22c181:
Formal Methods in Software Engineering

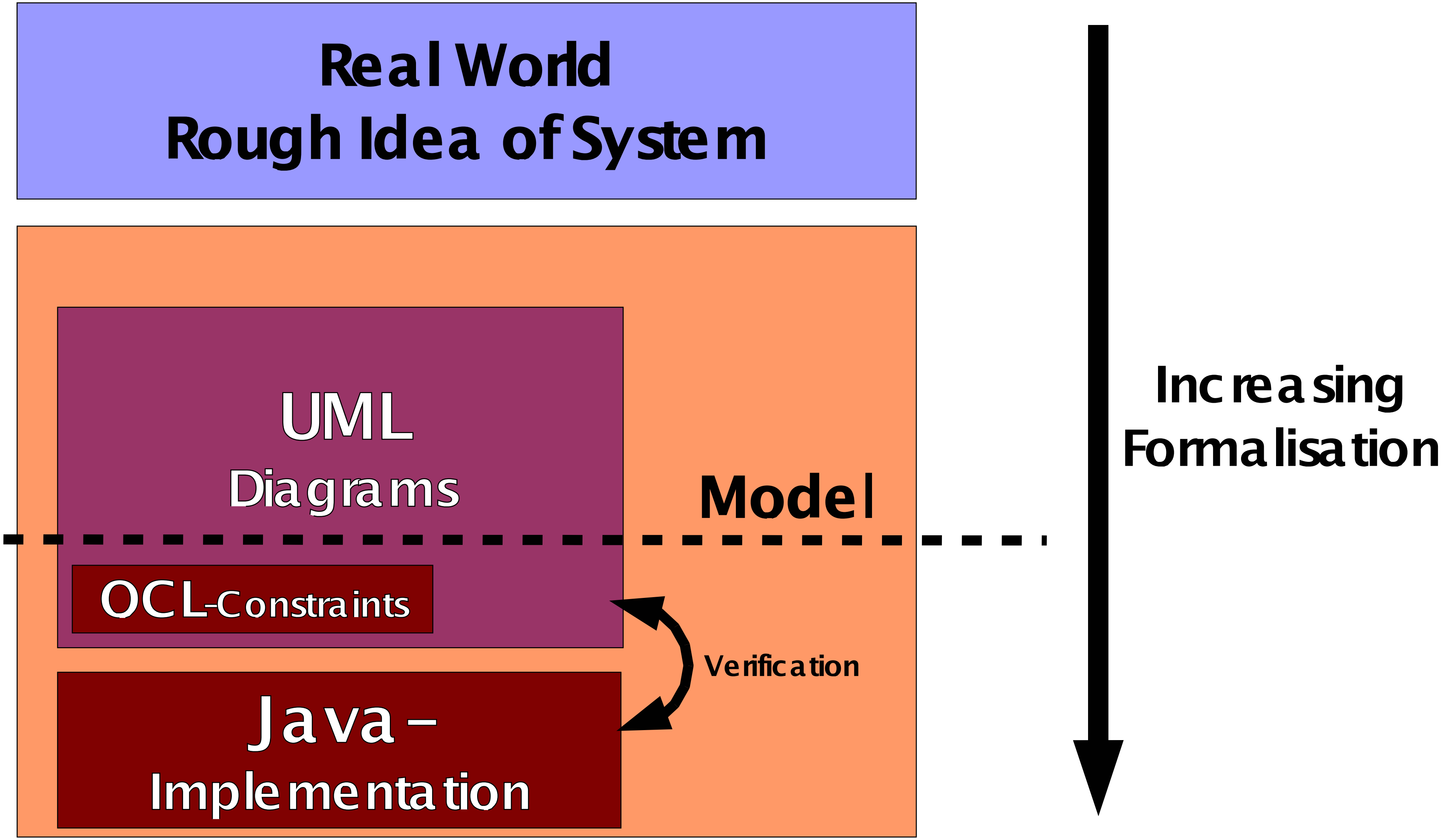
The University of Iowa

Spring 2008

Introduction to UML

Contents

Building Models



UML

Unified Modeling Language

- **Unified: end to many similar approaches.**

  Booch, Rumbaugh, Jacobsson

  Standardised by OMG (now version 2.0 in finalisation)

UML

Unified Modeling Language

- **Unified:** end to many similar approaches.

  Booch, Rumbaugh, Jacobsson

  Standardised by OMG (now version 2.0 in finalisation)

- **Modeling:** main (creative) process of software development

  Trend in SWE: emphasis on model, MDA/MDE

  Code abstraction, formal model

UML

Unified Modeling Language

- **Unified:** end to many similar approaches.

  Booch, Rumbaugh, Jacobsson

  Standardised by OMG (now version 2.0 in finalisation)

- **Modeling:** main (creative) process of software development

  Trend in SWE: emphasis on model, MDA/MDE

  Code abstraction, formal model

- **Language:** Provides notation, no method, no process

  Graphical, collection of different diagram types

UML Diagrams

- **Structural Diagrams**
- **Behavioural Diagrams**

UML Diagrams

- **Structural Diagrams**
  - **Class Diagrams**
  - **Component Diagrams**
  - **Composite Structure Diagrams**
  - **Object Diagrams**
  - **Deployment Diagrams**
  - **Package Diagrams**
- **Behavioural Diagrams**

UML Diagrams

- **Structural Diagrams**
  - **Class Diagrams**
  - **Component Diagrams**
  - **Composite Structure Diagrams**
  - **Object Diagrams**
  - **Deployment Diagrams**
  - **Package Diagrams**
- **Behavioural Diagrams**

UML Diagrams

- **Structural Diagrams**
  - **Class Diagrams**
  - **Component Diagrams**
  - **Composite Structure Diagrams**
  - **Object Diagrams**
  - **Deployment Diagrams**
  - **Package Diagrams**
- **Behavioural Diagrams**

UML Diagrams

- **Structural Diagrams**
  - Class Diagrams
  - Component Diagrams
  - Composite Structure Diagrams
  - Object Diagrams
  - Deployment Diagrams
  - Package Diagrams
- **Behavioural Diagrams**
  - Activity Diagrams
  - Interaction Diagrams
    - Sequence Diagrams
    - Collaboration Diagrams
    - ⋮
  - Use Case Diagrams
  - State Machine Diagrams

UML Diagrams

- Structural Diagrams
  - Class Diagrams
  - Component Diagrams
  - Composite Structure Diagrams
  - Object Diagrams
  - Deployment Diagrams
  - Package Diagrams
- Behavioural Diagrams
  - Activity Diagrams
  - Interaction Diagrams
    - Sequence Diagrams
    - Collaboration Diagrams
    - ⋮
  - Use Case Diagrams
  - State Machine Diagrams

UML Diagrams

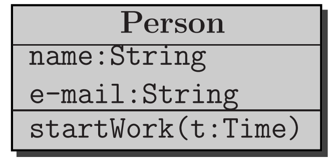
- Structural Diagrams
  - Class Diagrams
  - Component Diagrams
  - Composite Structure Diagrams
  - Object Diagrams
  - Deployment Diagrams
  - Package Diagrams
- Behavioural Diagrams
  - Activity Diagrams
  - Interaction Diagrams
    - Sequence Diagrams
    - Collaboration Diagrams
    - ⋮
  - Use Case Diagrams
  - State Machine Diagrams

Class Diagrams

Model static design view, define vocabulary (signature)

Class

Collection of similar objects in a system



Attributes

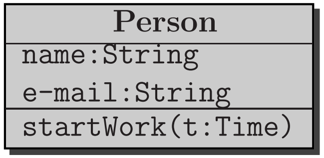
Operations/Methods

Class Diagrams

Model static design view, define vocabulary (signature)

Class

Collection of similar objects in a system

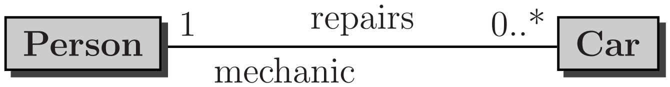


Attributes **Operations/Methods**

Association

Relation between classes

Relates pairs of class instances

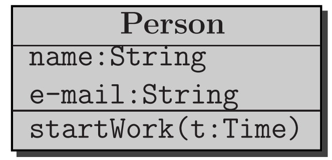


Class Diagrams

Model static design view, define vocabulary (signature)

Class

Collection of similar objects in a system

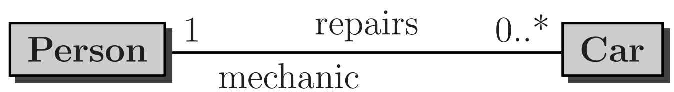


Attributes **Operations/Methods**

Association

Relation between classes

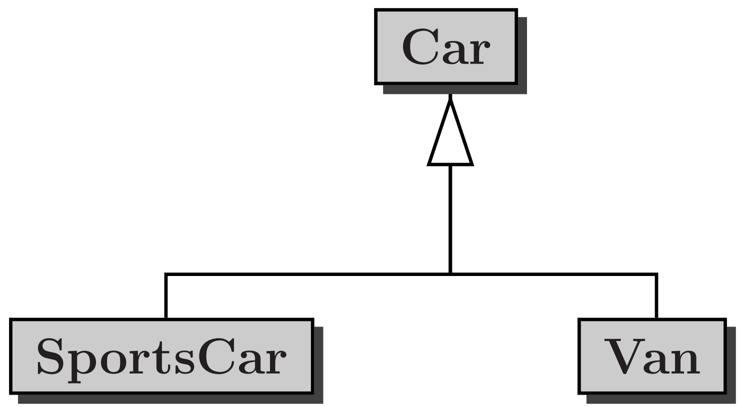
Relates pairs of class instances



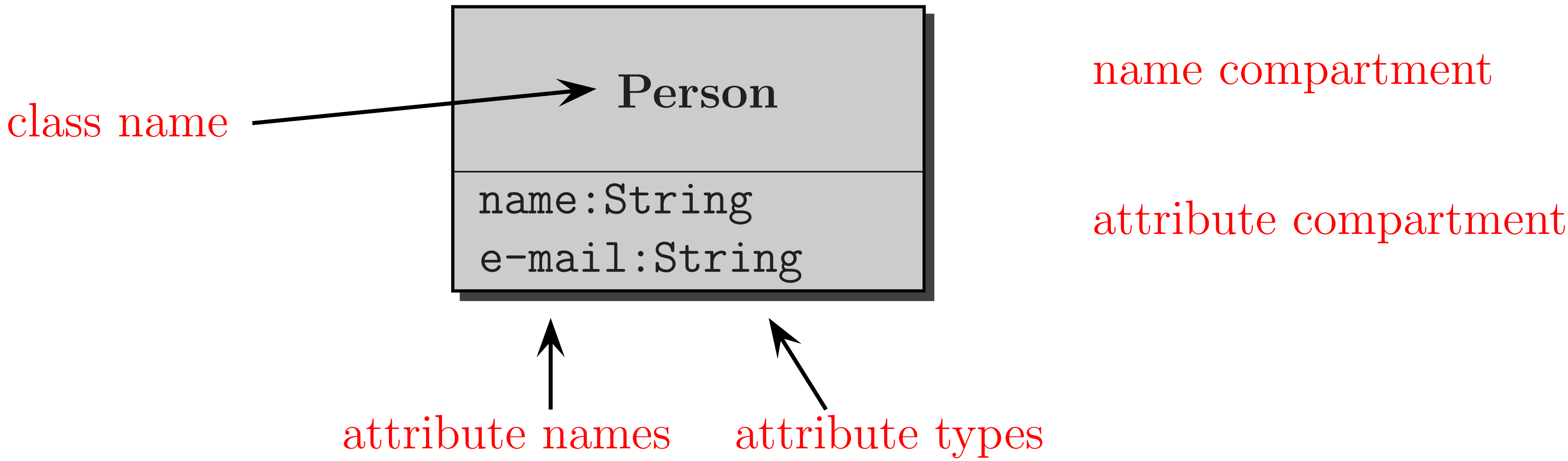
Generalisation/Inheritance

Specialisation-/Generalisation

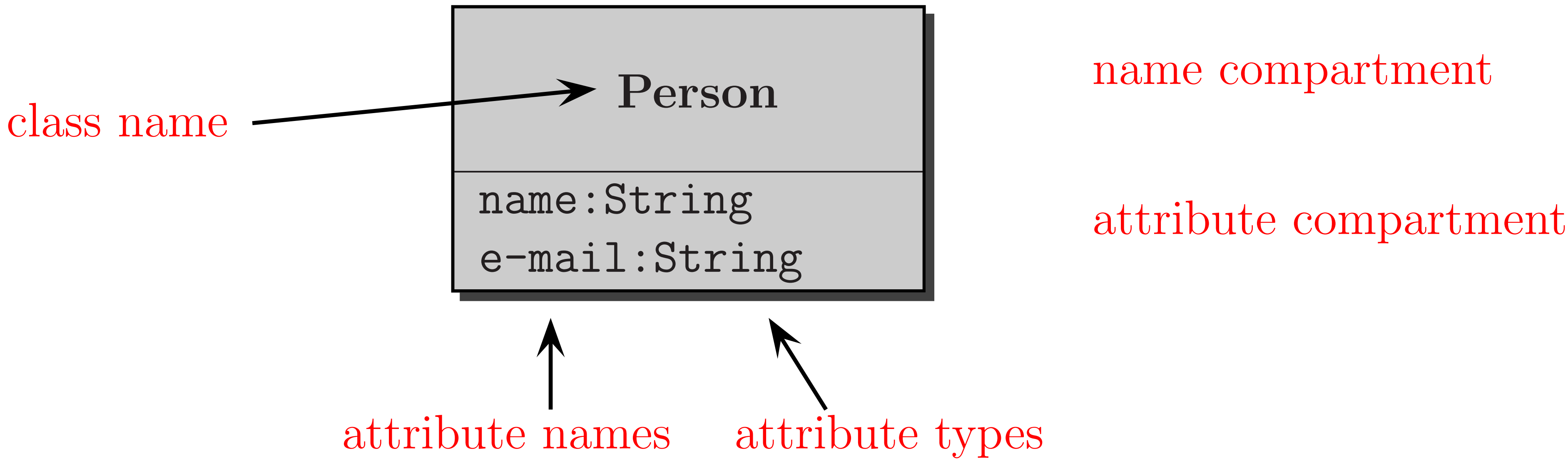
relationship between classes



Classes


Person
name:String
e-mail:String
class name
name compartment
attribute compartment
attribute names
attribute types

Classes



(JAVA) Semantics of Classes

For class C let $I(\mathrm{C}) \neq \emptyset$ be set of objects

"The objects that can have static type C"

