Class Diagrams II
(More advanced constructs)

Lecture : 10
Date : 8-11-2005

Yannis Tzitzikas
University of Crete, Fall 2005

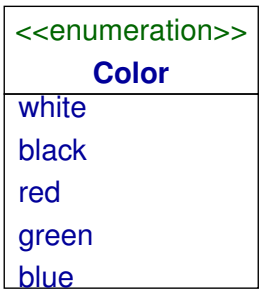
Outline

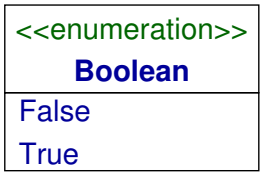
- Enumerations
- Class Scope for Operations and Attributes
- Derived Associations and Attributes
- Frozen
- Aggregation and Composition
- Collections for Multivalued Association Ends
- Association Class
- Qualified Associations
- Multiple and Dynamic Classification
- Parameterized Class
- Interfaces and Abstract Classes

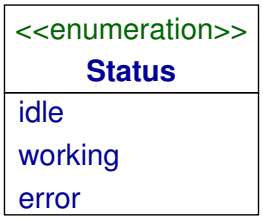
Enumerations

Are used to show a fixed set of values that do not have any properties other than their symbolic value.

Notation: <<enumeration>>

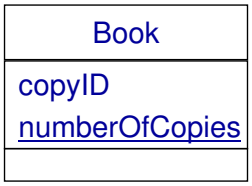






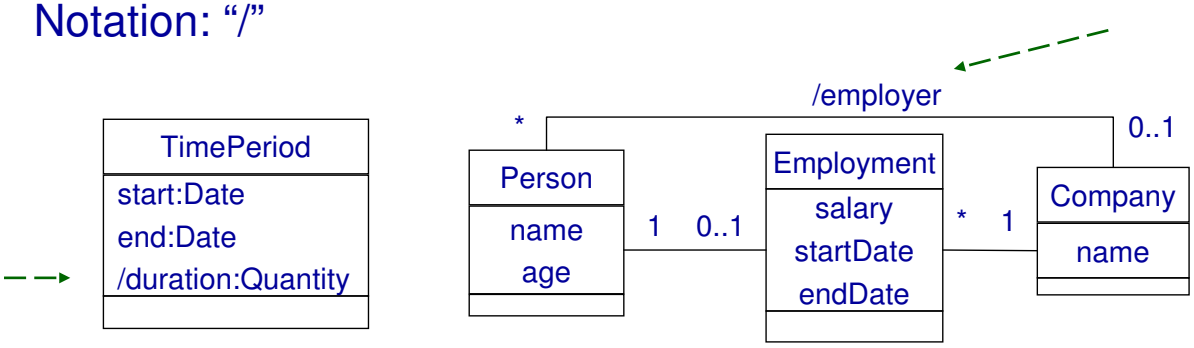
Class Scope Operations and Attributes

- Operations or attributes that apply to the class (not at the instances)
  - like static members in C++ or Java
- We denote them by underlying them



Derived Associations and Attributes

- Can be calculated from other Associations/Attributes
  - e.g. we can compute Person.age from Person.birthDate
- Notation: "/"



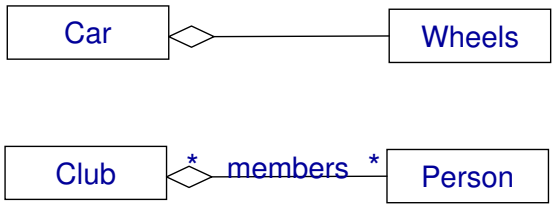
- *specification*:
  - derived features = indicate constraints between values, not a statement of what is calculated and what is stored
  - it will make the programmer to implement it
- *implementation*
  - e.g. cache (e.g. the duration is store in cache for efficiency)

Frozen

- Constraint on an attribute or association end
  - the value of the (attr. / association end) may not change during the lifetime of the source object
  - the initial value (even if null) is preserved
  - typically the constructors set these values
- Constraint on an class
  - all association ends and attributes associated with that class are frozen
- Frozen ≠ Read Only
  - e.g. the age (calculated) may be read-only but not frozen
  - read only is not UML standard
- Notations: {frozen}, {read only}

Aggregation and Composition

Aggregation; συνάθροιση, συσσωμάτωση
Composition: συγκρότηση, σύνθεση



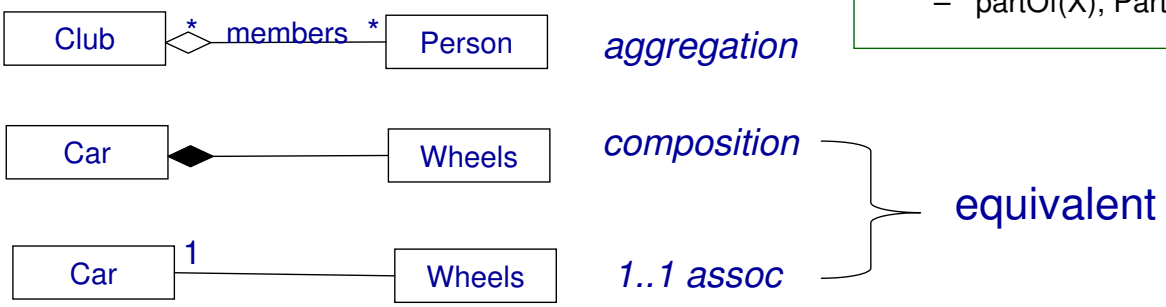
- **Aggregation** ≈ partOf
- But what is the difference with Association ?
  - Due to vagueness “think of it as a modelling placebo” [3 amigos]
- UML supports it but with no semantics

Aggregation and Composition

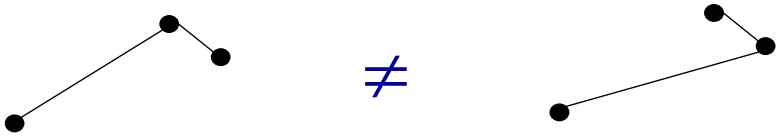
Composition: a stronger variety of aggregation

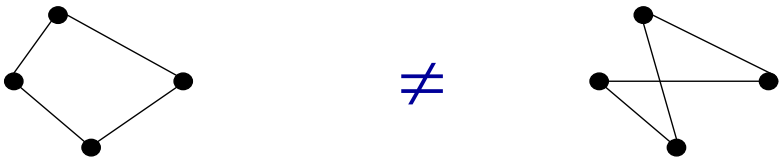
- the part object may belong to only one whole and the parts are expected to live and die with the whole
- deletion of the whole ===> cascades to its parts
  - but this can be also captured by multiplicity 1..1

- dependent part
  - delete(X) => delete(part(X))
- exclusive part
  - partOf(X), PartOf(Y) => X=Y



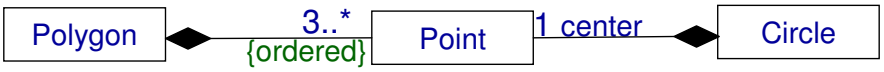
Aggregation and Composition





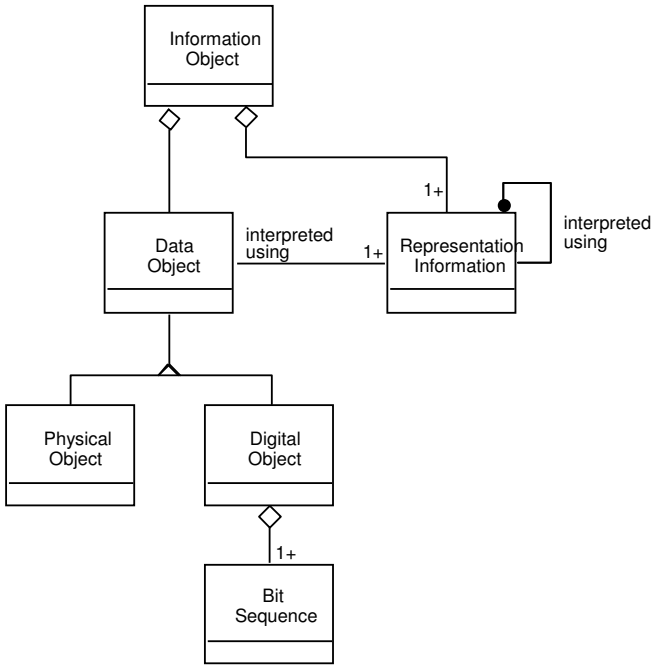
Collections for Multivalued Association Ends

- **Multivalued end**: one whose multiplicity's upper bound > 1 (e.g. *)
- Usual convention: unordered set (no order, no duplicates). However, we can change this assumption:
  - {ordered} : target objects from a list
  - {bag}: duplicates allowed
  - {hierarchy}: the target objects form a hierarchy
  - {dag}: the target objects form a directed acyclic graph

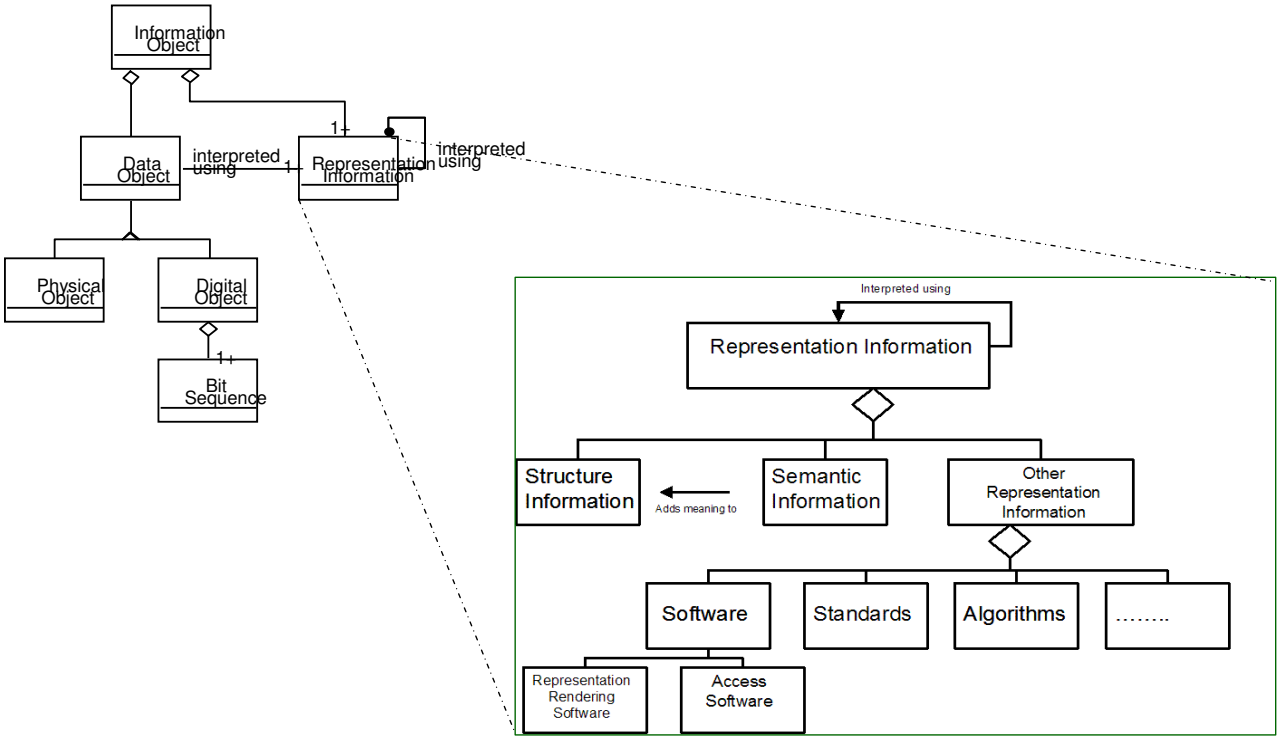


Example: OAIS Information Model

- *Reference Model for an Open Archival Information System (OAIS),* ISO 14721:2003
- OAIS = "An archive, consisting of an organization of people and systems, that has accepted the responsibility to preserve information and make it available for a Designated Community"

