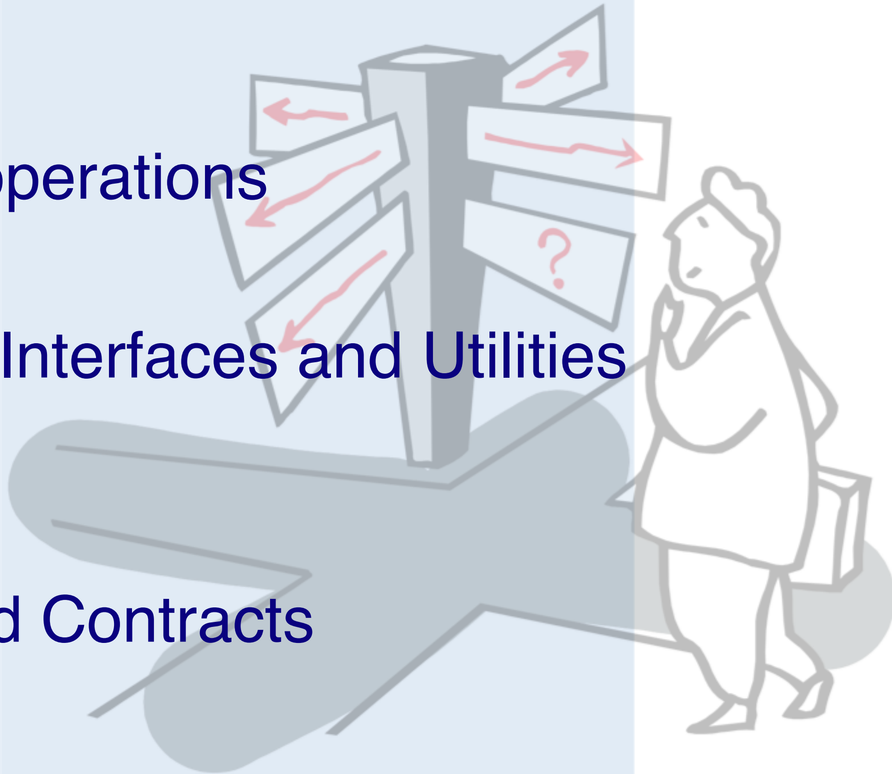
UNIVERSITÄT BERN

Introduction to Software Engineering

6. Modeling Objects and Classes

Roadmap

> UML Overview
> Classes, attributes and operations
> UML Lines and Arrows
> Parameterized Classes, Interfaces and Utilities
> Objects, Associations
> Inheritance
> Patterns, Constraints and Contracts



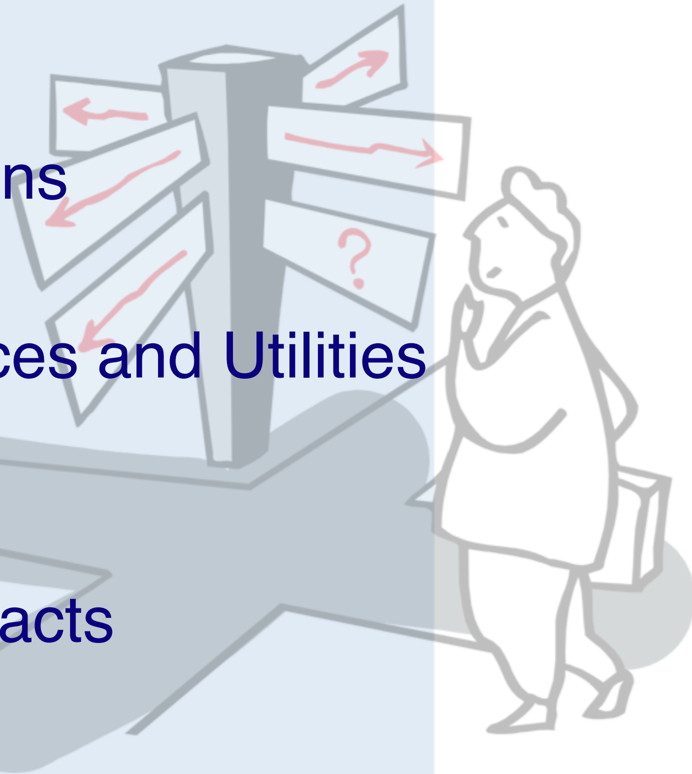
Sources

> *The Unified Modeling Language Reference Manual*, James Rumbaugh, Ivar Jacobson and Grady Booch, Addison Wesley, 1999.

> *UML Distilled*, Martin Fowler, Kendall Scott, Addison-Wesley, Second Edition, 2000.

Roadmap

- **UML Overview**
- Classes, attributes and operations
- UML Lines and Arrows
- Parameterized Classes, Interfaces and Utilities
- Objects, Associations
- Inheritance
- Patterns, Constraints and Contracts



UML

What is UML?

- uniform notation: Booch + OMT + Use Cases (+ state charts)
  - UML is *not* a method or process
  - … The *Unified Development Process* is

Why a Graphical Modeling Language?

- Software projects are carried out in *team*
- Team members need to *communicate*
  - ... sometimes even with the end users
- “One picture conveys a thousand words”
  - the question is only *which words*
  - Need for *different views* on the same software artifact

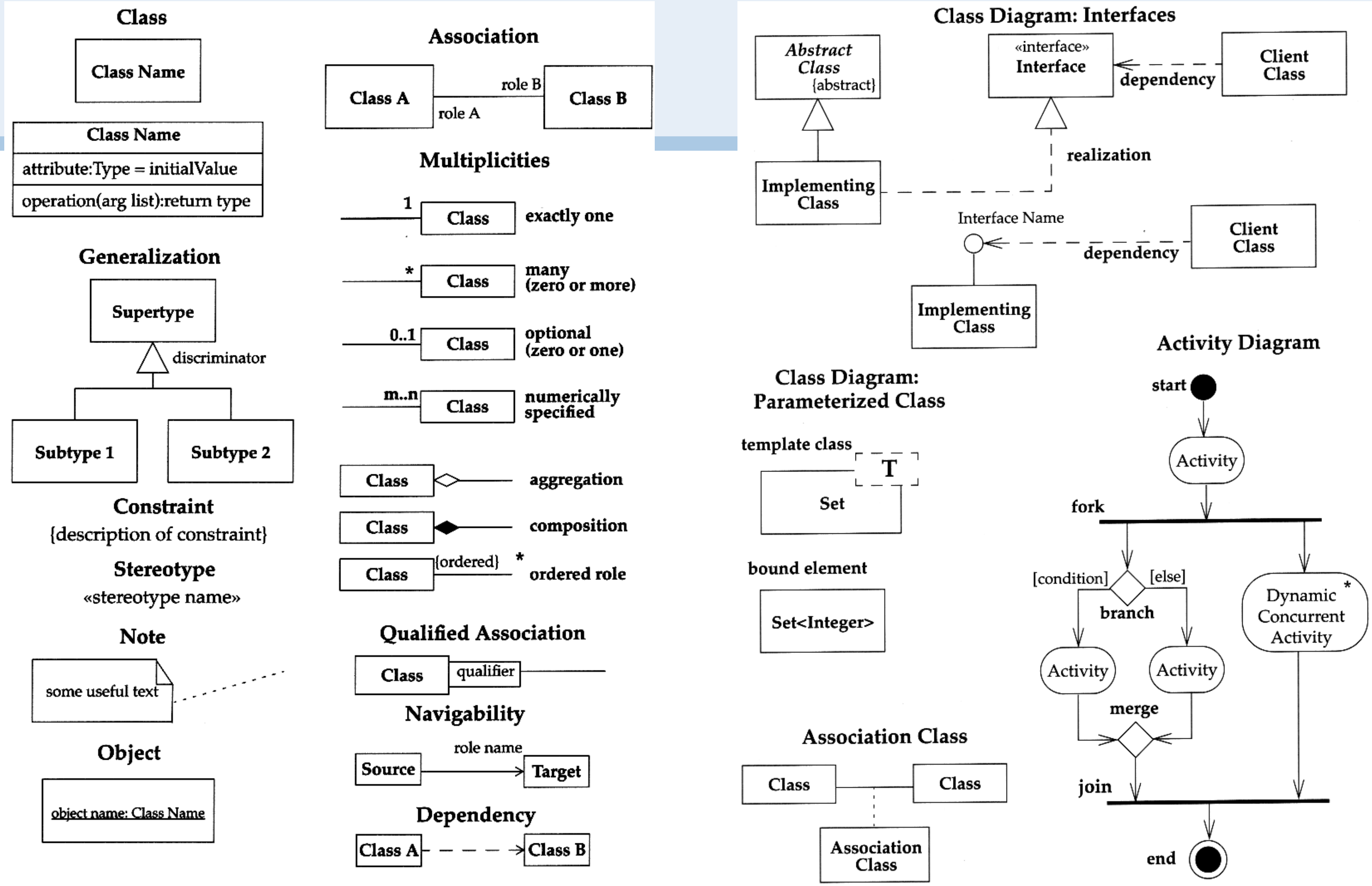
Why UML?

