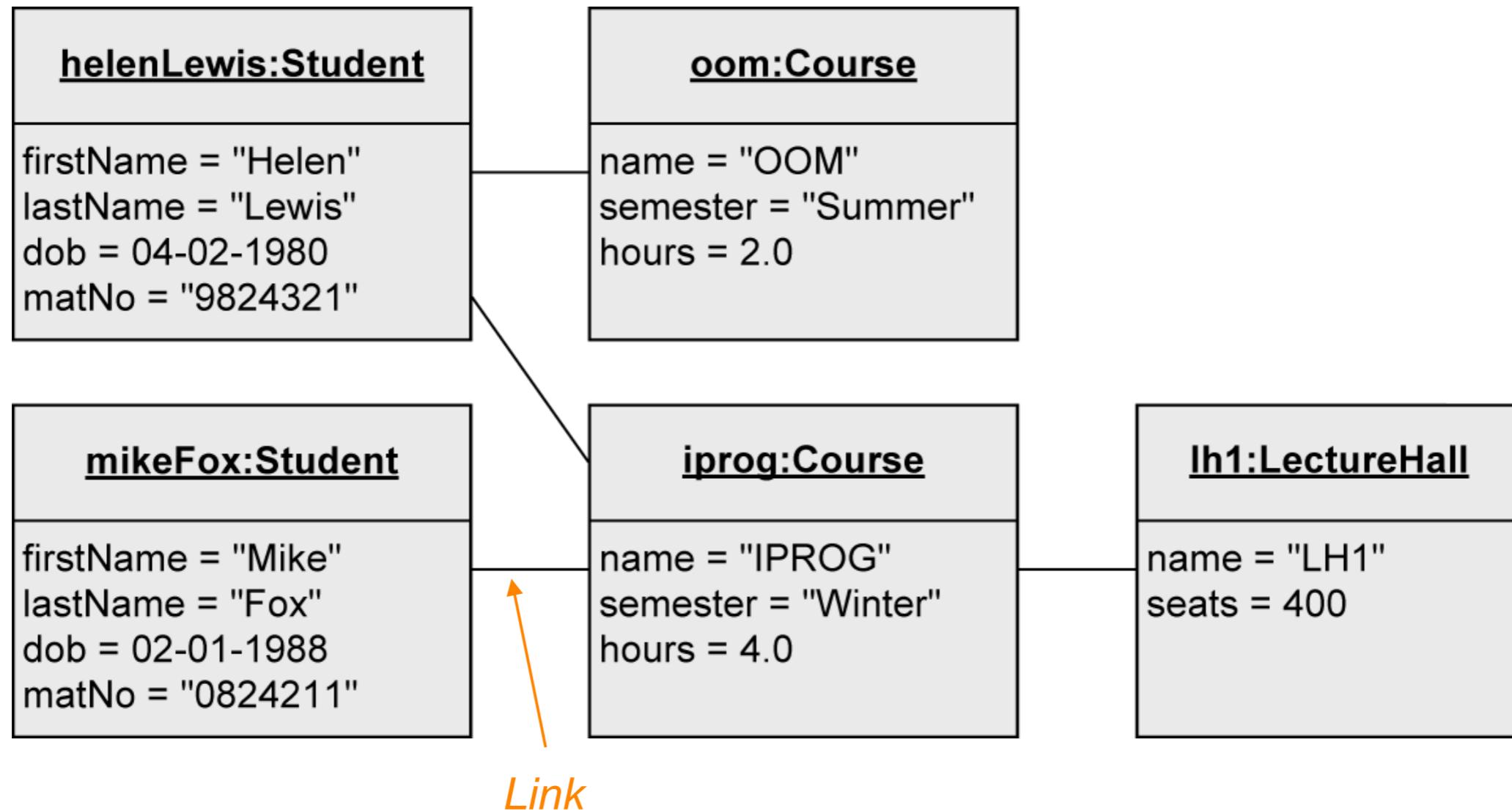
- Individuals of a system
- Alternative notations:





Object Diagram

- Objects of a system and their relationships (links)
- Snapshot of objects at a specific moment in time



From Object to Class

- Individuals of a system often have identical characteristics and behavior
- A class is a construction plan for a set of similar objects of a system

- Objects are instances of classes

Class

Person
firstName: String lastName: String dob: Date

- Attributes: structural characteristics of a class

- Different value for each instance (= object)

- Operations: behavior of a class

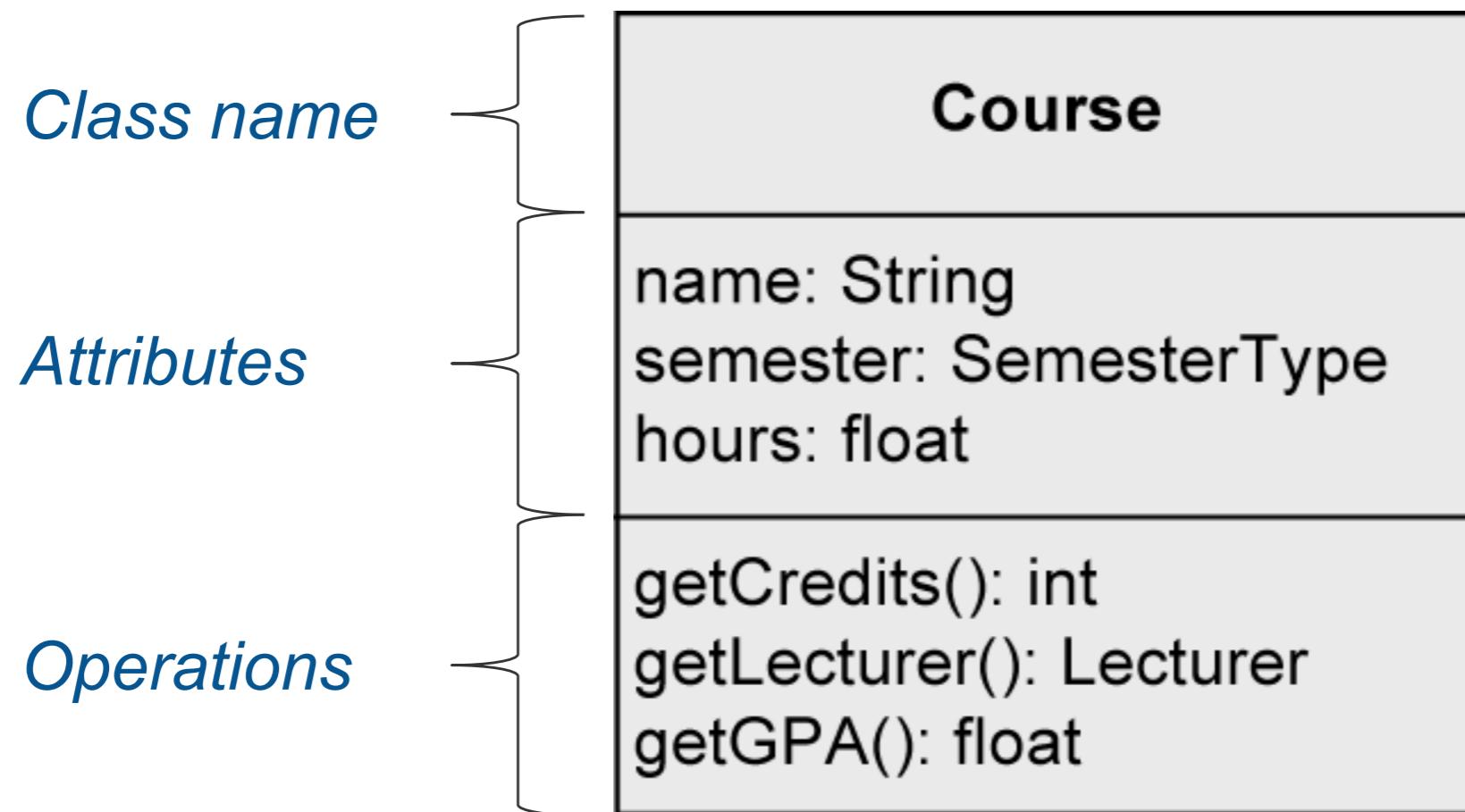
- Identical for all objects of a class
→ not depicted in object diagram

Object of that class

<u>maxMiller:Person</u>
firstName = "Max" lastName = "Miller" dob = 03-05-1973

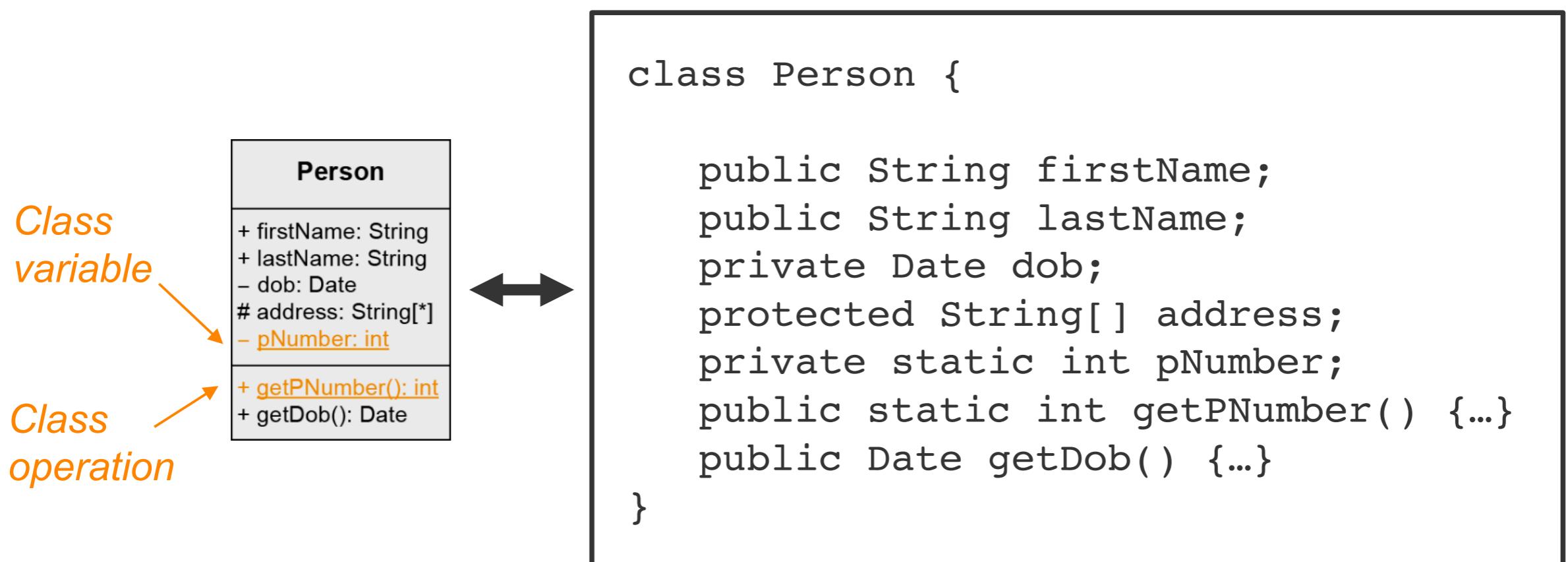


Class

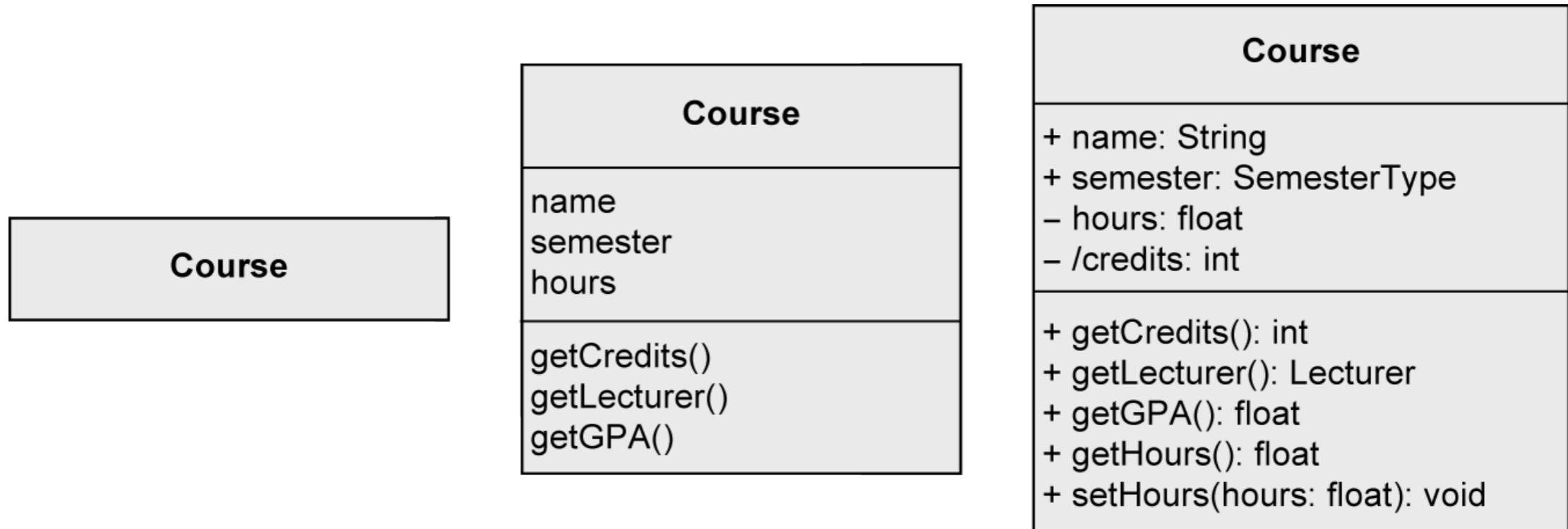


Class Variable and Class Operation

- Instance variable (= instance attribute): attributes defined on instance level
- Class variable (= class attribute, static attribute)
 - Defined only once per class, i.e., shared by all instances of the class
 - E.g. counters for the number of instances of a class, constants, etc.
- Class operation (= static operation)
 - Can be used if no instance of the corresponding class was created
 - E.g. constructors, counting operations, math. functions ($\sin(x)$), etc.
- Notation: underlining name of class variable / class operation

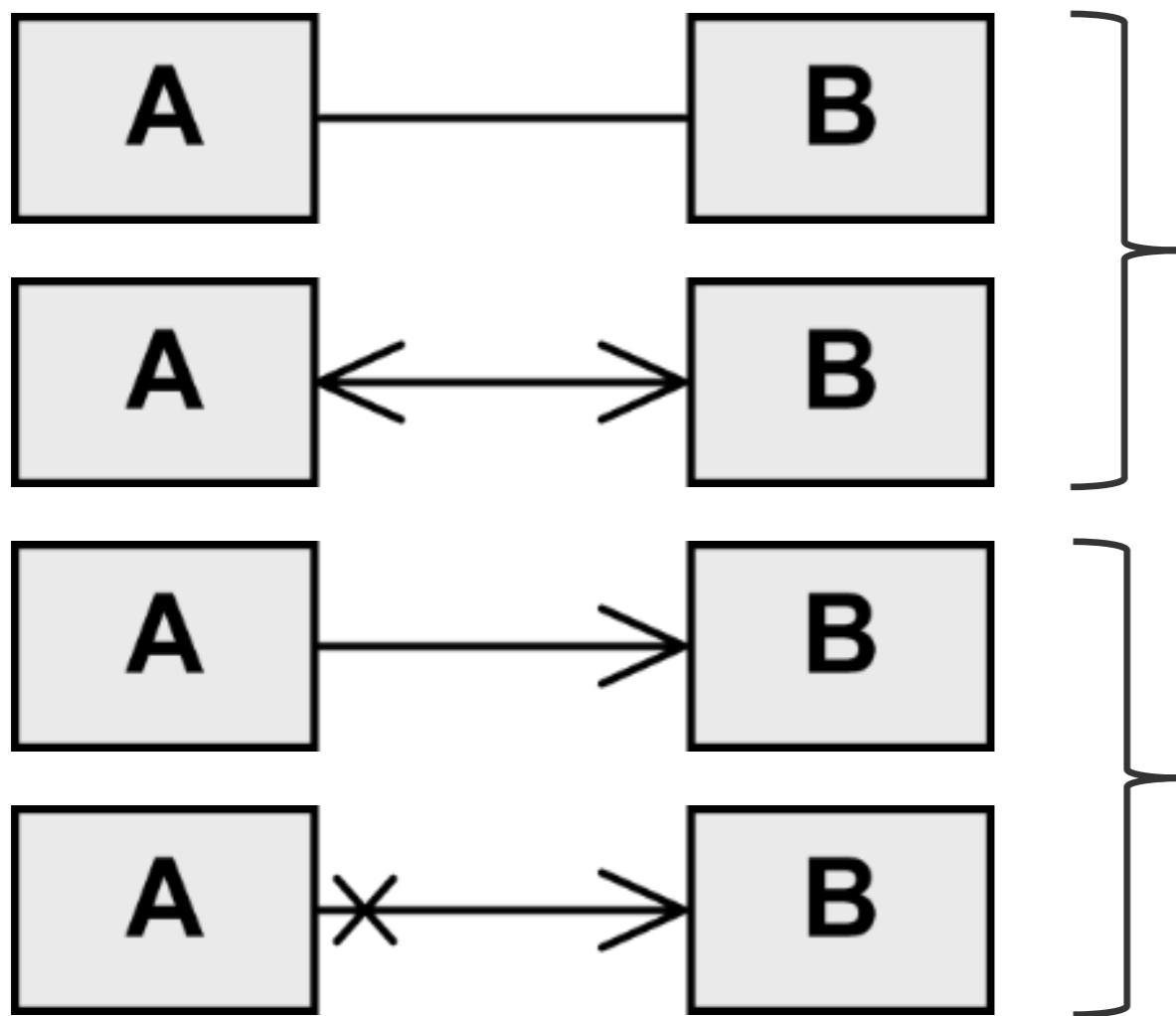


Specification of Classes: Different Levels of Detail

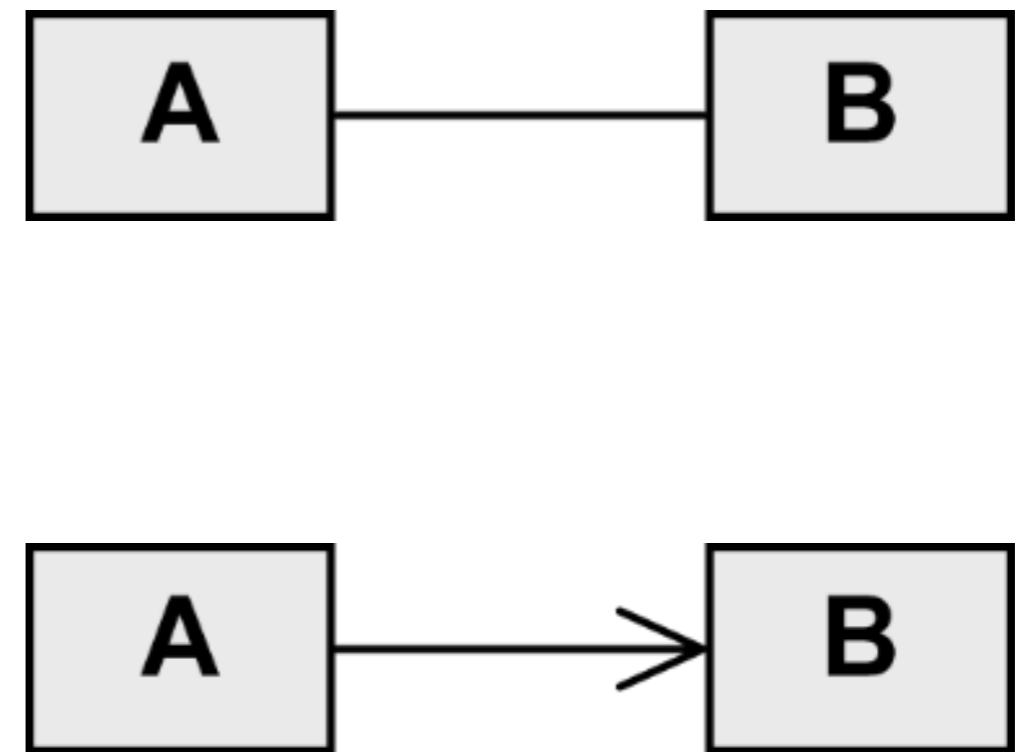


Navigability – UML Standard vs. Best Practice

UML standard

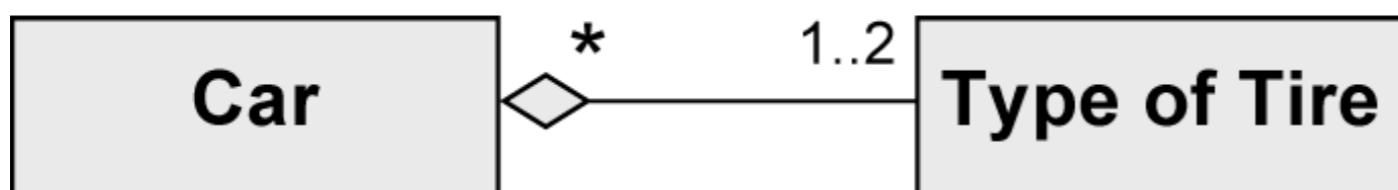
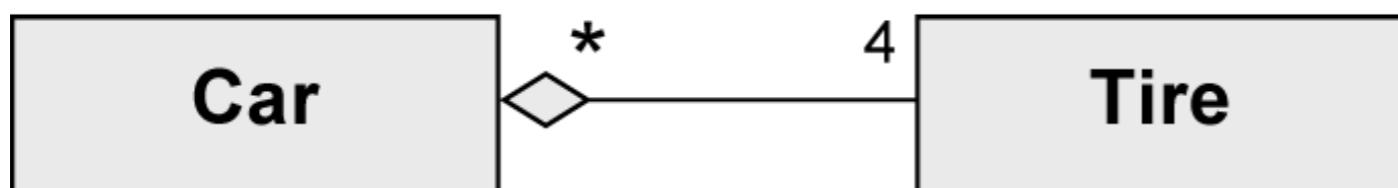
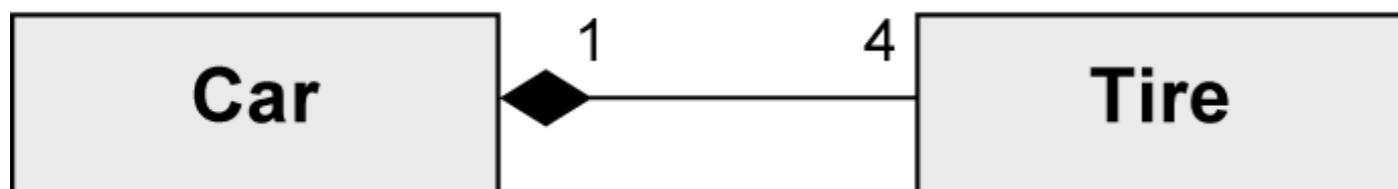
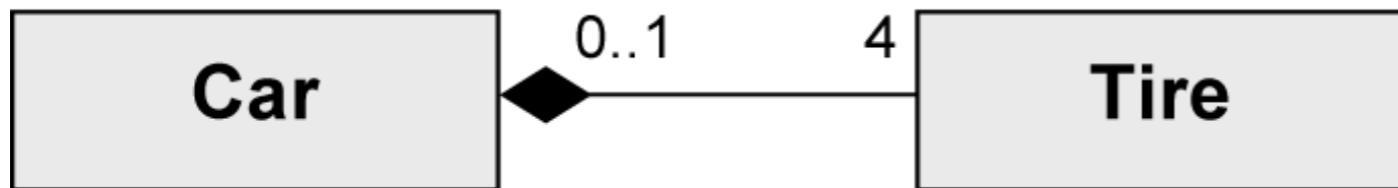


Best practice



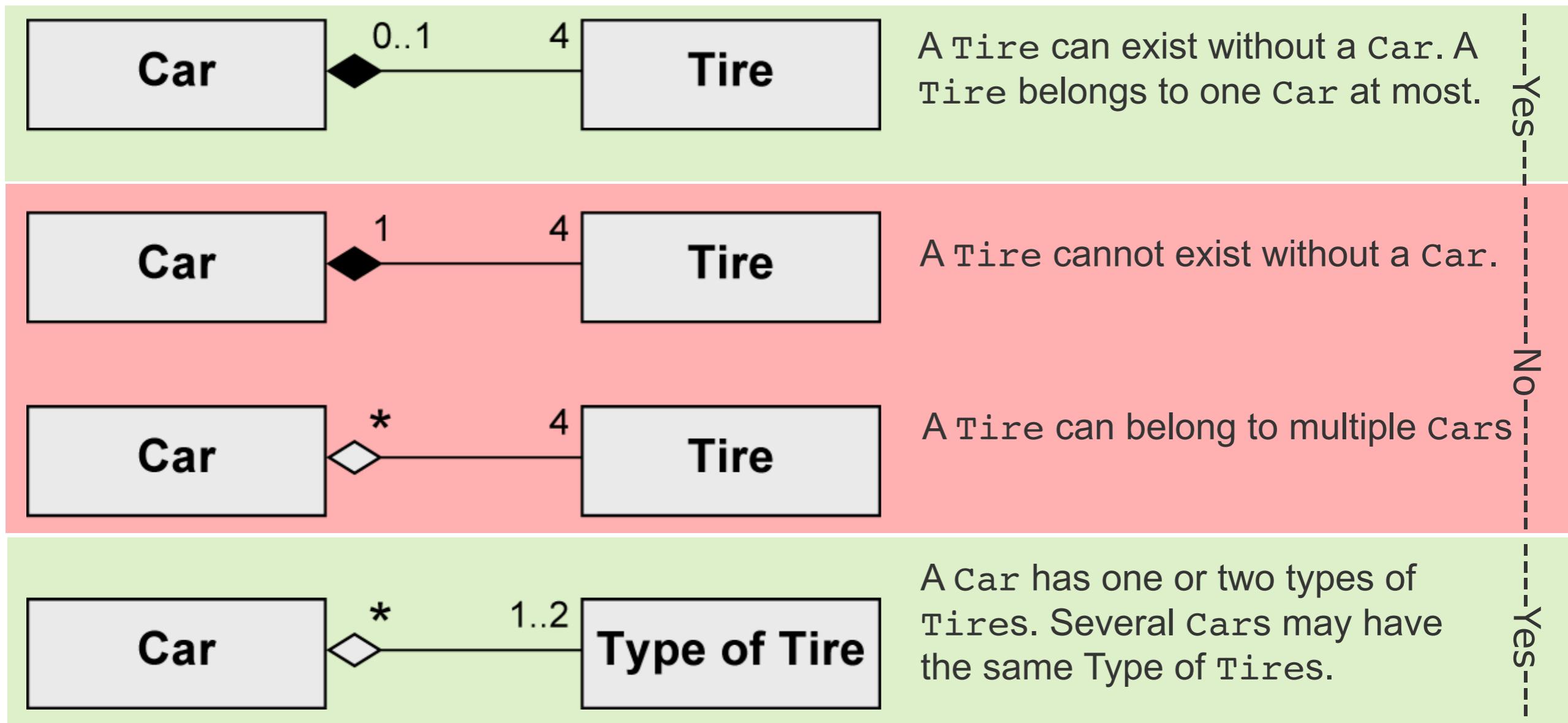
Shared Aggregation and Composition

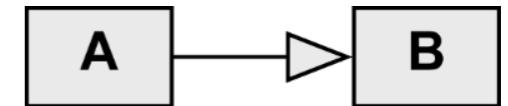
- Which model applies?



Shared Aggregation and Composition

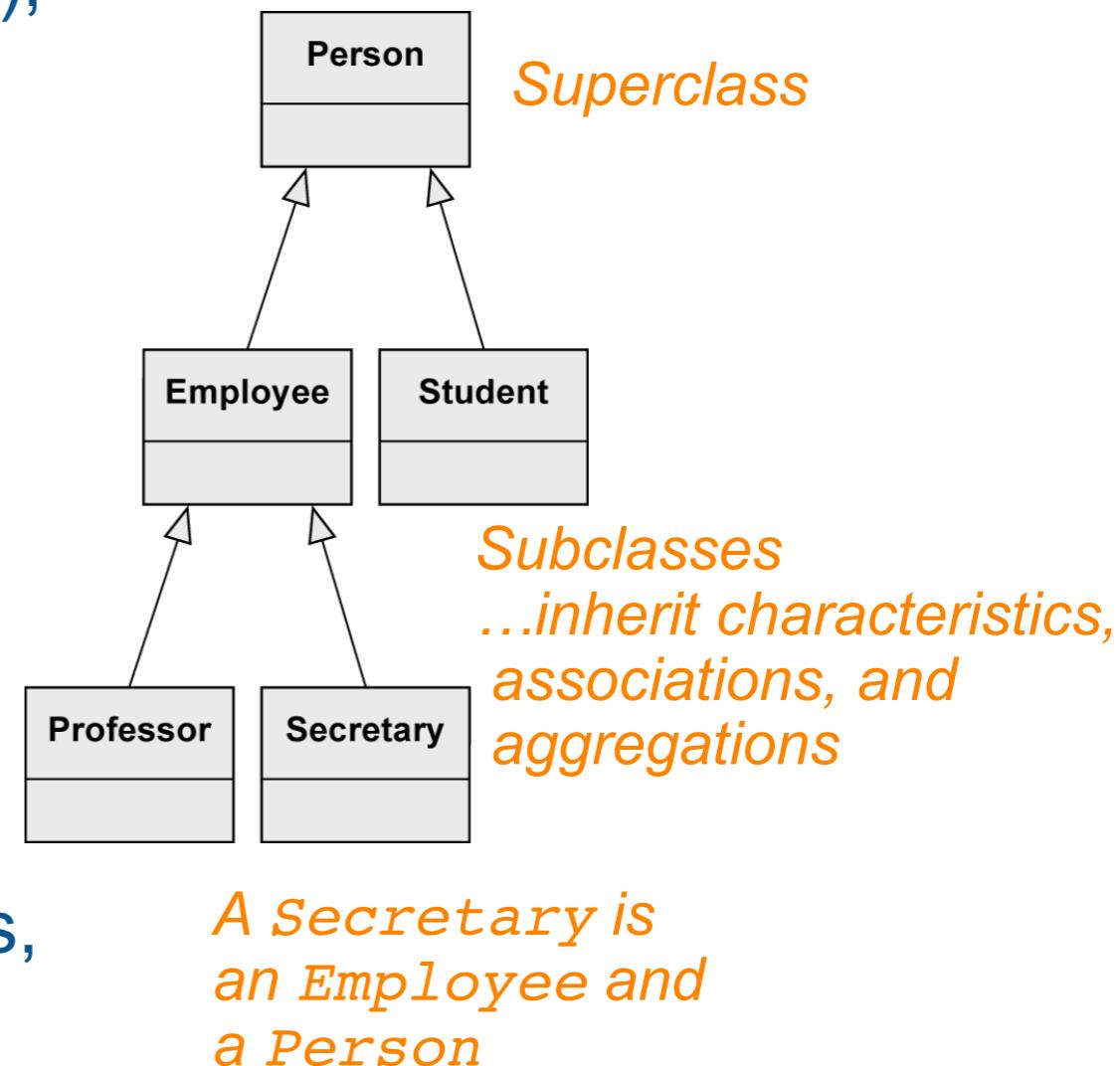
- Which model applies?





Generalization

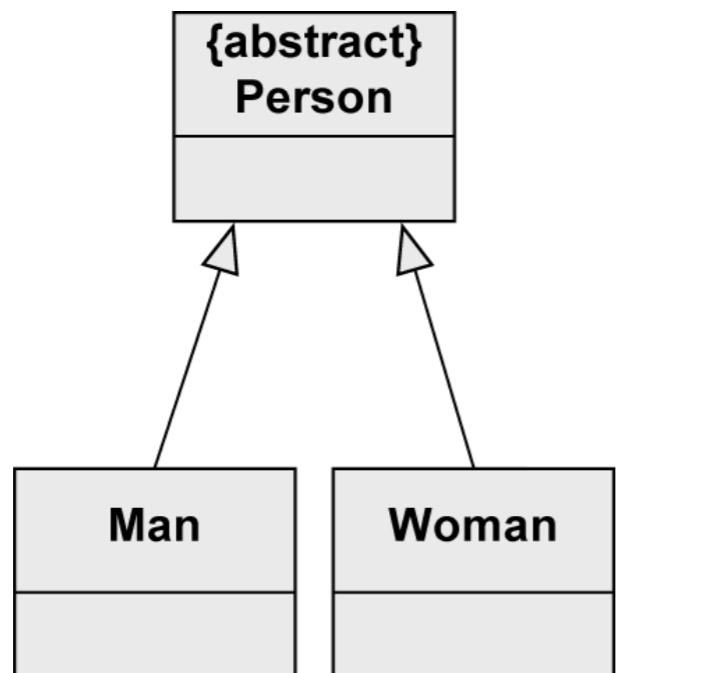
- Characteristics (attributes and operations), associations, and aggregations that are specified for a general class (superclass) are passed on to its subclasses.
- Every instance of a subclass is simultaneously an indirect instance of the superclass.
- Subclass inherits all characteristics, associations, and aggregations of the superclass except private ones.
- Subclass may have further characteristics, associations, and aggregations.
- Generalizations are transitive.



Generalization – Abstract Class

- Used to highlight common characteristics of their subclasses.
- Used to ensure that there are no direct instances of the superclass.
- Only its non-abstract subclasses can be instantiated.
- Useful in the context of generalization relationships.
- Notation: keyword **{abstract}** or class name in italic font.

