Ingegneria del Software

Corso di Laurea in Informatica per in Management

UML Class diagram essentials

Davide Rossi

Dipartimento di Informatica – Scienze e Ingegneria

Università di Bologna

Disclaimer

The content of these slides is intended to give an informal introduction to UML class diagrams. They contain several simplifications and even some intentional mistake.

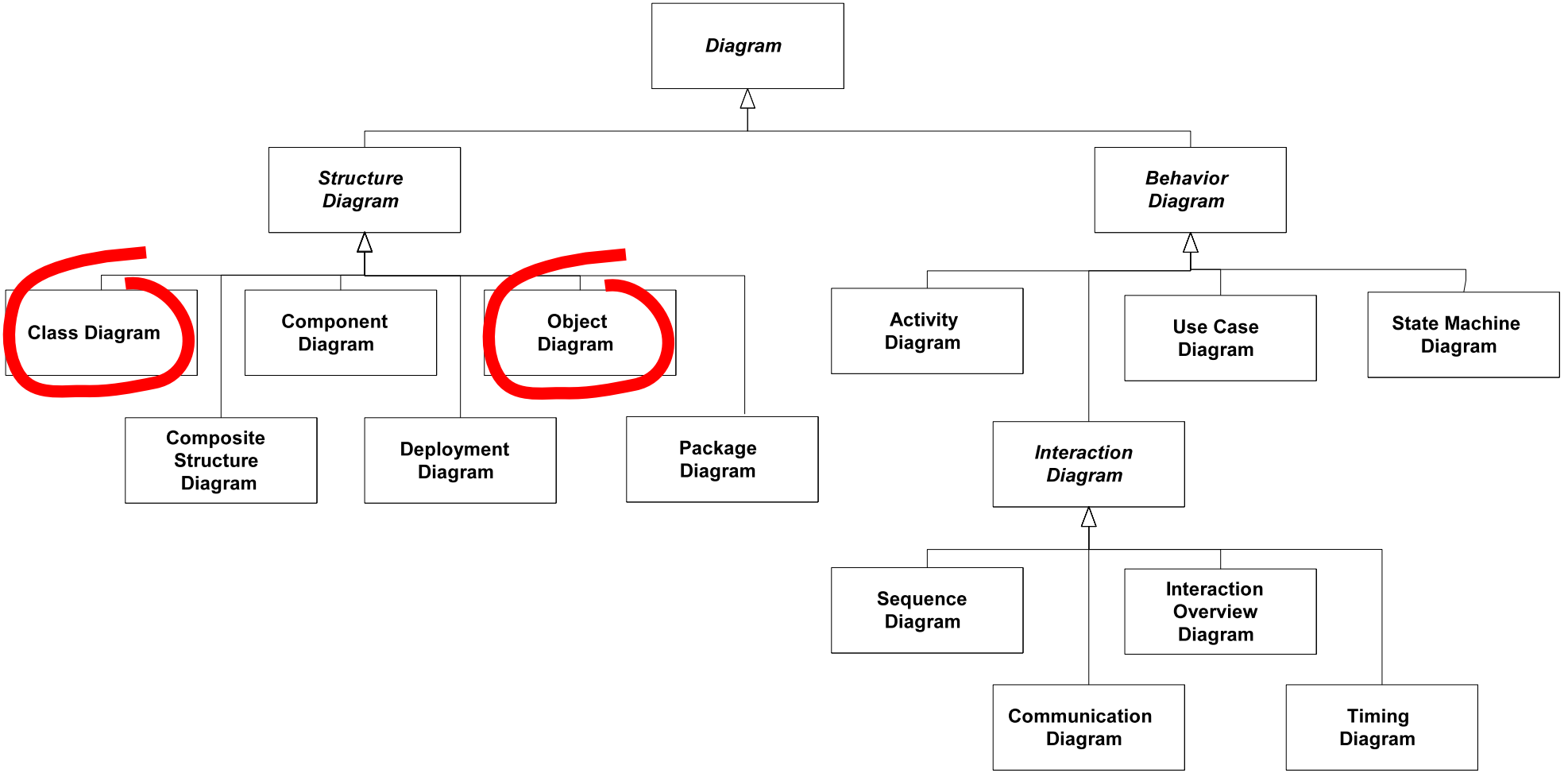


Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Component Diagram
Object Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Interaction Overview Diagram
Communication Diagram
Timing Diagram