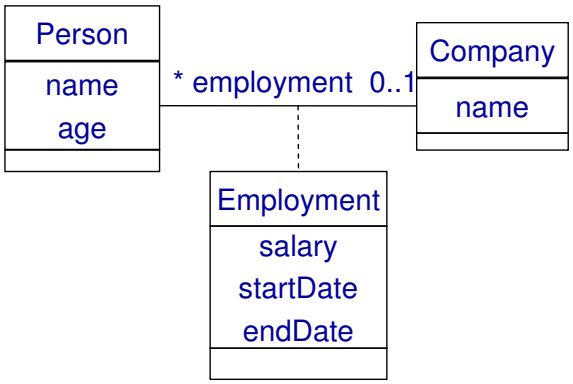


Example: OAIS Information Model (cont)



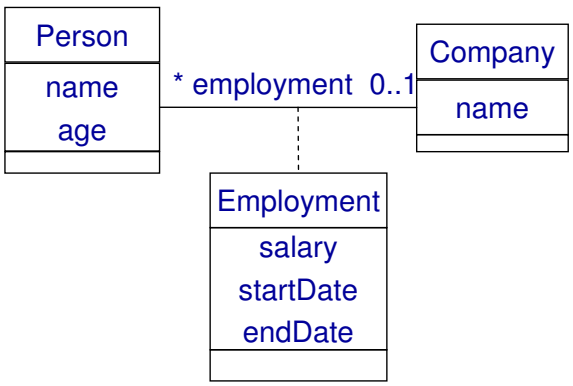
Association Class

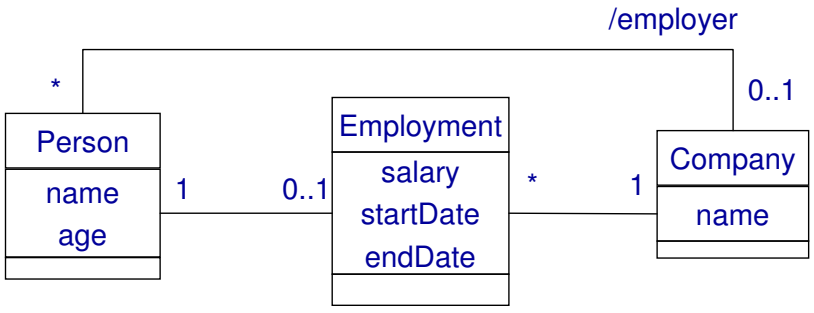
Allow to add attributes, operations and other features to associations



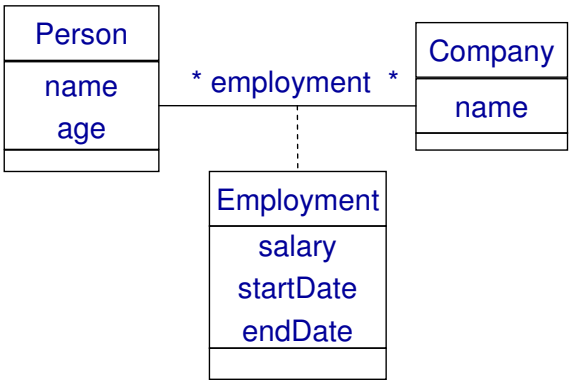
The association class adds the extra constraint that there can be only one instance of the association class between any two participating objects

Association Class
Promoting an Association Class to a Full Class





Association Class

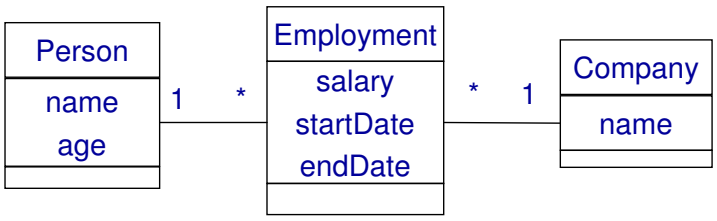


Are they equivalent?

NO!

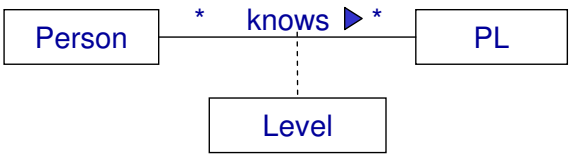
This diagram cannot capture the case of a person that has worked for the same company in different periods.

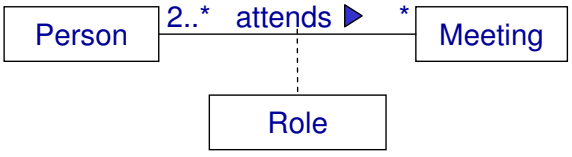
This can be captured here.



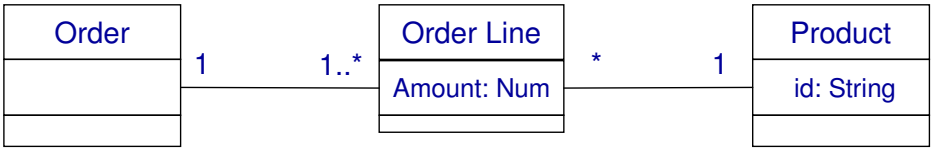
Association Class

Other examples where the association class constraint is useful.

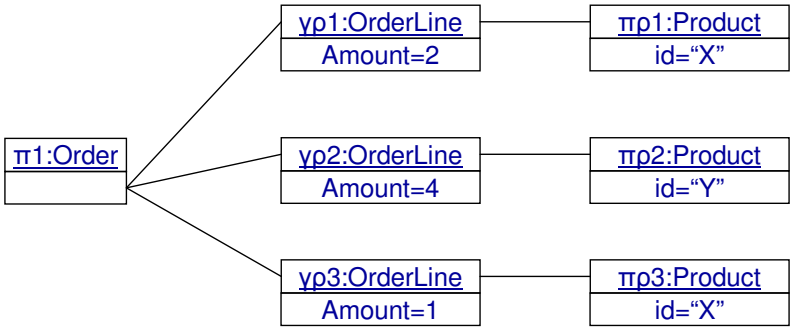




Another example

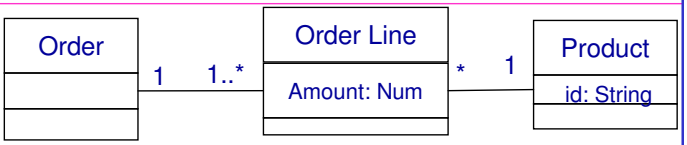


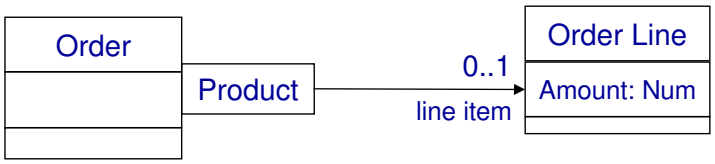
Object diagram:



The same product ("X") appears in two orderlines. How we could prohibit this?

Qualified Associations





Conceptual Perspective:

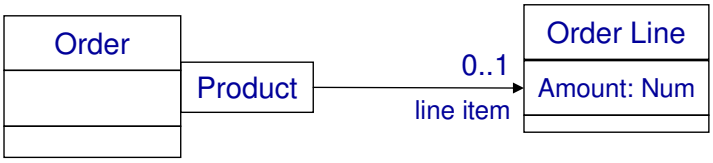
- we cannot have >1 orderlines (in the same Order) about the same Product

Specification Perspective:

an interface like:

```
class Order {
        public OrderLine getLineItem (Product aProduct);
        public void addLineItem (Number amount, Product forProduct)
```

Qualified Associations



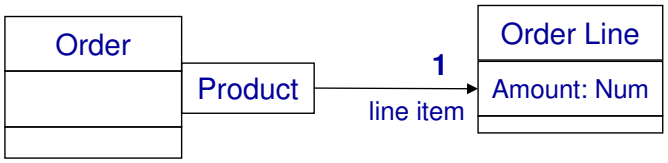
Implementation Perspective:

```
Class Order {
    private Map _lineItems;
}
```

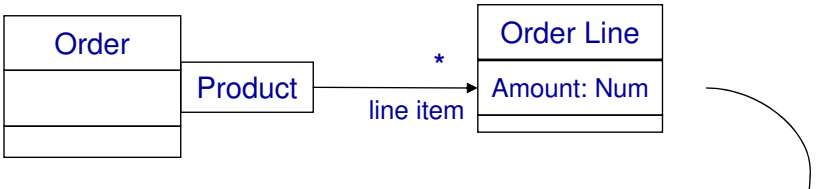
Similar constructs of PLs

- associative arrays, maps, dictionaries
- e.g in Java:
  - public interface **Map**
  - An object that maps keys to values. A map **cannot contain duplicate keys; each key can map to at most one value.**
  - The Map interface provides three *collection views*, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings.

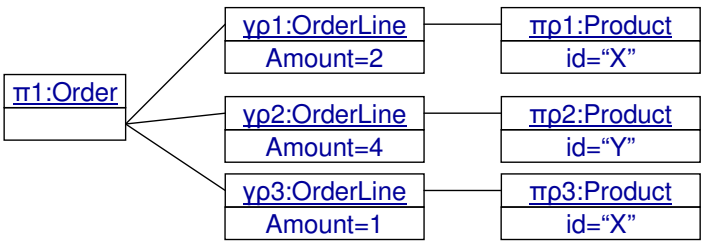
Qualified Associations & Multiplicities



There should be an orderline for every Product!



Compatible object diagram



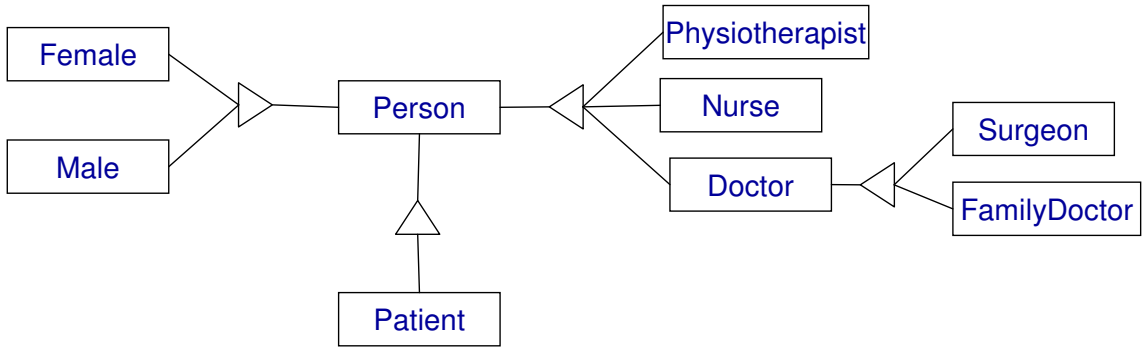
Multiple and Dynamic Classification

Classification refers to the relationship between an object and its type.