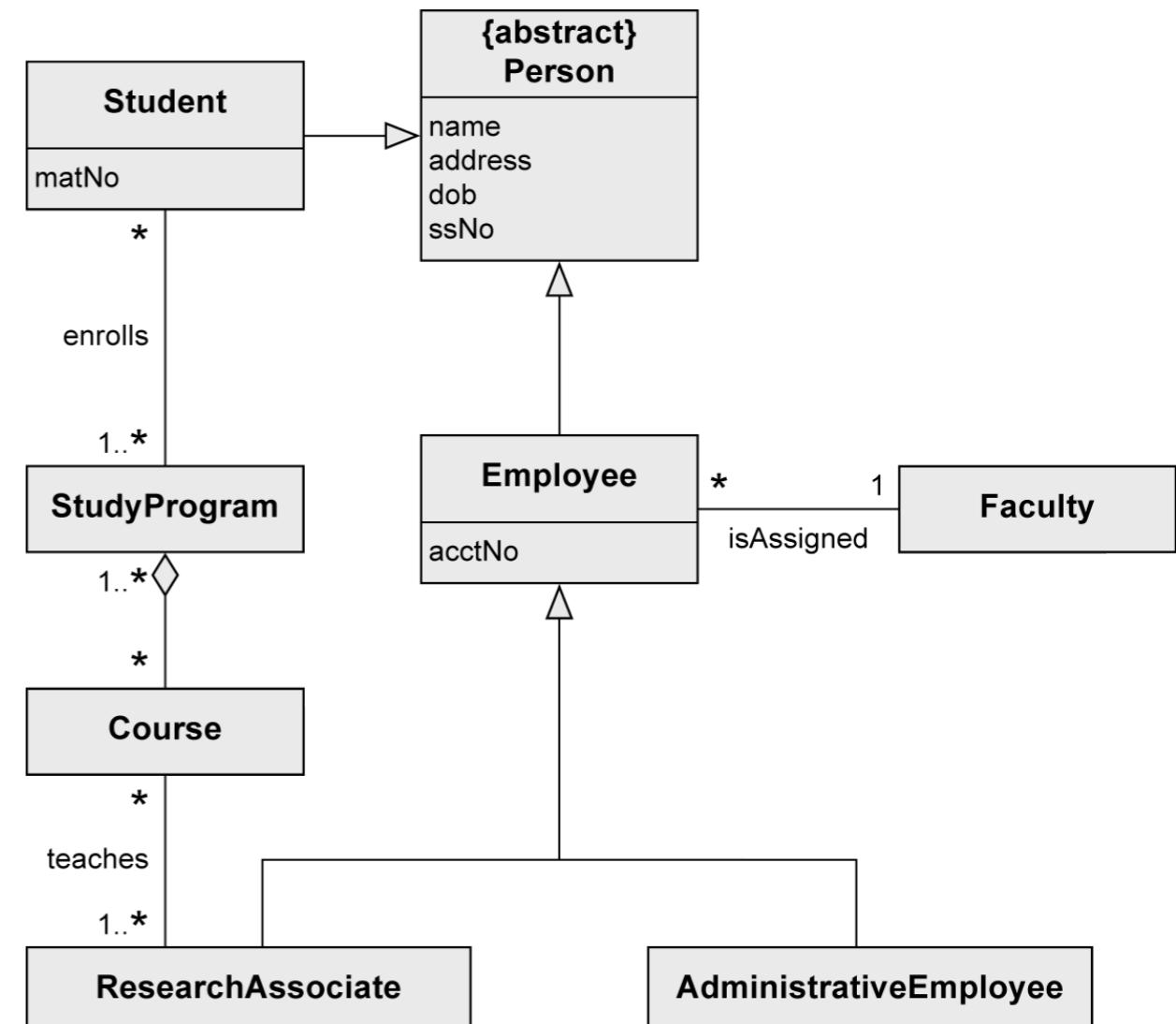
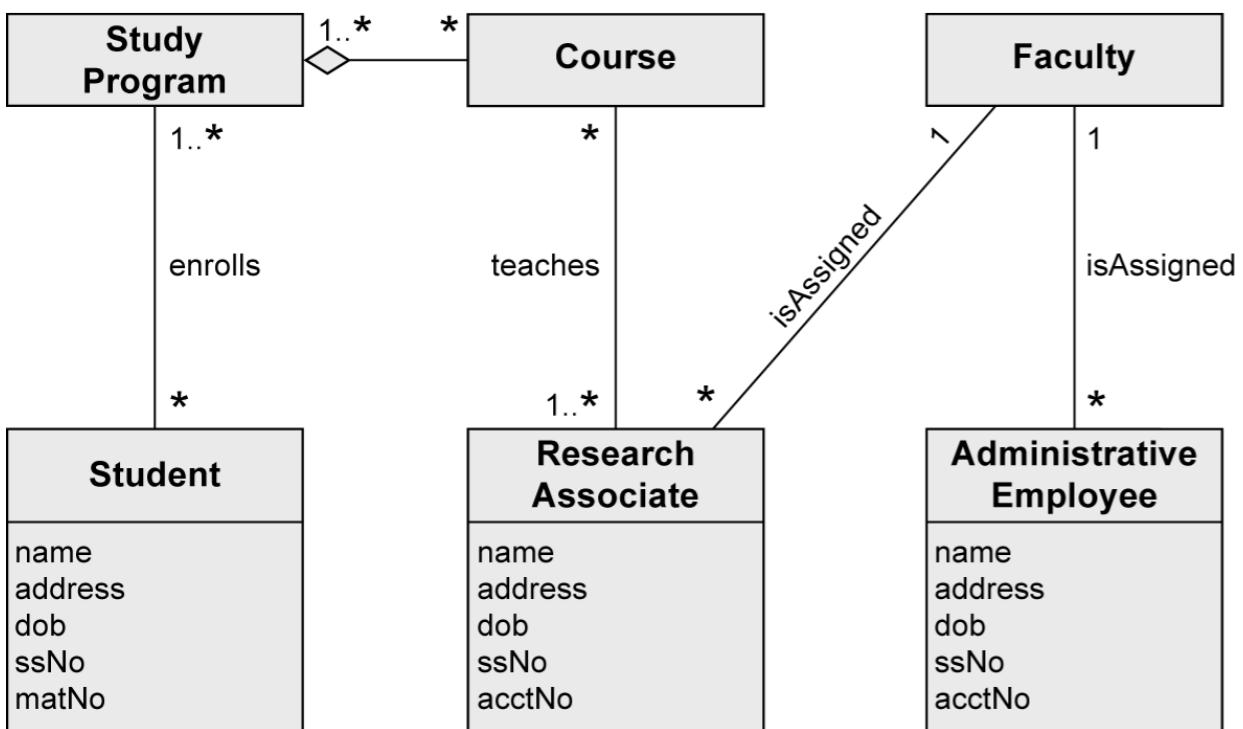
{abstract}
Person



No *Person-object* possible

Two types of Person: Man and Woman

With and Without Generalization



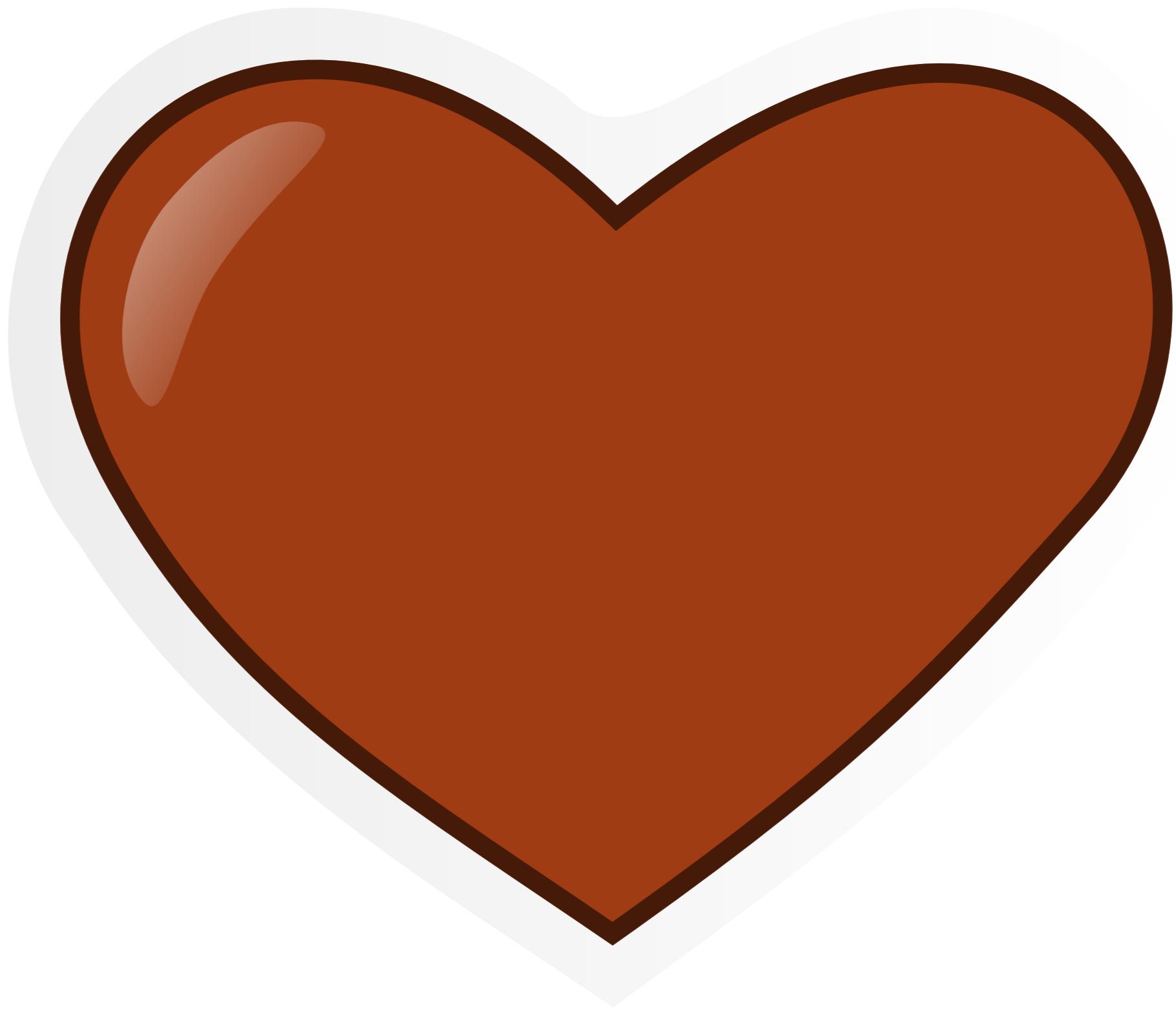
Model-driven Engineering



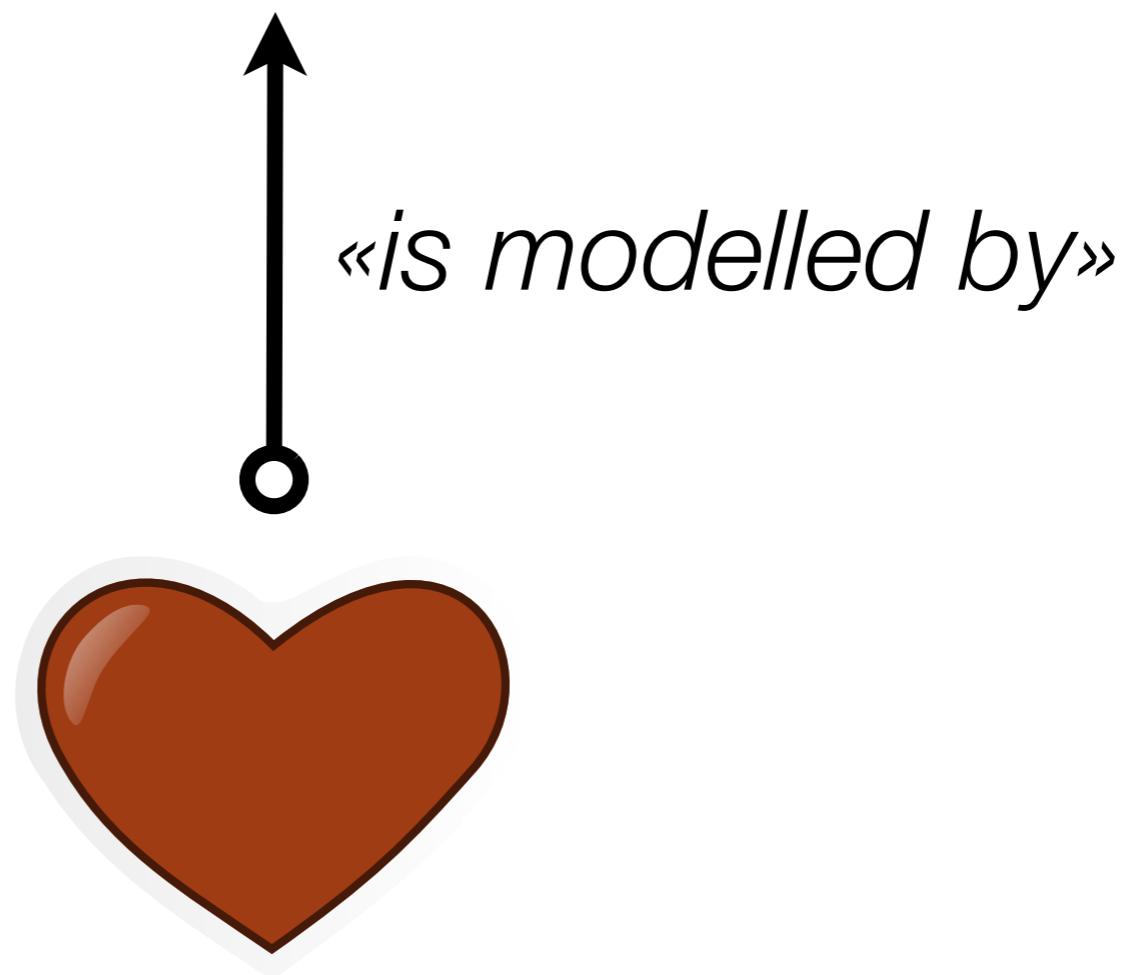
“

Why do we write programs,
instead of writing programs
that will write programs on
our behalf?

- JM Jezequel



I love model-driven engineering

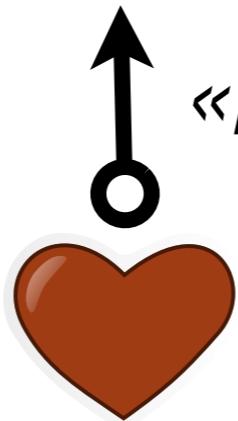


**A sentence contains a subject,
a verb, and a complement.**



«conforms to»

I love model-driven engineering



«is modelled by»

**A sentence contains a subject,
a verb, and a complement.**



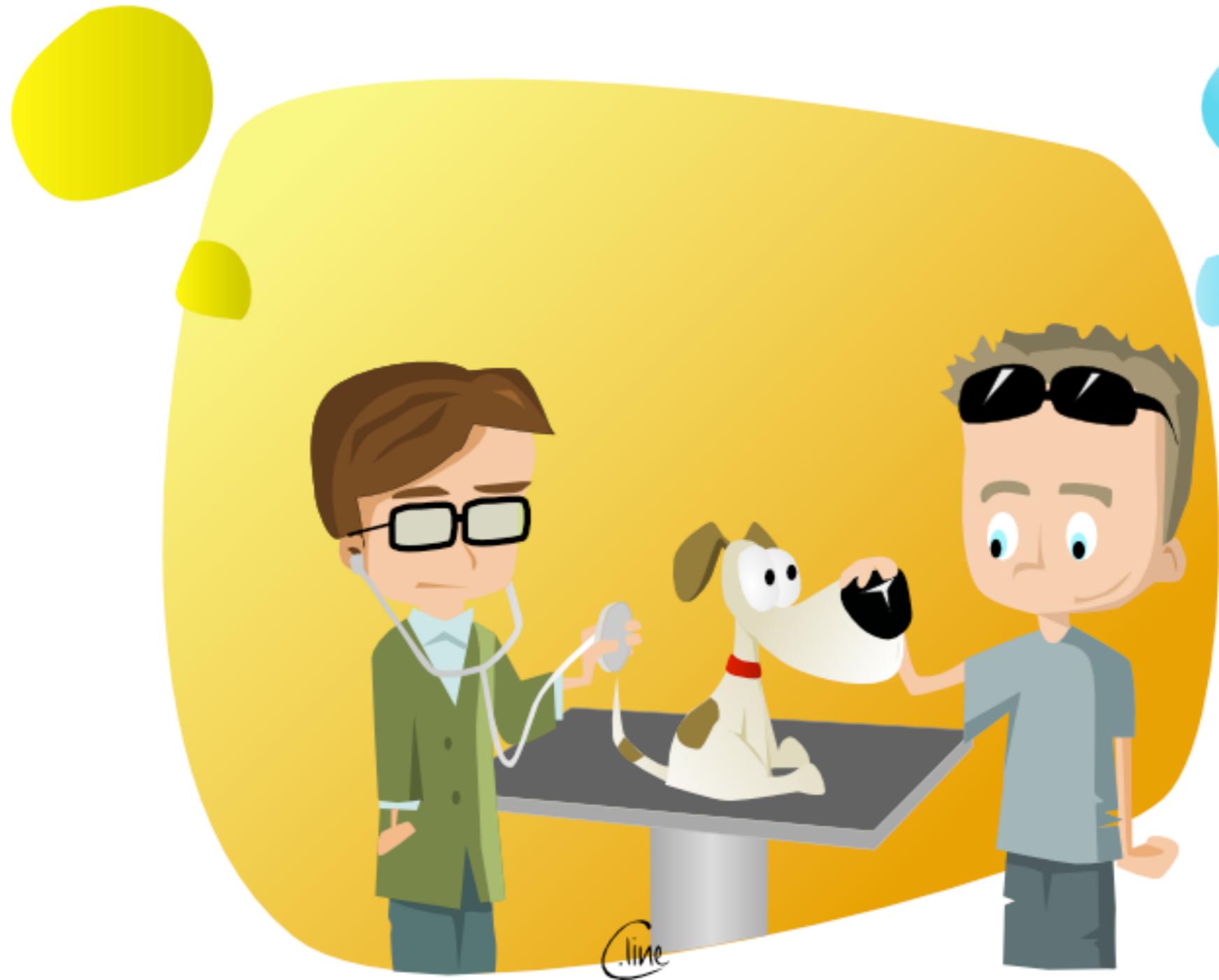
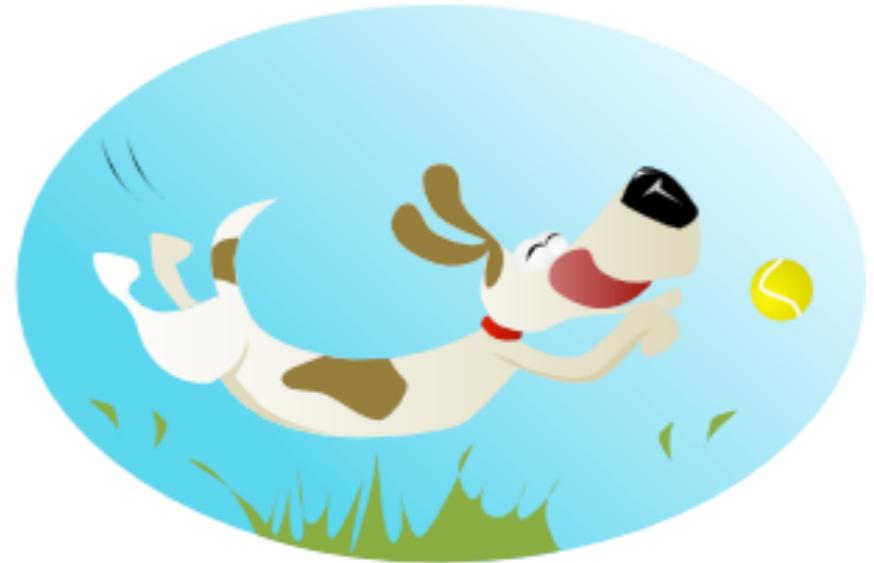
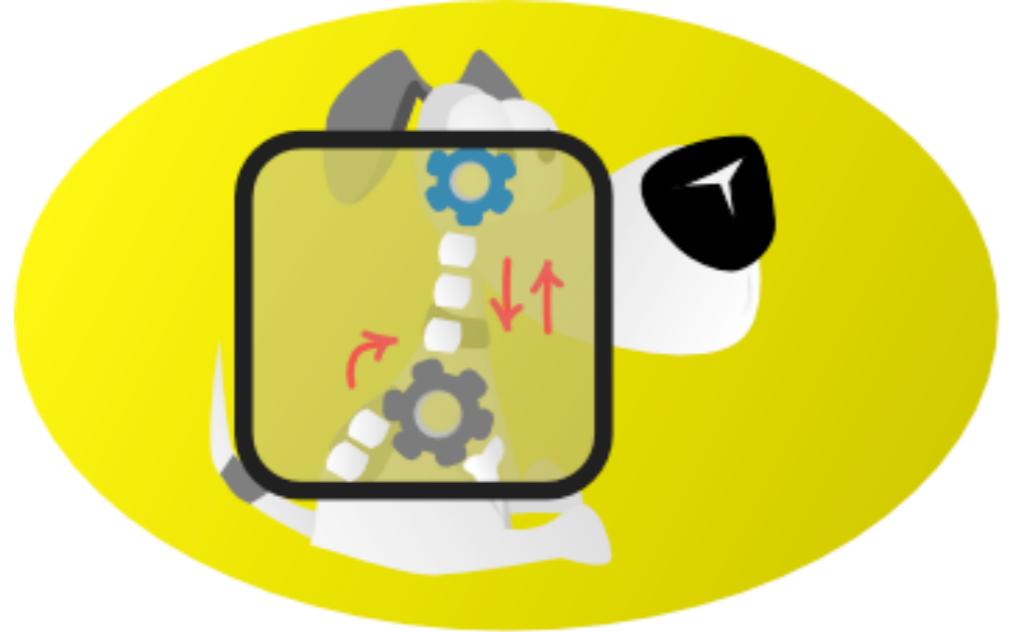
«conforms to»

A sentence contains a subject,
a verb, and a complement.



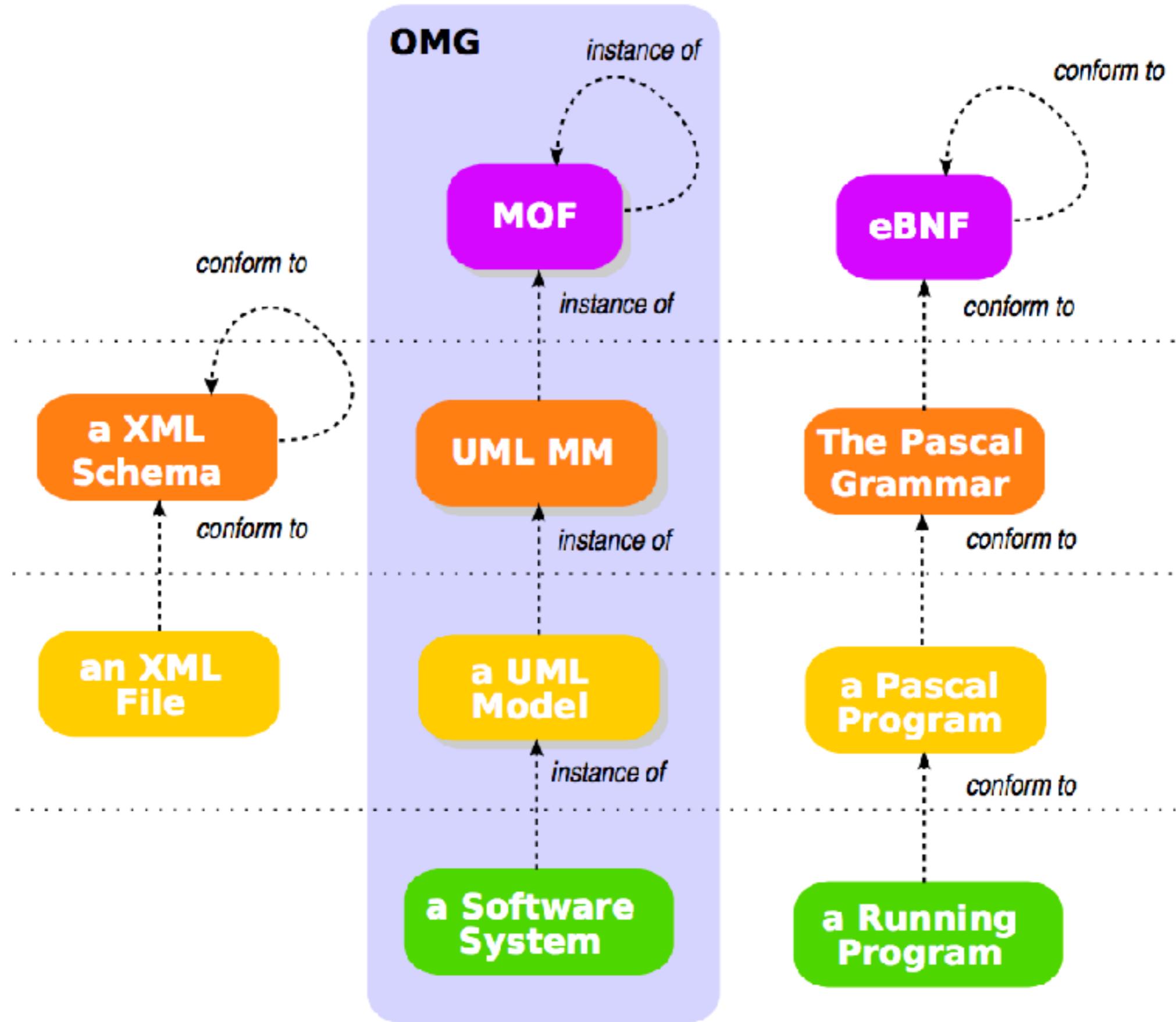
«conforms to»

I love model-driven engineering

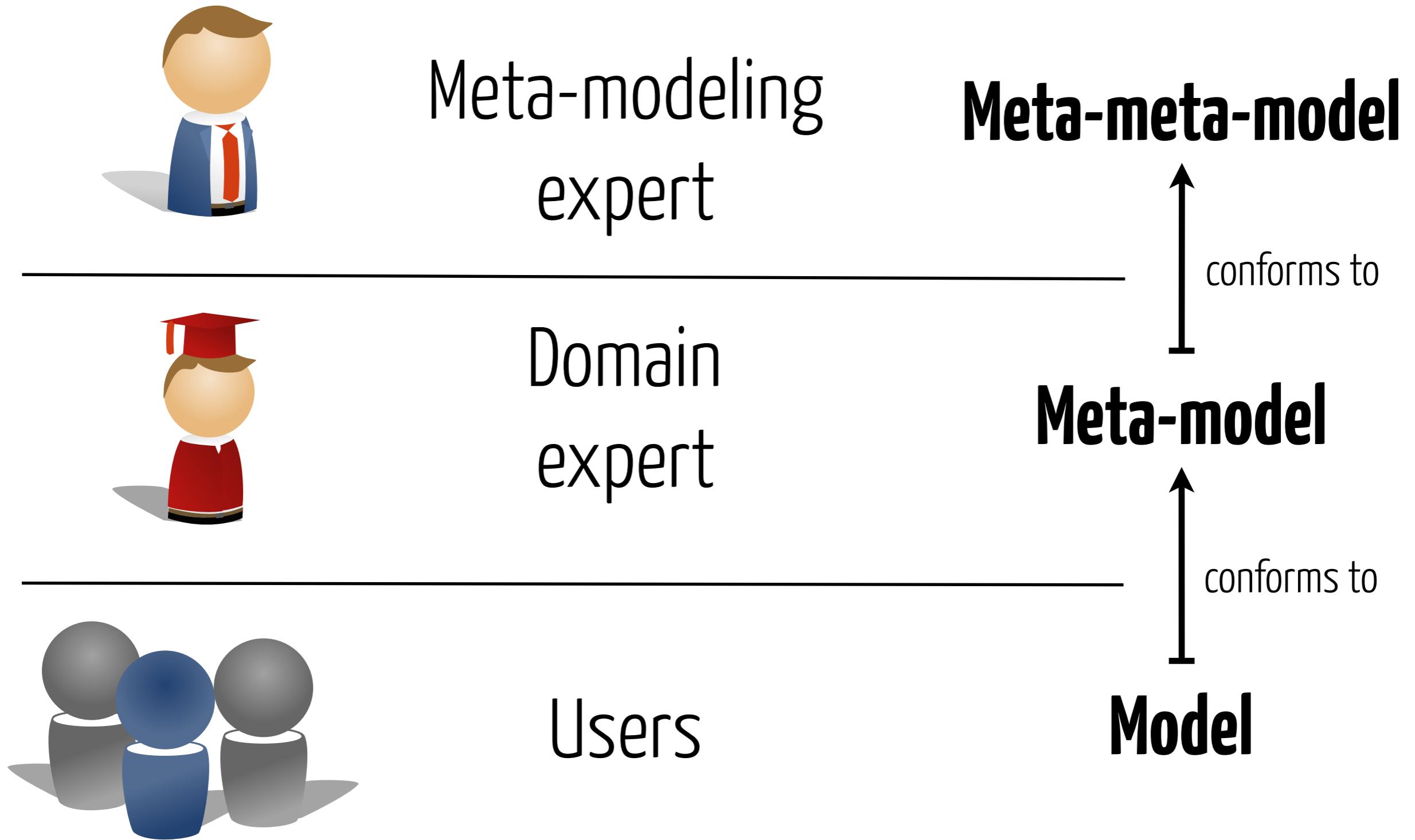


But, said Alice,
if the world has absolutely
No Sense,
who's stopping
us from
Inventing One?

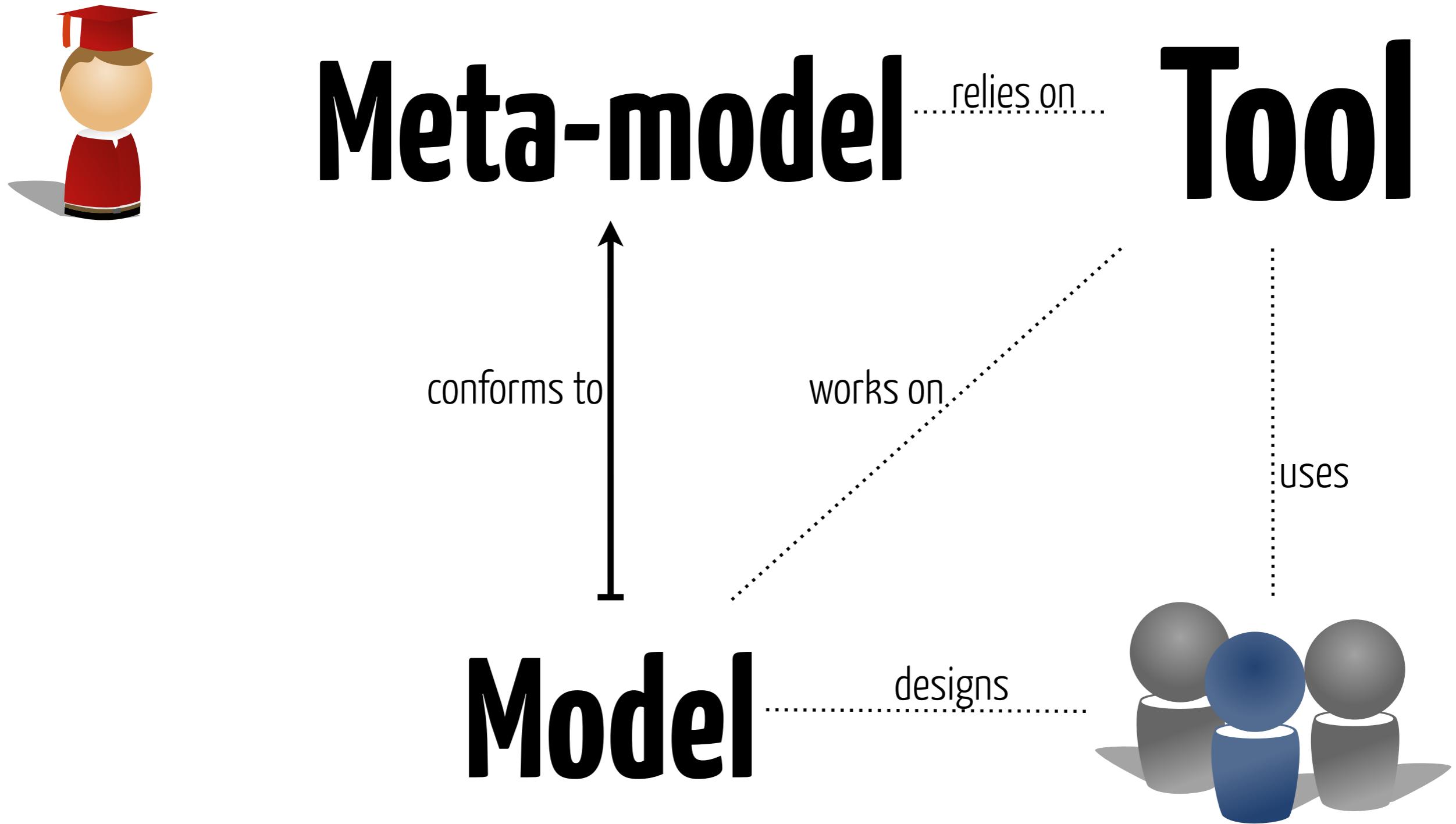




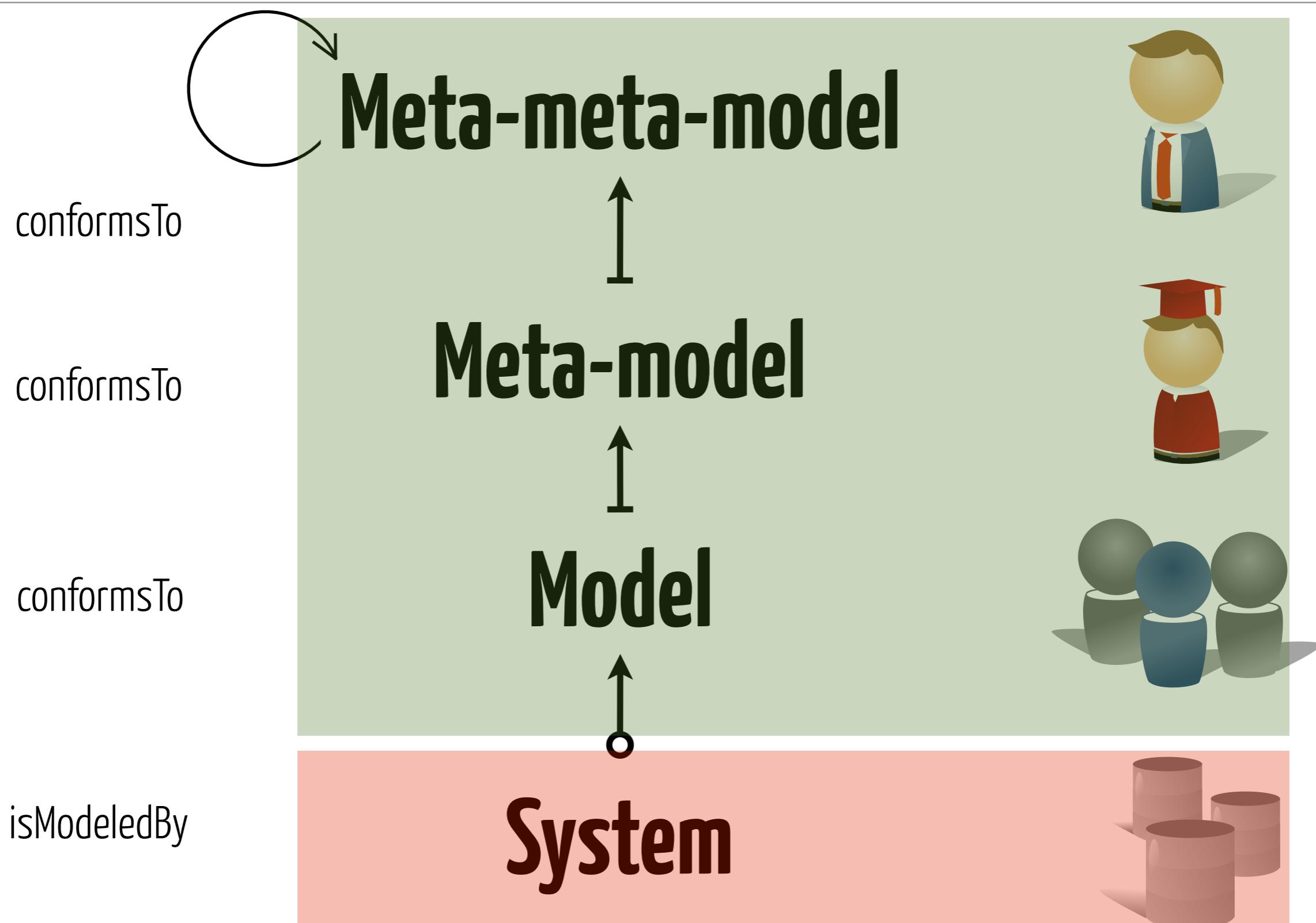
Actors & Artifacts



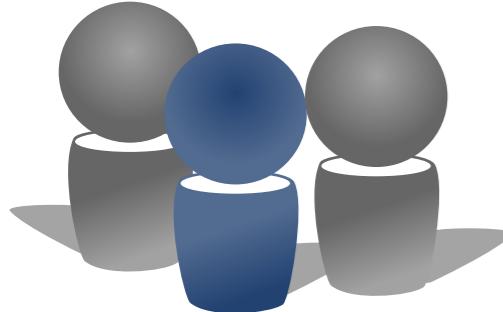
Capturing / Tooling Domain knowledge



The abstraction journey

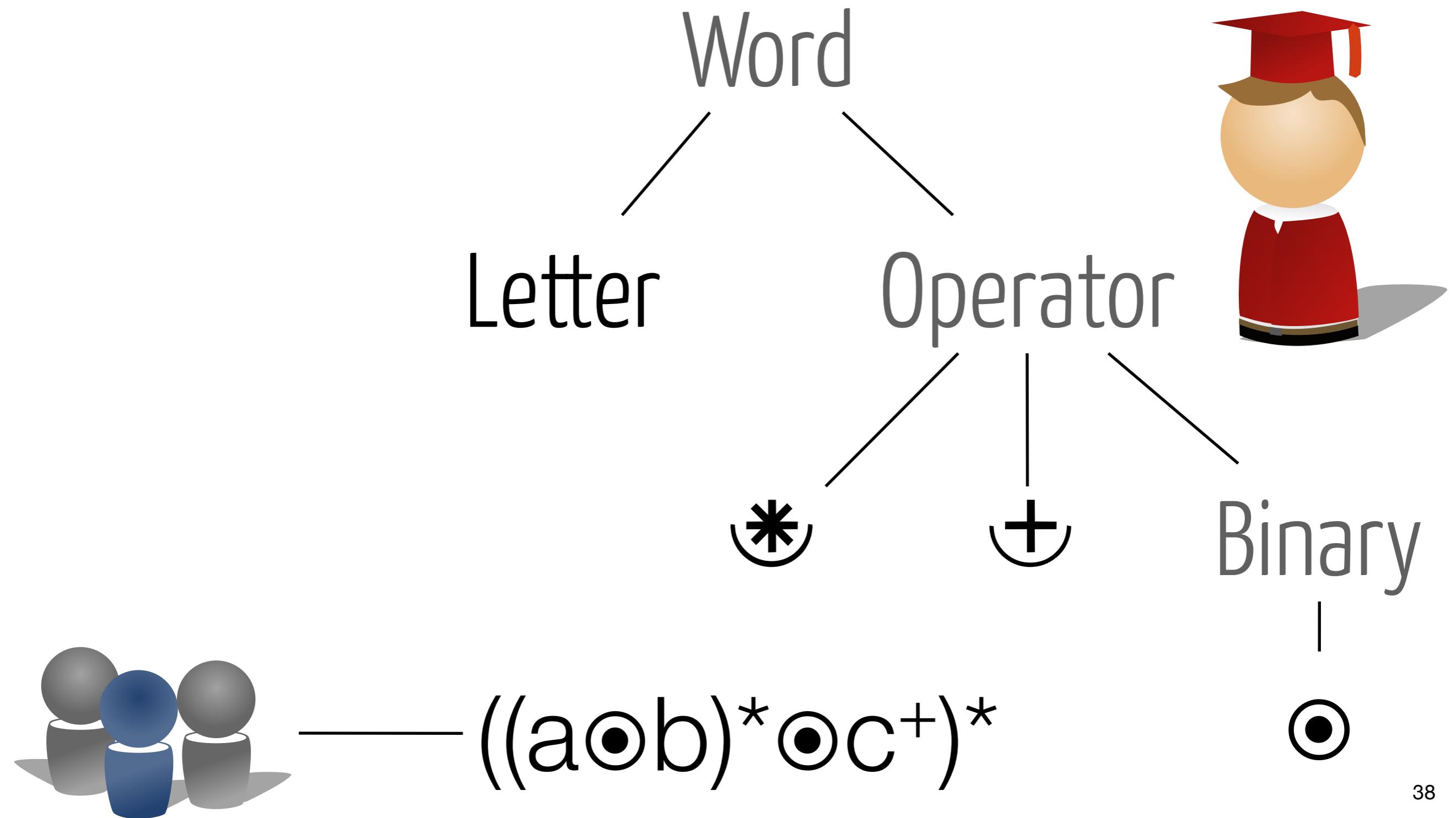


Example: Handling Regular Expressions



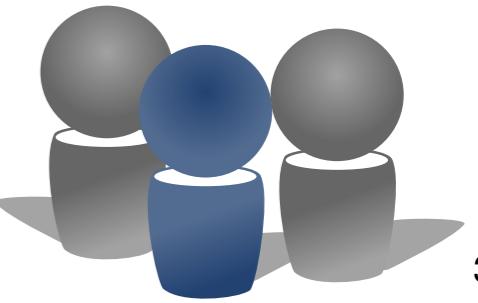
— ((aob)*oc+)*

Example: Handling Regular Expressions



Requirements #1: Modeling **concepts**

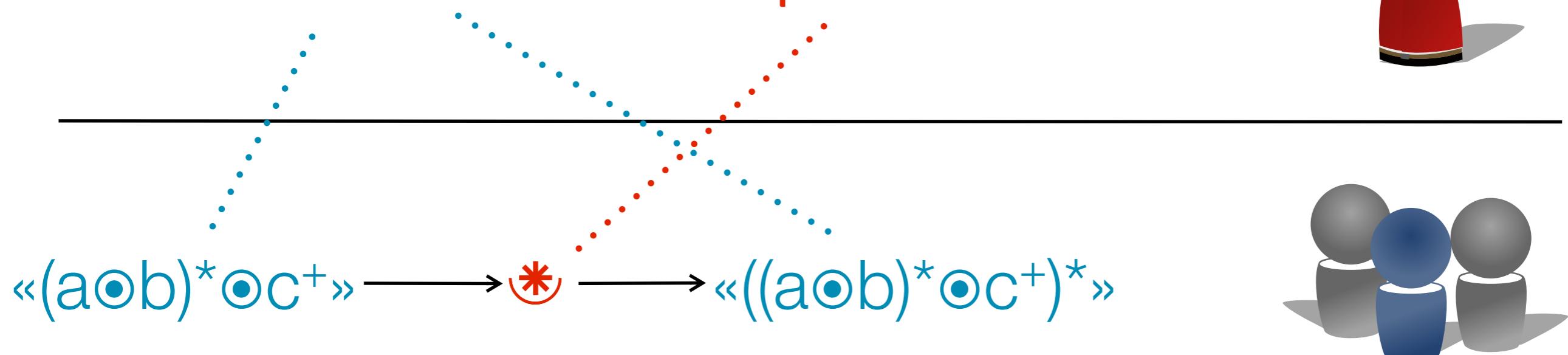
« $(a \odot b)^* \odot c^+$ » \longrightarrow  \longrightarrow « $((a \odot b)^* \odot c^+)^*$ »



Requirements #1: Modeling **concepts**

Word

Operator: *



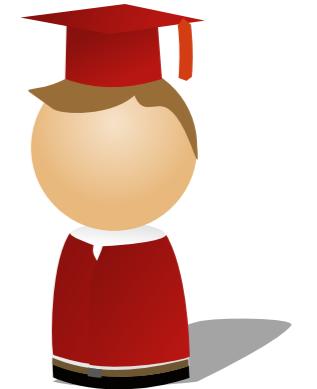
Requirements #1: Modeling **concepts**

Concept

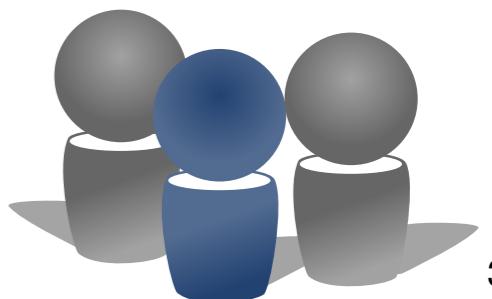


Word

Operator: *



«(a○b)*○c+» → * → «((a○b)*○c+)*»



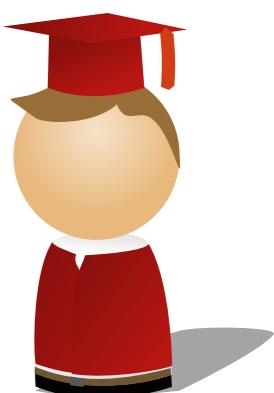
Starting at the beginning ...



