L3 Mention Informatique
Parcours Informatique et MIAGE

Génie Logiciel Avancé - Advanced Software Engineering

Advanced Elements of the UML

Burkhart Wolff
wolff@lri.fr

Main UML diagram type:

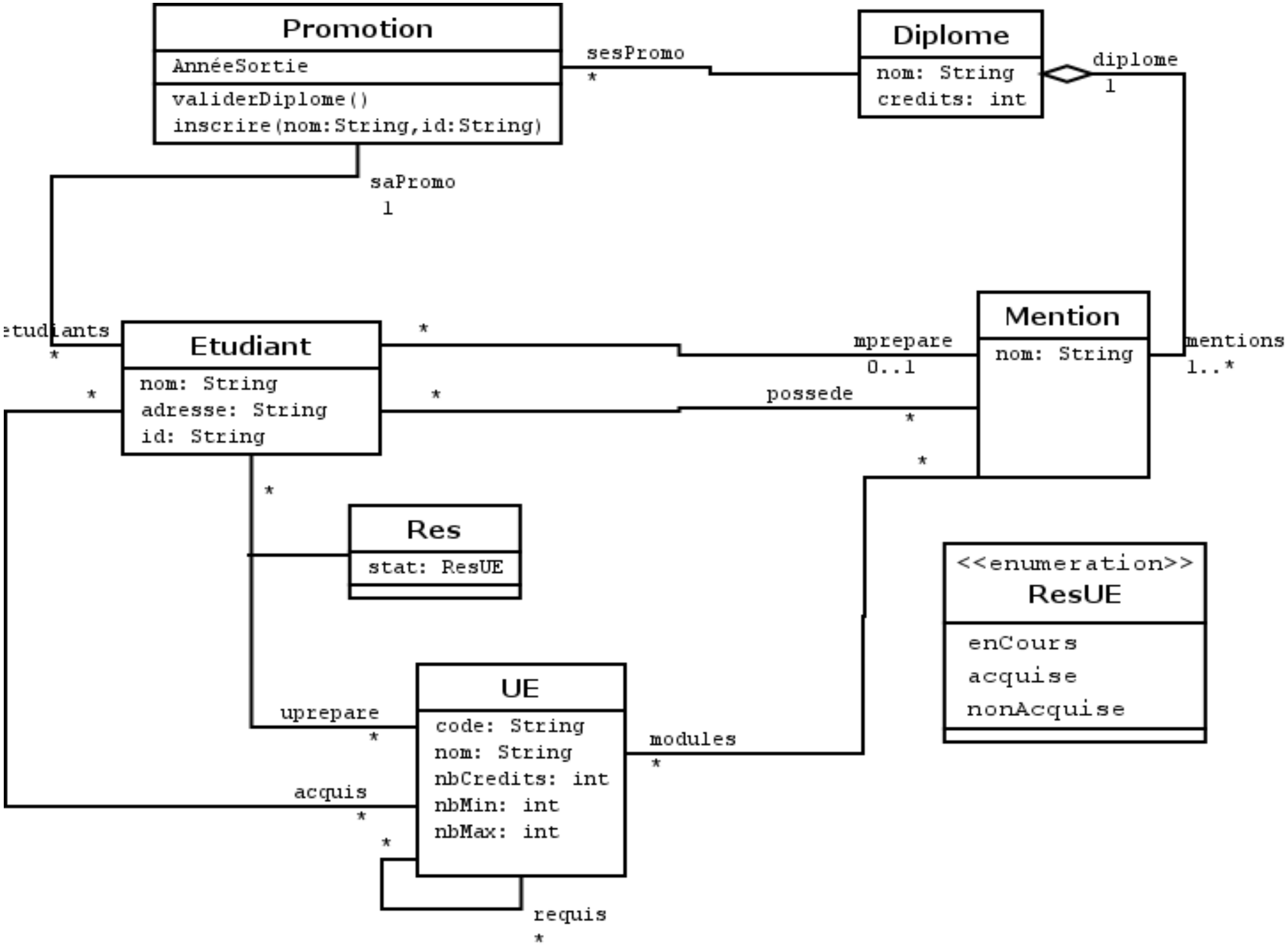
- **Class Diagrams** („Diagrammes de classes") :

  the static **structure** of the DATA of the system

  - the classes of interest to be represented in the system
  - the relations between classes
  - the attributes and the methods
  - the types, required/defined interfaces …

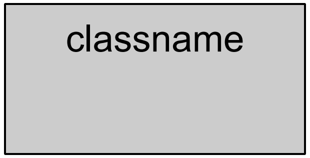
  can be used for top-level views as specific interfaces for local code ...

