# Software Engineering

Norbert Pataki

November 5, 2007

# UML

- Unified Modeling Language. Defined by the Object Management Group.
- standardized specification language for object-oriented modeling, analysis and design.
- a general-purpose modeling language that includes a graphical notation used to create an abstract model of a system
- it was designed to specify, visualize, construct, and document software-intensive systems.
- supports transformations into code: for example, C++, Java, C#, etc. (Case tools, like Rational Rose)

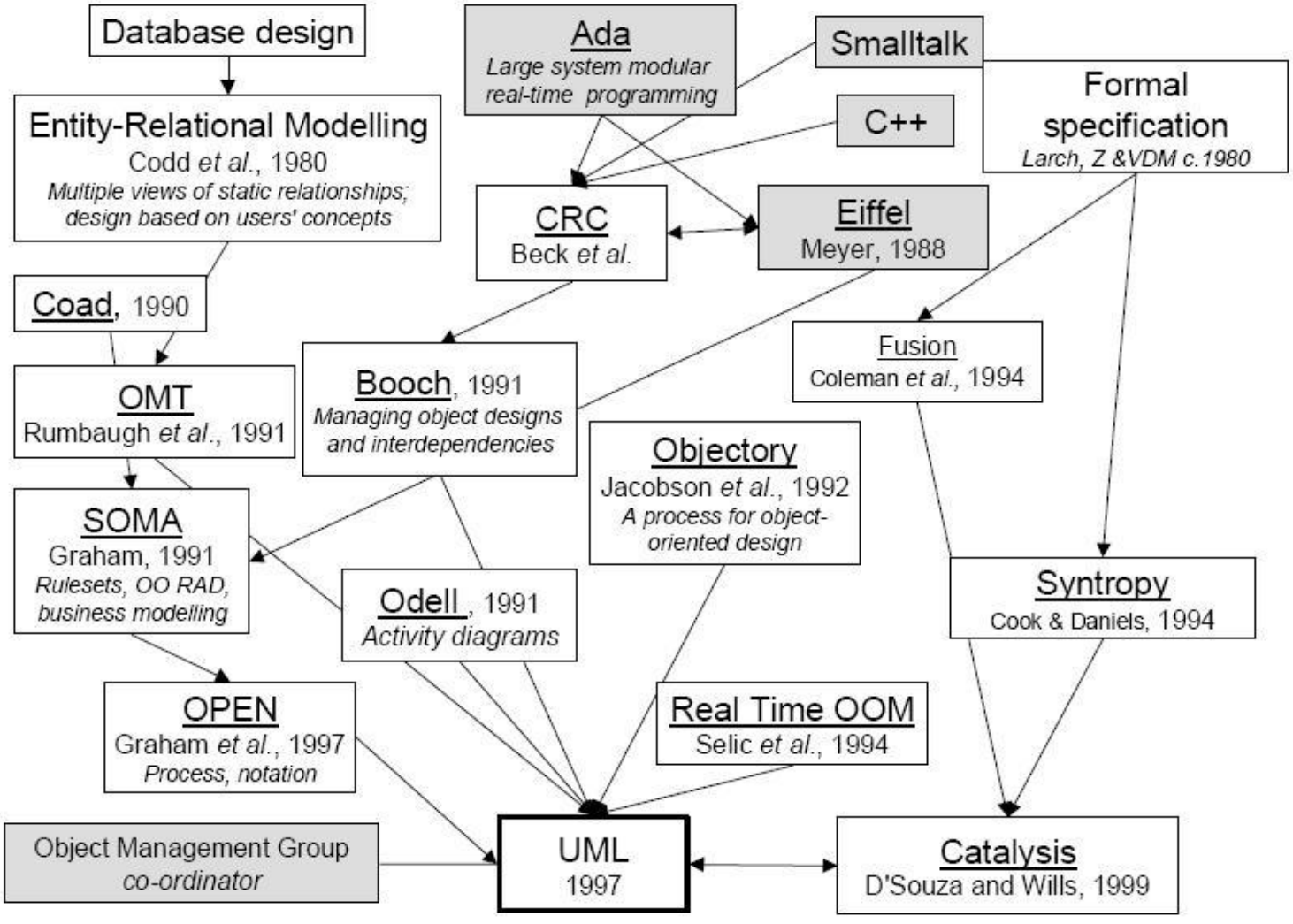
# UML

- it is not restricted to modeling software. It can be used for business process modeling, system engineering modeling, and representing organizational structures.
- UML is able to creating models from different views
- UML is able to describe the exact solution
- UML is a constructional tool
- Concepts, diagrams
- UML is extensible: profiles and stereotypes for customization.
- UML assists developers focusing on the design and the architecture. $\Rightarrow$ model-driven technologies

# Models vs UML Diagrams

- A diagram is a partial graphical representation of a system's model.
- The model also contains a ,,semantic backplane" – documentation such as written use cases that drive the model elements and diagrams.

