BON

Business Object Notation

What is it?

- Notation for modeling object oriented software

What is it? – 2

- Notation for modeling object oriented software
  - » **Static: specifies classes, class relationships**

What is it? – 3

- Notation for modeling object oriented software
  - » **Static: specifies classes, class relationships**
  - » **Dynamic: behavioural properties**

What is it? – 4

- Notation for modeling object oriented software
  - » **Static: specifies classes, class relationships**
  - » **Dynamic: behavioural properties**

- Method
  - » **Guidelines to be used when producing specifications and descriptions**

What is it? – 5

- Notation for modeling object oriented software
  - » **Static: specifies classes, class relationships**
  - » **Dynamic: behavioural properties**

- Method
  - » **Guidelines to be used when producing specifications and descriptions**

» Does not include

  - » **Entity-Relation models**
  - » **Finite state machines**

Characteristics of the Notation

- Simplicity
  - » **Concentrate on design aspects of the method**

Characteristics of the Notation – 2

- Simplicity
  - » **Concentrate on design aspects of the method**
- Generality
  - » **Not restricted to application domains**

Characteristics of the Notation – 3

- Simplicity
  - » **Concentrate on design aspects of the method**
- Generality
  - » **Not restricted to application domains**
- Design by Contract
  - » **Assertions for classes and features**

Characteristics of the Notation – 4

- Simplicity
  - » **Concentrate on design aspects of the method**
- Generality
  - » **Not restricted to application domains**
- Design by Contract
  - » **Assertions for classes and features**
- Two views
  - » **Graphical**
  - » **Textual** $\longrightarrow$ **Eiffel**

Characteristics of the Notation – 5

- Seamlessness
  - **Smooth transition from requirements through design to implementation all in one form of model**

Characteristics of the Notation – 6

- Seamlessness
  - » **Smooth transition from requirements through design to implementation all in one form of model**
- Reversibility
  - » **Direct mapping of design concepts to and from implementation concepts**

Characteristics of the Notation – 7

- Seamlessness
  - » **Smooth transition from requirements through design to implementation all in one form of model**
- Reversibility
  - » **Direct mapping of design concepts to and from implementation concepts**
- Scalability
  - » **Scales up to large designs**

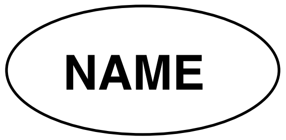
Tool Support

- Bon tools

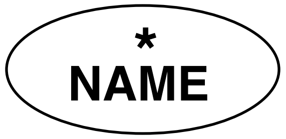
- Eiffel diagrams

Compressed Classes

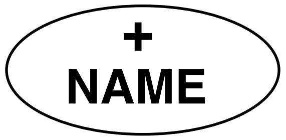
Use to draw views with lots of classes
- bird's eye view
- early stages of design



