L3 Mention Informatique
Parcours Informatique et MIAGE

Génie Logiciel Avancé - Advanced Software Engineering

A Brief Revision of UML

Burkhart Wolff
wolff@lri.fr

Plan of the Chapter

- The UML notation is used as document - core in SE processes (such as the V model)
- Syntax and semantics of class model elements and their visualization in diagrams
  - Class Invariants
  - Constraints
  - Operations
  - Pre- and Post-Conditions
- Syntax and semantics of state machines
  Specify system components for test and verification

The UML ...

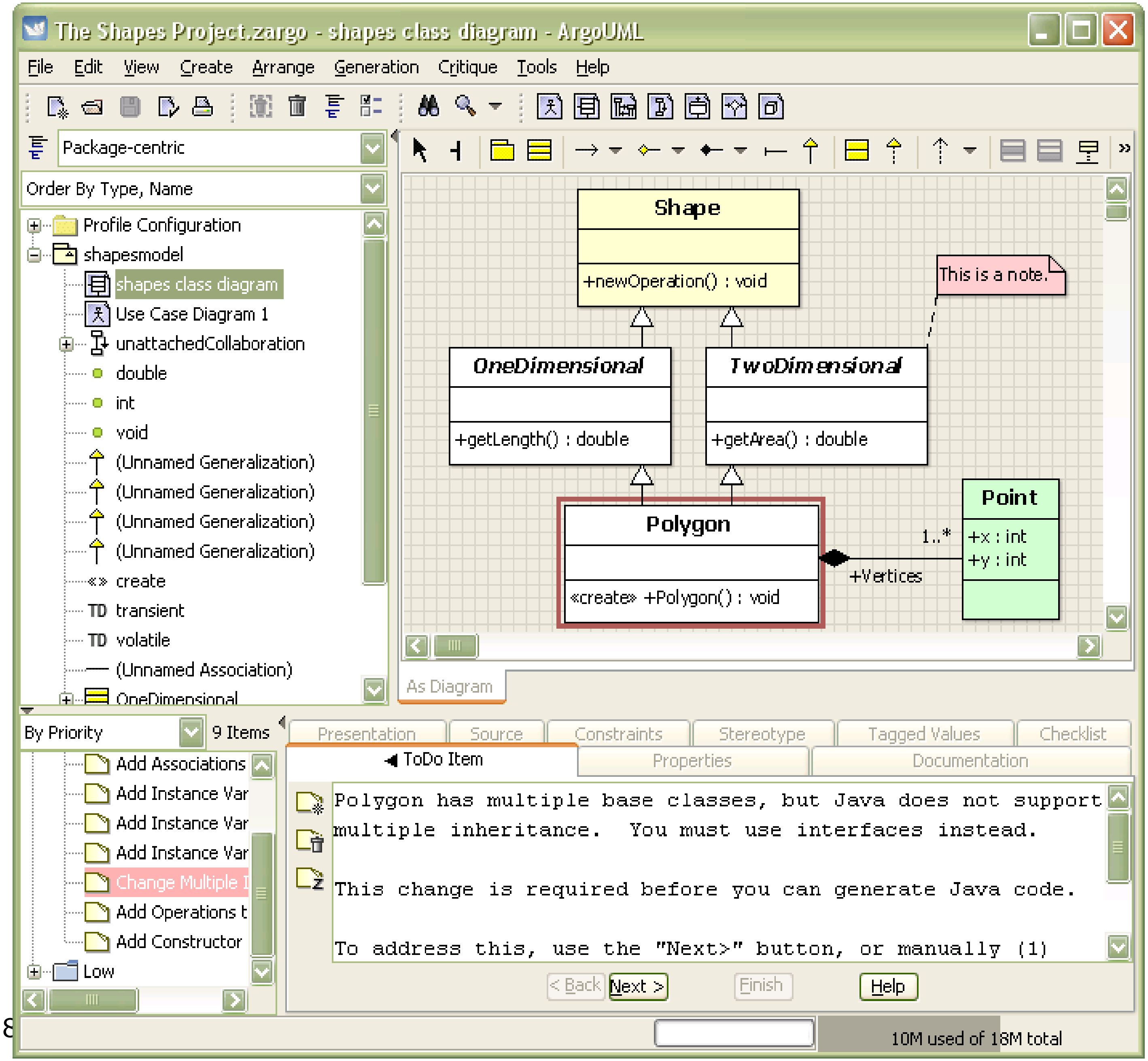
- ... is the Unified Modeling Language
- ... is a normed data-structure, a „technical format" of model-elements (that may contain other model-elements) with consistent naming for
  - various system descriptions
  - various code formats
- ... has various external representations
  - as XMI exchange format (XML-formal)
  - as ECore Model
  - as UML diagrams

The UML offers the advantage ...

- ... of being a basis for

  Integrated Development Environments

  (IDE's like ArgoUML, Poseidon,
  Eclipse + Plugins like Papyrus,
  Rational Rose, Prodigé, ...)



The Shapes Project.zargo - shapes class diagram - ArgoUML

```
Polygon has multiple base classes, but Java does not support
multiple inheritance.  You must use interfaces instead.

This change is required before you can generate Java code.

To address this, use the "Next>" button, or manually (1)
```

The UML offers the advantage ...

- ... to offer „object-oriented" specifications
- ... to offer a formal, mathematical semantics (well, at least to some parts of the UML)
- ... to be fairly widely used in industry, even if not always supported entirely or used in similar variants like SysML
- ... is the basis for a whole software-engineering paradigm called Model-Driven Engineering (MDE).

The UML 2.0 Diagrams (for corresp. models)

- UML, Version 1.1 : 9 types of diagrams

- UML, Version 2.0 adds

  4 more types of diagrams

  - structure composition
  - communication
  - packaging
  - temporal constraints (timing)

Bibliographie et Weberies

- UML 2.0, Martin Fowler, Campus Press, 2004
- Developing Applications with Java and UML, Paul R. Reed Jr., Addison Wesley, 2002
- UML 2 et les Design Patterns, G. Larman, Campus Press, 2005
- UML 2 en action, P. Roques, F. Vallée, Eyrolles 2004

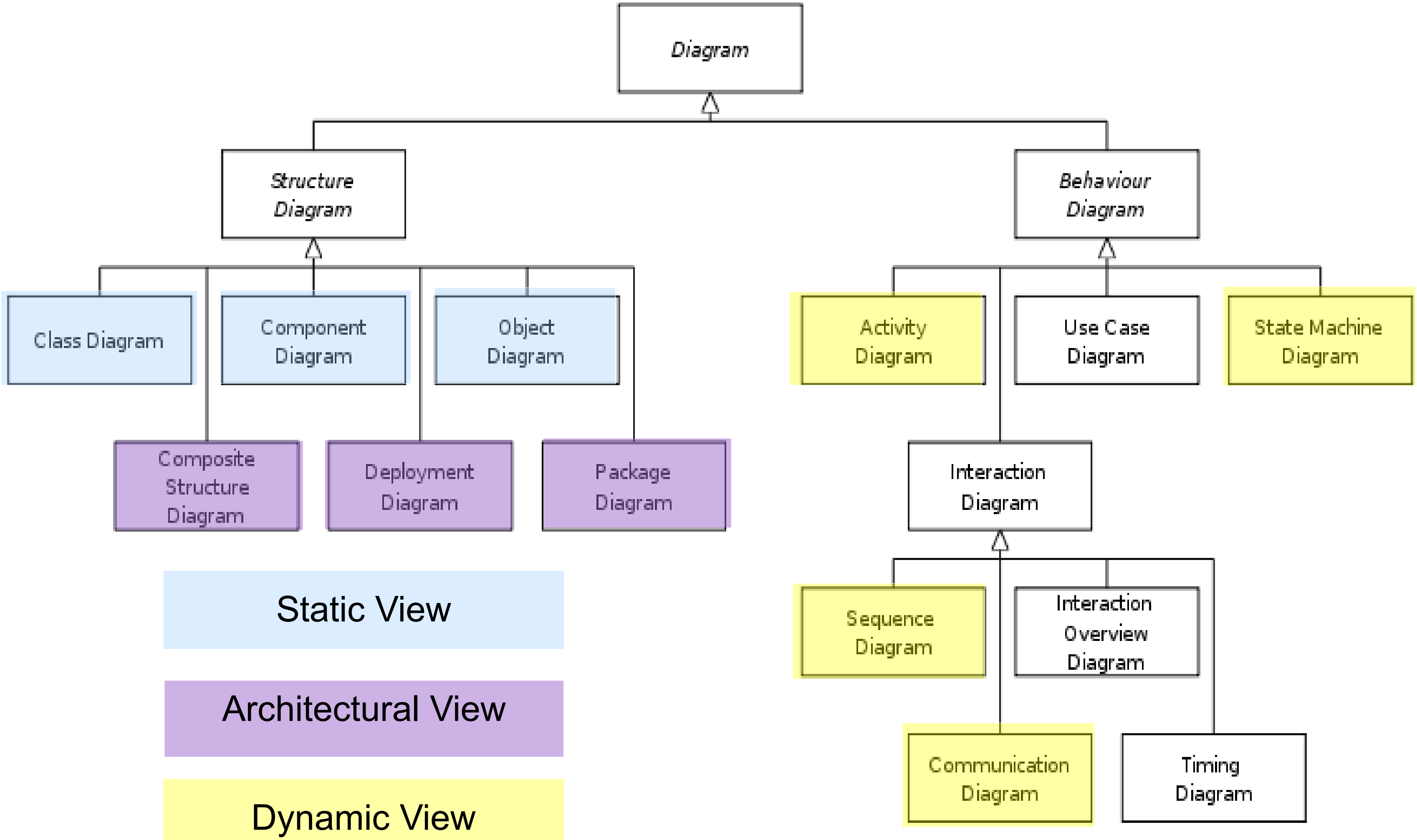
  Paul R. Reed Jr., Addison Wesley, 2002
- The Unified Modeling Language User Guide, Grady Booch, James Rumbaugh, Ivar Jacobson, Addison-Wesley, 2005
- Précis de Génie Logiciel,
  M.-C. Gaudel, B. Marre, F. Schlienger et G. Bernot, Masson, 1996
- The Science of Programming, D. Gries, Springer Verlag, 1981

