UNITE UNIVERSITÀ DEGLI STUDI DI TERAMO

UNITE SCOM

UML Class Diagrams

Prof. Romina Eramo

University of Teramo

Department of Communication Sciences

reramo@unite.it

Class Diagram

» Shows a set of classes, interfaces and their relationships (dependency, association and generalization)

» It can be seen as a graph where the nodes are the classes and interfaces, and the edges are the relationships

» They can also contain packages or subsystems (used to group elements)

» They model the static part of a system

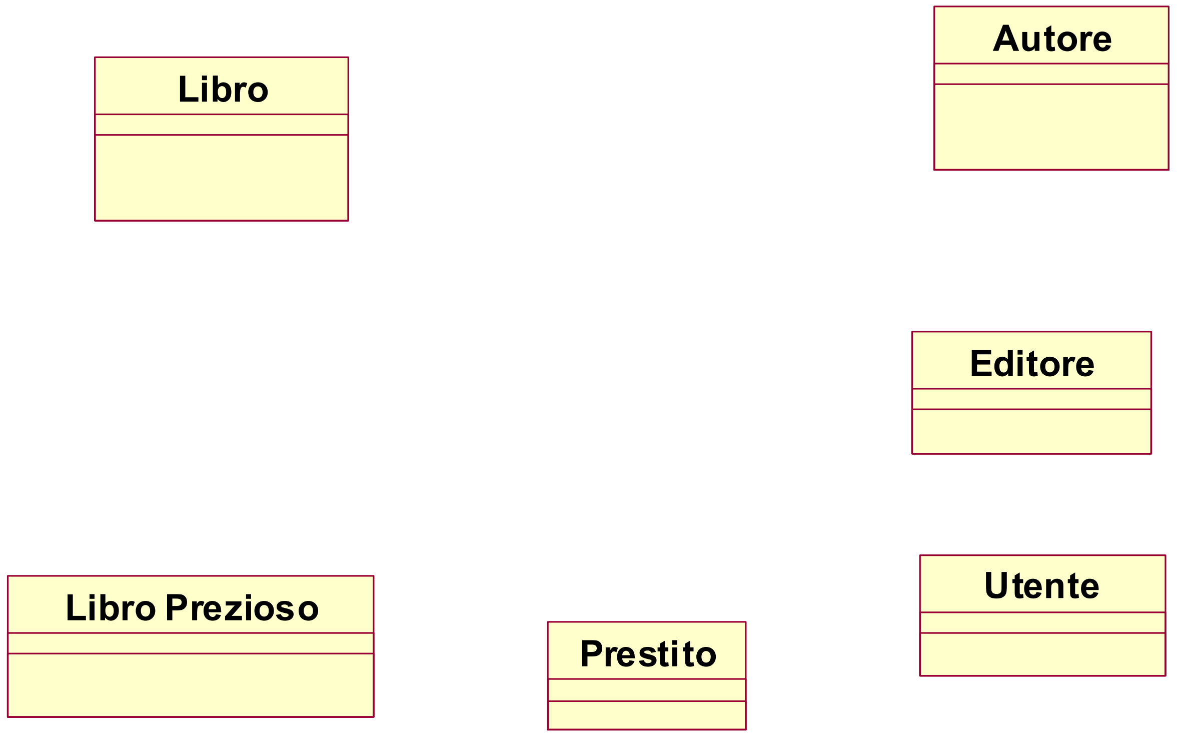
Class Diagram

- Model the vocabulary of a system
  - Classes model abstractions of things of a problem
  - Such abstractions are part of a system's vocabulary
- Model simple collaborations
  - Classes do not live alone
  - They work together to provide behavior that is greater than the sum of all the elements
- Model a logical database schema
  - Instead of ER schemes

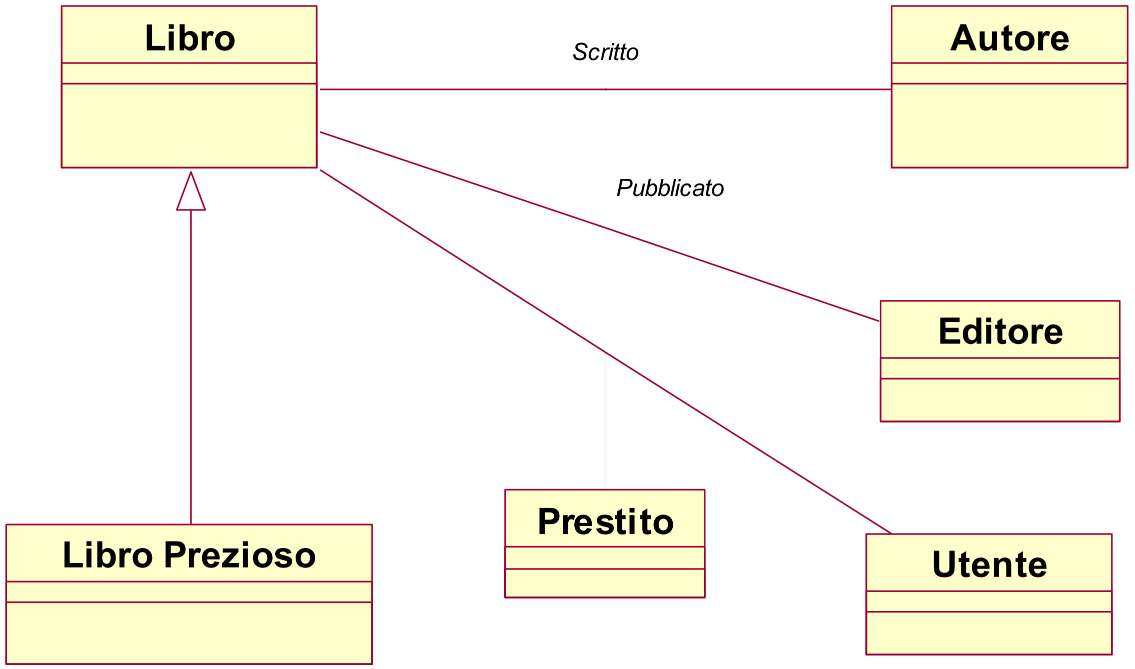
Example

» System for bibliographic archiving of ***texts*** , storing information on the ***publisher*** and ***author*** and identifying what are considered ***books precious*** . You also want to manage a ***loan operation*** by one or more ***users***