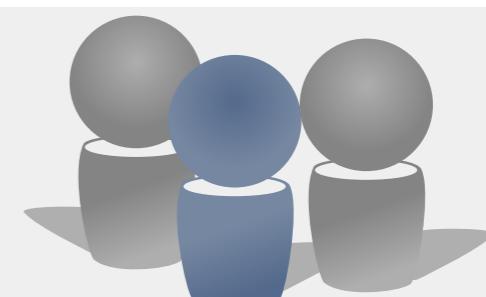
```
abstract public class NamedElement {  
    protected String name;  
}  
  
public class Concept extends NamedElement {  
    public Concept(String n) { this.name = n; }  
}
```



```
Concept word = new Concept("Word");  
Concept star = new Concept("KleenStar");
```



KleenStar(Word('A')) → A*

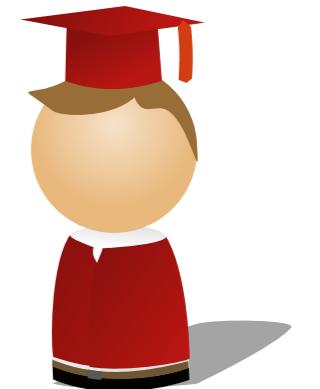


Requirements #2: Holding **Properties**

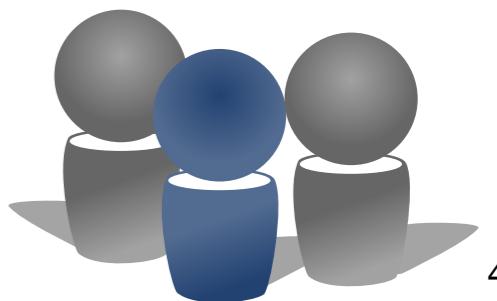
Concept



Letter



«a»



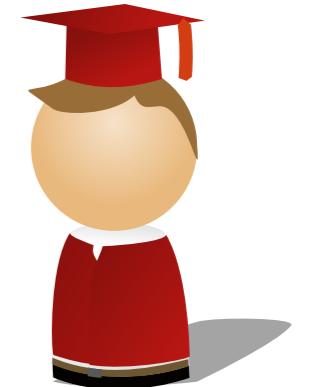
Requirements #2: Holding **Properties**

Concept



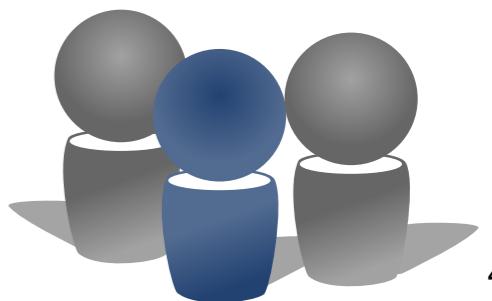
Letter

Char



«a»

' a '

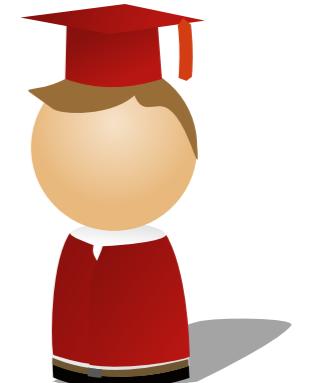


Requirements #2: Holding **Properties**

Concept • *contains:* → **Property**

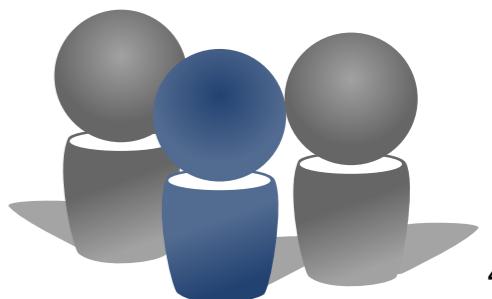


Letter • *symb: 1* → **Char**



«a»

' a '



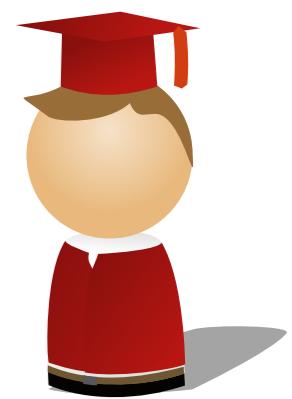


Implementing Properties

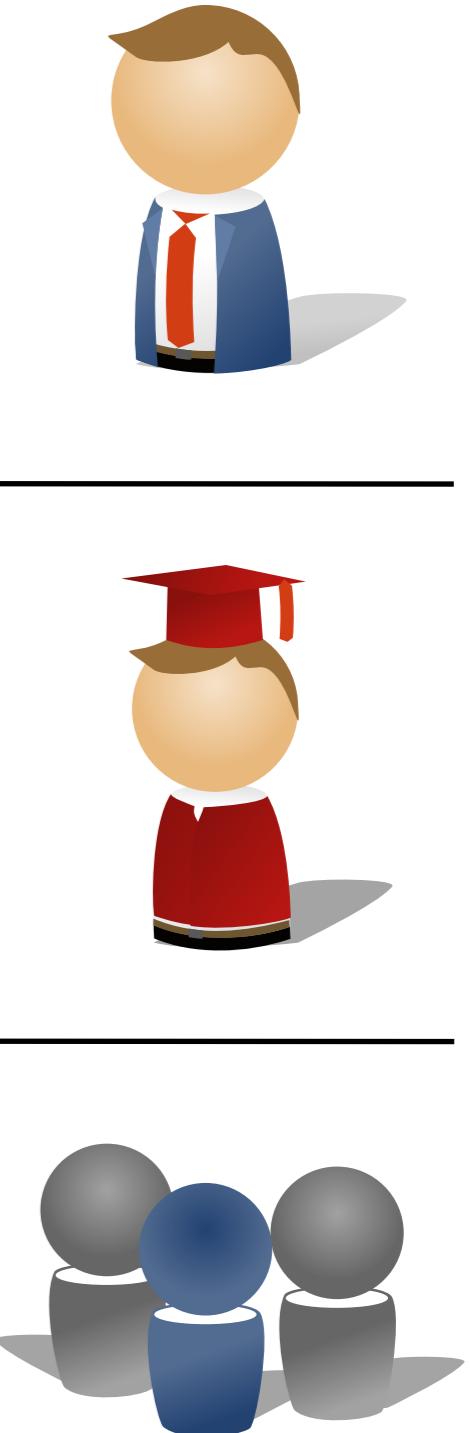
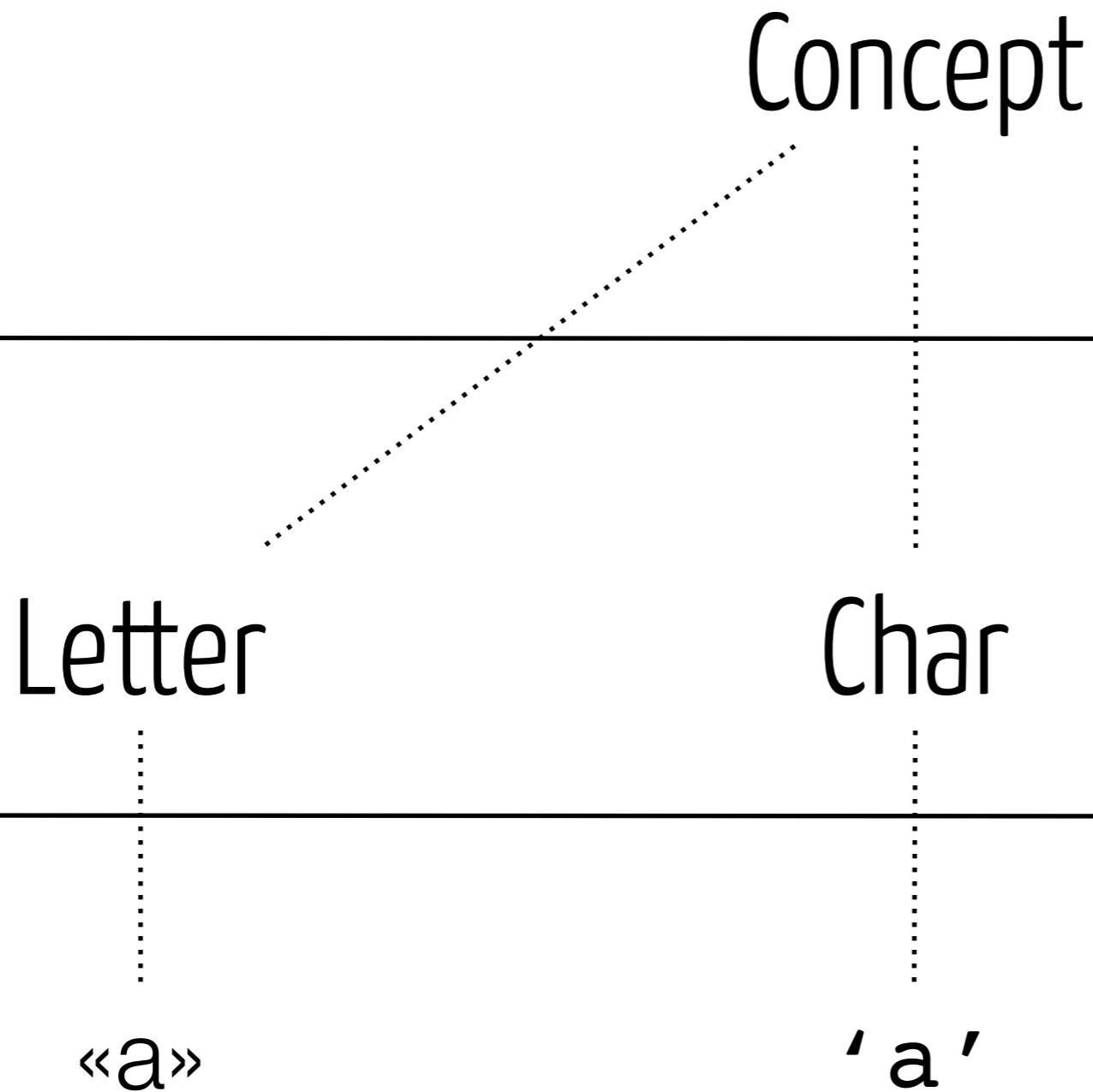
```
public class Property extends NamedElement {  
    private Concept kind; private int mult;  
  
    public Property(String n, Concept k, int m) {  
        this.name = n; this.kind = k; this.mult = m;  
    }  
}
```



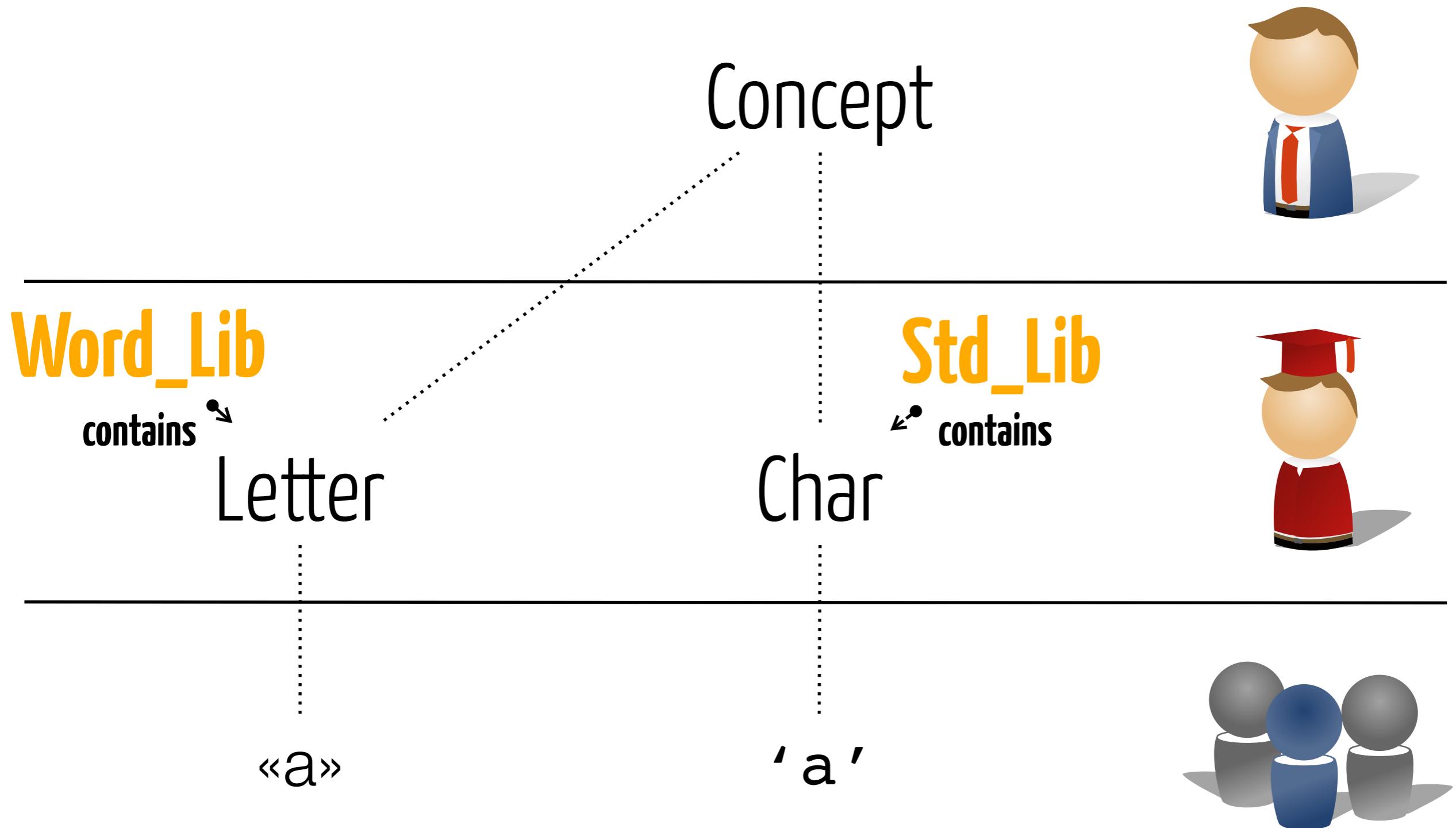
```
Concept ch = new Concept("Char");  
Concept letter = new Concept("Letter");  
letter.addProperty(new Property("symb", ch, 1));
```



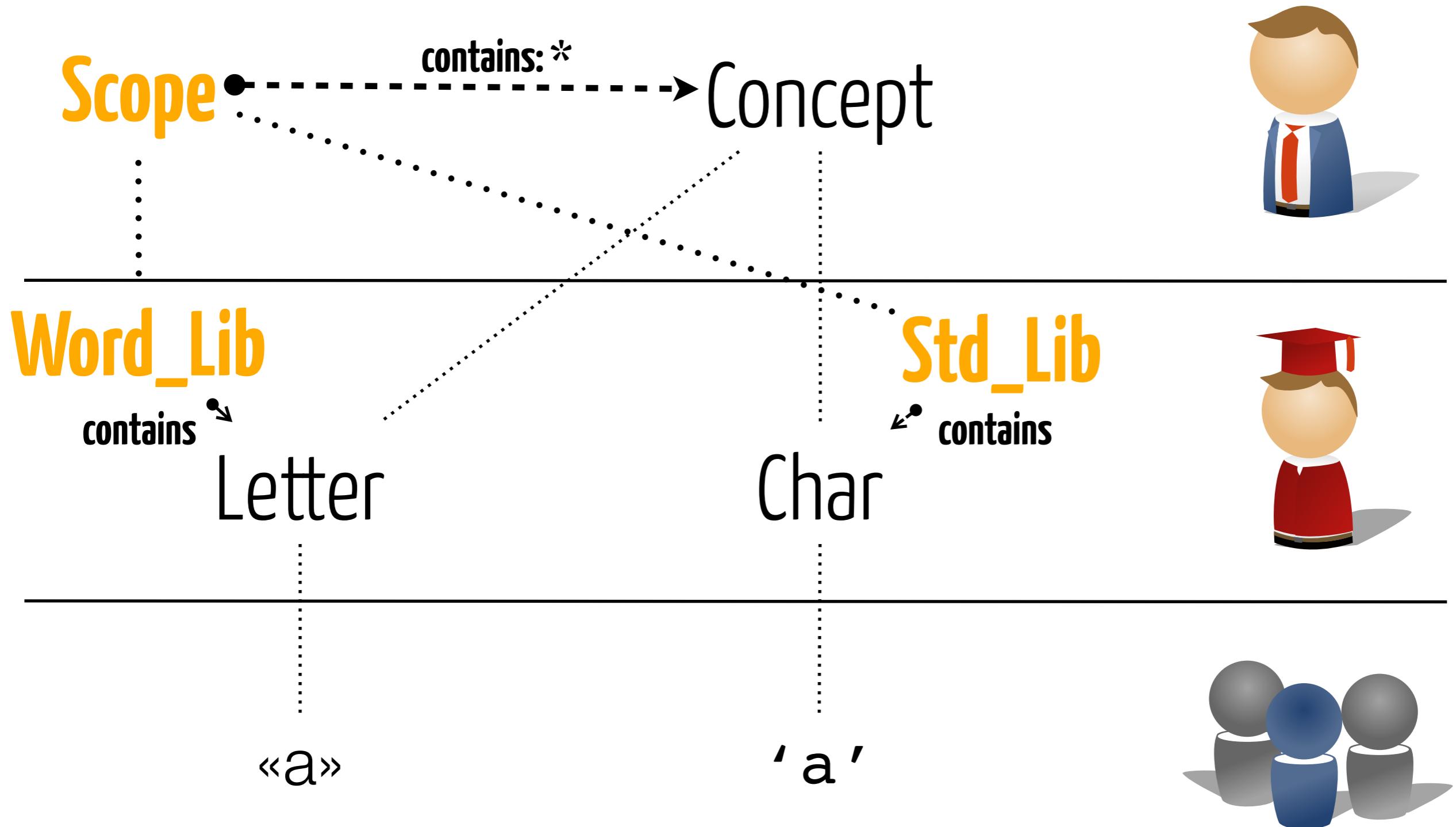
Requirements #3: Scoping Concepts



Requirements #3: Scoping Concepts



Requirements #3: Scoping Concepts



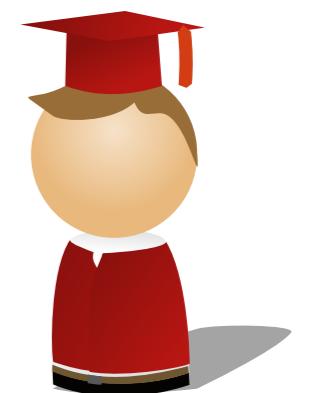


Implementing Concept Scoping

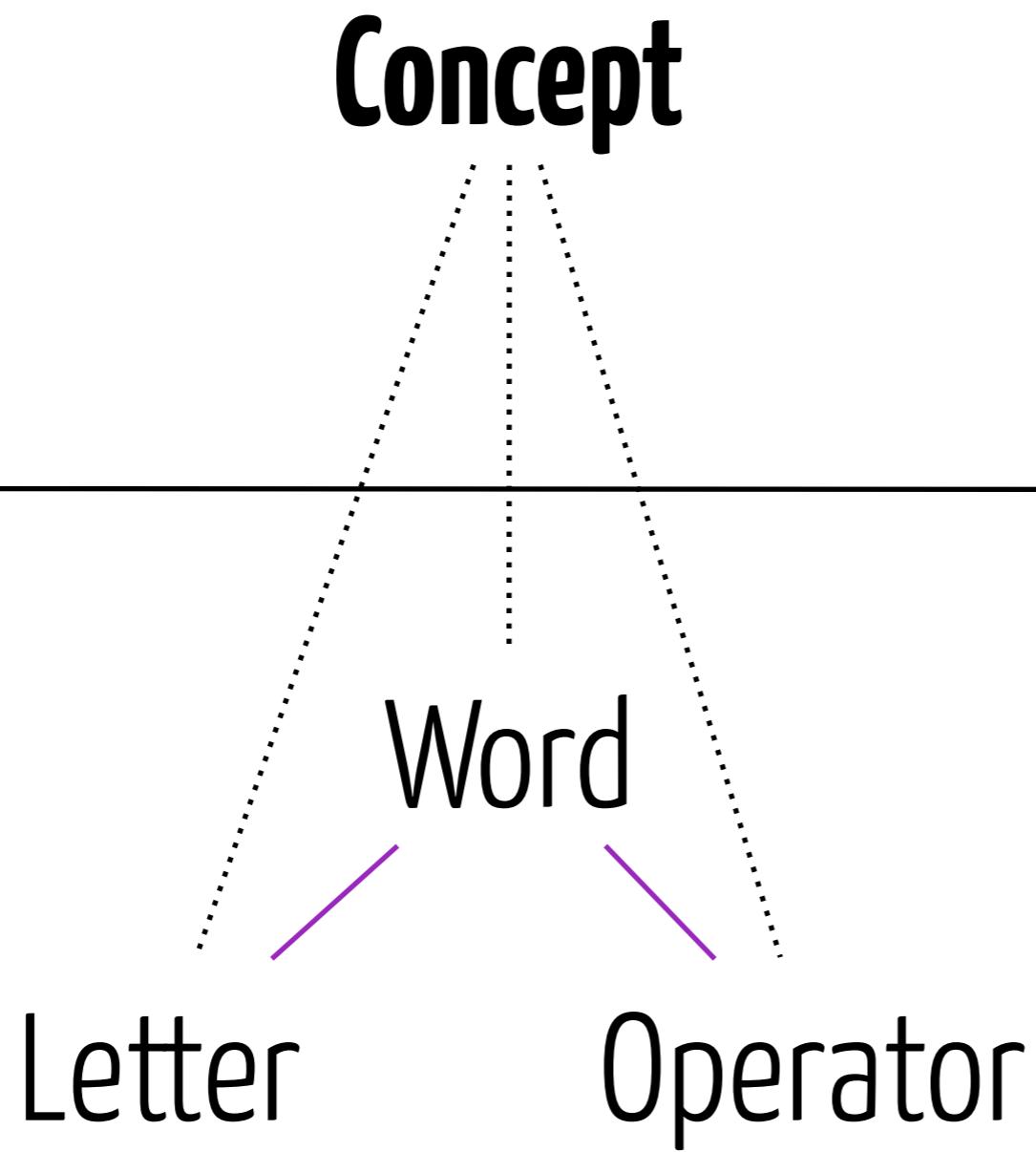
```
public class Scope extends NamedElement {  
    private ArrayList<Concept> concepts;  
  
    public Scope(String n) {  
        this.name = n;  
        this.concepts = new ArrayList<Concept>();  
    }  
}
```



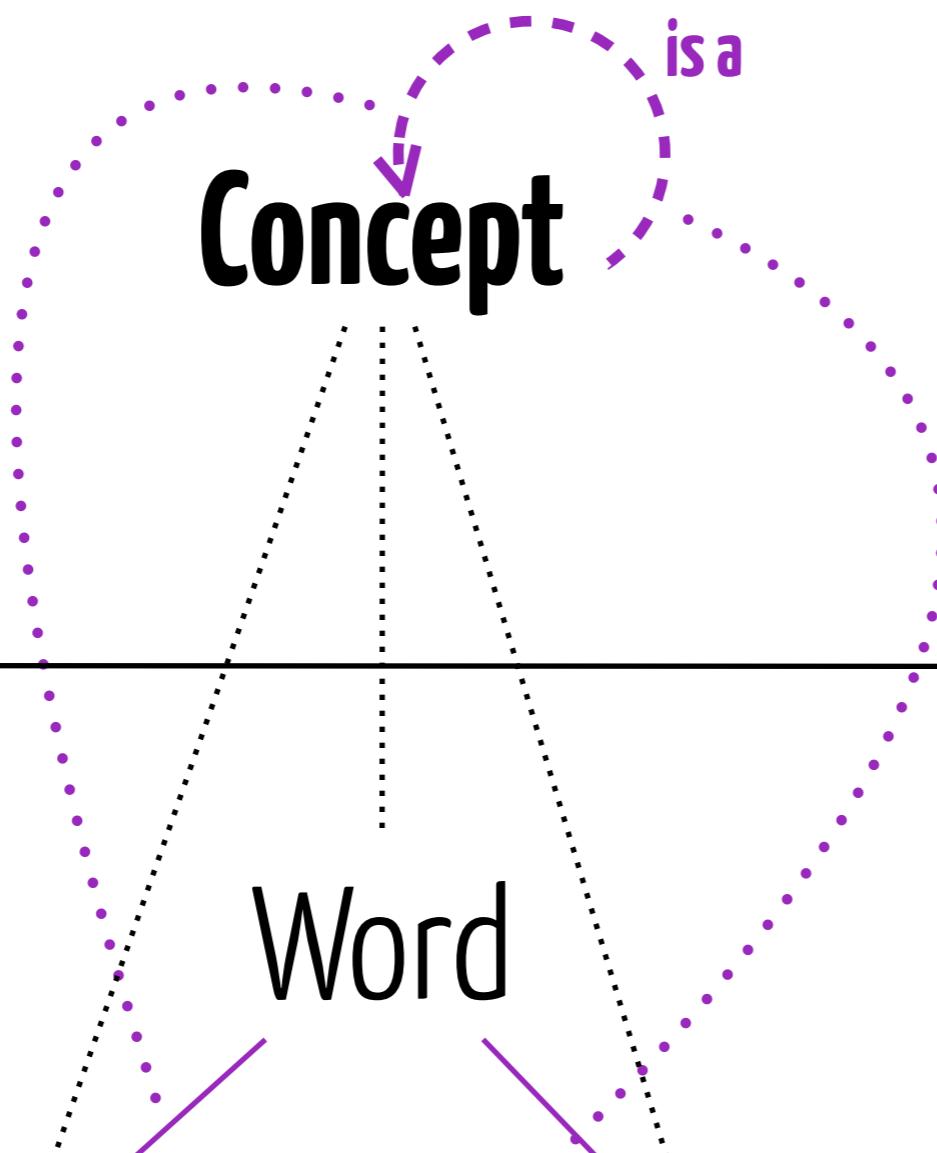
```
Scope words = new Scope("word_lib");  
Scope std = new Scope("std_lib");  
std.addConcept(char);  
words.addConcept(letter);
```



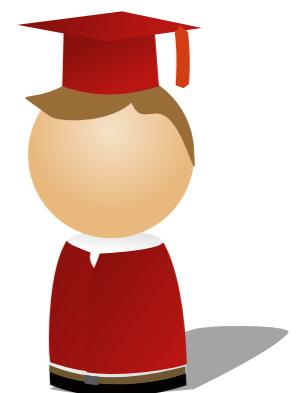
Requirements #4: Categorizing Concepts



Requirements #4: Categorizing Concepts



Letter Operator



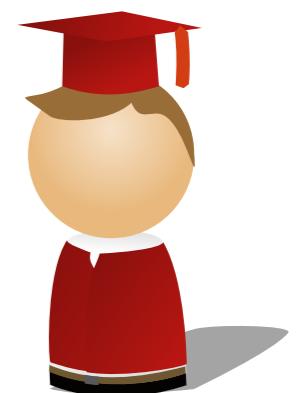


Implementing Relations between Concepts

```
public class Concept extends NamedElement {  
    private Concept isA;  
    // ...  
    public void refines(Concept parent) {  
        this.isA = parent;  
    }  
}
```



```
Concept word      = new Concept("Word");  
Concept letter    = new Concept("Letter");  
Concept operator  = new Concept("Operator");  
letter.refines(word);  
operator.refines(word);
```



Requirements #5: Containment ≠ Reference

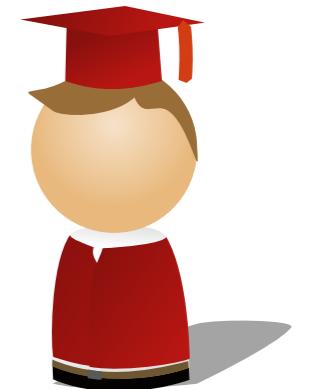
Concept



Char

Letter

Operator: ◎



'a'

'b'

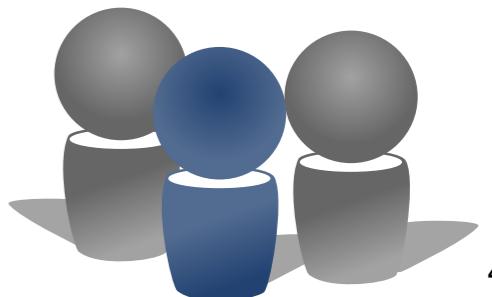
«

a

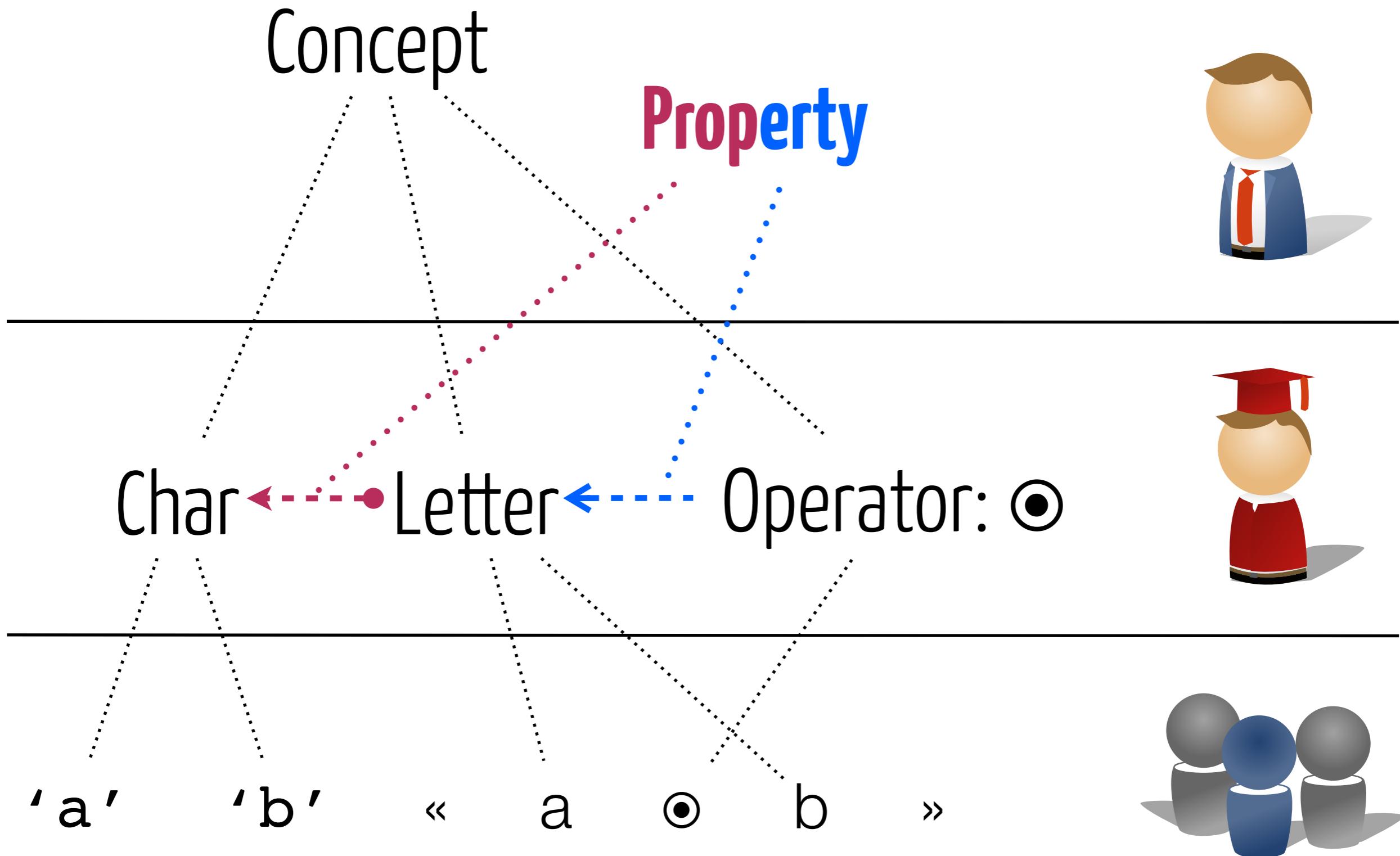
◎

b

»