Example: A Class Diagram

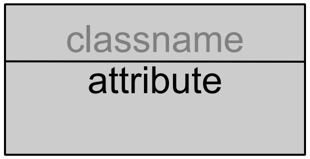


A propos Class Diagrams (1)

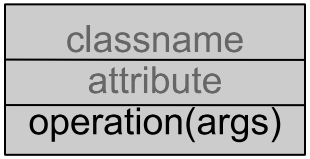
- Model-Elements
  - Class



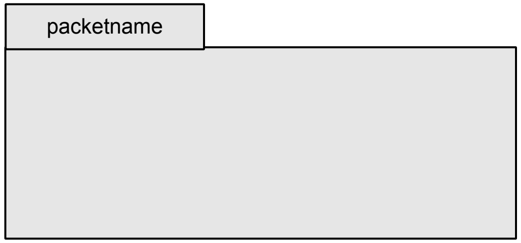
  - Attributes



  - Operations (methods)

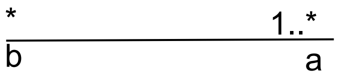


  - Packages (grouping mechanism for parts of a class model)

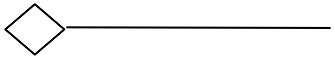


A propos Class Diagrams (2)

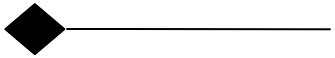
- Model-Elements
  - Association (with optional roles cardinalities)



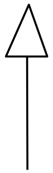
  - Aggregation (« has a » relationship with weak linkage)



  - Composition (« has a » relationship with strong linkage)

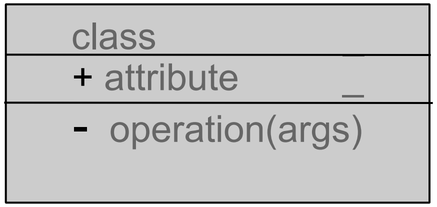


  - Specialisation (modelling of a „is-a“ relationship between classes)

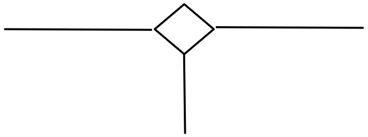


A propos Class Diagrams (3)

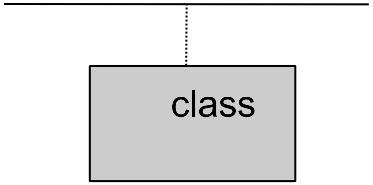
- ## Model-Elements
  - Visibilities
    (optional public
    and private, see more later)



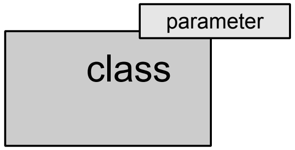
  - N-ary associations



  - *Association Class*
    (more complex constraints om relations)

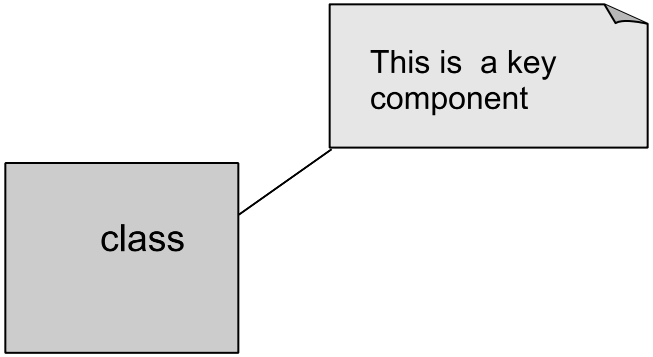


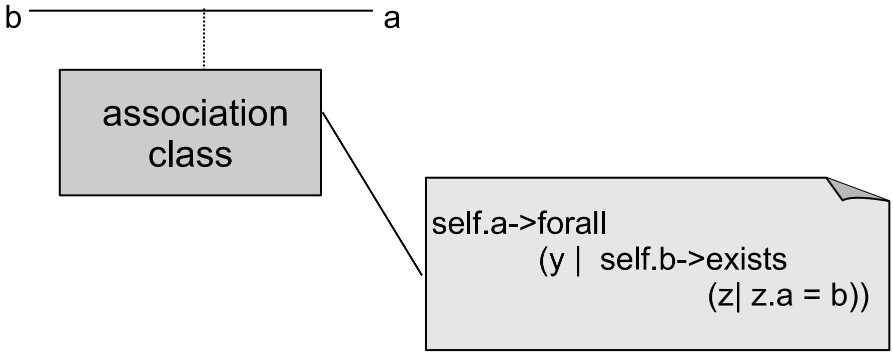
  - templates with parameter
    (usually classes like "Set(A)")



A propos Class Diagrams (4)

- Model-Elements
  - Annotations
  - ... typically on classes and individual operations
  - ... can be informal text as well as a mathematical notation like OCL (we will use our own notation)





```
self.a->forall
        (y |  self.b->exists
                   (z| z.a = b))
```

A propos Class Diagrams (1)

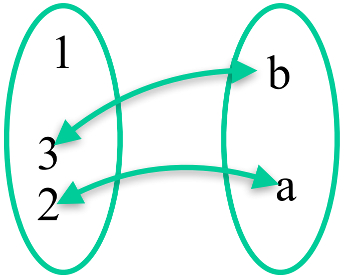
- Semantics: Classes are:
  - types of objects
  - tuples of „attributes"
  - associations represent (math.) relations of objects
  - aggregations represent (Collections of) of references to other objects
  - objects may be linked via references to each other into a state called „object graph"
  - cardinalities, etc. are INVARIANTS in this state, so constraints on the object graph

