Class Diagrams in Enterprise Architect

Department of Software

ENTERPRISE ARCHITECT

a class Diagrams

The class diagram is the heart of UML. It is based on the principles of object orientation (abstraction, encapsulation, heredity, etc.) and due to its versatility can be implemented in all phases of a project:

1-in the analysis phase it appears as the domain model and attempts to provide an image of reality.

2- The software is modeled with it in the design phase

3-in the implementation phase source code is generated.

ENTERPRISE ARCHITECT

a class Diagrams

a class diagram in the **Unified Modeling Language (UML)** is **a type of static structure diagram** that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

ENTERPRISE ARCHITECT

a class Diagrams

Purpose of Class Diagrams

1- Shows static structure of classifiers in a system

2- Diagram provides a basic notation for other structure diagrams prescribed by UML

3- Helpful for developers and other team members too

4- Business Analysts can use class diagrams to model systems from a business perspective

ENTERPRISE ARCHITECT

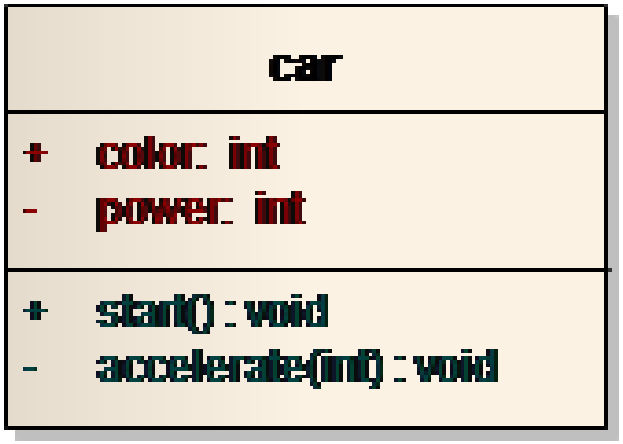
A class describes a number of instances which have the same attributes, constraints and semantics.

The toolbox of class diagram

1-Class

Class is represented by rectangle which either carry only the name of that class and have :

- **a-attribute**
- **b-operation**
- **c-relationship with other classes**

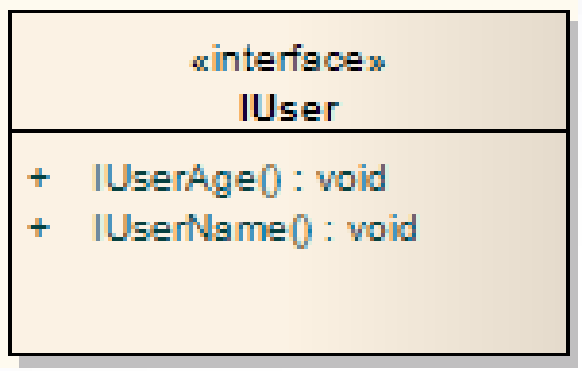


ENTERPRISE ARCHITECT

2- Interface

An Interface is a specification of behavior (or contract) that implementers agree to meet. By implementing an Interface, Classes are guaranteed to support a required behavior, which enables the system to treat non-related elements in the same way; that is, through the common interface. You also use Interfaces in a Composite Structure diagram.

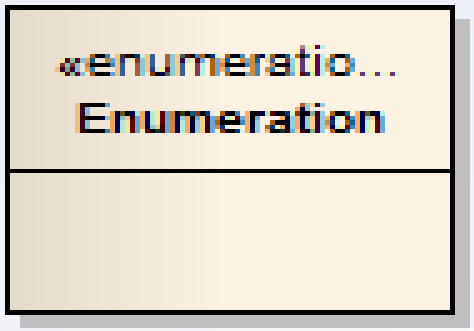
Interfaces are drawn in a similar way to a Class, with operations specified



ENTERPRISE ARCHITECT

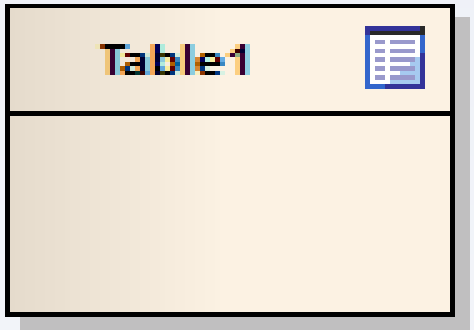
3- Enumeration

An Enumeration is a data type, whose instances can be any of a number of user-defined enumeration literals.



