Introduction to UML and Class modeling

Anastasija Nikiforova

Institute of Computer Science

Based on materials developed by Marlon Dumas, Fabrizio Maria Maggi, Luciano García-Bañuelos

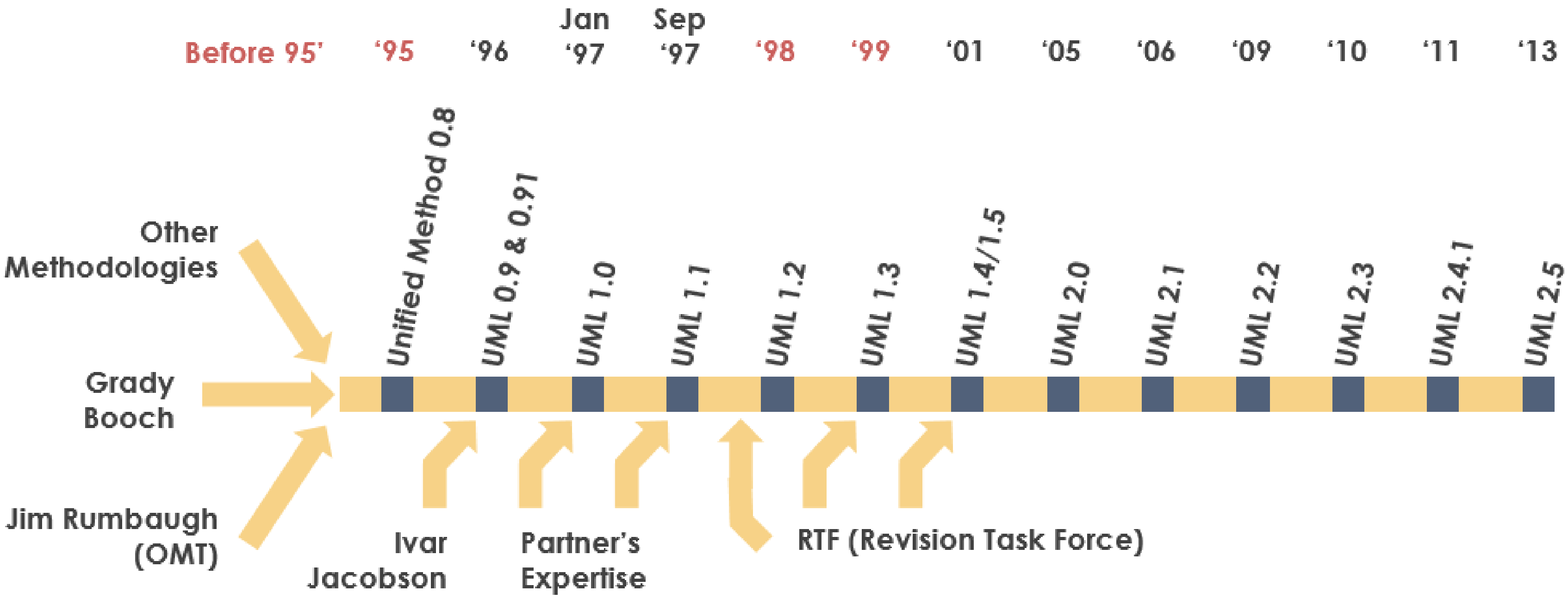
UML

- **Unified Modeling Language** (UML) is a **standardized, general-purpose modeling language**.
- UML includes a set of graphic notation techniques to create visual models of **object-oriented** (OO) software-intensive systems.
- It is considered to be created to forge a **common visual language** in the **complex world of software development** that would also **be understandable for business users and anyone who wants to understand a system**.

UML

- A standardized modeling language consisting of an **integrated set of diagrams**, developed to help system and software developers **for specifying, visualizing, constructing, and documenting the artifacts of software systems**, as well as **for business modeling** and **other non-software systems**. It helps project teams communicate, explore potential designs, and validate the architectural design of the software.

- The UML represents **a collection of best engineering practices** that have proven successful in the **modeling of large and complex systems**.