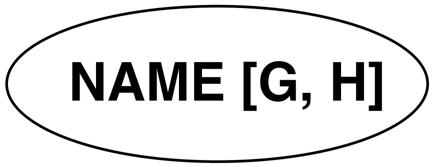
Shortest form



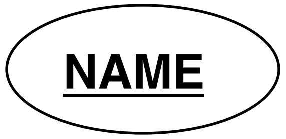
Deferred



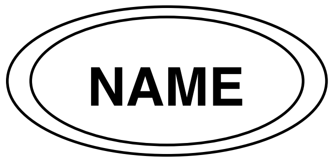
Implemented



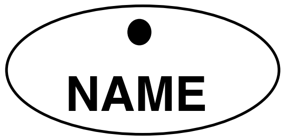
Parameterized



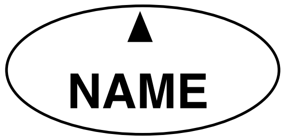
Reused library



Root
Instances may be separate processes

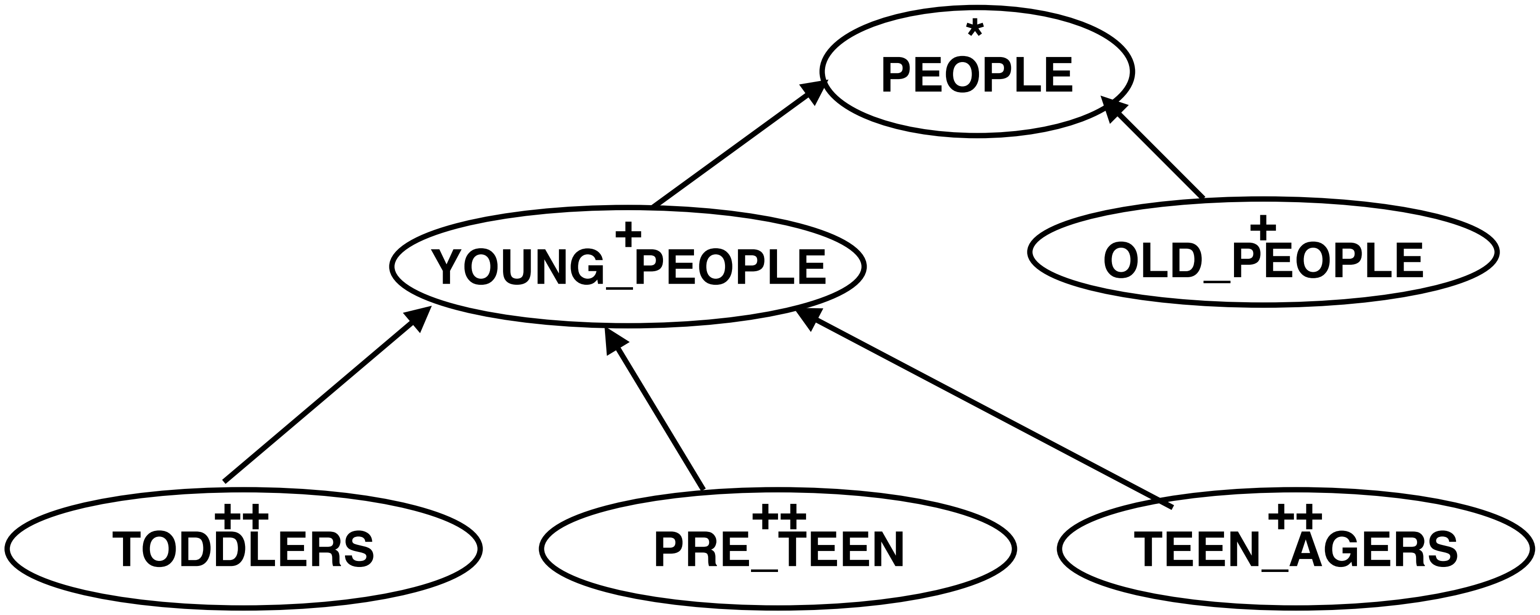


Persistent
Inherit STORABLE



Interfaces with outside world

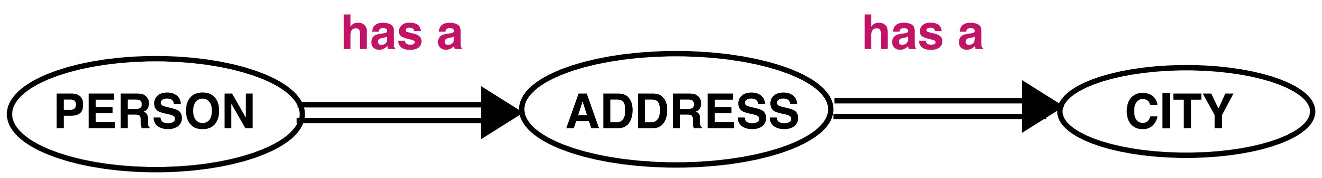
Inheritance Relations



Client–Supplier Association

Client A uses the services of **supplier** B

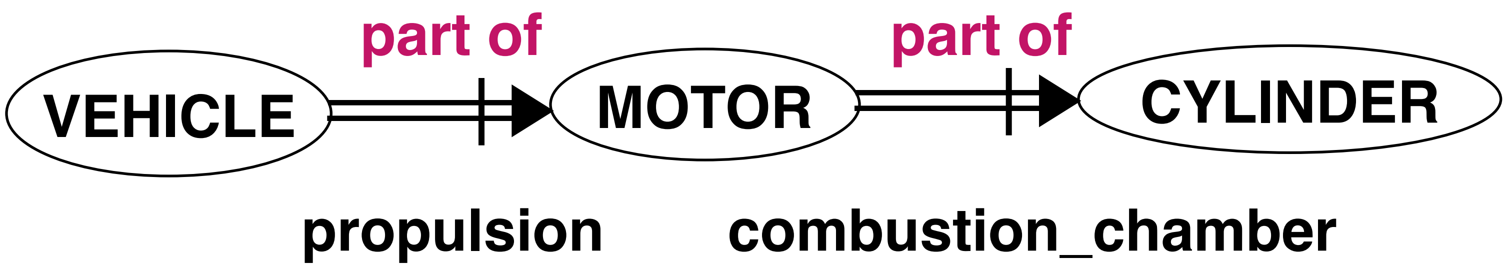