http://www.omg.org/gettingstarted/what_is_uml.htm

http://www.eecs.ucf.edu/~leavens/JML/

http://www.junit.org/

The UML 2.0 Diagrams (for corresp. models)



Static View

Architectural View

Dynamic View

Principal UML diagram types (1)

Structure	and	Vizualization
Use Case Models	and	Use Case Diagrams
Sequence Models	and	Sequence Diagrams
State Machines	and	State Charts
Class Models	and	Class Diagrams
Object Graphs	and	Object Diagrams

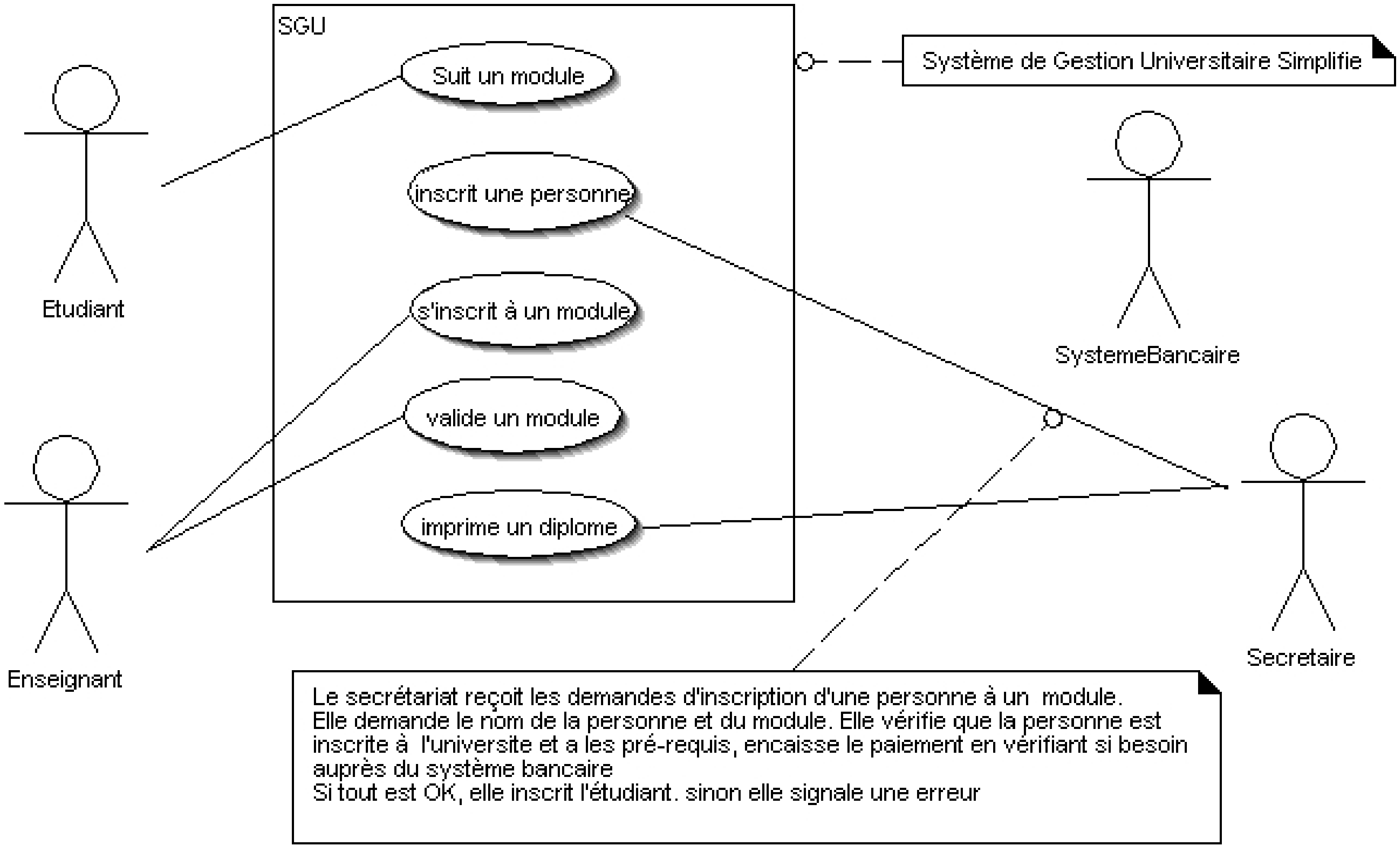
All these Model Elements are described in a UML-document itself, the „Meta-Object-Framework“ (MOF)

Principal UML diagram types (1)

- **Use Case Diagrams** („Diagrammes des cas d'utilisation") : models the system **operations by**
  - the **interactions** of the system with the external world (external agents communicating with the system seen as a black box.)
  - Just the priciple cases, the alternatives, the extensions

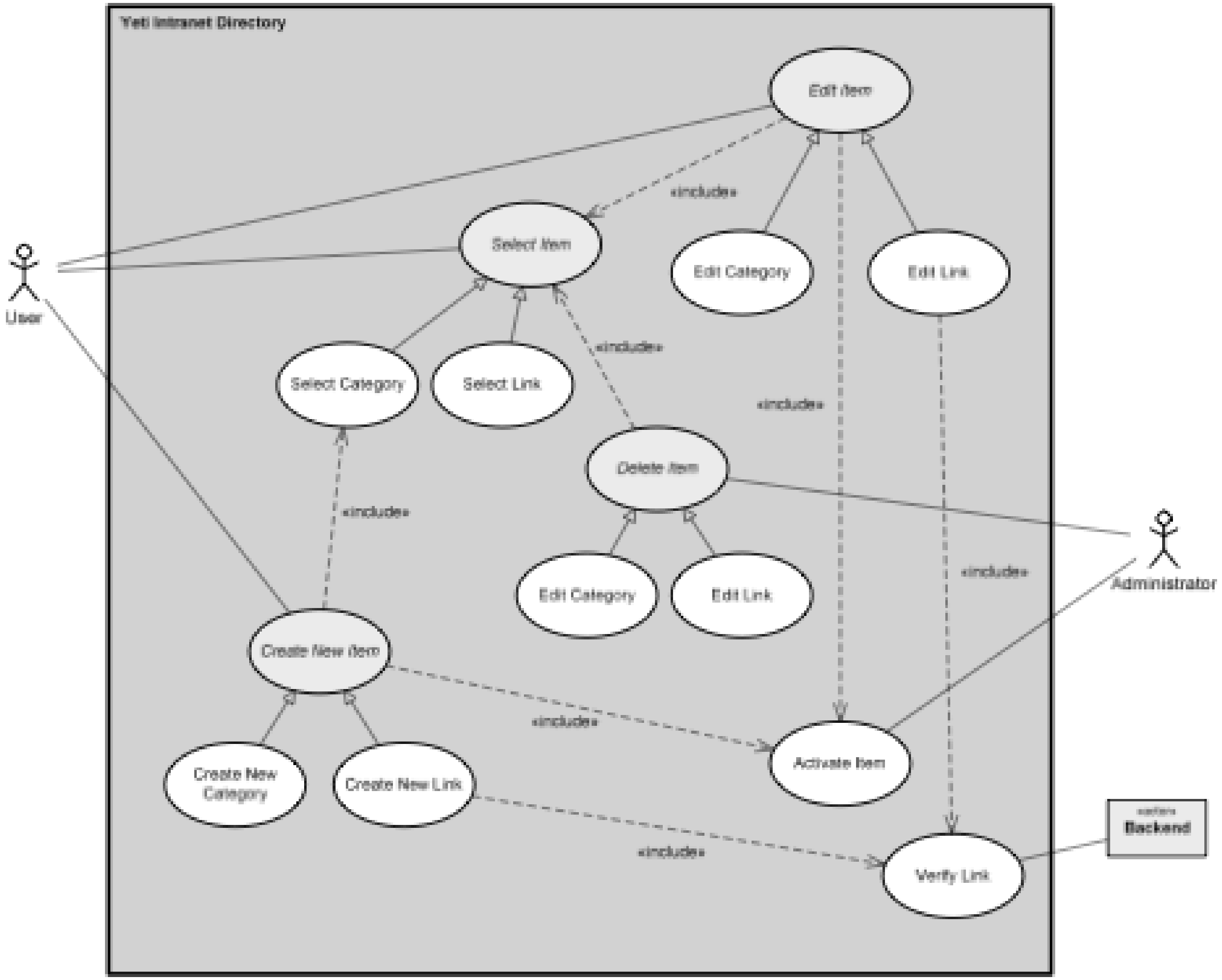
  Emphasis on (top-level) functionality !