The Null Type

The Null Type

Semantics of Null

Each class diagram contains implicitly the Null class

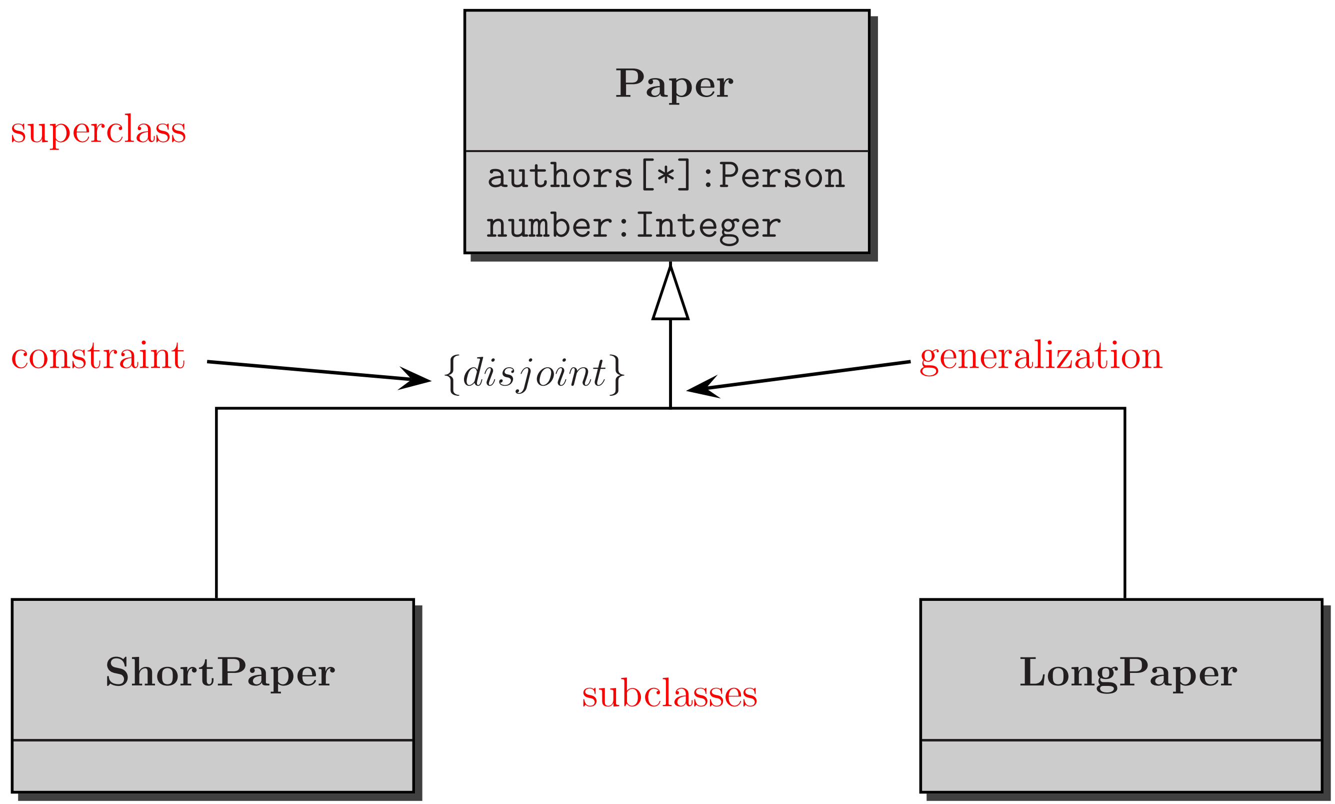
No attributes, no operations

$I(\mathrm{Null}) = \{\texttt{null}\}$

$\texttt{null} \in I(\mathrm{C})$ **for any class** C

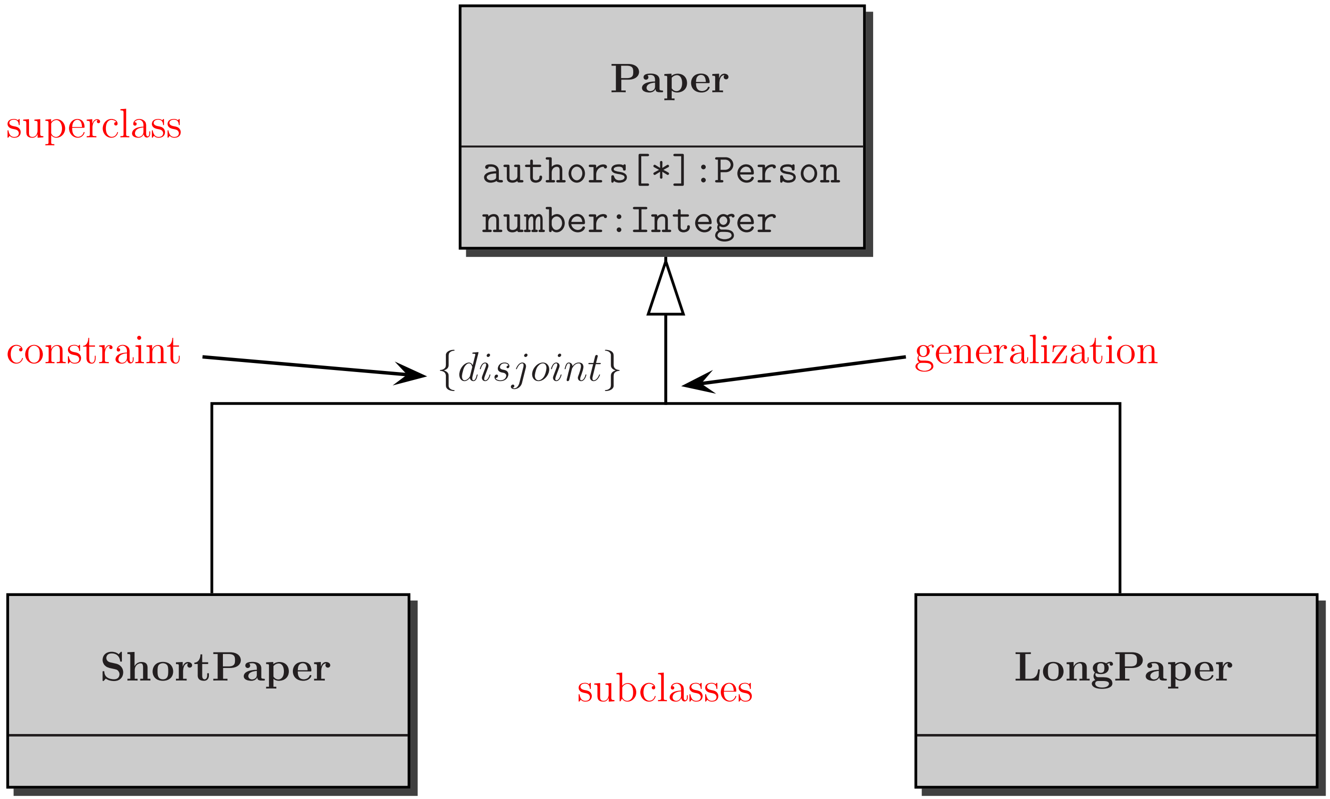
"$\texttt{null}$ **is typeable with any type** C**"**

Subclasses



superclass
Paper
authors[*]:Person
number:Integer
constraint
{disjoint}
generalization
ShortPaper
subclasses
LongPaper

Subclasses

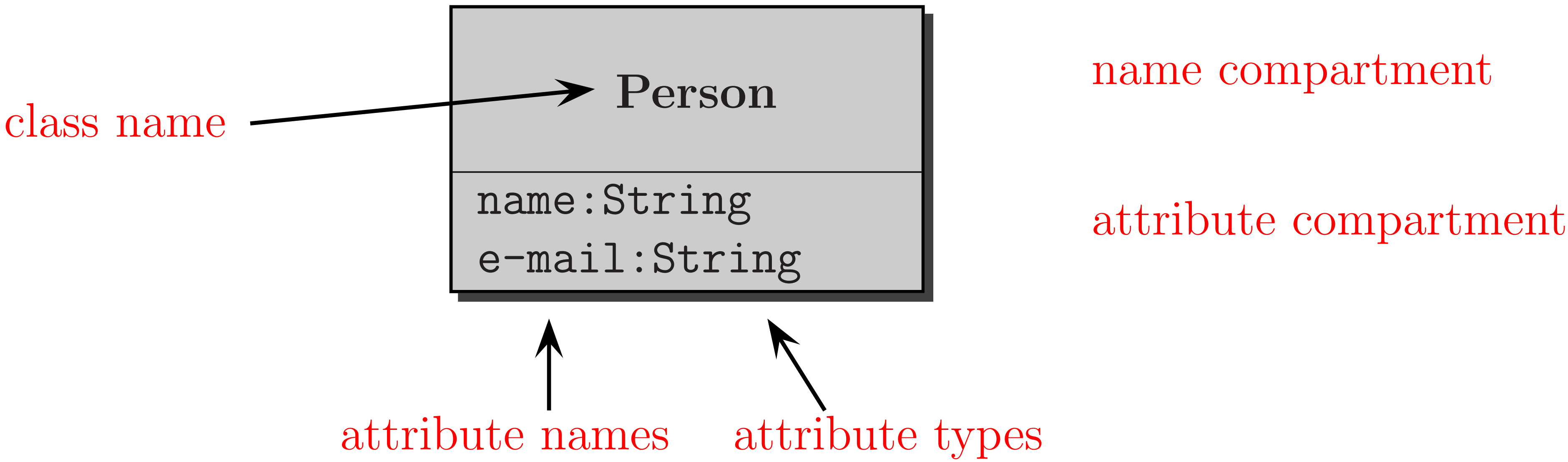


Semantics of Subclasses

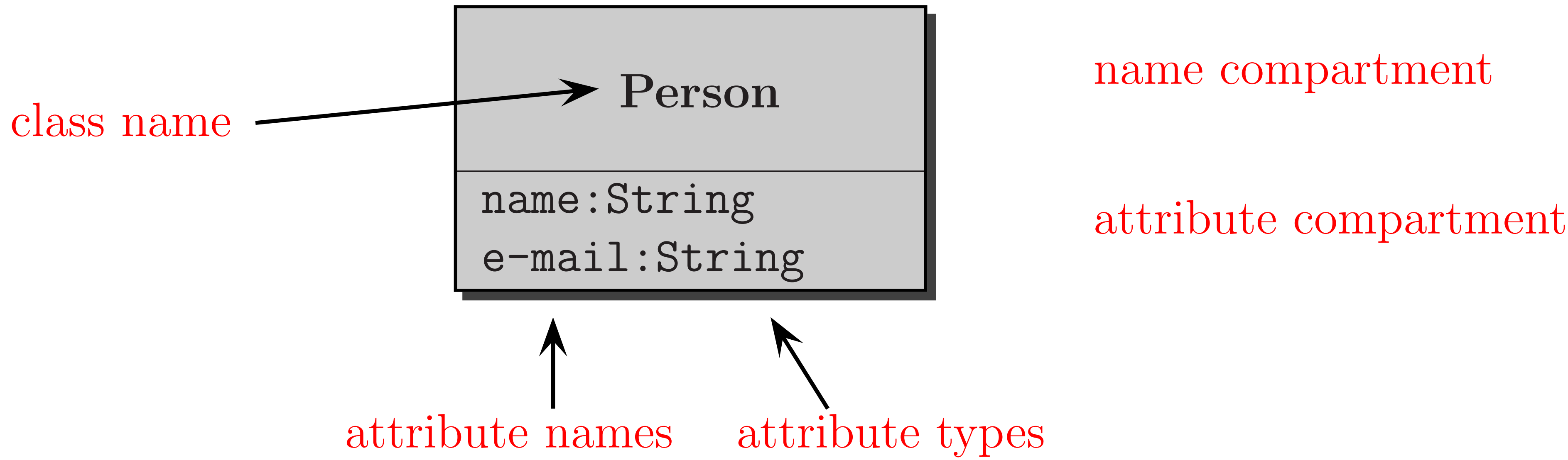
subclass relation: $I(\text{ShortPaper}) \subseteq I(\text{Paper})$

constraint: $I(\text{ShortPaper}) \cap I(\text{LongPaper}) = \{\texttt{null}\}$

Attributes


class name
Person
name:String
e-mail:String
name compartment
attribute compartment
attribute names
attribute types

Attributes



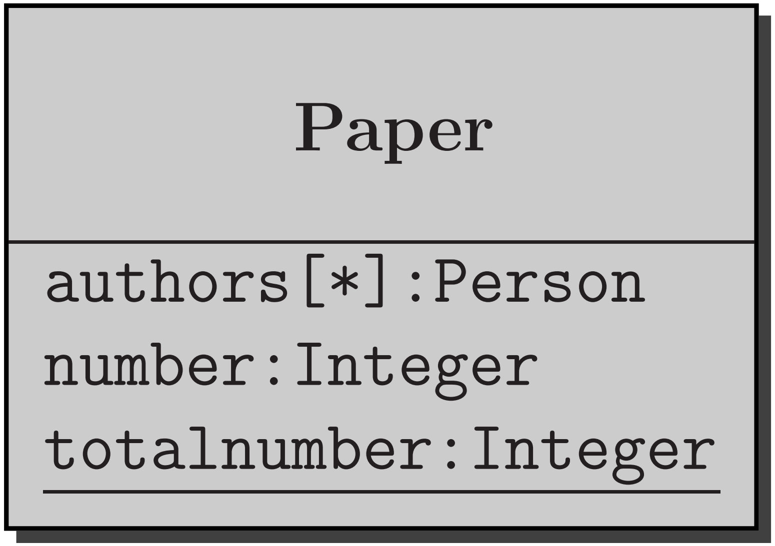
Semantics of Attributes

$I(\texttt{name})$ **is function from** $I(\mathrm{Person})$ **to** $I(\mathrm{String})$

$I(\texttt{name})(\underline{aPerson})$ **gives a string or** **`null`**

Always $f(\texttt{null}) = \texttt{null}$

Class (Static) Attributes



Class (Static) Attributes

Paper
authors[*]:Person number:Integer <u>totalnumber:Integer</u>

Semantics

$I(\texttt{totalnumber})$ **is an element of (not a function to)** $I(\texttt{Integer})$