Each client instance may be attached to one or more supplier instances



Client–Supplier Aggregation

Client A uses the services of **supplier** B

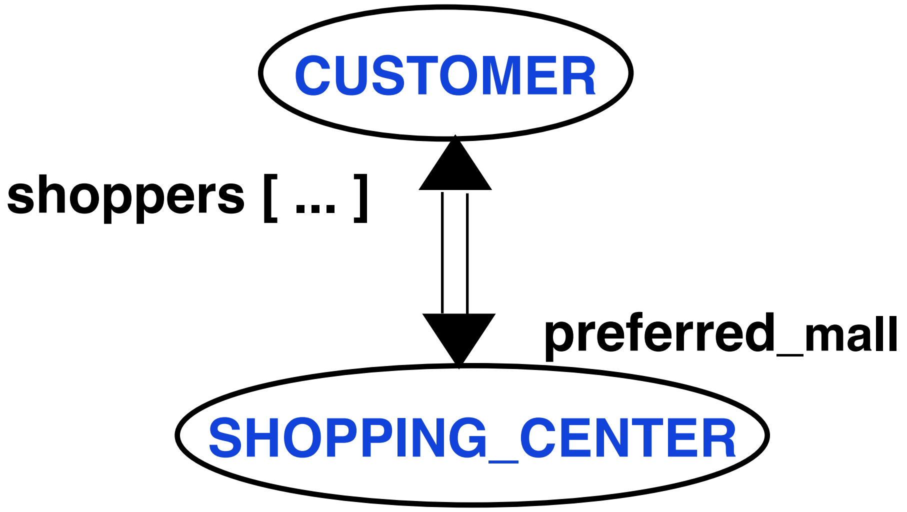
Each client instance is attached to one or more supplier instances that represent integral parts of the client instance



Difference between association and aggregation?

- **Consider expanded vs reference use**
- **Consider what happens when the client gets deleted**

Bidirectional Uses Links



- Client feature label is at the supplier side
- Generic classes can be used in labels