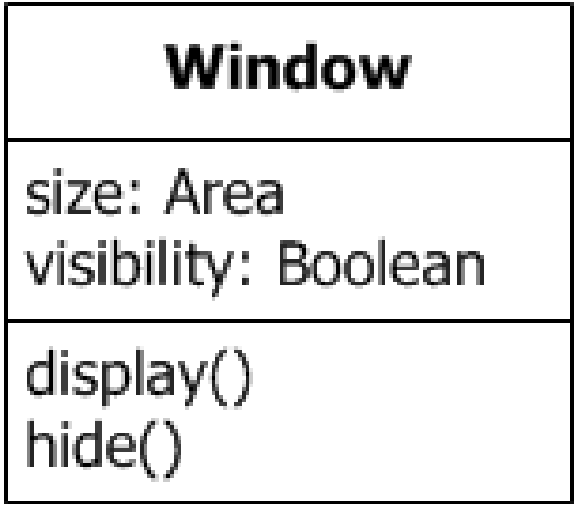
UML classes

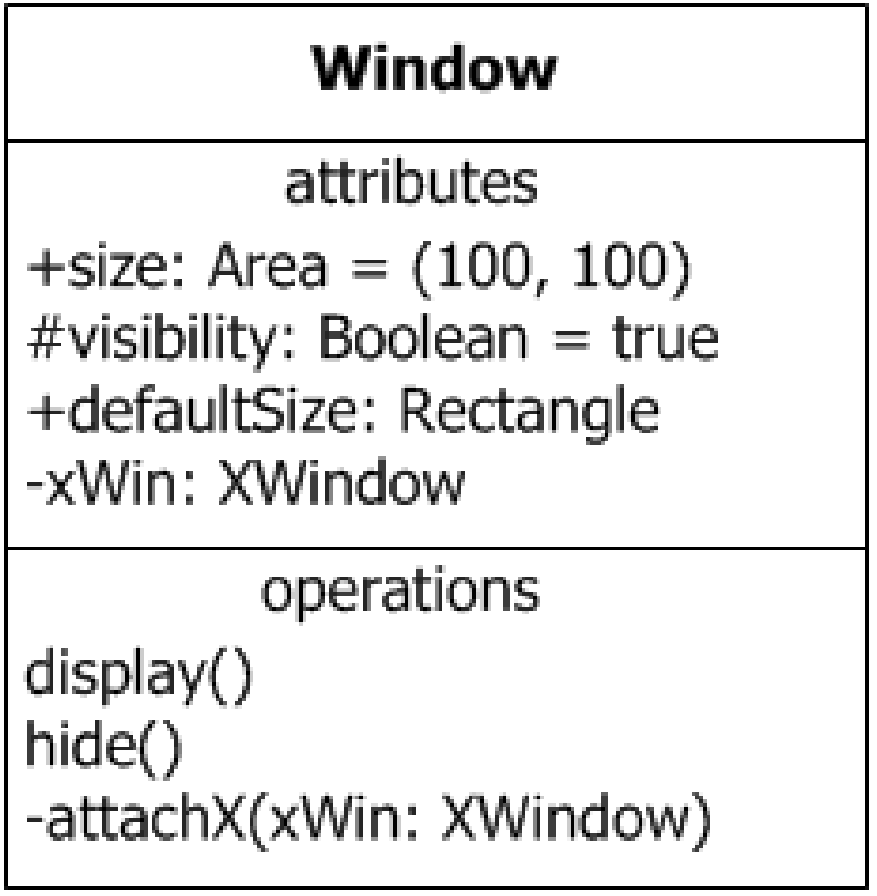
A class is a type representing objects sharing common features.

A class is depicted as a solid-outline rectangle with the class name in it and is potentially divided in compartments separated by horizontal lines below the name.

Usual compartments are the operations compartment and the attributes compartment.

Classes notation examples





Properties and operations

- Properties are structural features of a class.

  Simplified syntax:

  `[+,-,#,~][/]<name>[:<type>][<mult>]`
- Operations are behavioral features of a class.

  Simplified syntax:

  `[+,-,#,~]<name>[(<params>)][:<type>]`

Visibility kind

- **+** : public - visible to all
- **–** : private - visible within its namespace
- **#** : protected - visible to elements that have a generalization relationship to it
- **~** : package - visible to all elements within the nearest enclosing package

Multiplicity

`<multiplicity-range> ::= [<lower>..] <upper>`

Where **`*`** means unlimited upper bound.

`Ordered/unordered, unique/nonunique` can follow the range.