(i.e., $\mathrm{Paper}.\texttt{totalnumber}$ **is a constant)**

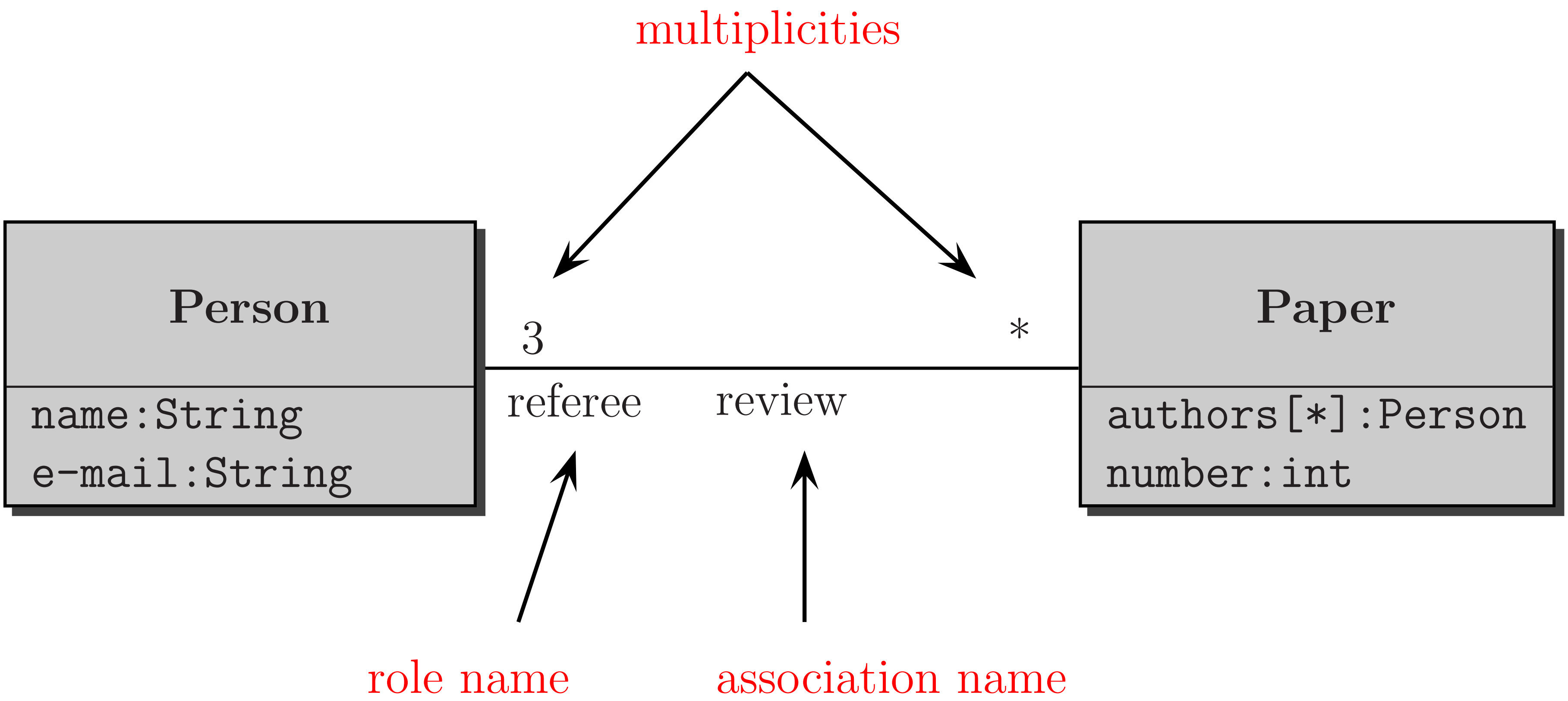
Multiplicities

Semantics

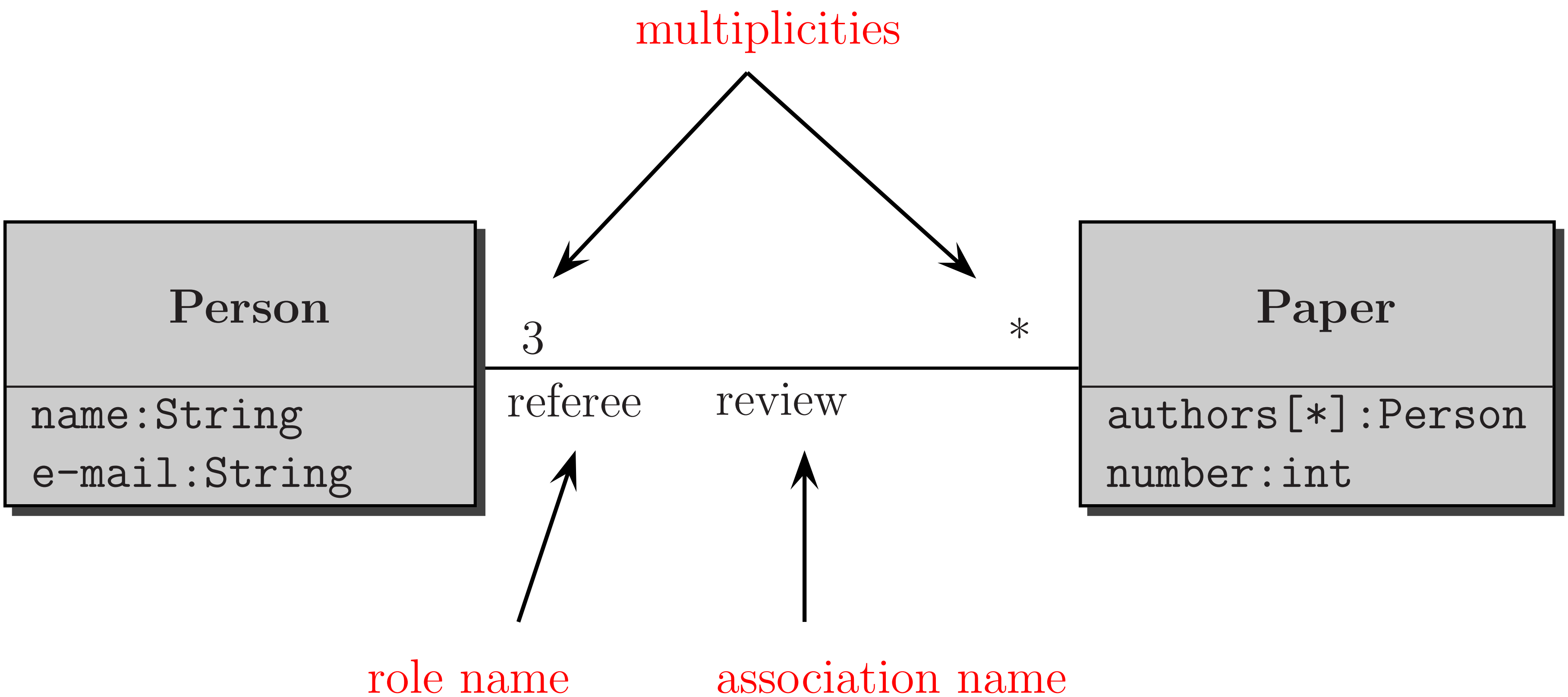
M	$I(M)$
$0..1$	$\{0, 1\}$
$0..*$	$\mathbb{N}$
$*$	$\mathbb{N}$
$1..3$	$\{1, 2, 3\}$
7	$\{7\}$
$15..19$	$\{15, 16, 17, 18, 19\}$
$1..3, 7, 15..19$	$\{1, 2, 3, 7, 15, 16, 17, 18, 19\}$

(i.e., the separator "," acts as set theoretic union)

Associations



Associations

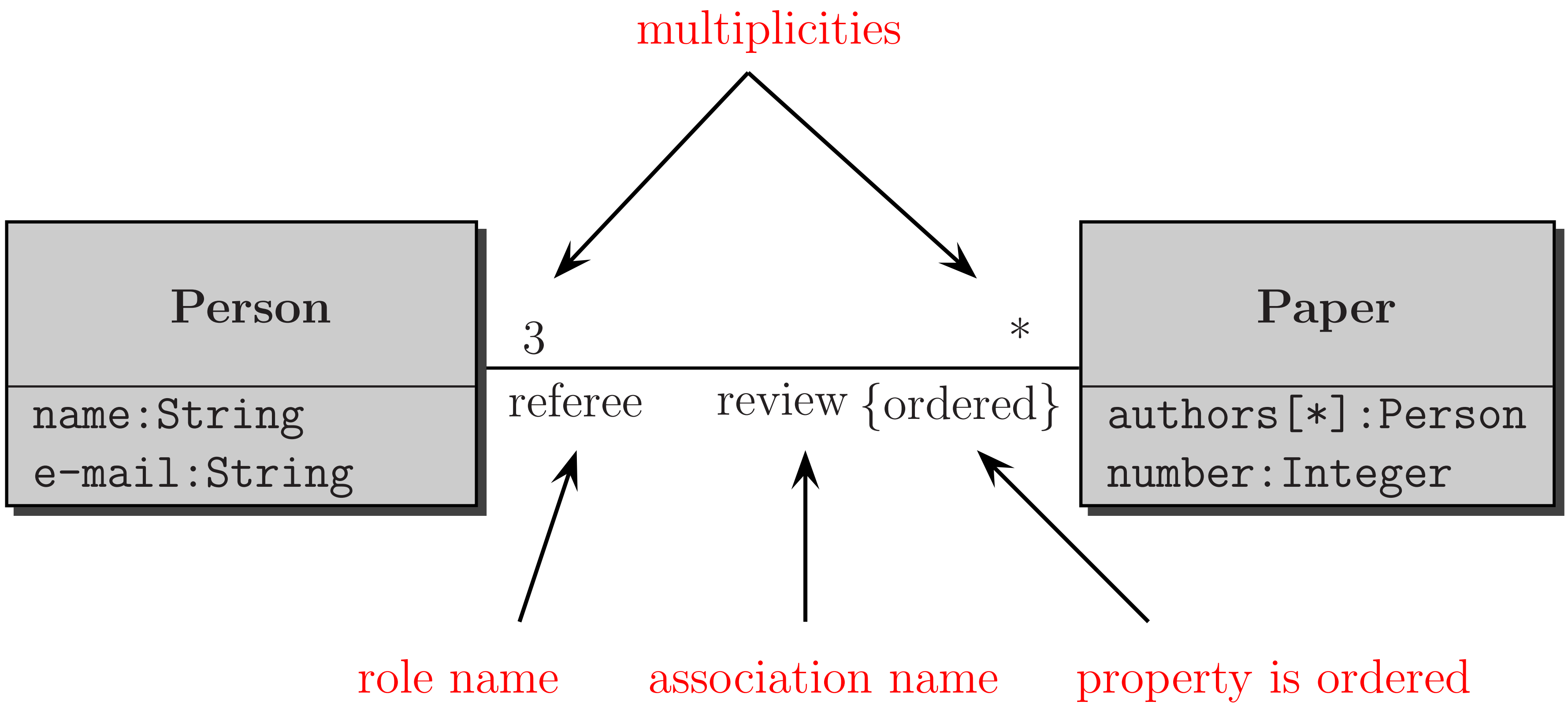


Semantics

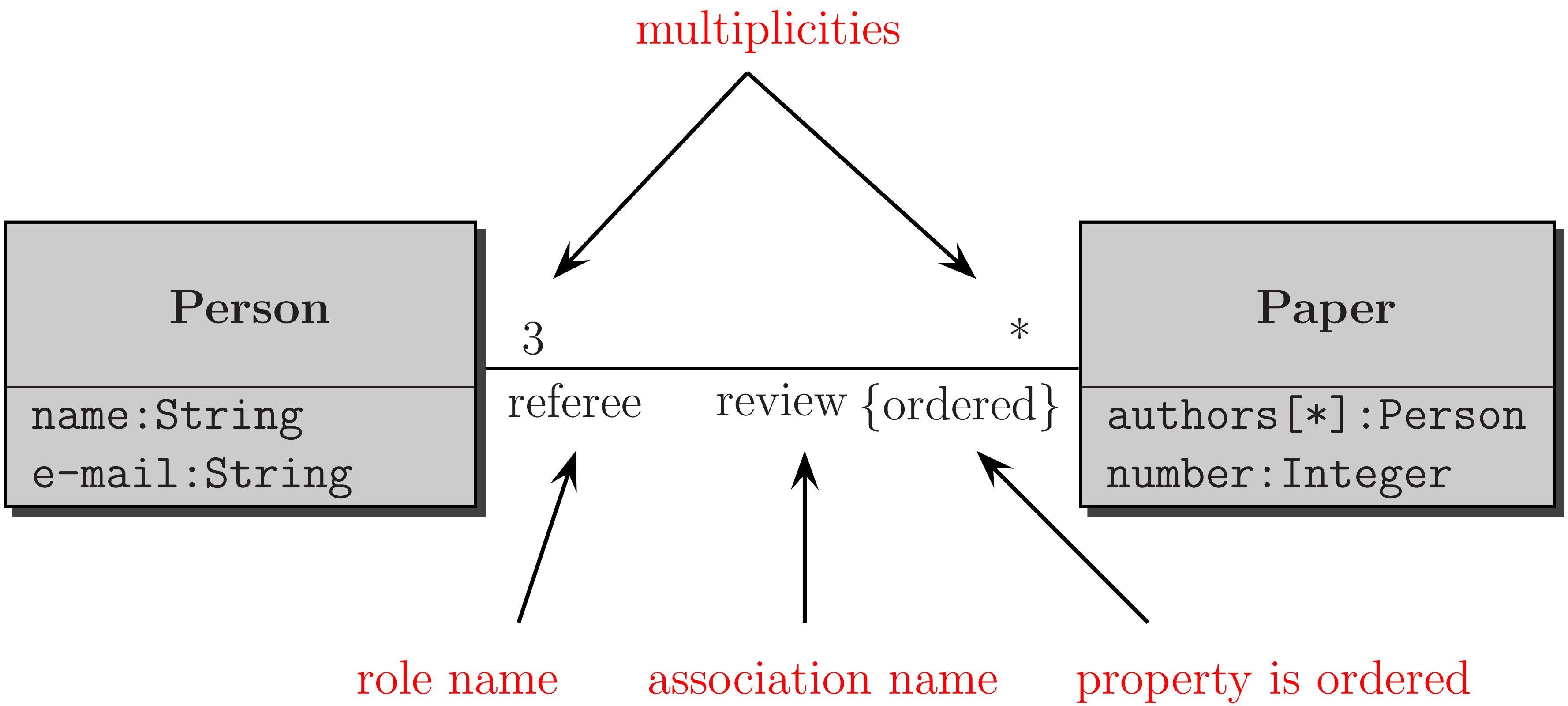
$I(\text{review})$ **is a relation between** $I(\text{Person}) \setminus \{\texttt{null}\}$ **and** $I(\text{Paper}) \setminus \{\texttt{null}\}$

Multiplicity 3 **requires: for all** $pap \in I(\text{Paper})$**,**
$card(\{pers \in I(\text{Person}) \mid \text{review}(pers, pap)\}) = 3$