Recall: What is a Relation in Mathematics

- Formally, a "relation" R is a set of pairs built over two sets A and B, so a subset of the Cartesian Product of A and B :

$$R \subseteq A \times B$$

- Example: A={1,2,3}, B={a,b}:



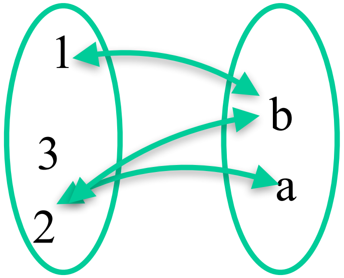
r = {(2,a),(3,b)}

Recall: What is a Relation in Mathematics

- Formally, a "relation" R is a set of pairs built over two sets A and B, so a subset of the Cartesian Product of A and B :

$$R \subseteq A \times B$$

- Example: A={1,2,3}, B={a,b}:



r' = {(2,a),(2,b),(1,b)}

A propos Class Diagrams (2)

- Attributes
  - can have simple type (Integer, Boolean, String, Real) or primitive type (see Date example) only !
  - in diagrams, attributes may NOT have collection type (use therefore **associations)**
  - In a requirement analysis model, everything is **public** by default

More Specific Details in UML 2

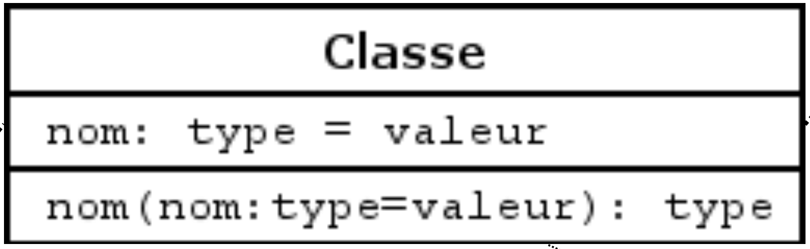
***Visibilities**:*

+: public

- : private

#: protected

/ : derived



Classe

nom: type = valeur

nom(nom:type=valeur): type

***Modifiers**:*

static

abstract

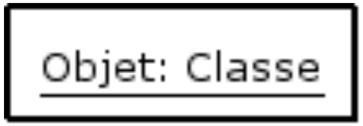
***Parameter modes**:*

in (par défaut)

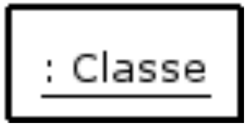
out

in out

***Instances**:*

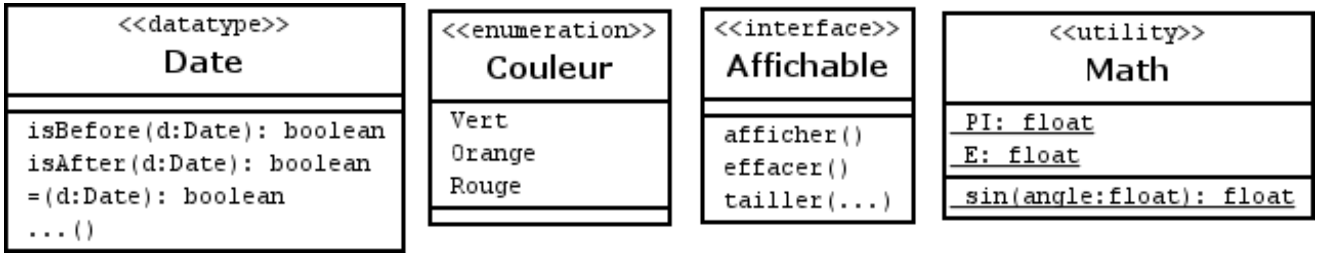


Objet: Classe



: Classe

Stéréotypes:



<<datatype>>
Date

isBefore(d:Date): boolean
isAfter(d:Date): boolean
=(d:Date): boolean
...()