Often used multiplicities are:

- 1
- 0..1
- 1..*
- `*`

Instances

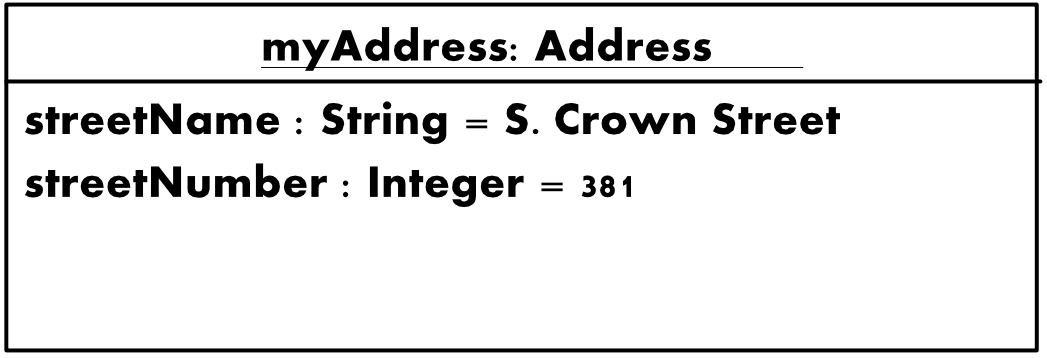
Instances (a.k.a. objects) are, well, instances. They obey the structure of class they are instances of.

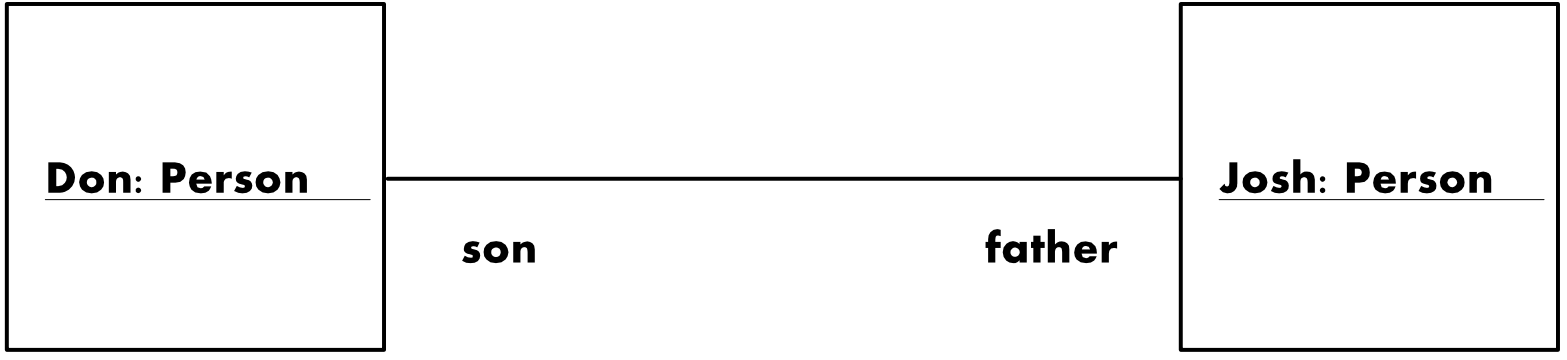
Objects of a class must contain values for each attribute that is a member of that class, in accordance with the characteristics of the attribute, such as its type and multiplicity.

Instances notation

The graphic notation is similar to that of classes but the name appears underlined and is a concatenation of the instance name (if any), a colon (‘:’) and the class it is an instance of.

Instances notation examples





Relationships

Several relationships can be drawn in class diagrams:

- Generalization
- Dependency
- Realization
- Association
- Aggregation
- Composition