# Some of the Influences on UML



# History

| Standard | Date |
|---|---|
| UML 0.8 | 1995 |
| UML 1.0 | 1997, January |
| UML 1.1 | 1997, September |
| UML 1.2 | 1998 |
| UML 1.3 | 1999 |
| UML 2.0 | 2004, October |

# Problems with UML

- Language Bloat: UML is large and complex. Many diagrams show redundant information. Some diagrams and constructs are infrequently used.
- Problems in learning and adopting: Hard to adopt and learn UML because of the language bloat.
- Only the code is in sync with the code. The real code is usually not a beautiful model.
- Compatible with every possible implementation language.
- Impedance mismatch: UML represents some systems more concisely or efficiently than others. $\Rightarrow$ Developers are influenced.
- Imprecise semantics: not complete and inconsistent definition. The formal definition is missing.

# UML's concepts

- an UML description consists of relations, elements and diagrams.

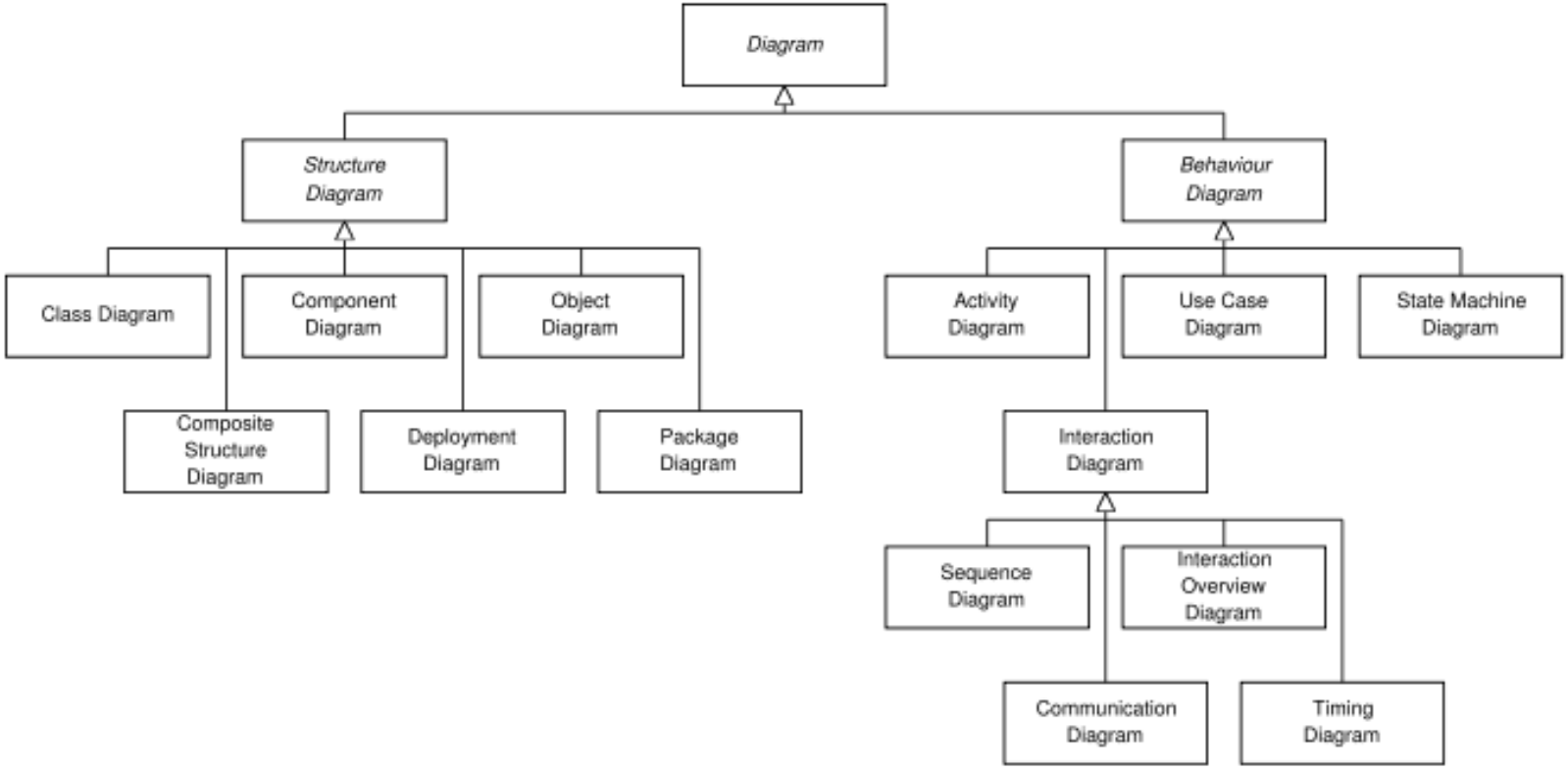
Relationship can be divided into four groups:

- Dependence relationship (semantical connection): company and bank account. The elements must exist on the same abstraction level.
- Associational relationship (structural connenection): company and employee
- Generalizational relationship (link between the general and special): bird and dove
- Implementational relationship (semantical connection between the notation and its implementation): bird and dove

# UML's elements

- Structural elements: object, class, use-case, etc.
- Manifestational elements: interaction, event, state machine, etc.
- Grouped elements: package, subsystem, etc.
- Annotational elements: comment, constraint, etc.

# UML's diagrams



# UML's diagrams

Structure diagrams emphasize what things must be in the system being modeled:

- Class diagram
- Component diagram
- Composite structure diagram
- Deployment diagram
- Object diagram
- Package diagram

# UML's diagrams

Behavior diagrams emphasize what must happen in the system being modeled:

- Activity diagram
- State Machine diagram
- Use case diagram

# UML's diagrams

Interaction diagrams, a subset of behavior diagrams, emphasize the flow of control and data among the things in the system being modeled:

- Communication diagram
- Interaction overview diagram (UML 2.0)
- Sequence diagram
- UML Timing Diagram (UML 2.0)

# UML's diagrams – Some remarks

- The Protocol State Machine is a sub-variant of the State Machine. Network communication protocols can be modeled with it.
- UML does not restrict UML element types to a certain diagram type. In general, every UML element can appear on almost all types of diagrams.

# Diagrams and view

In general, solutions should be represented from many views, for instance:

- View of usage
- Static structural view
- Dynamical view
- View of implementation
- Environmental view

# View of usage

- Who the system provides services (for people, other systems, programs)?
- Who can use these services?
- Can the requirements of the system's services be realized?
- The use case diagram expresses this view.

# Static structural view

- What are the parts of the system?
- What is the function of these parts?
- How the parts work together?
- Class Diagram, Object Diagram

# Dynamical view

- How the parts behave while the problem is being solved?
- How the parts of the system change?
- How the messages flow?
- Activity diagram, Sequence diagram, State machine diagram.

# View of implementation

- Software components
- Connection between these components
- Component diagram, Package diagram

# Environmental view

- Configuration of the software and hardware solution
- What are the hardware and the software requirements for the system?
- Deployment diagram may expresses this view

# UML and documentation

A solution can be documented with the assistance of UML:

- document the project plan
- document the phases of software-producing (requirements, analysis, etc.)
- document the prototypes

# Stereotypes

- Stereotypes allow you to extend the vocabulary of the UML so that you can create new model elements, derived from existing ones, but that have specific properties that are suitable for your problem domain.
- Stereotypes are used for classifying or marking the UML building blocks in order to introduce new building blocks that speak the language of your domain and that look like primitive, or basic, model elements.
- Notation: for instance, <<refine>> – a predefined stereotype.
- Many predefined stereotypes.

# Profiles

- provides a generic extension mechanism for building UML models in particular domains
- A profile is a collection of such extensions and restrictions that together describe some particular modeling problem and facilitate modeling constructs in that domain.
- based on additional stereotypes and tagged values that are applied to elements, attributes, methods, links, and link ends
- make UML proper to special areas: business modeling and others.