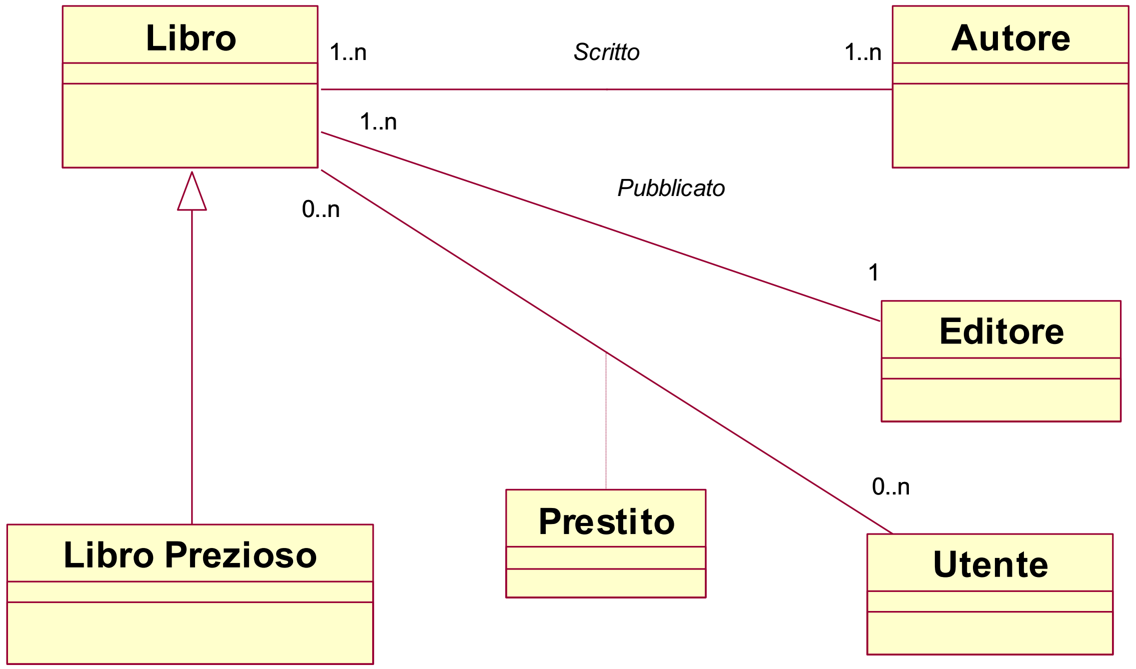
Example



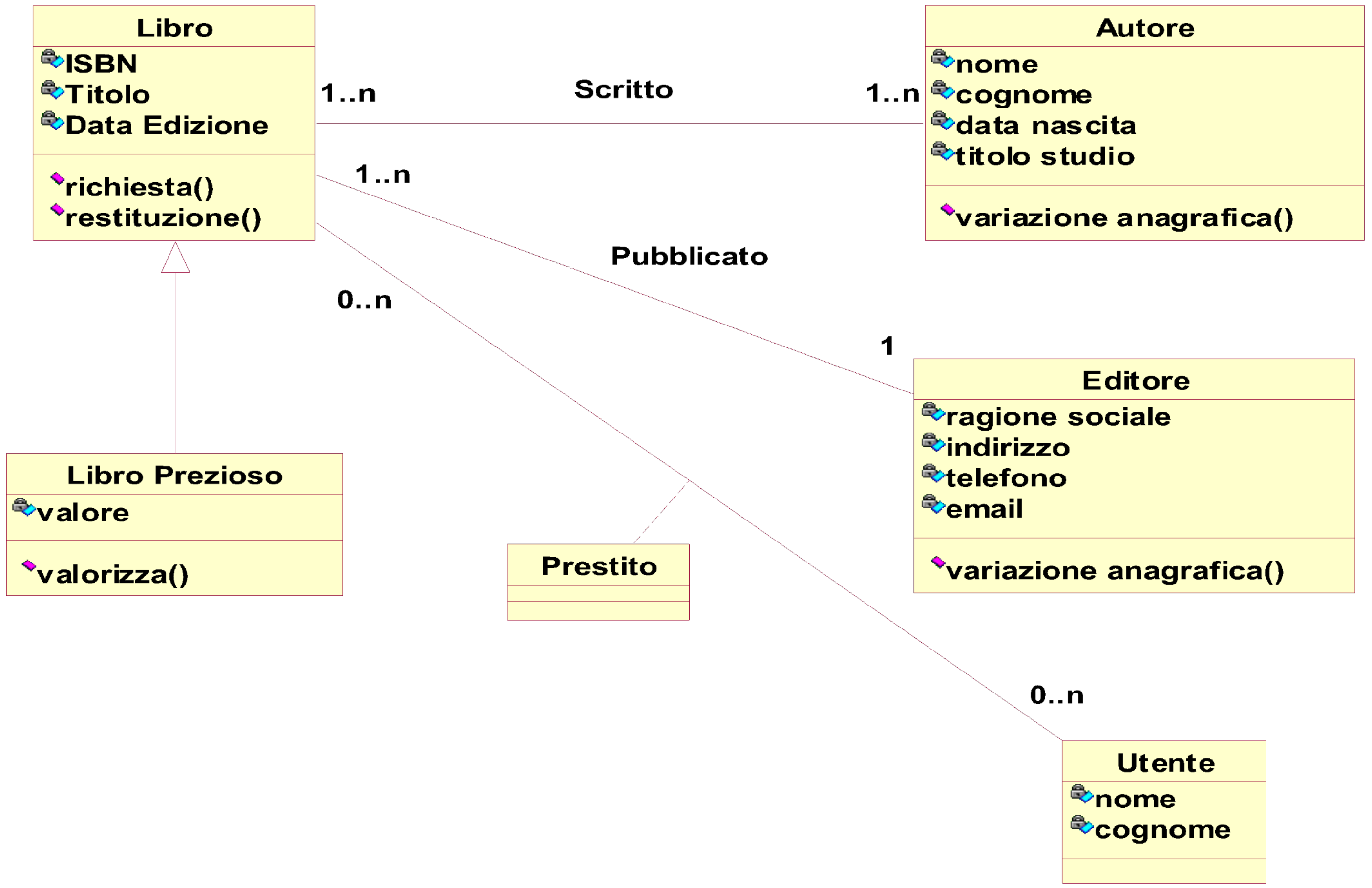
Example


Libro
Autore
Scritto
Pubblicato
Editore
Prestito
Utente
Libro Prezioso

Example

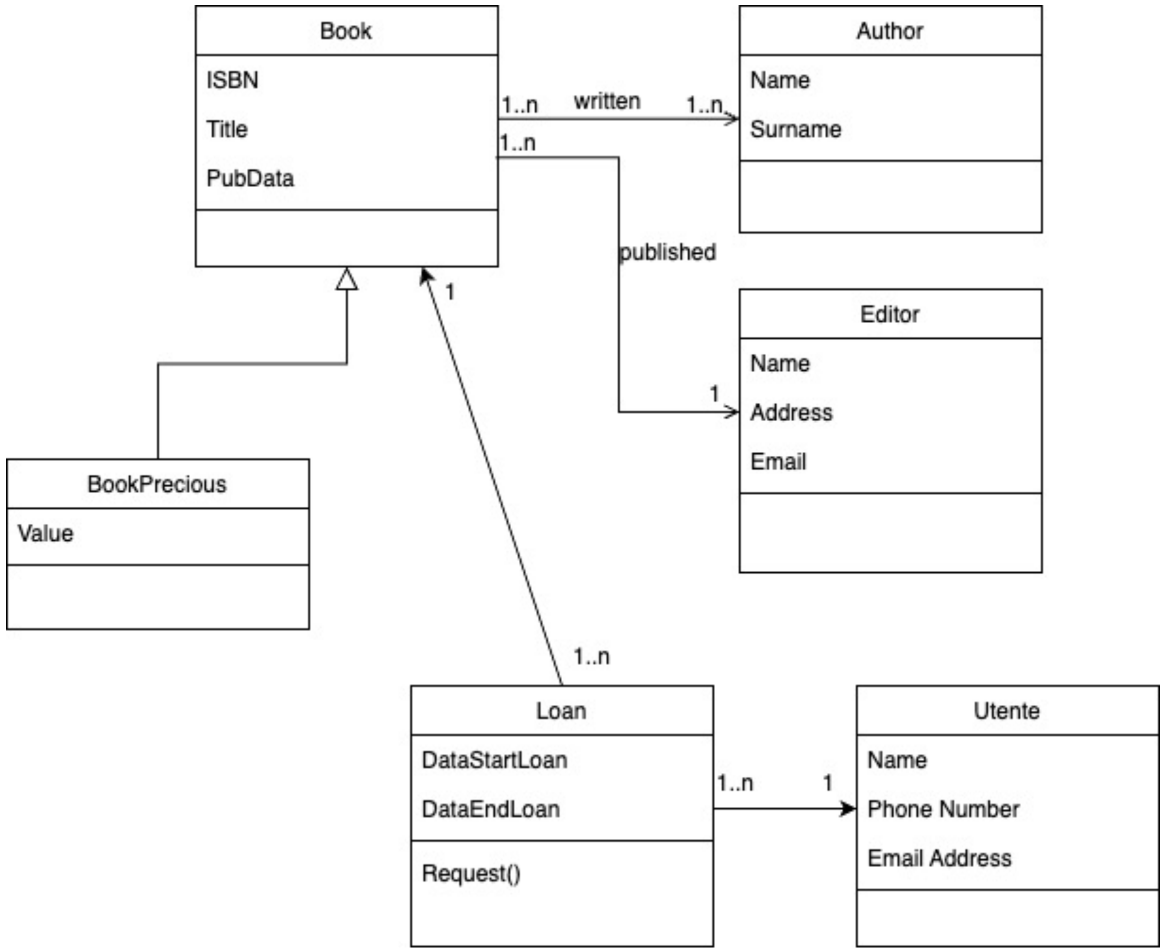

Libro
1..n
Scritto
1..n
Autore
1..n
Pubblicato
0..n
1
Editore
Libro Prezioso
Prestito
0..n
Utente