  Leave parameter unspecified

- Useful for recursive structures

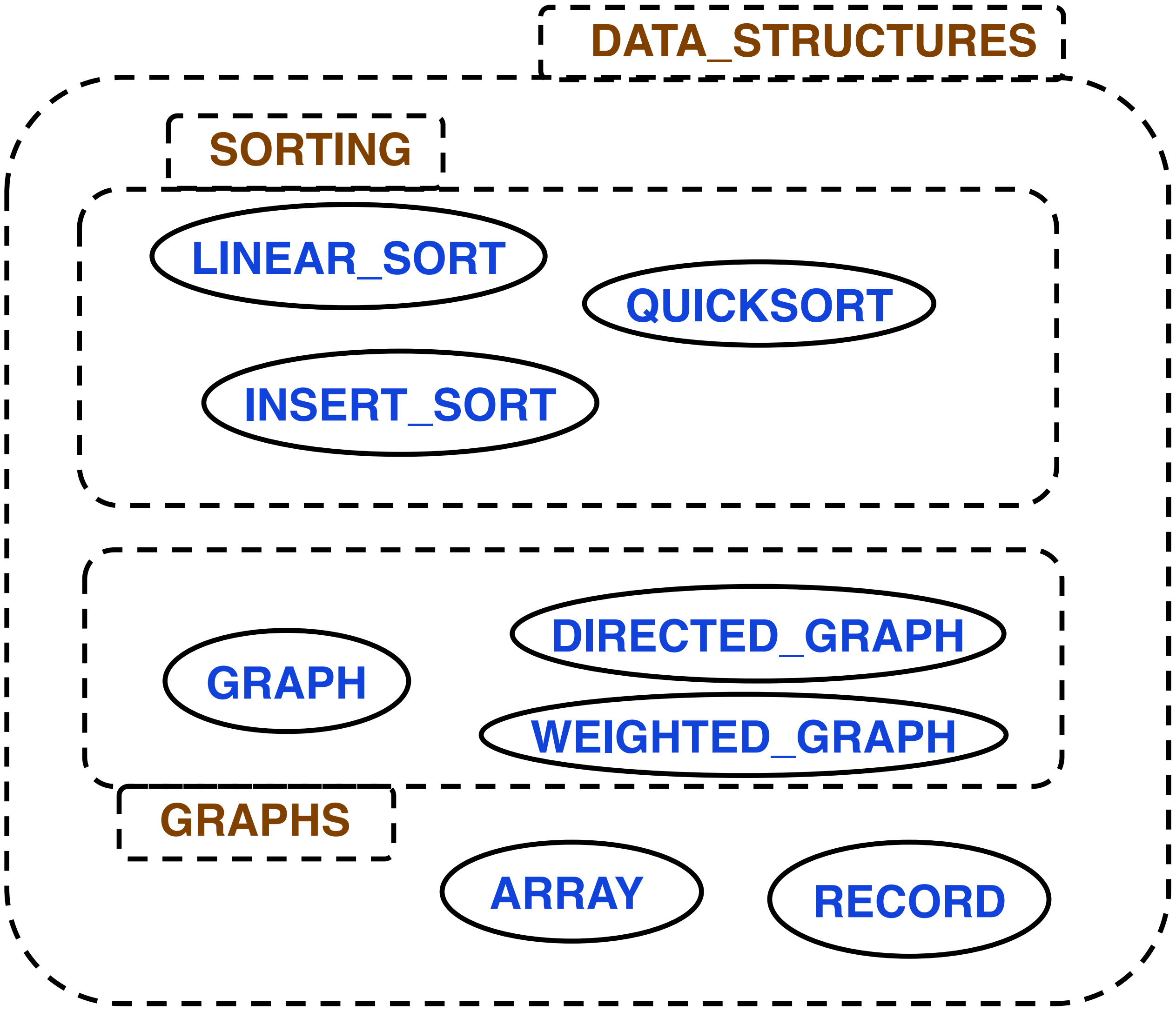
  lists, trees, graphs

Cluster

- Represents a group of classes, and possibly other clusters, according to some point of view

- Classes may be grouped differently depending on the characteristics of the specification one wants to highlight
  - » **Subsystem functionality, user categories, abstraction level, et cetera**

Cluster Example



Cluster Properties

- Clusters can be shrunk to hide their contents
  - » **Keep only the cluster name**

Cluster Properties – 2

- Clusters can be shrunk to hide their contents
  - » **Keep only the cluster name**
- Every class belongs to exactly one cluster