4- Table

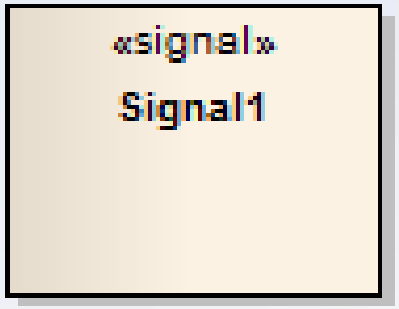
A *Table* is a stereotyped class. It is drawn with a small table icon in the upper right corner. You typically use this element in Data Modeling and Class diagrams



ENTERPRISE ARCHITECT

5- Signal

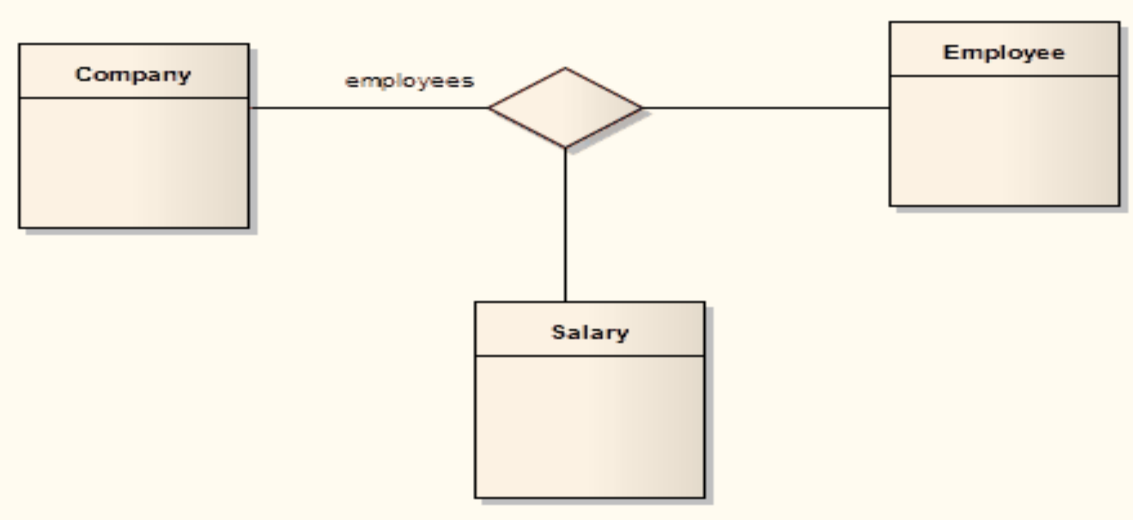
A Signal is a specification of Send request instances communicated between objects, typically in a Class or Package diagram.



ENTERPRISE ARCHITECT

6- Association

An Association element is used to model complex relationships between three or more elements, It is not a commonly-employed device, but can be used to good effect where there is a dependent relationship between several elements. It is generally used with the Associate connector, but the relationships can include other types of connector, typically in a Class diagram