Role Names (Directed Associations)



Role Names (Directed Associations)

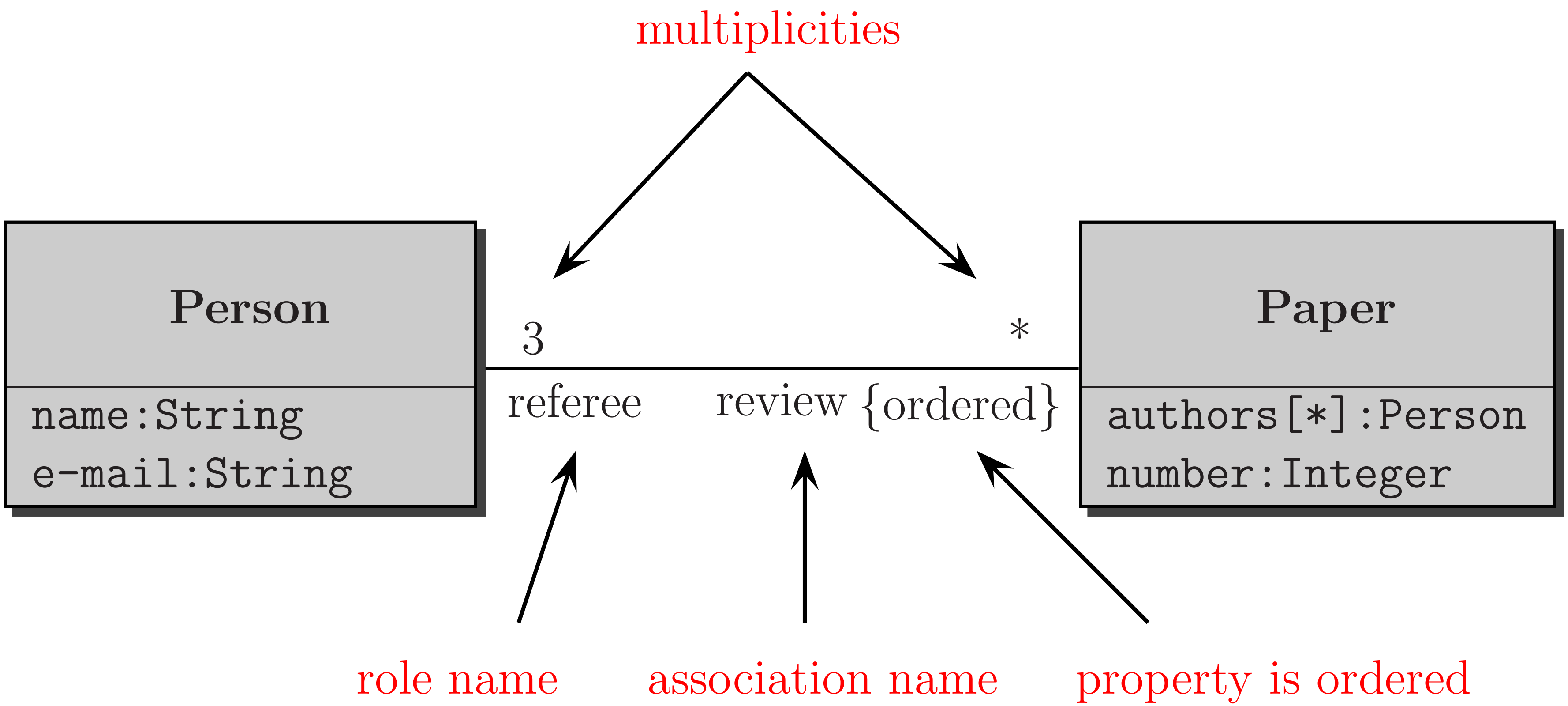


Semantics

$$I(\text{referee}) : \overbrace{I(\text{Paper})}^{\text{Client}} \rightarrow Set(\overbrace{I(\text{Person})}^{\text{Supplier}})$$ **(supplier multiplicity $\neq$ 1)**

$$I(\text{paper}) : I(\text{Person}) \rightarrow Sequence(I(\text{Paper}))$$ **(default role name $=$ client)**

Role Names Cont'd



Semantics of role names compatible with association semantics

$$I(\text{referee})(\underline{aPaper}) = \{\underline{aPerson} \mid \langle \underline{aPerson}, \underline{aPaper} \rangle \in I(\text{review})\}$$

Snapshots

A snapshot of a given class diagram $\mathcal{D}$ is a particular semantics I of $\mathcal{D}$

- UML object diagram (for $\mathcal{D}$) including
  - for each class C: objects $I(\mathrm{C})$ typeable with C
  - maps $I(\mathtt{a}) : I(\mathrm{C}) \to I(\mathrm{C}')$ for all attributes $\mathtt{a}$ of type C' in class C
  - association instances (pairs) in $I(\mathrm{C})\backslash\{\mathtt{null}\} \times I(\mathrm{C}')\backslash\{\mathtt{null}\}$
- an interpretation for operations/methods (Java: independent of snapshot)
- (standard) interpretation of predefined primitive data types and their operations (Integer, String, . . .)

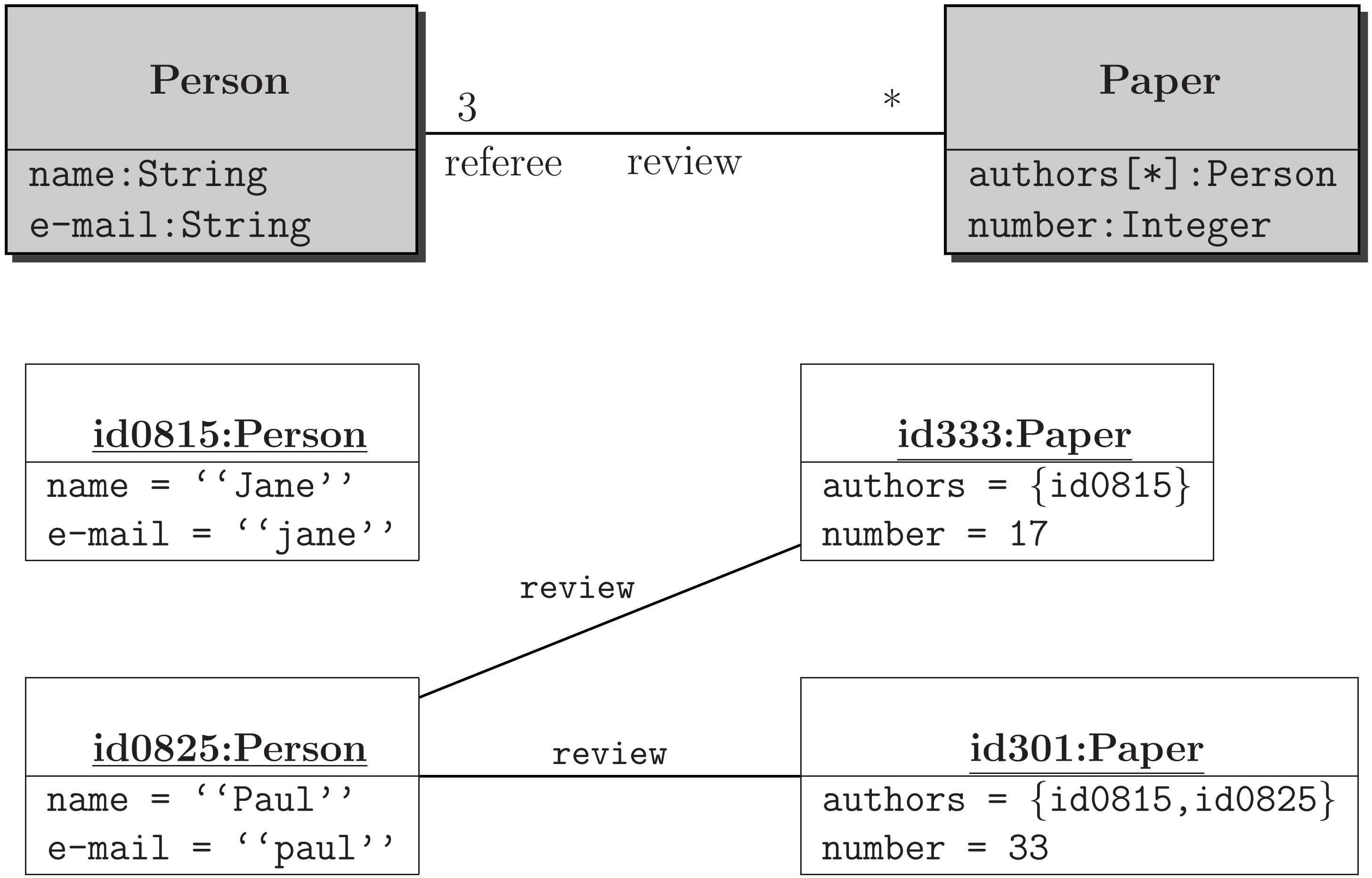
Snapshots

A snapshot of a given class diagram $\mathcal{D}$ is a particular semantics I of $\mathcal{D}$

- UML object diagram (for $\mathcal{D}$) including
  - for each class C: objects $I(\mathrm{C})$ typeable with C
  - maps $I(\mathrm{a}) : I(\mathrm{C}) \to I(\mathrm{C}')$ for all attributes a of type C' in class C
  - association instances (pairs) in $I(\mathrm{C})\backslash\{\mathtt{null}\} \times I(\mathrm{C}')\backslash\{\mathtt{null}\}$
- an interpretation for operations/methods (Java: independent of snapshot)
- (standard) interpretation of predefined primitive data types and their operations (Integer, String, . . .)

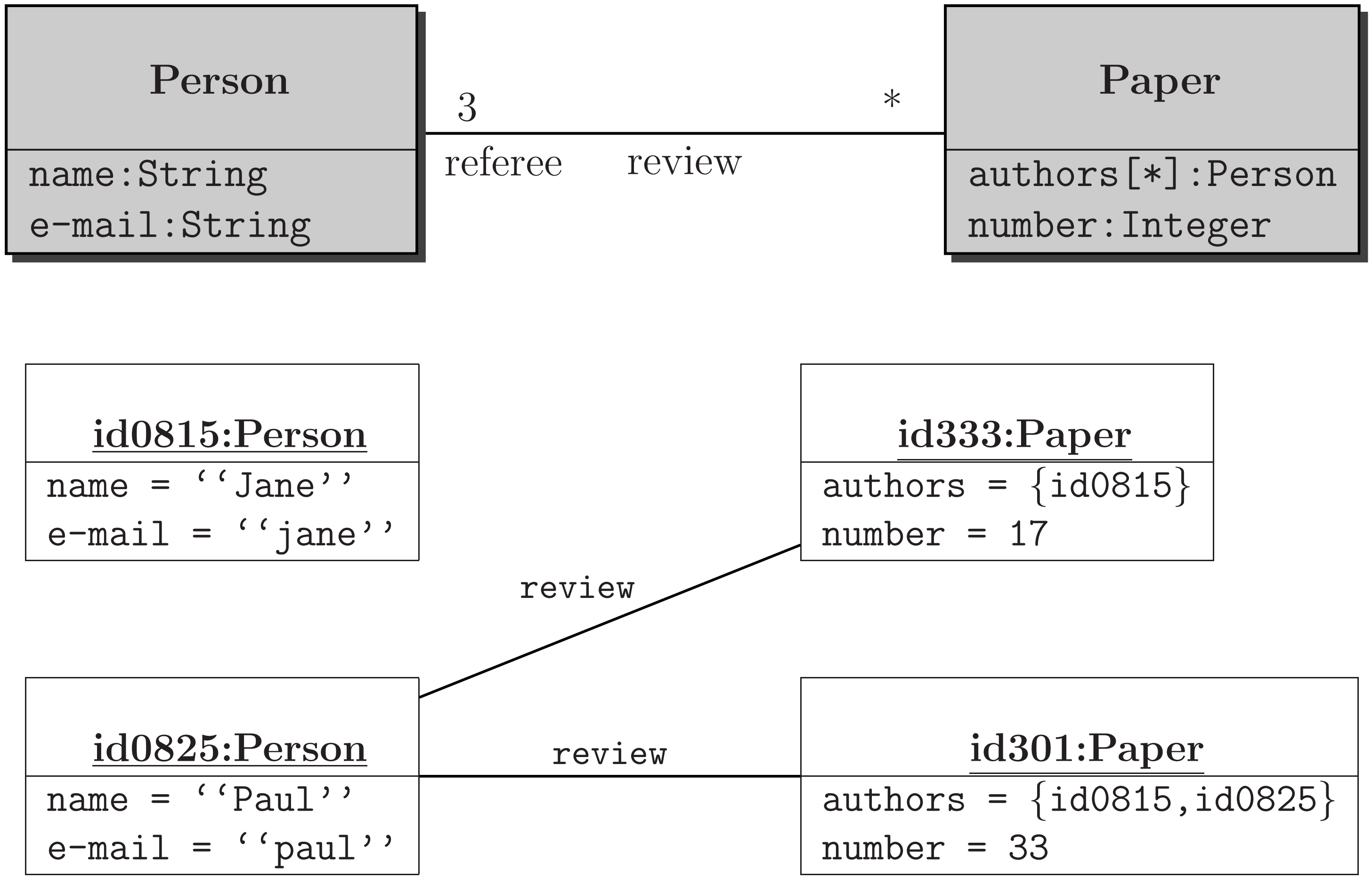
Multiplicities and constraints restrict set of admissible snapshots

Object Diagram: (Static Part of) Snapshot



Non-admissible (and unintended) instance — Why?