<<enumeration>>
Couleur

Vert
Orange
Rouge

<<interface>>
Affichable

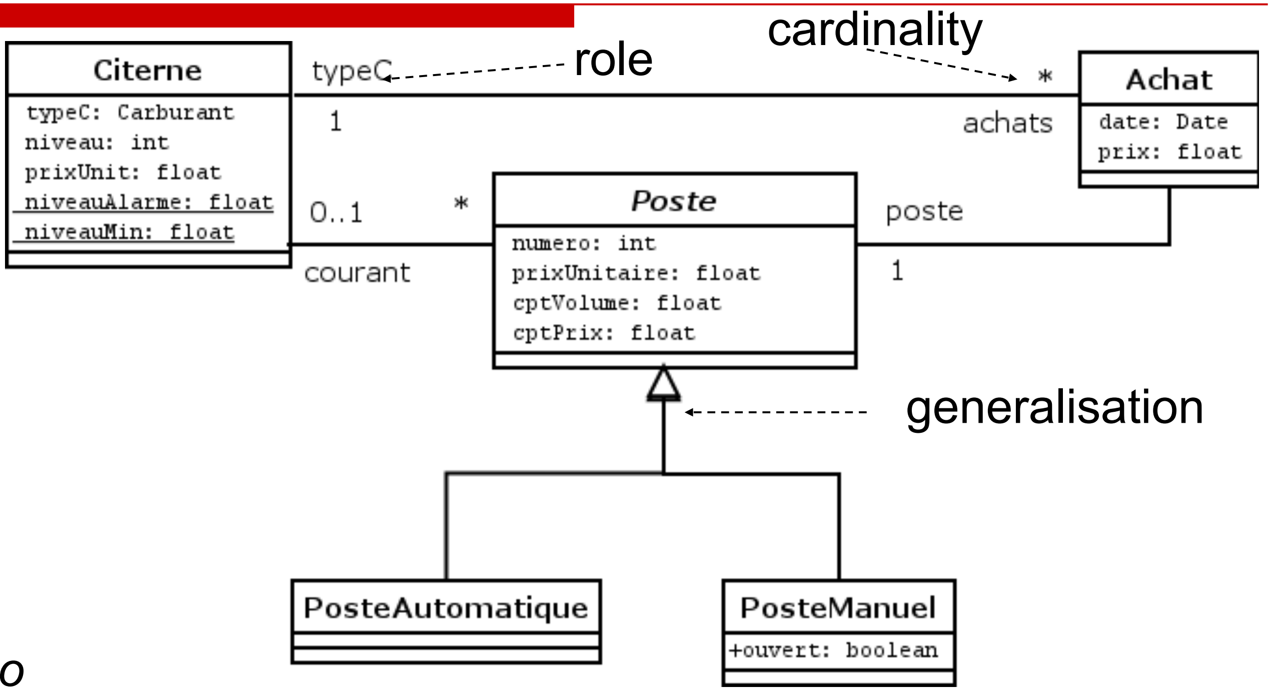
afficher()
effacer()
tailler(...)

<<utility>>
Math

PI: float
E: float

sin(angle:float): float

More Specific Details in UML 2



The roles were used to navigate across associations

for `a:Achat`, the expression `a.poste` denotes an instance of `Poste`.

for `c:Citerne`, the expression `c.achats` denotes an instance of `Achat`

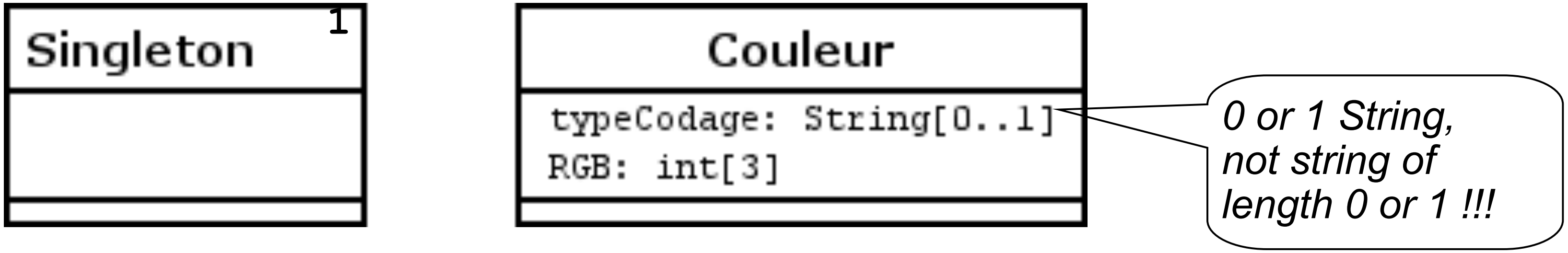
for `p:Poste`, the expression `p.courant` corresponds to a collection of 0 or 1 instances of `Citerne`.

More Specific Details in UML 2

Cardinalities in associations can be:

- 1, 2, or an integral number (no expression !)
- * (for « arbitrary », ...)
- an interval like 1..*, 0..1, 1..3, (**not** like 1..N)

Multiplicities on attributs and classes can be:



Singleton 1

Couleur

typeCodage: String[0..1]