- Single vs Multiple classification
  - Single: an object belongs to one type
  - Multiple: an object belongs to several types
- Static vs Dynamic classification
  - Static: an object cannot change type
  - Dynamic: an object can change type

Single and static classification is not very flexible for conceptual models.

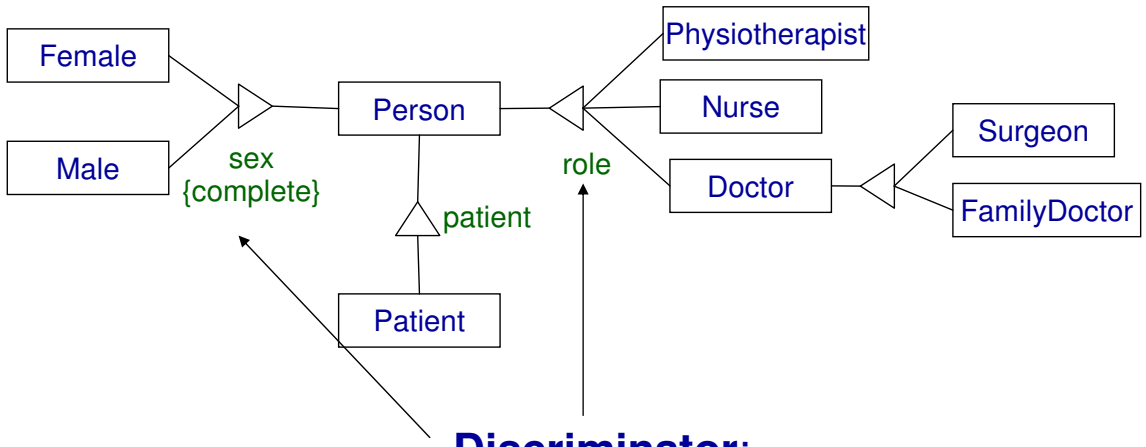
Multiple Classification



A person can be female and patient and nurse

However the model allows persons who are both male and female.

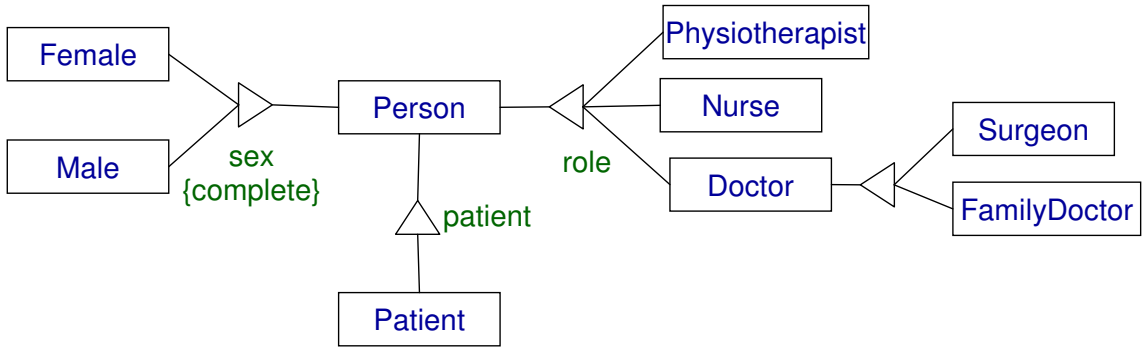
Multiple Classification Discriminators (UML V2: Generalization set)



Discriminator:

- indication of the basis of the subtyping
- all subtypes with the same discriminator <u>are disjoint</u>
- if a discriminator is marked by {complete} then any instance of the superclass must be an instance of one of the sybtypes of a group (the supertype is then called abstract).

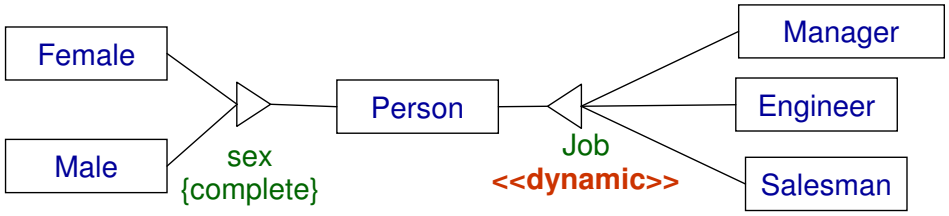
Multiple Classification Discriminators



- (Patient, Doctor)
  - is an illegal object because it misses sex
- (Female, Nurse, Surgeon)
  - is also illegal because it contains >1 types from the discriminator “role”

Dynamic Classification

- allows objects to change type within the subtyping structure
  - static classification does not

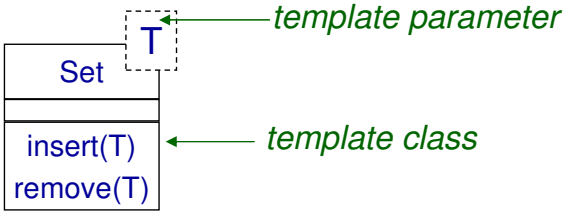


- multiple-dynamic classification needs care at the implementation perspective

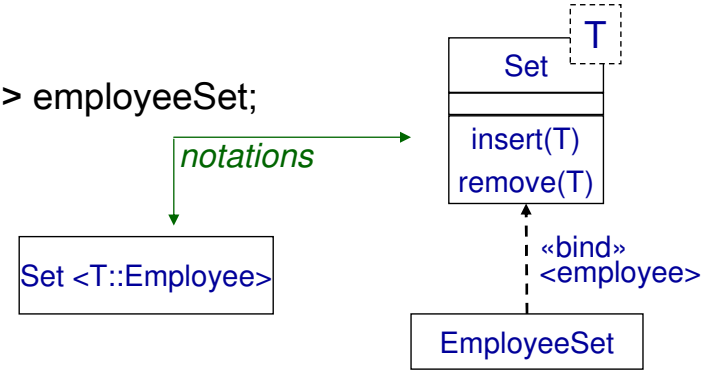
Parameterized Class

C++ has the notion of parameterized class or template useful for working with collections in a strongly typed language

```
Class Set <T> {
    void insert (T newElement);
    void remove (T newElement);
```



Then we can use Set <Employee> employeeSet;
this is called bound element



Interfaces and Abstract Classes in UML

Interface: a class that has only public operations with no method bodies

- So all of its features are abstract
- Like interfaces in Java, CORBA

- Notation: <<interface>>

Abstract class: a class that cannot be directly instantiated.

- Typically, an abstract class has one ore more operations that are abstract.
- Abstract operation: an operation with no implementation.

- Notation: italics, or {abstract}

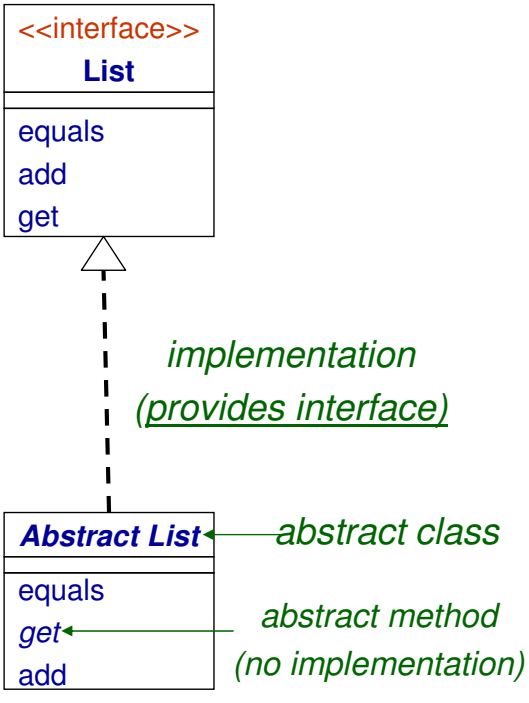
Interfaces and Abstract Classes in UML

Interface: a class that has only public operations with no method bodies

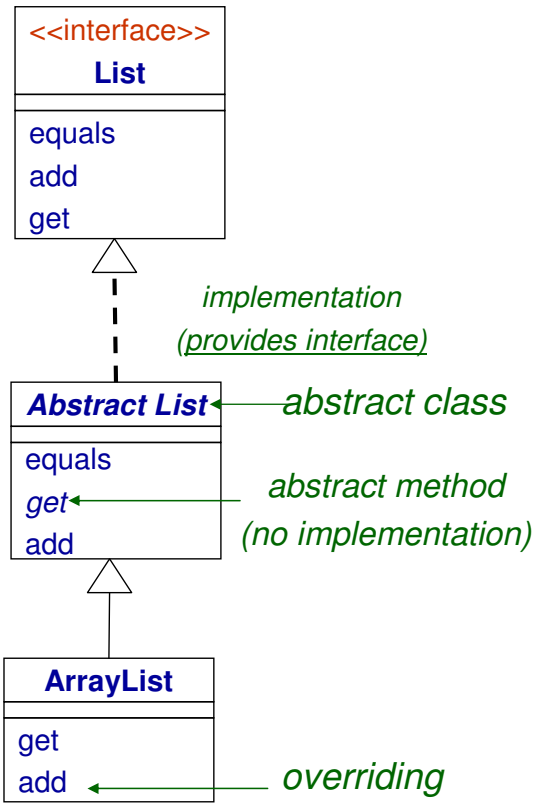
- Notation: <<interface>>

Abstract class: a class that cannot be directly instantiated.