Requirements #5: Containment ≠ Reference





Containment = property of «Property»

```
public class Property extends NamedElement {  
    // ...  
    private bool contained = false;  
    public void setContained(bool b) {  
        this.contained = b;  
    }  
}
```

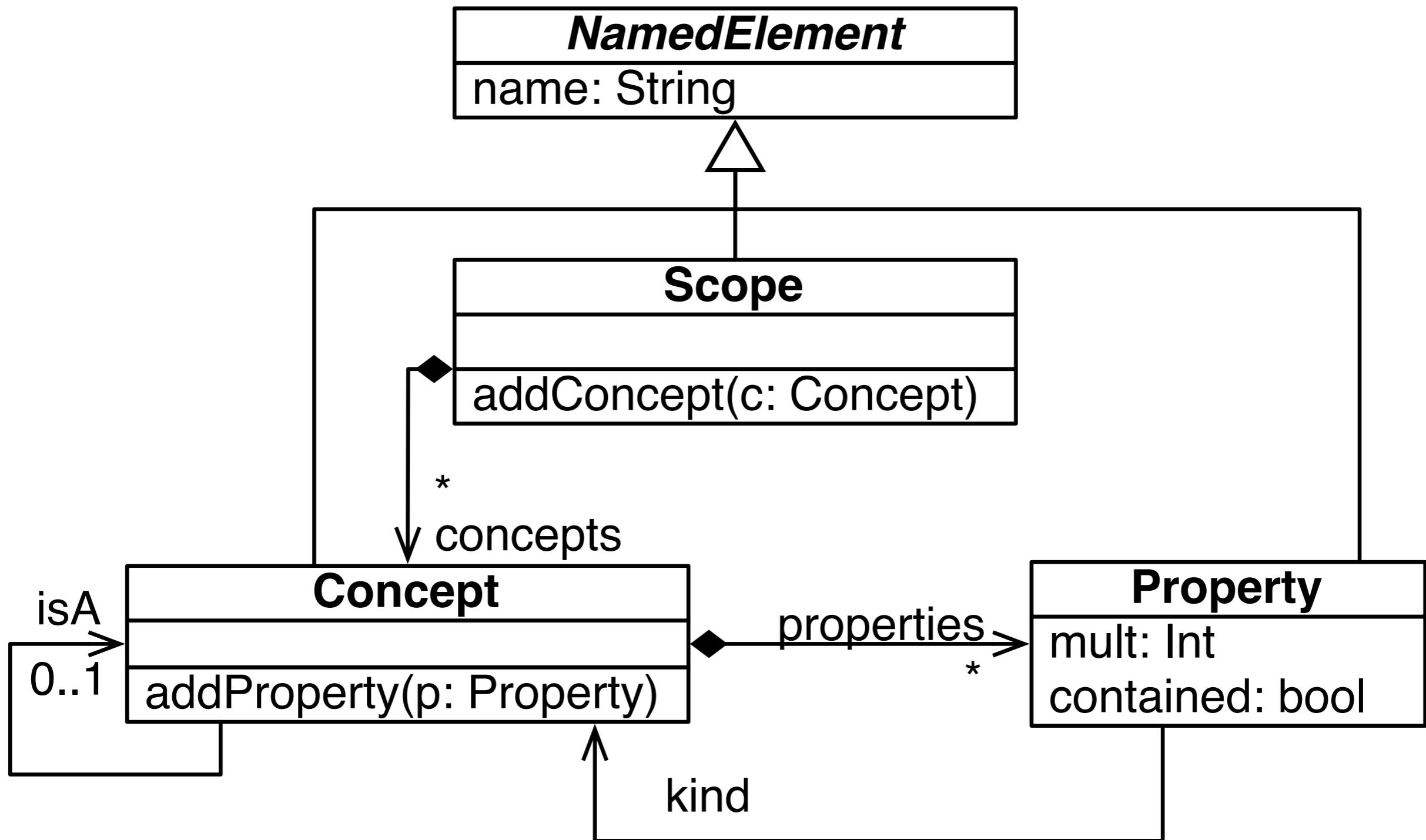


```
Property letSymb = new Property("symb", ch, 1);  
symb.setContained(true);  
letter.addProperty(letSymb);
```

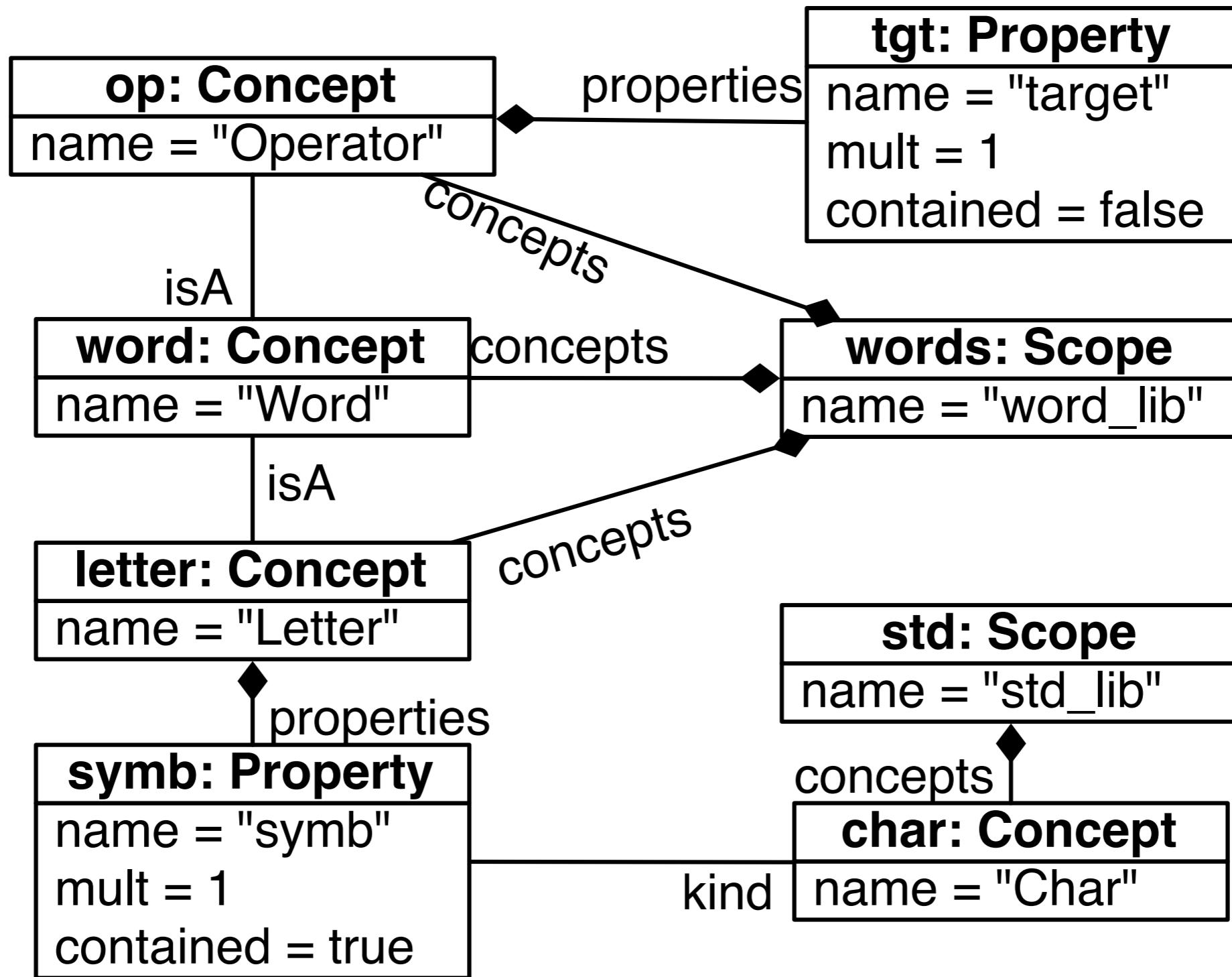


```
Property opWord = new Property("word", word, -1);  
operator.addProperty(opWord);
```

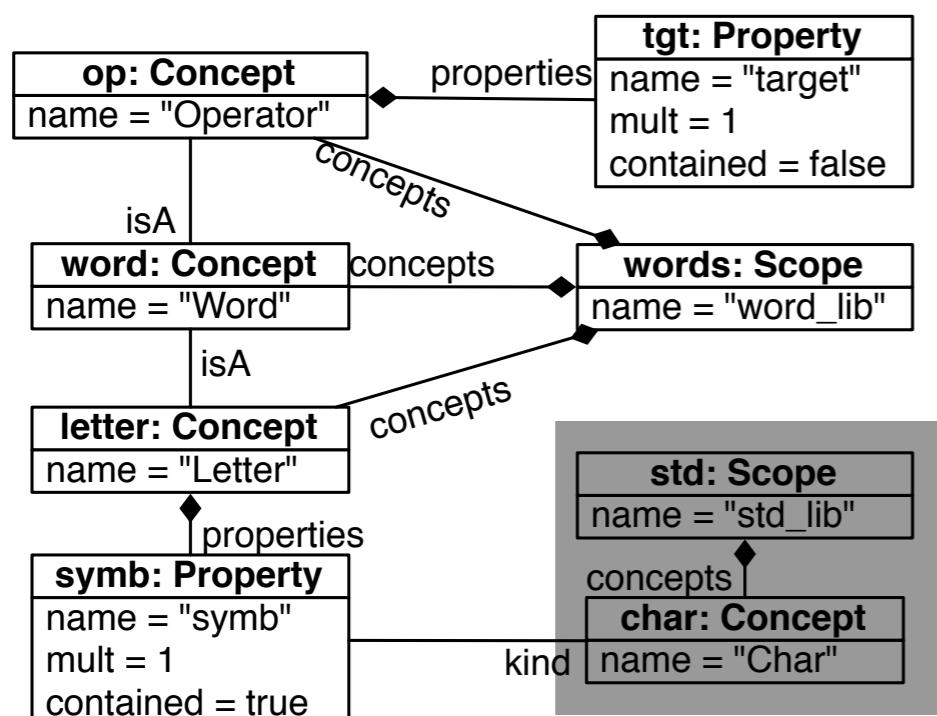
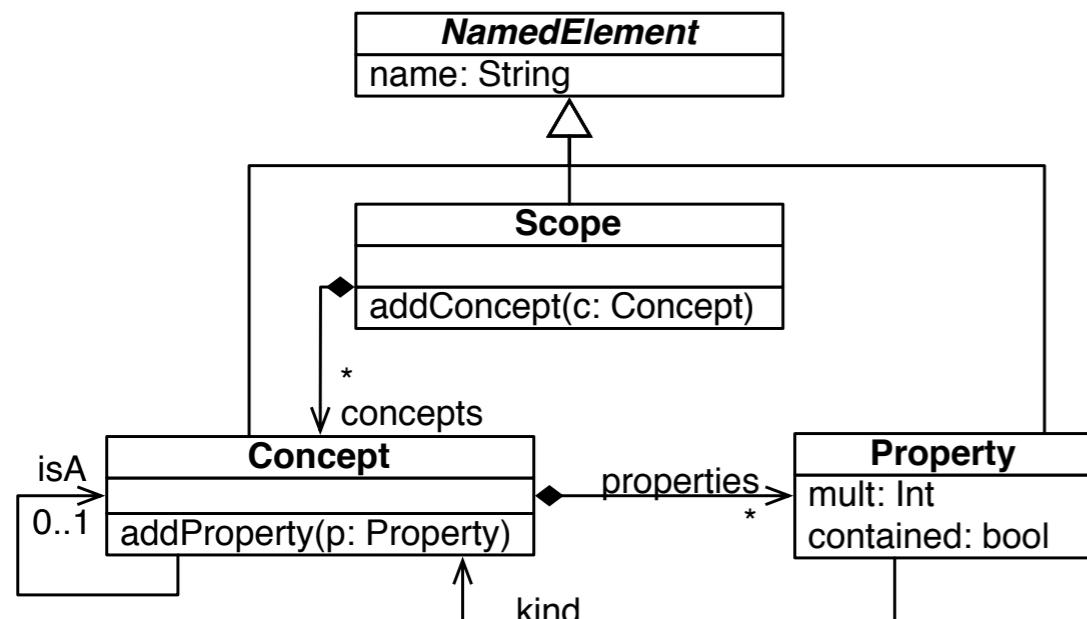
Meta-meta-model Summary (UML Class Diag.)



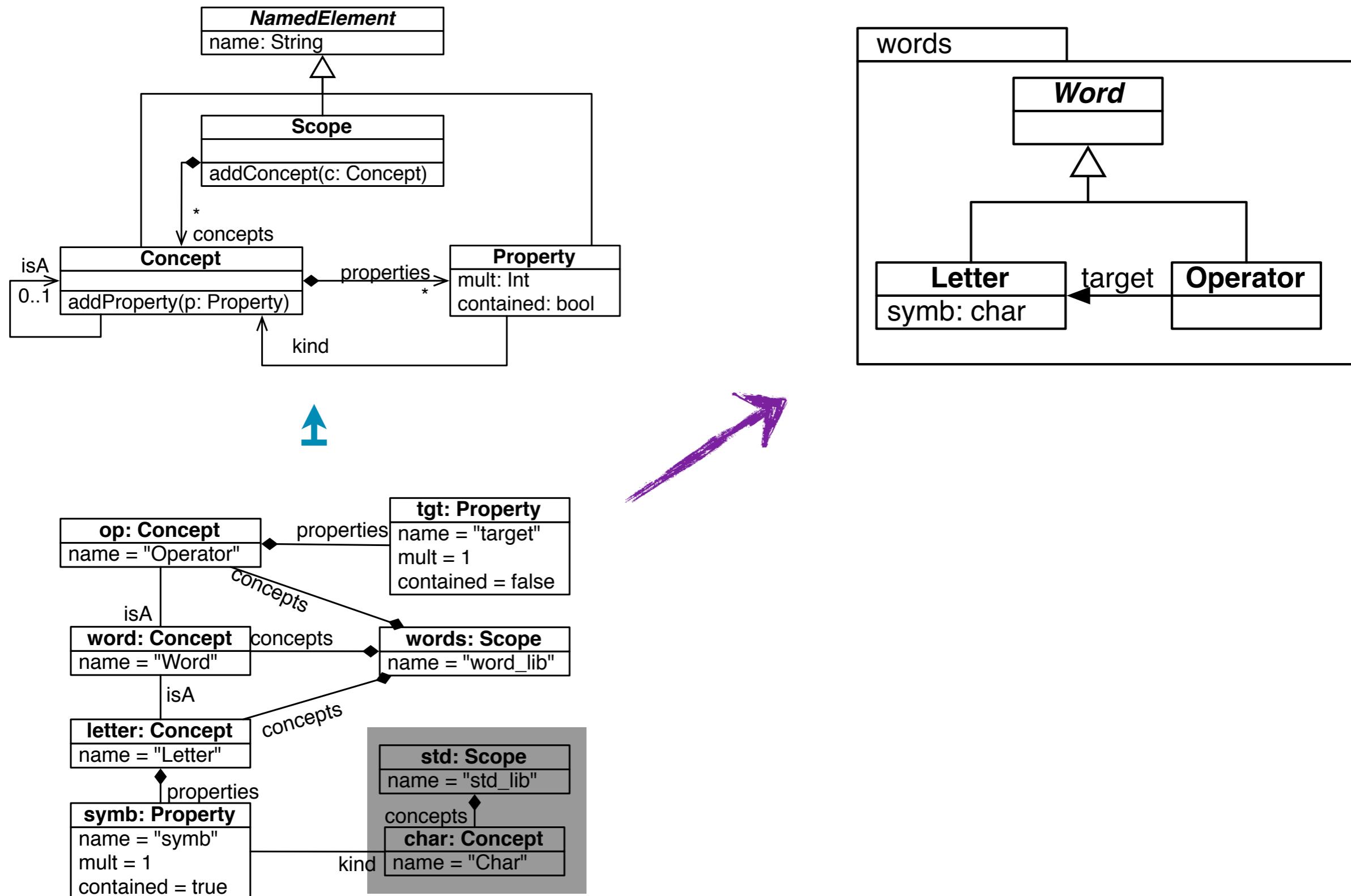
Meta-model Summary (UML Object Diag.)



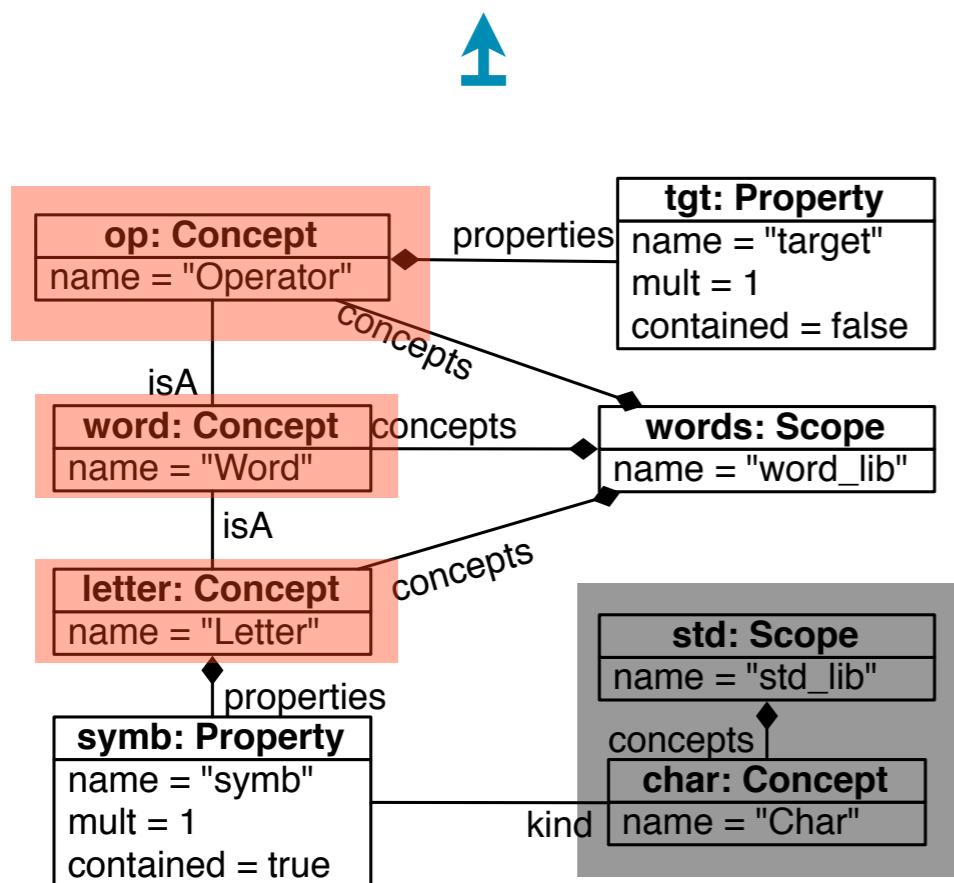
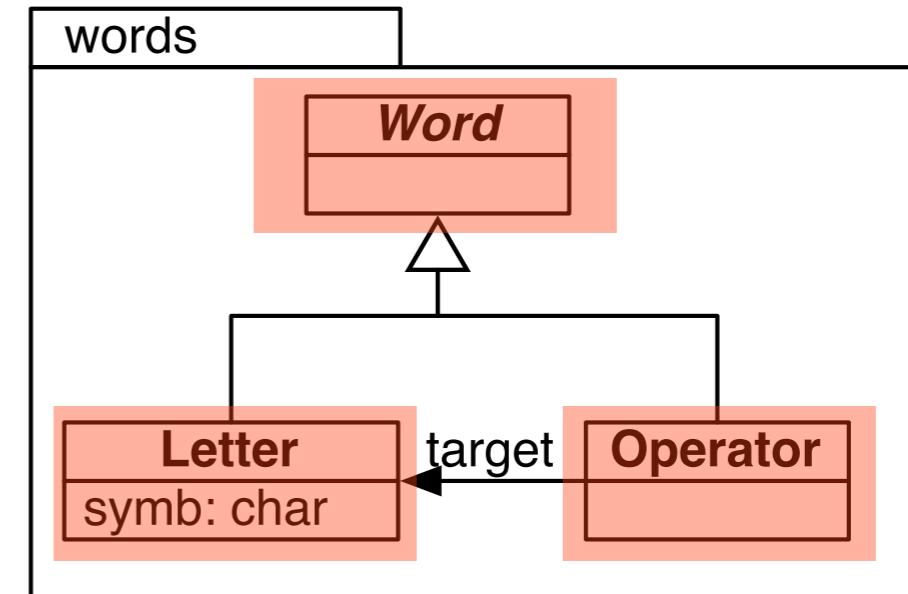
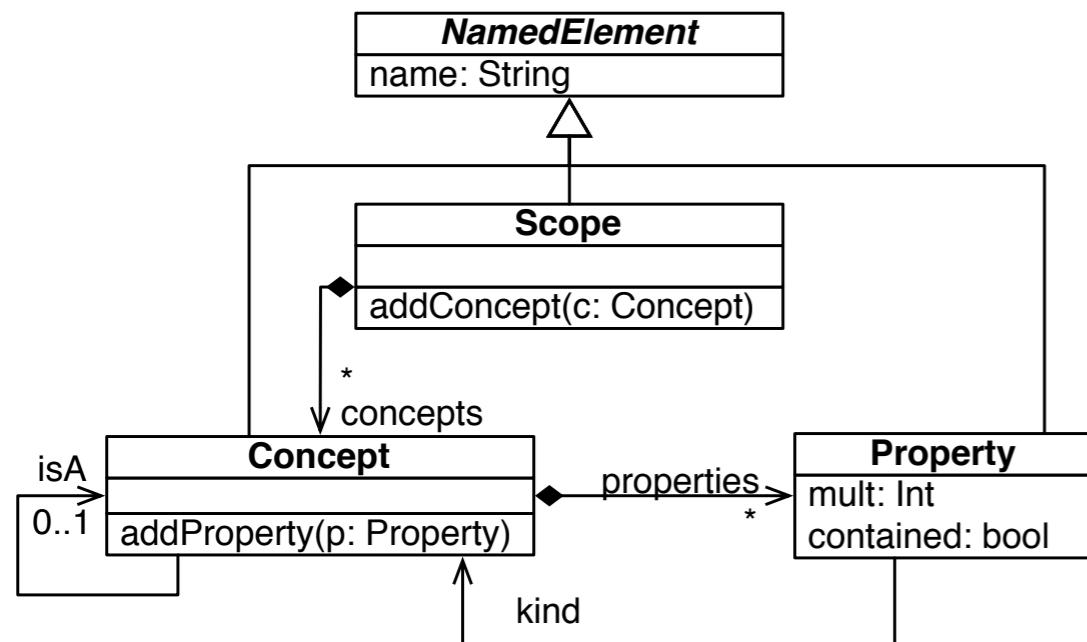
From Meta-models to Object-Oriented Classes



From Meta-models to Object-Oriented Classes



From Meta-models to Object-Oriented Classes

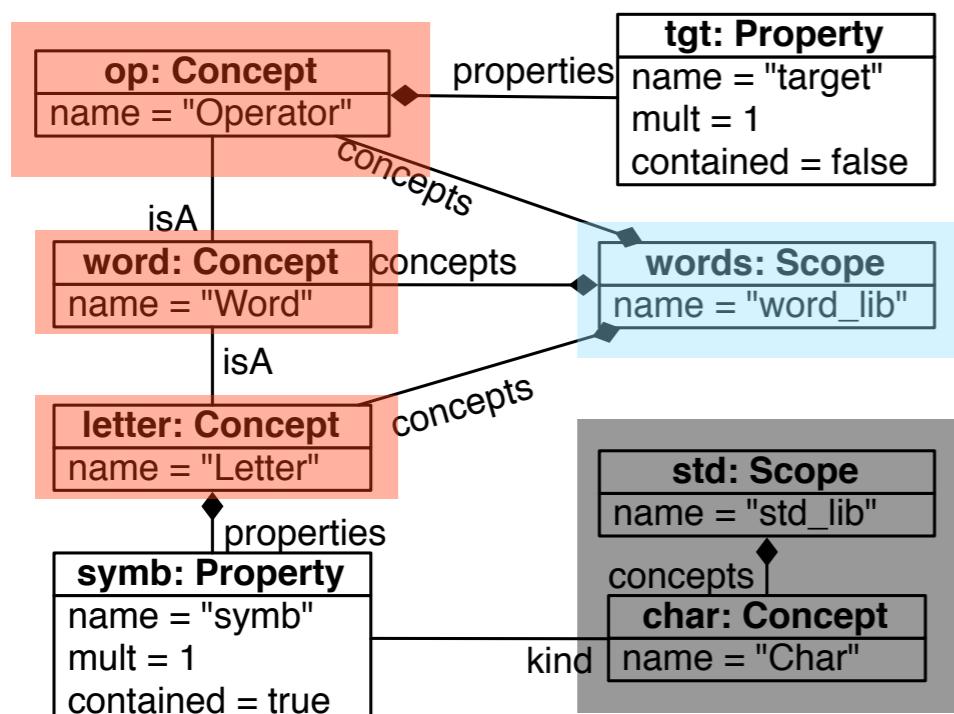
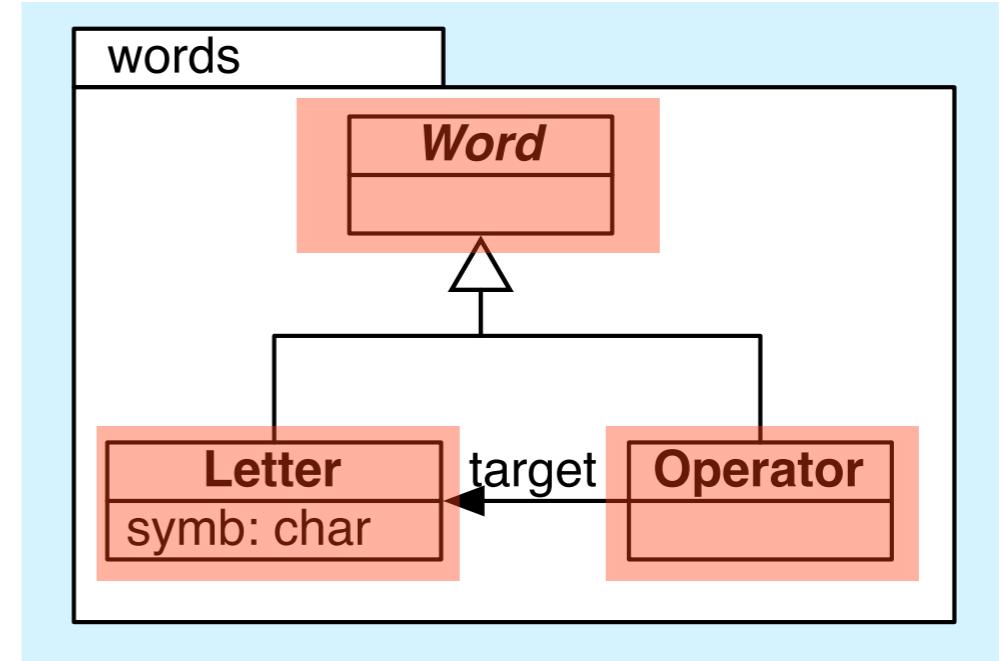
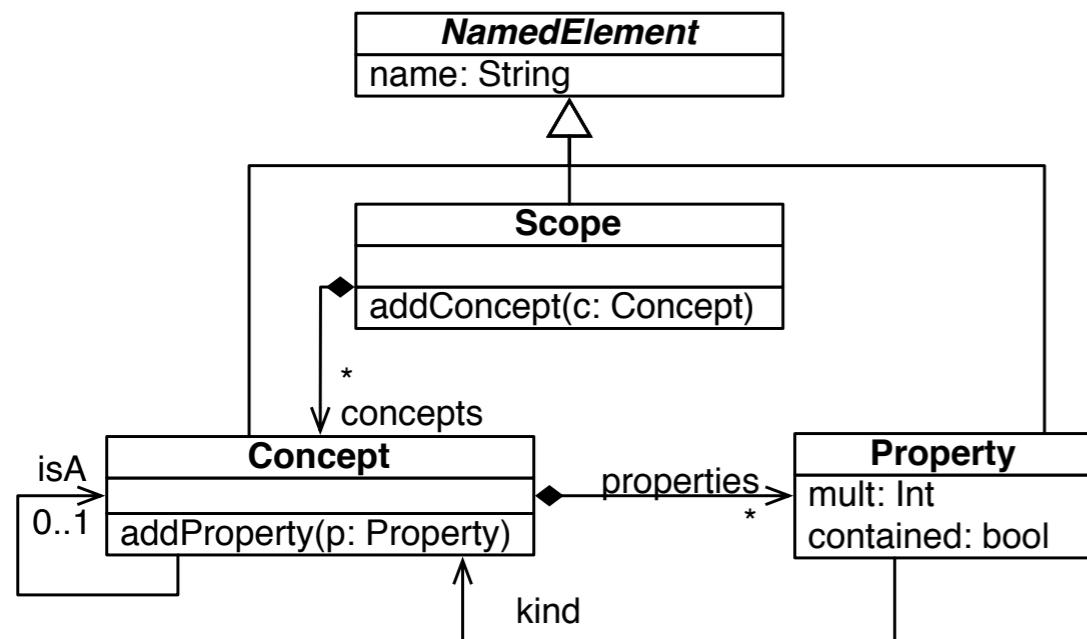


↑

Concept → Class

51

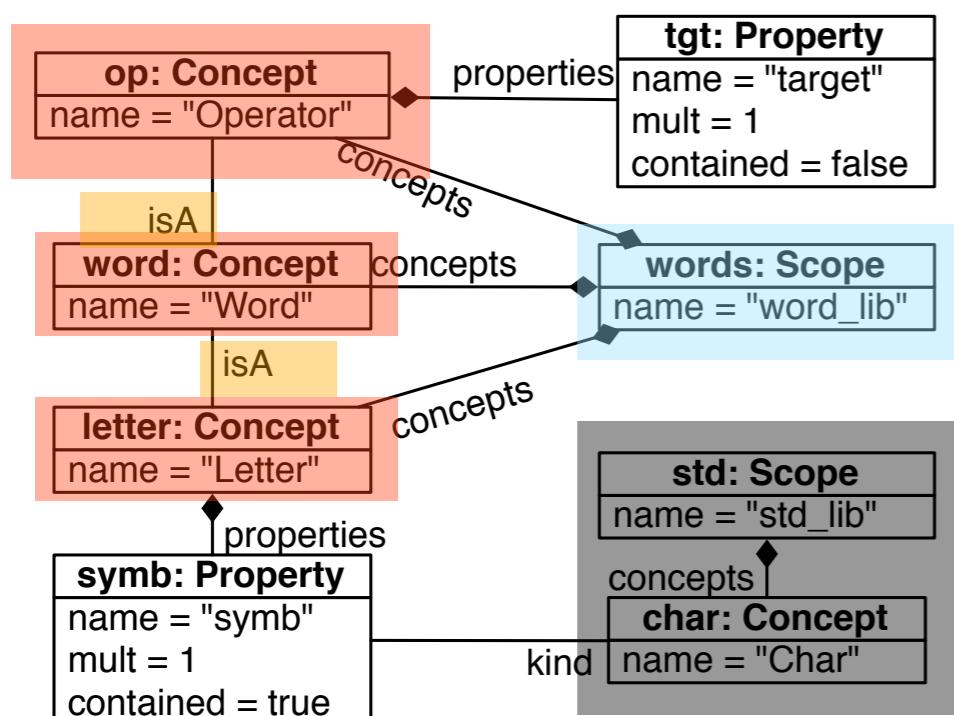
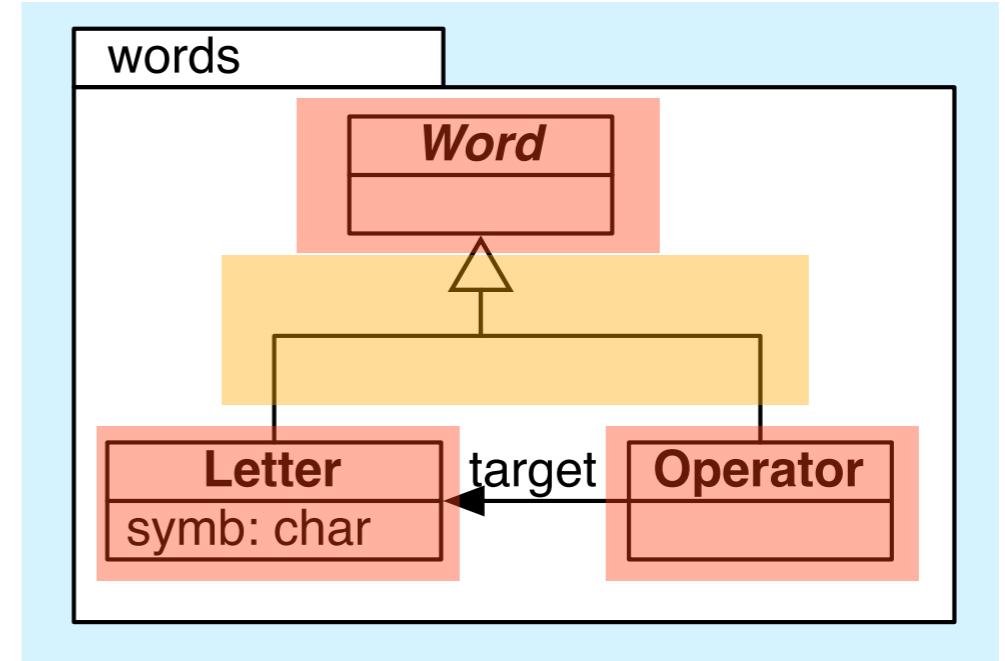
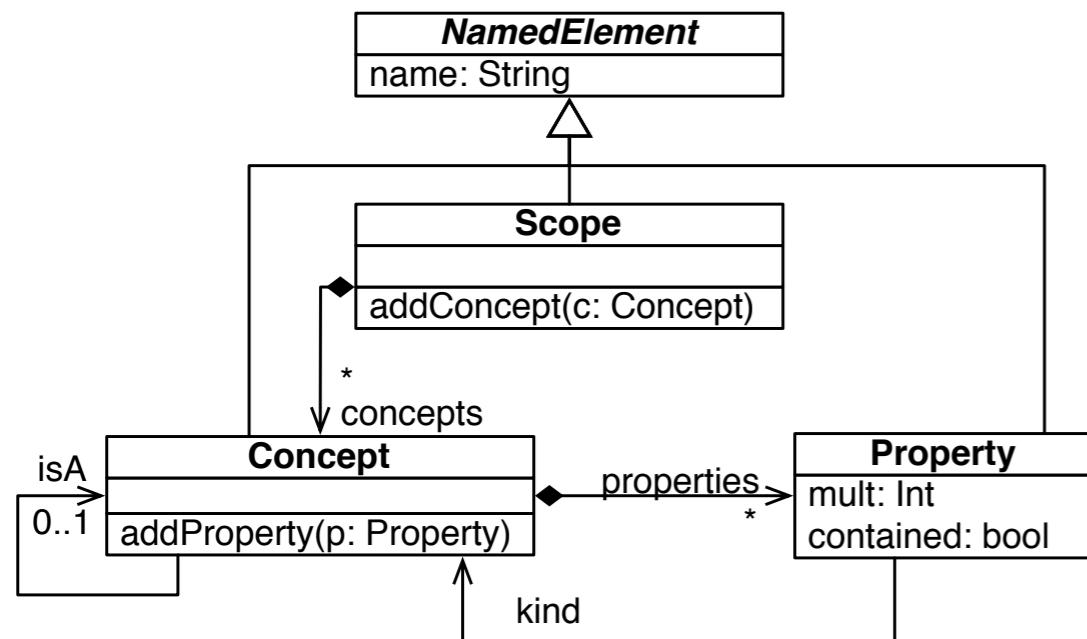
From Meta-models to Object-Oriented Classes