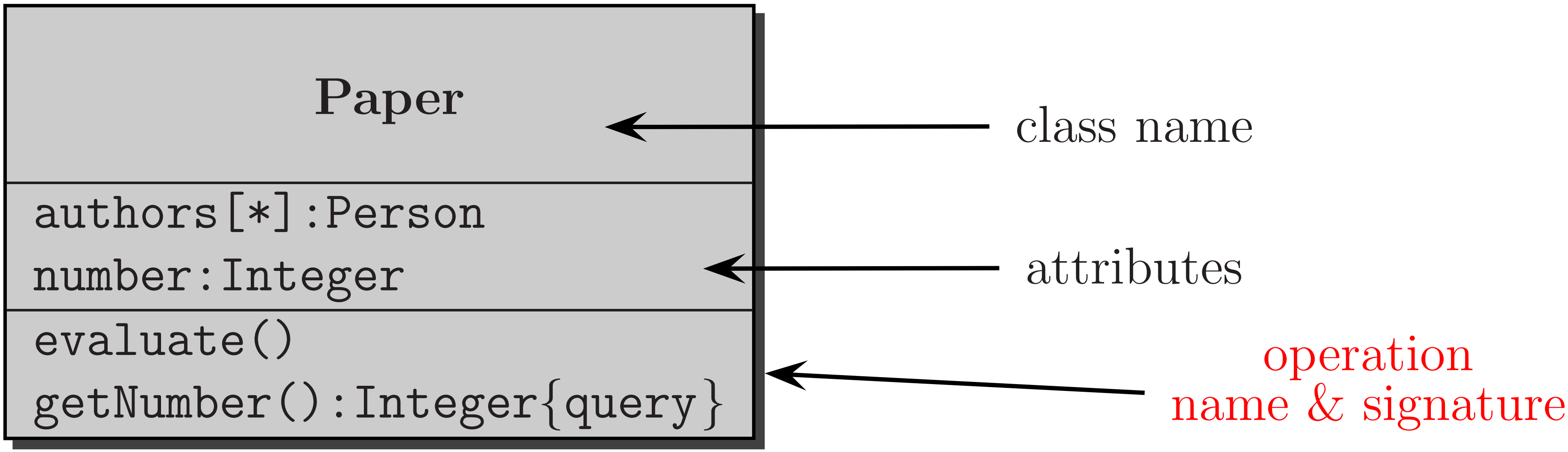
Object Diagram: (Static Part of) Snapshot



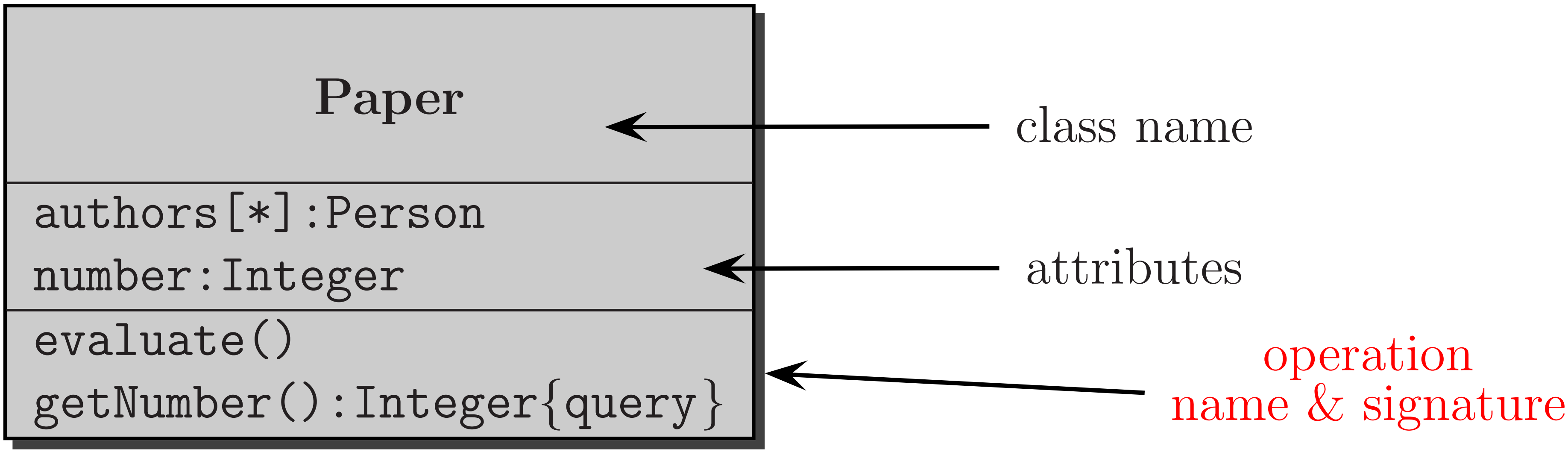
Non-admissible (and unintended) instance — Why?

How to exclude that the author of a paper is its own reviewer?

Operations: Queries



Operations: Queries

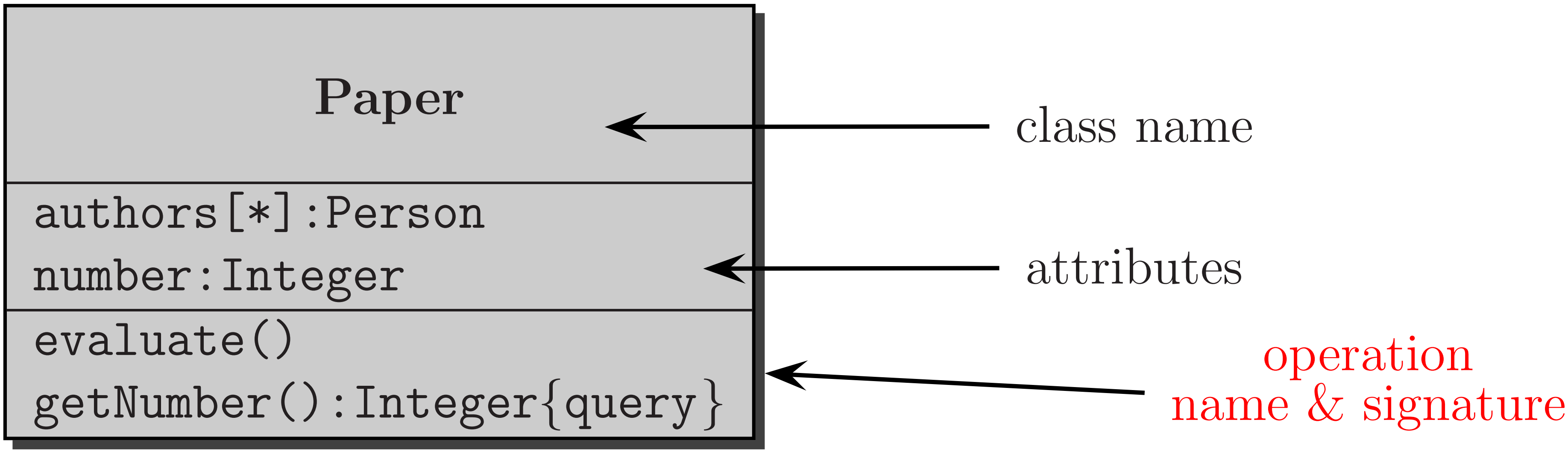


Semantics of queries (side-effect free operations, aka pure methods)

Function from owner and parameter classes to result class

$$I(\texttt{getNumber()}) \; : \; I(\overbrace{\text{Paper}}^{\text{Owner}}) \to I(\texttt{Integer})$$

Operations: Queries



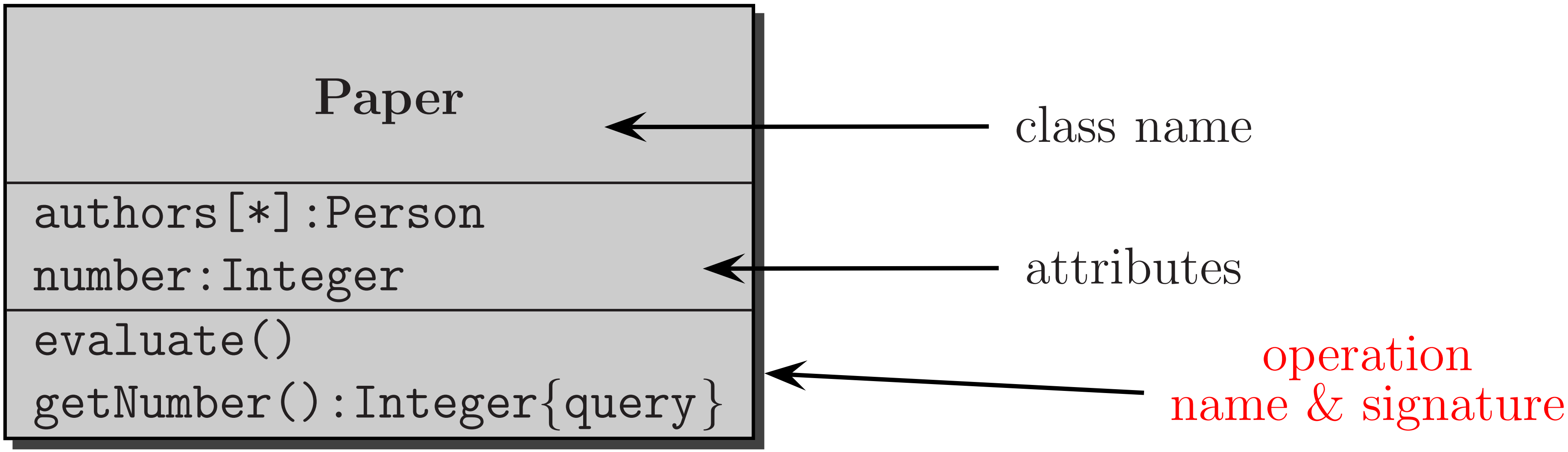
Semantics of queries (side-effect free operations, aka pure methods)

Function from owner and parameter classes to result class

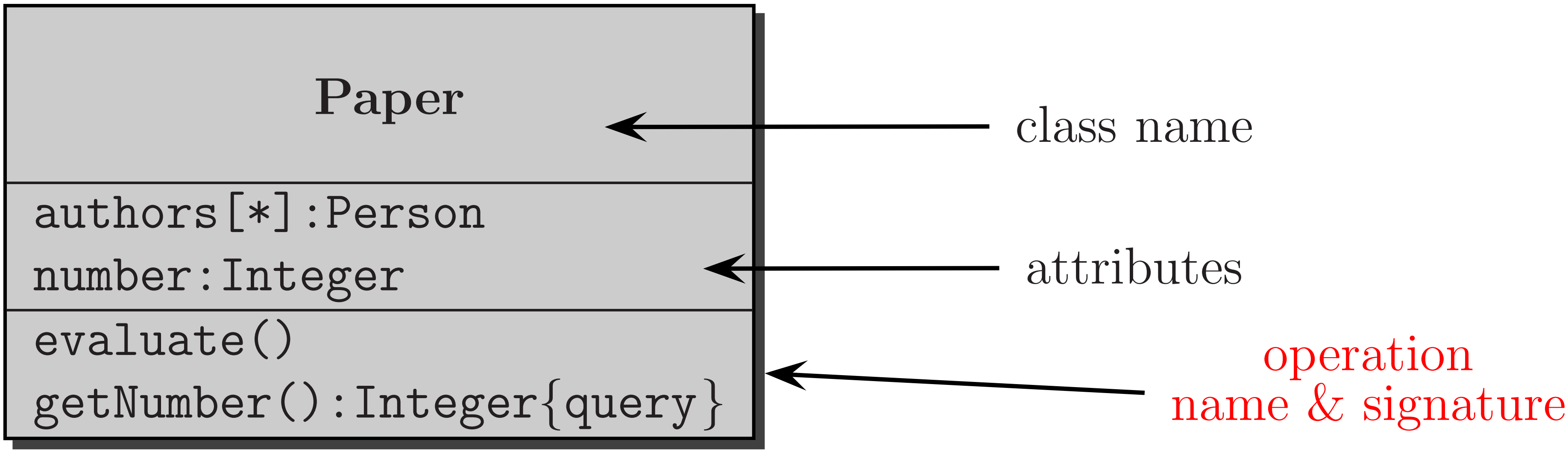
$$I(\mathtt{getNumber()}) \; : \; I(\overbrace{\text{Paper}}^{\text{Owner}}) \to I(\mathtt{Integer})$$

Semantics of static queries omits owner class argument.

Operations with Side Effects



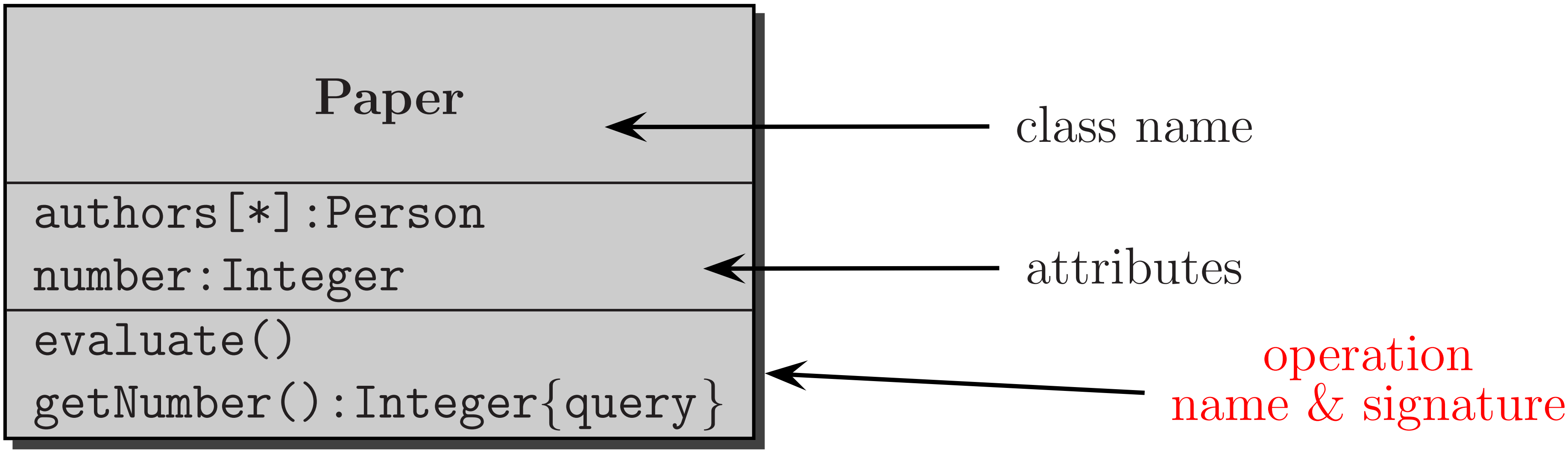
Operations with Side Effects



Semantics (operations w/o result)

Transition from snapshot to snapshot
(relation between sets of snapshots)