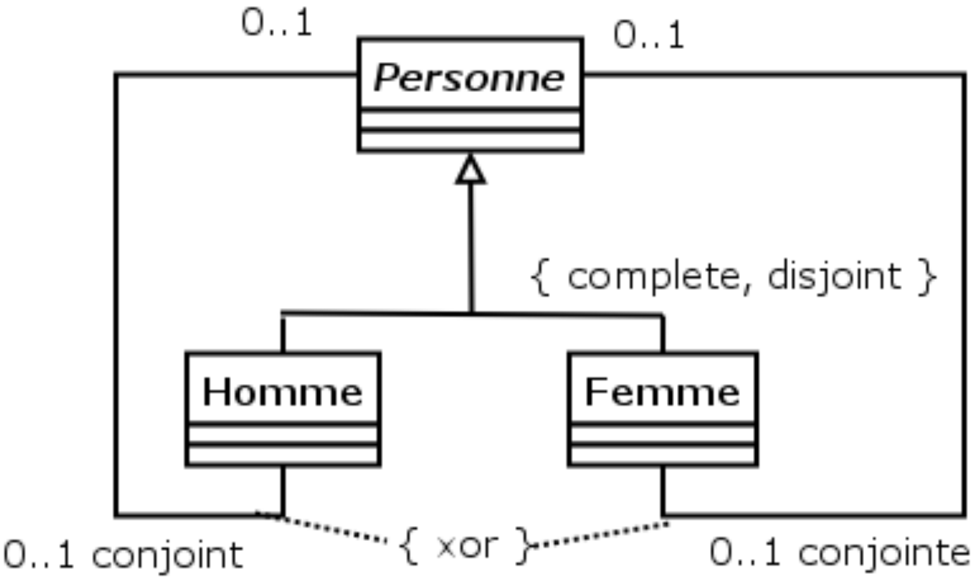
RGB: int[3]

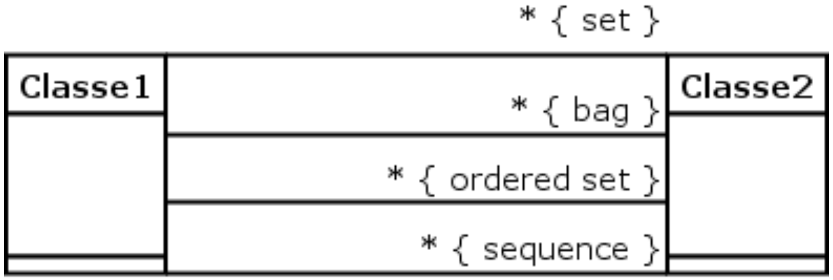
0 or 1 String, not string of length 0 or 1 !!!

More Specific Details in UML 2

Contraints on associations

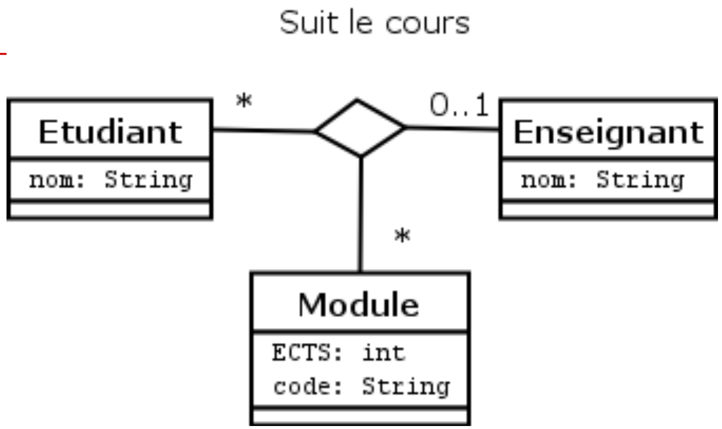
- For generalisation:
  - `complete, incomplete`
  - `disjoint, overlapping`
- Between associations
  - `xor`
- Collection Types may now also be specified !!!
  - `no duplicates, unordered`
  - `duplicates, unordered`
  - `no duplicates, ordered`
  - `duplicates, positioned`