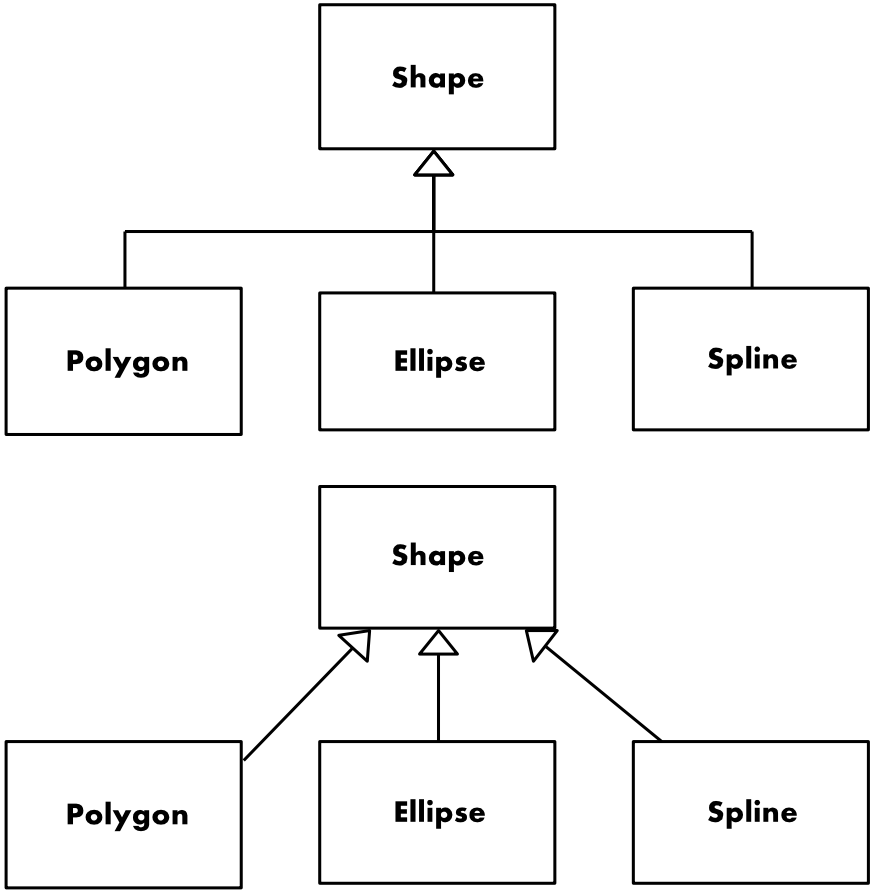
Generalization/specialization

Generalization/specialization are root concepts in MOF that assumes a more detailed behavior when used between classes.

Each Generalization relates a specific class to a more general class and an inheritance-like mechanism is assumed.

An “is-a” relationship exists between the two elements, corresponding to inheritance in programing languages.

Generalization: notation



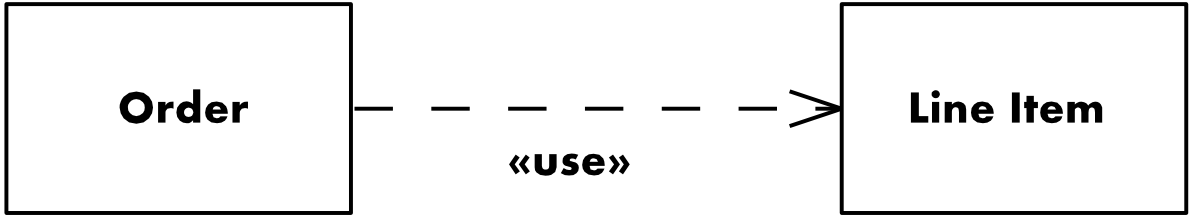
Dependency

A Dependency signifies a supplier/client relationship between model elements where **the modification of a supplier may impact the client** model elements.

A Dependency implies that the semantics of the clients are not complete without the suppliers.

Dependency: notation

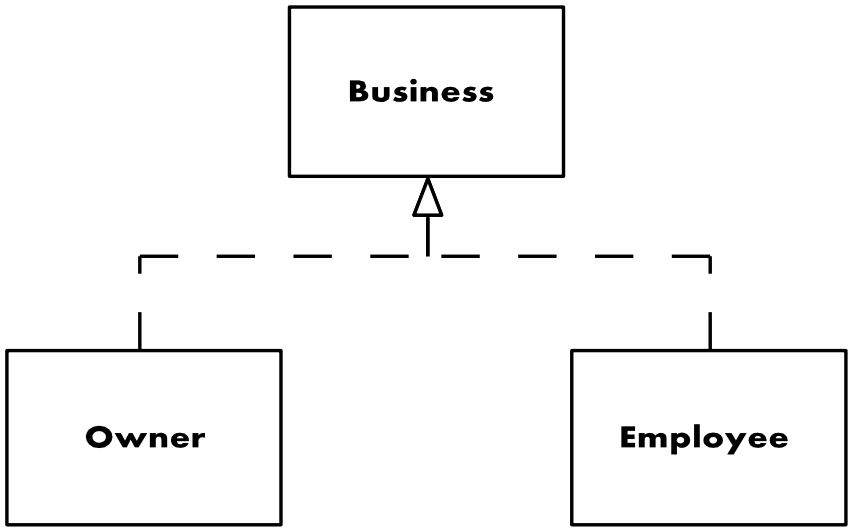
A Dependency is shown as a dashed arrow between two classes. The arrow may be labeled with an optional keyword or stereotype and an optional name.