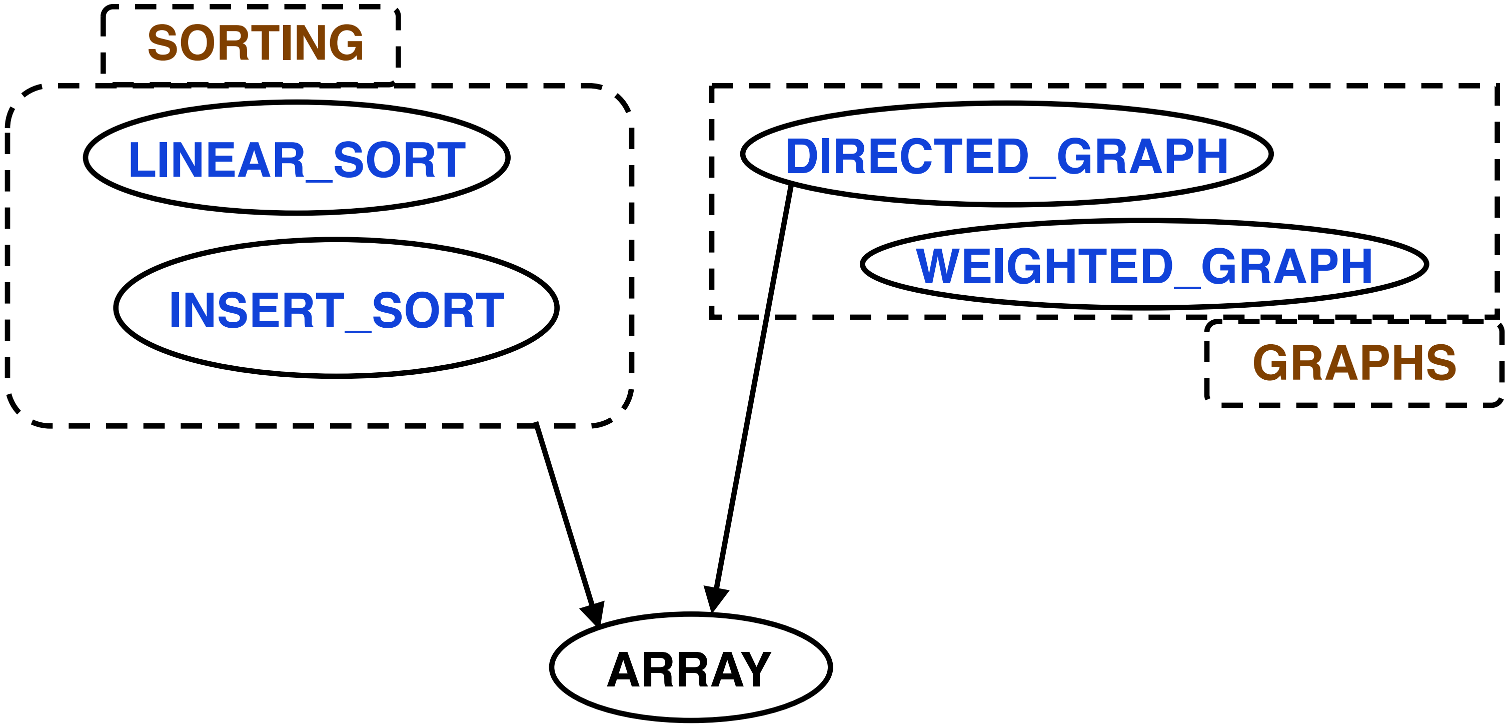
Cluster Properties – 3

- Clusters can be shrunk to hide their contents
  - » **Keep only the cluster name**
- Every class belongs to exactly one cluster
- Not a language construct; just a mechanism for dealing with abstraction