Before 95’ ‘95 ‘96 Jan ‘97 Sep ‘97 ‘98 ‘99 ‘01 ‘05 ‘06 ‘09 ‘10 ‘11 ‘13

Unified Method 0.8

UML 0.9 & 0.91

UML 1.0

UML 1.1

UML 1.2

UML 1.3

UML 1.4/1.5

UML 2.0

UML 2.1

UML 2.2

UML 2.3

UML 2.4.1

UML 2.5

Other Methodologies

Grady Booch

Jim Rumbaugh (OMT)

Ivar Jacobson

Partner’s Expertise

RTF (Revision Task Force)

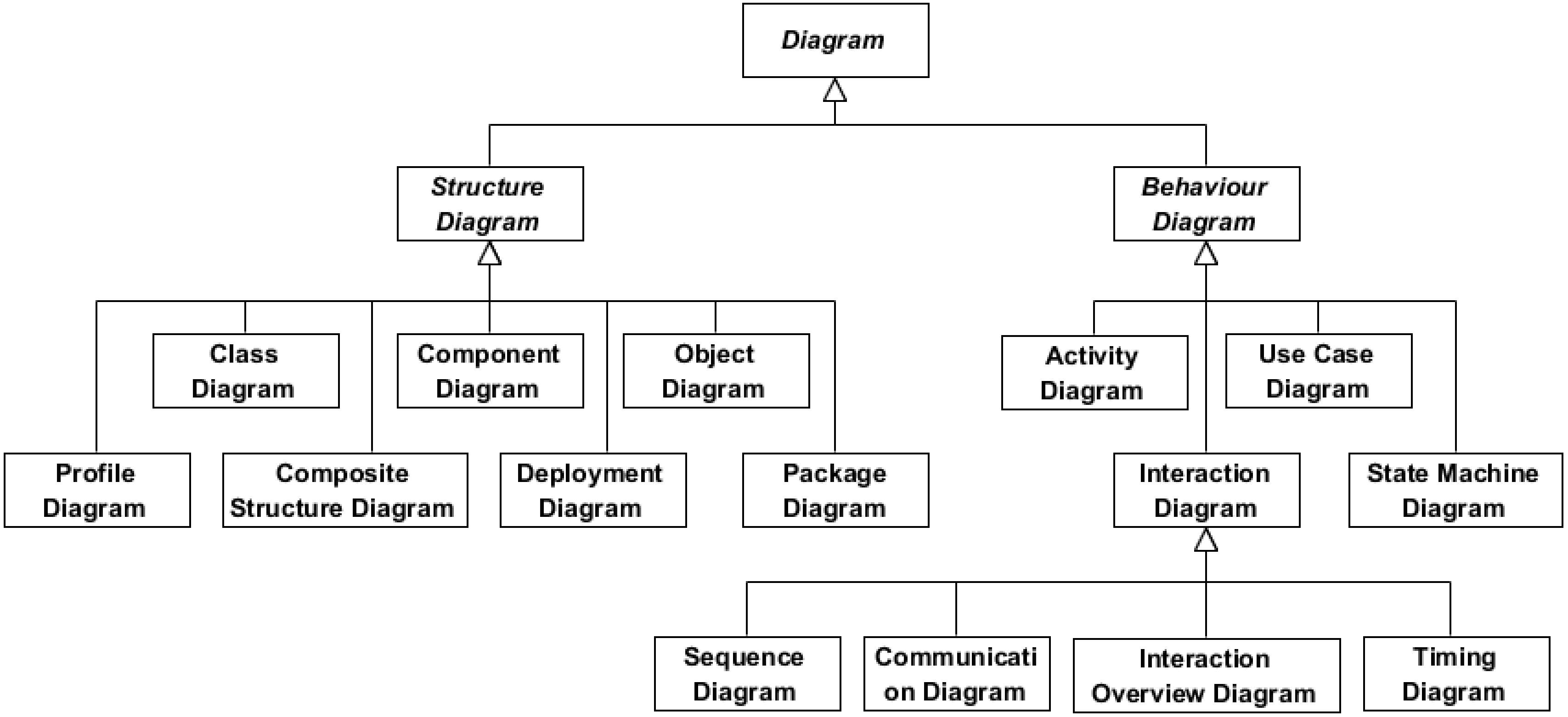
Before 95’ - Fragmentation ▶ 95’ - Unification ▶ 98’ - Standardization ▶ 99’ - Industrialization

Source: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

Why UML?