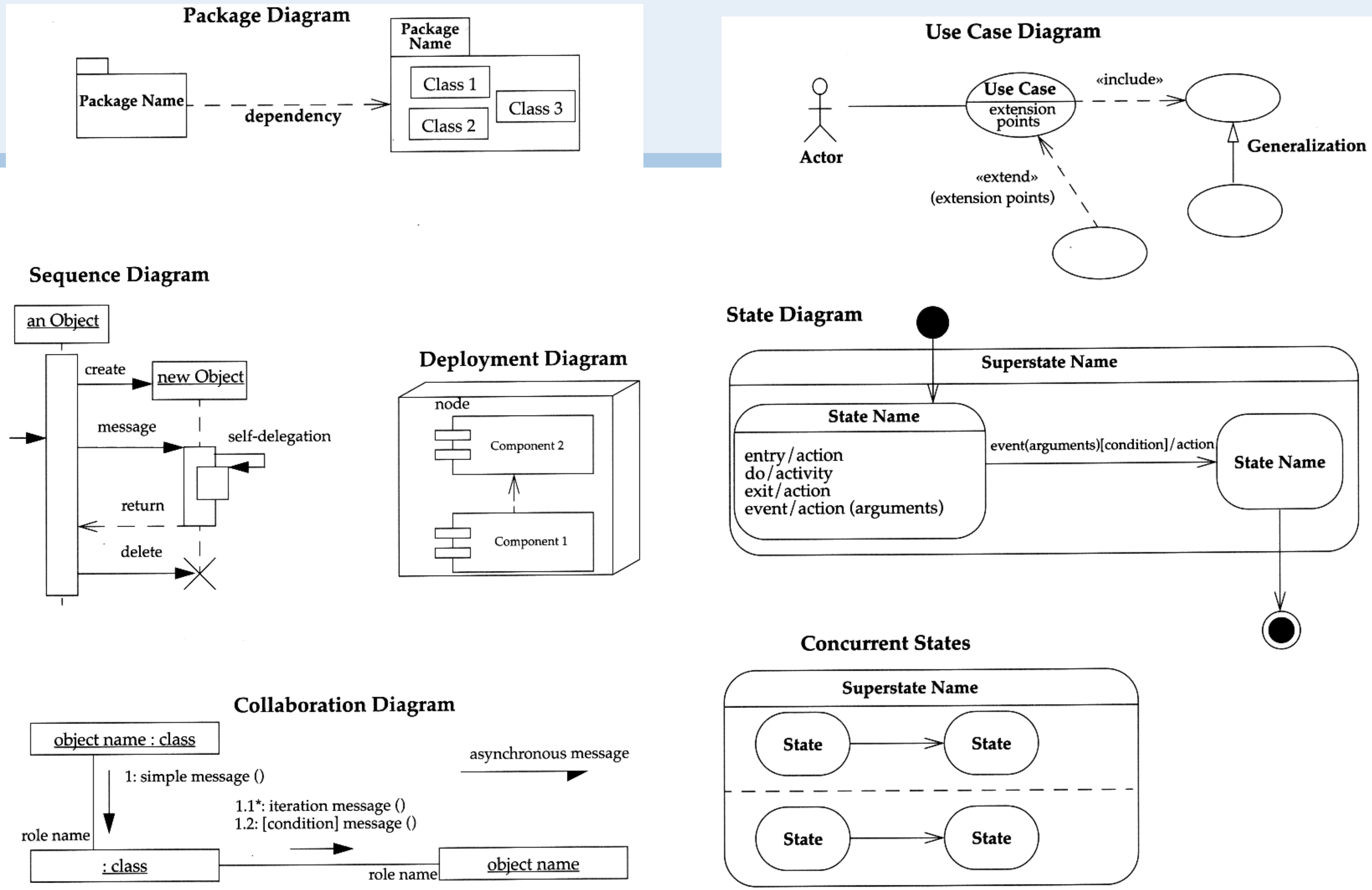
Why UML?

- Reduces *risks* by documenting assumptions
  - — domain models, requirements, architecture, design, implementation …
- Represents industry *standard*
  - — more tool support, more people understand your diagrams, less education
- Is reasonably *well-defined*
  - — ... although there are interpretations and dialects
- Is *open*
  - — stereotypes, tags and constraints to extend basic constructs
  - — has a meta-meta-model for advanced extensions

UML History

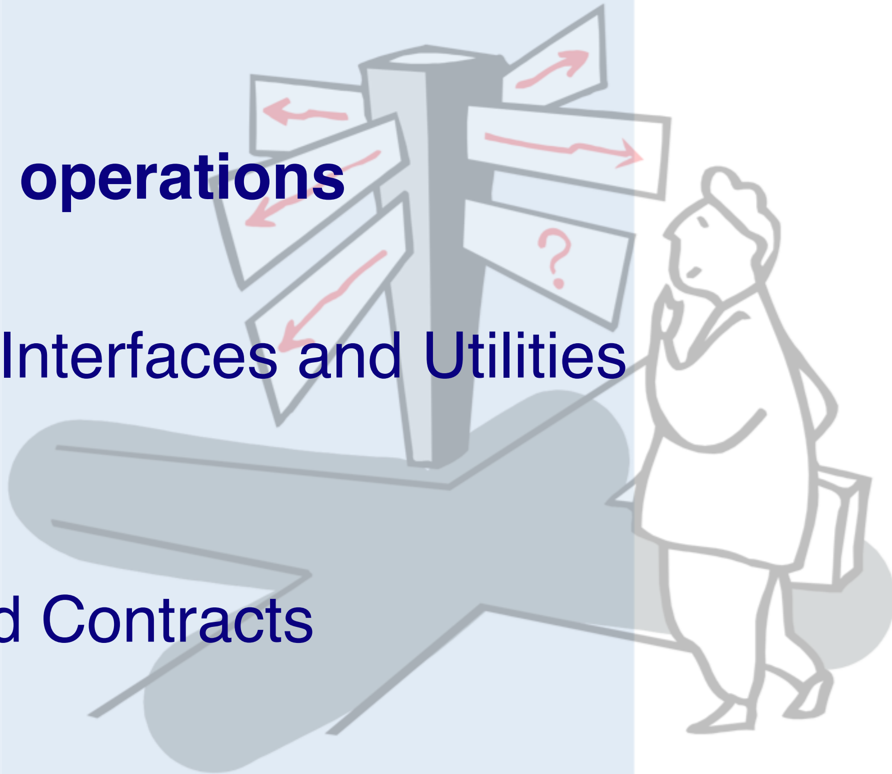
- 1994: Grady Booch (Booch method) + James Rumbaugh (OMT) at Rational
- 1994: Ivar Jacobson (OOSE, use cases) joined Rational
  - "The three amigos"
- 1996: Rational formed a consortium to support UML
- 1997: UML1.0 submitted to OMG by consortium
- 1997: UML 1.1 accepted as OMG standard
  - However, OMG names it UML1.0
- 1998-...: Revisions UML1.2 - 1.5
- 2005: Major revision to UML2.0, includes OCL


Class
Class Name
Class Name
attribute:Type = initialValue
operation(arg list):return type
Generalization
Supertype
discriminator
Subtype 1
Subtype 2
Constraint
{description of constraint}
Stereotype
«stereotype name»
Note
some useful text
Object
object name: Class Name
Association
Class A
role B
role A
Class B
Multiplicities
1
Class
exactly one
*
Class
many (zero or more)
0..1
Class
optional (zero or one)
m..n
Class
numerically specified
Class
aggregation
Class
composition
Class
{ordered}
*
ordered role
Qualified Association
Class
qualifier
Navigability
Source
role name
Target
Dependency
Class A
Class B
Class Diagram: Interfaces
Abstract Class
{abstract}
«interface»
Interface
dependency
Client Class
realization
Implementing Class
Interface Name
dependency
Client Class
Implementing Class
Class Diagram: Parameterized Class
template class
T
Set
bound element
Set<Integer>
Association Class
Class
Class
Association Class
Activity Diagram
start
Activity
fork
[condition]
[else]
branch
Dynamic Concurrent Activity
*
Activity
Activity
merge
join
end



UML Distilled

Roadmap



- UML Overview
- **Classes, attributes and operations**
- UML Lines and Arrows
- Parameterized Classes, Interfaces and Utilities
- Objects, Associations
- Inheritance
- Patterns, Constraints and Contracts

Class Diagrams

"Class diagrams show generic descriptions of possible systems, and object diagrams show particular instantiations of systems and their behaviour."