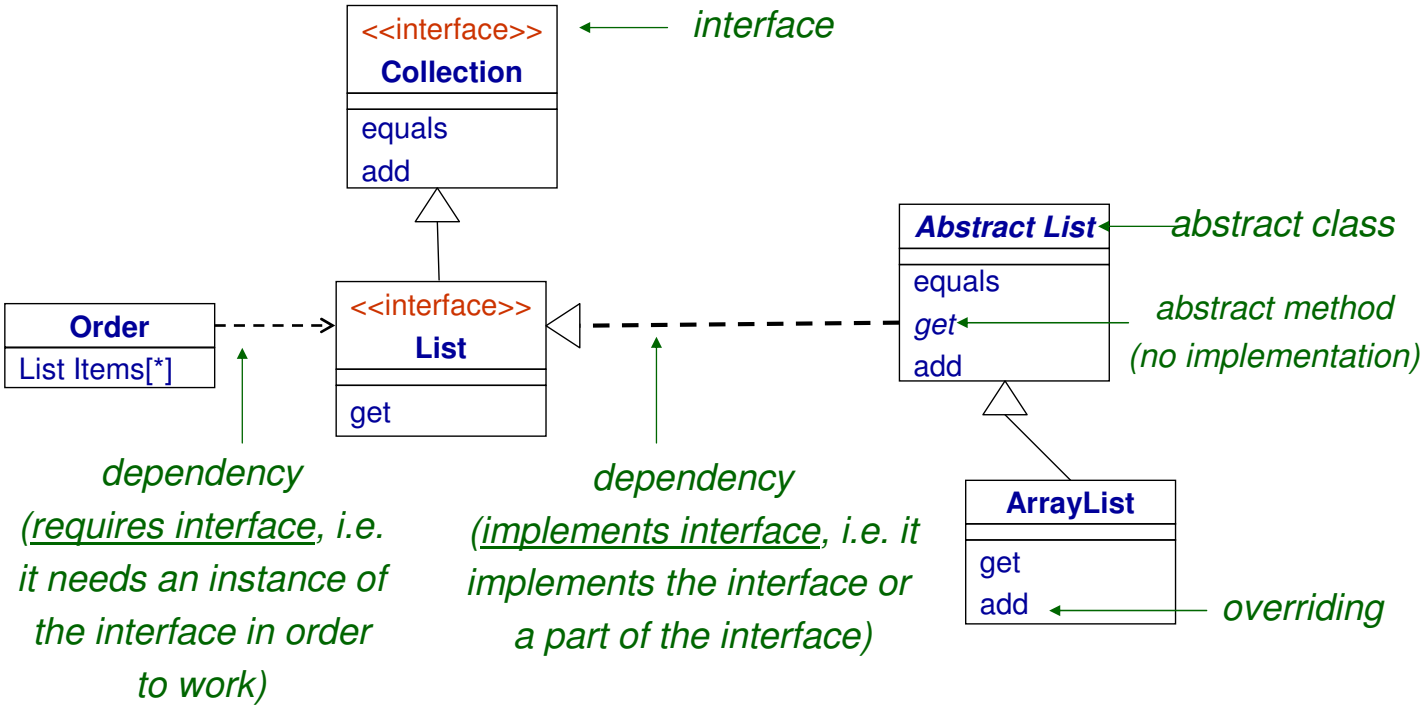
- Notation: italics, or {abstract}



Interfaces and Abstract Classes in UML

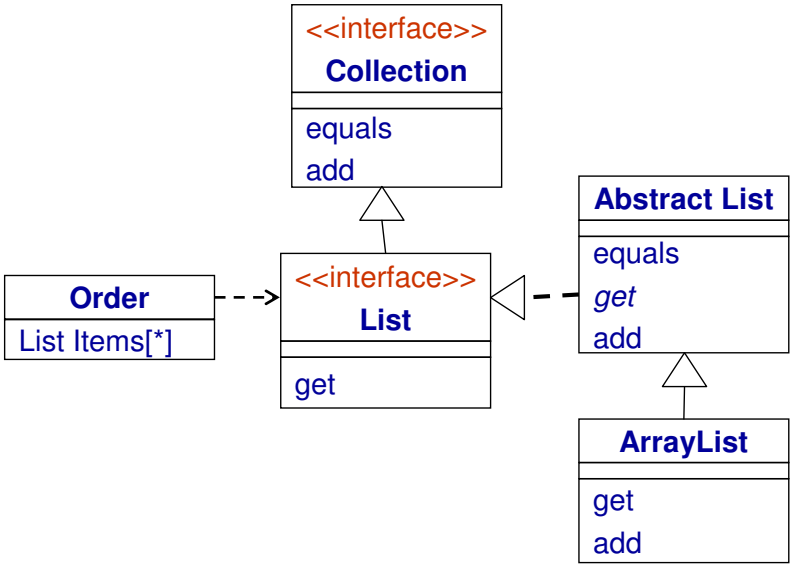


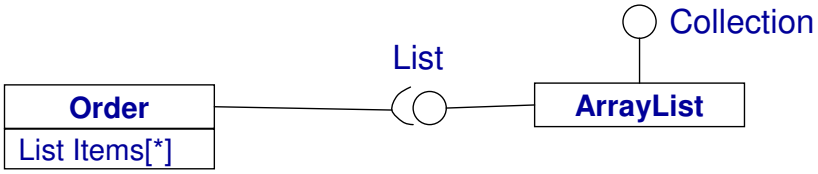
Interfaces and Abstract Classes: Example



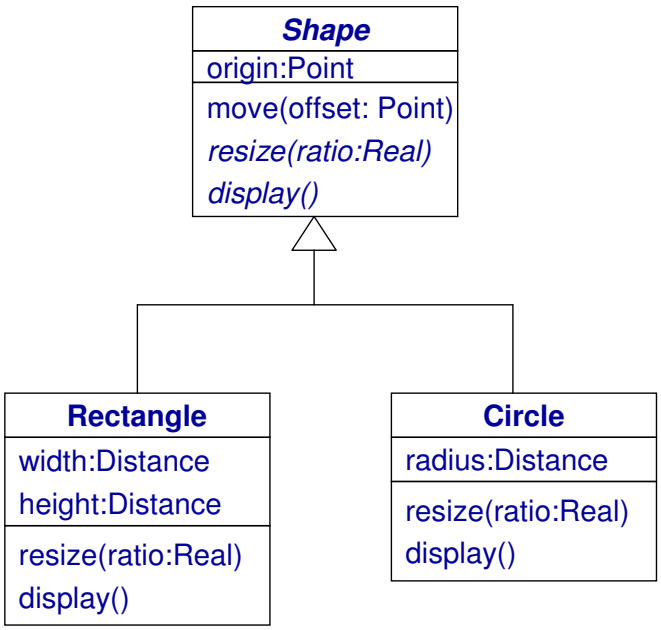
Dependency: w.r.t. updates (if the interface changes the implementation class should change too).

Interfaces and Abstract Classes: Example
A more compact representation

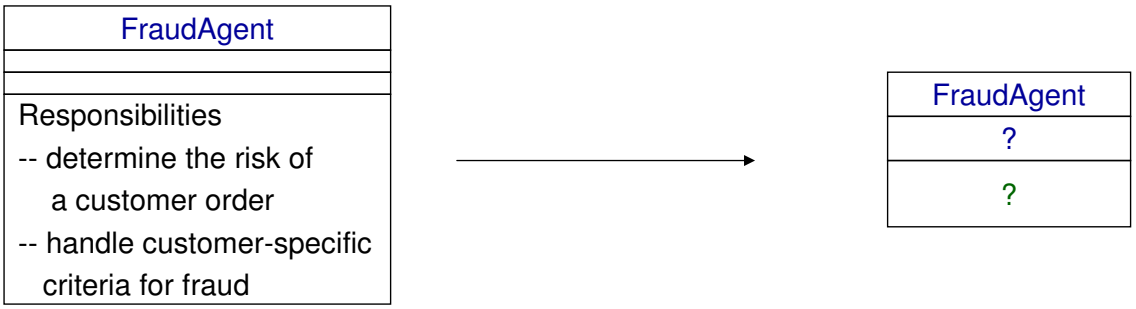




..



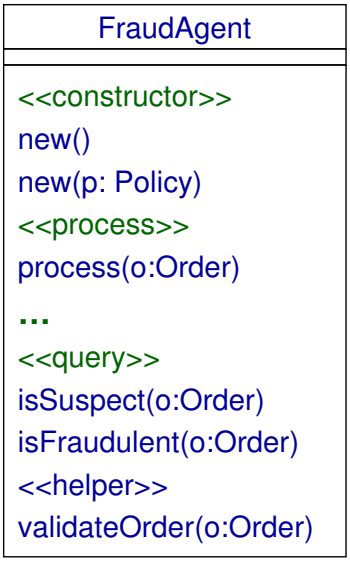
From Responsibilities to Attributes and Operations



- When drawing a class we don't have to show every attribute and every operation at once.
- We can choose to show only some or none of a class's attributes and operations.

Organizing Attributes and Operations

To better organize long list of attributes and operations, we can prefix each group with a descriptive category by stereotypes



Case Study
SIS-Telos

SIS-Telos

SIS-Telos: A knowledge representation language supporting a structurally object oriented data model

No operations

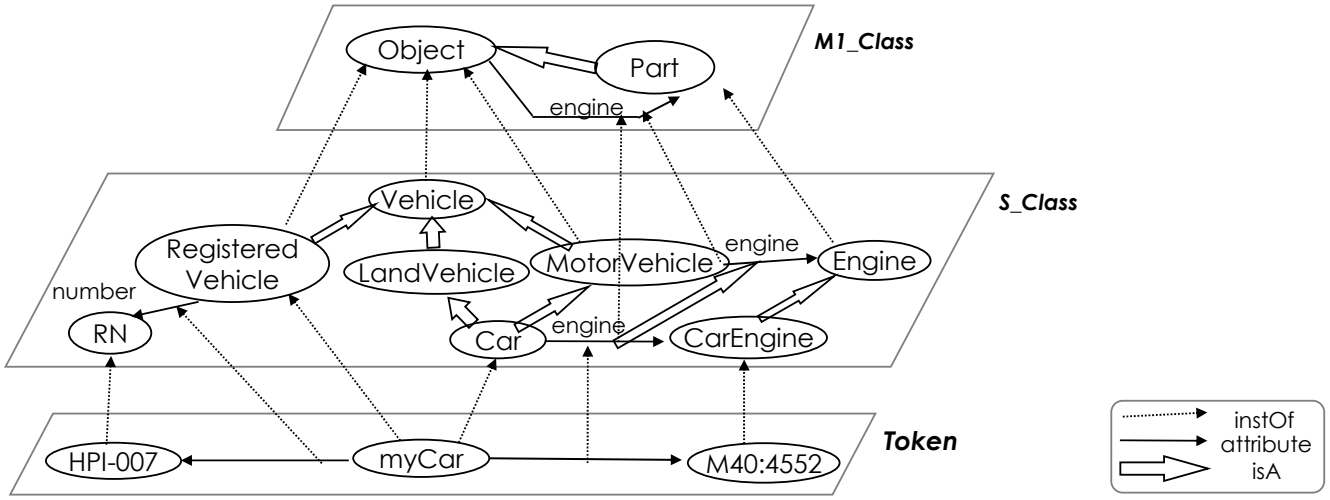
Features:

- Every object has a unique system identifier and a unique logical name
- Objects are structured along three main hierarchies:
  - the classification hierarchy — **instanceOf**
  - the generalization/specialization hierarchy — **isA**
  - the aggregation(attribute) hierarchy — **attribute**

SIS-Telos and SIS

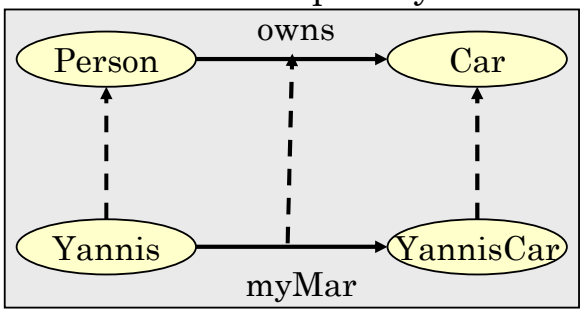
SIS-Telos
- object naming
- classification
- generalization
- attribution