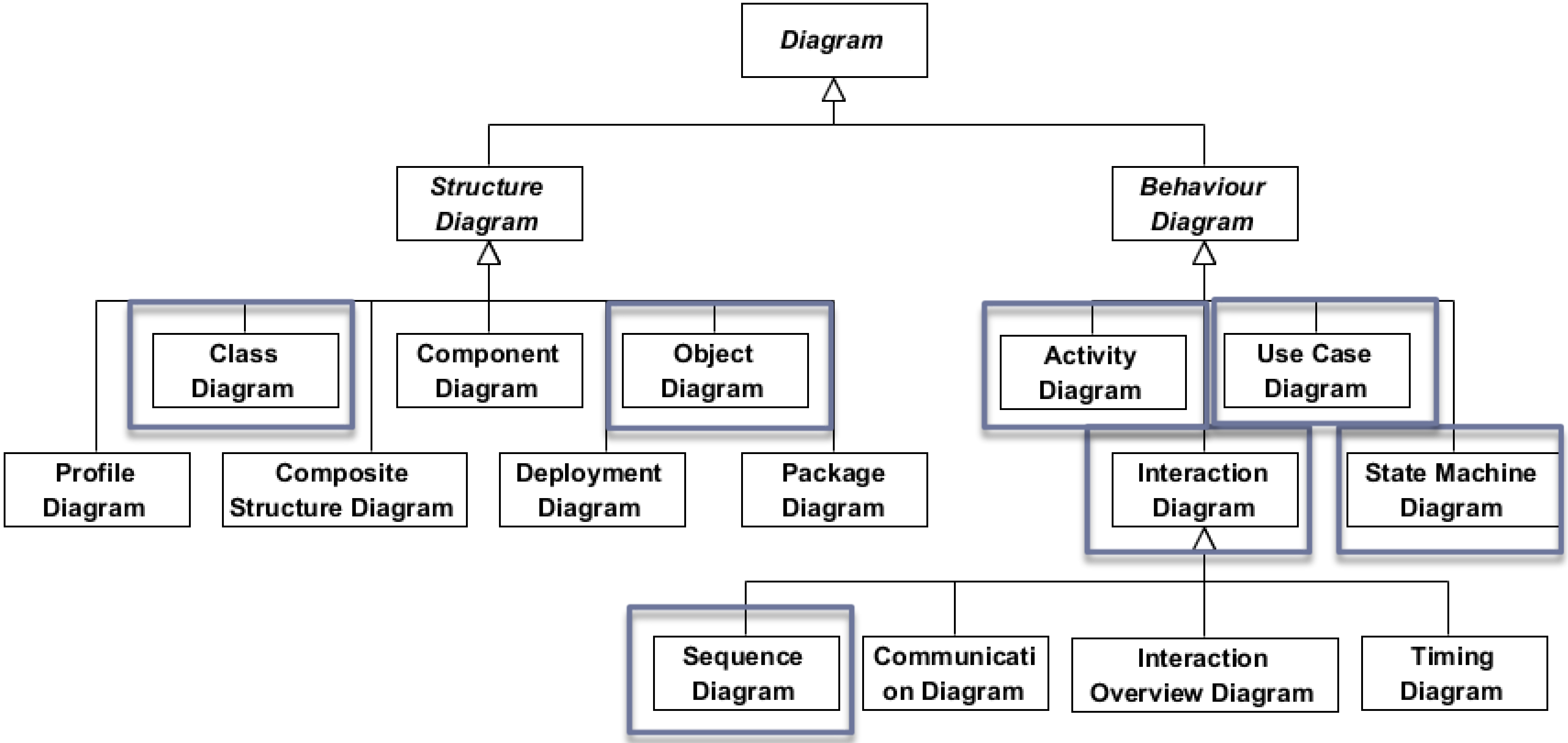
- The strategic value of software increases for many companies → the industry looks for techniques **to automate the production of software**, **to improve quality** and **reduce cost and time-to-market**. These techniques include component technology, visual programming, patterns and frameworks.
- Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale → there is need to solve recurring architectural problems, such as **physical distribution**, **concurrency**, **replication**, **security**, **load balancing** and **fault tolerance etc.**.
- Unified Modeling Language (UML) was designed to respond to these needs

But is it a silver bullet?