Attributes and operations are also collectively called *features*.

***Danger:** class diagrams risk turning into data models. Be sure to focus on behaviour*

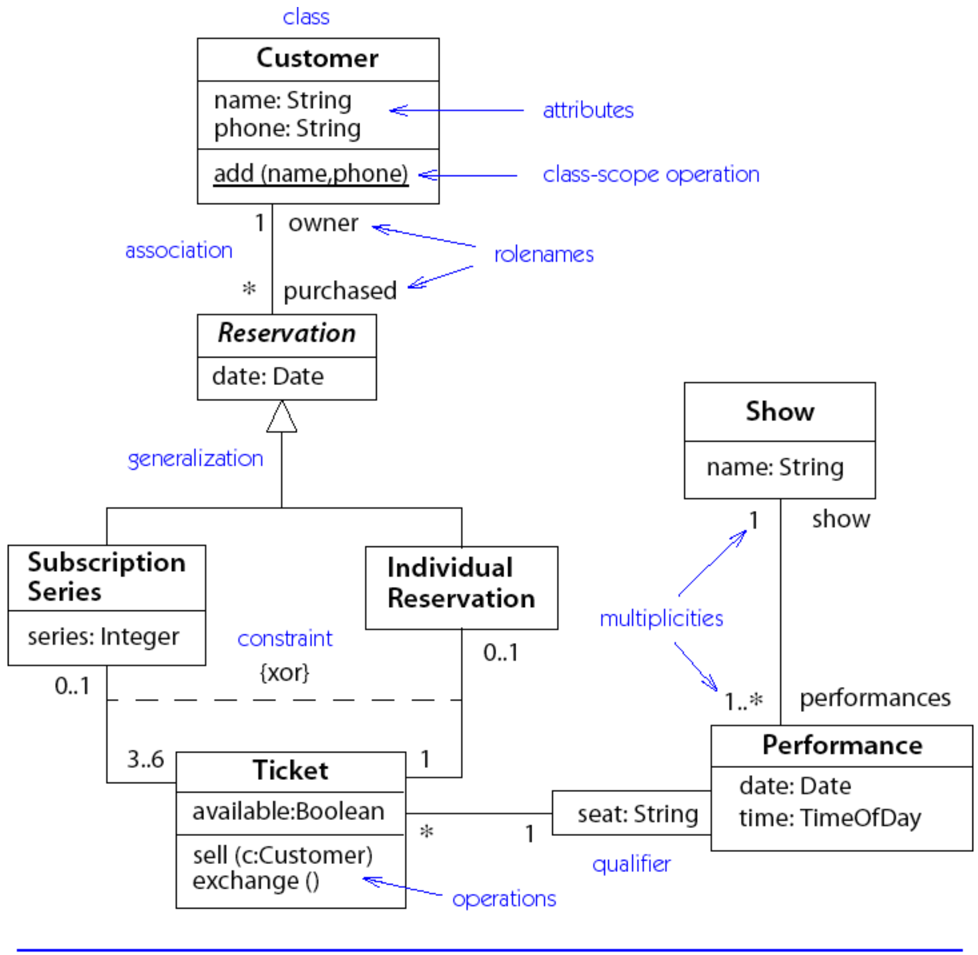
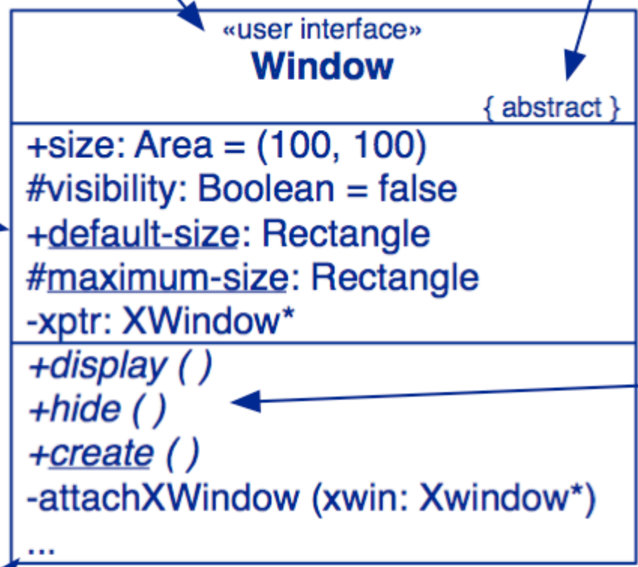


Figure 3-1. *Class diagram*

Visibility and Scope of Features

Stereotype
(what "kind" of class is it?)

User-defined properties
(e.g., readonly, owner = "Pingu")



Don't worry about visibility too early!

underlined attributes have *class scope*

+ = "public"
\# = "protected"
\- = "private"

italic attributes are *abstract*

An ellipsis signals that further entries are not shown

Attributes and Operations

Attributes are specified as:

name: type = initialValue { property string }

Operations are specified as:

name (param: type = defaultValue, ...) : resultType

Roadmap

- UML Overview
- Classes, attributes and operations
- **UML Lines and Arrows**
- Parameterized Classes, Interfaces and Utilities
- Objects, Associations
- Inheritance
- Patterns, Constraints and Contracts

UML Lines and Arrows

Constraint
(usually annotated)

Association
e.g., «uses»

Dependency
e.g., «requires», «imports» ...

Navigable
association
e.g., part-of

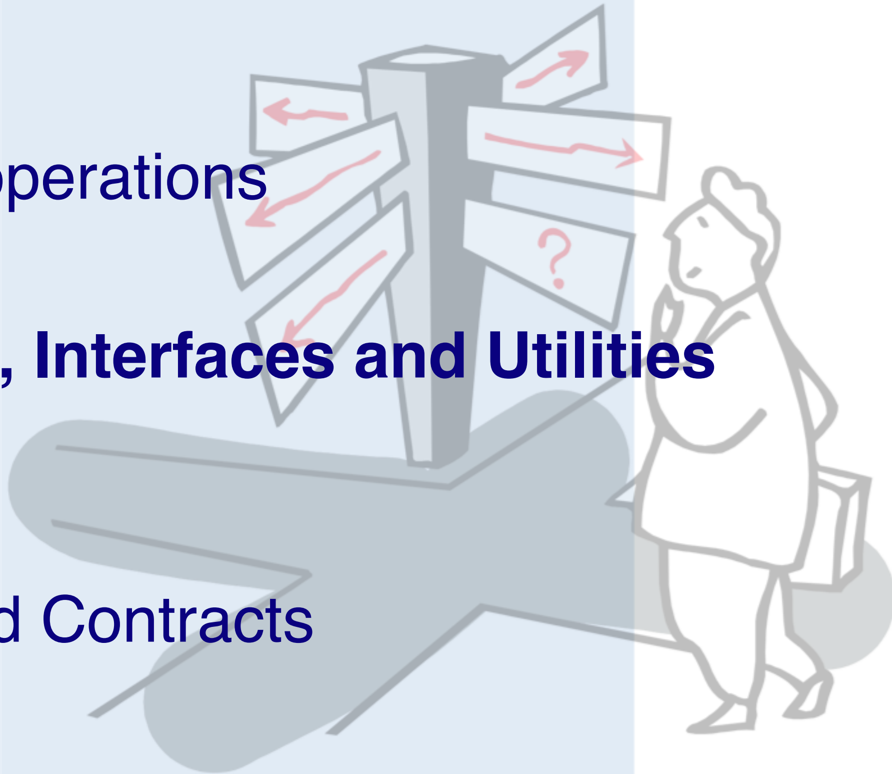
Realization
e.g., class/template, class/interface

"Generalization"
i.e., specialization (!)
e.g., class/superclass, concrete/abstract class

Aggregation
i.e., "consists of"

"Composition"
i.e., containment

Roadmap



- UML Overview
- Classes, attributes and operations
- UML Lines and Arrows
- **Parameterized Classes, Interfaces and Utilities**
- Objects, Associations
- Inheritance
- Patterns, Constraints and Contracts

Parameterized Classes

Parameterized (aka “template” or “generic”) classes are depicted with their parameters shown in a *dashed box*.