Operations with Side Effects

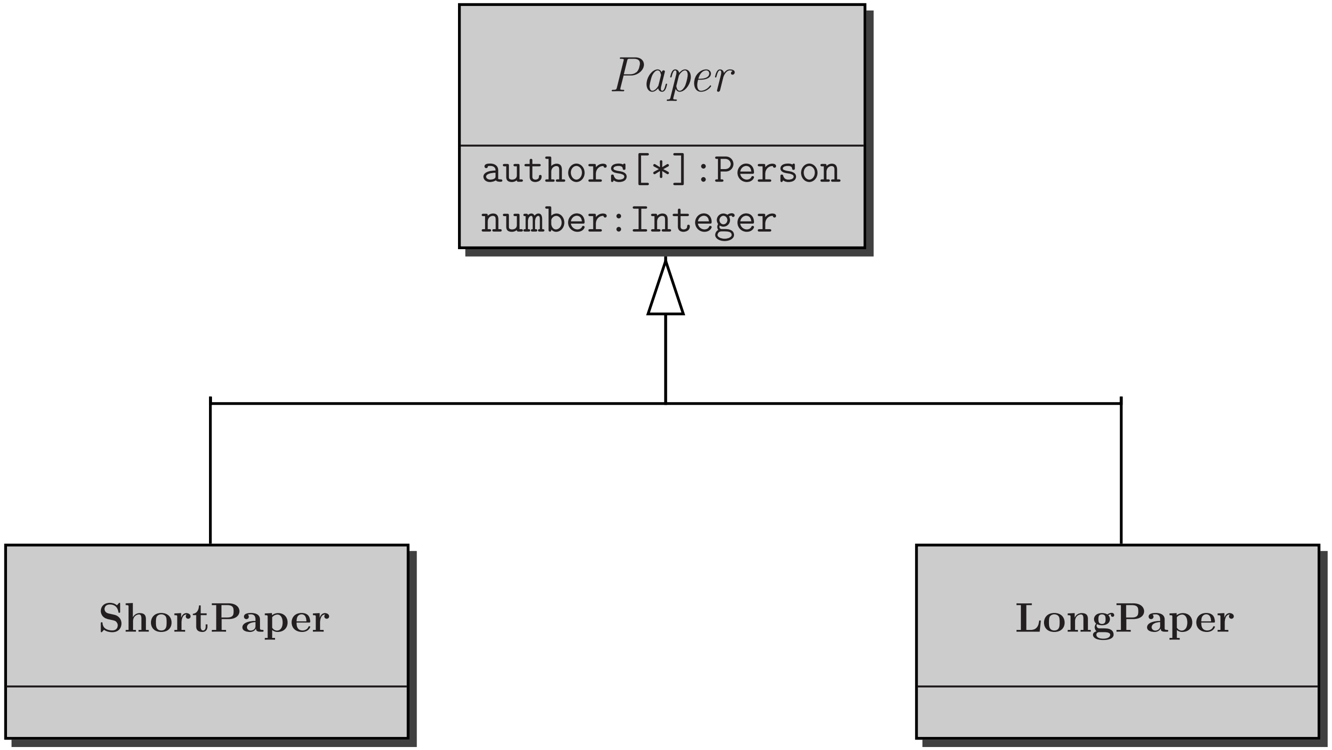


Semantics (operations w/o result)

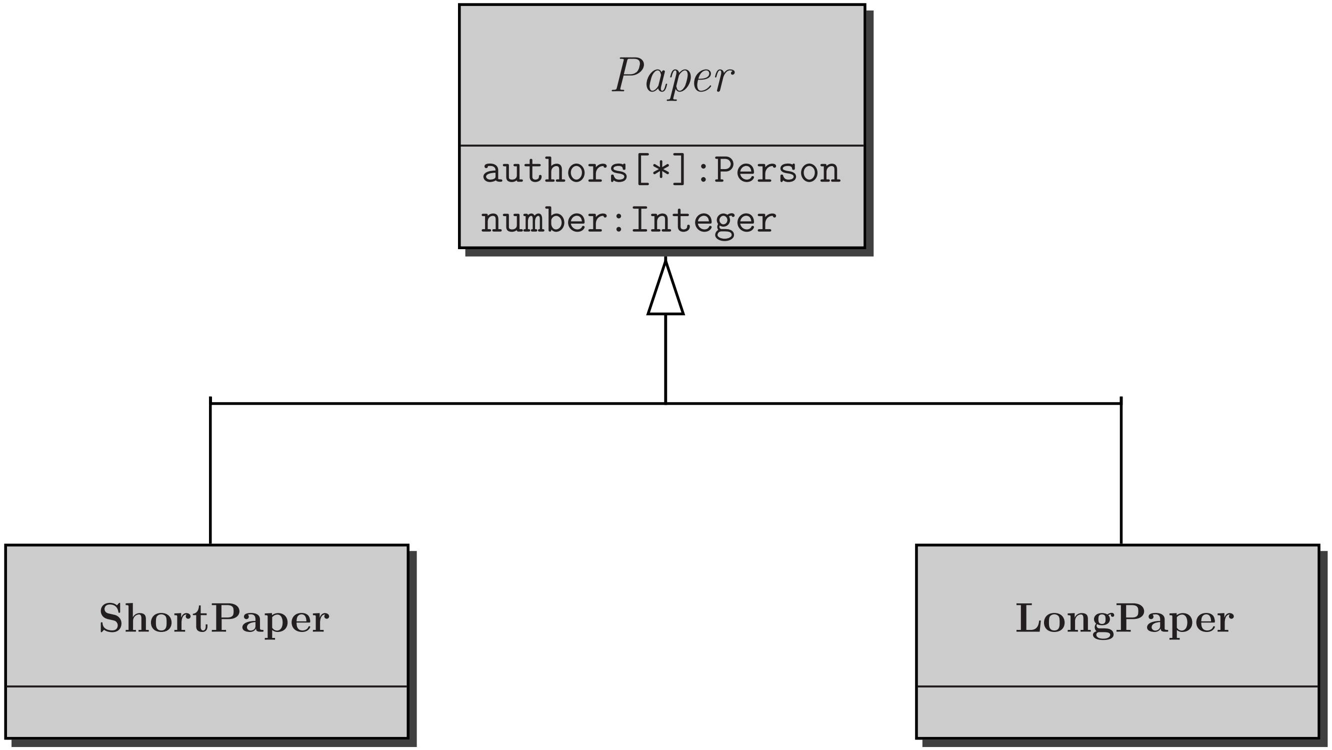
Transition from snapshot to snapshot
(relation between sets of snapshots)

We are not more specific for now: only queries allowed in OCL

Abstract Classes



Abstract Classes



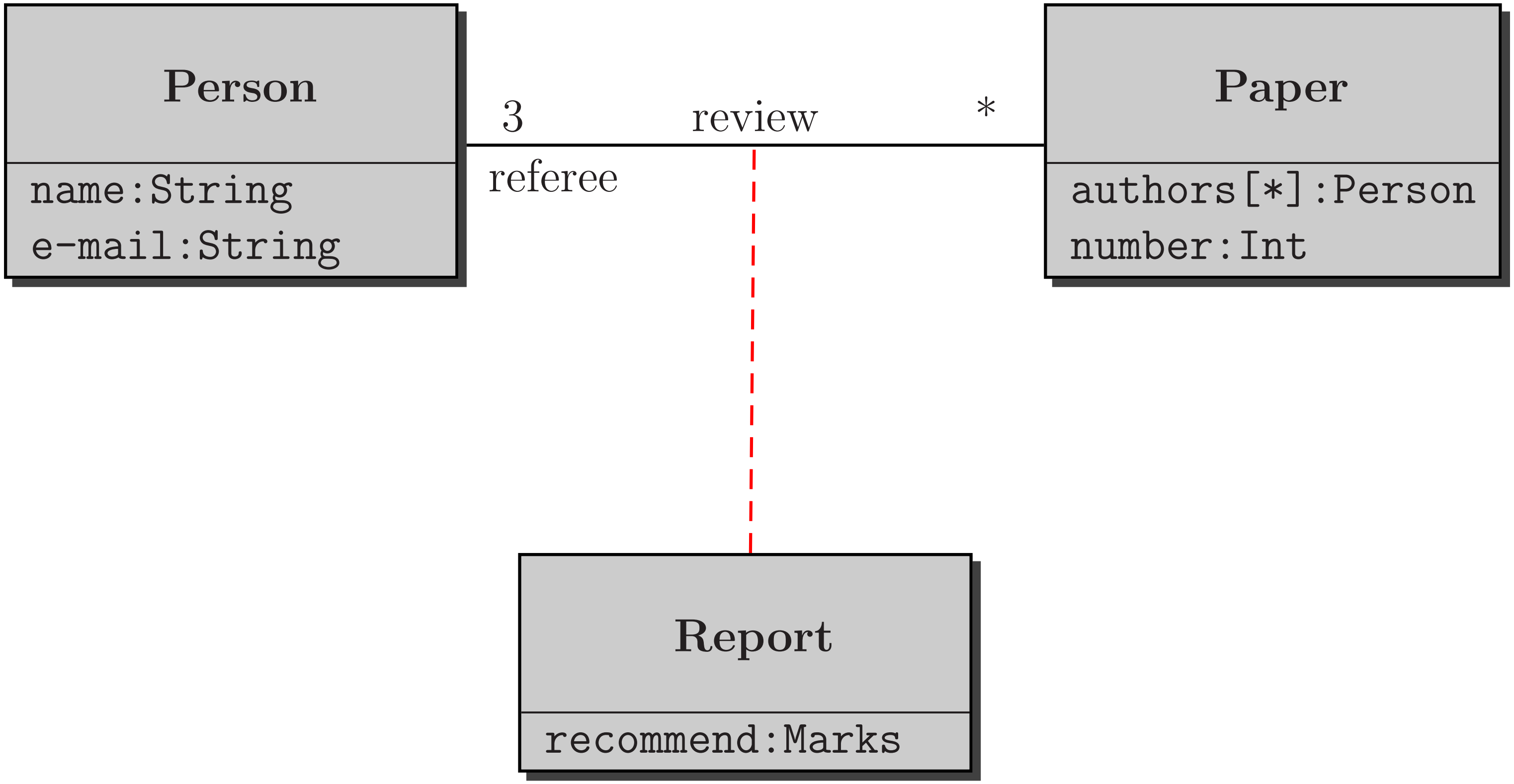
Semantics of Abstract Classes (and Interfaces)

$$I(\mathit{Paper}) = I(\text{ShortPaper}) \cup I(\text{LongPaper})$$

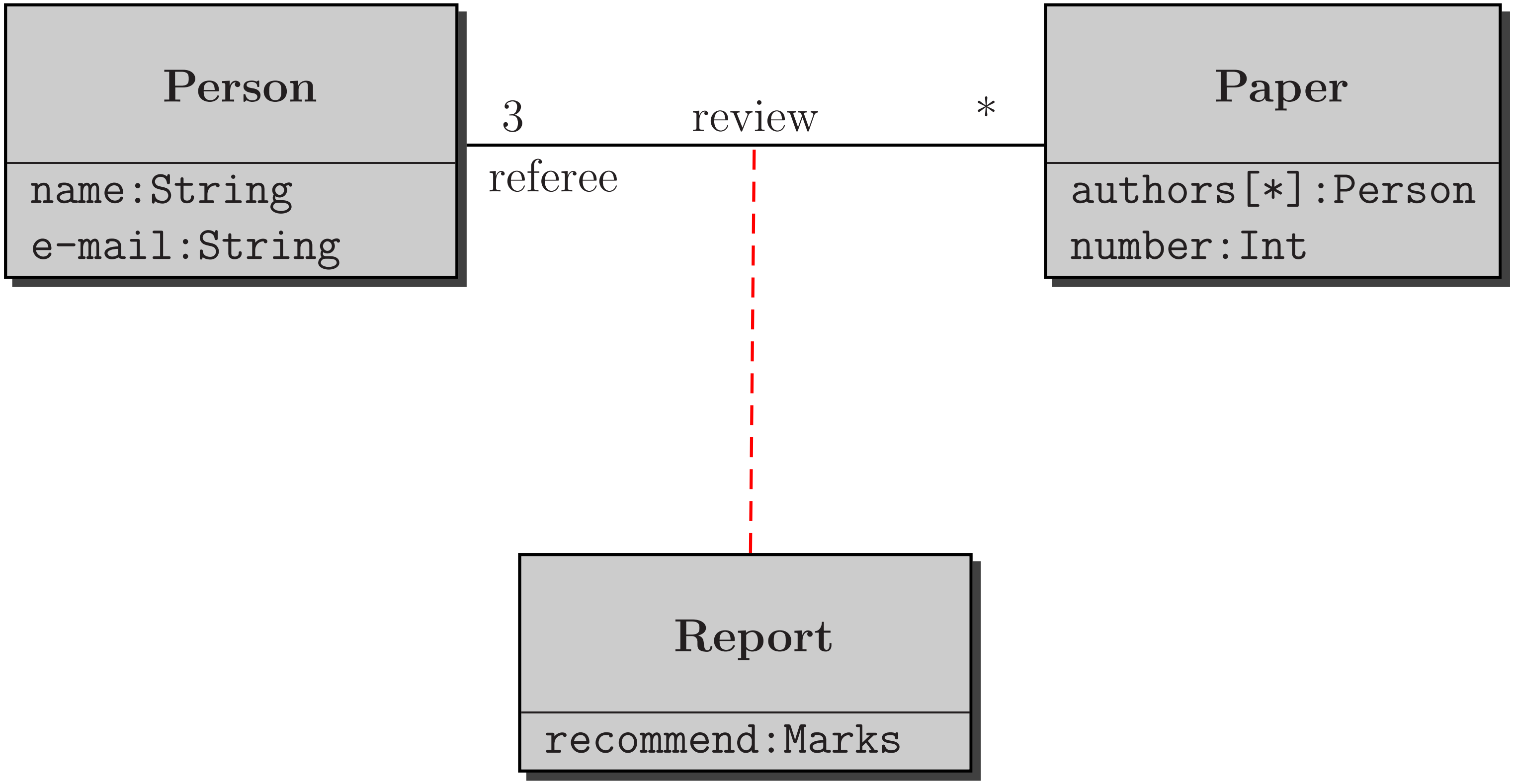
$$I(\text{ShortPaper}) \cap I(\text{LongPaper}) = \{\texttt{null}\}$$

No “direct” elements in $I(\mathit{Paper})$

Association Class



Association Class



Semantics

$I(\text{Report})$ **is a subset of** $I(\text{Person})\backslash\{\texttt{null}\} \times I(\text{Paper})\backslash\{\texttt{null}\}$

Data Types

≪data type≫
Integer

```
=(i2:Integer):Boolean
+(i2:Integer):Integer
+(i2:Real):Real
-(i2:Integer):Integer
-(i2:Real):Real
(i2:Integer):Integer
(i2:Real):Real
\(i2:Integer):Real
\(i2:Real):Real
abs:Integer
div(i2:Integer):Integer
mod(i2:Integer):Integer
max(i2:Integer):Integer
min(i2:Integer):Integer
```

≪data type≫
String

```
=(i2:String):Boolean
size:Integer
concat(string2:String):String
toUpper(string2:String):String
toLower(string2:String):String
substring(lower:Integer, upper:Integer):String
```