# Class diagram

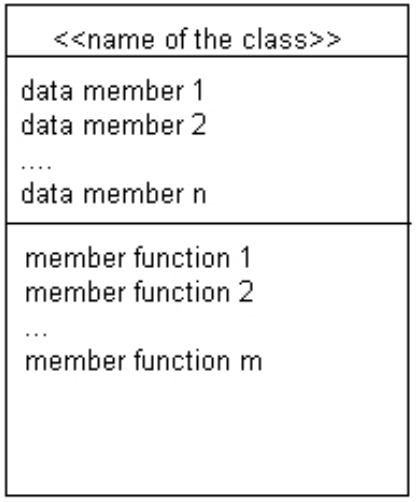
- a type of static structure diagram that describes the structure of a system by showing the system's classes, their members, and the relationships between the classes.
- a class diagram is a connected graph that describes the structure of solution. The vertexes of the graph are classes. The edges of the graph are relations (between classes).

# Relations

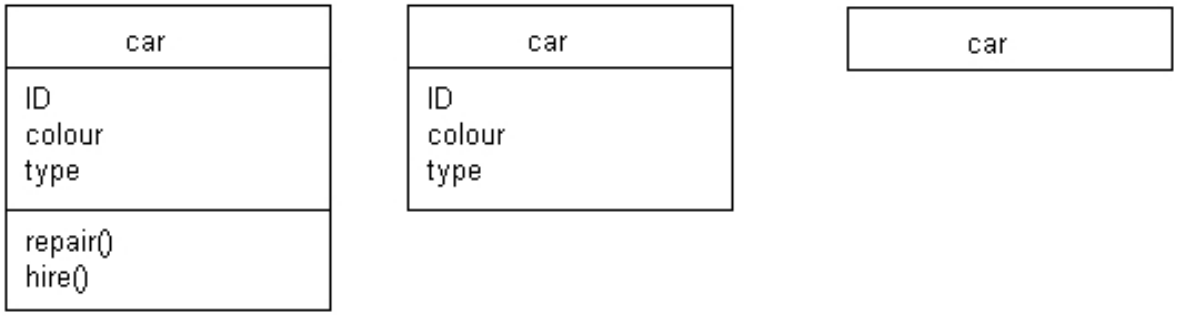
The relation can be:

- Inheritance
- Aggregation
- Composition
- Association
- Dependency

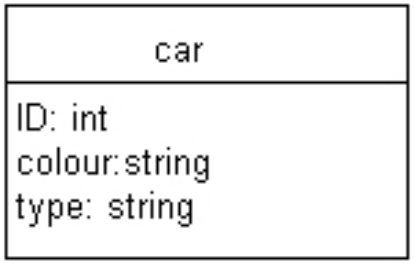
# Class diagram



# Class diagrams



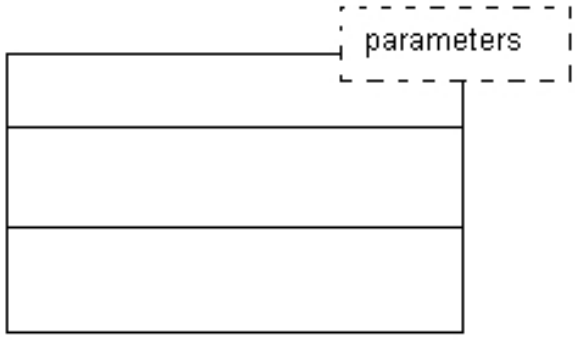
# Class diagrams



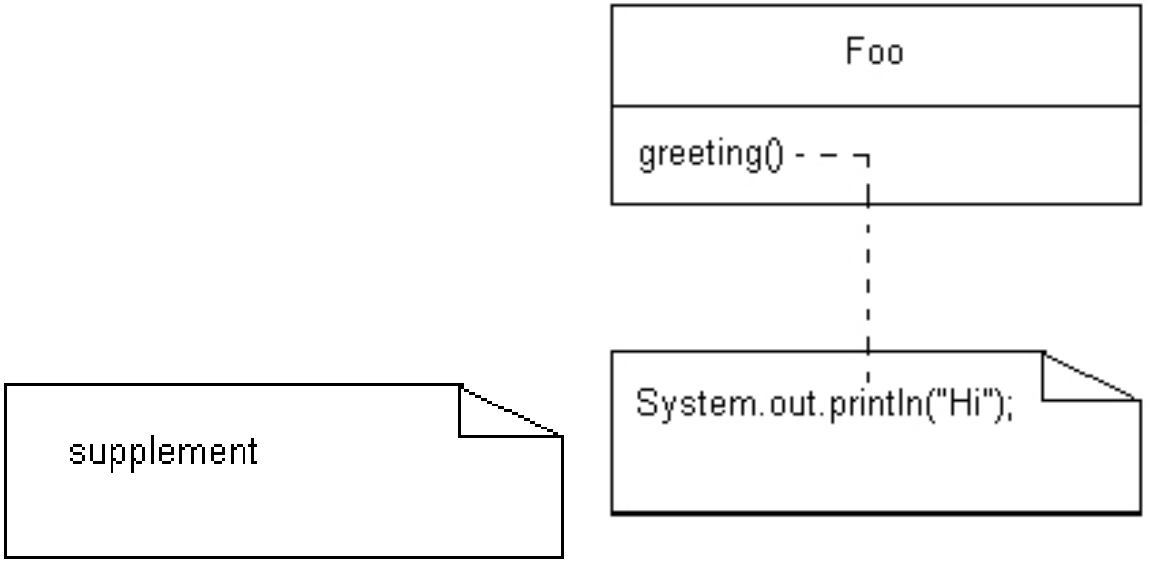
# Visibility of members

- public: + (the member is available from everywhere)
- private: - (the member is available from inside the class)
- protected: # (the member is available from inside the class or its descendants)

# Class template



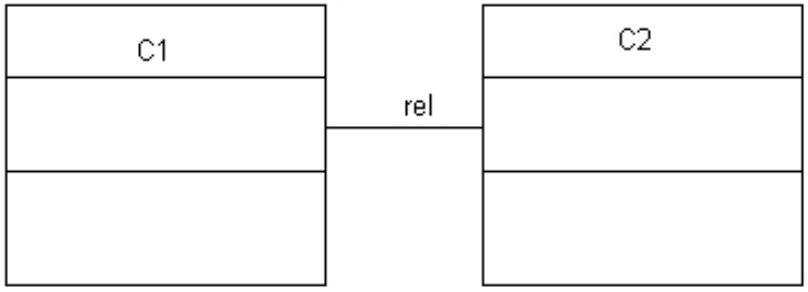
# Annotations



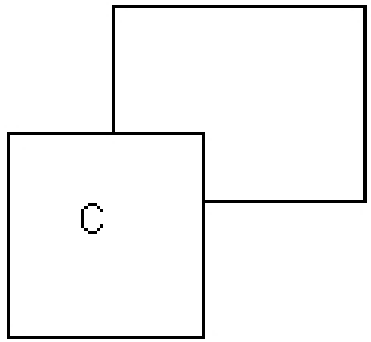
# Relationship - Association

- Basic relationship among classes
- class level: association
- object level: link