Semantic Index System (SIS)
- DBMS functionality
- bidirectional link storage
- recursive (navigational) queries
- schema evolution at run-time

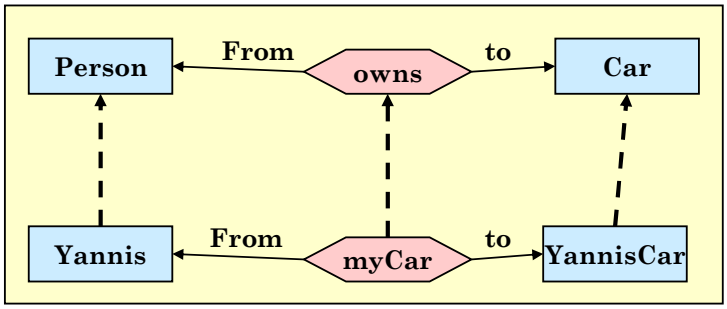


Objects are partitioned to **Individuals** and **Attributes**

Conceptually



SIS-Telos Implementation



Logical Names:
Individual: "**Yannis**"
Attribute: "**owns from Person**"
"**myCar from Yannis**"

Primitive Data Types:
Telos_String
Telos_Integer
Telos_Real
Telos_Time

Individual Object

Attribute Object

Instantiation/Classification

- Every object (individual or attribute) should be classified to one of the instantiation levels:
  - **Token** (objects here denote atomic objects)
  - **S_Class** (objects here denote classes, i.e. sets of atomic objects)
  - **M1_Class** (objects here denote metaclasses, i.e. sets of sets of objects)
  - …
- Instantiation has set-membership semantics
  - a Token object can be classified to a S_Class object
  - a S_Class object can be classified to a M1_Class object
- Multiple Classification: an object can be classified to one or more classes

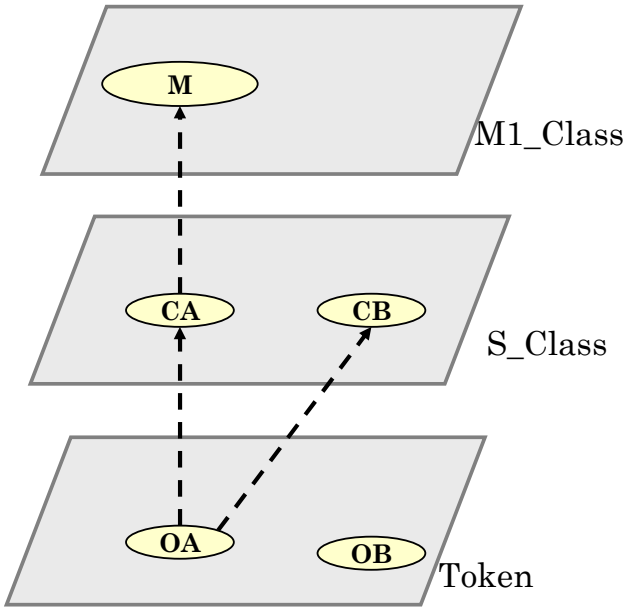
InstanceOf between Individuals

```
Tell Individual M in M1_Class
end
```

```
Tell Individual CA in S_Class, M
end
```

```
Tell Individual OA in Token, CA, CB
end
```

```
Tell Individual OB in Token
end
```



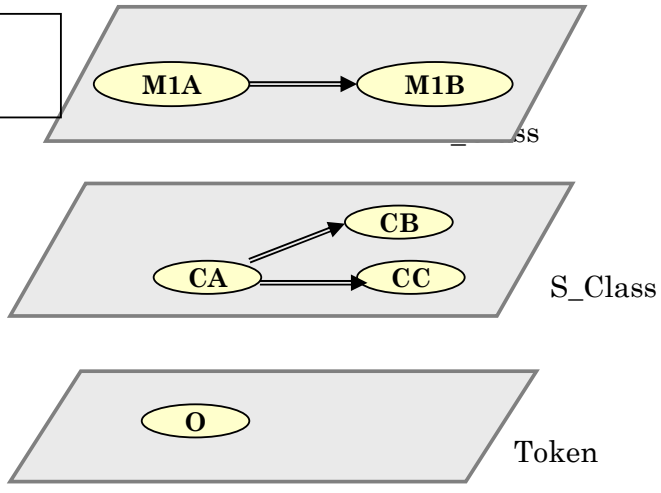
Gereralization/Specialization (IsA) relationships

- IsA links can relate objects of the same instantiation level (except Tokens) and same type
  - individuals with individuals
  - attributes with attributes
- The specialization has subset-semantics
- Multiple Specialization/Generalization
  - Integrity Constraint: The IsA lattice must me acyclic
- Inheritance
  - A subclasses inherits all the attributes of its superclasses
  - A subclass may refine the range of an inherited attribute by specializing it

IsA between Individuals

```
Tell Individual M1A   in M1_Class isA M1B
end
```

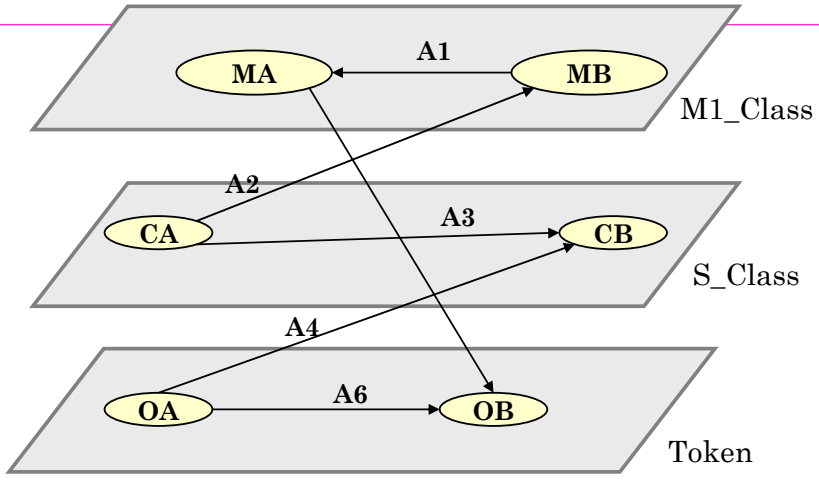
```
Tell Individual A   in S_Class isA B,C
end
```



Attribution

- Attributes are first class objects thus can be structured along classification, generalization and attribution.
- Attributes may relate
  - an Individual with an Individual
  - an Attribute with an Individual
- The instantiation level of an attribute should be less or equal to the minimum of the instantiation levels of its ends.

Attributes between Individuals



Implicit declaration of attributes

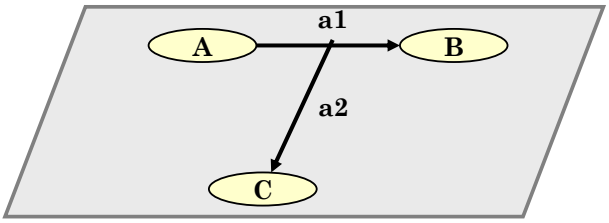
```
Tell Individual OA   in  Token
  with attribute
       A4: CB
       A6: OB
end
```

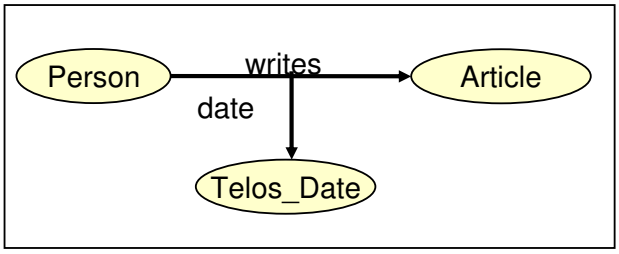
Explicit declaration of attributes

```
Tell Attribute A4
  from   : OA
  to     :CB
  in Token
end
```

Attributes from Attributes to Individuals

```
Tell Attribute a1
    from :A
    to    :B
 in Token
    with attribute
           a2: C
end
```





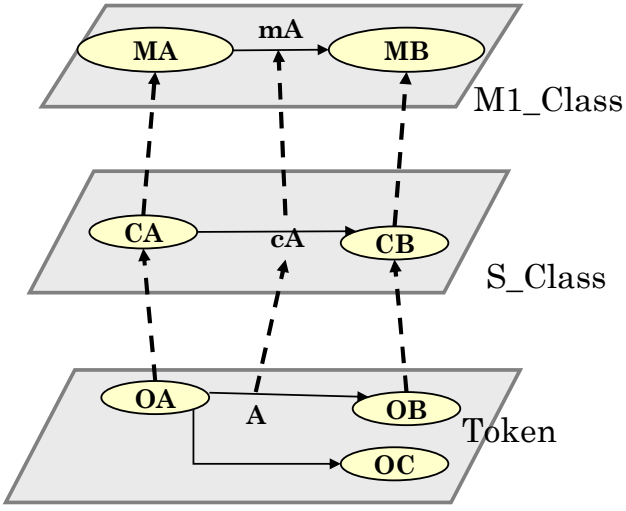
InstanceOf between Attributes

An attribute A1 might be InstanceOf- A2 provided that the origin and the destination of A1 are instances of the origin and the destination of A2

```
Tell Individual M  in M1_Class
   with attribute
           mA : MB
end
```

```
Tell Individual CA  in S_Class, MA
  with mA
      cA  : CB
end
```

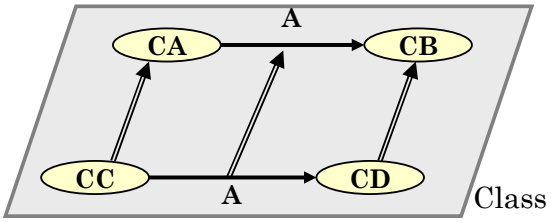
```
Tell Individual OA  in Token, CA
  with attribute
          : OC
  with cA
      A  : OB
 end
```



IsA between Attributes

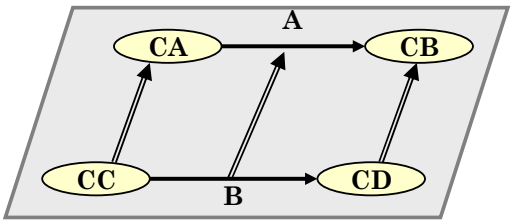
Attributes classes might be Isa-related provided that the origin and the destination object of the subclass are subclasses of the origin and the destination object of the relevant superclass

```
Tell Individual CC    in S_Class isA  CA
  with attribute
            A : CD
end
```

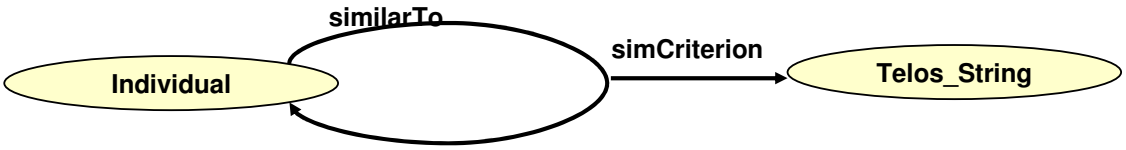



```
Tell Individual CC    in S_Class isA  CA
end

Tell AttributeClass B
     from :CC
     to     :CD
  in S_Class isA A from CA
end
```