Cluster Properties – 4

- Clusters can be shrunk to hide their contents
  - » **Keep only the cluster name**
- Every class belongs to exactly one cluster
- Not a language construct; just a mechanism for dealing with abstraction
- Implement in Eiffel with directory structure
  - » **Each cluster is a directory**

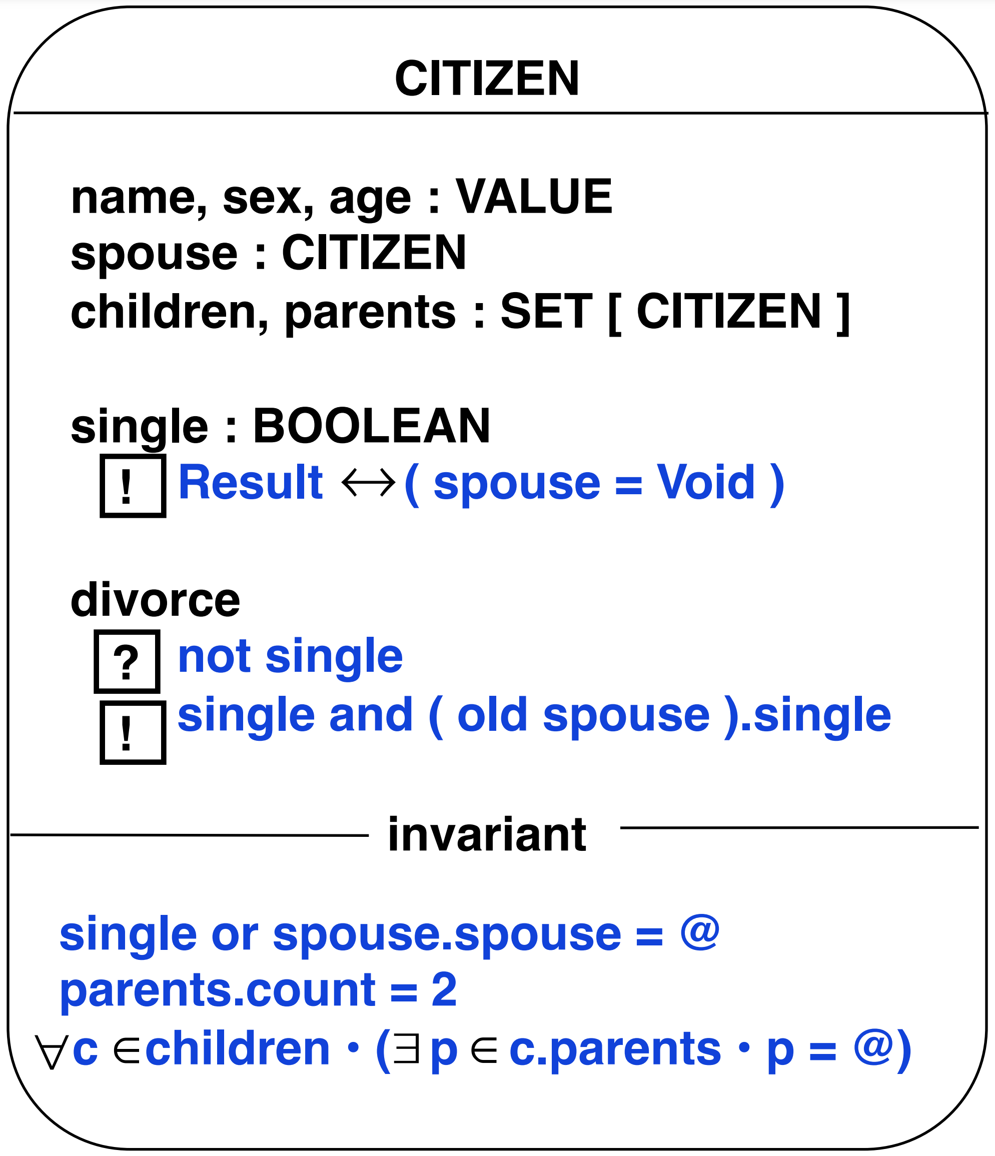
Inheritance & Clusters



- All classes in sorting inherit from ARRAY
- Only DIRECTED_GRAPH inherits from ARRAY