Example



Libro
ISBN
Titolo
Data Edizione
richiesta()
restituzione()
1..n
Scritto
1..n
Autore
nome
cognome
data nascita
titolo studio
variazione anagrafica()
1..n
Pubblicato
0..n
1
Editore
ragione sociale
indirizzo
telefono
email
variazione anagrafica()
Libro Prezioso
valore
valorizza()
Prestito
0..n
Utente
nome
cognome

Example



Book
ISBN
Title
PubData
Author
Name
Surname
1..n
written
1..n
1..n
published
1
Editor
Name
Address
Email
1
BookPrecious
Value
1..n
Loan
DataStartLoan
DataEndLoan
Request()
1..n
1
Utente
Name
Phone Number
Email Address

Object

» Informally, an object represents a physical, conceptual, or software entity
  - physical entity: tractor
  - conceptual entity: chemical process
  - software entities: list, queue...

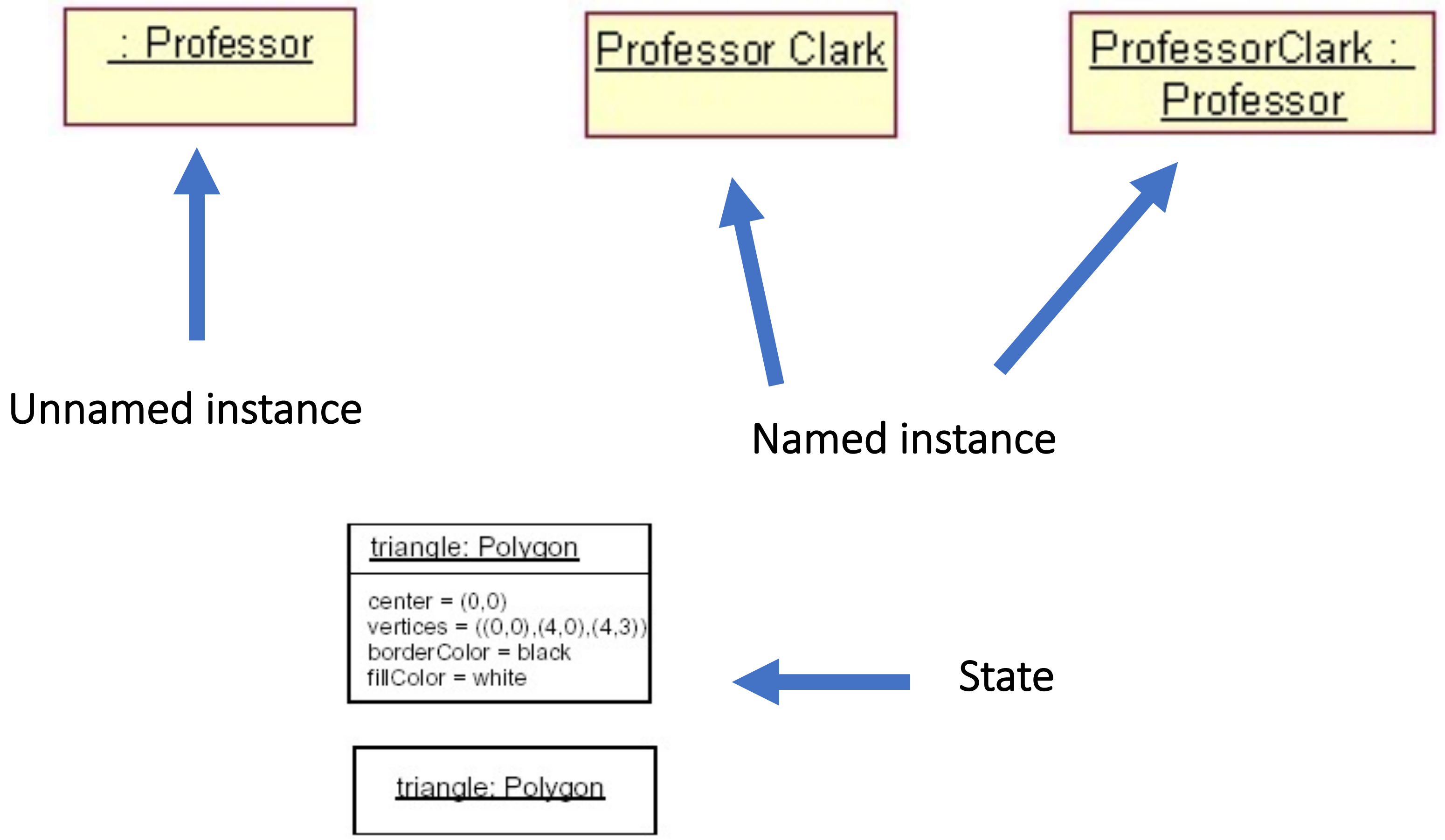
» Formally
  - Concrete manifestation of an abstraction
  - Entity with a well-defined boundary and identity *that* encapsulates *state* and *behavior*
  - Instance of a class

» E.g.: Schumacher's Ferrari, Barrichello's Ferrari, My computer

Object

- State
  - Possible condition in which the object could exist and generally changes over time
  - Implemented using properties (attributes) with values, and links to other objects
- Behavior
  - Determines how an object acts and reacts to requests from another object
  - Represented by the set of messages to which it can respond (operations)
- Identity
  - It makes it possible to distinguish between two objects even if they have the same state and the same value in its attributes

Representation in UML



center = (0,0)
vertices = ((0,0),(4,0),(4,3))
borderColor = black
fillColor = white

Class

» Description of a group of objects with common properties (attributes), behavior (operations), relationships, and semantics
  - An object is an instance of a class

» Abstraction that
  - Emphasize relevant characteristics
  - Suppresses other features

Class example

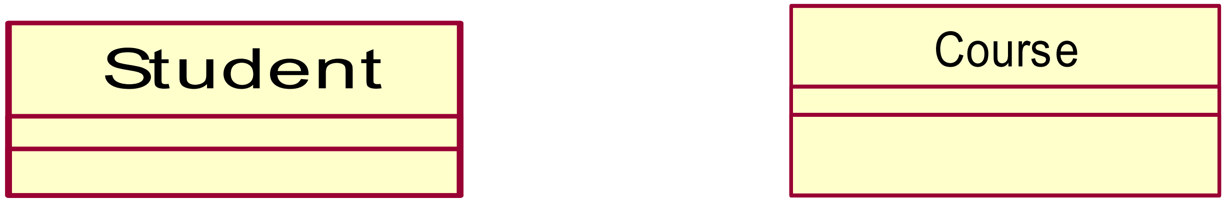
- First name
  - Course
- Property
  - Name, Place, Duration, Credits, Start, End
- Behavior
  - Adding student
  - Student cancellation
  - Check if it is full

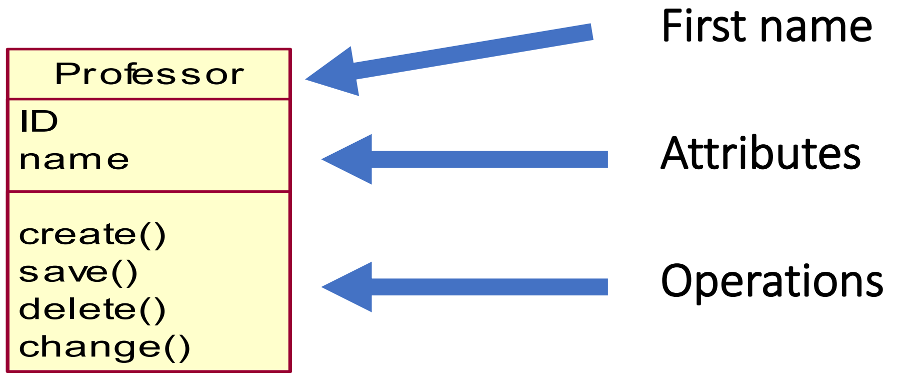
Relationship between class and object

» Class is an abstract definition of an object
  - Defines the structure and behavior of each object in the class
  - It serves as *a template* for creating objects