Example: Use Case Diagram (Analysis)



Le secrétariat reçoit les demandes d'inscription d'une personne à un module.
Elle demande le nom de la personne et du module. Elle vérifie que la personne est inscrite à l'universite et a les pré-requis, encaisse le paiement en vérifiant si besoin auprès du système bancaire
Si tout est OK, elle inscrit l'étudiant. sinon elle signale une erreur

Example: Use Case Diagram (Design)



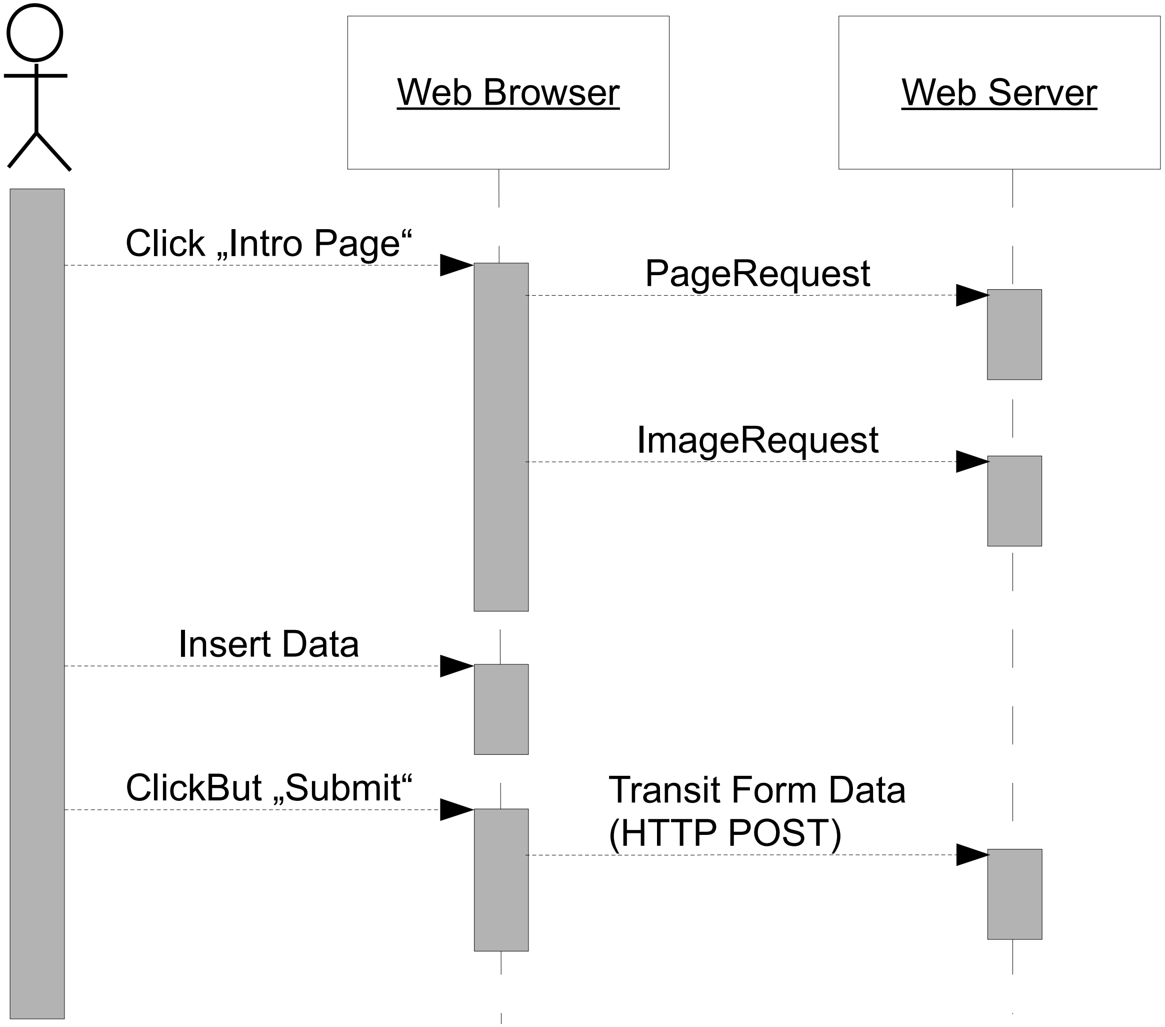
Principal UML diagram types (2)

- **Interaction Diagram** („Diagrammes d'interaction"): the interacion between objects for realizing a functionality
  - **SequenceDiagram**: privileged temporal description of exchanges of events. Notions of utilization **scenarios**.
  - **Collaboration Diagram**: centered around objects and top-level collaborations of them.