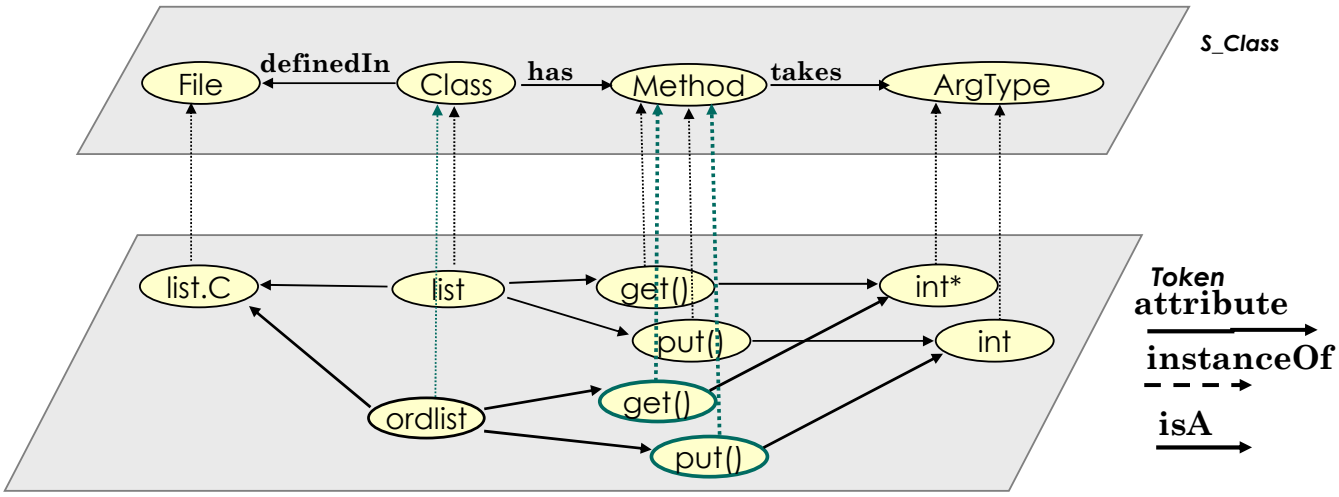
"Omega" classes



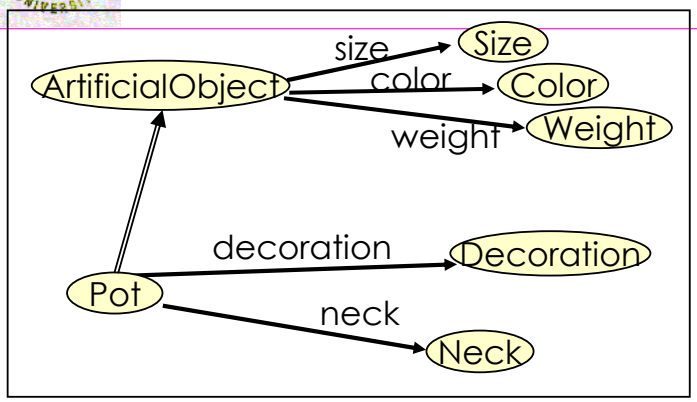

```
Tell AttributeClass similarTo
     from :Individual
     to     :Individual
  in S_Class
      with attribute
            simCriterion: Telos_String
end
```

Modeling Example: Static Analysis of Software

Multivalued attributes



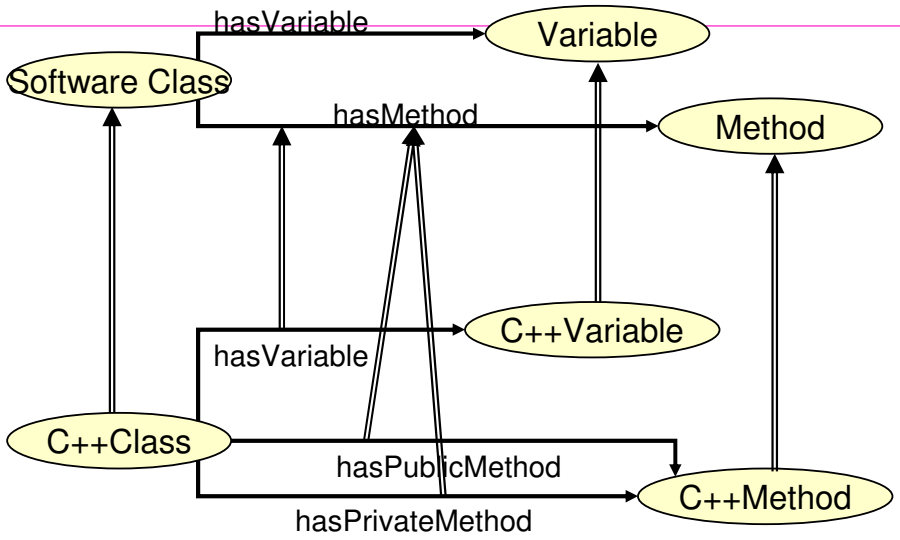
Modeling Example: Inheritance of Attributes




```
Tell Individual  Pot in S_Class
                    isA ArtificialObject
  with attribute
        decoration : Decoration
        neck       : Neck
end
```

```
Tell Individual  Pot11 in Token, Pot
with size
          : Size11
        color
          : Brown
        weight:
          : weight11
        decoration
          :  Minoan
        neck
          :  NeckWideOpen
end
```

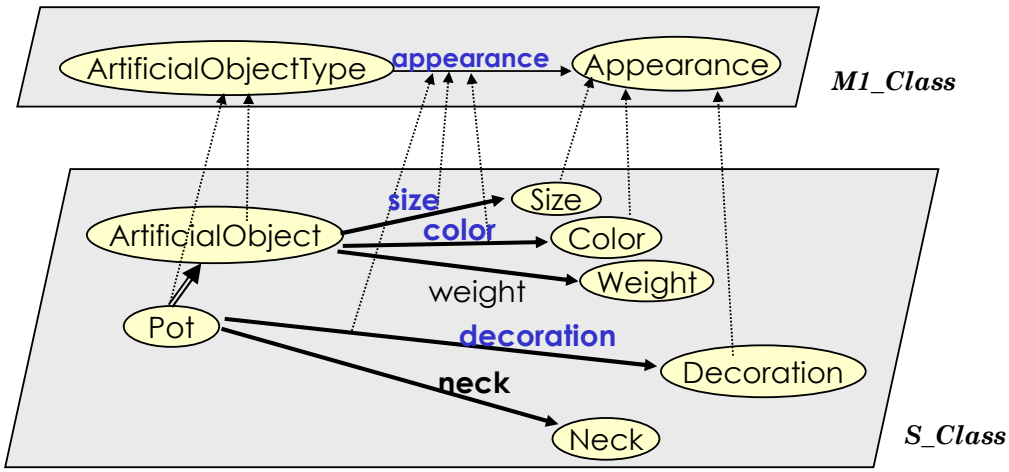
Specialization of Inherited Attributes




```
Tell Individual  C++Class
       in S_Class isA SoftwareClass
  with attribute
        hasVariable : C++Variable
 end
```

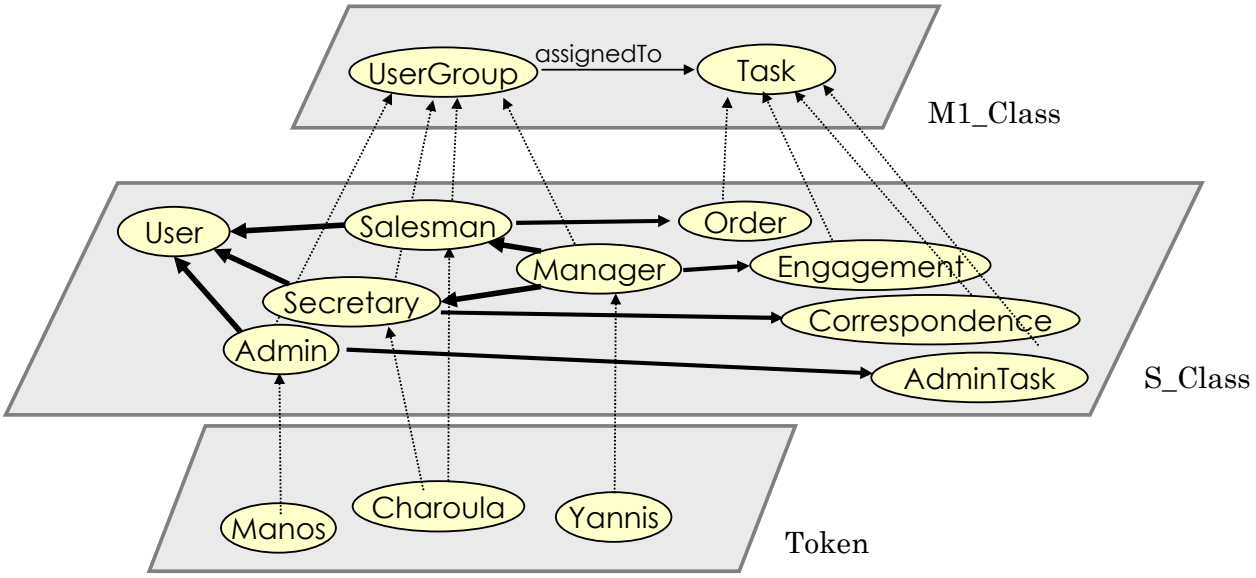
```
Tell Attribute  hasPublicMethod  in S_Class
    from    : C++Class
    to      : C++Method
in S_Class,  isA  hasMethod from SoftwareClass
end
```


Metacategories (attribute metaclasses)

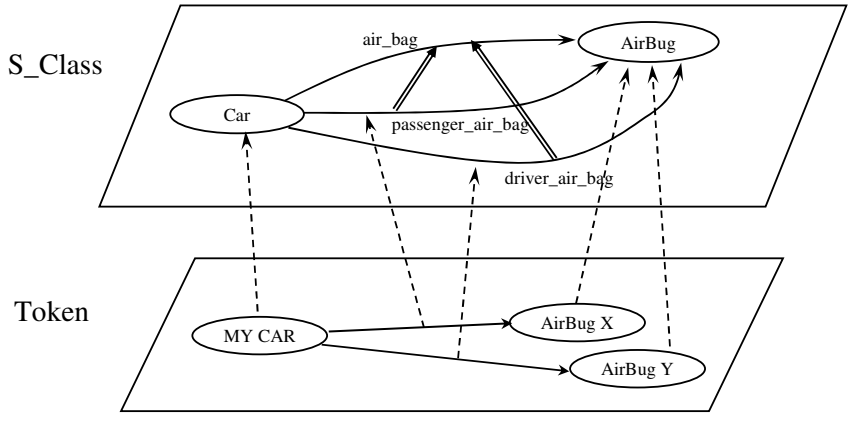


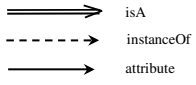

```
Tell Individual  Pot in S_Class, ArtificialObjectType  isA ArtificialObject
  with appearance
        decoration : Decoration
  with attribute
        neck       : Neck
end
```