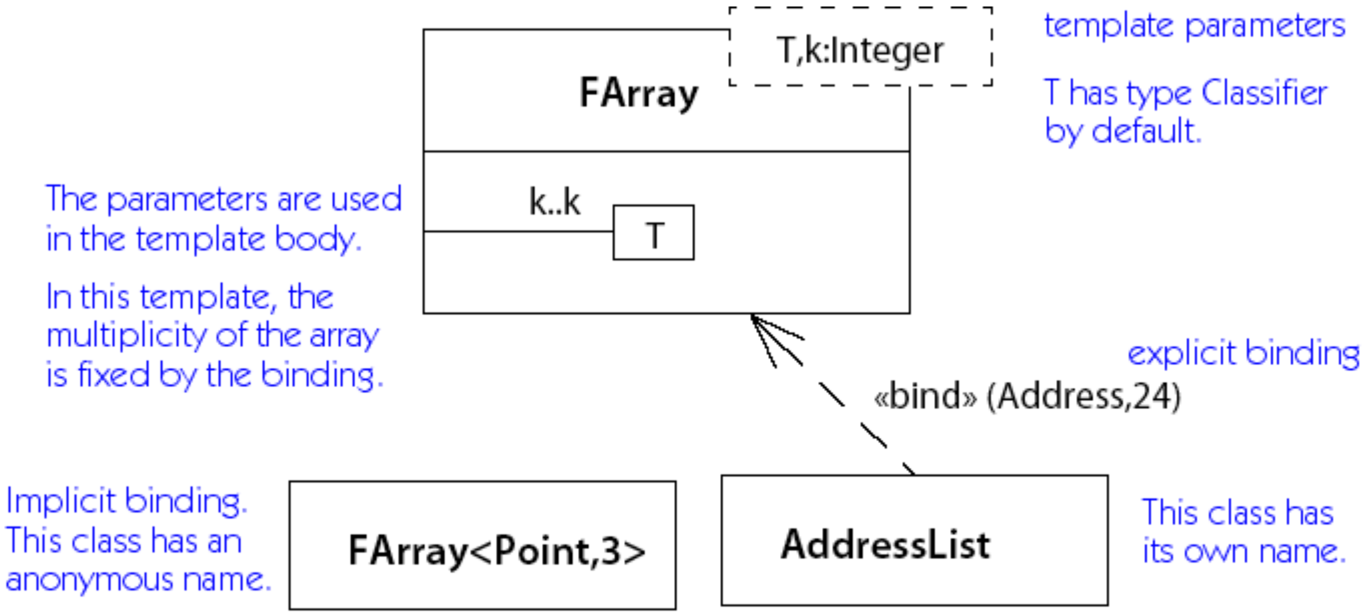


Figure 13-180. *Template notation with use of parameter as a reference*

Interfaces

Interfaces, equivalent to abstract classes with no attributes, are represented as classes with the stereotype «interface» or, alternatively, with the "Lollipop-Notation":

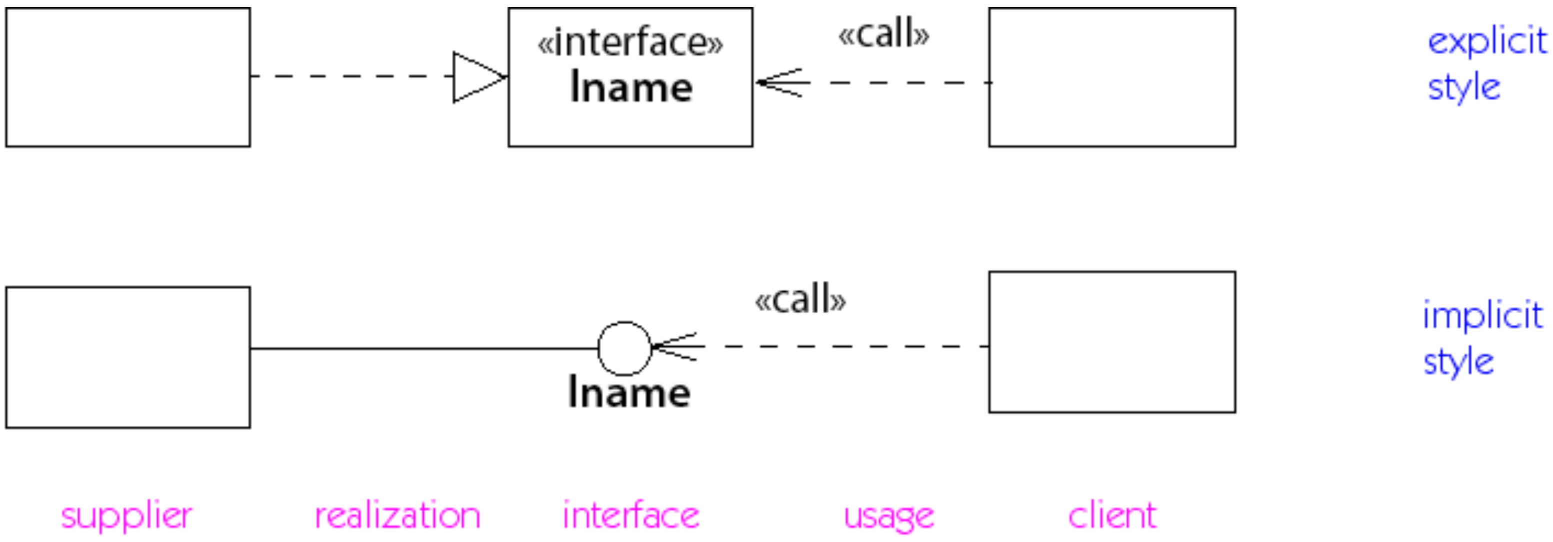
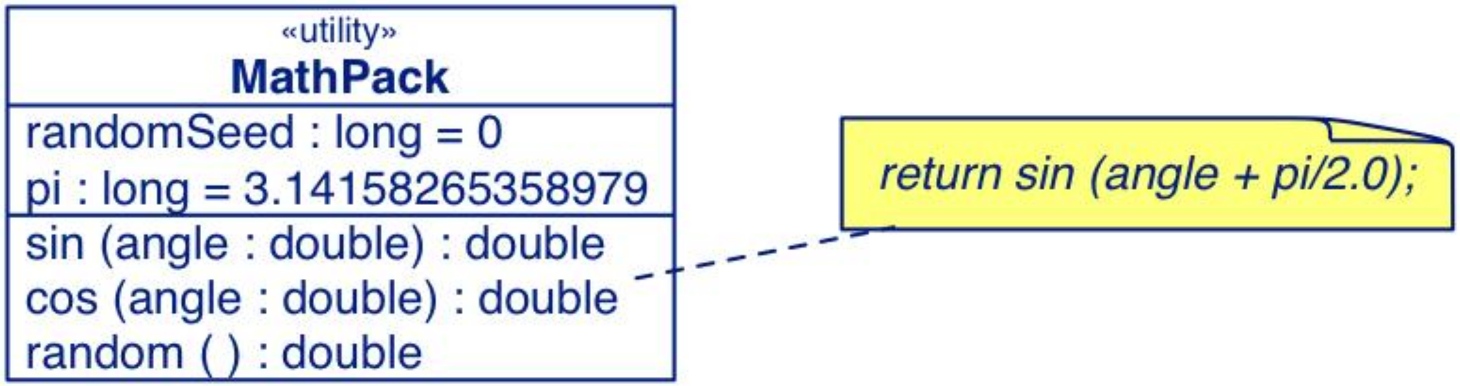


Figure B-5. *Realization of an interface*

Utilities

A utility is a grouping of global attributes and operations. It is represented as a class with the stereotype «utility». Utilities may be parameterized.

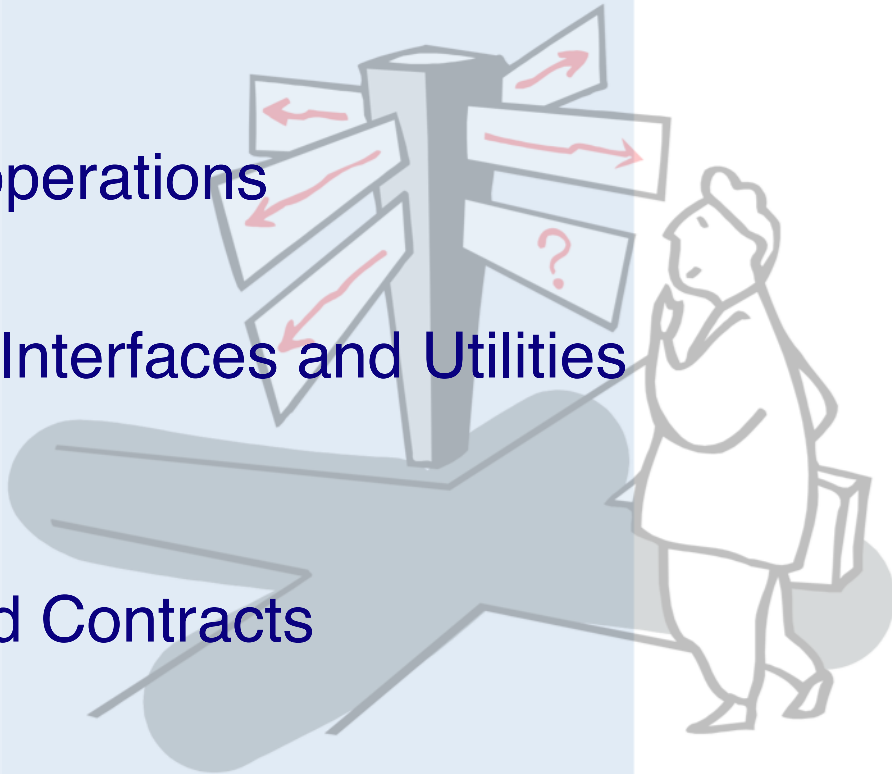


NB: A utility's attributes are already interpreted as being in class scope, so it is redundant to underline them.