Data Types

Syntax

UML stereotype data type

No attributes

Semantics

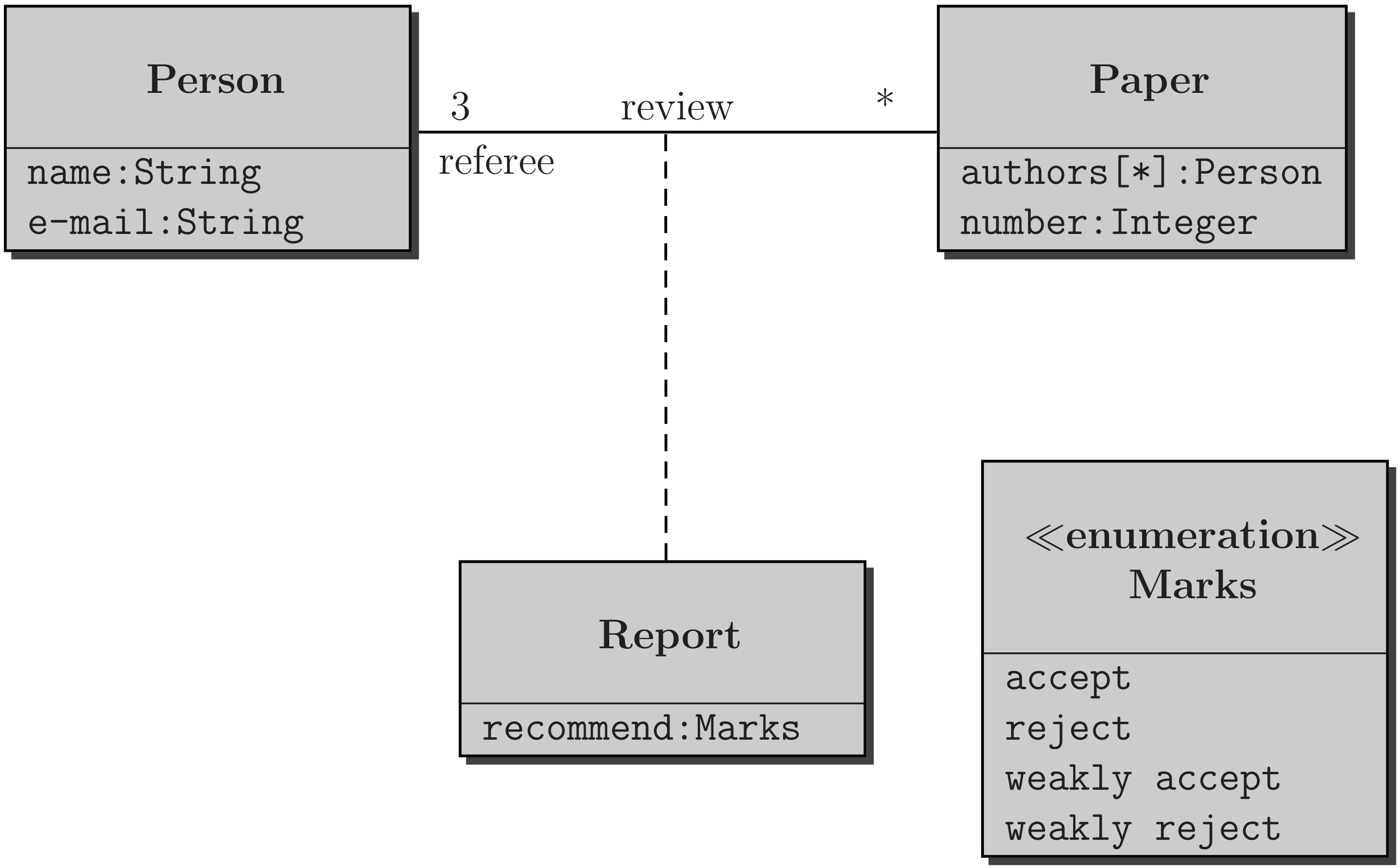
Semantics $I(C)$ of a data type class C fixed for all snapshots I

$I(\mathtt{Integer})$ **is the same in all snapshots**

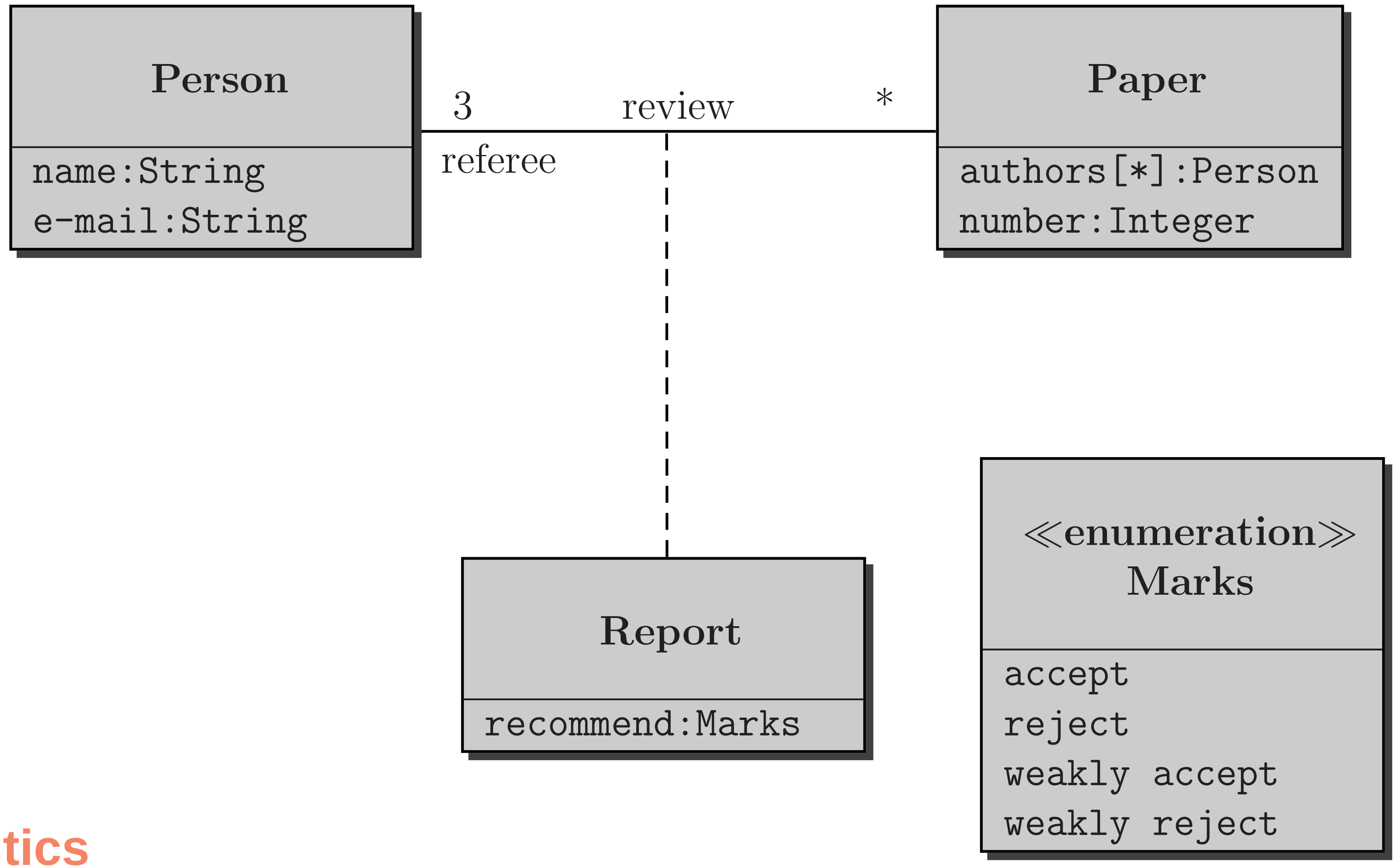
All operations are queries (no side effects)

$$I(+) \,:\, I(\mathtt{Integer}) \times I(\mathtt{Integer}) \rightarrow I(\mathtt{Integer})$$

Enumerations



Enumerations

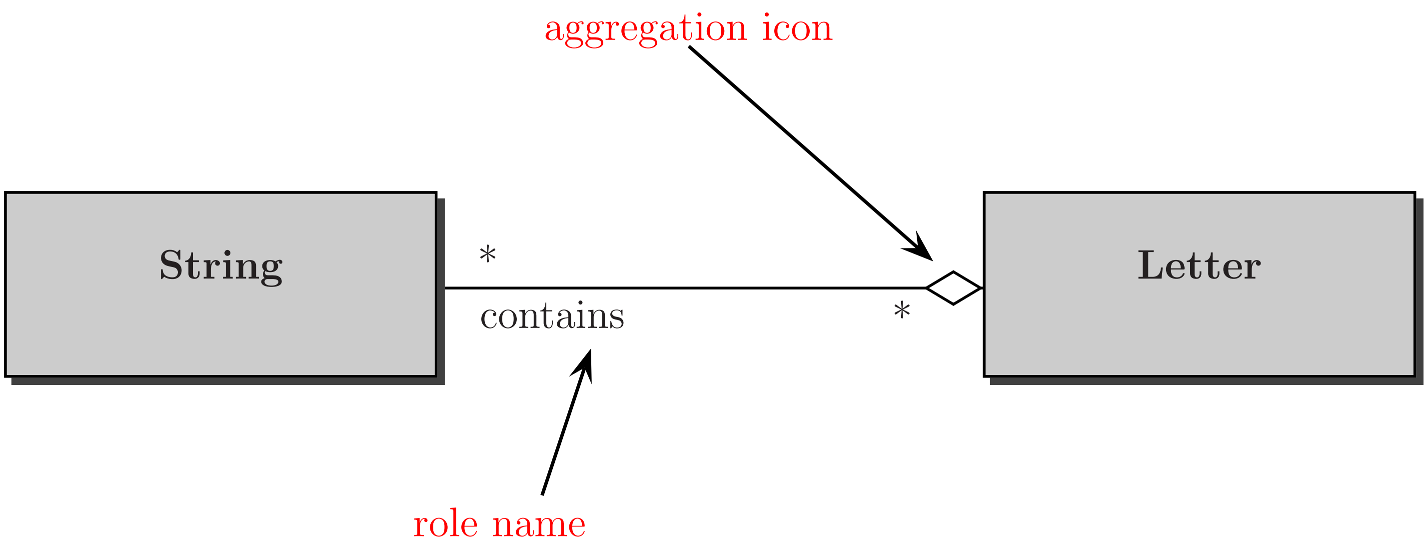


Semantics

Special data type: finite number of instances, explicit representation

$$I(\texttt{Marks}) \supseteq \{\texttt{accept}, \texttt{reject}, \texttt{weakly accept}, \texttt{weakly reject}\}$$