Source: https://www.tekportal.net/unified-modeling-language/

Should you use all of them within a project?



Source: https://www.tekportal.net/unified-modeling-language/

Stakeholders and viewpoints

- There are a lot of different models / diagrams to get used to.

- The reason for this is that it is possible to look at a system from many different viewpoints - a software development have many stakeholders playing a part, including:

  - Analysts
  - Designers
  - Coders
  - Testers
  - QA
  - The Customer
  - Technical Authors

These people are interested in different aspects of the system, and each of them require a different level of detail.

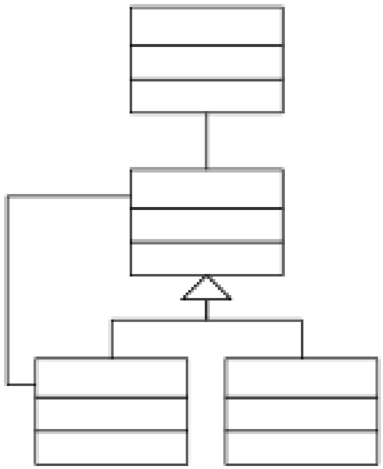
E.g., a programmer / coder needs to understand the design of the system and be able to convert the design to a low-level code, while a technical writer is interested in the behavior of the system as a whole, and needs to understand how the product functions.

UML attempts to provide a language that all stakeholders can benefit from at least one UML diagram.

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

Modeling viewpoints (1/3)

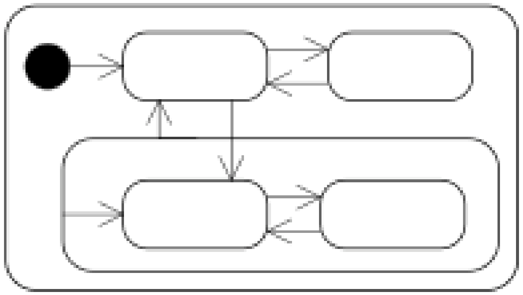
- Class model
  - Describes the structure of objects on the system in terms of attributes, operations and their relationships
  - It provides a context for describing the other viewpoints



- Purposes
  - Understanding the domain
  - Establishing a vocabulary
  - Structuring the system
  - Producing, maintaining and documenting code

Modeling viewpoints (2/3)

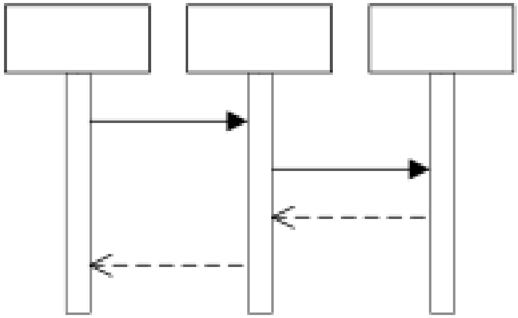
- State model
  - Describes sequencing of operations
    - The set of valid states, the events that mark changes on the current state and the constraints to be observed
  - Each state diagram shows the state model for a single class in the system



- Purposes
  - Understanding object behavior
  - Simulation, verification
  - Specifying executable controllers

Modeling viewpoints (3/3)

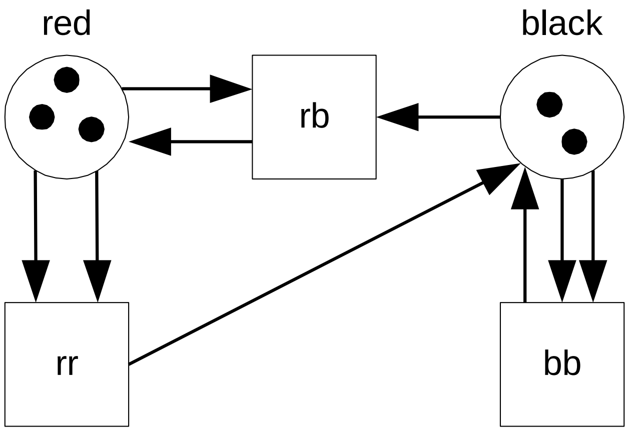
- Interaction model
  - Describes the interaction among objects in the system
  - Think about objects as entities collaborating to the global goal. How entities interact to this end?