Graphical BON Class (Uncompressed)

No need to show all features, just those of interest for the view



CITIZEN

name, sex, age : VALUE
spouse : CITIZEN
children, parents : SET [CITIZEN]

single : BOOLEAN
$\boxed{!}$ Result $\leftrightarrow$ (spouse = Void)

divorce
$\boxed{?}$ not single
$\boxed{!}$ single and (old spouse).single

invariant

single or spouse.spouse = @
parents.count = 2
$\forall$c $\in$children • ($\exists$ p $\in$ c.parents • p = @)

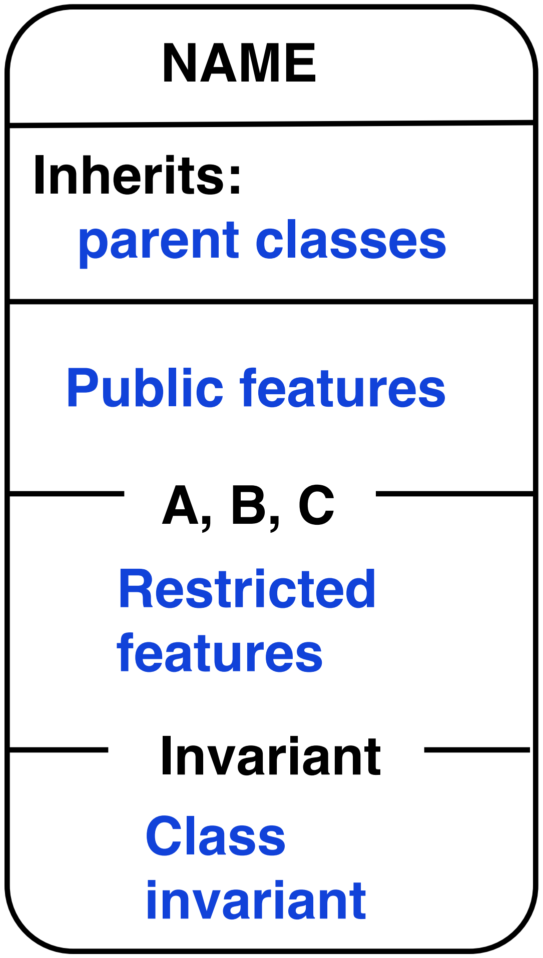
Assertion Language

- Queries and commands can be documented with a precondition and a postcondition
- Follow Eiffel language with respect to inheritance and redefinition of assertions
- Use predicate calculus and set theory

Graphical Form	Textual Form
$\boxed{?}$ precondition	**require** precondition
$\boxed{!}$ postcondition	**ensure** postcondition
the_invariant	**invariant** the_invariant

Typed Class Interface

- Early phases concentrate on public features
- Restricted features produced during detail design
- Arbitrary number of sections, each with export list
- Each feature has a signature and optionally a behavioural specification
- Conventions
  - » **Classes all in upper case**
  - » **features all in lower case**
  - » **use underscore for longer names**



Class Feature Decorators

Feature names have an optional decorator showing status

name* – deferred

name^{+} – effective

name^{++} – redefined

name : TYPE – result type

---

new_name { ^ CLASS_NAME . old_name }
– rename clause

name : { TYPE – aggregation result type

$\rightarrow$ name : TYPE – input argument

Class Feature Signatures

- Each feature has a signature

  attributes & no parameter queries

  name : TYPE

  queries

  name (arg : ARG_TYPE; ...) : RESULT_TYPE

  commands

  name (arg : ARG_TYPE; ...)