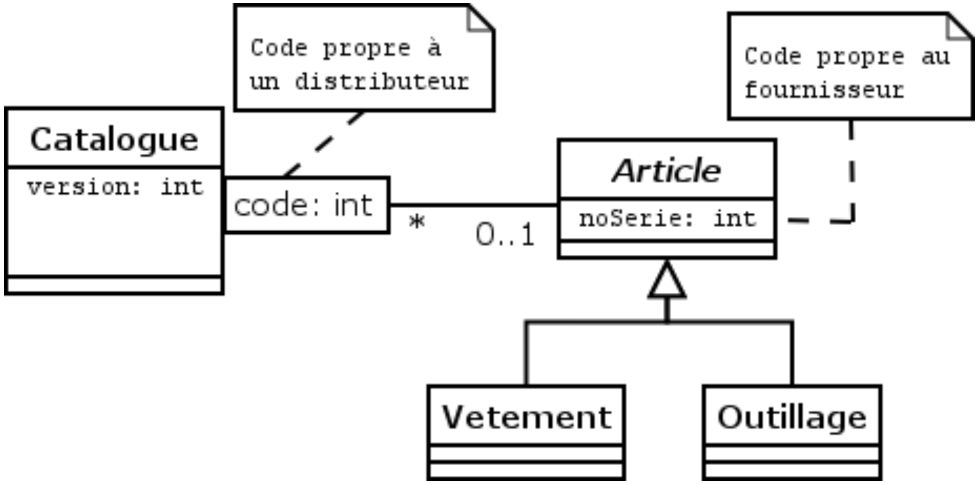


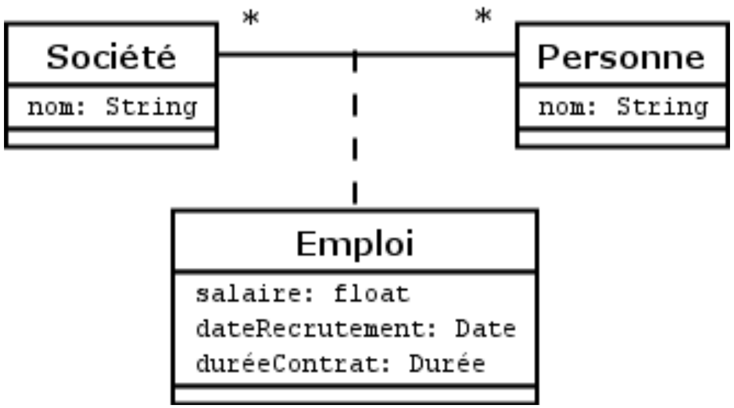
More Specific Details in UML 2

N-ary Associations



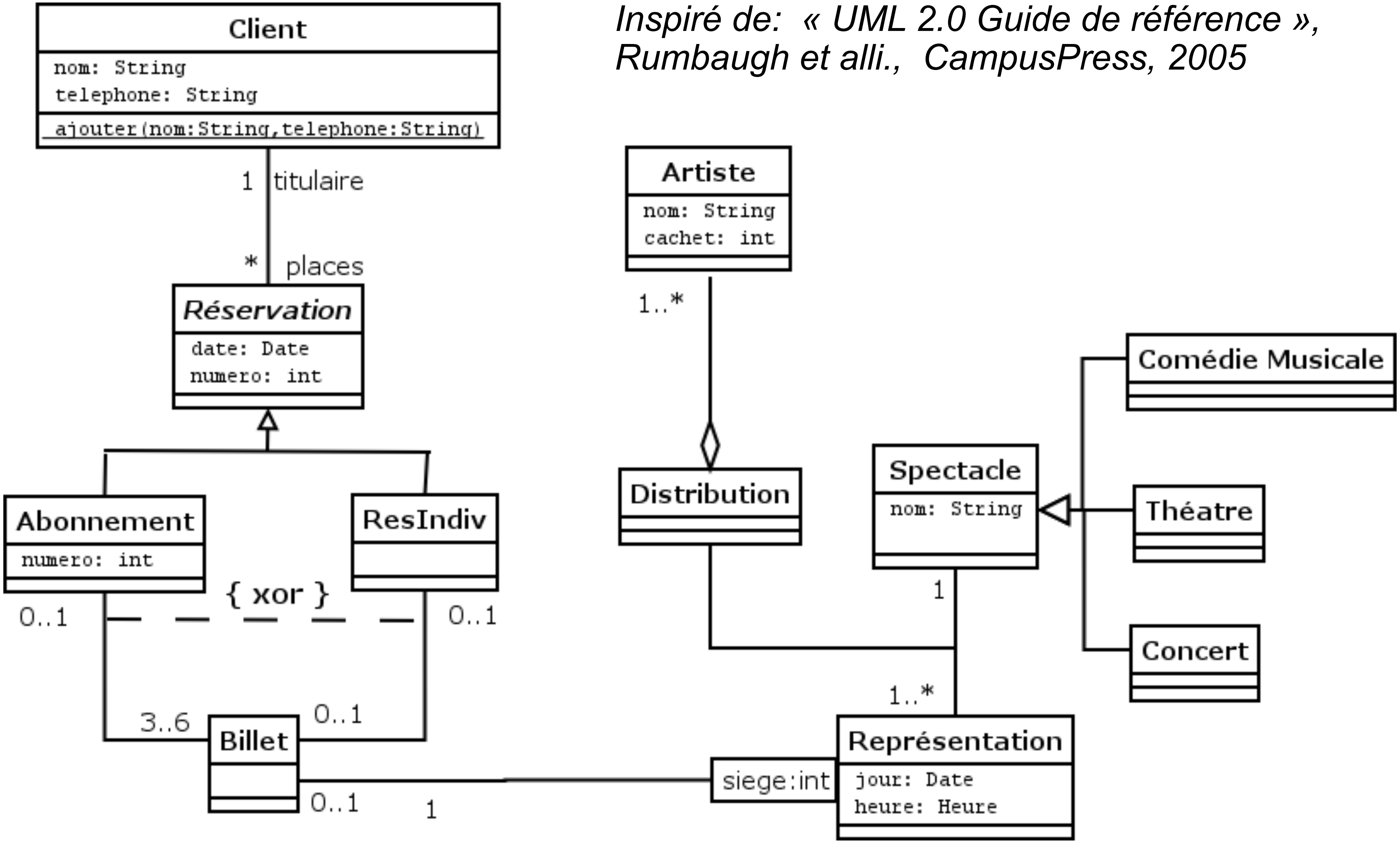
Association with attributes





Putting all together …

Inspiré de: « UML 2.0 Guide de référence », Rumbaugh et alli., CampusPress, 2005



Principal UML diagram types (5)