- Purposes
  - Understanding interactions between user and system or between system components
  - Documenting scenarios
  - Producing test cases

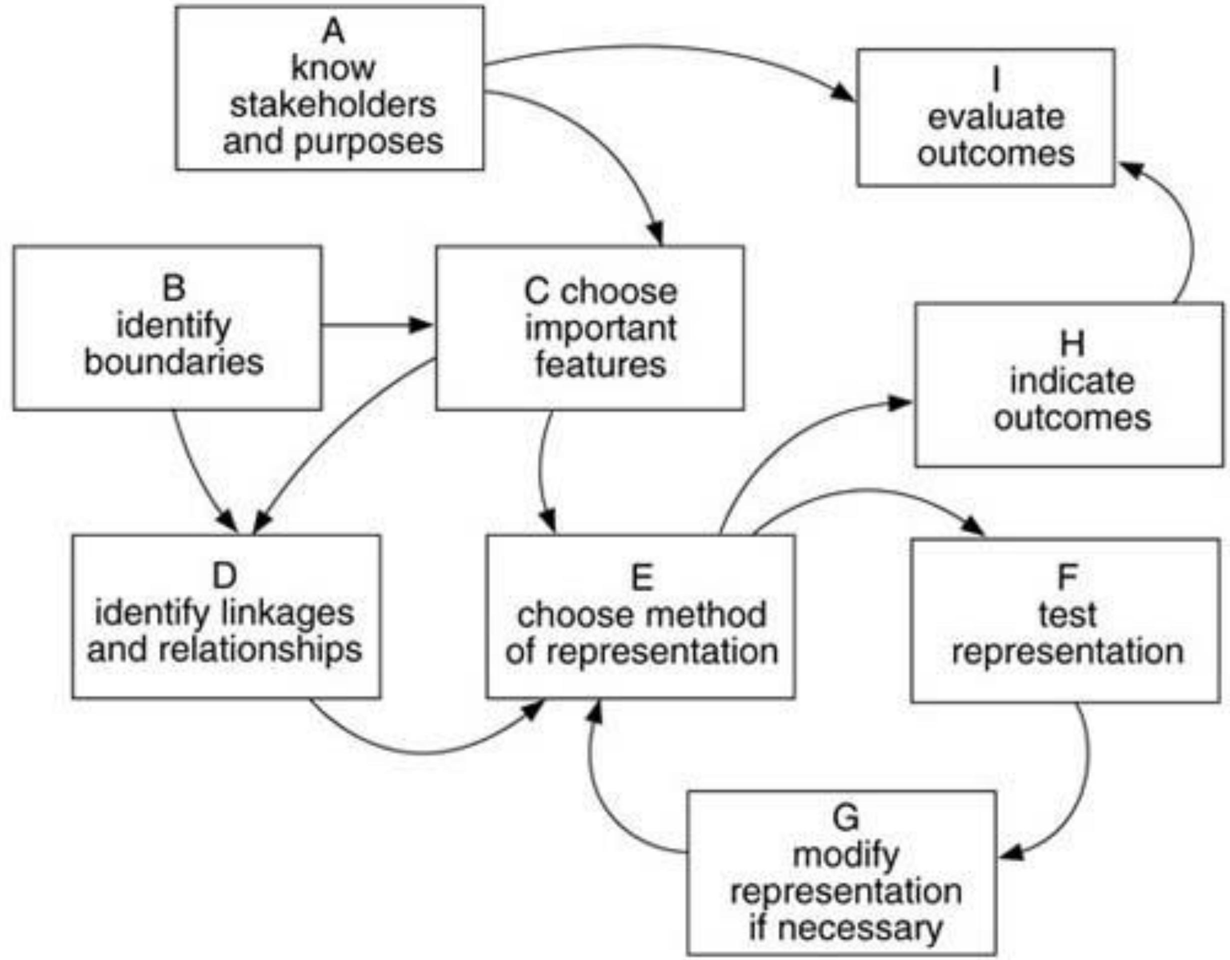
Modeling viewpoints (4/4)