Realizations

A Realization is a kind of Dependency and is shown as a dashed line with a triangular arrowhead at the end that corresponds to the realized Element.

The realising must realize, or implement, the behavior that the other specifies.



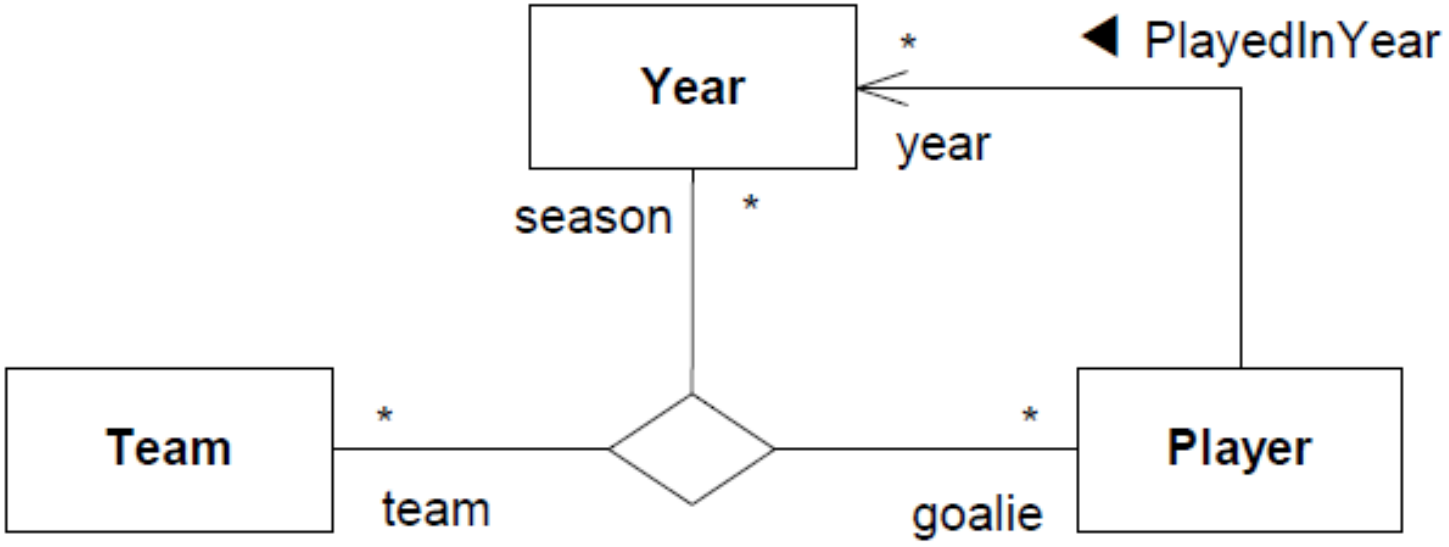
Association

An association declares that there can be links between instances of the associated types. A link is a tuple with one value for each end of the association.

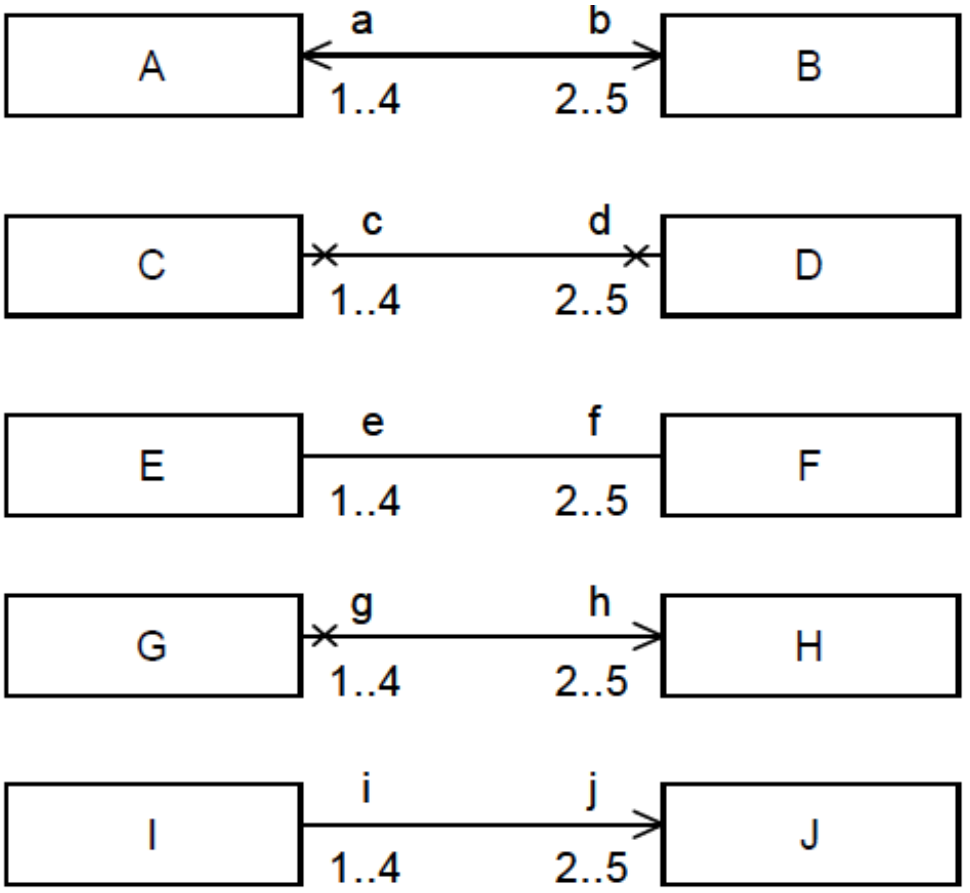
Associations have names that can be displayed and labels with arrows that help reading the direction of the label.