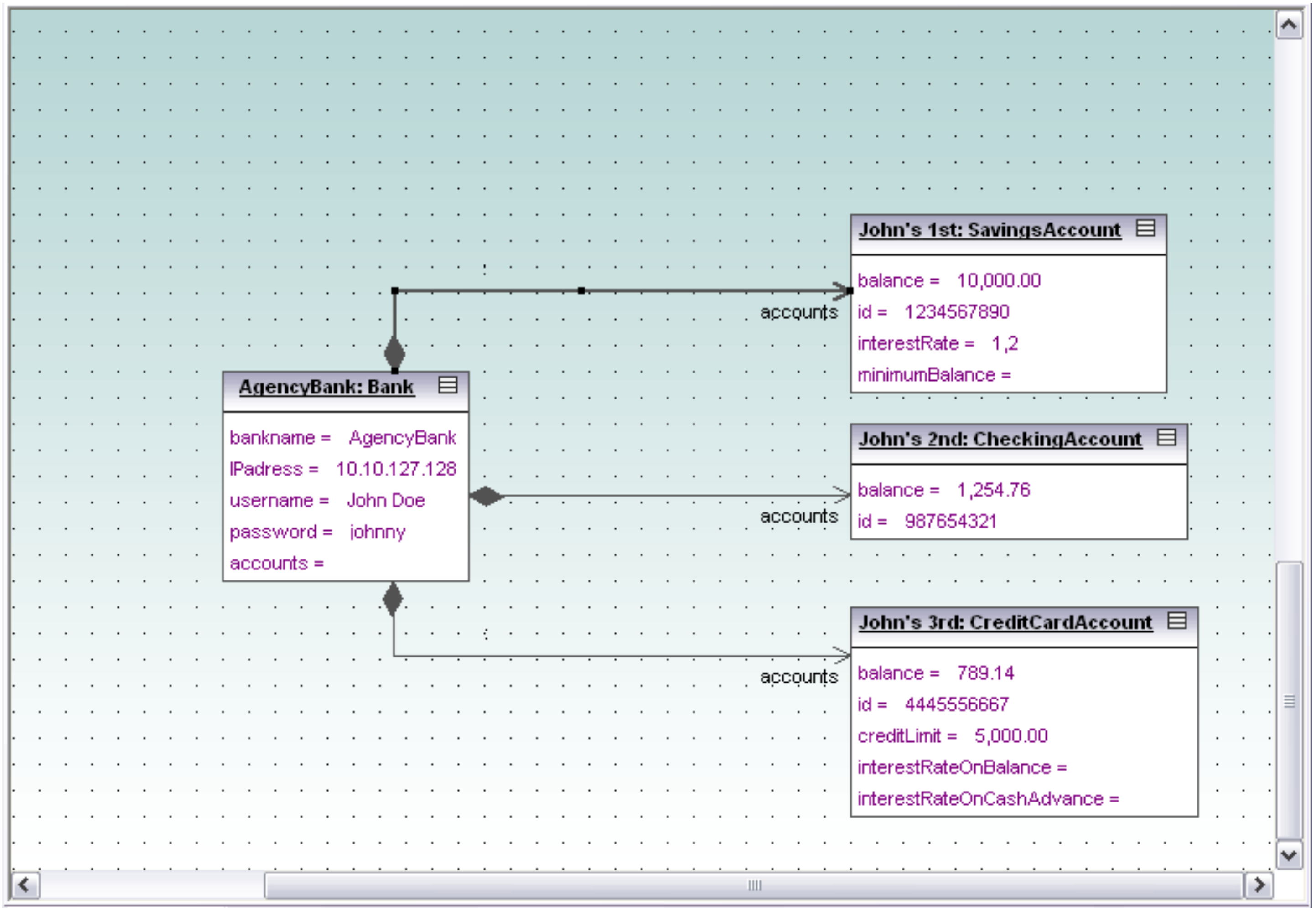
- **Object Graphs or “Object Model”** („Diagrammes d'objects“) :

  denote a concrete system state,

- typically used in connection with a Class Diagram
  - attributes have concrete values
  - associations were replaced by directed arcs representing the links

  can be used for debugging purposes ...
  (semantics: fully clear).

Example Object Diagram



AgencyBank: Bank

bankname = AgencyBank
IPadress = 10.10.127.128
username = John Doe
password = johnny
accounts =