» Objects are grouped into classes

Representation in UML





Class: Name

- » Represents a name, i.e. an entity
- » Text string
  - letters
  - numbers
  - some special characters
- » First name
  - Simple
  - Path (prefix + ":" + class name)
- » Convention
  - Capital initial letter

Class: Attributes

» Property of a class that describes a set of values that attribute instances can take on

» Represents properties of the things you are modeling that are shared by all objects of that class

» Example

- Wall has a height, width and depth
- Customer has a name, address, telephone number

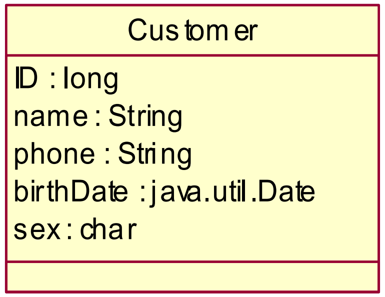
Class: Attributes

» First name
  - Text string

» Guy
  - E.g. int, float, double, String

» Initial value

» Convention
  - Lowercase initial letter



Class: Operations

» Implementation of a service that can be requested by any object

» It is an abstraction of something that is shared among all objects of that class

» Represents a verb or phrase

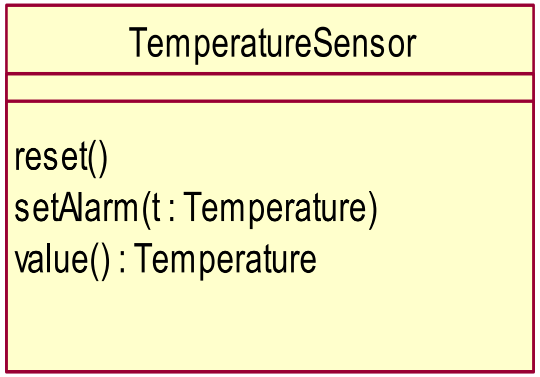
» Generally invoking an operation changes the state of the object

» Examples

   – Rectangle has move, resize operations

Class: Operations

- » First name
  - – Text string
- » Signature
  - – comma separated list of
    - » Default value type name
- » Return type
  - – For the "functions"
- » Convention
  - – Lowercase initial letter



Abstract classes

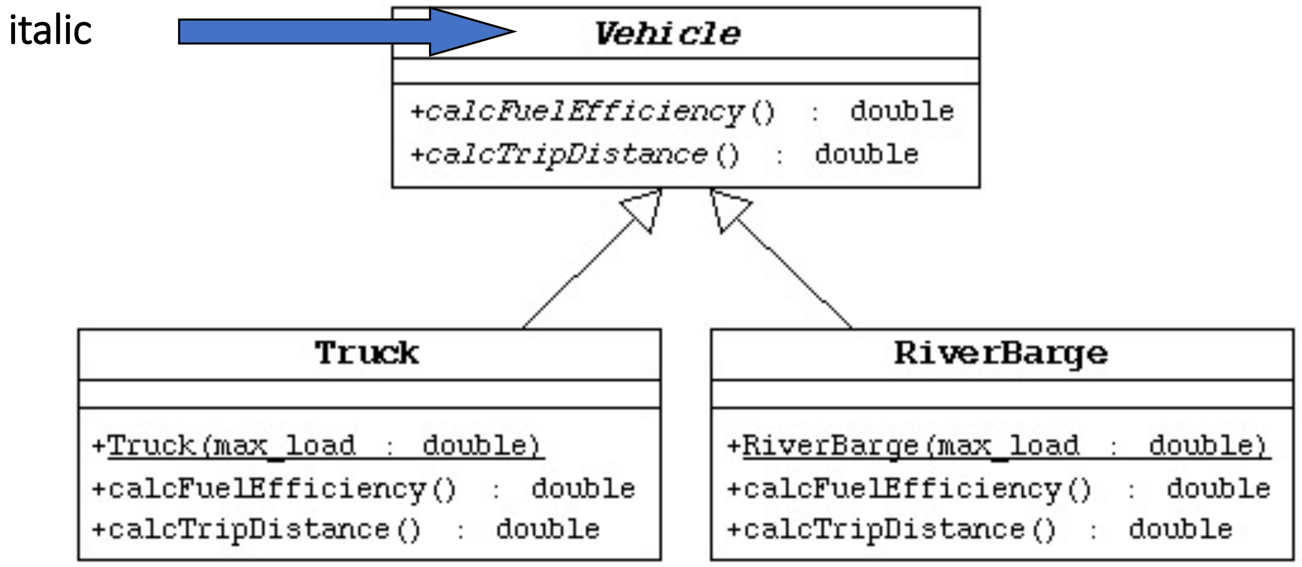
» Class with operations whose body is not defined

» You can declare an abstract class even if it has no abstract operations

» It cannot be instantiated, that is, no instances of that class can exist

» May contain operations that have implementation

Representation in UML


italic
Vehicle
+*calcFuelEfficiency*() : double
+*calcTripDistance*() : double
Truck
+Truck(max_load : double)
+calcFuelEfficiency() : double
+calcTripDistance() : double
RiverBarge
+RiverBarge(max_load : double)
+calcFuelEfficiency() : double
+calcTripDistance() : double

Relations

» A relationship is a connection between things (i.e. classes, interfaces, components, packages)

» A relationship provides a pathway for communication between objects

Association

» Represents a structural relationship between objects of different classes

» Bi-directional connection between classes, you can navigate from one object of one class to another and vice versa

» It is possible to have circular associations, that is, between objects of the same class

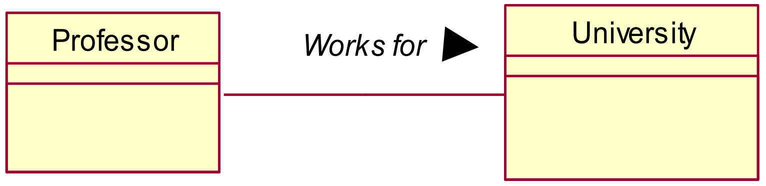
» Association that connects two classes is called binary; n-air connecting n classes (little used)

» Represented by a continuous line connecting the two classes

Association: Name

» Describes the nature of the association

» It is possible to give a direction to the name by means of a triangle pointing to the direction you want the direction to read

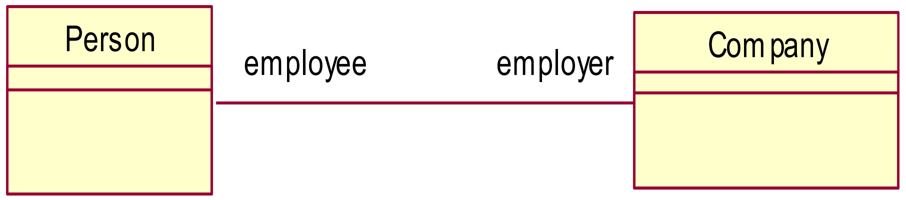


Association: Role

» Specify the appearance that a class plays in the association

» It has a name and is placed next to the class that plays that role in the association compared to the other

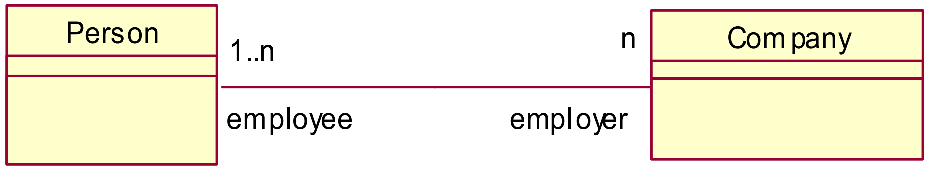
» The use of the role or name is mutually exclusive



Association: Multiplicity

- » Defines how many objects participate in a relationship
  - – Specifies the number of instances of a class that is related to ONE instance of the other class