Concept → Class

Scope → Package

From Meta-models to Object-Oriented Classes

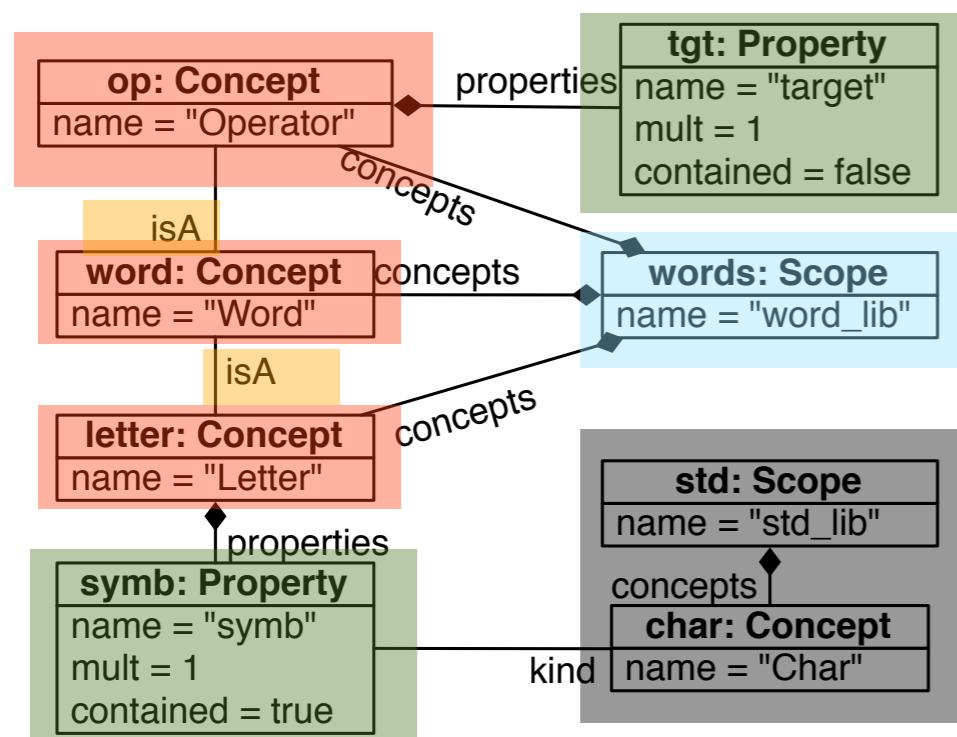
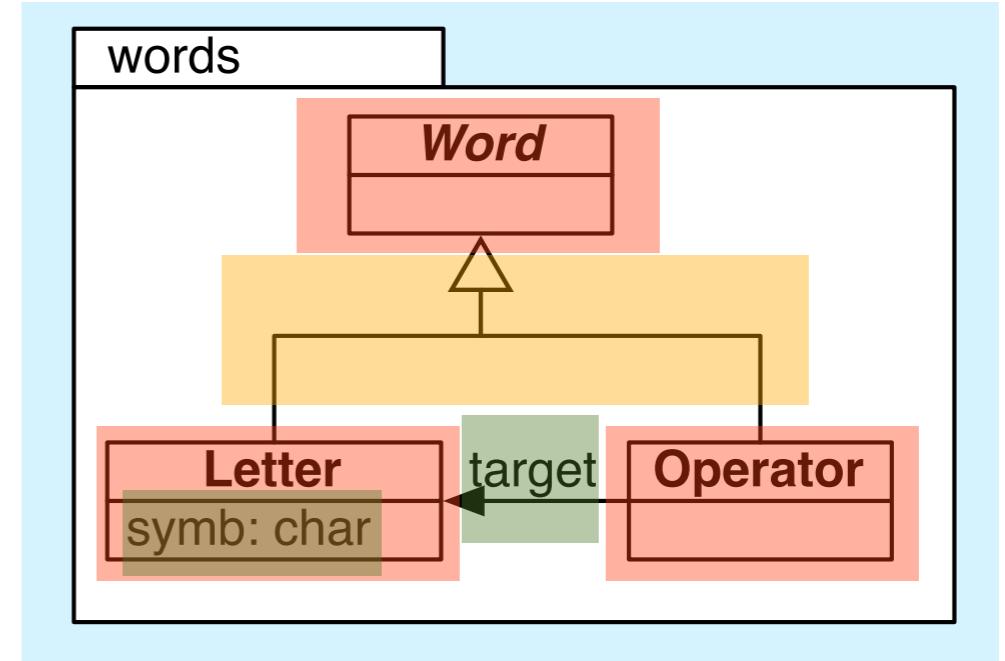
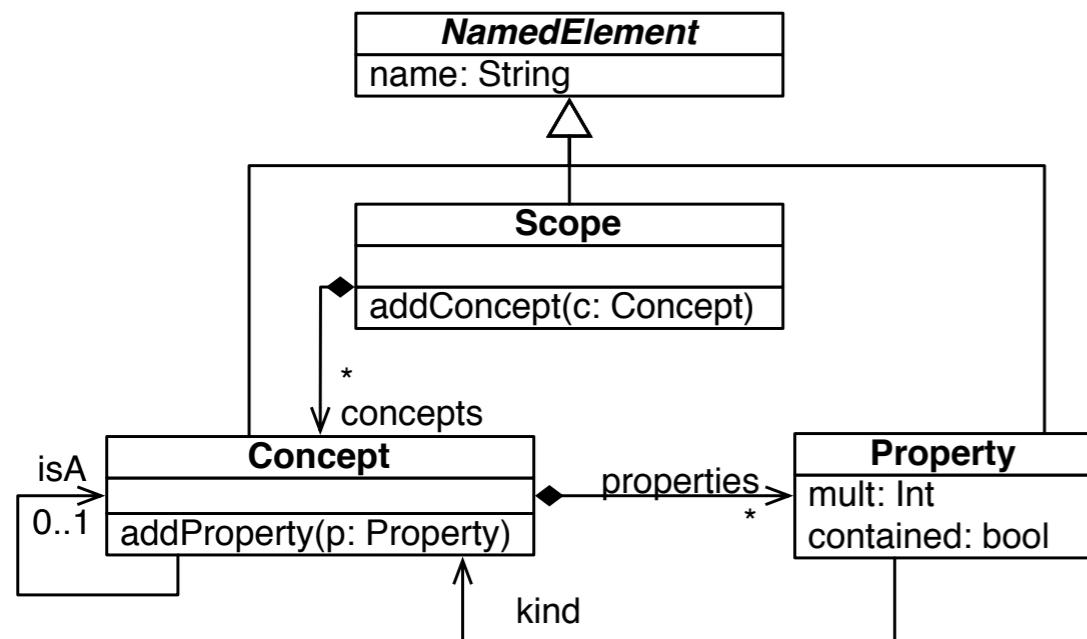


Concept → Class

Scope → Package

isA → Inheritance

From Meta-models to Object-Oriented Classes



Concept → Class

Scope → Package

isA → Inheritance

Property → Attribute

Bootstrapping a Modelling Language



**A sentence contains a subject,
a verb, and a complement.**



«conforms to»

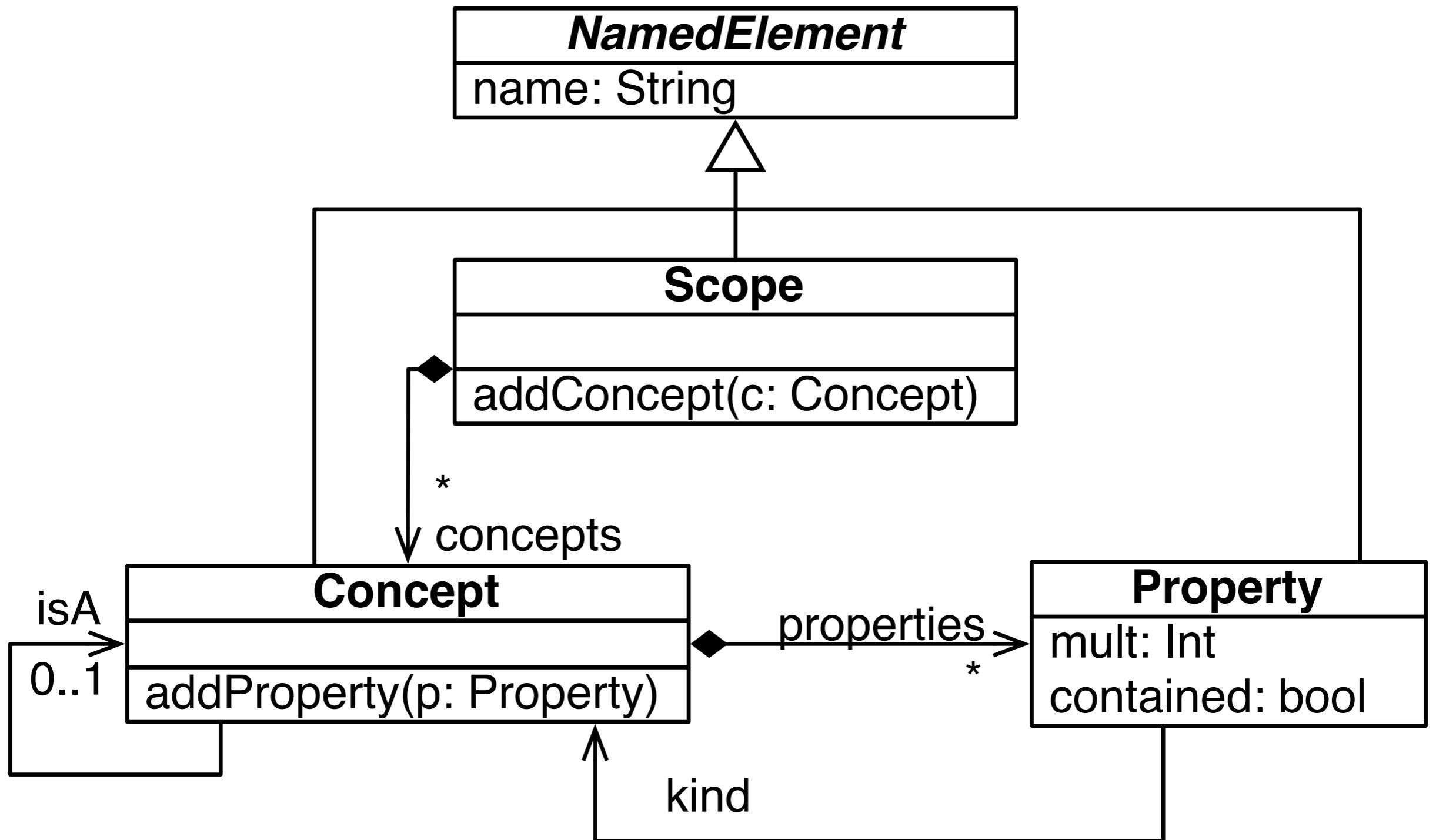
A sentence contains a subject,
a verb, and a complement.



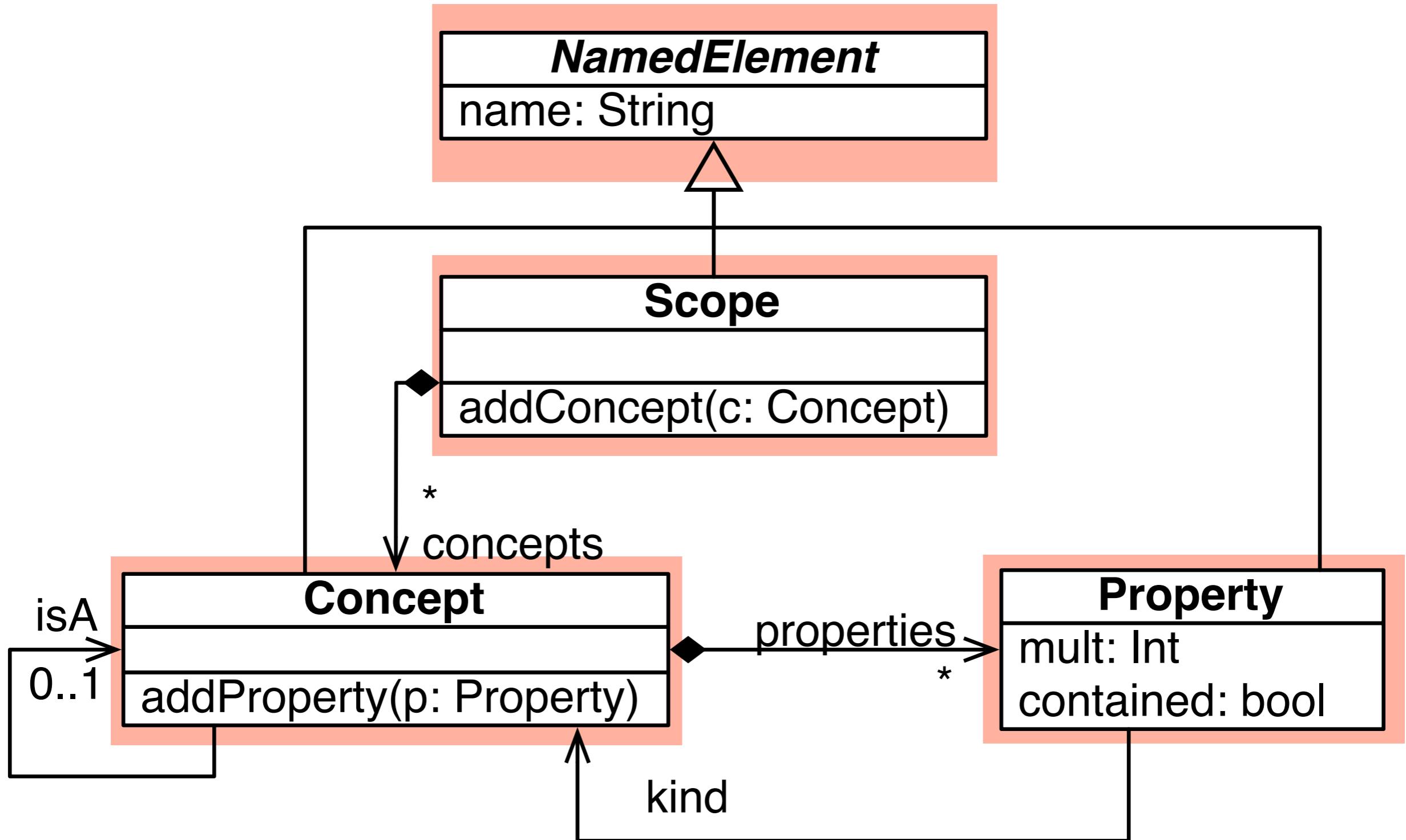
«conforms to»

I love model-driven engineering

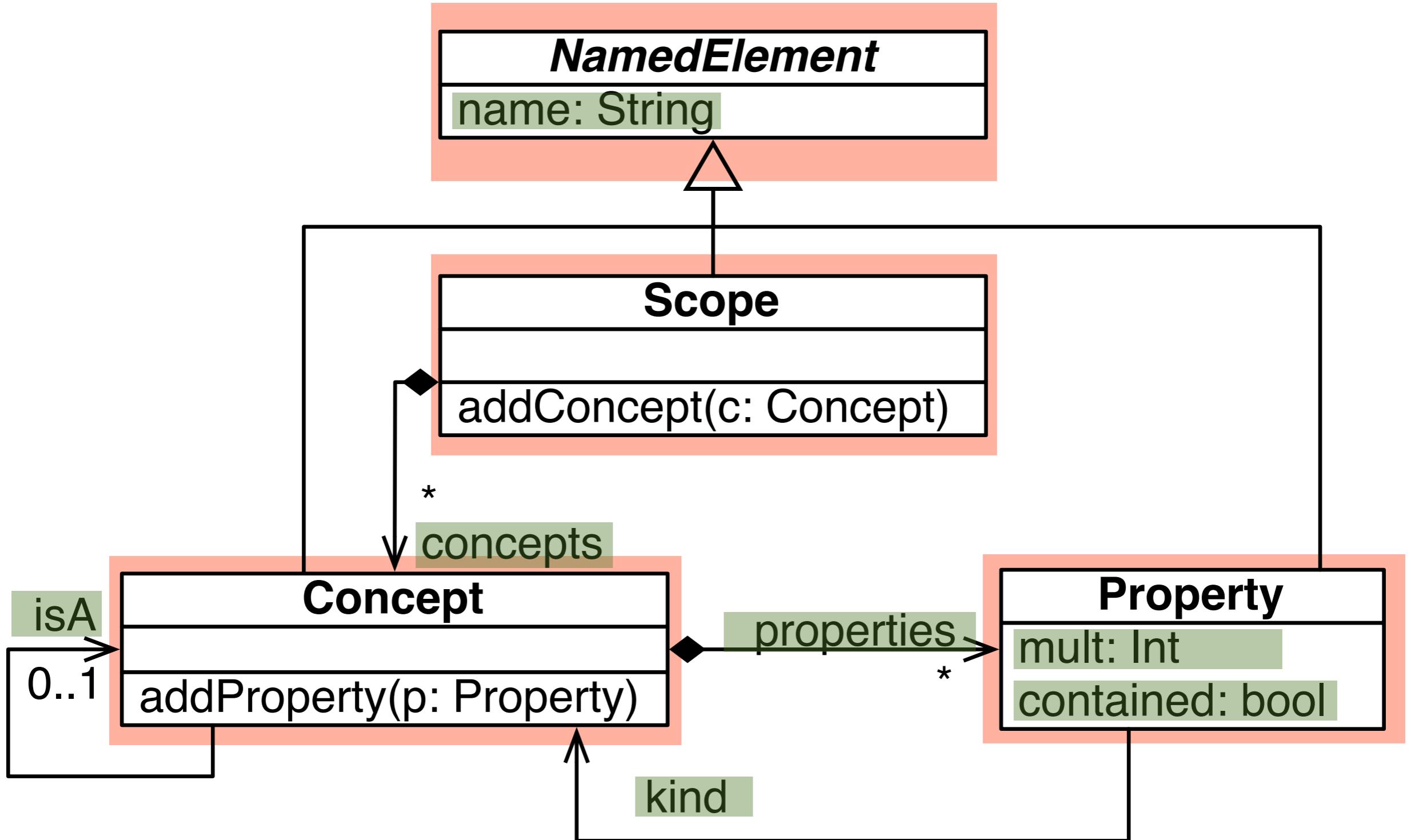
Meta-meta-model bootstrap



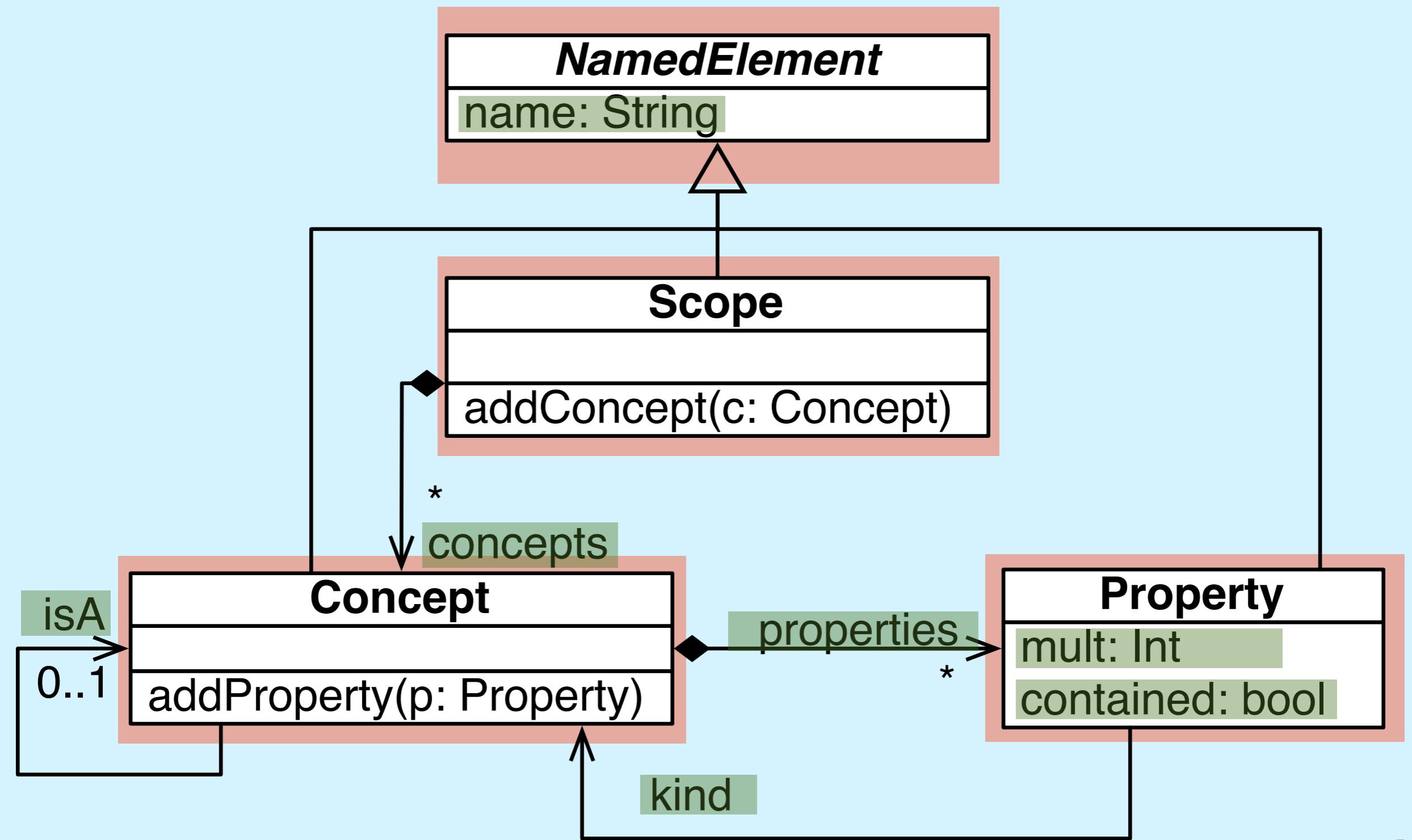
Meta-meta-model bootstrap



Meta-meta-model bootstrap



Meta-meta-model bootstrap



Step #0: Using UML to **bootstrap** (minimal)

**Concept
name: String**



Step #0: Using UML to **bootstrap** (minimal)

Concept
name: String



: Concept
name = "Concept"

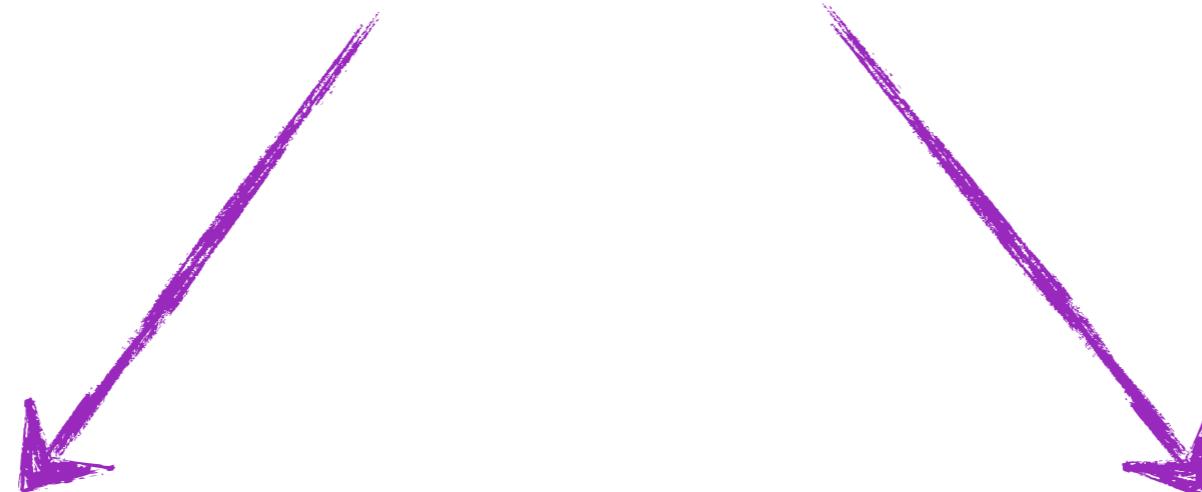
: Concept
name = "Property"



Code Generation

: Concept
name = "Concept"

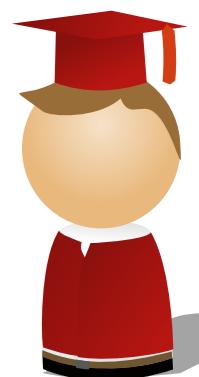
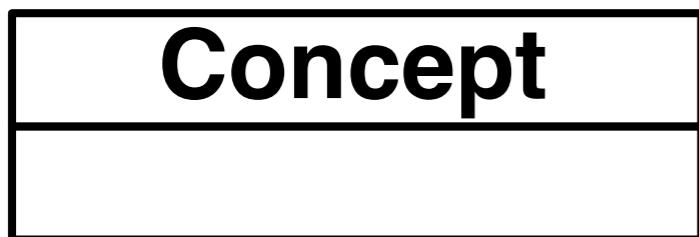
: Concept
name = "Property"



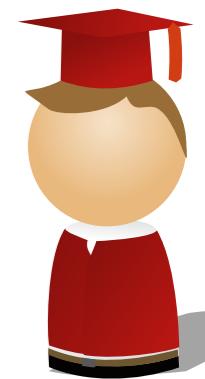
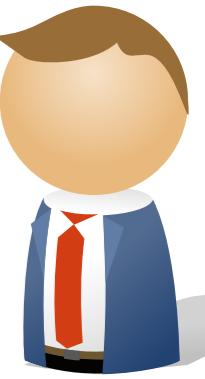
Concept

Property

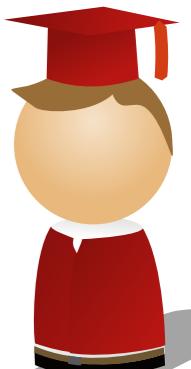
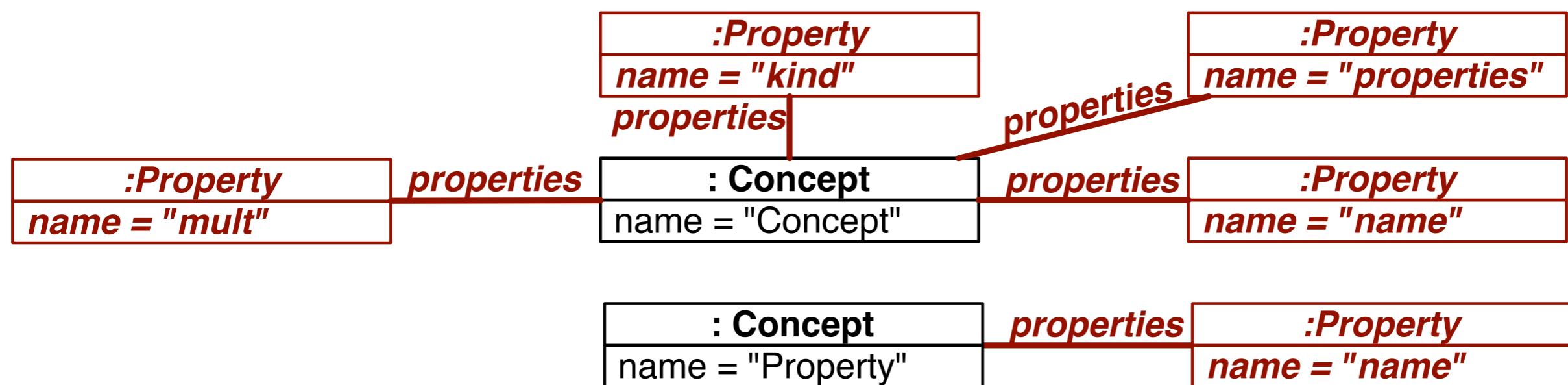
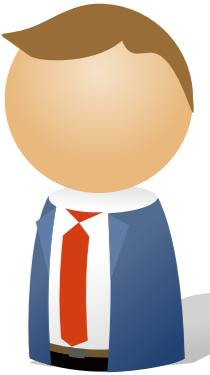
Step #1: Holding **Properties** (UML helper)



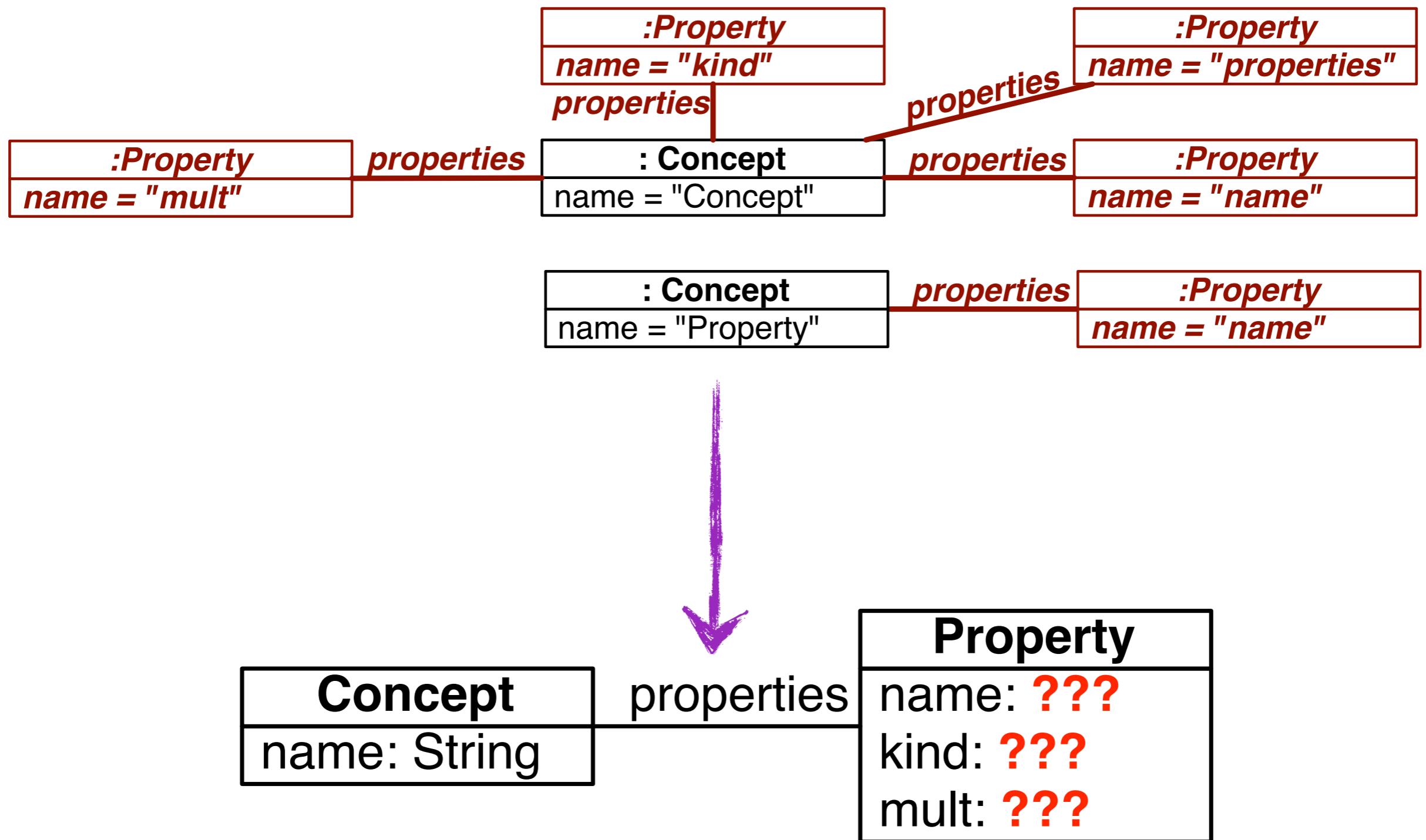
Step #1: Holding **Properties** (UML helper)



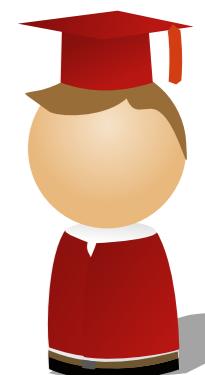
Step #1: Holding **Properties** (UML helper)



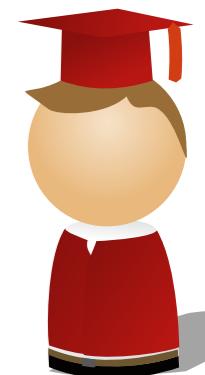
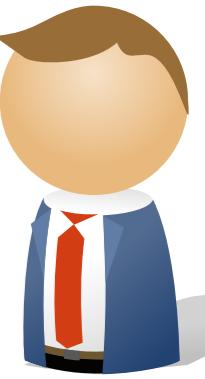
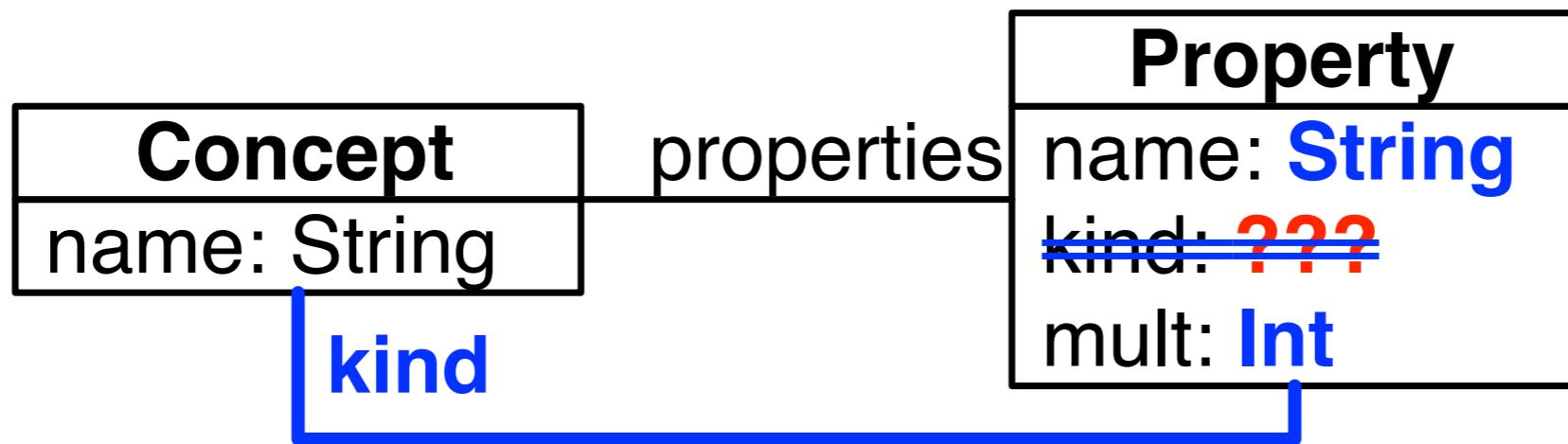
Code Generation



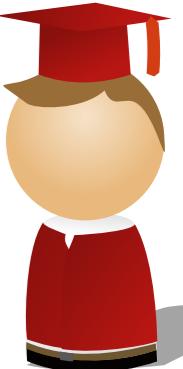
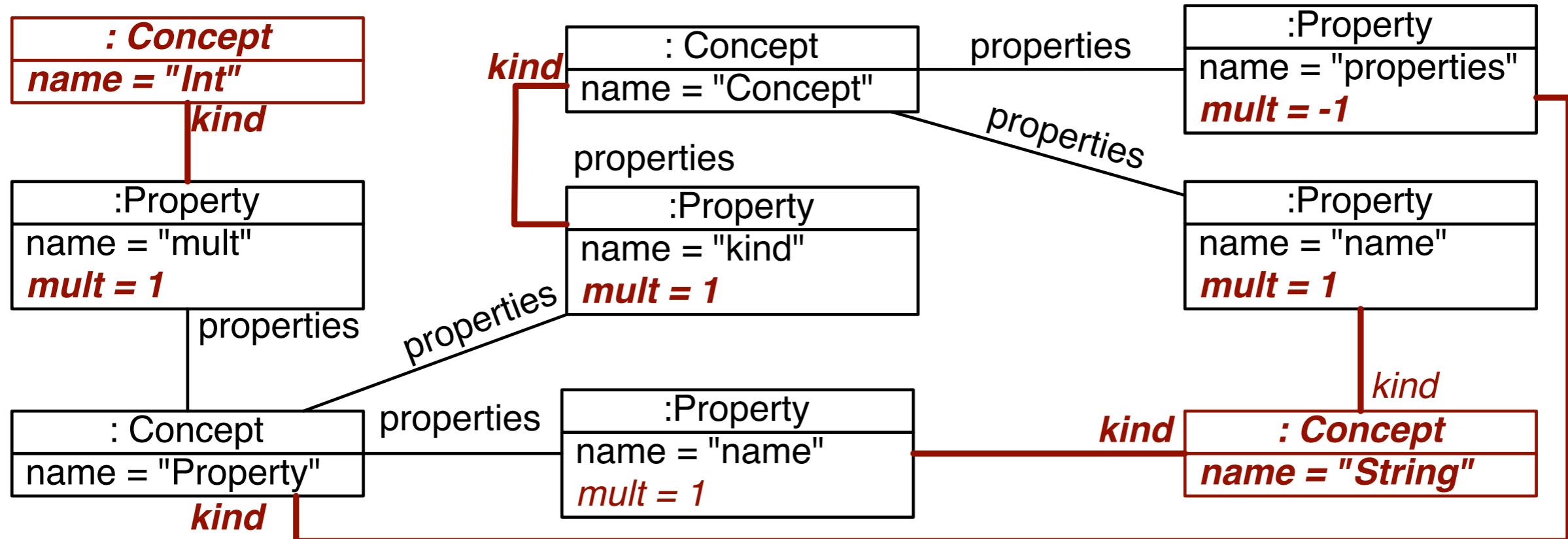
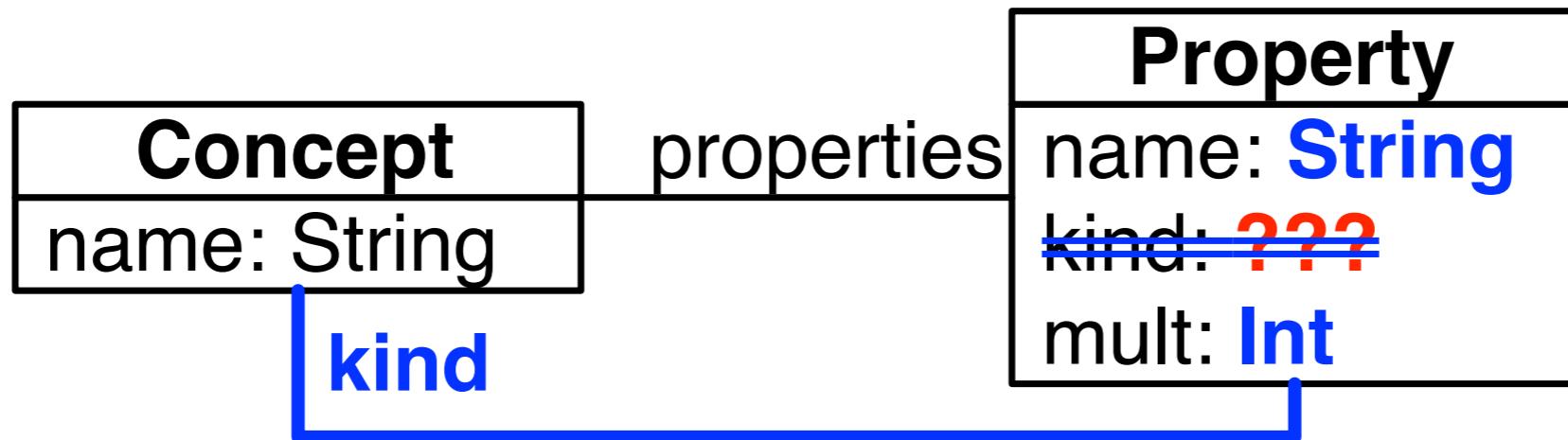
Step #2: Typed **Properties** (UML helper')