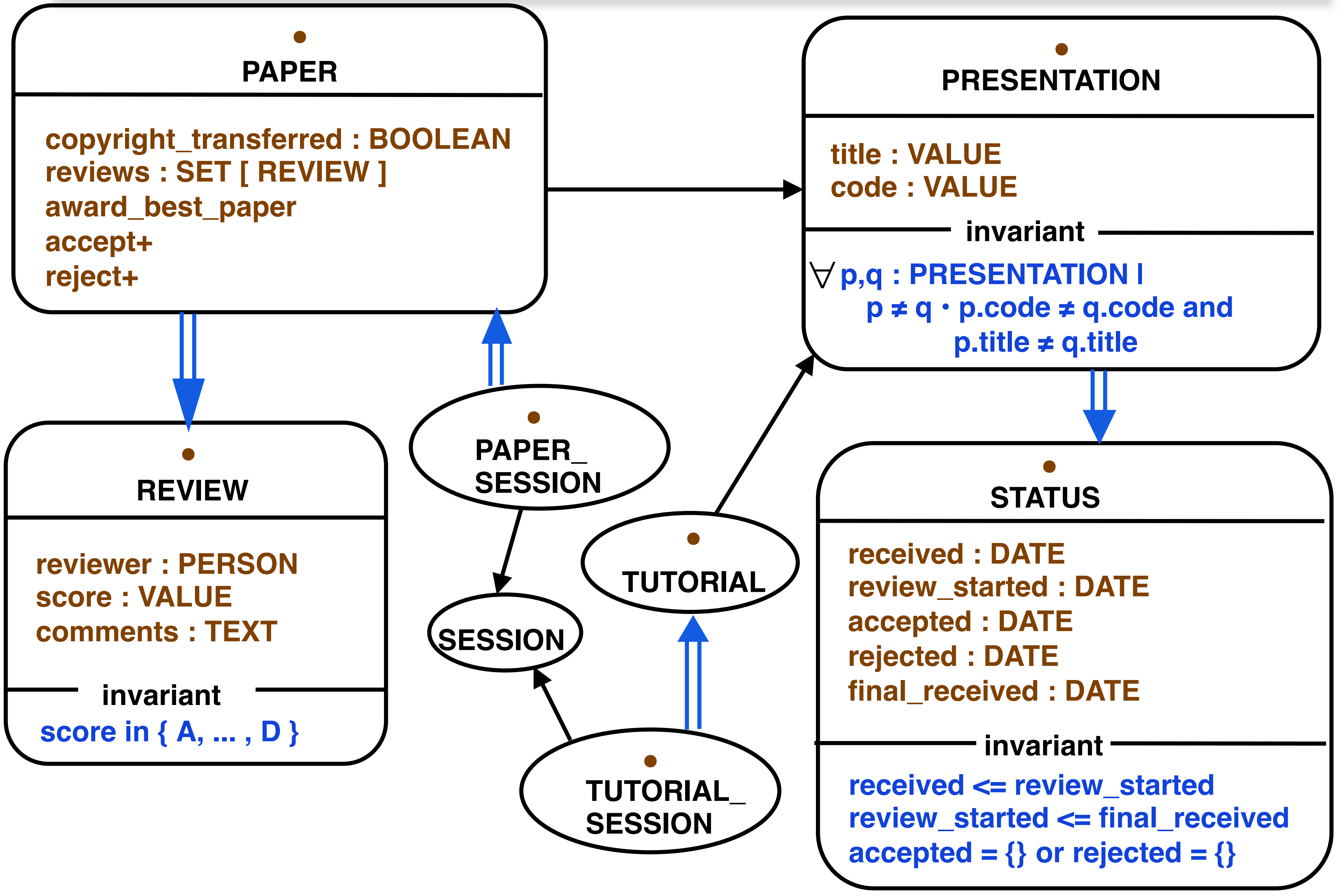
- Types may be expanded types

Graphical View Rule

Graphical view is not used for just one class

Always have two or more classes with inheritance and/or uses relations among them

Views Show Part of a Design



Gunnar Gotshalks