- » Applied at the end of each association



Association: Multiplicity

Value	Description
No	Unlimited number of instances
1 (default)	Only one instance
0..n	Zero or more instances
1..n	One or more instances
0..1	Zero or one instance
<literal>*	Exact number of instances
<literal>..n	Exact number or more instances
<literal>..<literal>	Specified range of instances
<literal>..<literal>, <literal>	Range plus specified number of instances
<literal>..<literal>, <literal>..<literal>	The number of instances will be in one of the specified ranges
* Where <literal> is an integer greater than or equal to 1	

UNITE UNIVERSITÀ DEGLI STUDI DI TERAMO

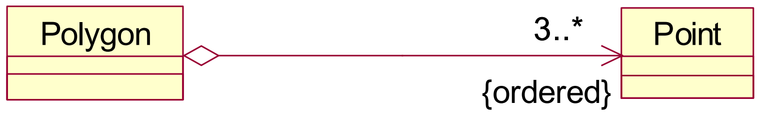
UNITE SCOM

Association: Multiplicity

» If Multiplicity greater than 1 then the set of associated elements may or may not be ordered

» Specified by constraint at the end of the association
  - **unordered** : elements form an unordered set (default)
  - **ordered** : elements they have an ordering and i duplicates are not allowed ({ordered})

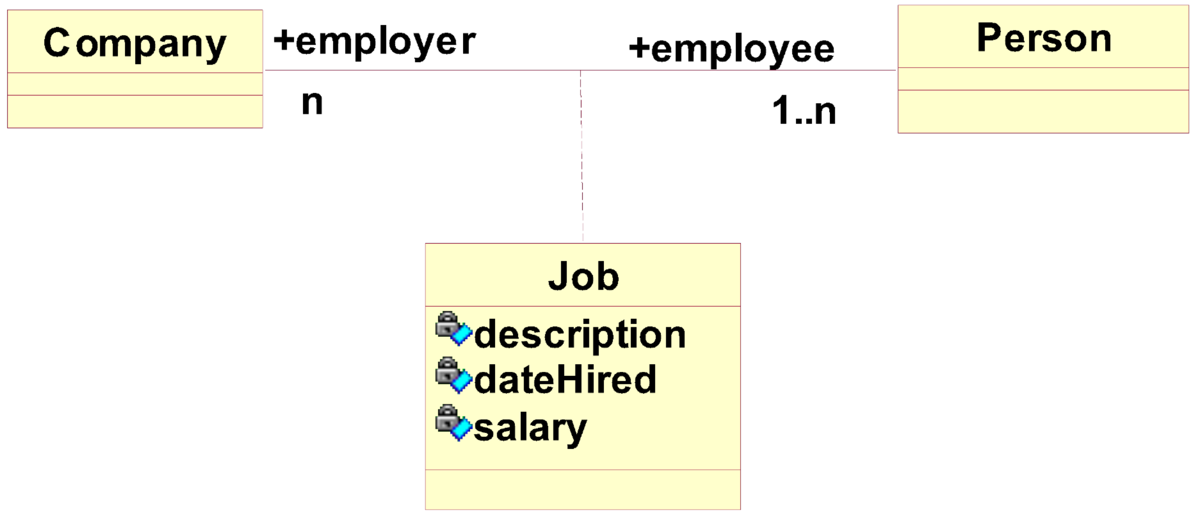
» Ordered relationship is specified by generating code dependent on the implementation language



Association: Association Class

» An association can own _ beyond at the multiplicity , roles and visibility Also from the property structural and behavioral

» Example



Association: Aggregation

- » Association between classes shows a *peer-to-peer* structural relationship
  - It is not possible to distinguish a class that is conceptually more important than the others (they are all at the same level)

- » There is a need to model situations in which a class expresses a notion that is conceptually greater than the others that constitute it

- » Represents a relationship *has -a* , *consists -of* , *contains* , *is -part-of*

Association: Aggregation

- It is necessary to use relationships of the "whole-part" type (*whole -part*), in which there is a class that represents the "larger" concept (the whole) made up of the remaining ones that represent the smaller concepts (the parts)
- Such relationships are called **Aggregation**
- Represented with a line that connects the related objects using a diamond placed next to the complete class

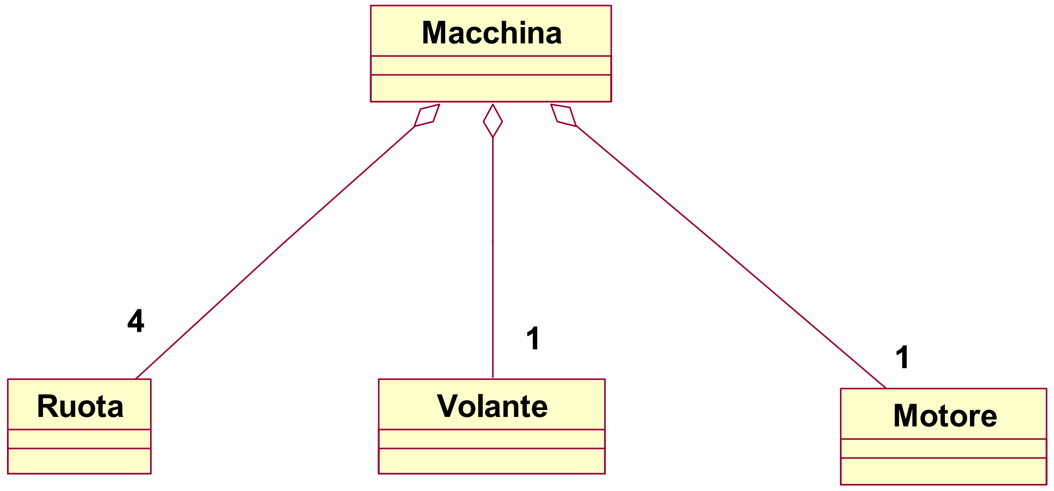
Association: Aggregation

» Example
  - Car made up of wheels, engine, steering wheel, seats, ...

» Difference compared to purely conceptual association

» Circular aggregations make no sense
  - Class A composed of a Class B; B composed of a Class C which in turn is composed of Class A