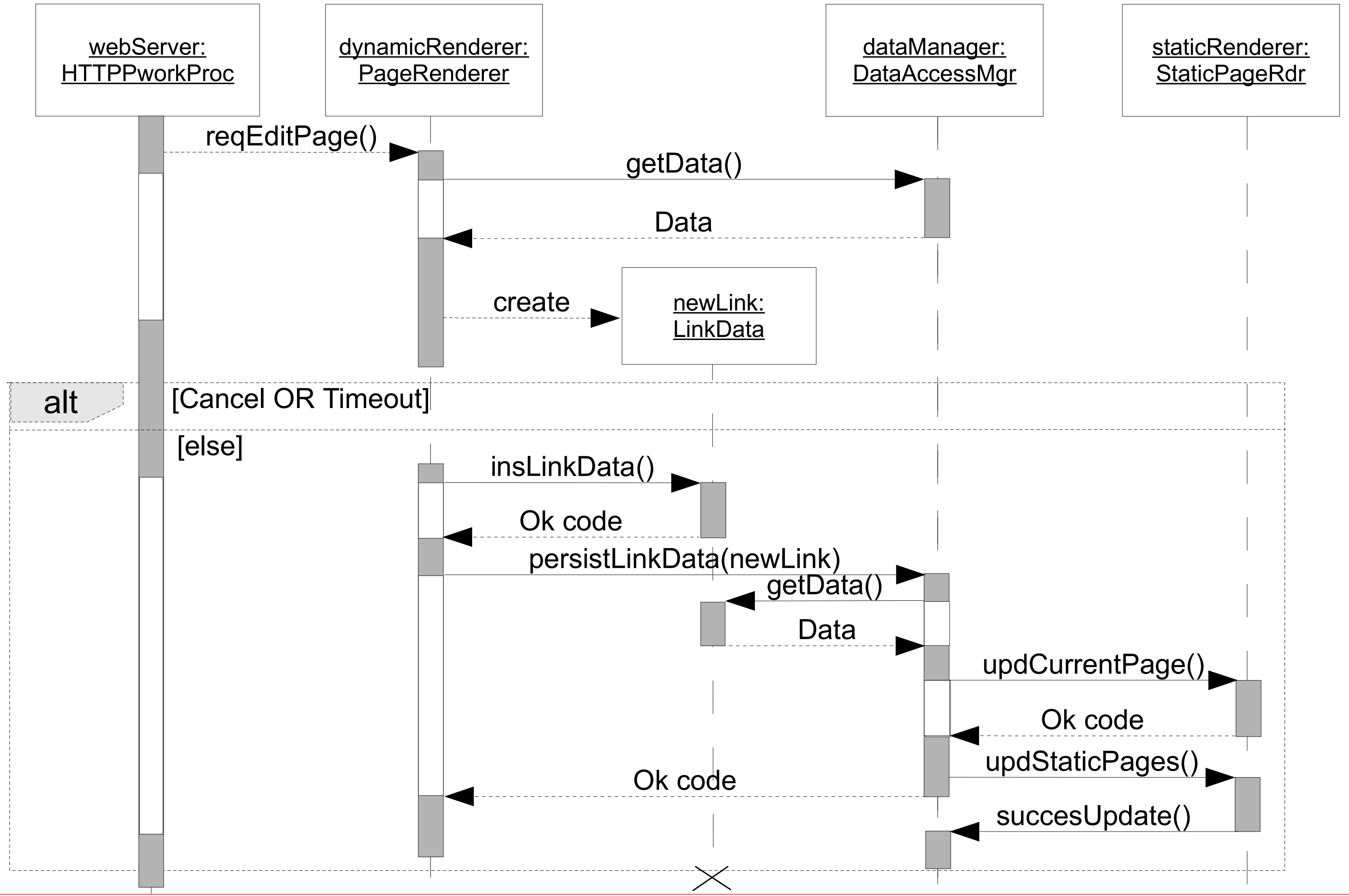
Example: Sequence Diagram (analysis-level)

- **SequenceDiagram**: privileged temporal description of exchanges of events, in particular exchanges between the stake-holders and the system.
  - no messages between actors
    (outside the model)
  - no intern messages of the system
    (this is described in a design-level sequence diagram)
  - No loops, conditions, etc.
- Objectives:
  - describe particular use-cases
  - describe abtract tests

Example: Sequence Diagram (analysis-level)



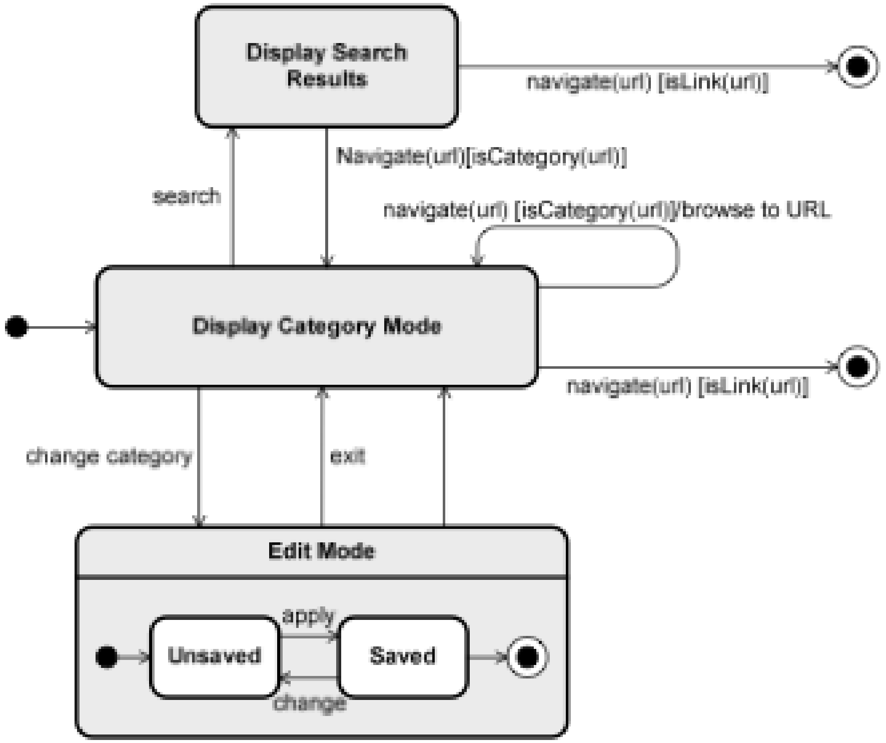
Example: Sequence Diagram (design-level)



Principal UML diagram types (3)

- **State Charts** (ou « machine à états ») :
  a description of **behaviour** by (hierarchical) automata
  - interesting if an object reacts on events (asynchronous as well as synchronous) by the external environment
  - or if the internal state of an object leads to a somewhat interesting life-cycle of an object (transitions between well-characterized states of the object)

Example: **State Chart (design level)**



Display Search Results
navigate(url) [isLink(url)]
Navigate(url)[isCategory(url)]
search
navigate(url) [isCategory(url)]/browse to URL
Display Category Mode
navigate(url) [isLink(url)]
change category
exit
Edit Mode
apply
Unsaved
Saved
change

Summary: State Charts

- Two types can be distinguished:
  - **Semantics of Diagrams for requirements analysis:** many.
  - **Semantics of Diagrams for system design:** **many**.

    Can be interpreted in by automata, process calculi, Labelled Transition Systems (LTL) in several, reasonable ways (depends on context and application).

Main UML diagram type: Class Diagrams

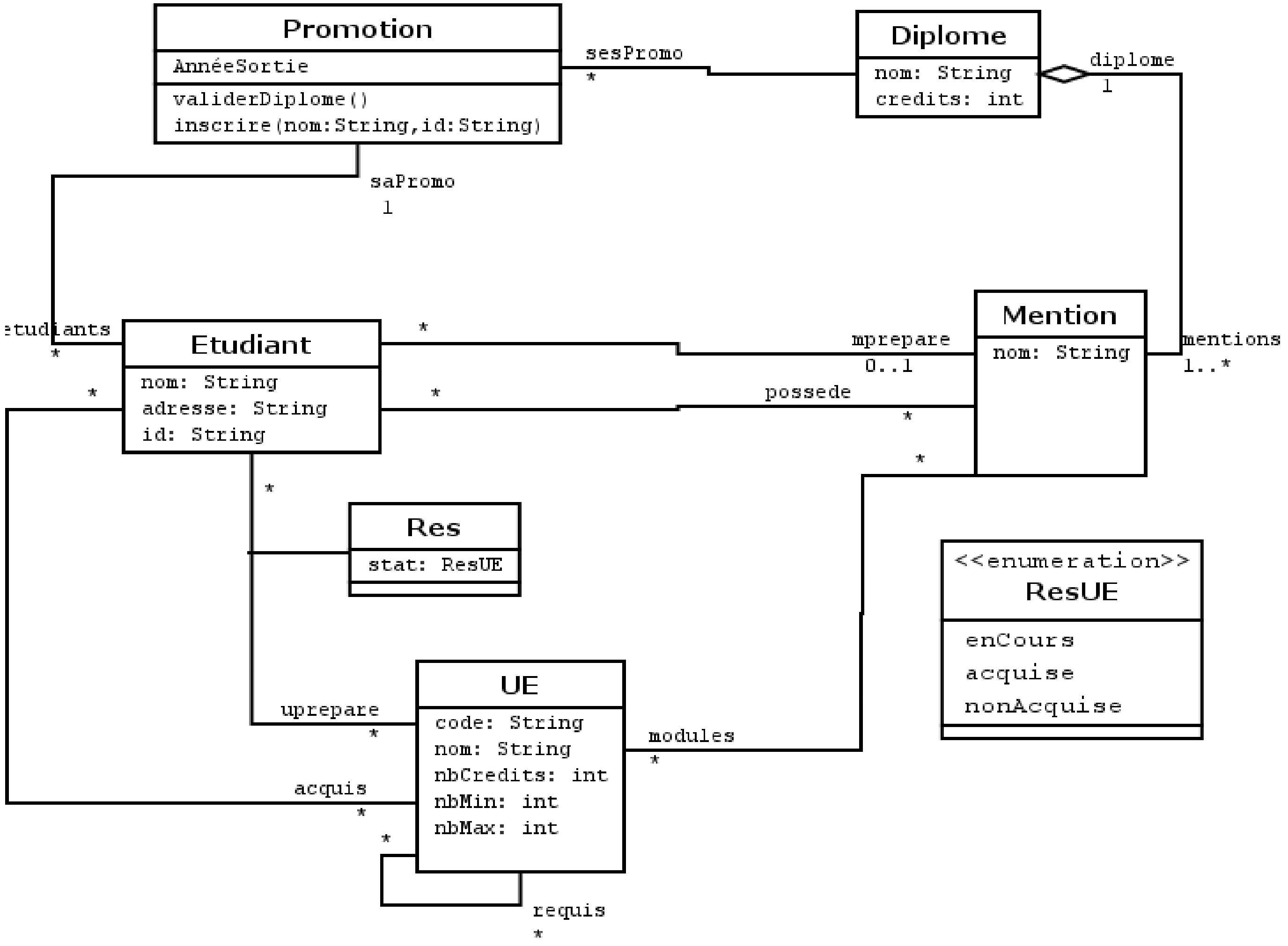
- **Class Diagrams** („Diagrammes de classes") :

  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 …

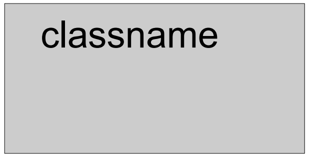
  can be used for top-level views as specific interfaces for local code ...