Example



SIS-TELOS
Attribute Classification

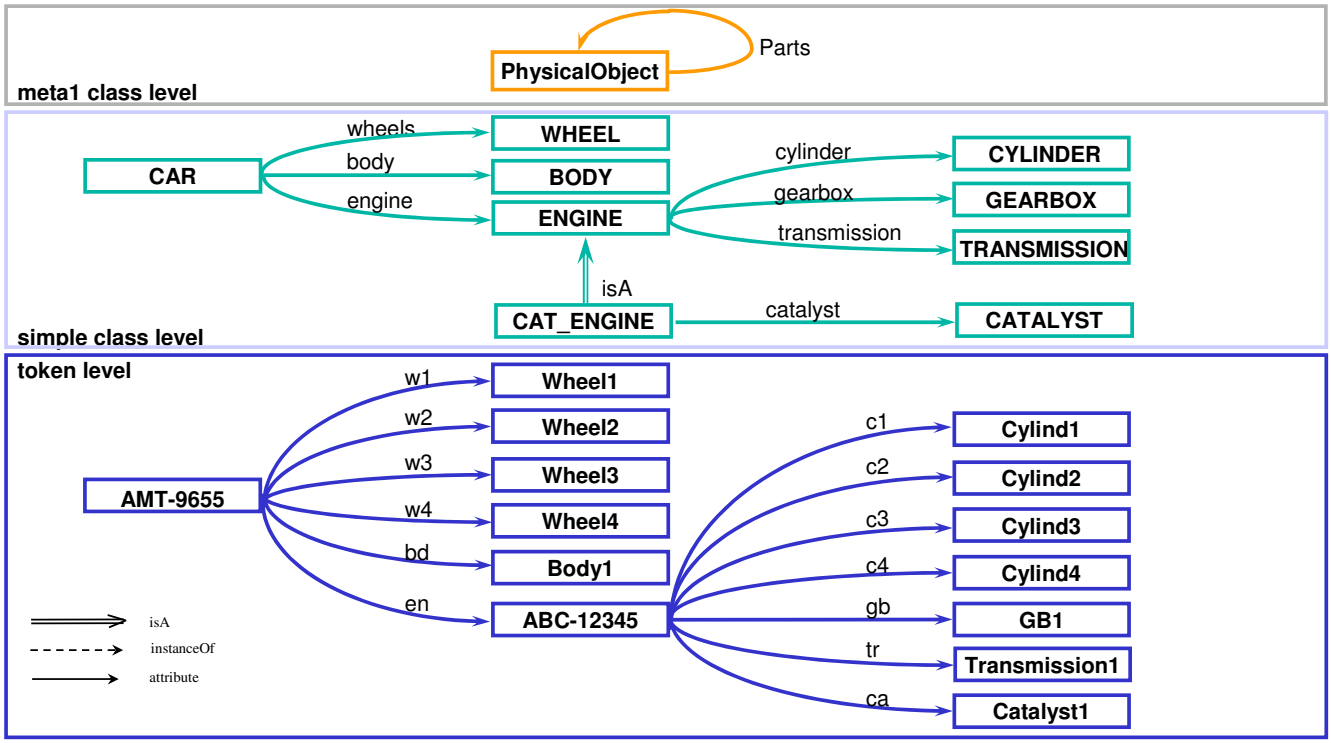


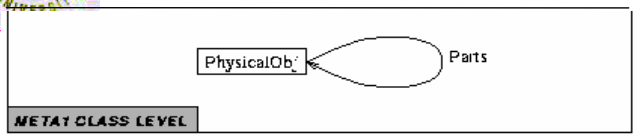


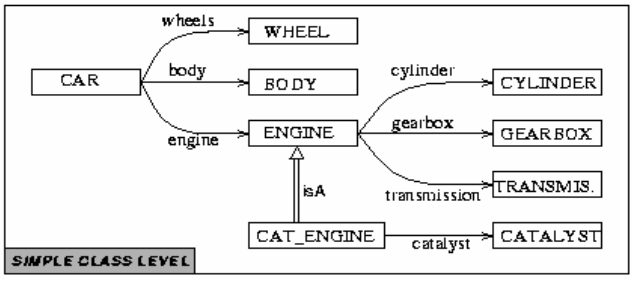
RETELL Attribute *air_bag*
from : *Car*
to : *AirBag*
in M1_Class
end

RETELL Attribute *passenger_air_bag*
from : *Car*
to : *AirBag*
in S_Class
isA *air_bag*
end

RETELL Attribute *driver_air_bag*
from : *Car*
to : *AirBag*
in S_Class
isA *air_bag*
end







RETELL Individual *CAR* in S_Class, *PhysicalObj*
with *Parts*
wheels : *WHEEL*;
body : *BODY*;
engine : *ENGINE*
end *CAR*

RETELL Individual *WHEEL* in S_Class, *PhysicalObj*
end *WHEEL*

RETELL Individual *BODY* in S_Class, *PhysicalObj*
end *BODY*

RETELL Individual *PhysicalObj* in M1_Class
with attribute
Parts : *PhysicalObj*
end *PhysicalObj*

RETELL Individual *ENGINE* in S_Class, *PhysicalObj*
with *Parts*
cylinder : *CYLINDER*;
gearbox : *GEARBOX*;
transmission : *TRANSMISSION*
end *ENGINE*

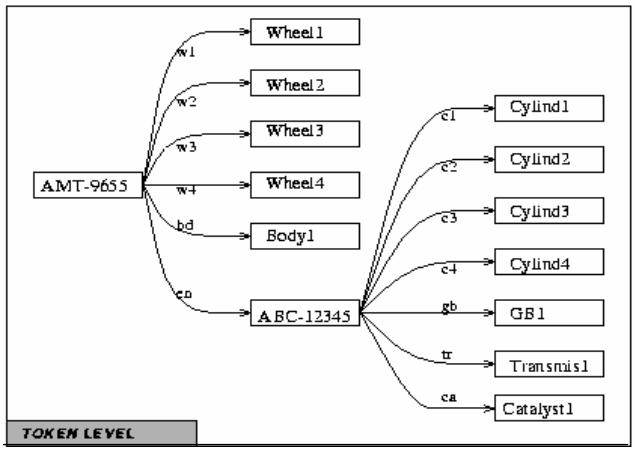
RETELL Individual *CYLINDER* in S_Class, *PhysicalObj*
end *CYLINDER*

RETELL Individual *GEARBOX* in S_Class, *PhysicalObj*
end *GEARBOX*

RETELL Individual *TRANSMISSION* in S_Class, *PhysicalObj*
end *TRANSMISSION*

RETELL Individual *CAT_ENGINE* in S_Class, *PhysicalObj*
isA *ENGINE*
with *Parts*
catalyst : *CATALYST*
end *CAT_ENGINE*

RETELL Individual *CATALYST* in S_Class, *PhysicalObj*
end *CATALYST*



RETELL Individual (*AMT-9655*) in Token, *CAR*
with *wheels*
w1 : *Wheel1*;
w2 : *Wheel2*;
w3 : Wheel3;
w4 : *Wheel4*
with *body*
bd : *Body1*
with *engine*
en : (*ABC-12345*)
end (*AMT-9655*)

RETELL Individual *Wheel1* in Token, *WHEEL*
end *Wheel1*

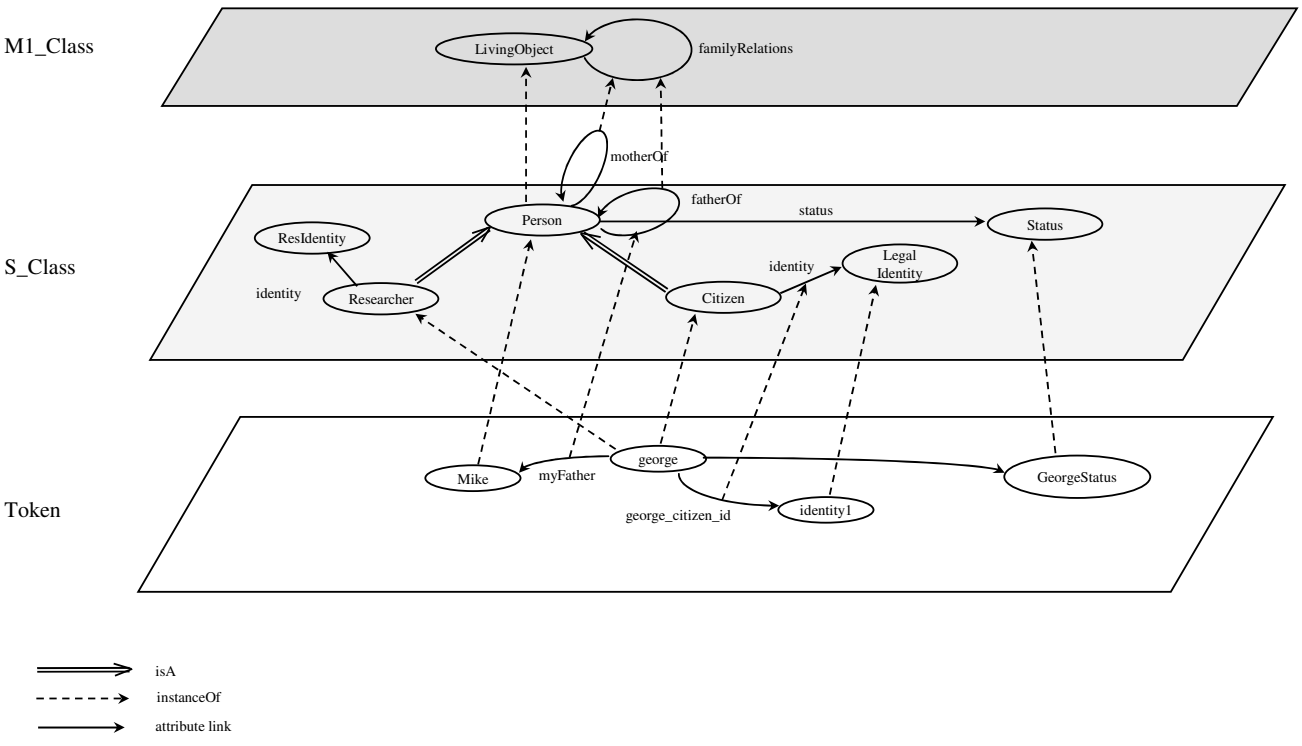
RETELL Individual *Wheel2* in Token, *WHEEL*
end *Wheel2*

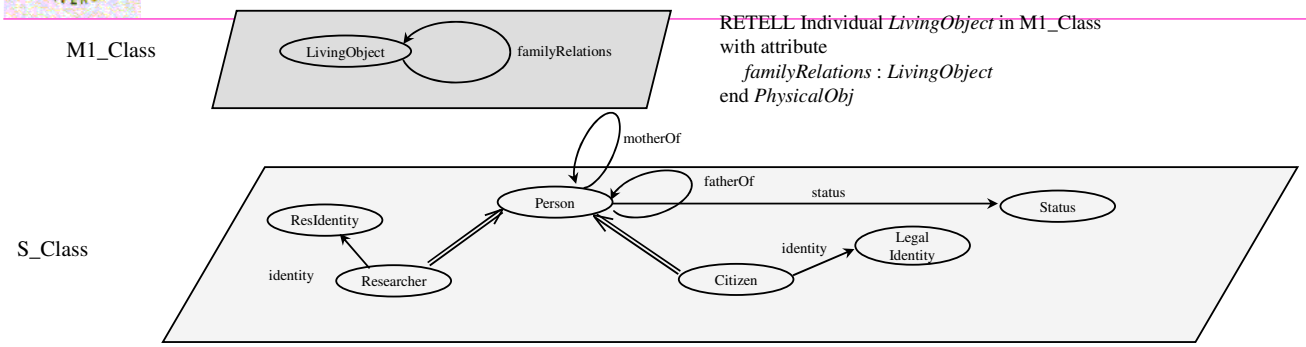
RETELL Individual *Wheel3* in Token, *WHEEL*
end *Wheel3*