Association: Aggregation



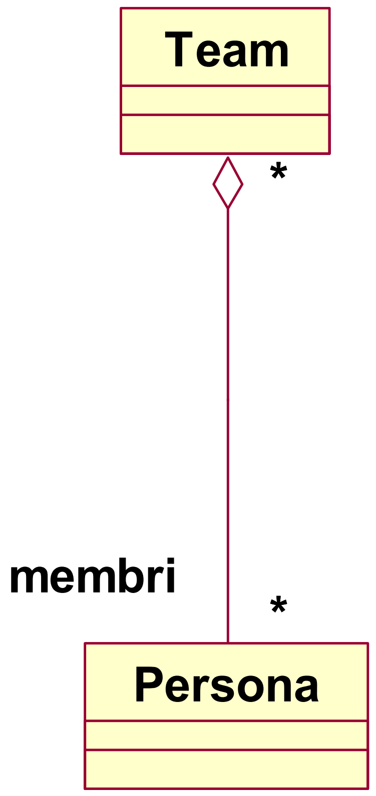
Association: Shared Aggregation

» Special case of normal aggregation

» The part class can be part of any integer

» It is shared if the multiplicity in the integer part is greater than one

Association: Shared Aggregation



Association: Composition

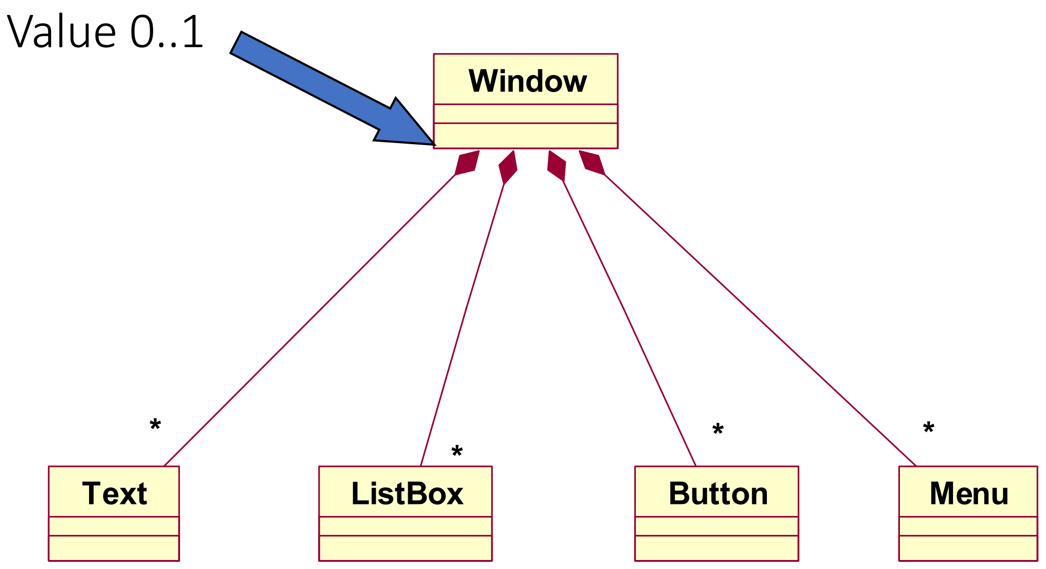
» Form of aggregation with a strong connotation of possession and an (almost) coincidence of the life cycle between instances from the classes “ part ” and the instance from the class “ everything ” (class composed)

» Set off can also be generated at a later time at the creation of the instance from the class composed , but one once these are generated they live and are destroyed with the instance from the class composed of membership

Association: Composition

- » Composite class takes care of eliminating the instances of its parts at a time prior to its destruction
- » An object can only be part of one composite object at a time
  - – Example: A *Frame* belongs to only one *Window*
- » However, in an aggregation a part can be shared between multiple compounds
  - – Example: A *Wall* can belong to multiple *Rooms*

Association: Composition



Association vs. Dependence

» Association is a structural relationship (therefore "persistent"), which highlights semantically related classes

» Dependence has a transitory character, a bond that is established (or at least so it should be) temporarily, for the period of time necessary to use a service, to create an object, etc., and then loses meaning

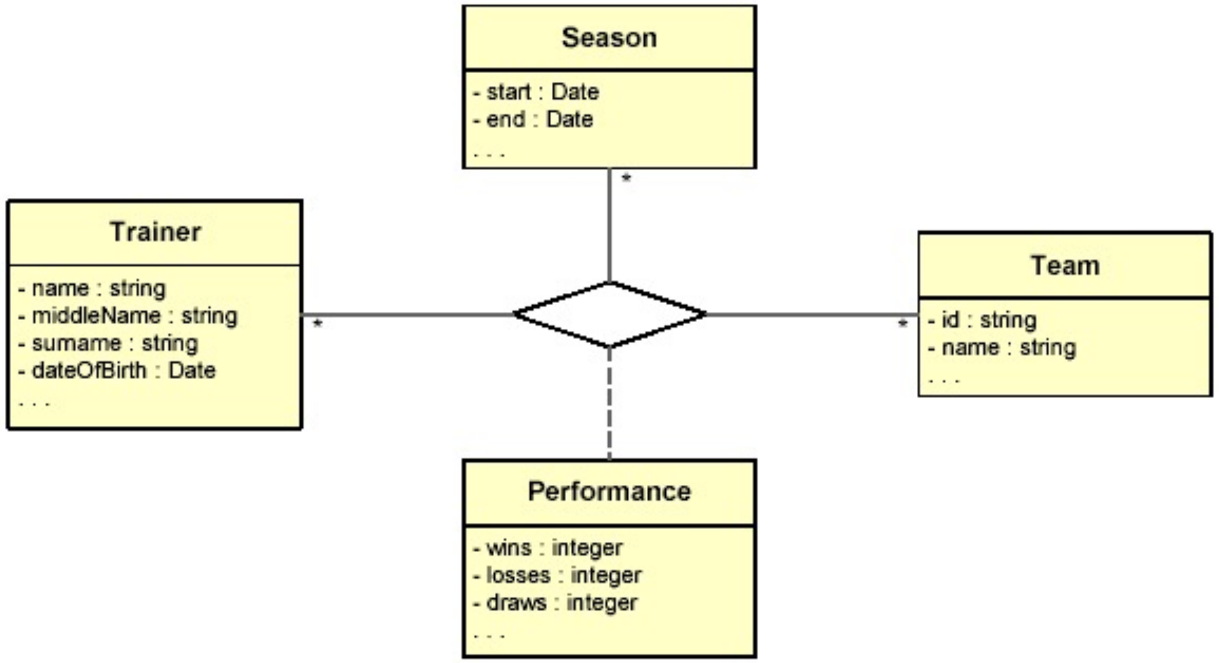
n-air association

» Relationship involving more than two classes

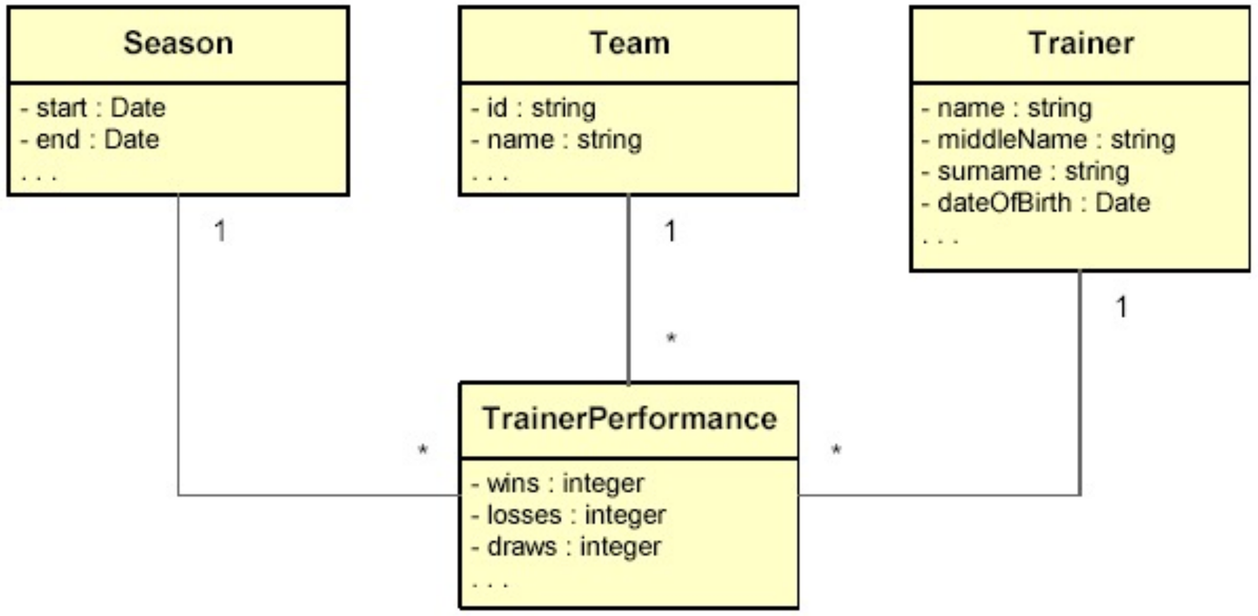
» Represented by a rhombus where the classes belonging to the relationship are connected

» Difficulty in attributing the values of multiplicities. They specify, for each class, the potential number of instances of the class that can participate in the relationship, fixing the values of the other *n–1* classes

n-air association



n-air association



Transformation of an n-ary association into binary associations

Generalization

- It's a relation Between a What more general (said superclass or parent) ed a more specific (said subclass or daughter)
- He comes said Also “ *is-a-kind-of* ” relationship
- Objects son can to be used in place of parent objects (Liskov substitutability principle) but not the vice versa , that is the father is not a substitute for the son