- Discrete event model
  - Describes relations between events
  - Can capture concurrency and "resource contention"



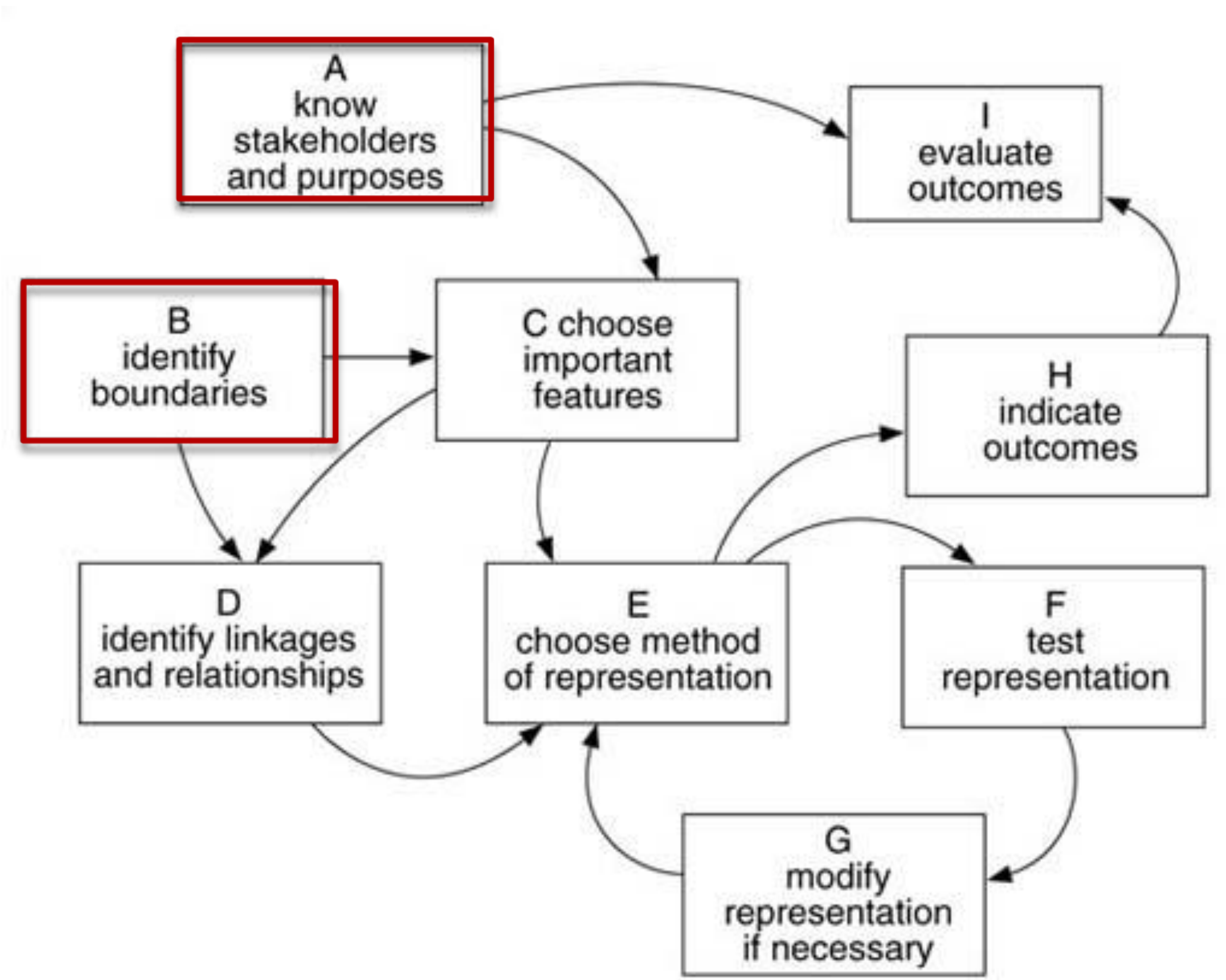
- Purposes
  - Analyzing systems with concurrency
  - Performance prediction
  - Bottleneck identification

All in all…


A
know stakeholders and purposes
I
evaluate outcomes
B
identify boundaries
C choose important features
H
indicate outcomes
D
identify linkages and relationships
E
choose method of representation
F
test representation
G
modify representation if necessary