Example: A Class Diagram

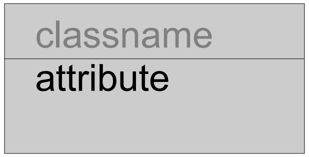


A propos Class Diagrams (1)

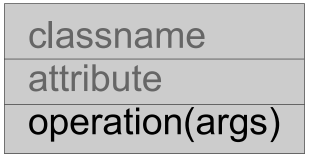
- Model-Elements
  - Class



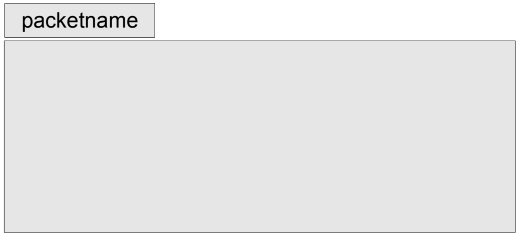
  - Attributes



  - Operations (methods)



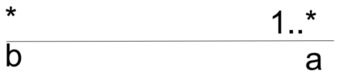
  - Packages (grouping mechanism for parts of a class model)



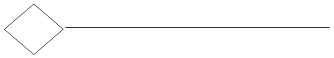
A propos Class Diagrams (2)

Model-Elements

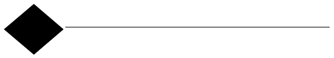
- Association (with optional roles cardinalities)



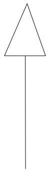
- Aggregation (« has a » relationship with weak linkage)



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



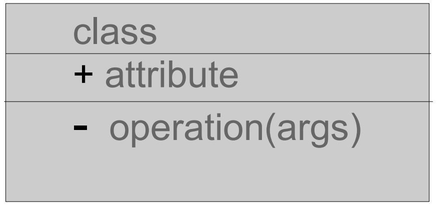
- Specialization (modeling of a „is-a" relationship between classes)



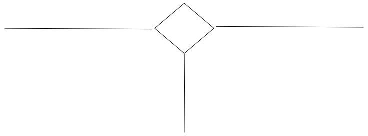
A propos Class Diagrams (3)

Model-Elements

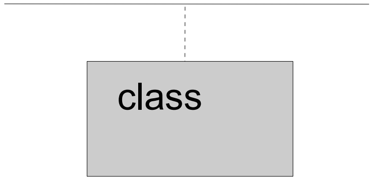
- Visibilities
  (optional public
  and private, see more later)



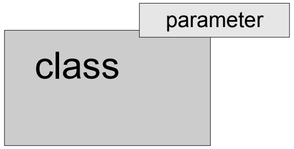
- N-ary associations



- *Association Class*



- templates with parameter
  (usually classes)



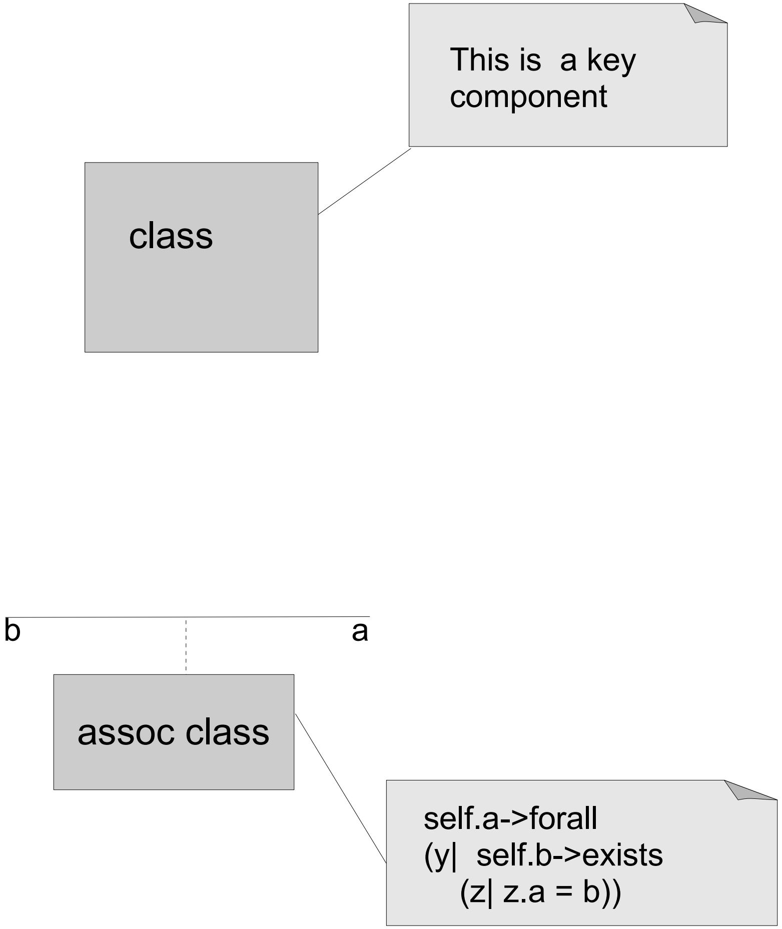
A propos Class Diagrams (4)

- Model-Elements

  - Annotations

    ... typically on classes

    ... can be informal text as
    well as a mathematical notation
    or OCL (see next part !)



This is a key component

class

b

a

assoc class

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

A propos Class Diagrams (1)

- Semantics: Classes are:
  - types of objects
  - tuples „attributes" AND
    association ends (« roles »),
    which are collections (Set, Sequence, Bag) 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.

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 (we will refine this notion later)

A propos Class Diagrams (3)

- operations (in an analysis class diagram)
  - we will only distinguish operations linked to a use-case diagram
  - we will sometimes not even link them to a specific class – this will come later.
- operations (in an design class diagram)
  - a complete interface;
    can be compiled from a JAVA Interface !

Class Diagrams in Requirements Analysis

The **static aspects** of a model were represented by

- **The class diagram**

  Classes with their attributes

  Class hierarchies via inheritance

  Relations between classes (associations + cardinalities)

  The „roles“ at the association ends give an intuitive semantics

- The **invariants** make the description complete ...
  ce qui n'est pas exprimable directement dans le diagramme

  Plages de valeurs ou contraintes sur des attributs

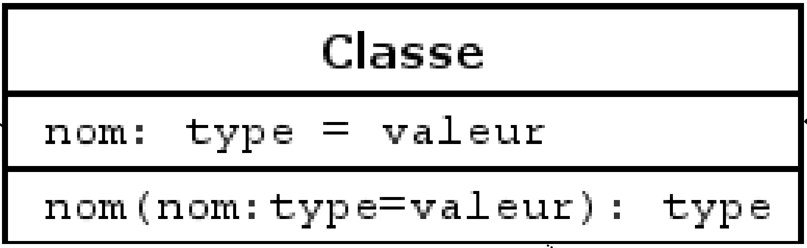
  Contraintes complexes sur une association isolée