Generalization

- The child inherits all the properties of its fathers, i.e. attributes and operations
- Sometimes the child can override the parent's operations
- Relation
  - Transitive: if C generalizes (i.e. inherits) B and B inherits from A, it follows that C also generalizes A
  - Antisymmetric: if A inherits from B, the opposite is absolutely not true. If this is true then A and B are equal

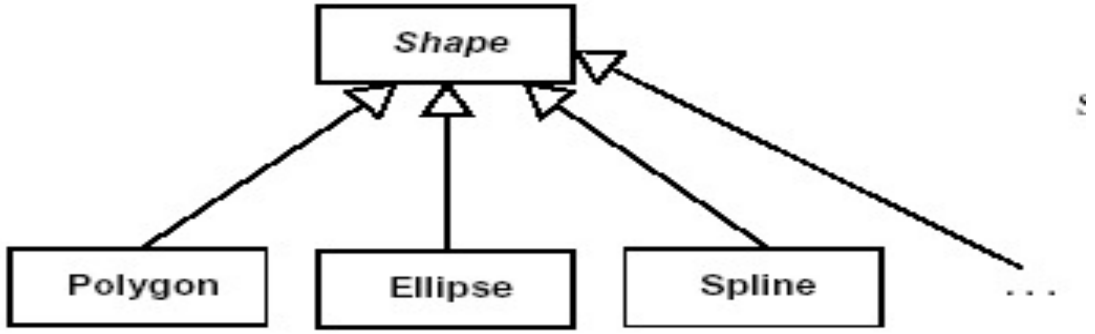
Generalization

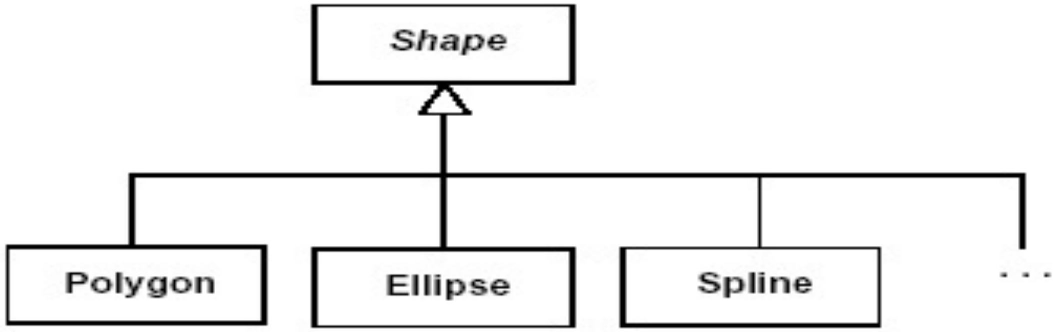
» Drawn with a closed arrow pointing towards the father

» Types of generalization

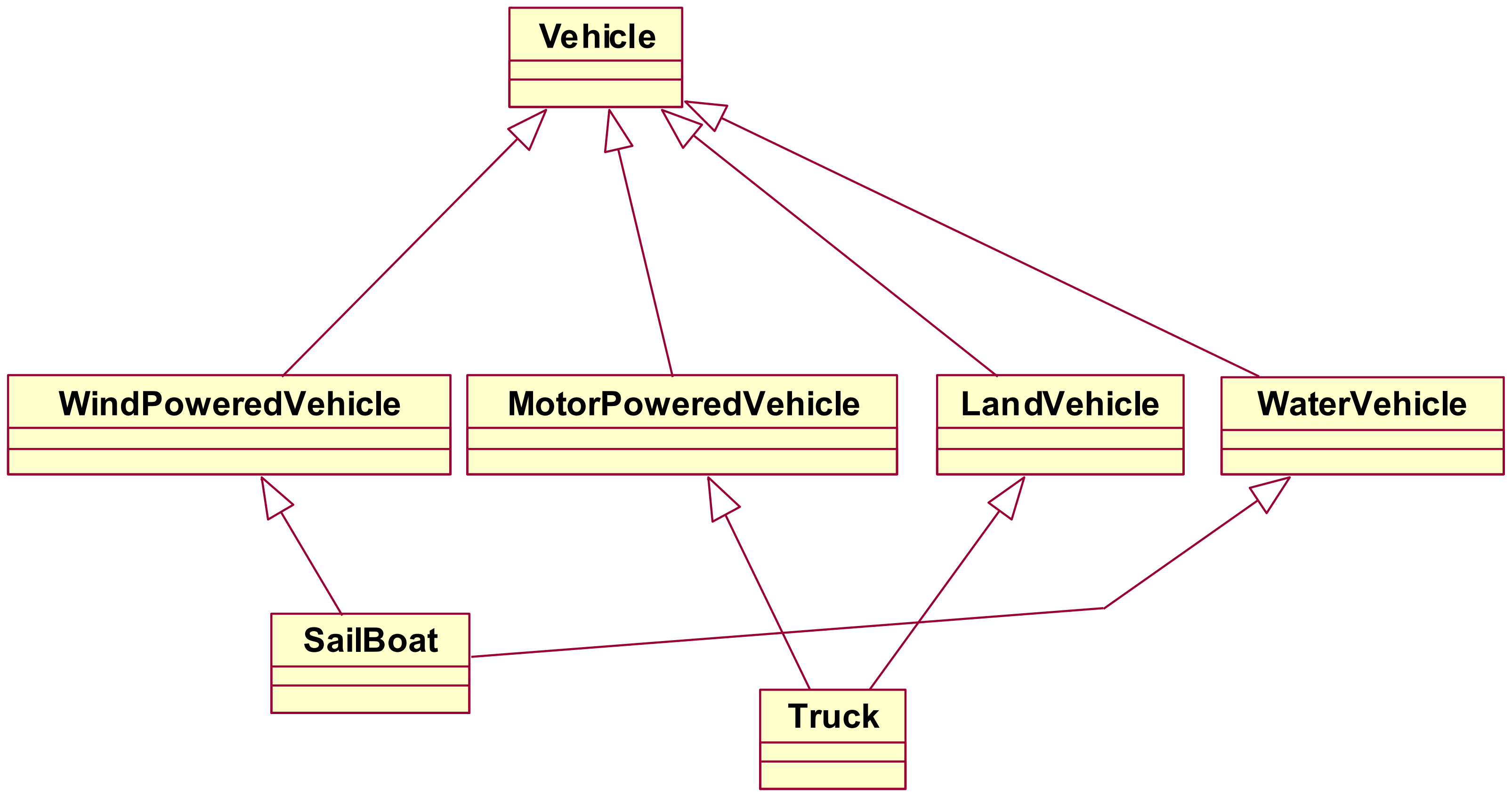
- Single (Java)
  - » You have only one father
- Multiple (C++ and NO Java)
  - » It is possible to have multiple fathers

Generalization





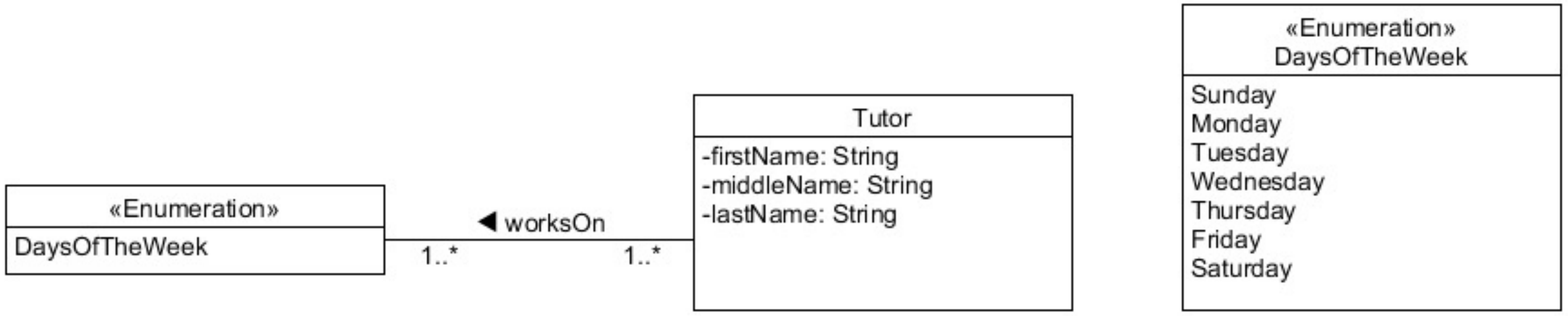
Generalization



Enumeration

» Enumerations are model elements in class diagrams that represent user-defined data types

» Enumerations contain sets of named identifiers that represent the values of the enumeration.