A "note" is a text comment associated with a view, and represented as box with the top right corner folded over.

Roadmap

> UML Overview
> Classes, attributes and operations
> UML Lines and Arrows
> Parameterized Classes, Interfaces and Utilities
> **Objects, Associations**
> Inheritance
> Patterns, Constraints and Contracts



Objects

Objects are shown as rectangles with their name and type underlined in one compartment, and attribute values, optionally, in a second compartment.

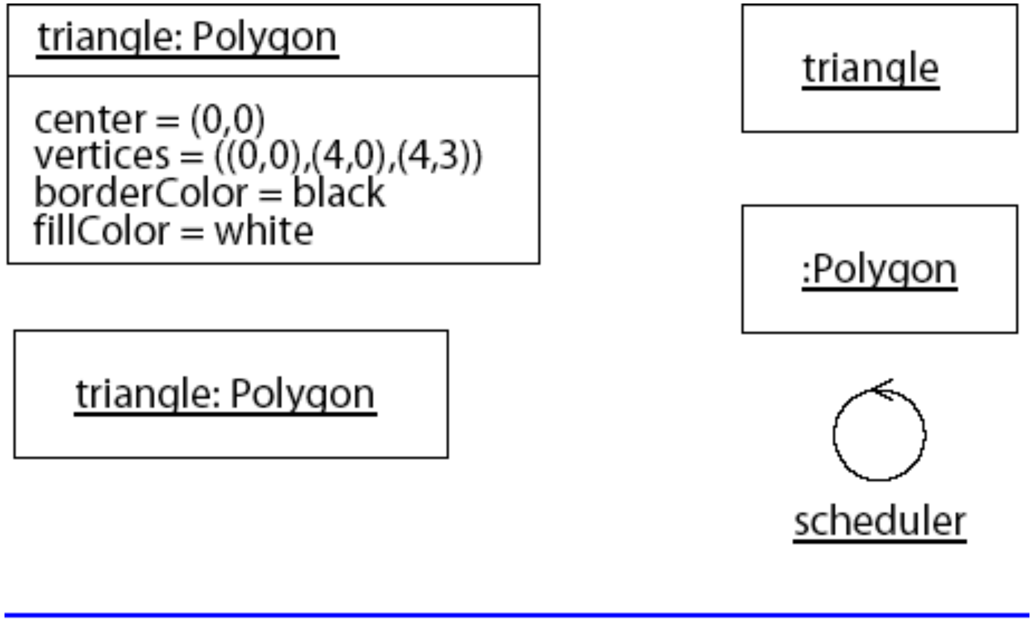


Figure 13-134. *Object notation*

At least one of the name or the type must be present.

Associations

Associations represent *structural relationships* between objects

- usually *binary* (but may be ternary etc.)
- optional *name* and *direction*
- (unique) *role names* and *multiplicities* at end-points

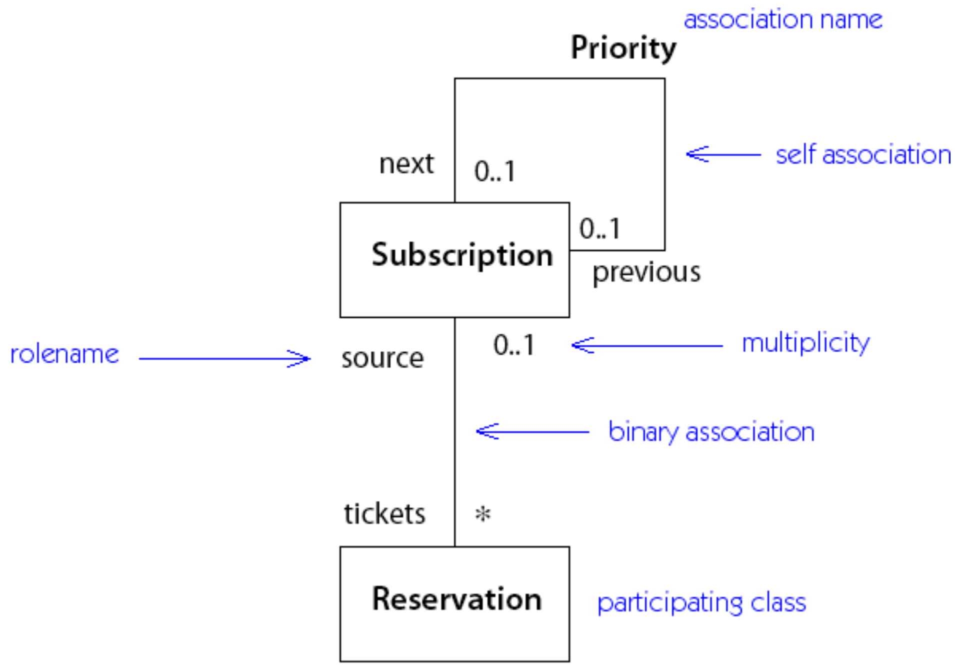


Figure 4-2. *Association notation*

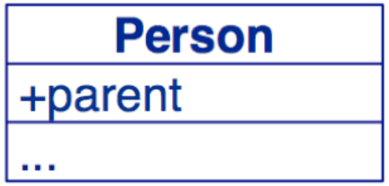
Multiplicity

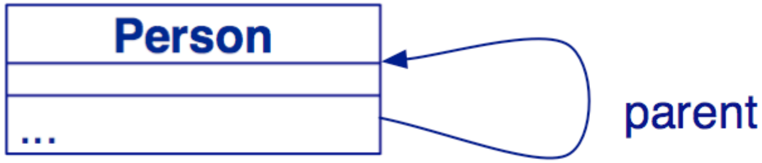
> The multiplicity of an association constrains how many entities one may be associated with
  - Examples:

0..1	Zero or one entity
1	Exactly one entity
*	Any number of entities
1..*	One or more entities
1..n	One to n entities
	And so on …

Associations and Attributes

> Associations may be implemented as attributes
  — But need not be …





Aggregation and Composition

Aggregation is denoted by a *diamond* and indicates a *part-whole dependency*:

A *hollow diamond* indicates a *reference*; a *solid diamond* an *implementation* (i.e., ownership).

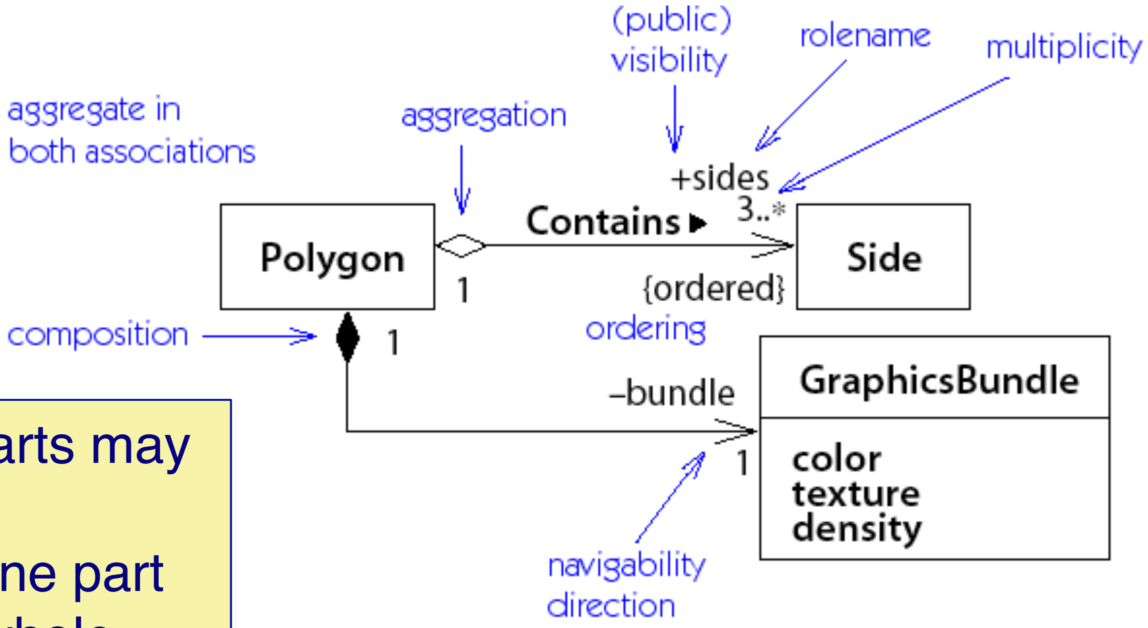


Figure 13-29. *Various adornments on association ends*

Aggregation: parts may be shared.
Composition: one part belongs to one whole.