John's 1st: SavingsAccount

balance = 10,000.00
id = 1234567890
interestRate = 1,2
minimumBalance =

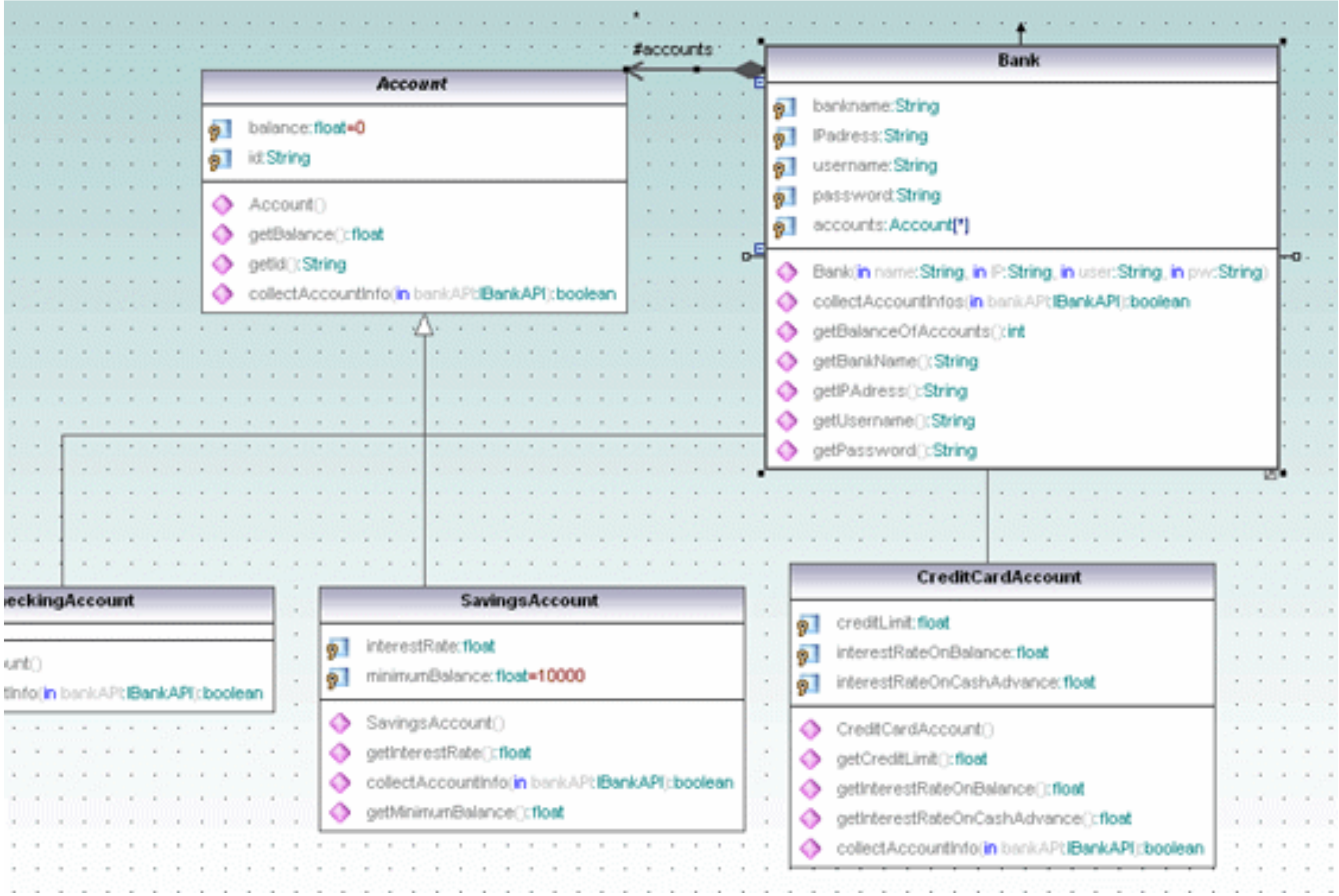
John's 2nd: CheckingAccount

balance = 1,254.76
id = 987654321

John's 3rd: CreditCardAccount

balance = 789.14
id = 4445556667
creditLimit = 5,000.00
interestRateOnBalance =
interestRateOnCashAdvance =

Example Object Diagram


#accounts
*
Account
balance:float=0
id:String
Account()
getBalance():float
getId():String
collectAccountInfo(in bankAPI:IBankAPI):boolean
Bank
bankname:String
IPadress:String
username:String
password:String
accounts:Account[*]
Bank(in name:String, in IP:String, in user:String, in pw:String)
collectAccountInfos(in bankAPI:IBankAPI):boolean
getBalanceOfAccounts():int
getBankName():String
getIPAdress():String
getUsername():String
getPassword():String
eckingAccount
unt()
tInfo(in bankAPI:IBankAPI):boolean
SavingsAccount
interestRate:float
minimumBalance:float=10000
SavingsAccount()
getInterestRate():float
collectAccountInfo(in bankAPI:IBankAPI):boolean
getMinimumBalance():float
CreditCardAccount
creditLimit:float
interestRateOnBalance:float
interestRateOnCashAdvance:float
CreditCardAccount()
getCreditLimit():float
getInterestRateOnBalance():float
getInterestRateOnCashAdvance():float
collectAccountInfo(in bankAPI:IBankAPI):boolean

Summary: Class and Object Diagrams

- Class Diagrams represent an abstract data-model of a system. The UML allows to sufficient precision such that they can be compiled to, for example, Java Interfaces.
- Class Diagrams allow to SPECIFY certain aspects of a data-model, for example the relation of objects in a state
- Object Models denote a concrete State of a Class Model
- Multiplicities and Cardinalities express INVARIANTS on (valid) Object Models to a given Class Model – with this respect, serves as Specification of States.