Identifier

» A sequence of unrestricted length of *letters* and *digits*

» They are used for *class* names, *attributes* names, *operations* names, and are *case-sensitive*

» Keywords and true, false, and null literals do not identify

» Examples

- Product
- product
- $userName

Literal (1)

» Represents the constant value that each primitive type (either String or null) can take

» Types

- Int
- Floating
- Boolean: `true` o `false`
- Char
- String
- `null`

Literal: Int (2)

- Type is `int`
- When used as a suffix `L` or `l` becomes `long`
- Expressed as
  - decimal (base 10): number
  - octal (base 8): `0`number
  - hexadecimal (base 16): `0x`number
- Examples
  - Decimal `int`: `10`
  - Decimal `long`: `10L`
  - Octal `int`: `010`
  - Hexadecimal `int`: `0x10`

Literal: Int (3)

- » Largest decimal literal type `int` is `2147483648` (2^{31})
- » From 0 to `2147483647` a literal `int` can appear and be used anywhere you can use a `int`
- » `2147483648` can only appear as a negative number
- » Largest decimal literal of type `long` is `9223372036854775808L` (2^{63})

Literal: Int (4)

- Largest **positive** type literal `int` hexadecimal and octal are `0x7fffffff` and `017777777777` respectively (`2147483647`)
- Largest **negative** type literal `int` hexadecimal and octal are `0x80000000` and `020000000000` that represent `-2147483648`
- `0xffffffff` and `037777777777` represent -1 in hexadecimal and octal respectively

Literal: Floating (5)

- Composed of
  - Integer Part
  - Decimal point (.)
  - Fractional Part
  - Exponent: `E` or `e` followed by a signed integer
  - Suffix
    - `F` or `f` `float`
    - `D` or `d` `double` (default)
- Examples
  - Double: `1e1, 2., .3, 0.0, 3.14`
  - Float: `1e1f, 2.f, .3f, 0f, 3.14f`

Literal : Floating (6)

» The larger the positive float literal it is
`3.40282347e+38f`

» Literal positive finite float not zero the smaller it is
`1.40239846e-45f`

» The larger the double positive literal it is
`1.79769313486231570e+308`

» Literal positive double finite not zero smaller is
`4.94065645841246544e-324`

Literal : Character (7)

» Expressed as a character or *escape* sequence enclosed in single quotesTipo è sempre `char`

» Examples

- `‘c’`
- `‘\\’`
- `‘\n’, ‘\t’, ‘\b’, ‘\r’, ‘\’’, ‘\”’`

Literal : String (8)

» Expressed as a sequence of zero or more characters enclosed in double quotes

» Each character can be escaped

» Examples

- `“welcome”`
- `“\””`
- `“\n”`
- `“ciao” + “ciao”`

Data Types

» It allows you to express the nature of the data

» Indicates how the datum can be represented and interpreted

» The same sequence can represent an integer or a character

» Determines the range of values that a datum can take

» Specifies possible operations on the data

Types

» Each type of data has
  - A name
    » `int, double, char`
  - A set of possible literals
    » `3, 3.1, ‘c’`
  - A set of lawful operations
    » `+, *, /, %,` ……

» UML considers
  - **Primitive Types**
  - **Reference Types**

Primitive Types

- Logical
  - `boolean`
- Numeric
  - Integral
    - `byte, short, int, long` e `char`
  - Floating
    - `double e float`

Primitive Types: Boolean

- » Value `boolean` represents a condition of truth or falsehood
- » An attribute typed `boolean` can represent a two-state value
  - – like a switch that is on or off
- » `true` and `false` are the only allowed values
- » Example
  - `- boolean b = false;`

Primitive types: Char

» Characters

» It is represented by the encoding scheme Unicode

Primitive Types: Char

Characters	Name	Value Unicode
`\b`	Backspace	`\u0008`
`\t`	Tabulazione	`\u0009`
`\n`	Avanzamento riga	`\u000a`
`\r`	Invio a capo	`\u000d`
`\″`	Doppi apici	`\u0022`
`\′`	Apici singoli	`\u0027`
`\\`	Backslash	`\u005c`

Primitivi type: byte, short, int e long

» Representation is made by means of two-complement notation

Type	Dimension	Range
`byte`	`1 byte`	`-128 to 127`
`short`	`2 byte`	`-32768 to 32767`
`int`	`4 byte`	`-2147483648 to 2147483647`
`long`	`8 byte`	`-9223372036854775808 to 9223372036854775807`

Primitive guys: float e double

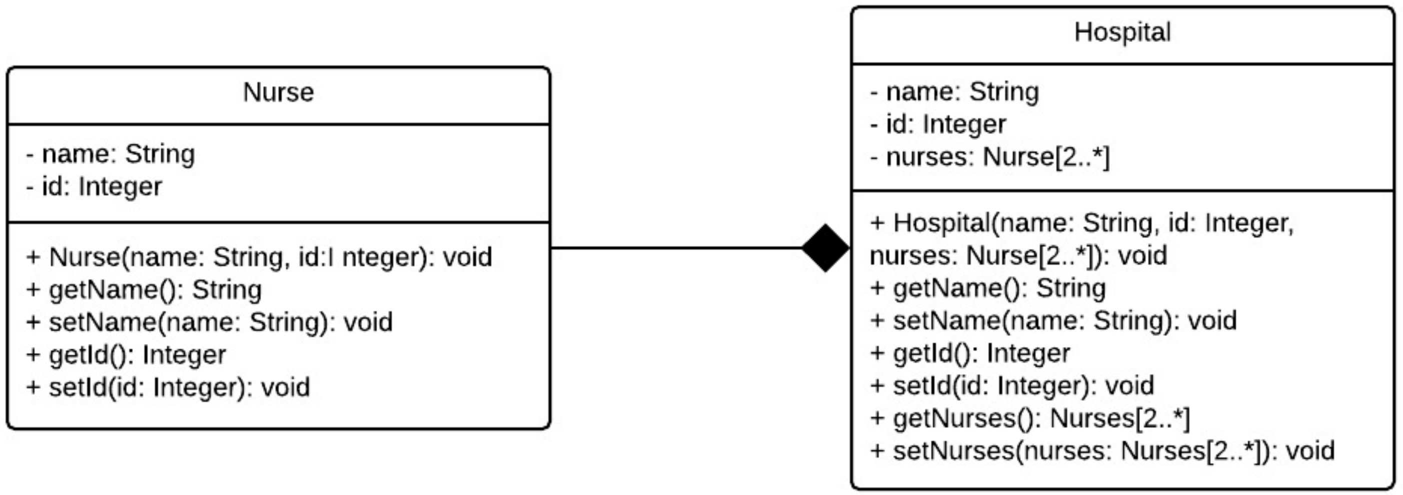
» Define numbers with fractional parts

» Doubles are called double-precision numbers

Type	Dimension	Range
`float`	`4 byte`	`±3,40282347E+38F` (6 or 7 significant decimal places)
`double`	`8 byte`	`±1,79769313486231570E+308` (15 significant decimal places)

Type Reference

» They are pointers (references) to Objects of a Class



Nurse
- name: String
- id: Integer
+ Nurse(name: String, id:I nteger): void
+ getName(): String
+ setName(name: String): void
+ getId(): Integer
+ setId(id: Integer): void
Hospital
- name: String
- id: Integer
- nurses: Nurse[2..*]
+ Hospital(name: String, id: Integer, nurses: Nurse[2..*]): void
+ getName(): String
+ setName(name: String): void
+ getId(): Integer
+ setId(id: Integer): void
+ getNurses(): Nurses[2..*]
+ setNurses(nurses: Nurses[2..*]): void