Association Classes

An association may be an instance of an association class:

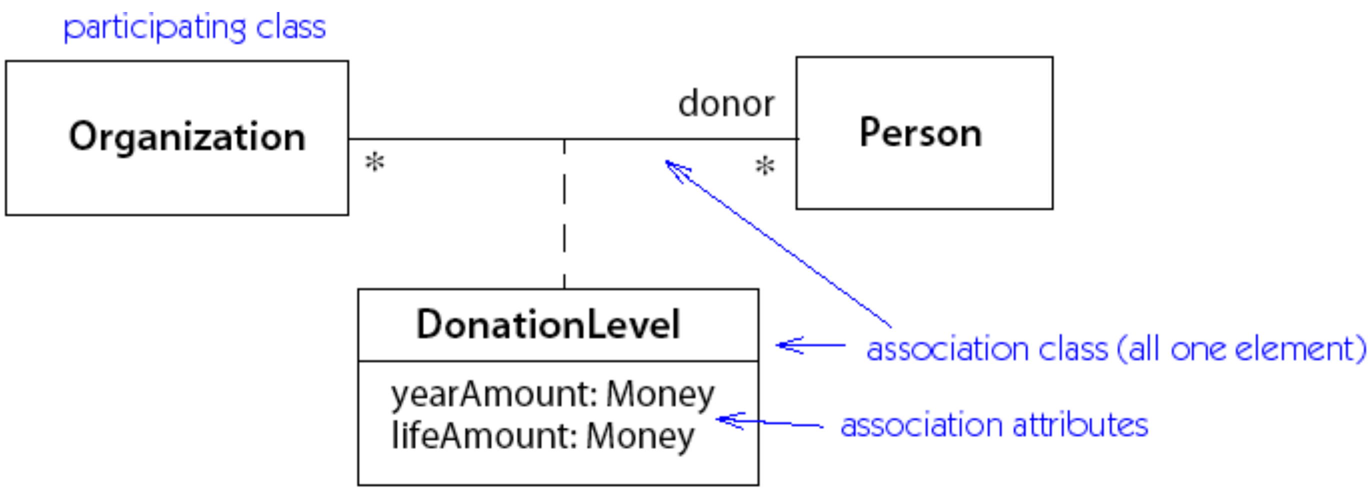


Figure 4-3. *Association class*

In many cases the association class only stores attributes, and its name can be left out.

Qualified Associations

A <u>qualified association</u> uses a special *qualifier value* to identify the object at the other end of the association.

NB: Qualifiers are part of the association, not the class

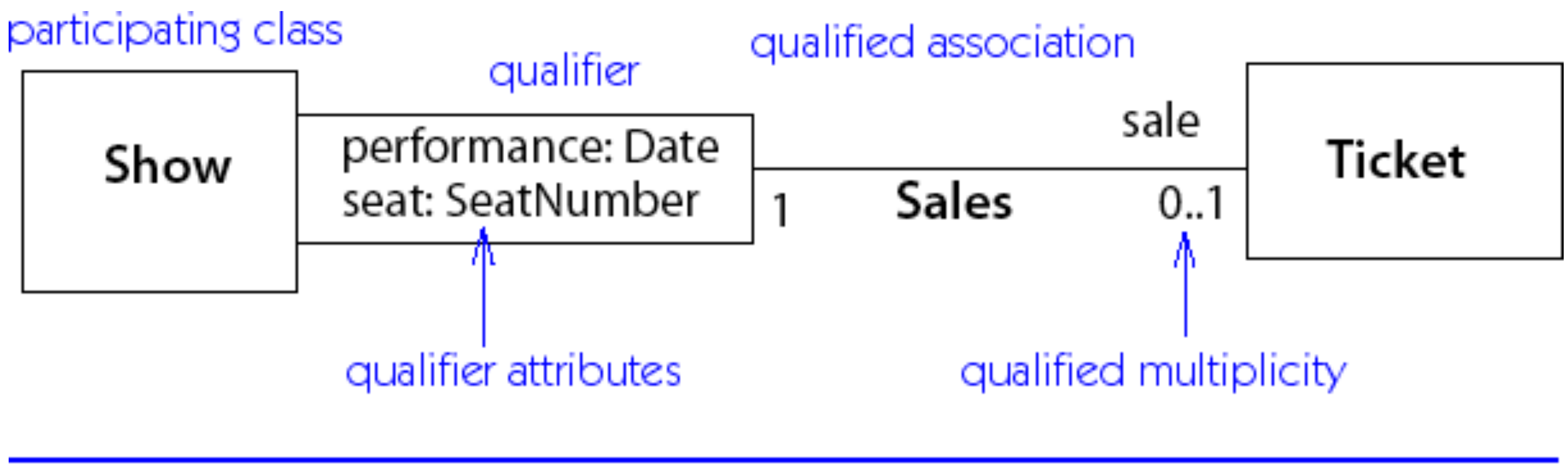
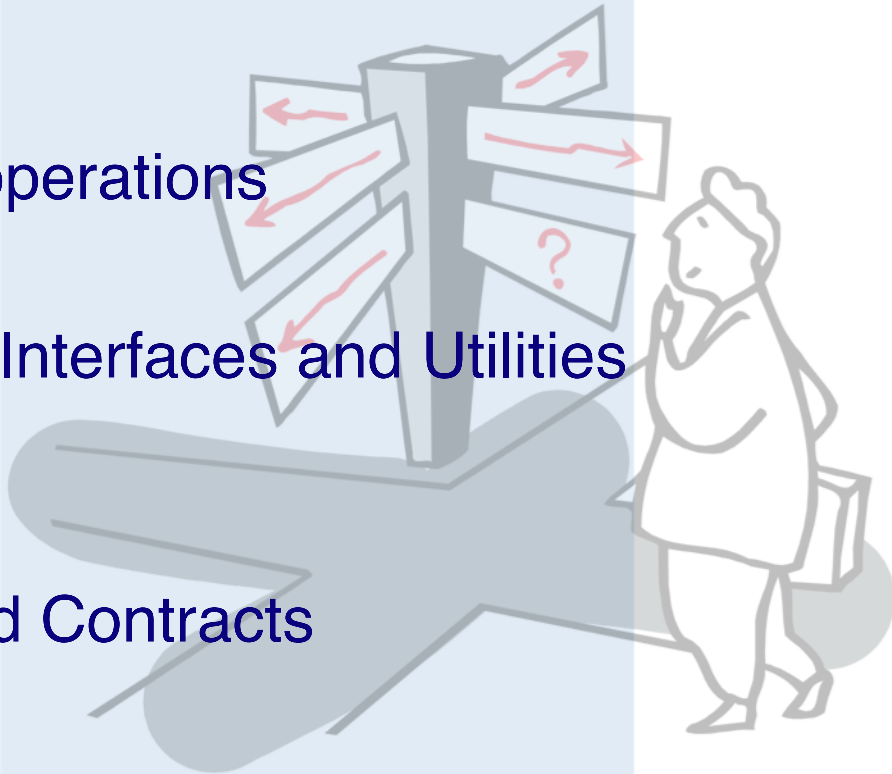


Figure 4-4. *Qualified association*

Roadmap



- UML Overview
- Classes, attributes and operations
- UML Lines and Arrows
- Parameterized Classes, Interfaces and Utilities
- Objects, Associations
- **Inheritance**
- Patterns, Constraints and Contracts

Generalization

A <u>subclass</u> specializes its superclass:

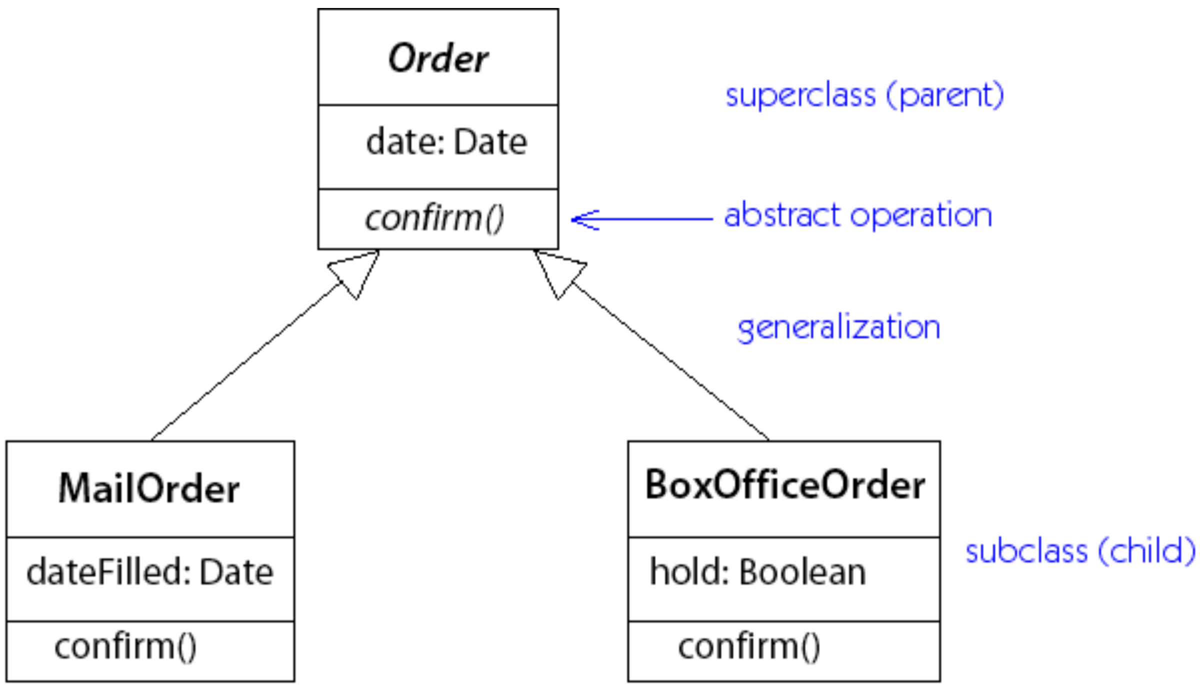


Figure 4-7. *Generalization notation*

What is Inheritance For?

- New software often builds on old software by *imitation*, *refinement* or *combination*.
- Similarly, classes may be *extensions*, *specializations* or *combinations* of existing classes.

Generalization expresses ...

Conceptual hierarchy:

- conceptually related classes can be organized into a *specialization* hierarchy
  - — people, employees, managers
  - — geometric objects ...

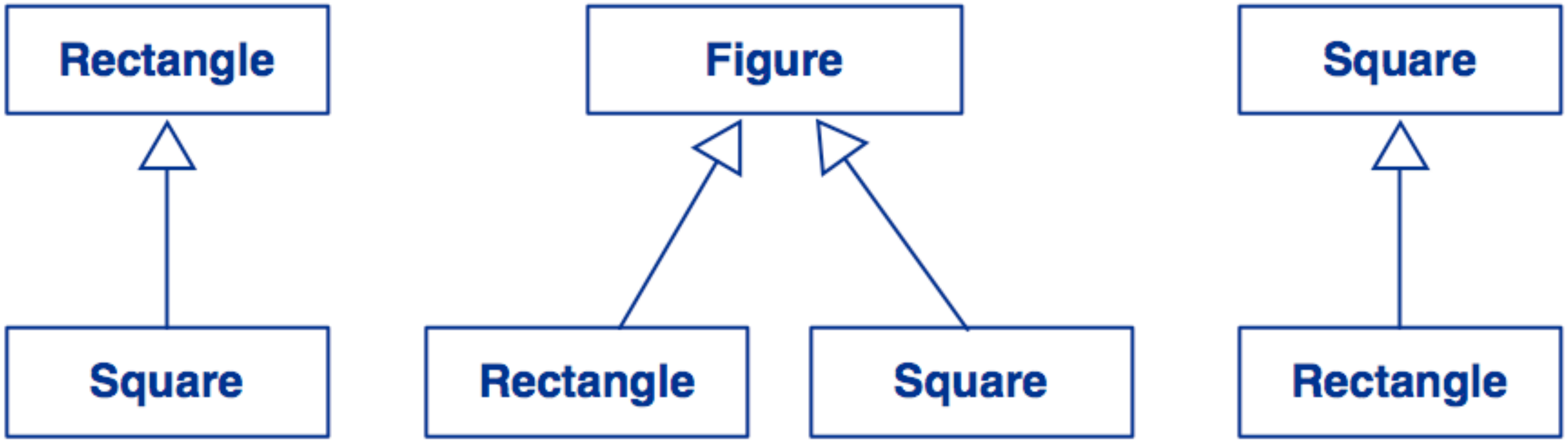
Polymorphism:

- objects of distinct, but related classes may be *uniformly treated* by clients
  - — array of geometric objects

Software reuse:

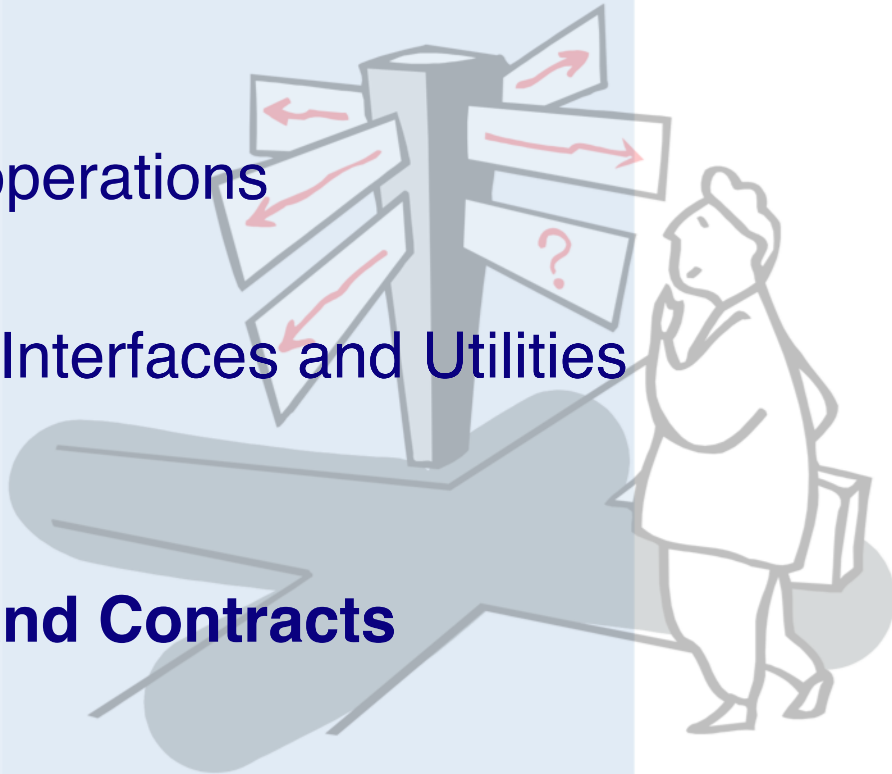
- related classes may *share* interfaces, data structures or behaviour
  - — geometric objects ...

The different faces of inheritance



Is-a *Polymorphism* *Reuse*

Roadmap



- UML Overview
- Classes, attributes and operations
- UML Lines and Arrows
- Parameterized Classes, Interfaces and Utilities
- Objects, Associations
- Inheritance
- **Patterns, Constraints and Contracts**