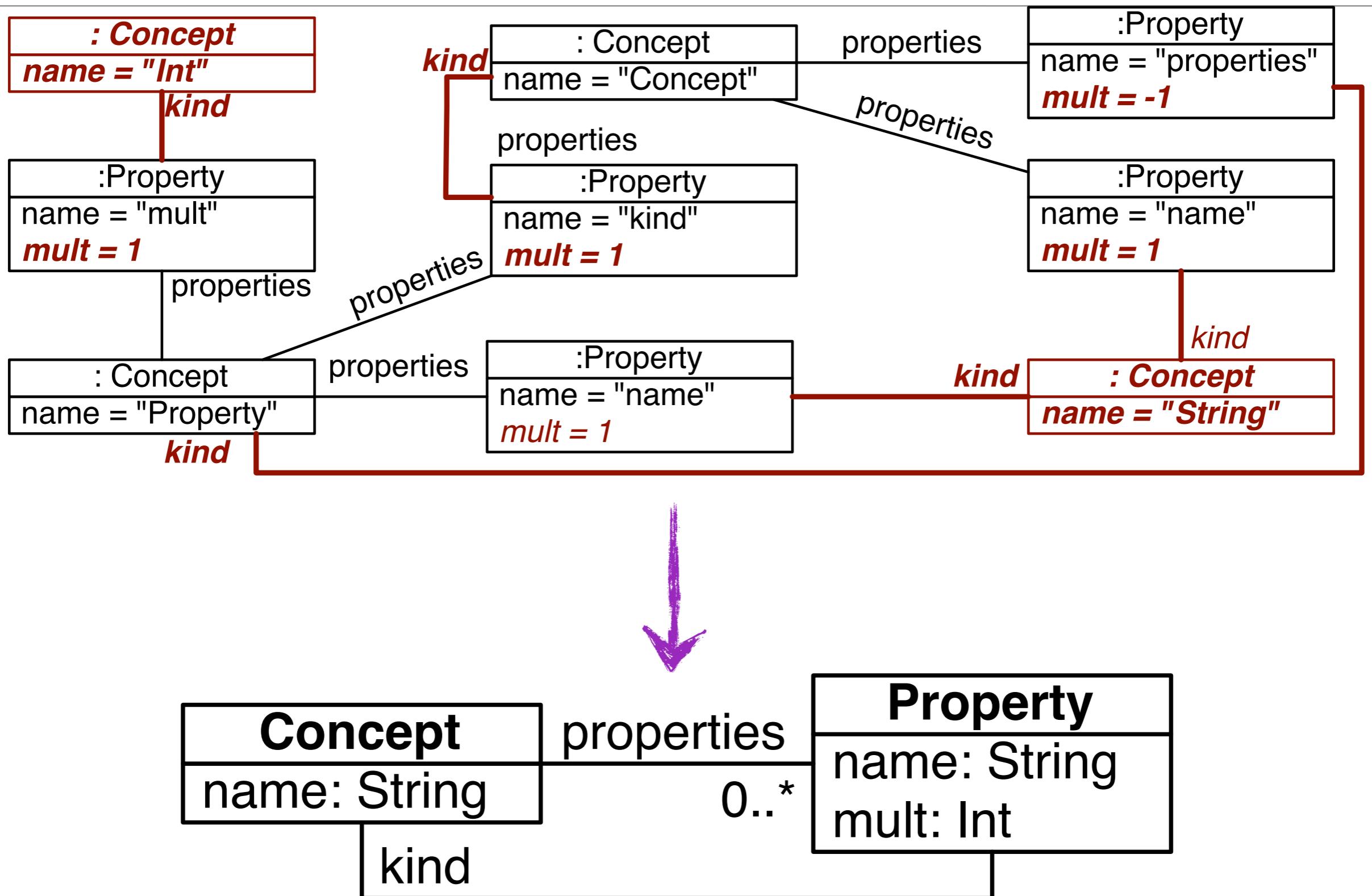
Step #2: Typed **Properties** (UML helper')



Step #2: Typed Properties (UML helper')

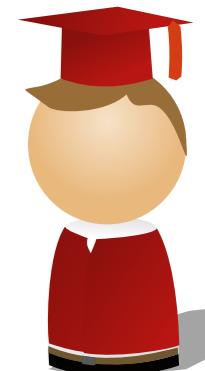
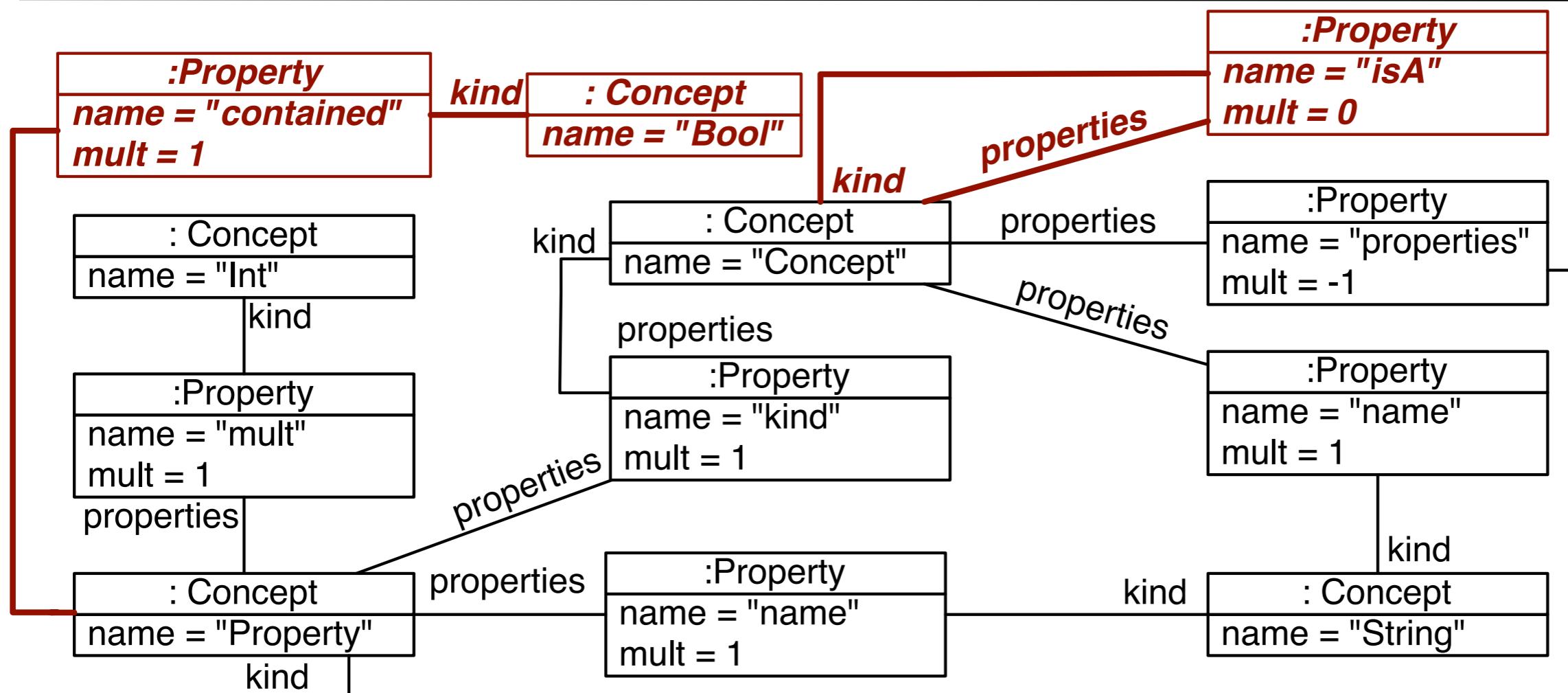
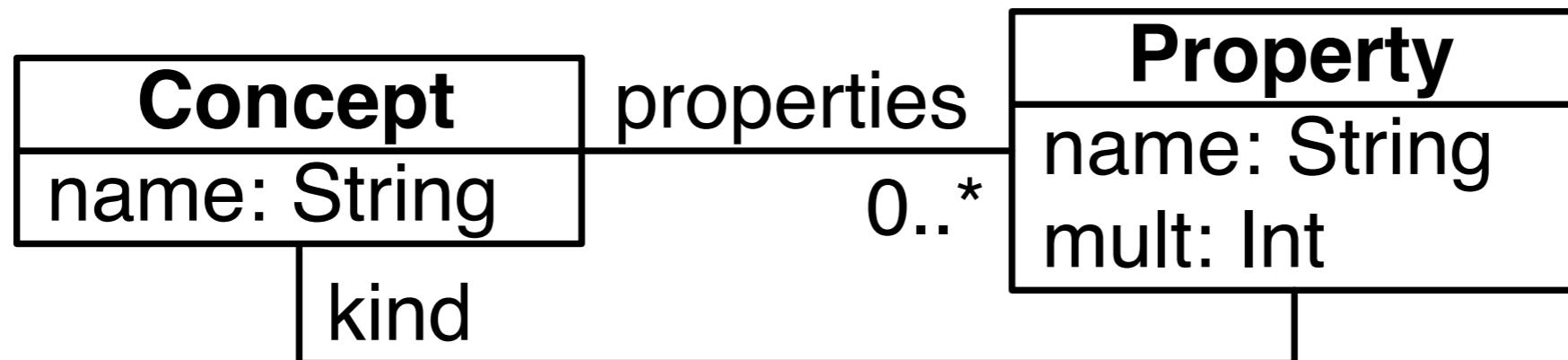


Code Generation

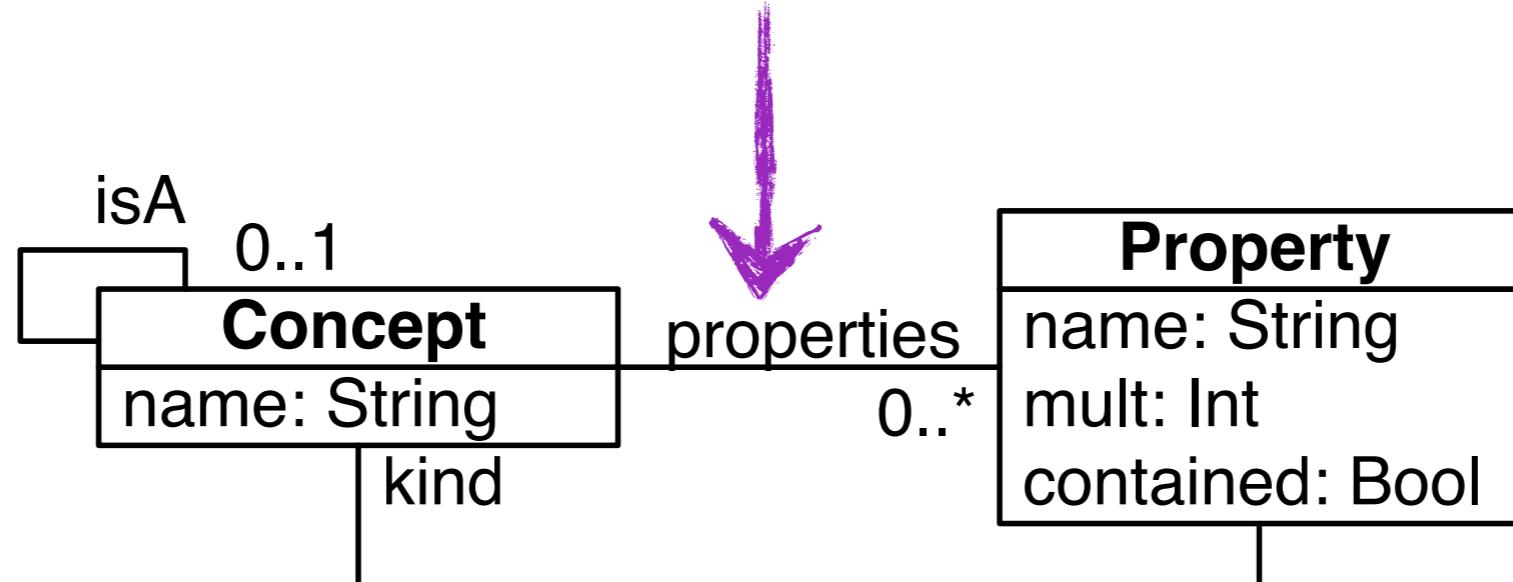
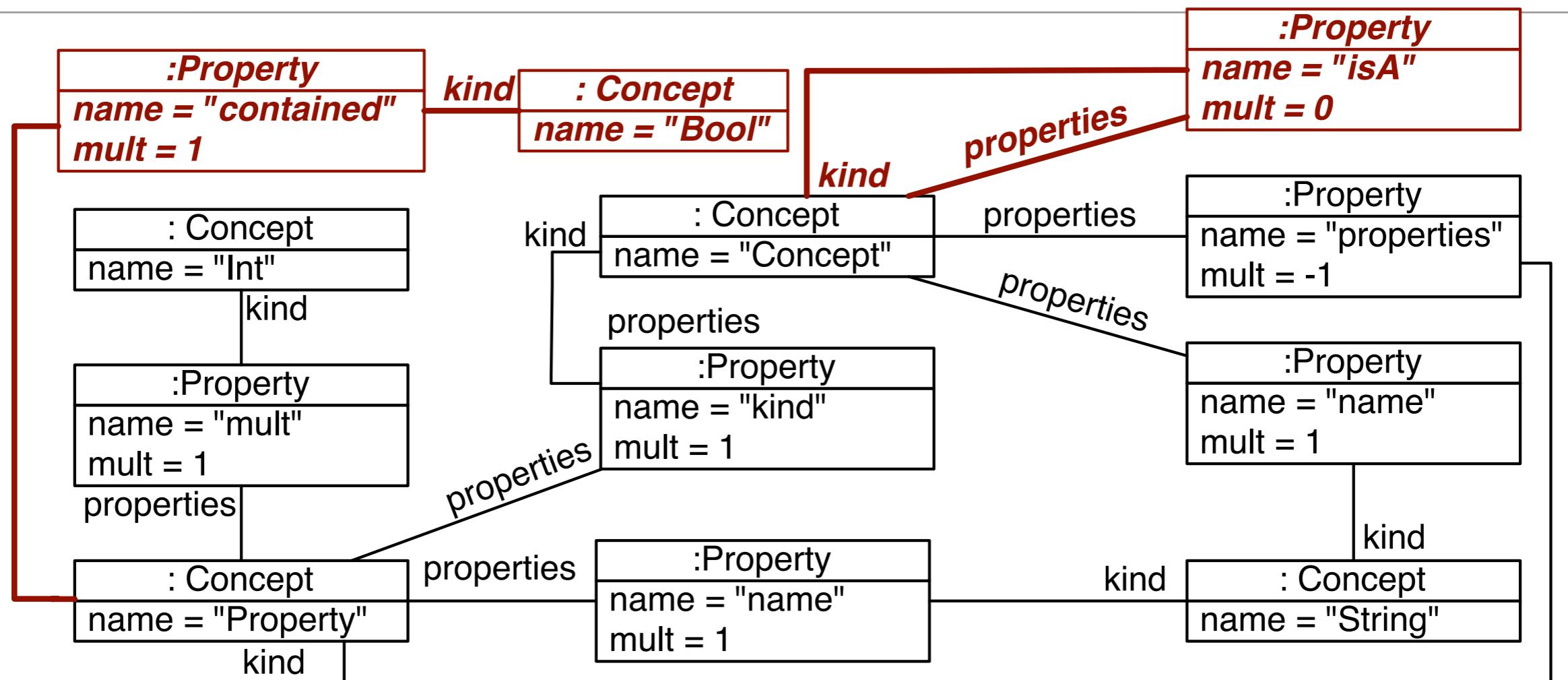


Step #3: Adding Inheritance & Containment

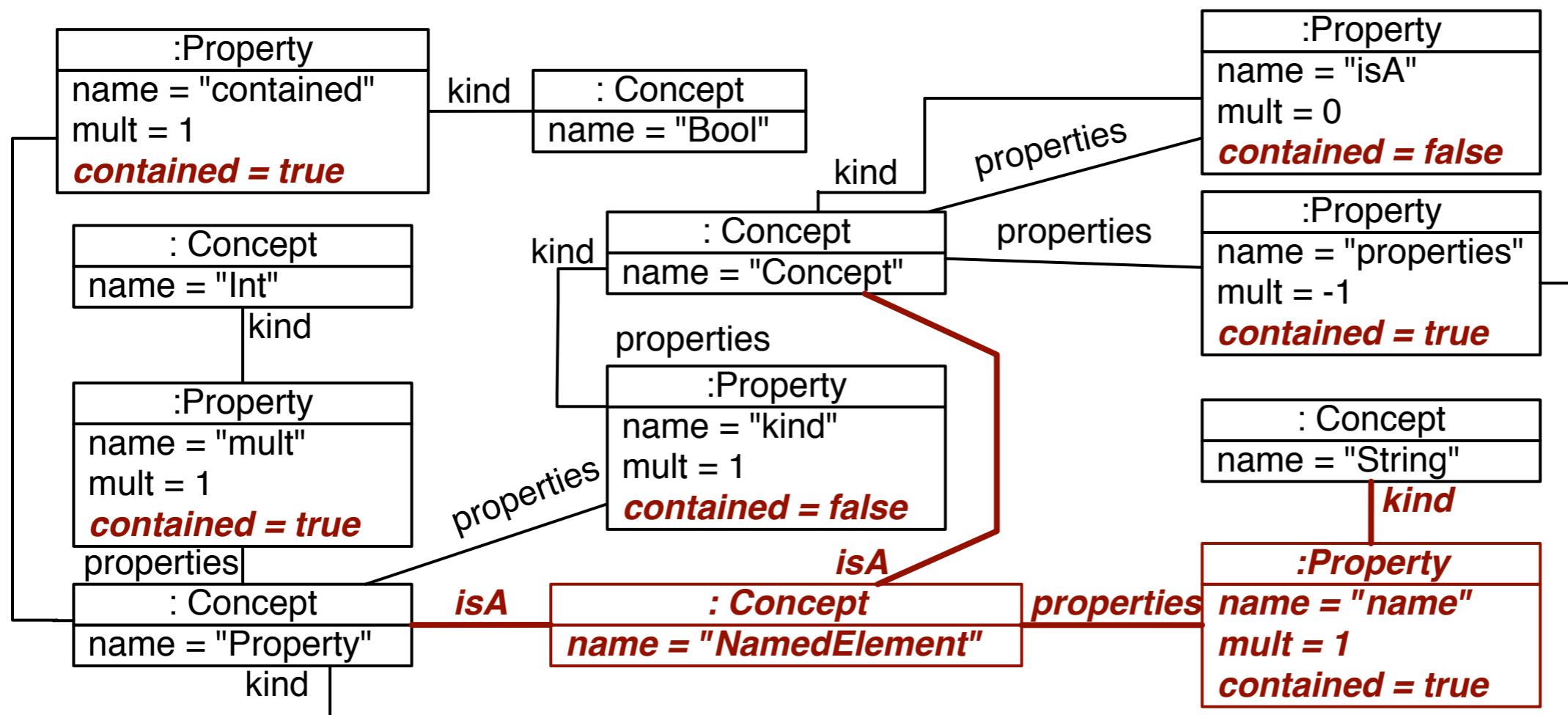
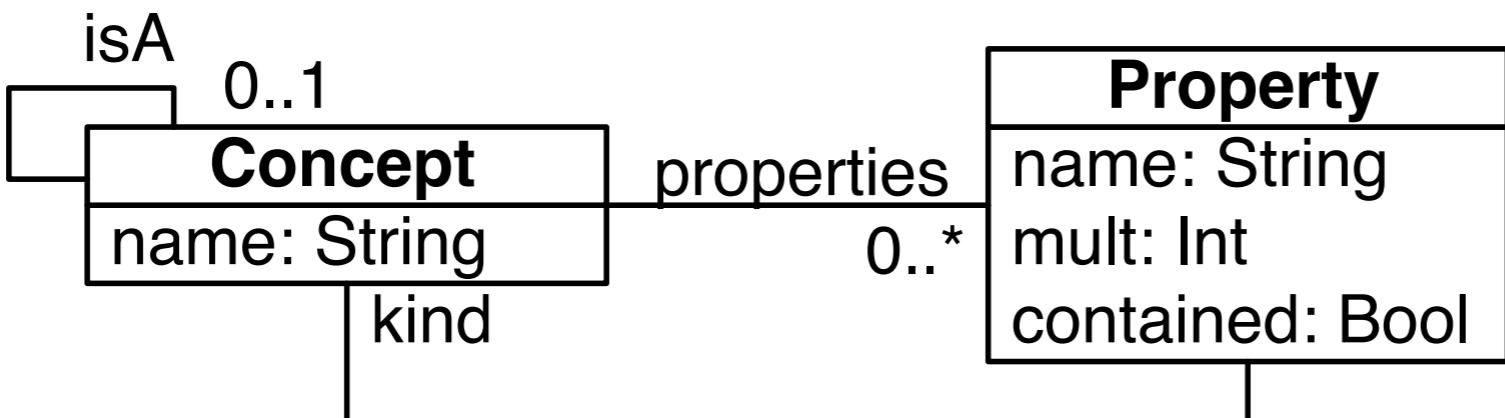
UXL



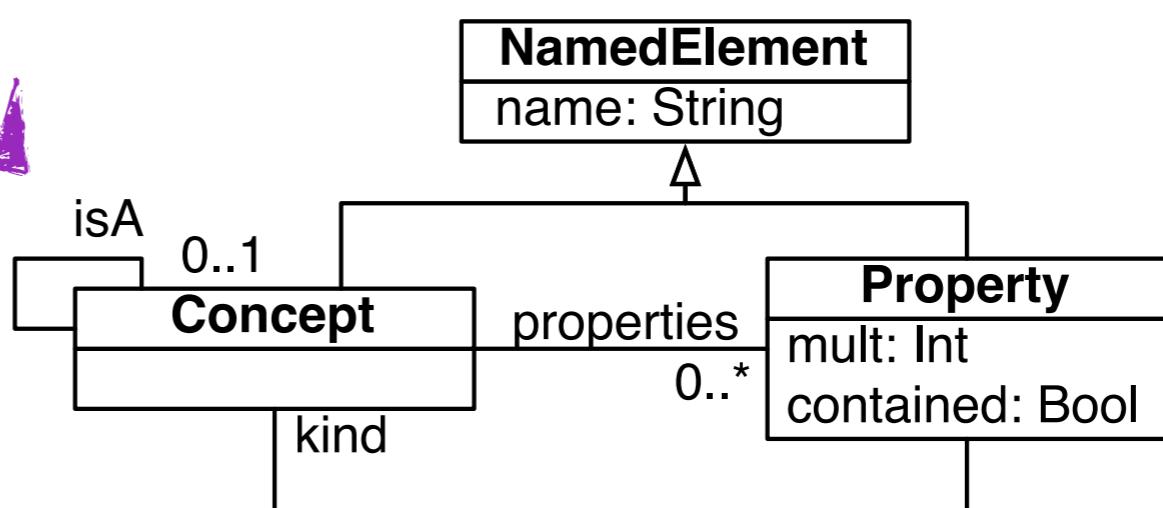
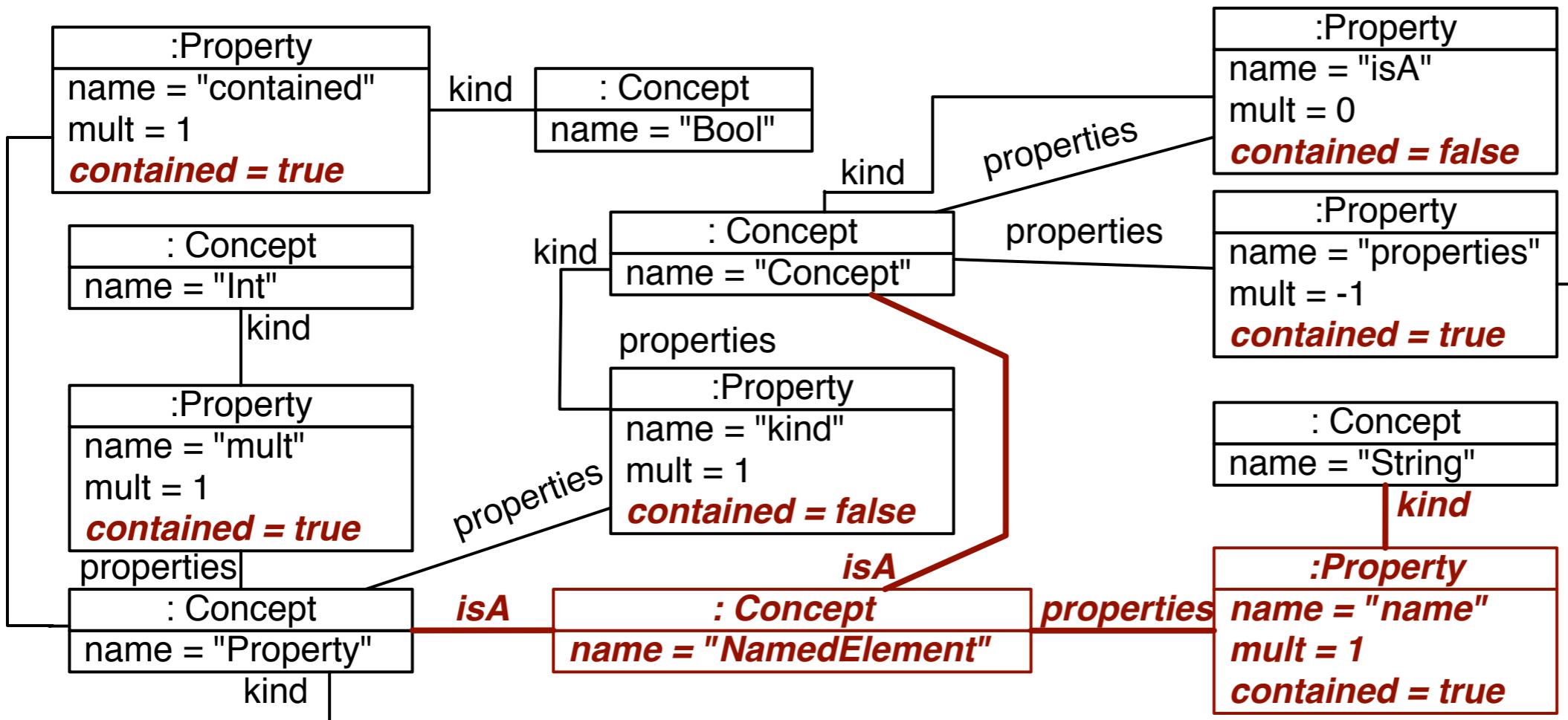
Code Generation



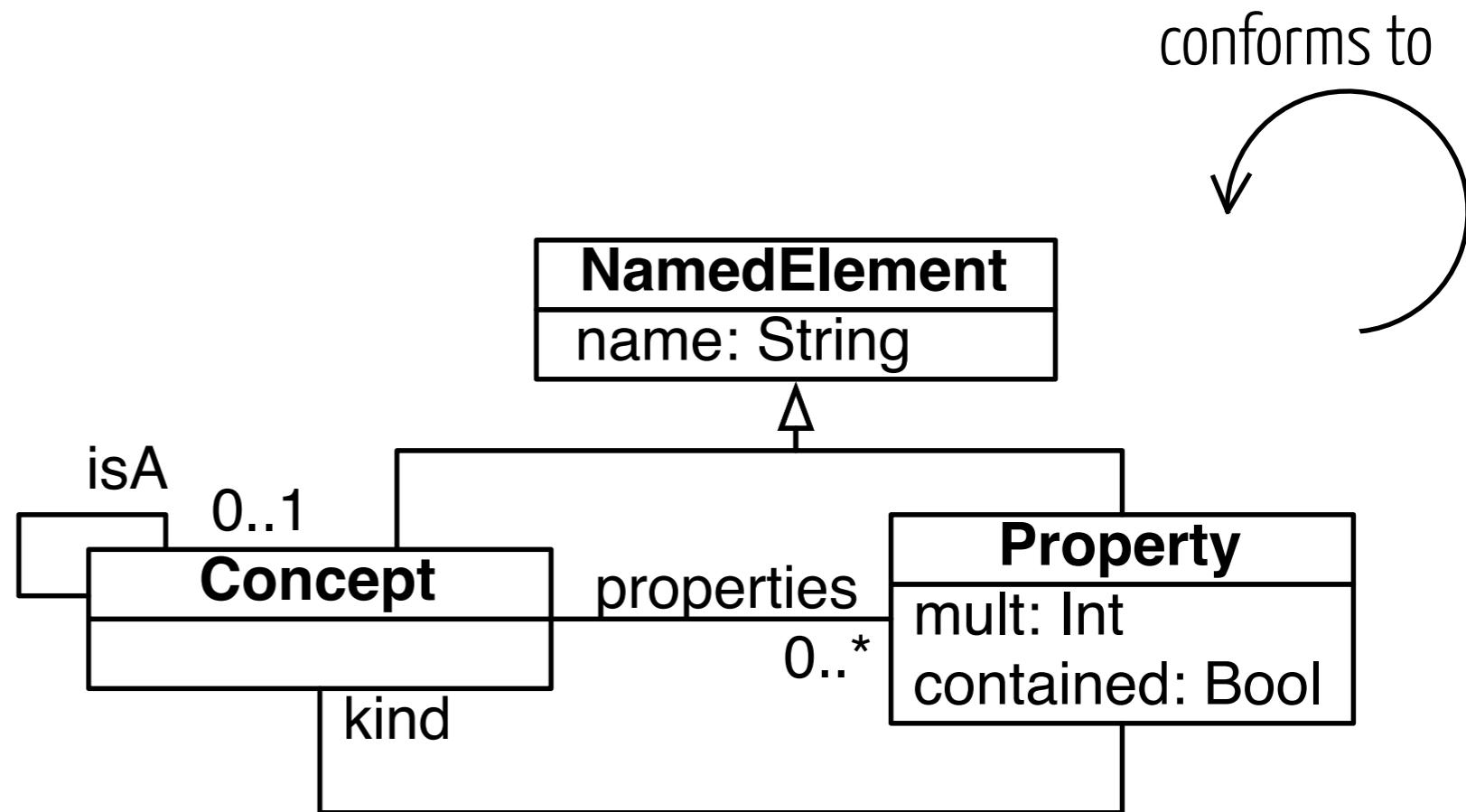
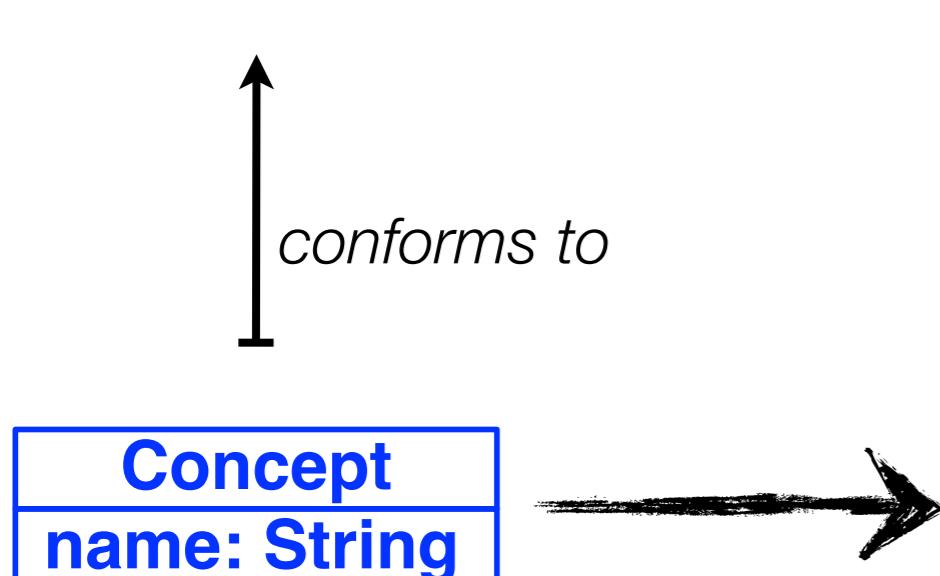
Step #4: Expressive enough to refactor



Code Generation



We just bootstrapped a modeling language!



**Time for
a break!**



**From models
to
domain-specific
languages**



Secret Compartments

- To open the **Diagon Alley** wall:
 - «From the trash can,
 - Tap three bricks up,
 - and two across.»
- To open the **secret door**:
 - Pull the left candlestick,
 - Push the 3rd brick from the right,
 - Pull Vol. 7 of the Encyclopedia.