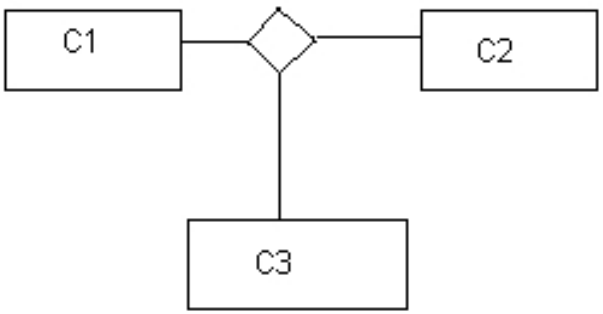
# Relationship - Association



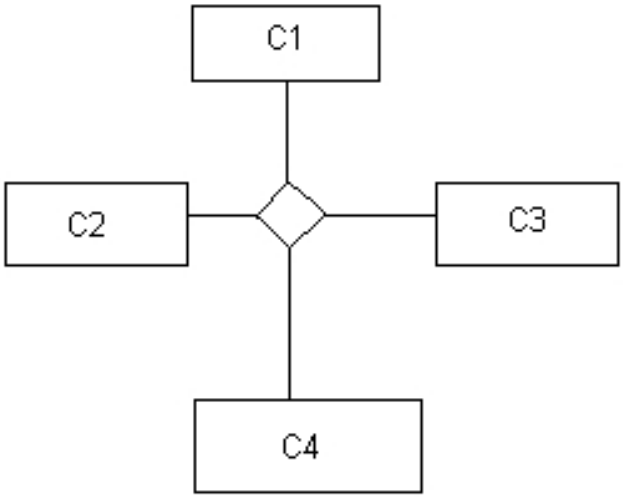
# Reflexive association



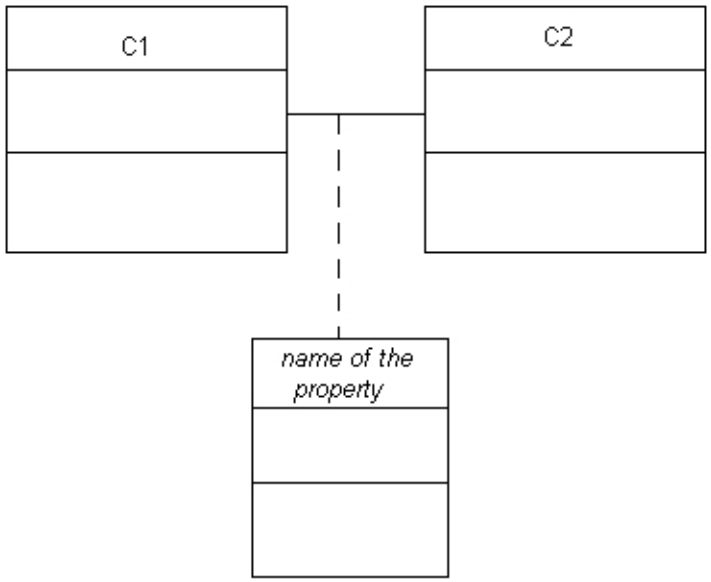
# Association among 3 classes



# Association among 4 classes



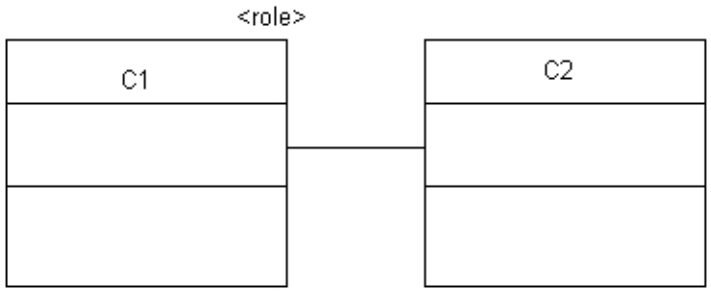
# Association class



# Multiplicity

- More objects of a class may be applied in a relation.
- Default notation: one object per class taking part in the relation.
- Common multiplicities are:
  - 0..1 means no instances, or one instance
  - 1 means exactly one instance
  - 0..* or * means zero or more instances
  - 1..* means one or more instances (at least one)
  - i means i instances
  - i..j means at least i but no more than j instances.

# Roles

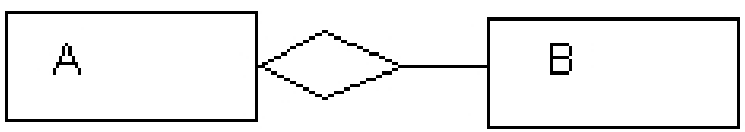


# Relationship - Aggregation

- Aggregation is a special association (stronger than the association).
- expresses “is-a-part-of”

# Relationship - Aggregation

A is an aggregation of B:

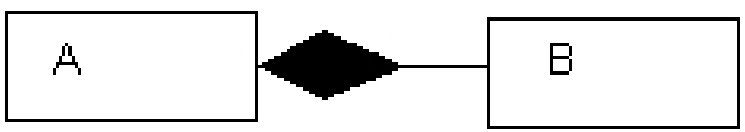


# Relationship - Composition

- Composition is a special aggregation - the strongest aggregational connection between classes
- It expresses that a class contains an other one (physically).
- The components live the same lifecycle

# Relationship - Composition

A is a composition of B:

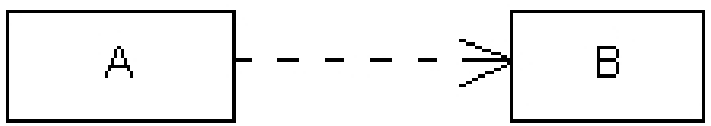


# Difference between Composition and Aggregation

- The whole of a composition must have a multiplicity of 0..1 or 1, indicating that a part must be for only one whole.
- The whole of an aggregation may have any multiplicity.

# Dependency

- A dependency exists between two defined elements if a change to the definition of one would result in a change to the other.



# Generalization (inheritance)

- The generalization relationship indicates that one of the two related classes (the subtype) is considered to be a specialized form of the other (the supertype) and supertype is considered as generalization of subtype. In practice, this means that any instance of the subtype is also an instance of the supertype.
- It expresses “is a” relationship.
- Multiple inheritence is supported by UML.

# Generalization (inheritance)