Design Patterns as Collaborations

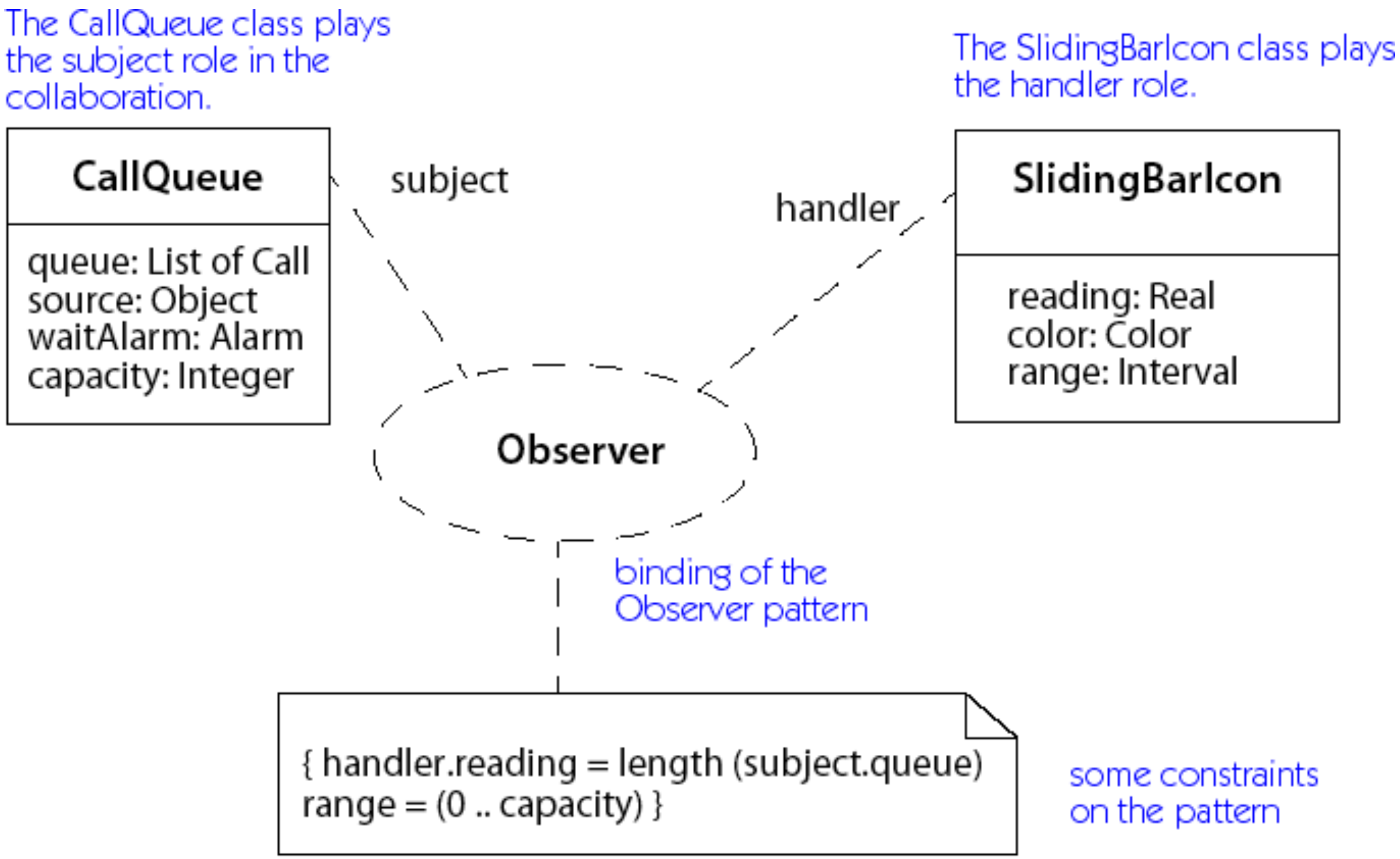


Figure 13-144. *Binding of a pattern to make a collaboration*

Constraints

Constraints are *restrictions* on values attached to classes or associations.

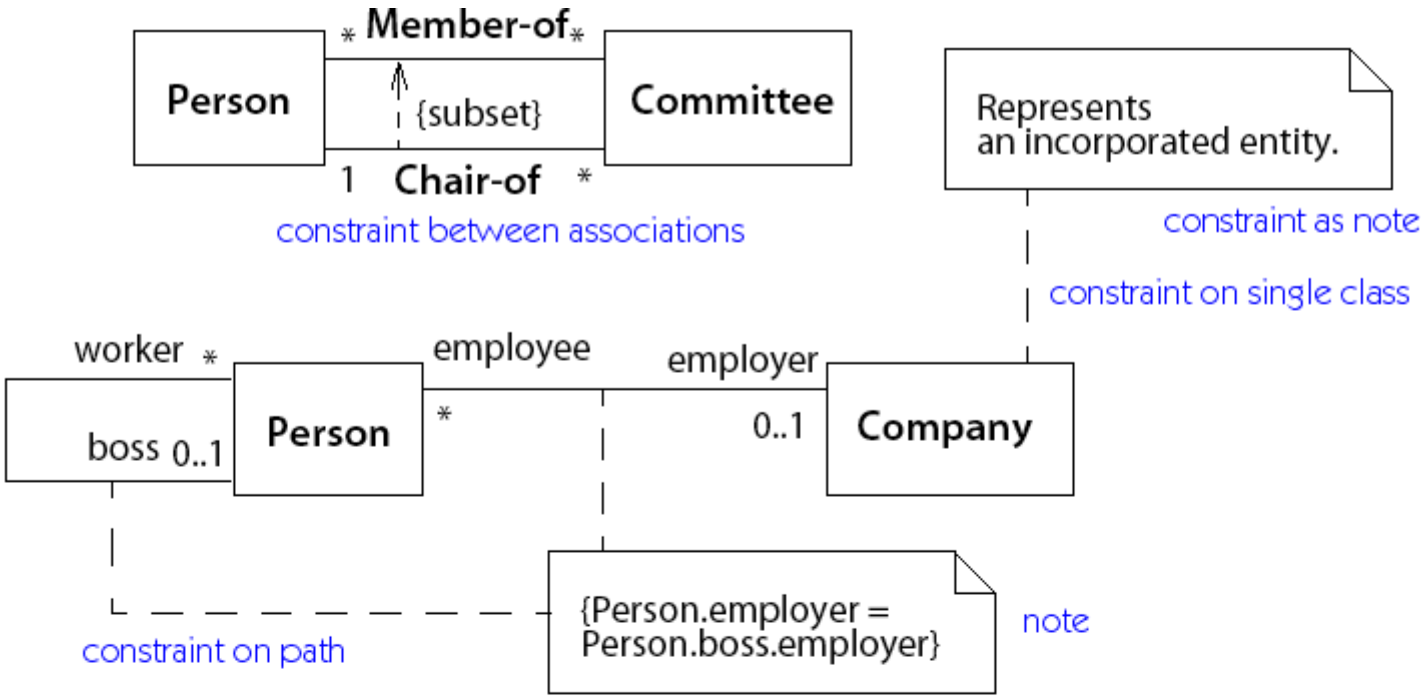


Figure 4-12. *Constraints*

OCL — Object Constraint Language

> Used to express queries and constraints over UML diagrams
  - Navigate associations:
    - *Person.boss.employer*
  - Select subsets:
    - *Company.employee->select(title="Manager")*
  - Boolean and arithmetic operators:
    - *Person.salary < Person.boss.salary*

www.omg.org

Design by Contract in UML

Combine constraints with stereotypes:

NB: «invariant», «precondition», and «postcondition» are predefined in UML.

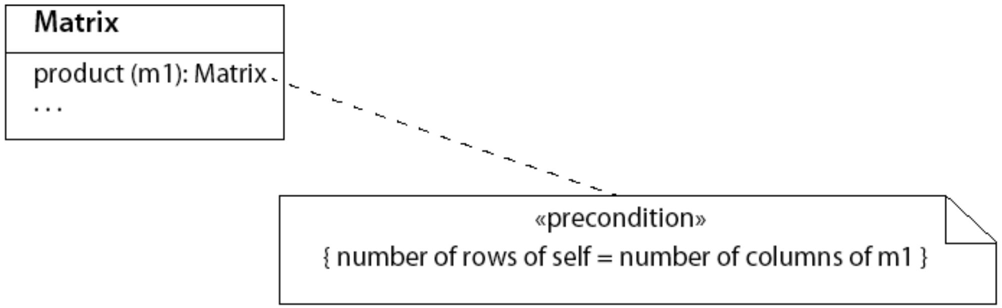


Figure 13-147. *Precondition*

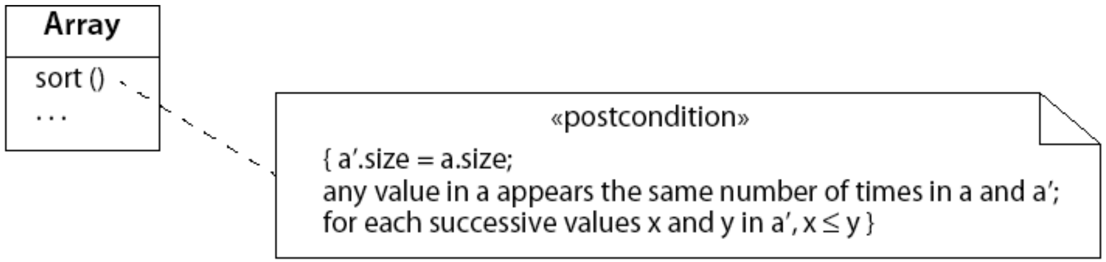


Figure 13-145. *Postcondition*

Using the Notation

During Analysis:

- — Capture classes visible to *users*
- — Document *attributes and responsibilities*
- — Identify *associations and collaborations*
- — Identify *conceptual hierarchies*
- — Capture all *visible features*

During Design:

- — Specify *contracts and operations*
- — *Decompose* complex objects
- — Factor out *common interfaces* and functionalities

The graphical notation is only one part of the analysis or design document. For example, a data dictionary cataloguing and describing all names of classes, roles, associations, etc. must be maintained throughout the project.

What you should know!

- How do you represent classes, objects and associations?
- How do you specify the visibility of attributes and operations to clients?
- How is a utility different from a class? How is it similar?
- Why do we need both named associations and roles?
- Why is inheritance useful in analysis? In design?
- How are constraints specified?

Can you answer the following questions?

- Why would you want a feature to have class scope?
- Why don't you need to show operations when depicting an object?
- Why aren't associations drawn with arrowheads?
- How is aggregation different from any other kind of association?
- How are associations realized in an implementation language?

License