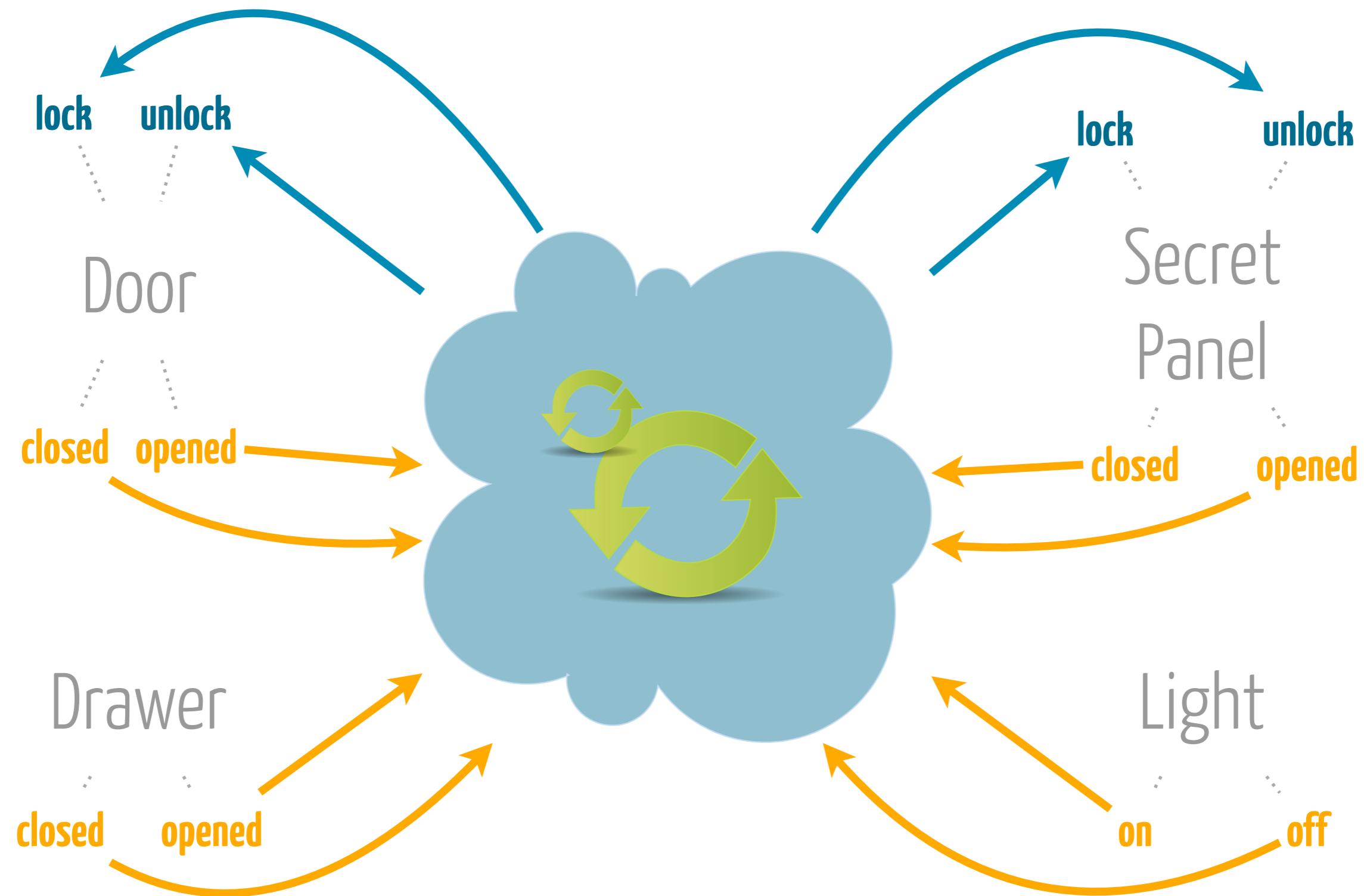


Miss Grant's Secret Drawer

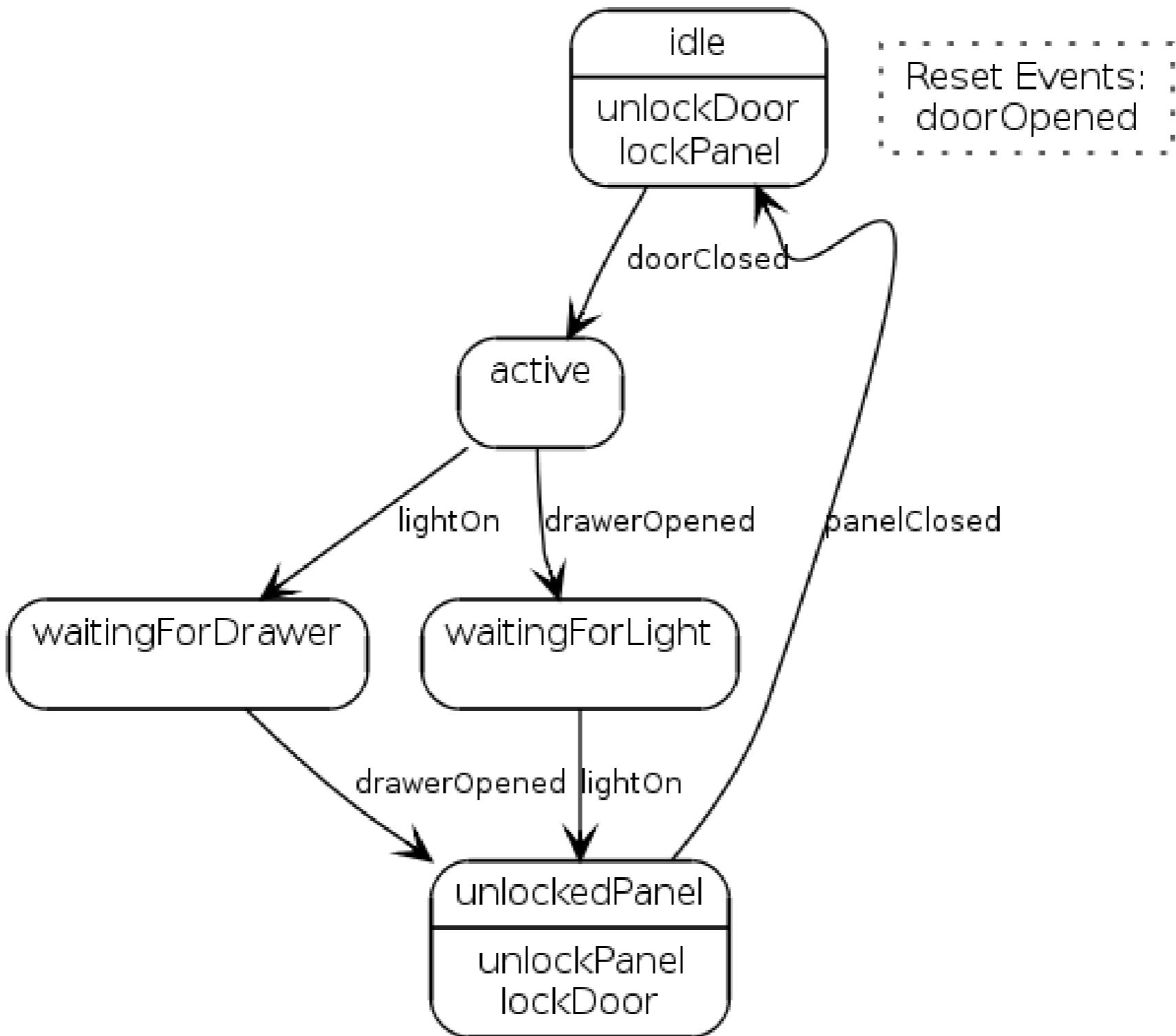


«To **open** it, she has to **close the door**, then
open the second drawer in her chest and
turn her bedside light on—in either order.
Once these are done, **the secret panel is
unlocked** for her to open.»

Miss Grant's Environment



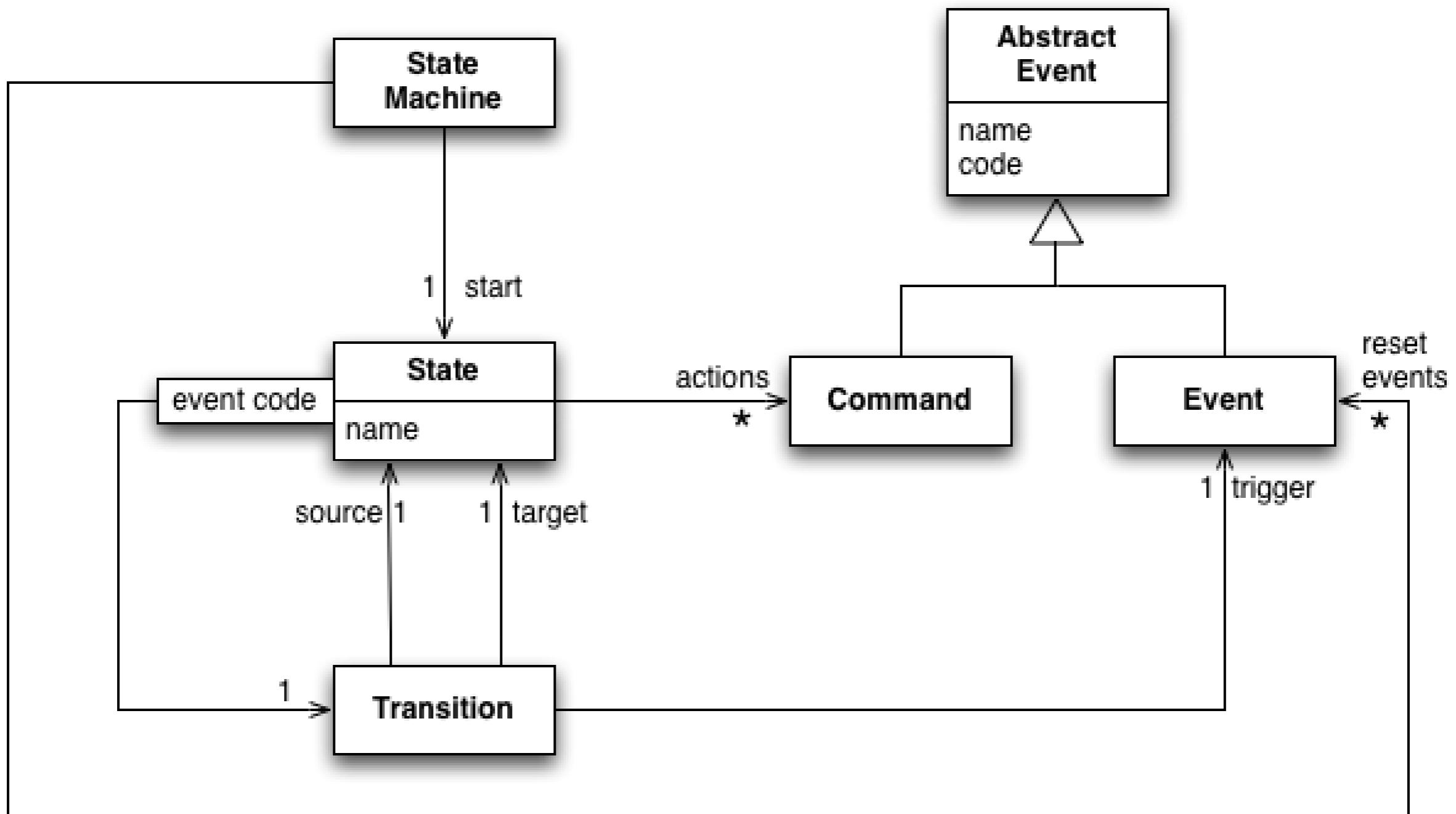
Formalising Miss Grant's Controller



Modeling Controllers

- **Events:**
 - A code sent by the environment
 - Inform about context changes
- **Commands:**
 - A code sent to the environment
 - Change the context (e.g., unlock)
- **States**
 - Controller's current situation
 - Might trigger Commands
 - Reacts to events with Transitions
- **Transitions**
 - Associate an event to a next State

Associated Structural Concepts



Programming the Controller

[1/2]

```
Event doorClosed = new Event("doorClosed", "D1CL");
Event drawerOpened = new Event("drawerOpened", "D2OP");
Event lightOn = new Event("lightOn", "L1ON");
Event doorOpened = new Event("doorOpened", "D1OP");
Event panelClosed = new Event("panelClosed", "PNCL");

Command unlockPanelCmd = new Command("unlockPanel", "PNUL");
Command lockPanelCmd = new Command("lockPanel", "PNLK");
Command lockDoorCmd = new Command("lockDoor", "D1LK");
Command unlockDoorCmd = new Command("unlockDoor", "D1UL");

State idle = new State("idle");
State activeState = new State("active");
State waitingForLightState = new State("waitingForLight");

State waitingForDrawerState = new State("waitingForDrawer");
State unlockedPanelState = new State("unlockedPanel");

StateMachine machine = new StateMachine(idle);
```

Programming the Controller

[2/2]

```
idle.addTransition(doorClosed, activeState);
idle.addAction(unlockDoorCmd); idle.addAction(lockPanelCmd);

activeState.addTransition(drawerOpened, waitingForLightState);
activeState.addTransition(lightOn, waitingForDrawerState);

waitingForLightState.addTransition(lightOn, unlockedPanelState);
waitingForDrawerState.addTransition(drawerOpened,
unlockedPanelState);

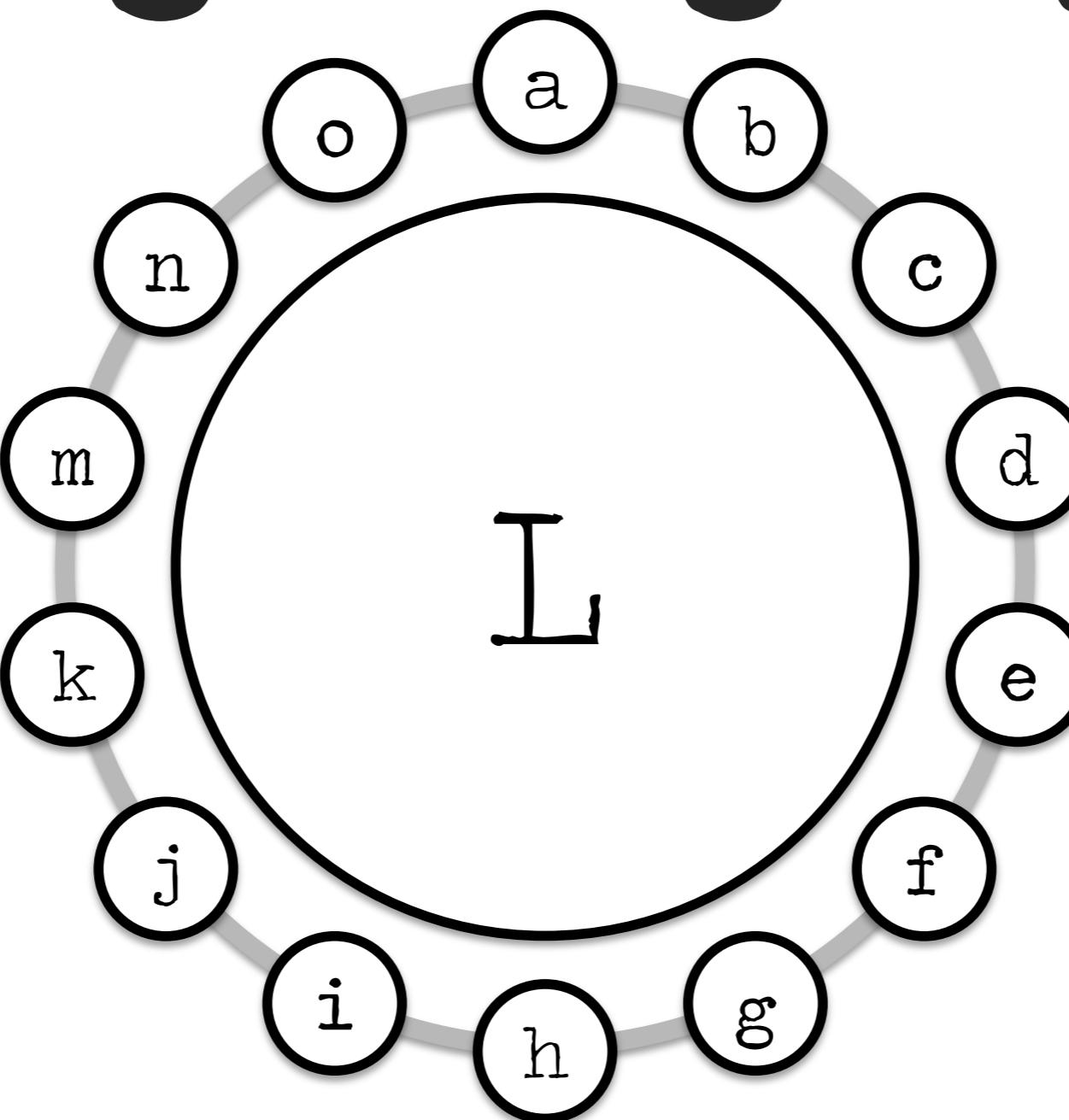
unlockedPanelState.addAction(unlockPanelCmd);
unlockedPanelState.addAction(lockDoorCmd);
unlockedPanelState.addTransition(panelClosed, idle);

machine.addResetEvents(doorOpened);
```

Wait!

Where the hell
is my domain?

Big Language



with many first
class concepts!

What the hell

is my domain?

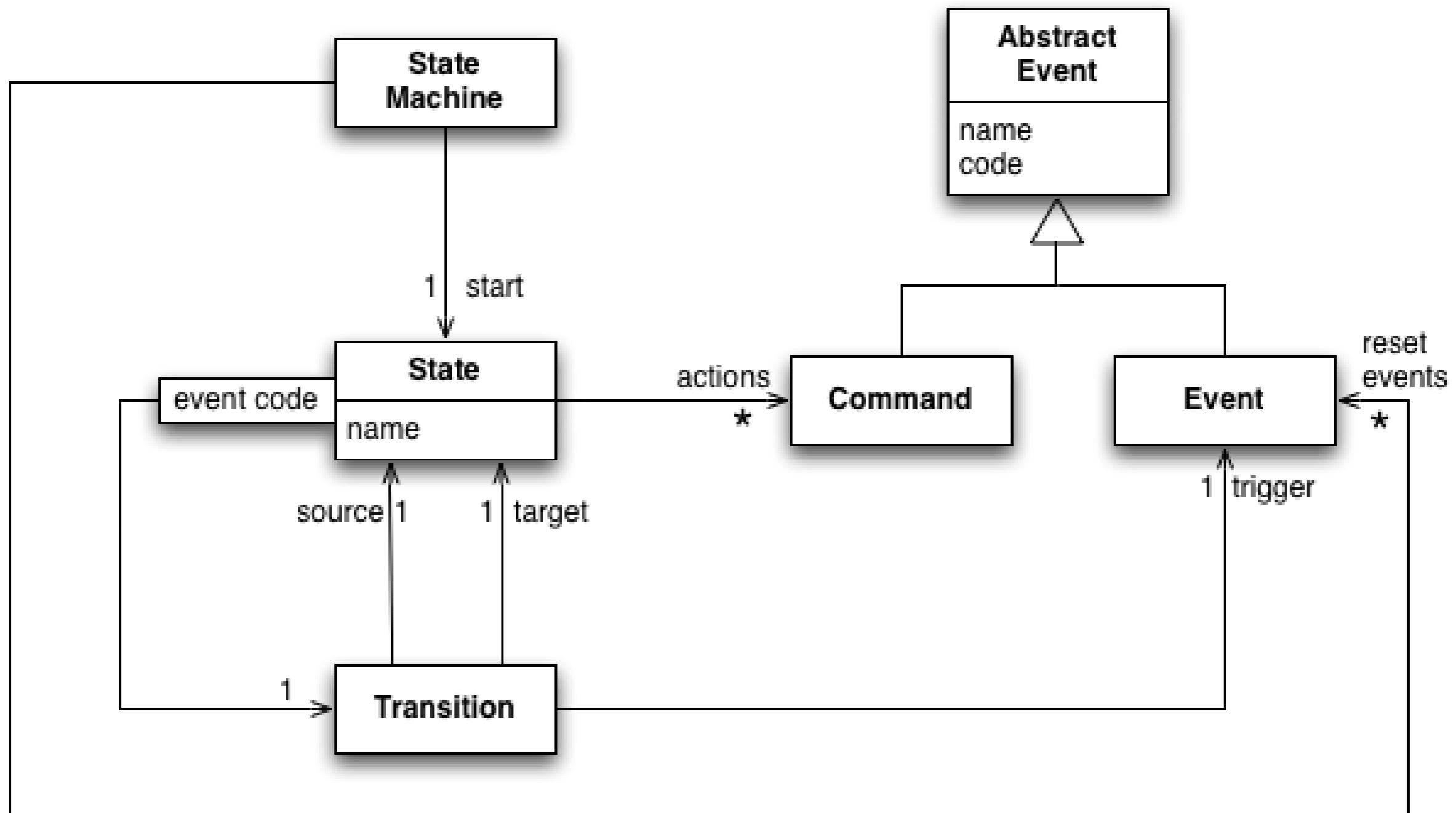
Controller

From «**Programming**
the **model** of the
Controller» ...

... to

«Modeling the Controller»

Associated Structural Concepts



events

doorClosed	D1CL
drawerOpened	D2OP
lightOn	L1ON
doorOpened	D1OP
panelClosed	PNCL

end

resetEvents
 doorOpened
end

commands

unlockPanel	PNUL
lockPanel	PNLK
lockDoor	D1LK
unlockDoor	D1UL

end
state idle

actions {**unlockDoor** **lockPanel**}
doorClosed => **active**
end

state active

drawerOpened => **waitingForLight**
 lightOn => **waitingForDrawer**
end

state waitingForLight

lightOn => **unlockedPanel**
end

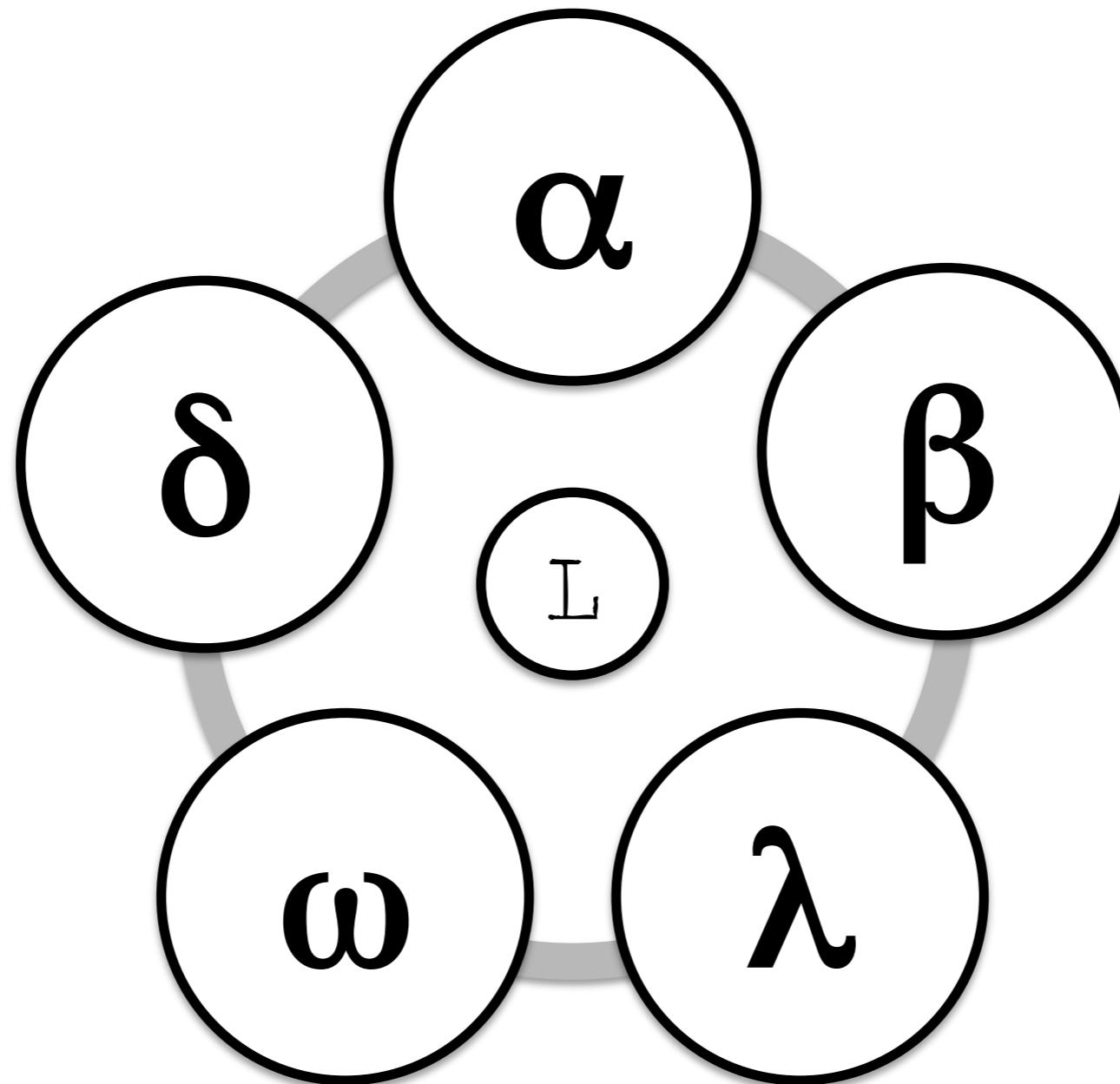
state waitingForDrawer

drawerOpened => **unlockedPanel**
end

state unlockedPanel

actions {**unlockPanel** **lockDoor**}
 panelClosed => **idle**
end

Small Language

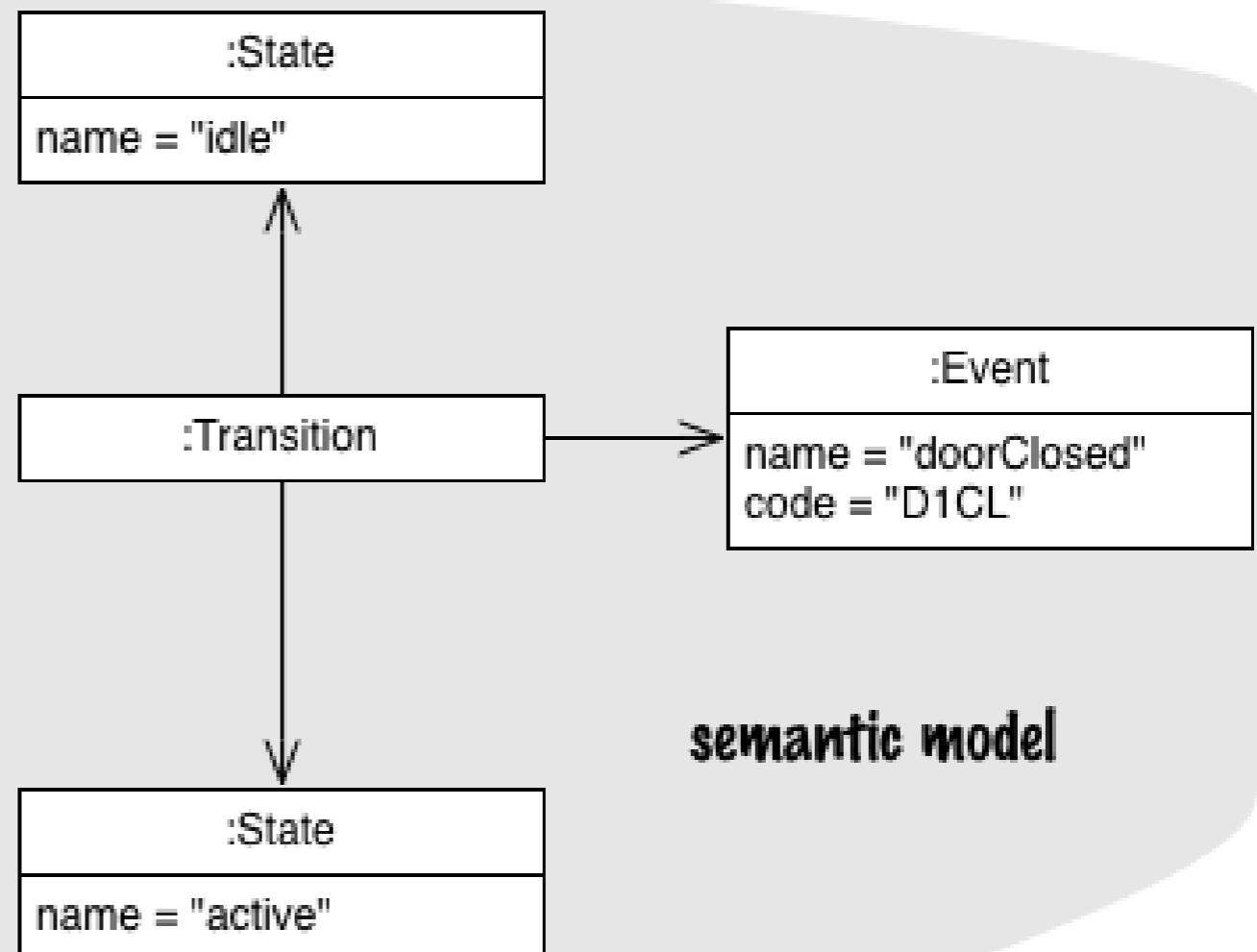
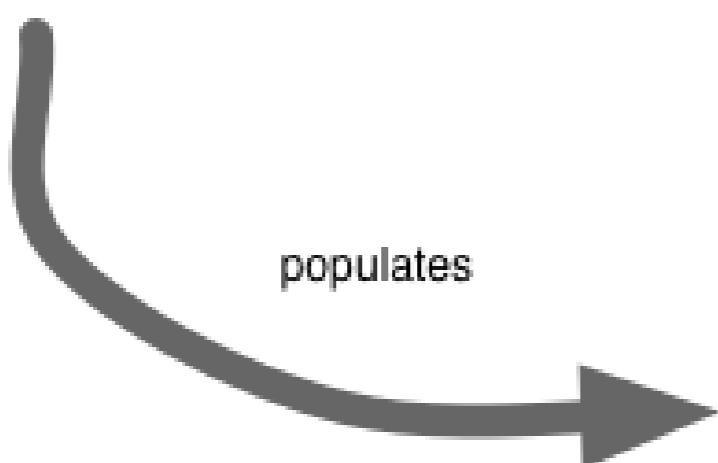


with a few, orthogonal
and powerful concepts

```
events
  doorClosed  D1CL
end

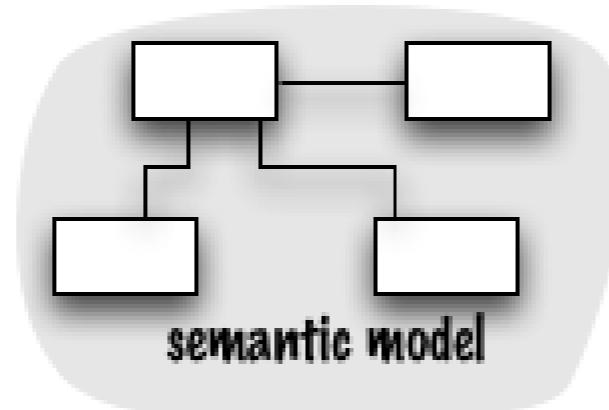
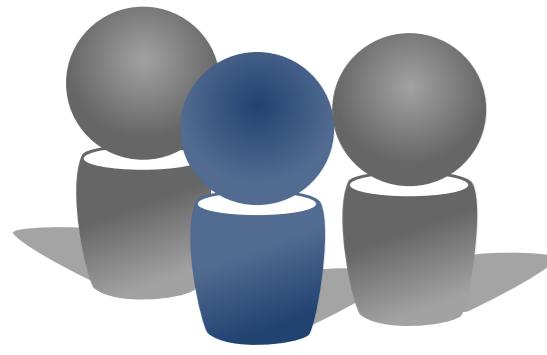
state idle
  doorClosed => active
end
```

input

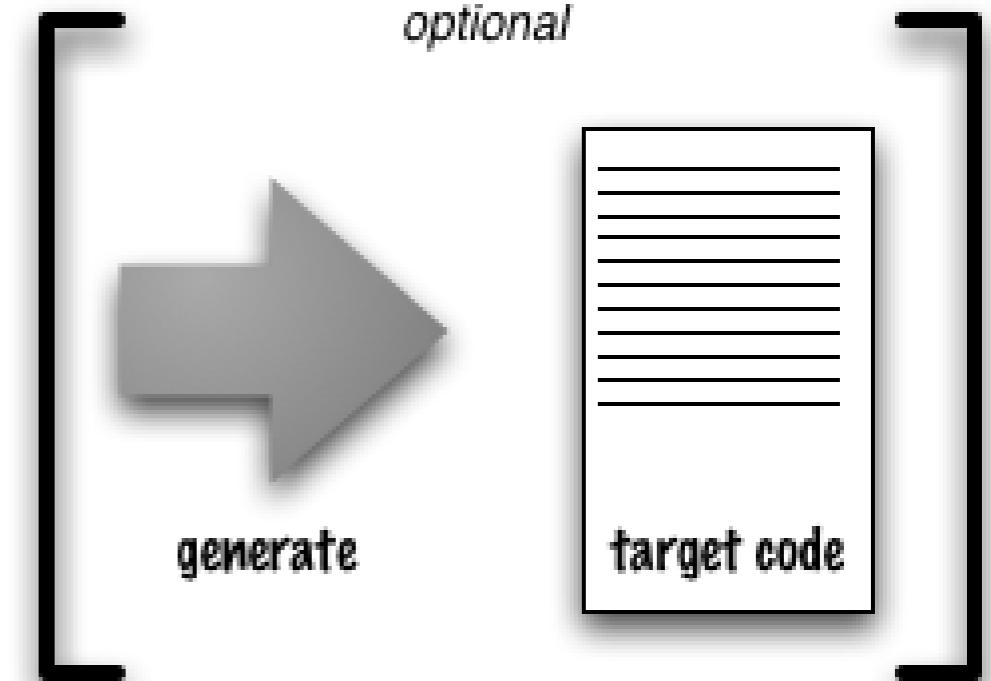


semantic model

[Fowler]



parse

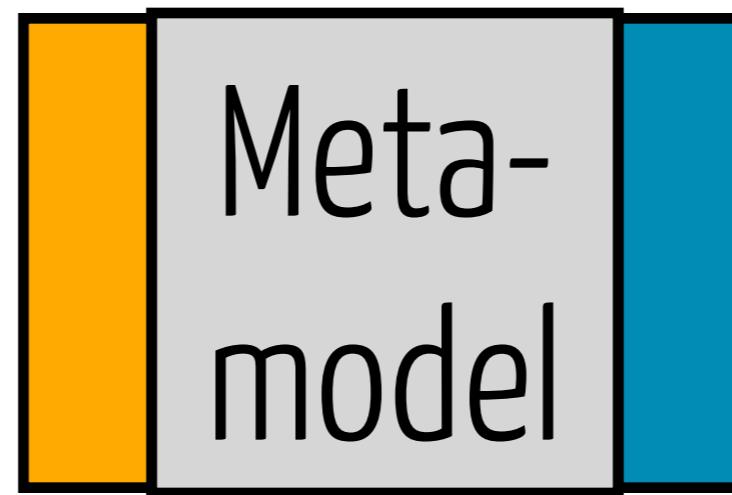


[Domain-Specific Languages]

Semantic Model \equiv Meta-model

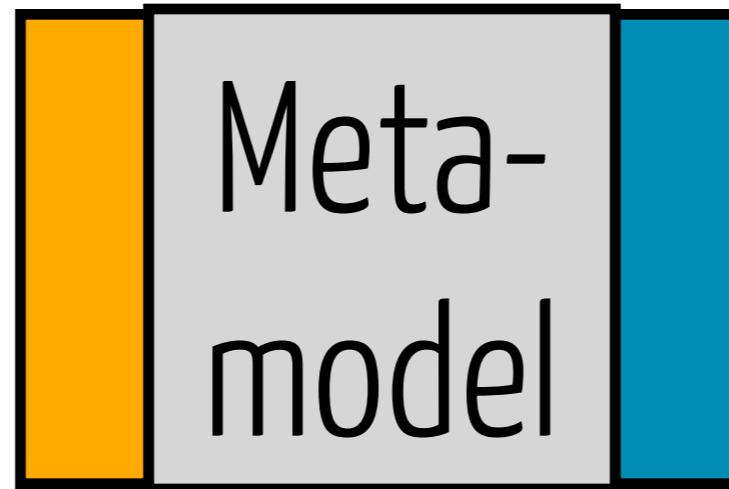
- The «**essence**» of a domain, according to a **given viewpoint**
- Should be **usable with or without a DSL**
- Interfaces to **exploit** the meta-model

Population
Interface

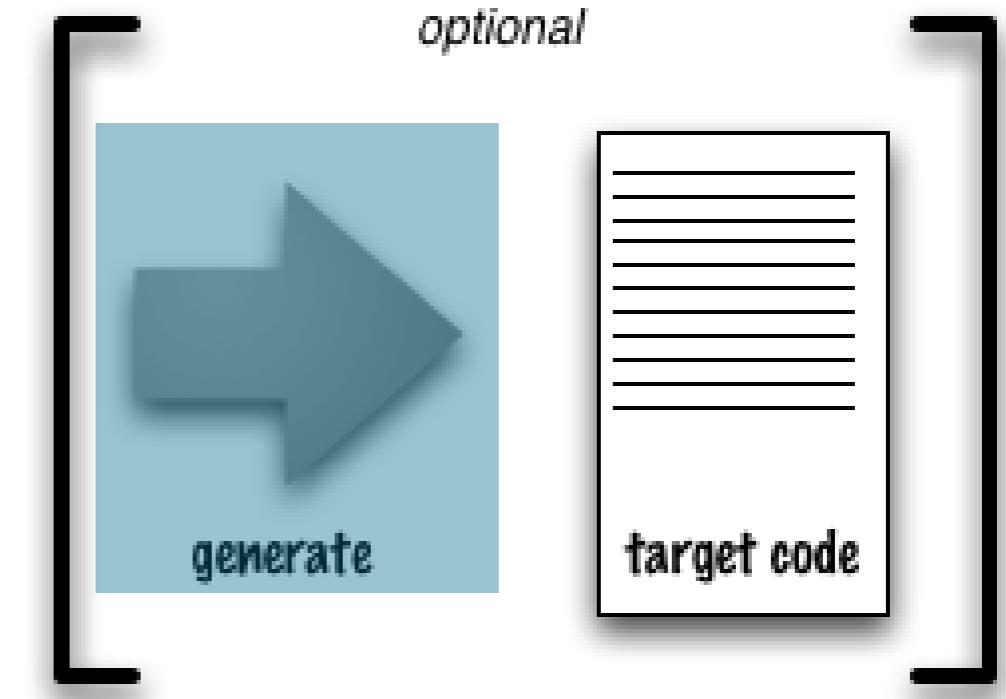
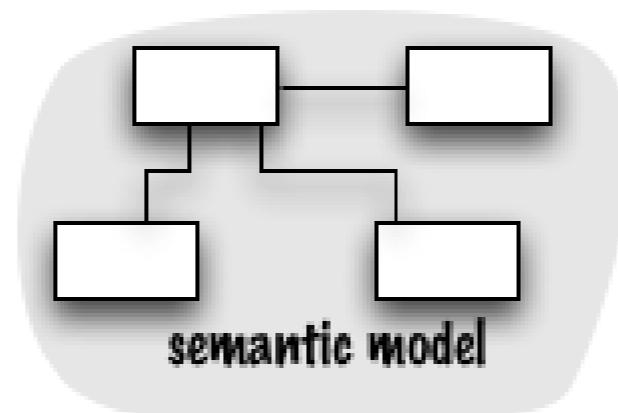


Operational
Interface

Population Interface



Operational Interface

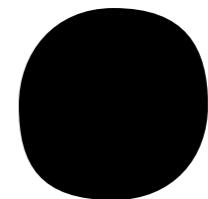


DSLs
versus
GPLs



This is not a
compilation
course!

	more in GPLs	more in DSL
Domain Size	large and complex	
Designed by	guru or committee	
Language Size	large	
Turing-completeness	almost always	
User Community	large, anonymous and widespread	
In-language abstraction	sophisticated	
Lifespan	years to decades	
Evolution	slow, often standardized	
Incompatible Changes	almost impossible	



[Voelter]

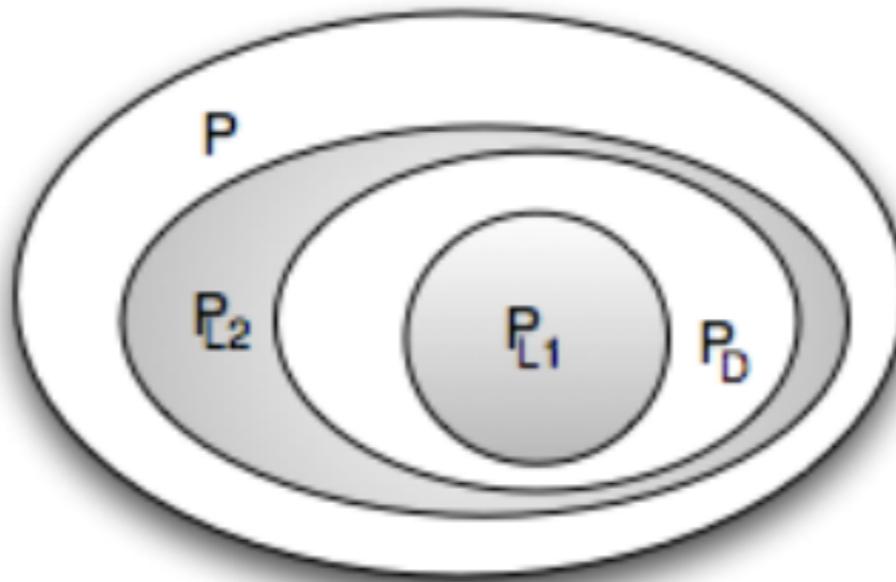
	more in GPLs	more in DSL
Domain Size	large and complex	smaller and well-defined
Designed by	guru or committee	a few engineers and domain experts
Language Size	large	small
Turing-completeness	almost always	often not
User Community	large, anonymous and widespread	small, accessible and local
In-language abstraction	sophisticated	limited
Lifespan	years to decades	months to years (driven by context)
Evolution	slow, often standardized	fast-paced
Incompatible Changes	almost impossible	feasible
		[Voelter]

A photograph of a lioness and her cub in a field of tall, dry grass. The lioness is on the left, looking towards the right, while her cub is partially visible behind her, also looking in the same direction. The scene is set in a natural, outdoor environment.