RETELL Individual *Wheel4* in Token, *WHEEL*
end *Wheel4*

RETELL Individual *Body1* in Token, *BODY*
end *Body1*

RETELL Individual (*ABC-12345*) in Token,*CAT_ENGINE*
with *cylinder*
c1 : *Cylind1*;
c2 : *Cylind2*;
c3 : *Cylind3*;
c4 : *Cylind4*
with *gearbox*
gb : *GB1*
with *transmission*
tr : *Transmis1*
with *catalyst*
ca : *Catalyst1*
end (*ABC-12345*)





RETELL Individual *LivingObject* in M1_Class
with attribute
familyRelations : *LivingObject*
end *PhysicalObj*

RETELL Individual *Person* in S_Class, *LivingObject*
with *familyRelations*
motherOf : *Person* ;
fatherOf : *Person*
with attribute
status : *Status*
end *Person*

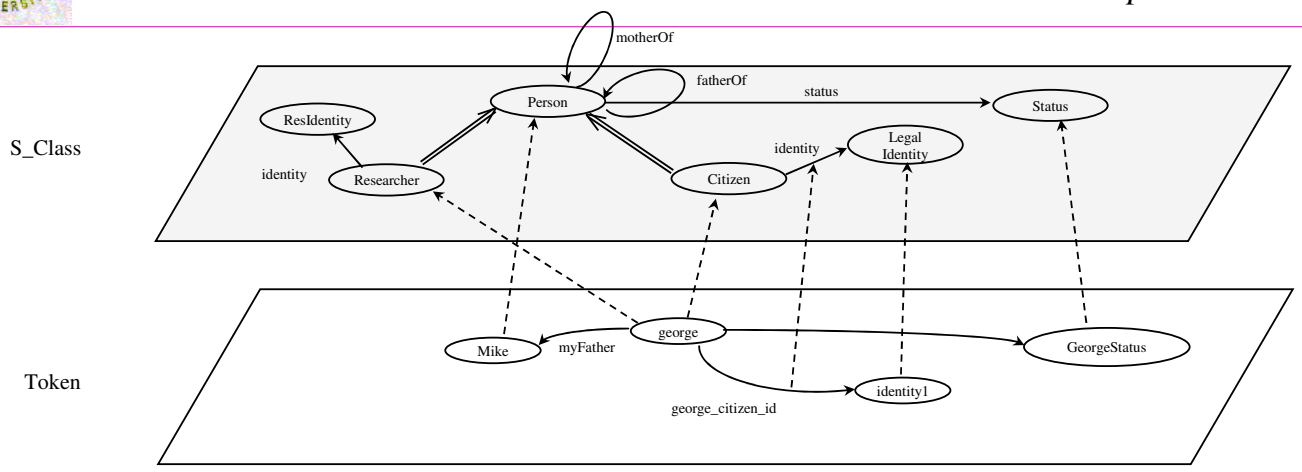
RETELL Individual *Status* in S_Class
end *Status*

RETELL Individual *Citizen* in S_Class
isA *Person*
with attribute
identity : *LegalIdentity*
end *Citizen*

RETELL Individual *Researcher* in S_Class
isA *Person*
with attribute
identity : *ResIdentity*
end *Researcher*

RETELL Individual *LegalIdentity* in S_Class
end *LegalIdentity*

RETELL Individual *ResIdentity* in S_Class
end *ResIdentity*



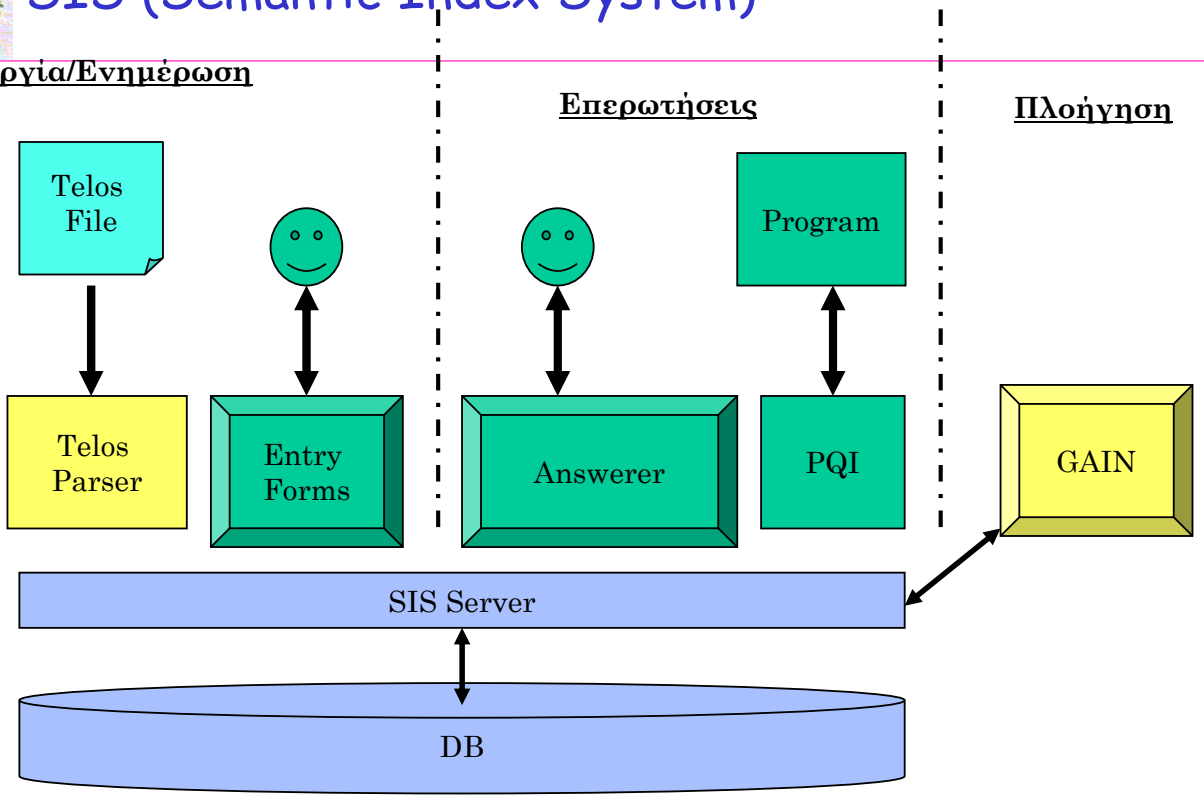
RETELL Individual *George* in Token, *Researcher*, *Citizen*
with *fatherOf*
myFather : *Mike*;
with *status*
: *GeorgeStatus*;
with *identity* from *Citizen*
*george_citizen_*id : *identity1*
end *George*

RETELL Individual *GeorgeStatus* in *Token*, *Status*
end *GeorgeStatus*

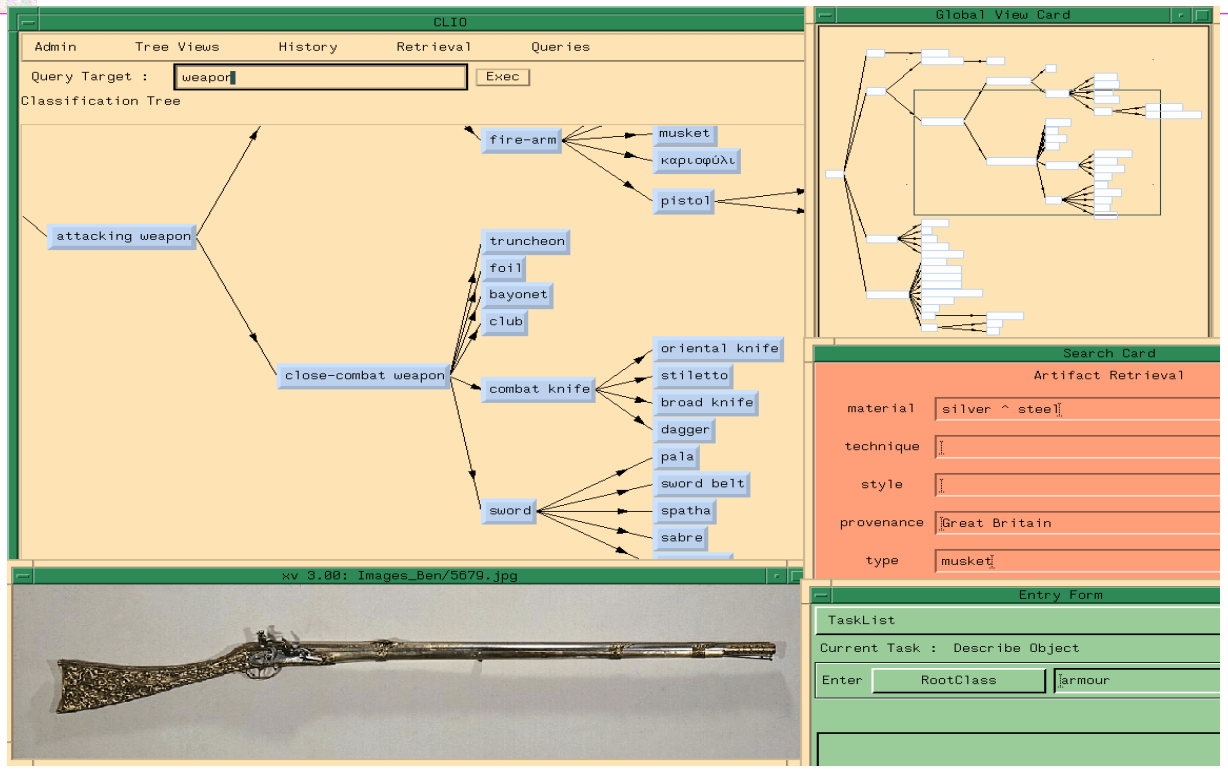
RETELL Individual *identity1* in Token, *LegalIdentity*
end *identity1*

RETELL Individual *Mike* in Token, *Person*
end *Mike*

SIS (Semantic Index System)



GAIN



Applications of SIS

- Current Applications :
  - CLIO Cultural Information System
  - National Monument Record of Greece
  - Paul Getty VCS Prototype
  - SIB Static Analyzer and Class Management System
  - TMS Thesaurus Management System
- ….

References

- **Telos: Representing Knowledge About Information Systems,**John Mylopoulos, Alex Borgida, Matthias Jarke, Manolis Koubarakis, Information Systems,8(4),1990
- SIS-Telos: http://www.ics.forth.gr/isl/r-d-activities/sis.html