  Contraintes globales sur un ensemble d'attributs/associations

  Contraintes sur un ensemble d'instances des classes

More Specific Details in UML 2

***Visibilities**:*
+: public
- : private
#: protected
/ : derived



Classe

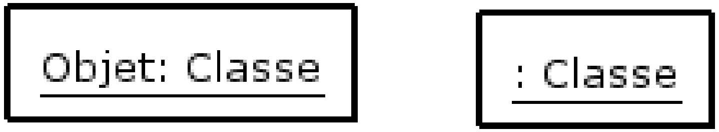
nom: type = valeur

nom(nom:type=valeur): type

***Modifiers**:*
static
abstract

***Parameter modes**:*
in (par défaut)
out
in out

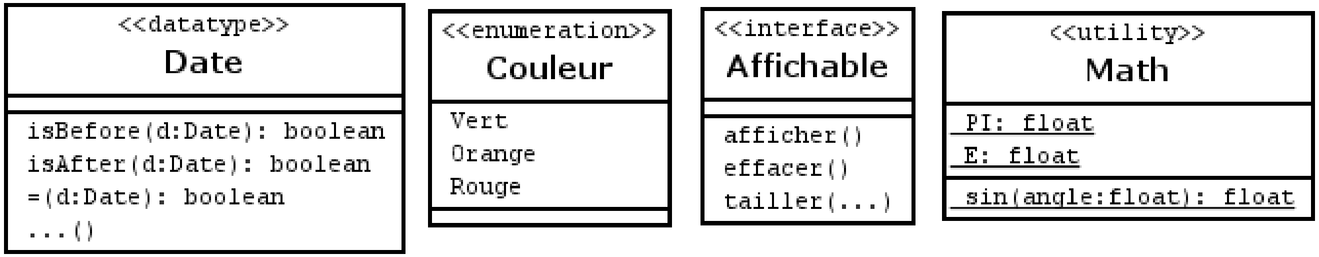
***Instances**:*



Objet: Classe

: Classe

Stéréotypes:



<<datatype>>
Date

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

<<enumeration>>
Couleur

Vert
Orange
Rouge

<<interface>>
Affichable

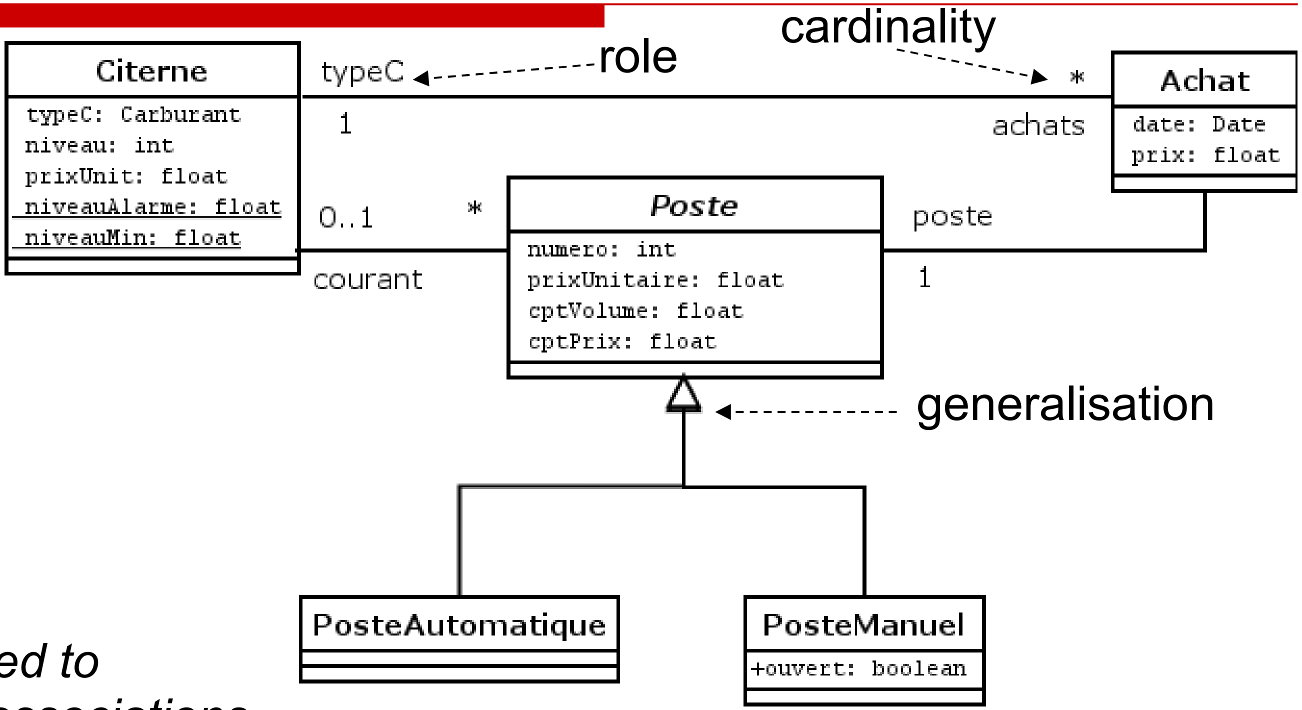
afficher()
effacer()
tailler(...)

<<utility>>
Math

PI: float
E: float

sin(angle:float): float

More Specific Details in UML 2



The roles were used to navigate accross associations

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

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

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

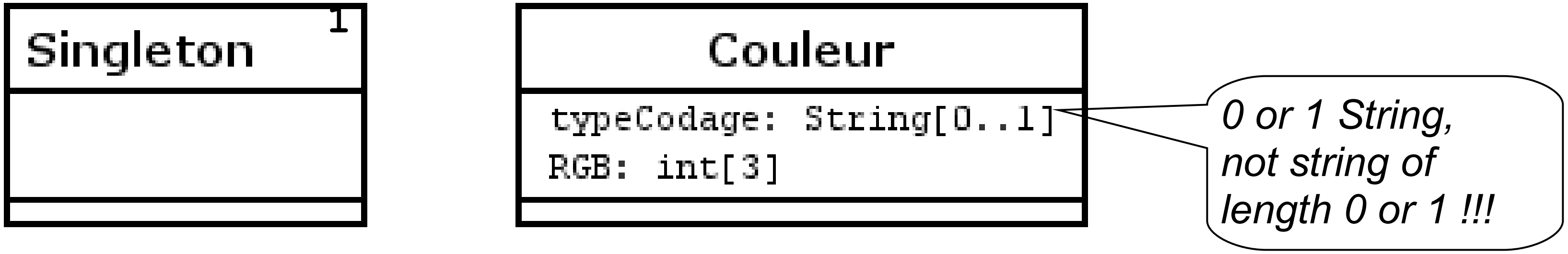
Le nom de classe peut servir de rôle par défaut (si pas d'ambiguïté)

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)
  - on donnera systématiquement les cardinalités
  - Attention à la distinction: une instance (1), au plus une instance (0..1), une collection d'instances (* ou 1..*)

Multiplicities on attributs and classes can be:



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

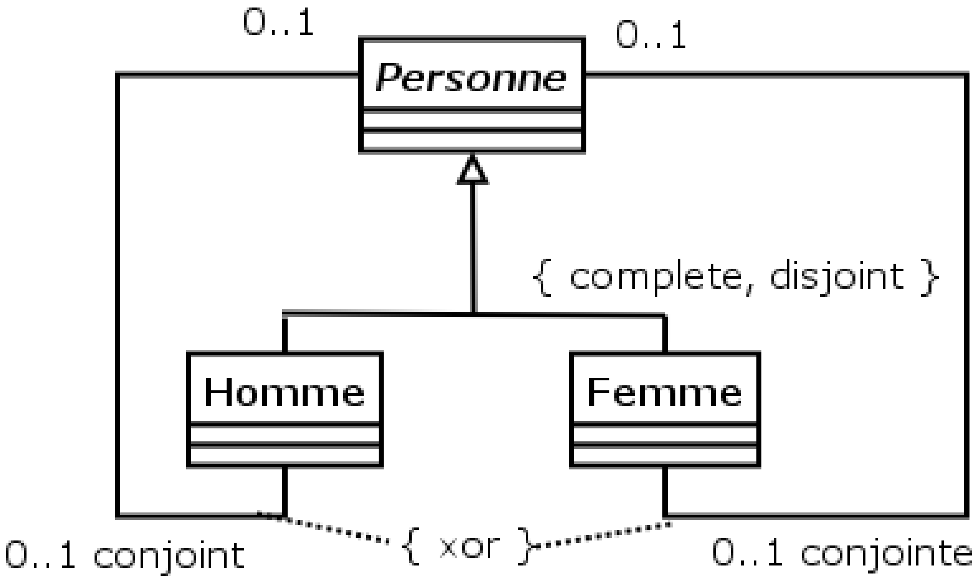
More Specific Details in UML 2

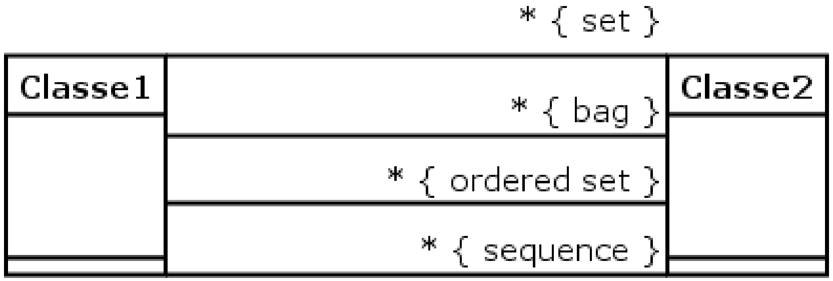
Contraints on associations

- For generalisation:
  - `complete,incomplete`
  - `disjoint,overlapping`

- Between associations
  - `xor`

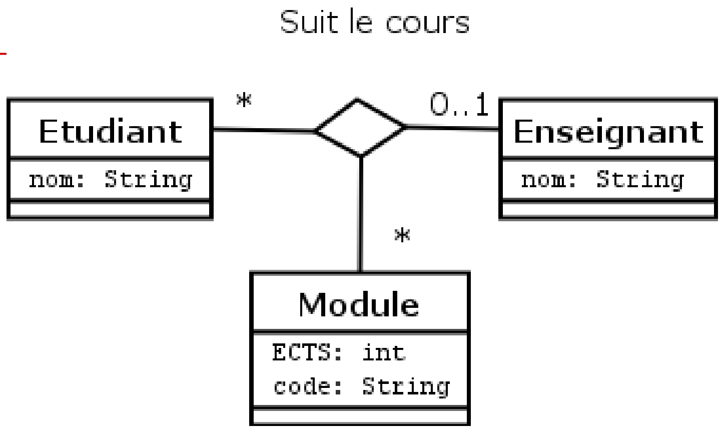
- Collection Types may now also be specified !!!
  - `no duplicates, unordered`
  - `duplicates, unordered`
  - `no duplicates, ordered`
  - `duplicates, positioned`



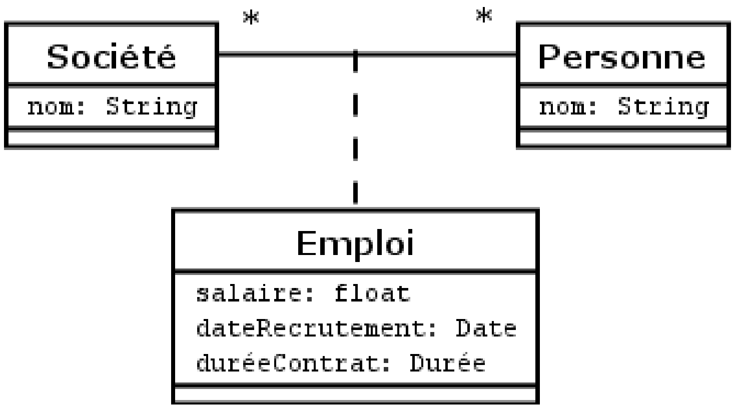


More Specific Details in UML 2

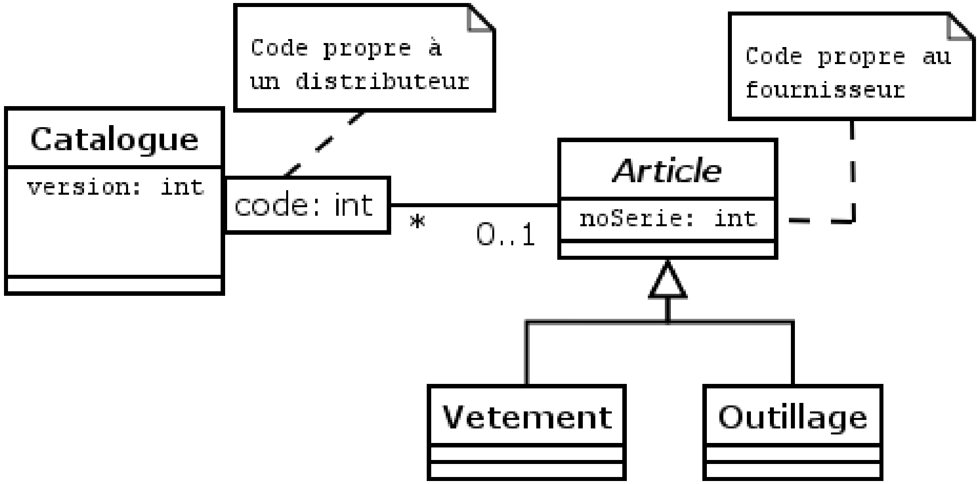
N-ary Associations



Association with attributes

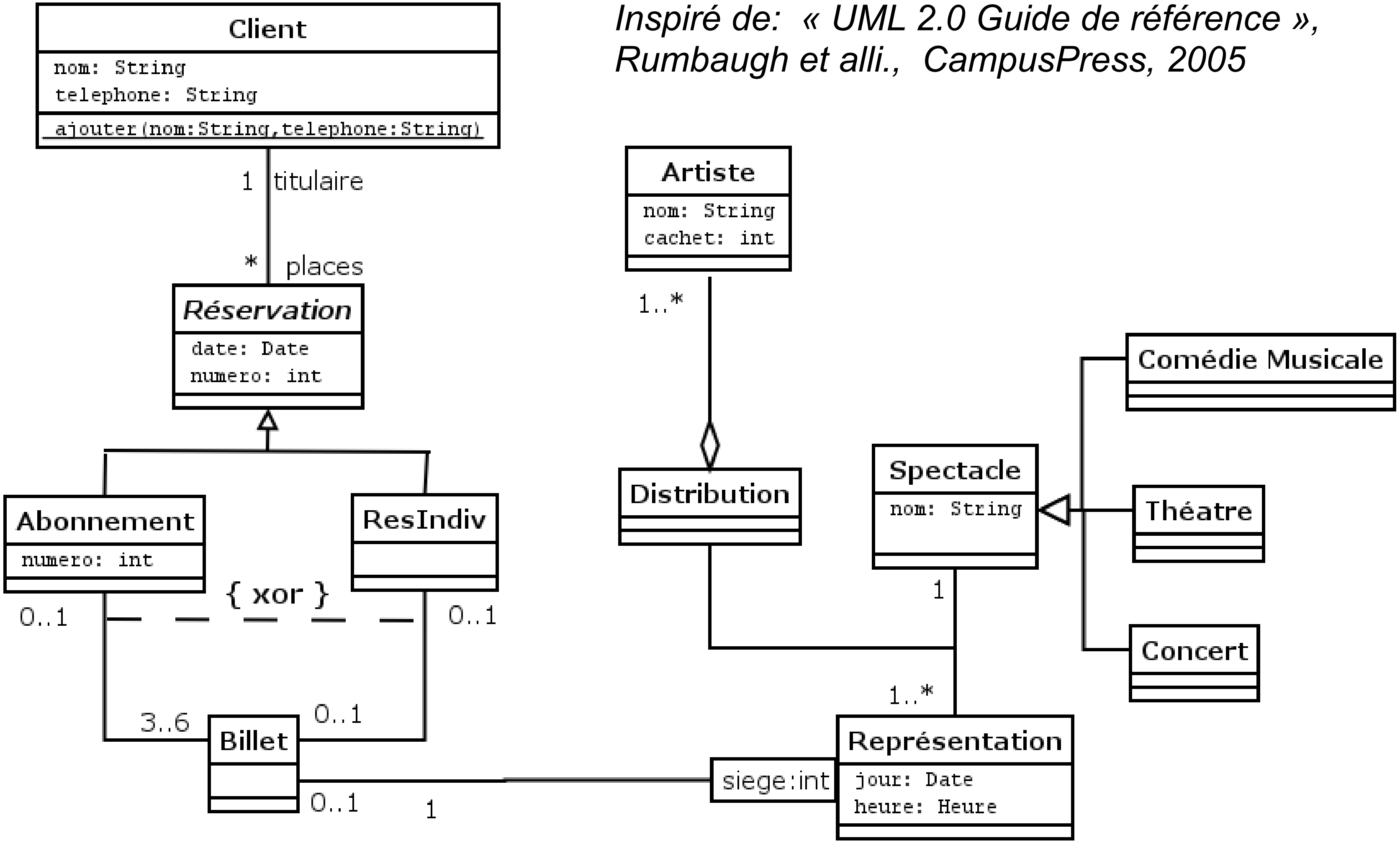


Association « qualified »