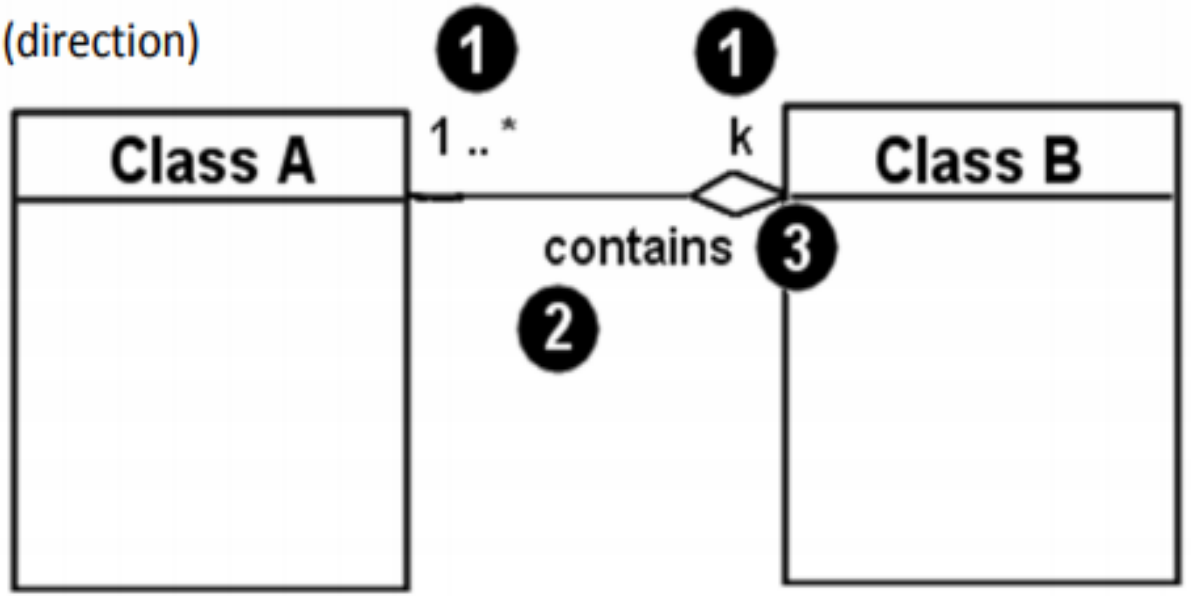
Relationship between classes

*** Association**

An *Association* implies two model elements have a relationship, usually implemented as an instance variable in one Class. This connector can include named roles at each end, multiplicity, direction and constraints. Association is the general relationship type between elements. To connect more than two elements in an association, you can use the N-Ary Association element.

Associational relationships

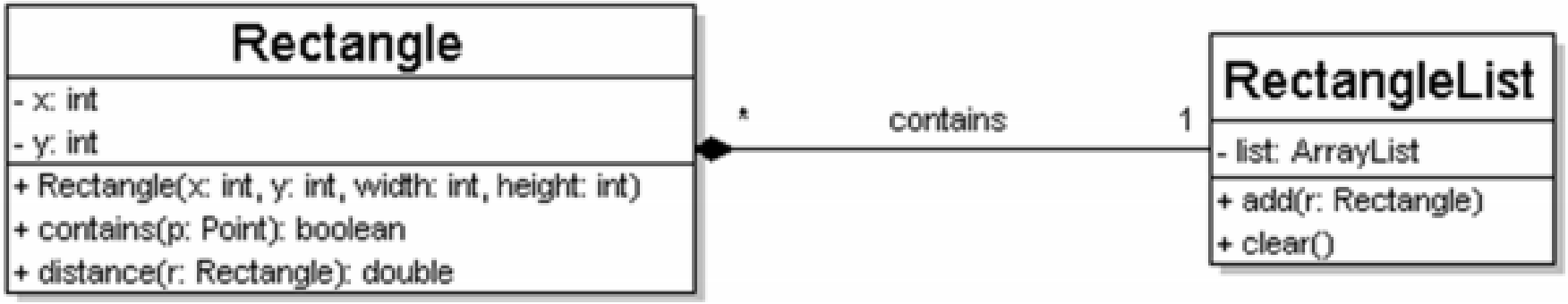
- associational (usage) relationships
  1. multiplicity (how many are used)
     - * ⇒ 0, 1, or more
     - 1 ⇒ 1 exactly
     - 2..4 ⇒ between 2 and 4, inclusive
     - 3..* ⇒ 3 or more (also written as "3..")
  2. name (what relationship the objects have)
  3. navigability (direction)