Association ends can have names, multiplicities and navigation arrows or crosses.

Association Class: notation



Navigation: arrows and crosses

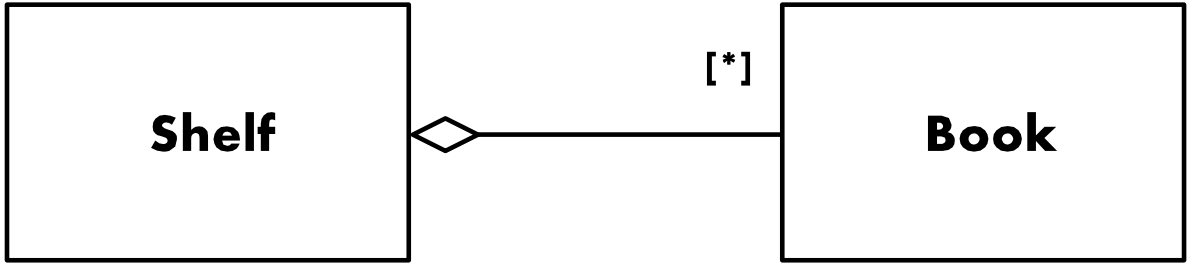


Navigation: arrows and crosses

- **Show all arrows and crosses**: navigation and its absence are made completely explicit.
- **Suppress all arrows and crosses**: no inference can be drawn about navigation.
- **Suppress all crosses**: suppress arrows for Associations with navigability in both directions, and show arrows only for Associations with one-way navigability: In this case, the two-way navigability cannot be distinguished from situations where there is no navigation at all; however, the latter case occurs rarely in practice.

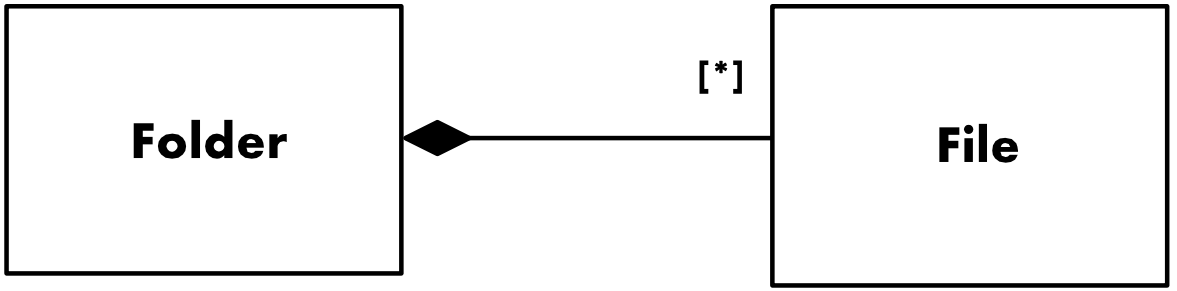
Aggregation

An *aggregation* (real name: *shared aggregation*) tells us that part instance is independent from the composite.



Composition

Composition (real name: *composite aggregation*) is a whole/part binary association. The composite object has responsibility for the existence and storage of the composed objects.