Putting all together …

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



Principal UML diagram types (5)

- **Object Diagrams** („Diagrammes d'objects") :

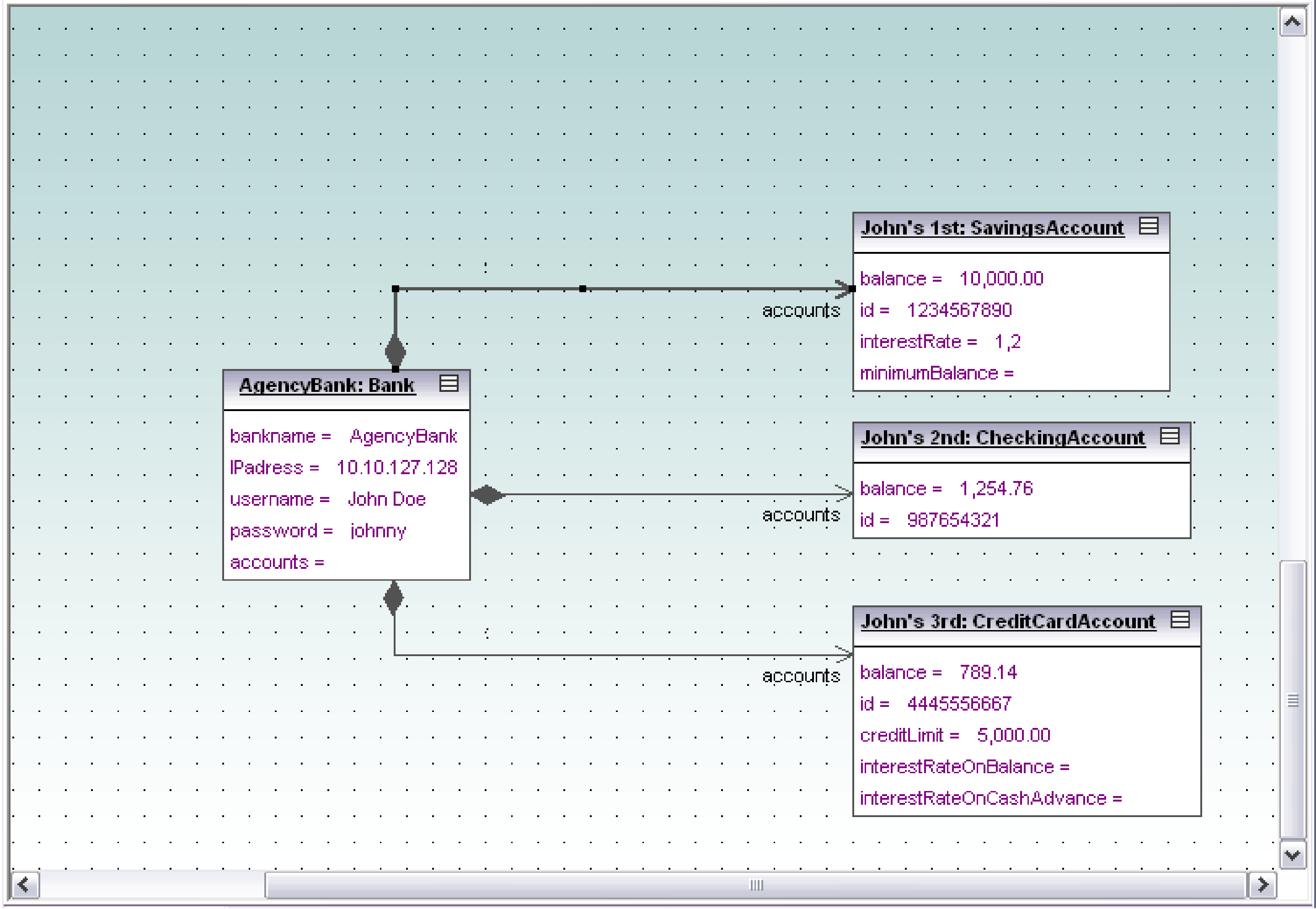
  denote a concrete system state,

- 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

- Corresp. 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

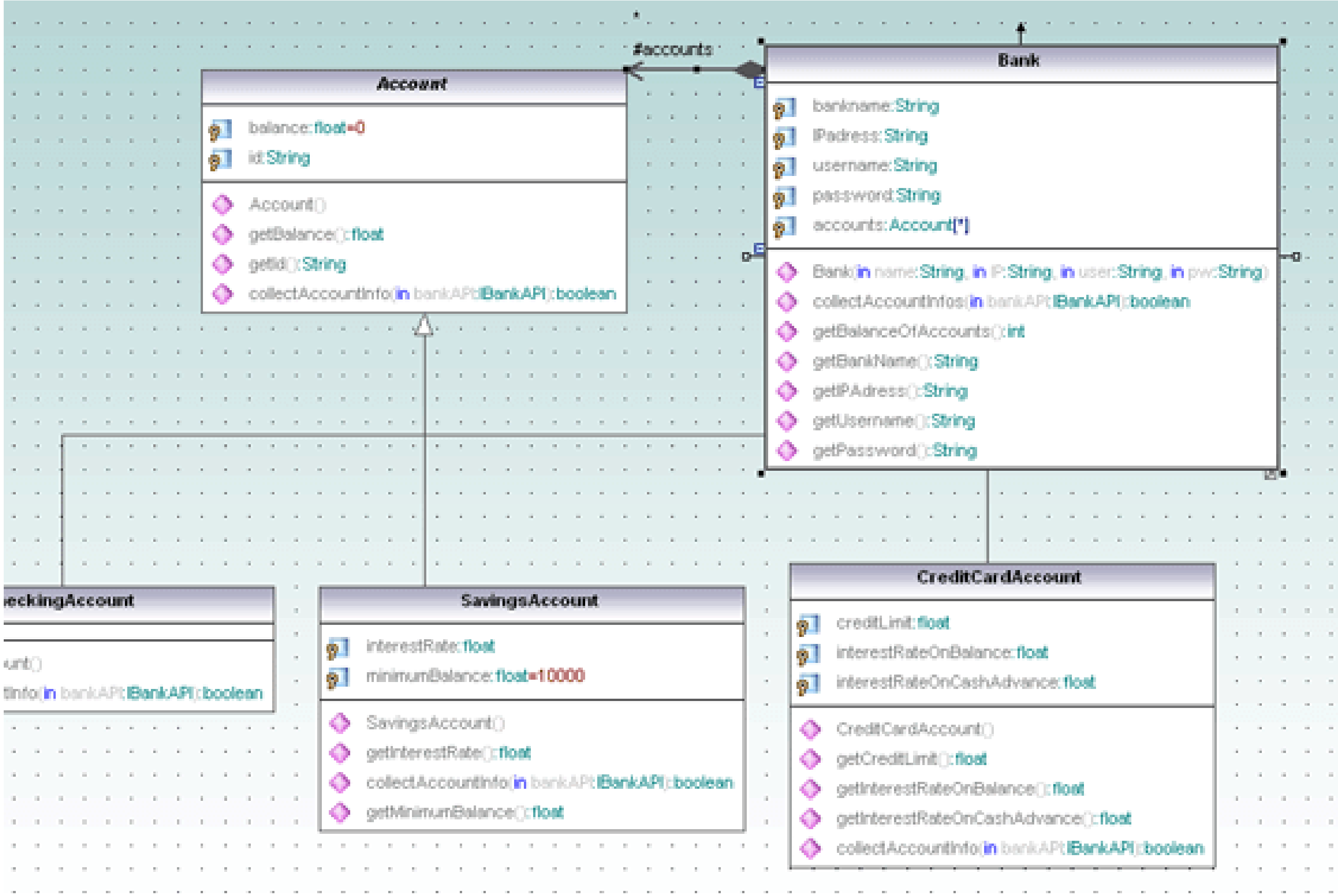
balance = 1,254.76
id = 987654321

John's 3rd: CreditCardAccount

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

Example Object Diagram

- Class 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: Object Diagrams

- Object Models denote a concrete State of a Class Model; Class Diagram denote (essentially) a Signature of the elements in the state, as well as the possible operations on them.

  Multiplicities and Cardinalities express INVARIANTS on (valid) Object Models to a given Class Model – with this respect, serves as Specification of States.