Multiplicity of associations

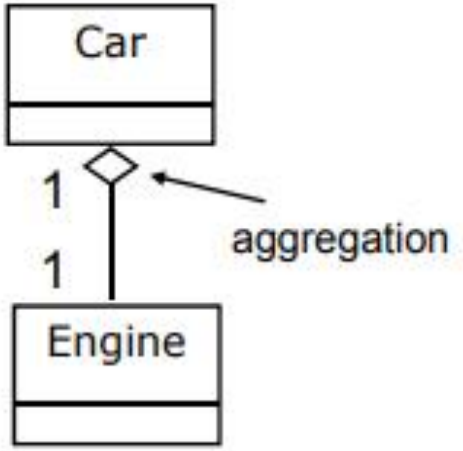
- one-to-one
  - each student must carry exactly one ID card

- one-to-many
  - one rectangle list can contain many rectangles

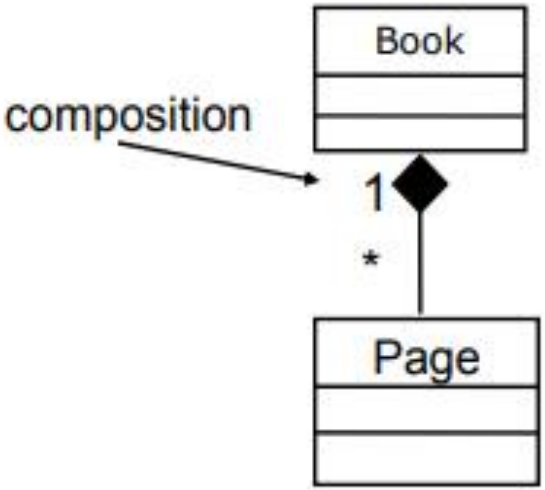


Association types

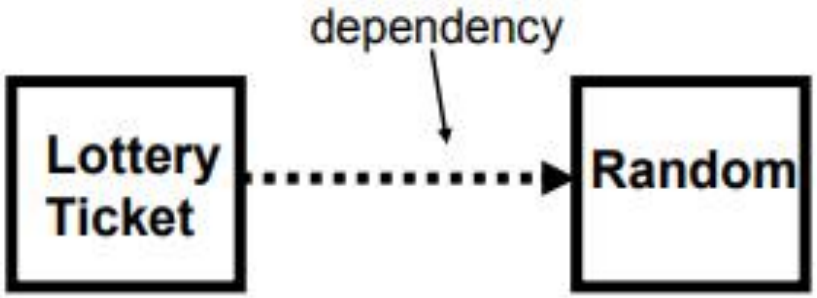
- **aggregation**: "is part of"
  - symbolized by a clear white diamond