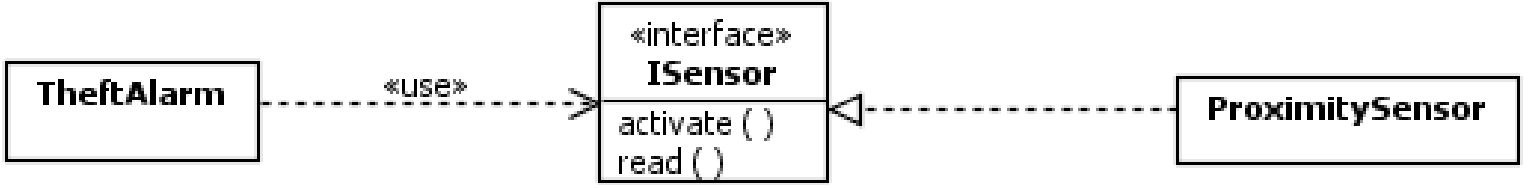
Abstract classes

An abstract class has no direct instances: its instances are instances of one of its specializations.

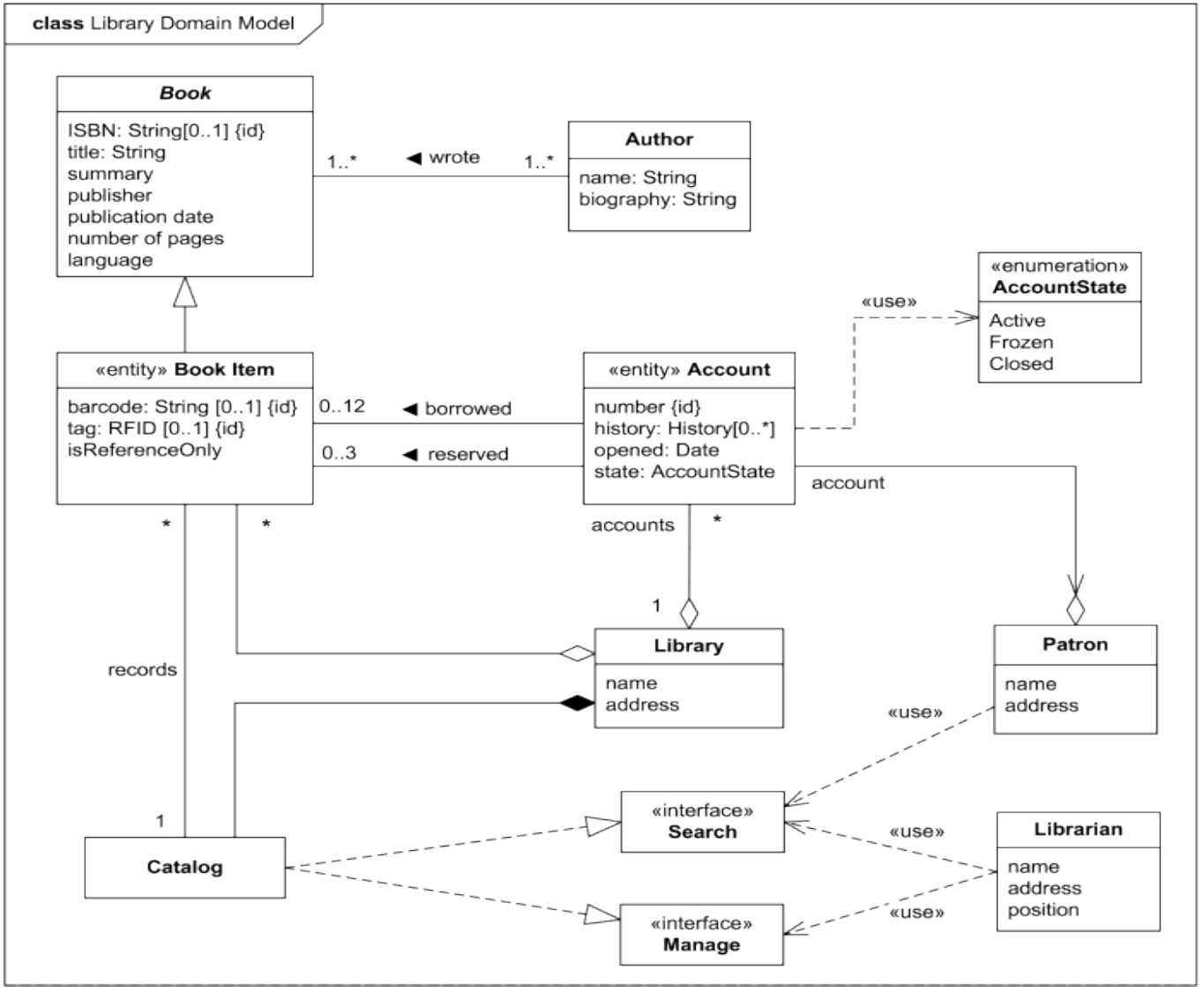
The name of an abstract class is shown in *italics* or/and using the textual annotation `{abstract}`.