Aggregations



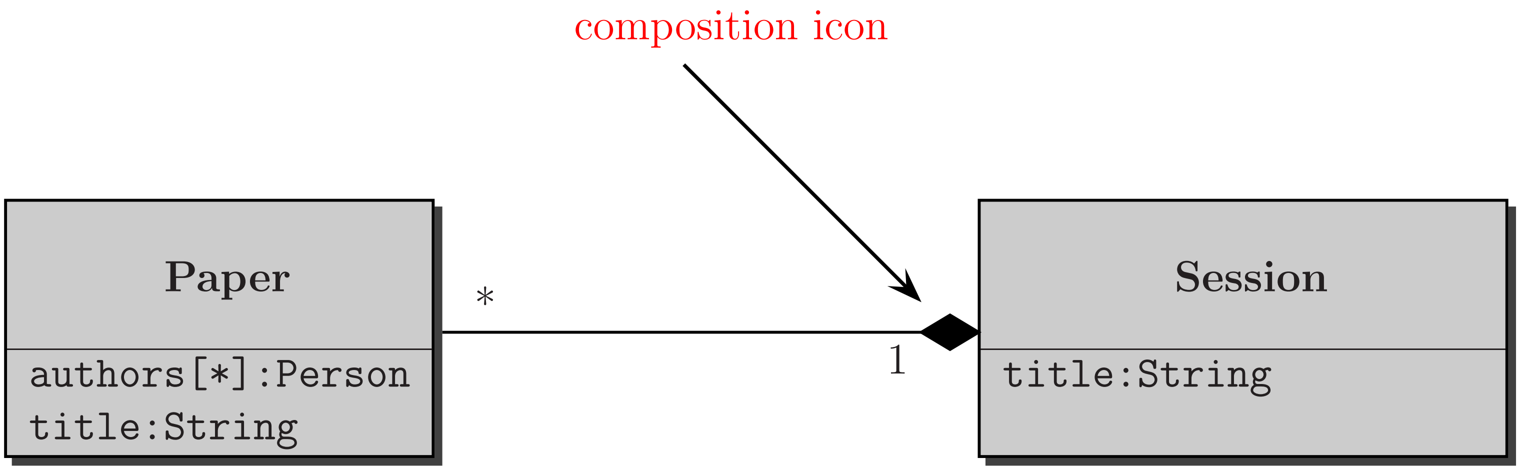
Semantics

Same (formal) semantics as an association

Pragmatics

Part-of relationship, though may be shared with other objects

Compositions



Semantics

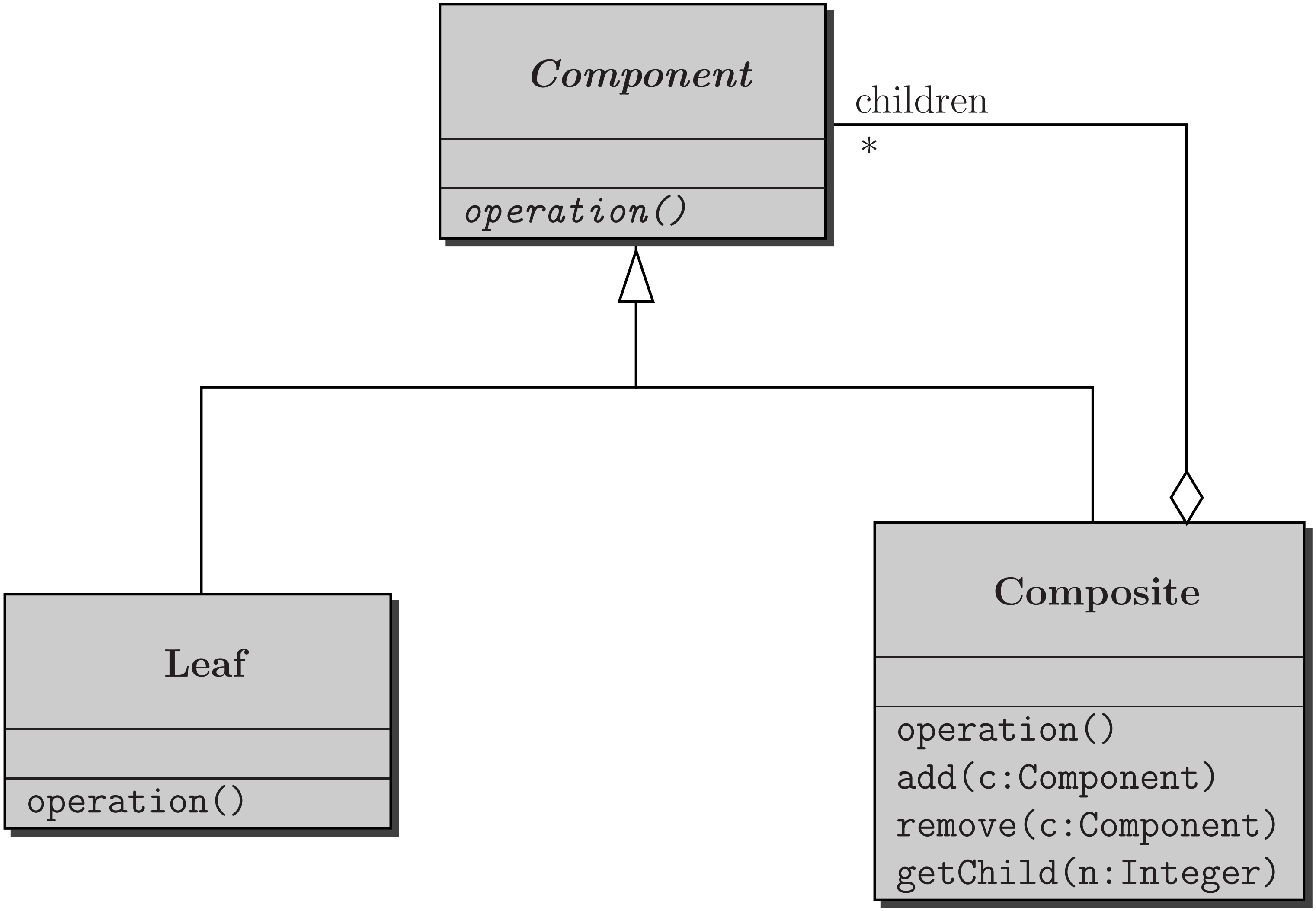
Same (formal) semantics as an association

Pragmatics

Owned-by, object lifetime controlled by client (= owner)

Client multiplicity $0..1$ or 1

Example: The Composite Pattern



Use Case Diagrams. . .

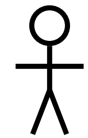
Means for specifying required user scenarios of a system

Use Case Diagrams. . .

Means for specifying required user scenarios of a system

Key concepts:

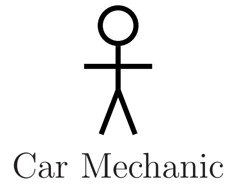
- **Actor** — **Role that user plays wrt system; outside the system.**



Use Case Diagrams. . .

Means for specifying required user scenarios of a system

Key concepts:



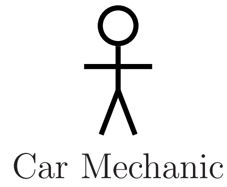
- **Actor** — **Role that user plays wrt system; outside the system.**

- **Use Case** — **Set of scenarios subsumed under common user goal**

Use Case Diagrams. . .

Means for specifying required user scenarios of a system

Key concepts:

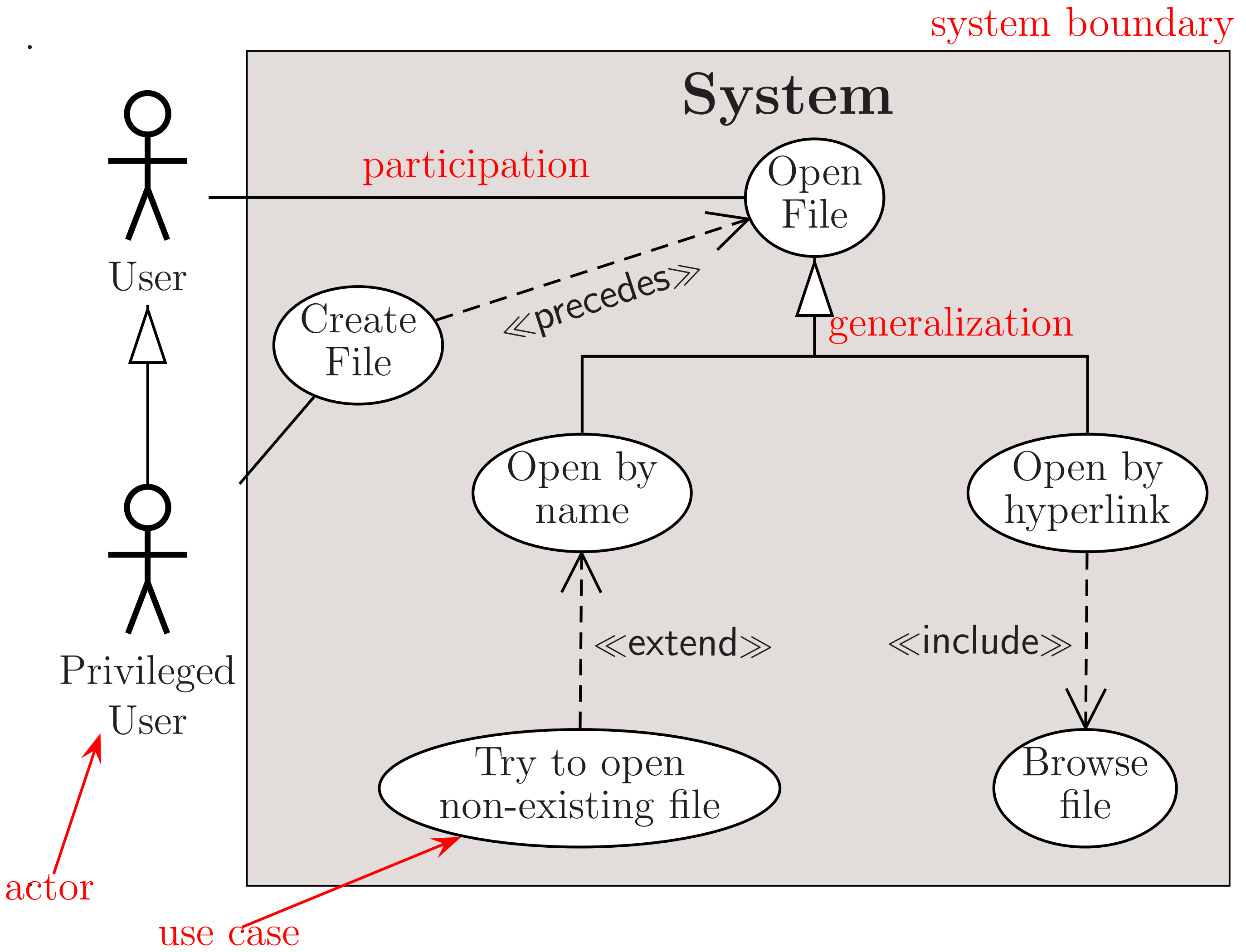


- **Actor** — **Role that user plays wrt system; outside the system.**

- **Use Case** — **Set of scenarios subsumed under common user goal**

- **Subject** — **System implementing the use case**
- **Connections between use cases, actors and other use cases (stereotypes ≪include≫ and ≪extend≫)**

Use Case Syntax



Use Case Relations and Stereotypes

Relations

Participation: communication of instances of actor/use case
Can be directed (even towards actor) if communication one-way

Generalization (actors): communicates with at most as many use cases

Generalization (use cases): more general case

Use Case Relations and Stereotypes

Relations

Participation: communication of instances of actor/use case
Can be directed (even towards actor) if communication one-way

Generalization (actors): communicates with at most as many use cases

Generalization (use cases): more general case

Stereotypes

Include: behaviour always contains that of included use case

Extend: behaviour can be augmented with that of extending use case
Much confusion about precise semantics — **discouraged**

Precedes: user-defined stereotype, not UML standard

Use Cases — Important Issues

Actors usually persons, but can be also (other) system

Use relations among use cases and among actors with great care

Danger: clutter up use cases with detailed analysis

Danger: Deal with implementation aspects by ≪extend≫

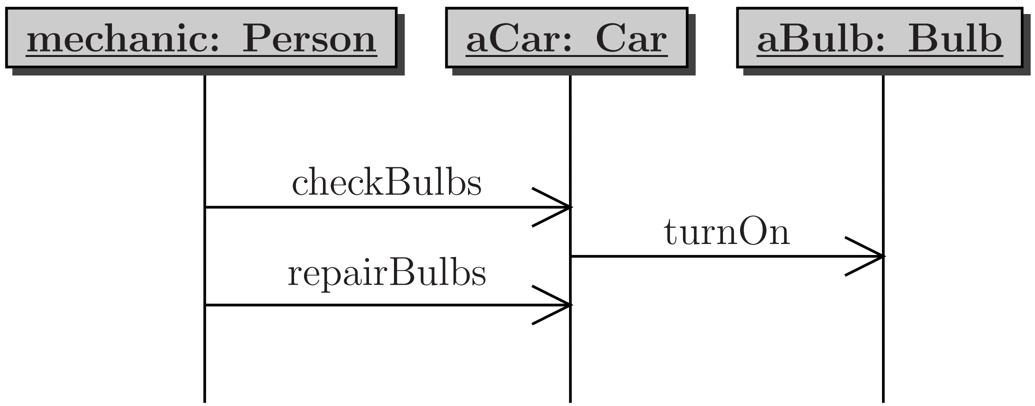
Use cases can be effective help in requirements analysis

Use cases can be bad and confusing "programming language"

Sequence Diagrams

Interaction

Scenario realised by system run



Sequence Diagrams

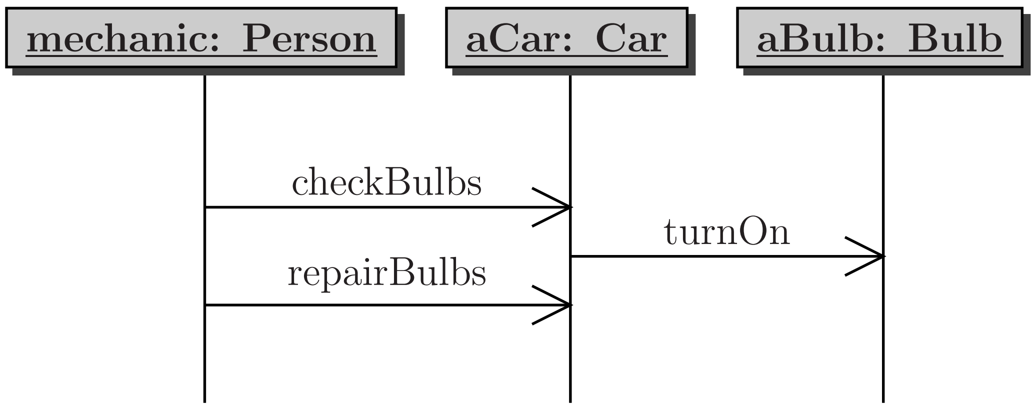
Interaction

Scenario realised by system run

Lifeline

Individual participant in inter-

action; instance of class or actor



Sequence Diagrams

Interaction

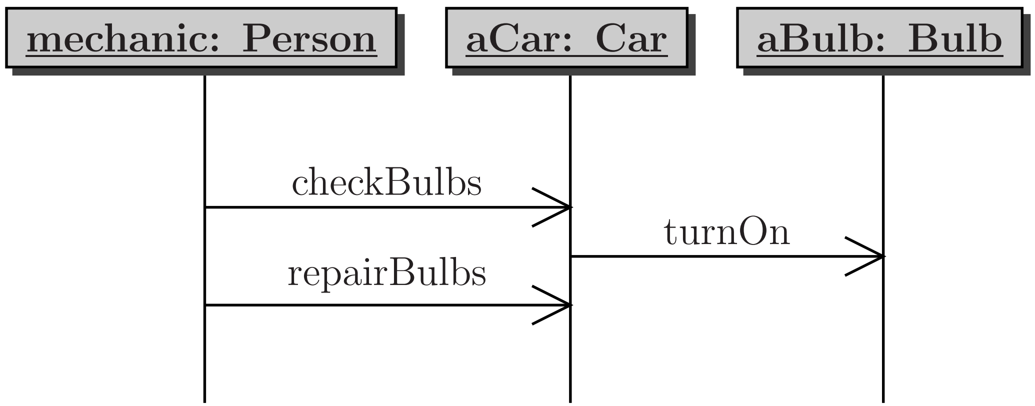
Scenario realised by system run

Lifeline

Individual participant in interaction; instance of class or actor

Message

Individual communication between lifelines of interaction



Sequence Diagrams

Interaction

Scenario realised by system run

Lifeline

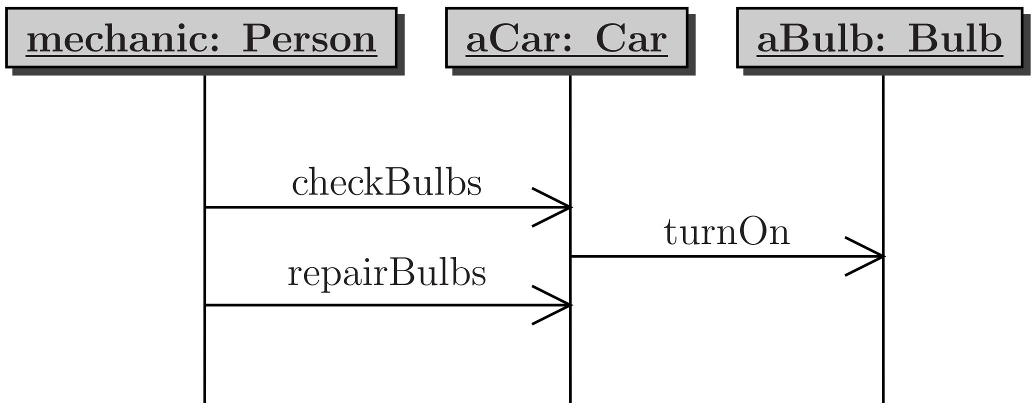
Individual participant in interaction; instance of class or actor

Message

Individual communication between lifelines of interaction

Execution Occurrence

Segment of lifeline being "active"



Sequence Diagrams (cont'd)

- Sequence diagrams model communication in ONE scenario
- Sequence diagrams refine (part of) ONE use case
- Different scenarios require different sequence diagrams
- Sequence diagrams are not algorithms!