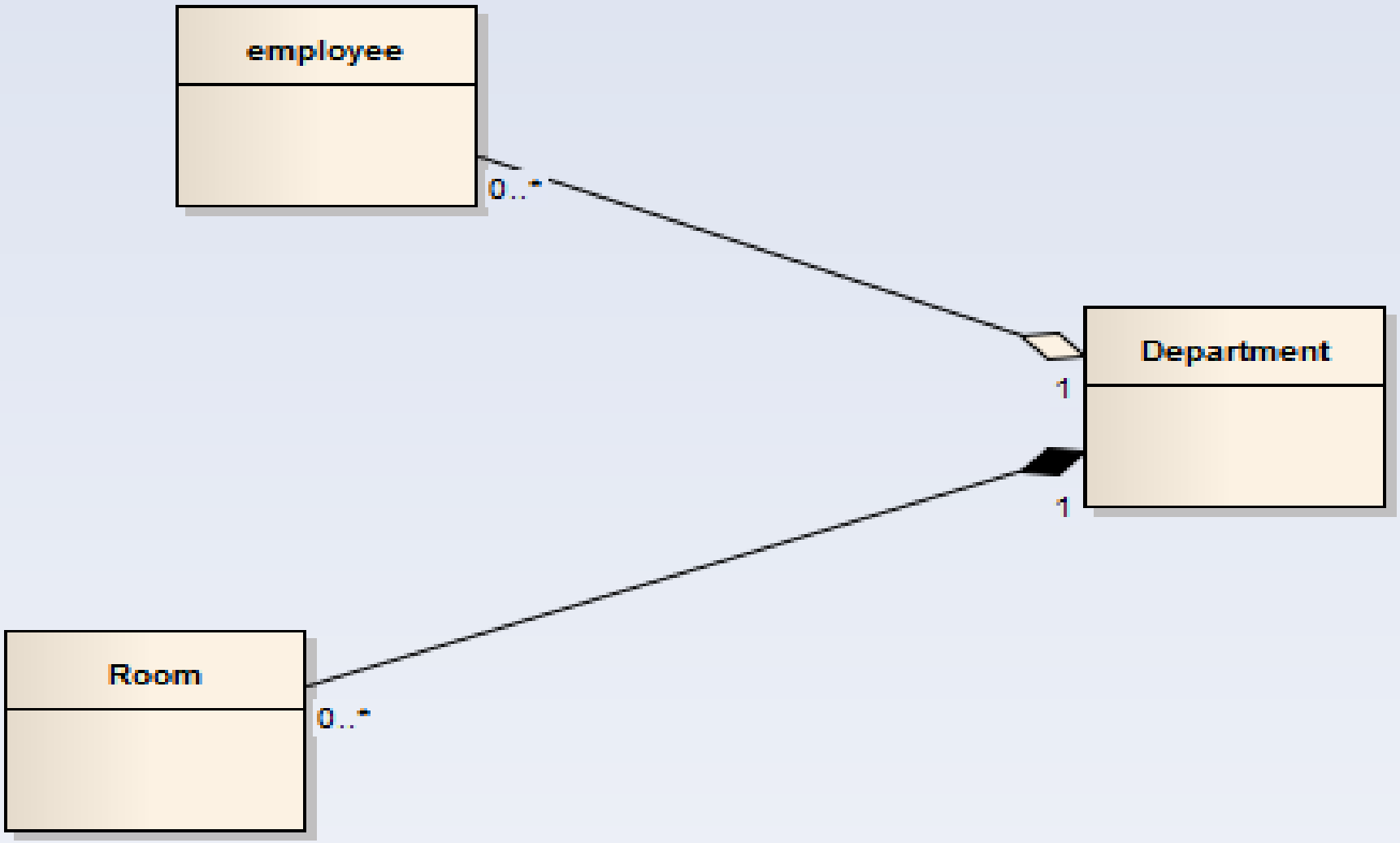
- **composition**: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond



- **dependency**: "uses temporarily"
  - symbolized by dotted line
  - often is an implementation detail, not an intrinsic part of that object's state