Your
DSL

You

#ABCWorldNews



A DSL L_D for D is a language that is specialized to en-

coding P_D programs.

more efficient
smaller

Shorter Programs

More
Accessible
Semantics

For a limited
Domain!

Domain Knowledge
encapsulated in
language

!!

Using patterns in a
language is a
symptom of lack of
specialization

Overspecification! Requires Semantic Analysis!

```
int[] arr = ...  
for (int i=0; i<arr.size(); i++) {  
    sum += arr[i];  
}
```

```
int[] arr = ...  
List<int> l = ...  
for (int i=0; i<arr.size(); i++) {  
    l.add( arr[i] );  
}
```

Linguistic Abstraction

```
for (int i in arr) {  
    sum += i;  
}
```

Declarative!
Directly represents Semantics.

```
seqfor (int i in arr) {  
    l.add( arr[i] );  
}
```

Def: DSL

A DSL is a **language** at D that provides **linguistic abstractions** for **common patterns and idioms** of a language at D-1 when used within the domain D.

Def: DSL cont'd

A good DSL does **not** require the use of patterns and idioms to express **semantically interesting** concepts in D.

Processing tools do not have to do "semantic recovery" on D programs.

Declarative!

[Voelter]

Computation Models



Decision Table

Premium Customer	X	X	Y	Y	N	N
Priority Order	Y	N	Y	N	Y	N
International Order	Y	Y	N	N	N	N
Fee	150	100	70	50	80	60
Alert Rep	Y	Y	Y	N	N	N

conditions

consequences

The diagram shows five arrows originating from the right side of the slide and pointing towards the columns of the decision table. The first three arrows point to the columns labeled X, Y, and N under the 'Premium Customer' row, which are grouped together and labeled 'conditions'. The last two arrows point to the columns labeled 150, 70, and 60 under the 'Fee' row, which are grouped together and labeled 'consequences'.

Example: FIT

The screenshot shows a Microsoft Word document window titled "result.htm - Microsoft Word". The menu bar includes File, Edit, View, Insert, Format, Tools, Table (which is highlighted), Window, Documents To Go, and Help. The toolbar contains icons for file operations like Open, Save, Print, and Find, along with a Recount button. The status bar at the bottom shows Page 1, Sec 1, 1/1, At 1", Ln 1, Col 1, REC, TRK, EXT, OVR, and E.

Basic Employee Compensation

For each week, hourly employees are paid a standard wage per hour for the first 40 hours worked, 1.5 times their wage for each hour after the first 40 hours, and 2 times their wage for each hour worked on Sundays and holidays.

Here are some typical examples of this:

Payroll.Fixtures.WeeklyCompensation			
StandardHours	HolidayHours	Wage	Pay()
40	0	20	\$800
45	0	20	\$950
48	8	20	\$1360 <i>expected</i> \$1040 <i>actual</i>

Business Rules

```
if  
  passenger.frequentFlier  
then  
  passenger.priorityHandling = true;
```

```
if  
  mileage > 25000  
then  
  passenger.frequentFlier = true;
```

Example: DRools

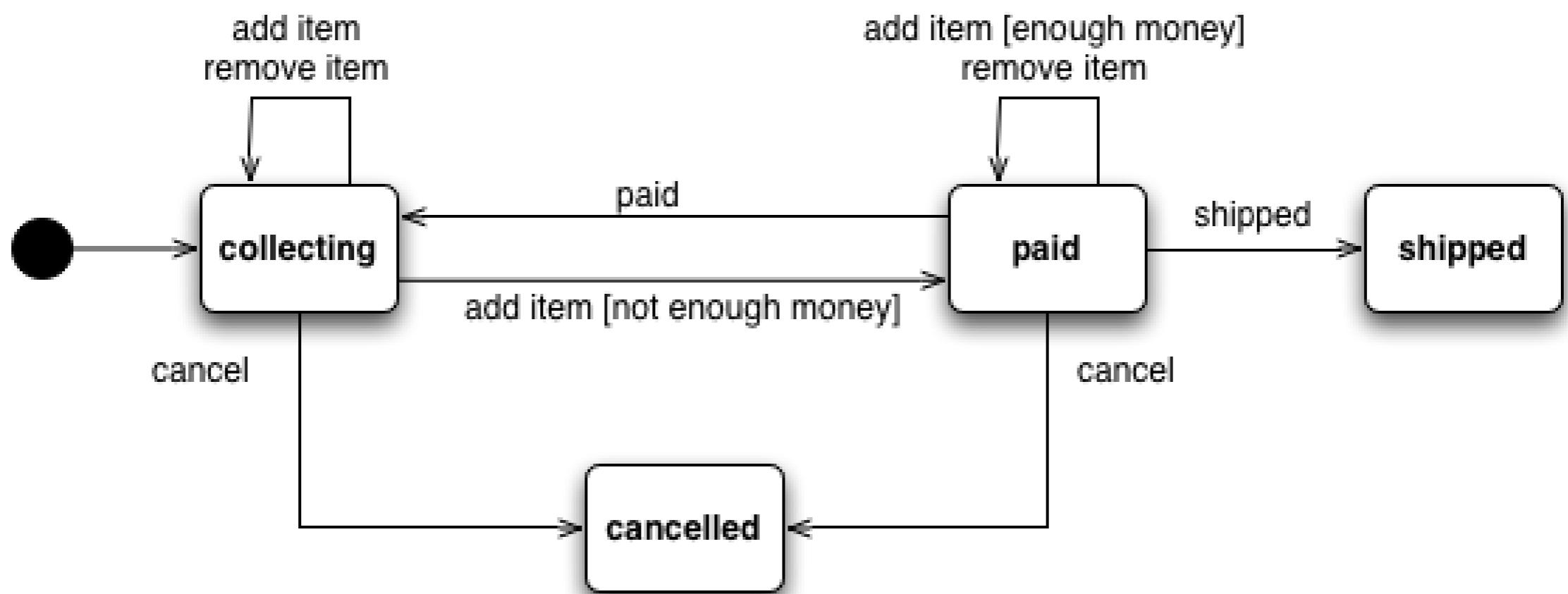
```
package com.comp.drools;

import function com.comp.drools.DroolsHelper.error;
import function com.comp.drools.DroolsHelper.ok;

rule "Account balance is less than 100"
    no-loop
when
    $account : Account( balance < 100 )
then
    error(kcontext, $account); // consequence
end

rule "Account balance is More than 100"
    no-loop
when
    $account : Account( balance > 100 )
then
    ok(kcontext, $account);
end
```

State Machine



Example: JIRA Workflows

JIRA Dashboards Projects Issues Agile More Create Issue Quick Search ?

Administration Search JIRA admin 2

Projects Add-ons User Management **Issues** System

ISSUE TYPES

Issue Types Issue Type Schemes Sub-Tasks

WORKFLOWS

Workflows Workflow Schemes

SCREENS

Screens Screen Schemes Issue Type Screen Schemes

FIELDS

Custom Fields Field Configurations Field Configuration

Workflows

Subject Proposal 2 **ACTIVE** SHARED BY 1 PROJECT

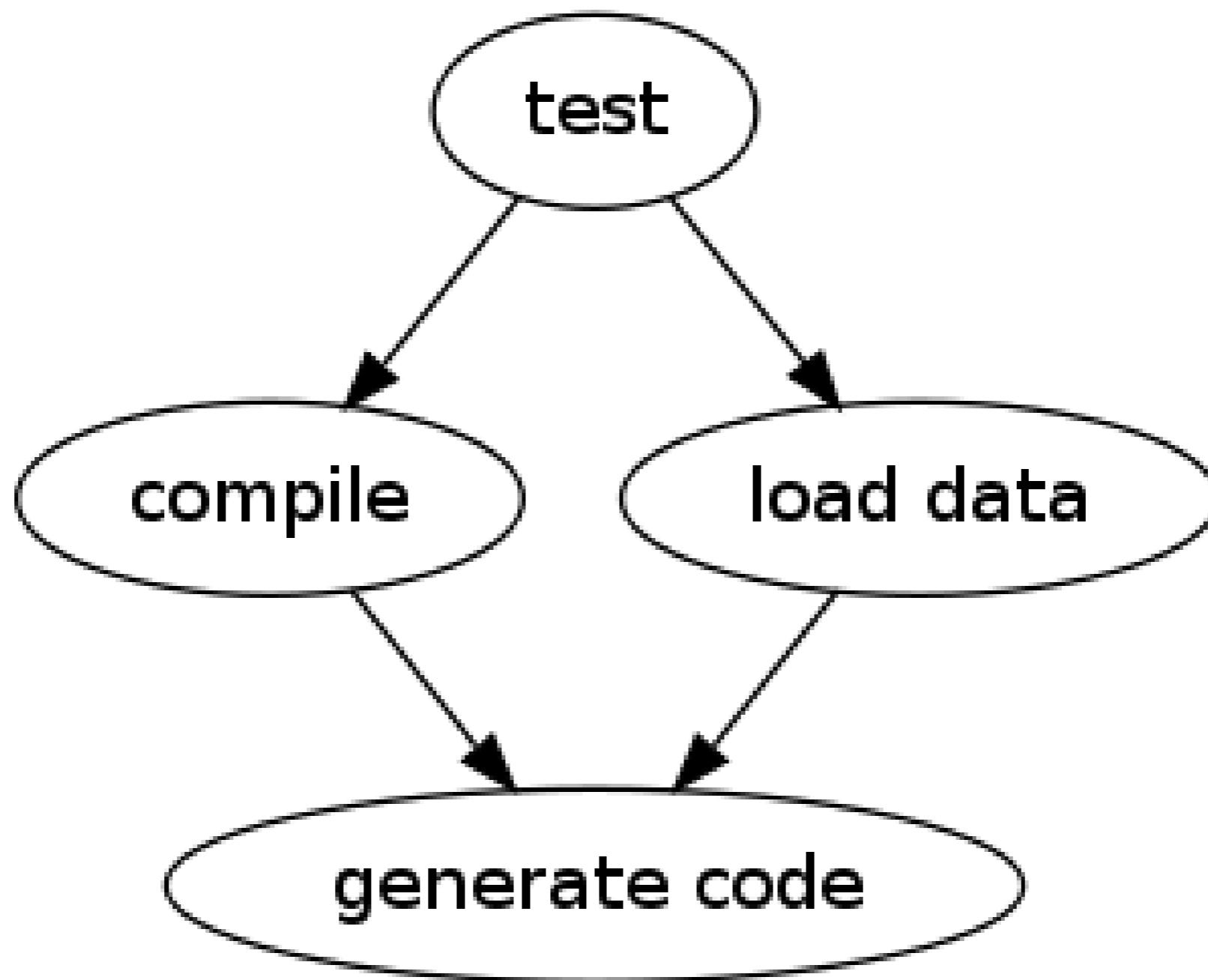
This workflow was last edited by you at 19/Sep/13 3:05 PM.

Diagram Text Export ▾

```
graph TD; Create[Create] --> Open[Open]; Open -- Accept --> Accepted[Accepted]; Open -- Reject --> Rejected[Rejected]; Rejected -- Resubmit --> Open;
```

The diagram illustrates a workflow titled "Subject Proposal 2". It begins with a "Create" step, which leads to an "Open" step. From the "Open" step, two paths emerge: one leading to "Accepted" (indicated by a person icon with a yellow outline) and another leading to "Rejected" (indicated by a person icon with a green checkmark). A "Resubmit" button is located near the "Rejected" step, which has a curved arrow pointing back to the "Open" step.

Dependency Network



Example: Build Automation

COMPILER = gcc

FLAGS = -O -Wall

PROGRAM = myProg

\$ (PROGRAM) : main.o mathlib.o

**\$ (COMPILER) \$ (FLAGS) -o **
 \$ (PROGRAM) main.o mathlib.o

main.o: main.c mathlib.h

\$ (COMPILER) \$ (FLAGS) -c main.c

mathlib.o: mathlib.c mathlib.h

\$ (COMPILER) \$ (FLAGS) -c mathlib.c

DSL

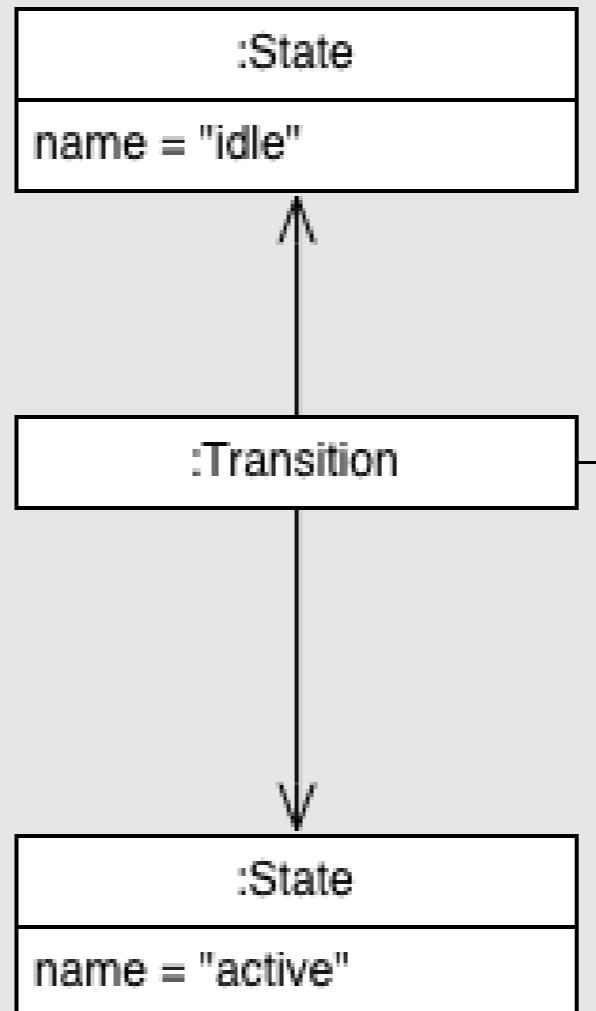
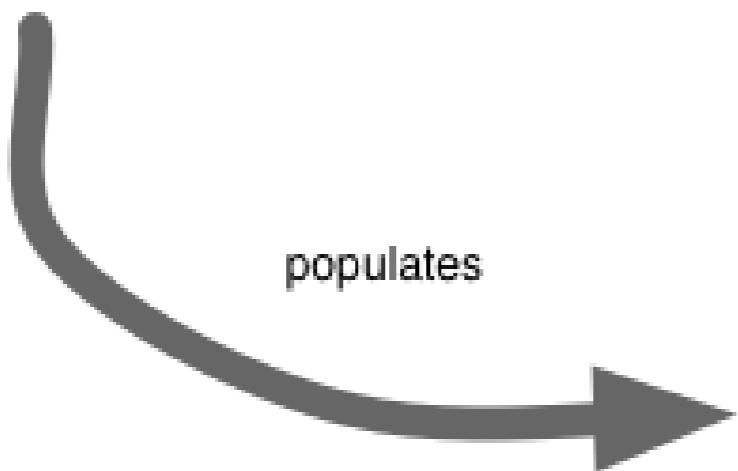
implementation

overview

```
events
  doorClosed  D1CL
end

state idle
  doorClosed => active
end
```

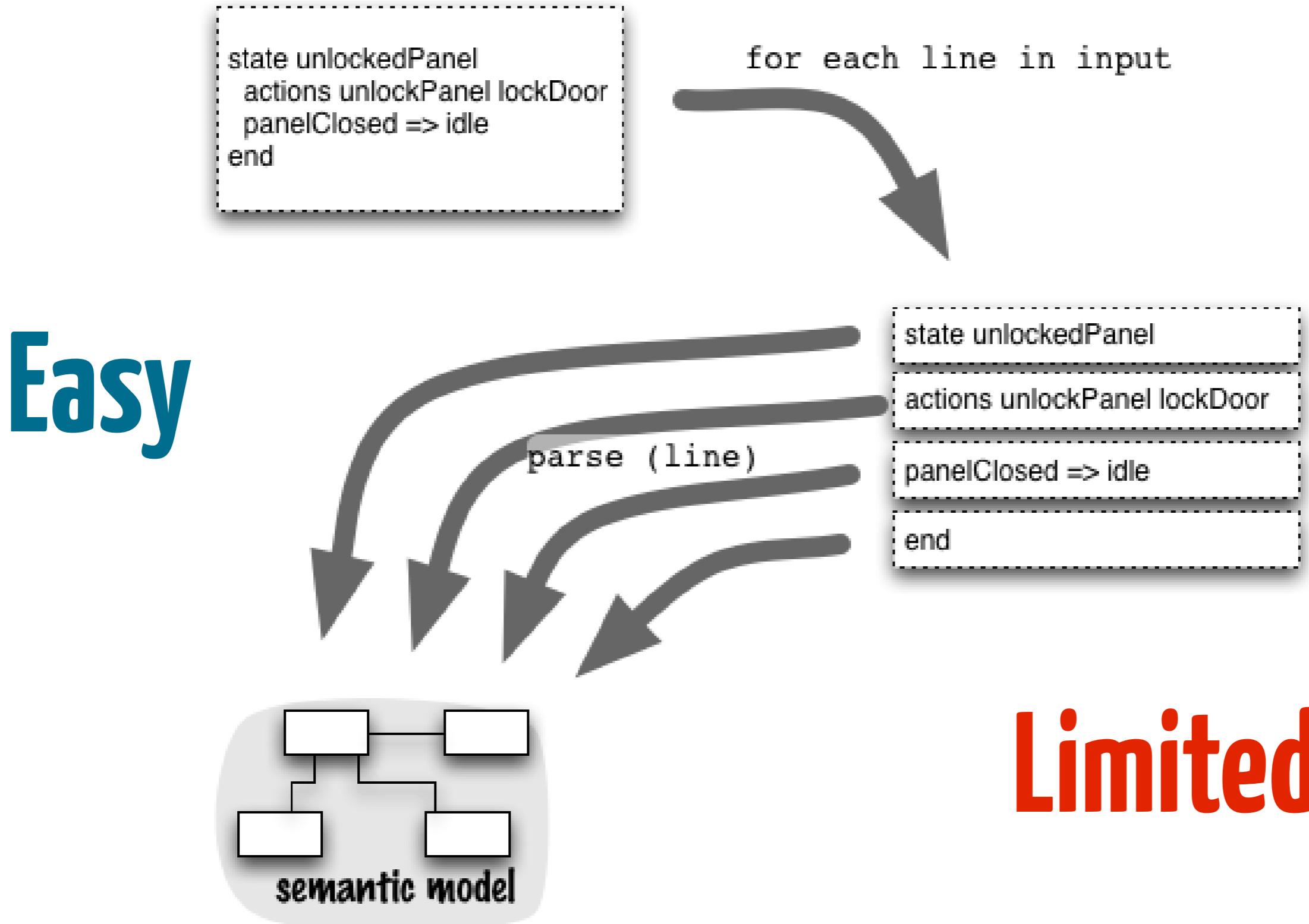
input



semantic model

[Domain-Specific Languages]

Delimiter-directed Translation



Syntax-directed Translation

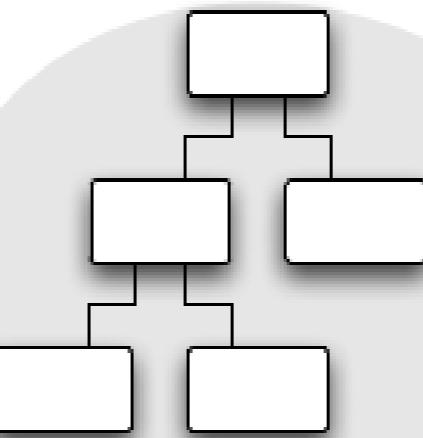
Input Text

```
events
  doorClosed D1CL
  drawOpened D2OP
end
```

```
declarations : eventBlock commandBlock;
eventBlock  : Event-keyword eventDec* End-keyword;
eventDec   : Identifier Identifier;
```

Grammar

Parser



Syntax Tree

Introducing Grammars

declarations : eventBlock commandBlock;

eventBlock : Event-keyword eventDec* End-keyword;

eventDec : Identifier Identifier;

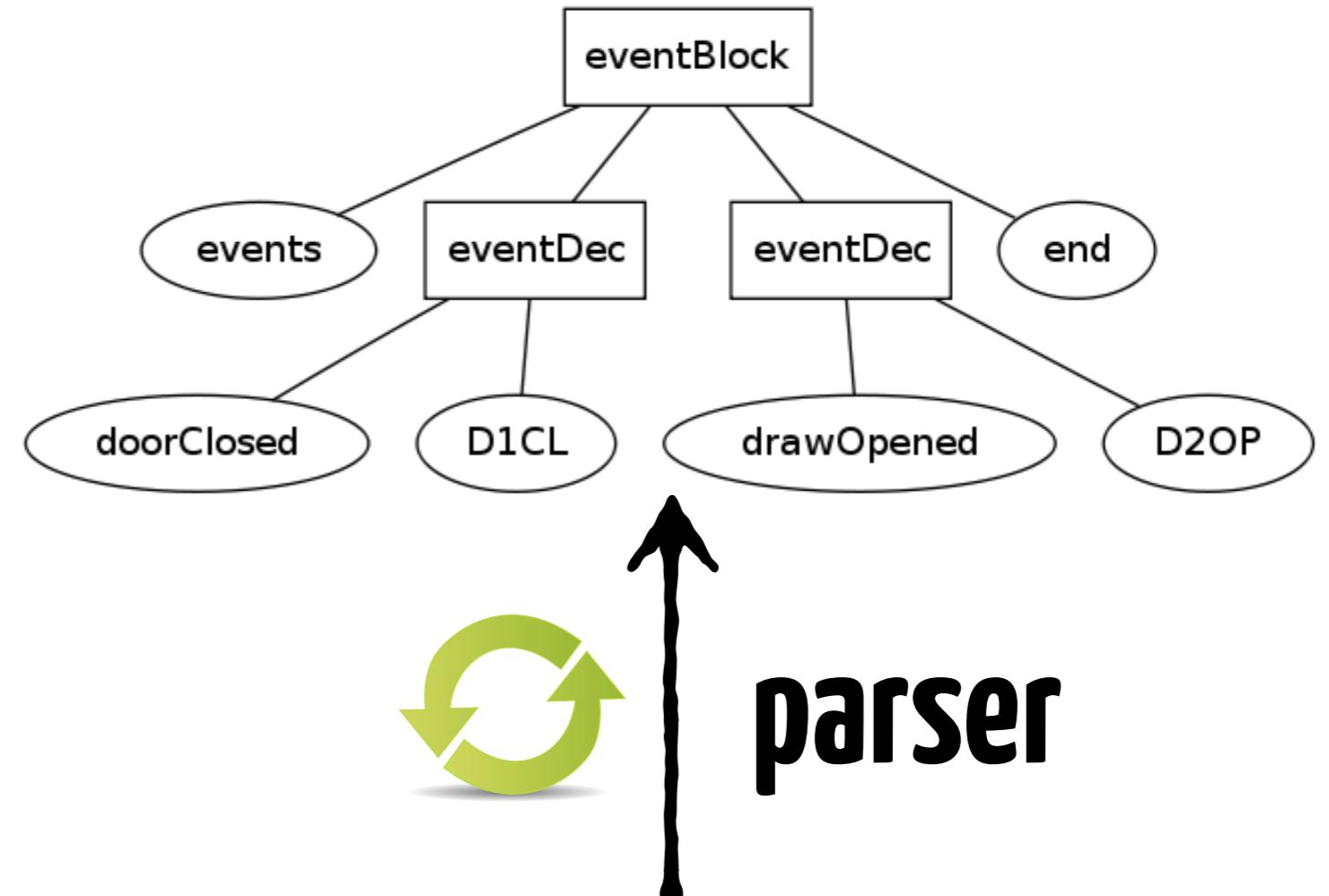
commandBlock : Command-keyword commandDec* End-keyword;

commandDec : Identifier Identifier;

This is not a
compilation
course!

From Words to Parse Tree

events
doorClosed D1CL
drawOpened D2OP
end



lexer



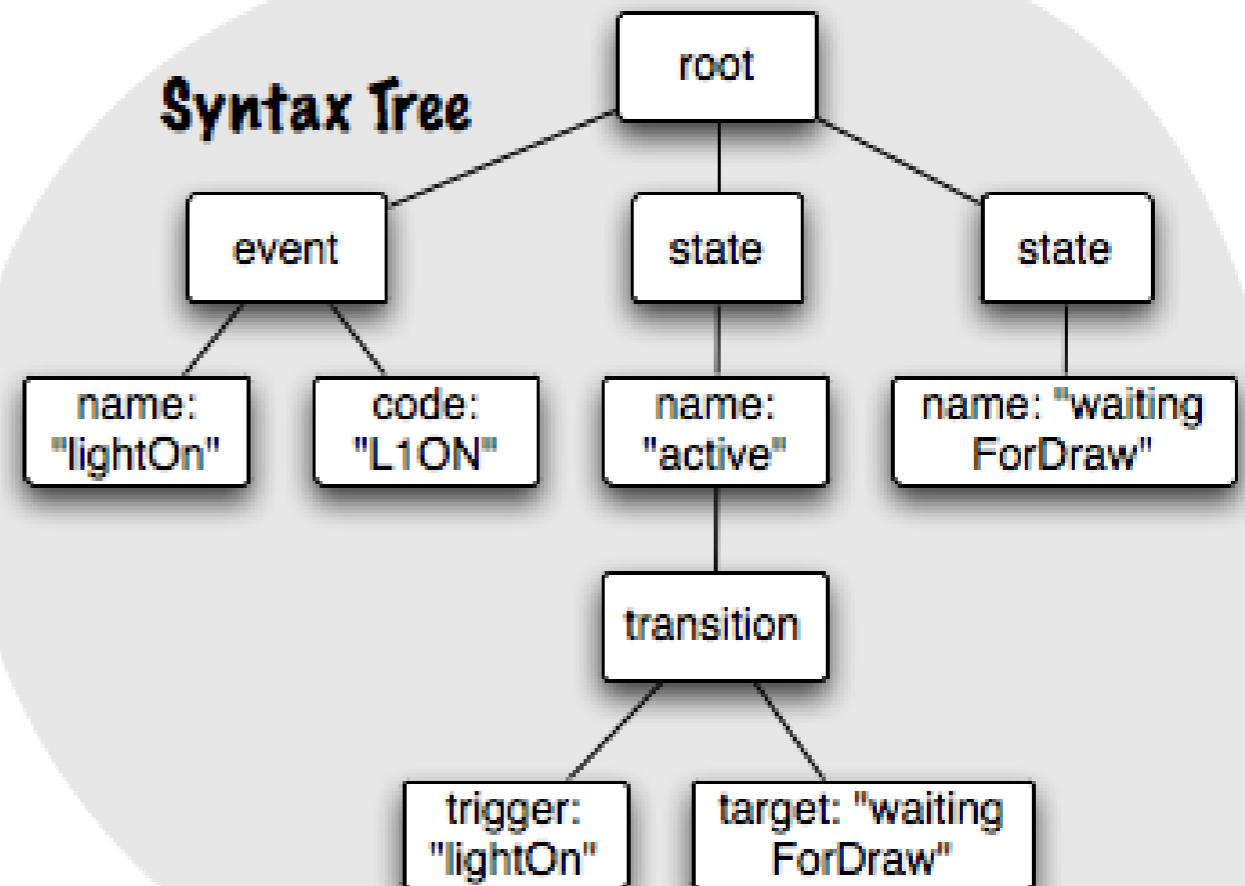
[Domain-Specific Languages]

[Event-keyword: "events"]
[Identifier: "doorClosed"]
[Identifier: "D1CL"]
[Identifier: "drawOpened"]
[Identifier: "D2OP"]
[End-keyword: "end"]

DSL Text

```
events
  lightOn L1ON
end
state active
  lightOn => waitingForDraw
end
state waitingForDraw end
```

Syntax Tree



:Transition

lightOn : Event
code = "L1ON"

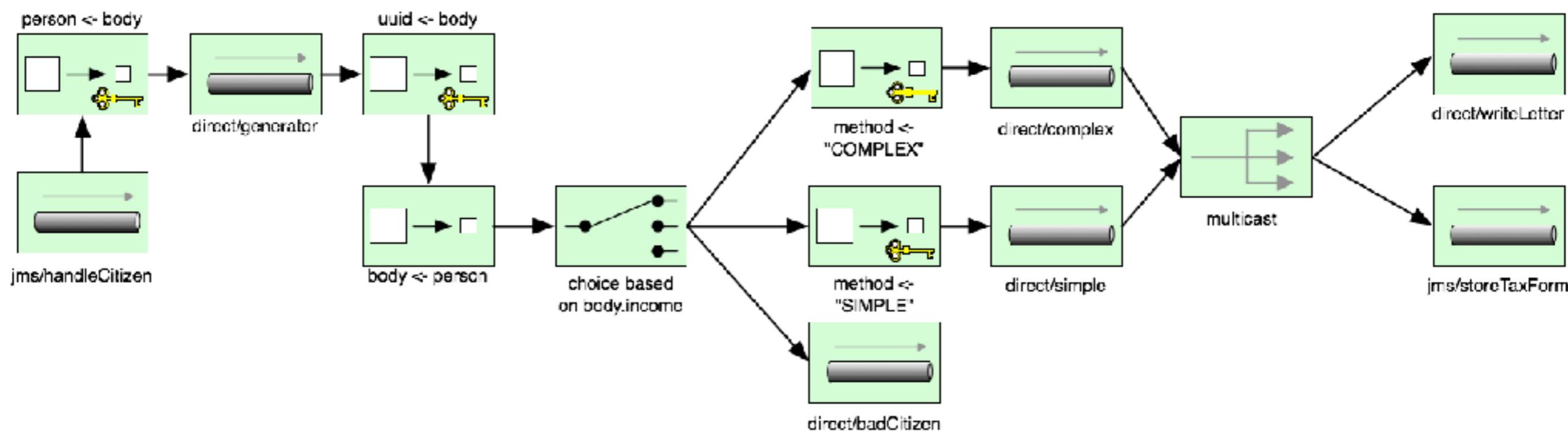
"L1ON"

active : State

waitingForDraw : State

Semantic Model

Graphical DSL



e.g., enterprise integration patterns



```
// Route to handle a given Person
from(HANDLE_CITIZEN)
    .log("      Routing ${body.lastName} according to income ${body.income}")
    .log("      Storing the Person as an exchange property")
    .setProperty("person", body())
    .log("      Calling an existing generator")
    .to("direct:generator")
    .setProperty("p_uuid", body())
    .setBody(simple("${property.person}"))
    .choice()
        .when(simple("${body.income} >= 42000"))
            .setProperty("tax_computation_method", constant("COMPLEX"))
            .to("direct:complexTaxMethod")
        .when(simple("${body.income} >= 0 && ${body.income} < 42000"))
            .setProperty("tax_computation_method", constant("SIMPLE"))
            .to("direct:simpleTaxMethod")
        .otherwise()
            .to("direct:badCitizen").stop() // stopping the route for bad citizens
    .end() // End of the content-based-router
    .setHeader("person_uid", simple("${property.person.uid}"))
    .multicast()
        .parallelProcessing()
        .to("direct:generateLetter")
        .to(STORE_TAX_FORM)
;

```

Embedded DSL

```
events
  doorClosed      D1CL
  drawerOpened      D2OP
  lightOn          L1ON
  doorOpened       D1OP
  panelClosed      PNCL
end
```

```
resetEvents
  doorOpened
end
```

```
commands
  unlockPanel  PNUL
  lockPanel     PNLK
  lockDoor      D1LK
  unlockDoor    D1UL
end
```

```
state idle
  actions {unlockDoor lockPanel}
  doorClosed => active
end

state active
  drawerOpened => waitingForLight
  lightOn => waitingForDrawer
end

state waitingForLight
  lightOn => unlockedPanel
end

state waitingForDrawer
  drawerOpened => unlockedPanel
end

state unlockedPanel
  actions {unlockPanel lockDoor}
  panelClosed => idle
end
```

```
events
  doorClosed      D1CL
end
```

```
commands
  lockPanel      PNLK
  unlockDoor    D1UL
end
```

```
state idle
  actions {unlockDoor lockPanel}
  doorClosed => active
end
```

```
state active
  ...
end
```

```
events
```

```
  doorClosed      D1CL
```

```
end
```

```
commands
```

```
  lockPanel      PNLK
```

```
  unlockDoor    D1UL
```

```
end
```

```
state idle
```

```
  actions {unlockDoor lockPanel}
```

```
  doorClosed => active
```

```
end
```

```
state active
```

```
  ...
```

```
end
```

```
event :doorClosed, "D1CL"
```

```
command :lockPanel, "PNLK"
```

```
command :unlockDoor, "D1UL"
```

```
state :idle do
```

```
  actions :unlockDoor, :lockPanel
```

```
  transitions :doorClosed => :active
```

```
end
```

```
state :active do
```

```
  ...
```

```
end
```



```
event :doorClosed, "D1CL"

command :lockPanel, "PNLK"
command :unlockDoor, "D1UL"

state :idle do
  actions :unlockDoor, :lockPanel
  transitions :doorClosed => :active
end

state :active do
  ...
end
```

```
main :: IO ()  
main = either print print . fmap generate $ buildApp "example" $ do  
    addSensor button $ onPin 9  
    addActuator light $ onPin 12  
    defineStates [offline, online]  
    actionsWhen offline `execute` [ set light `to` off ]  
    actionsWhen online `execute` [ set light `to` on ]  
    transitionsFrom online `are` [ when button `is` pressed $ goto offline ]  
    transitionsFrom offline `are` [ when button `is` pressed $ goto online ]  
    start offline
```