Interfaces

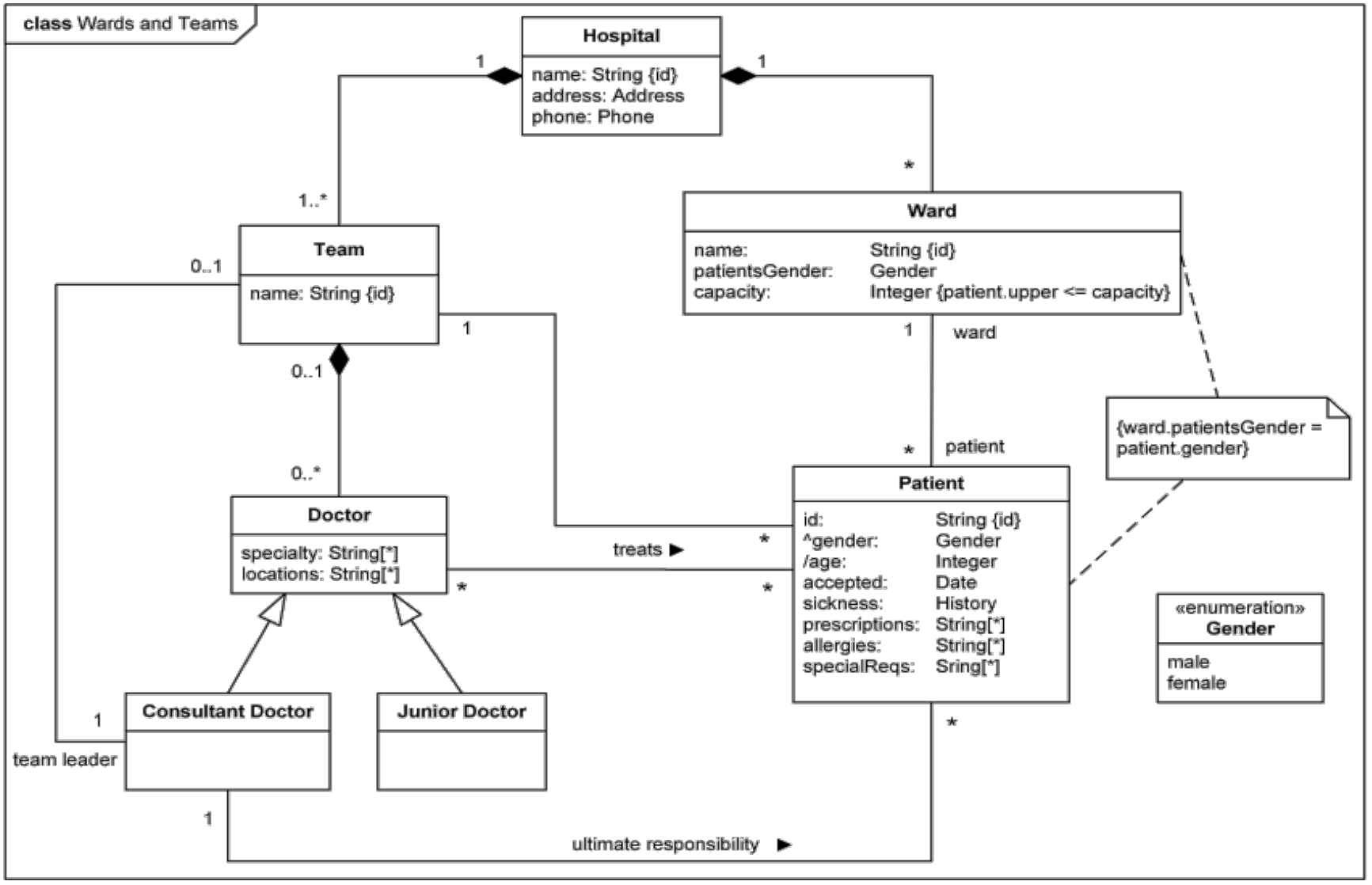
Interfaces declare coherent services that are implemented by classes that implement them. An Interface specifies a contract; any instance of a class that realizes the interface shall fulfill that contract.



Domain models examples (analysis models)



[http://www.uml-diagrams.org]



[http://www.uml-diagrams.org]

Design models examples