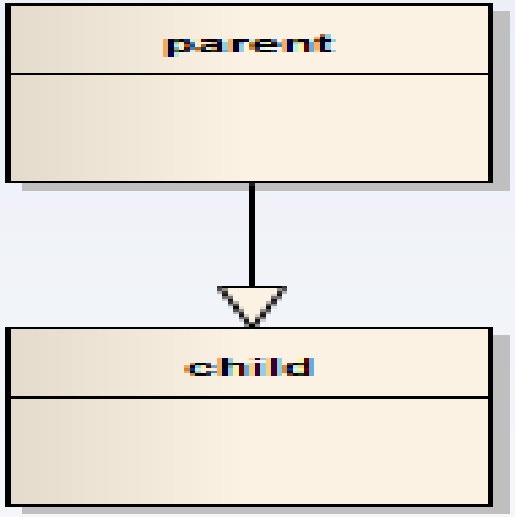
Composition/aggregation example



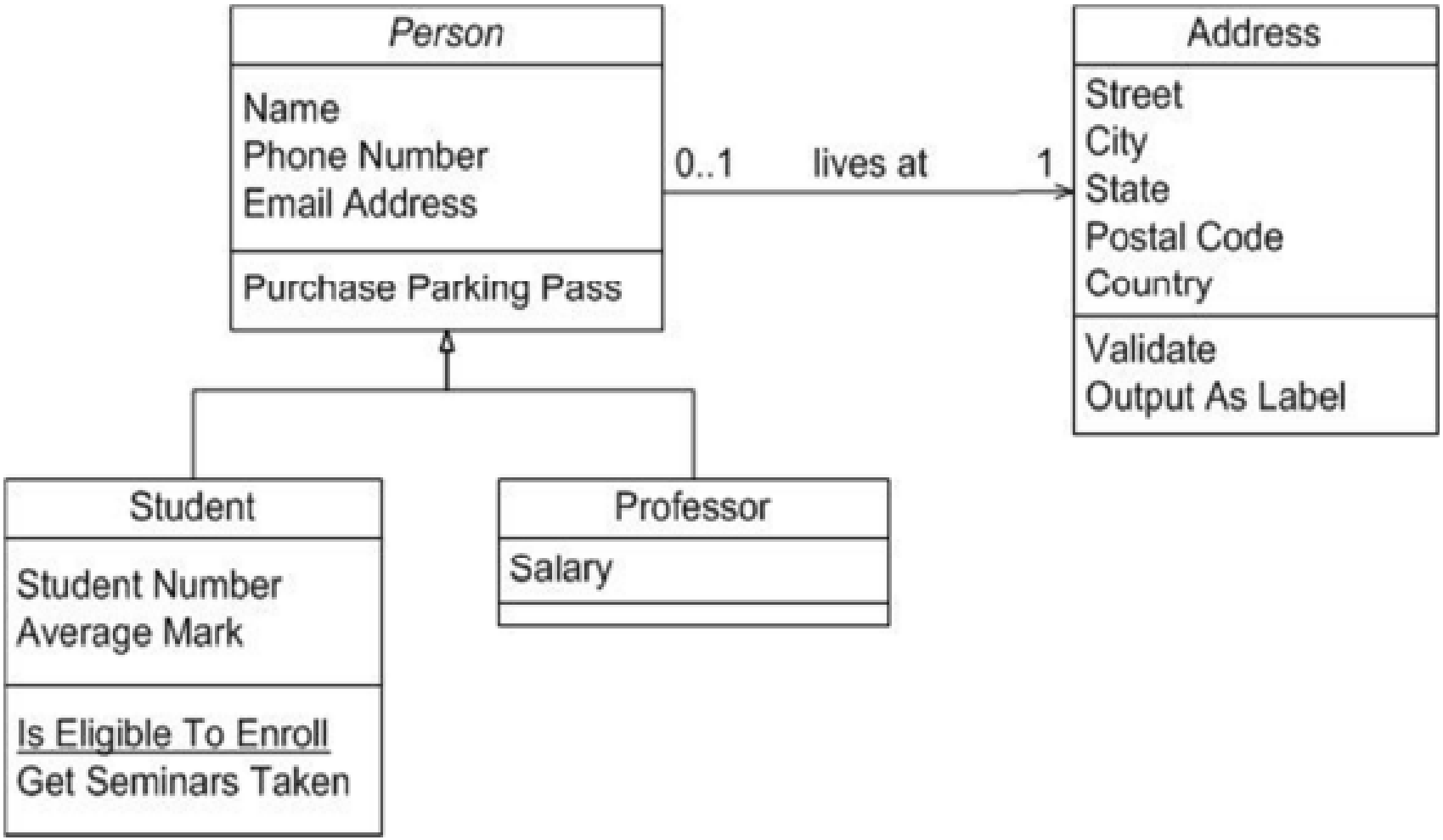
ENTERPRISE ARCHITECT

*** Generalization (inheritance) relationship**

A *Generalization* is used to indicate inheritance. Drawn from the specific classifier to a general classifier, the generalize implication is that the source inherits the target's characteristics. It is used typically in Class, Component, Object, Package, Use Case and Requirements diagrams.



UML example: people



Thank You