The logical structure of the conceptual model of systems modelling, The Open University

All in all…



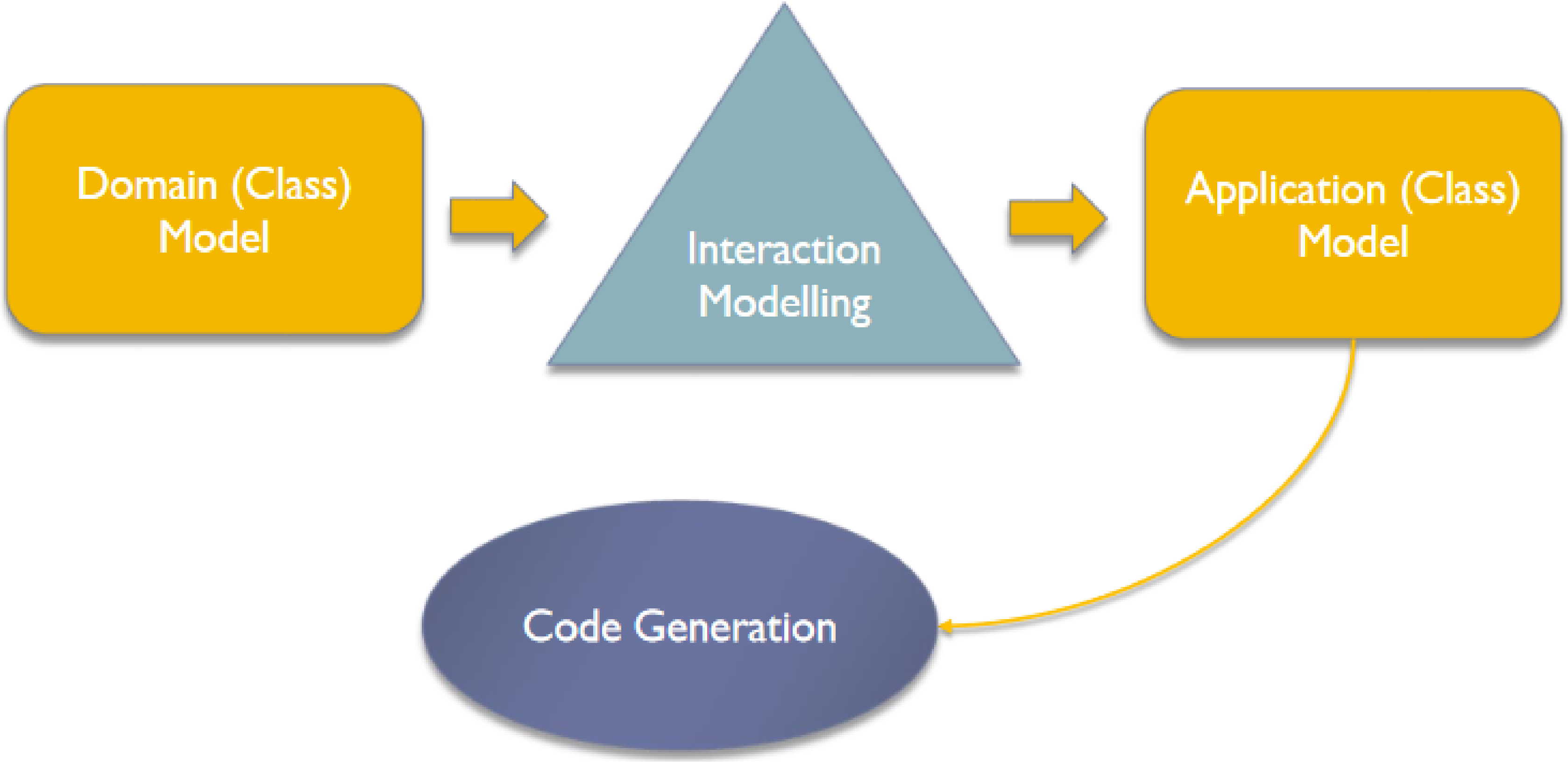
The logical structure of the conceptual model of systems modelling, The Open University

UML models

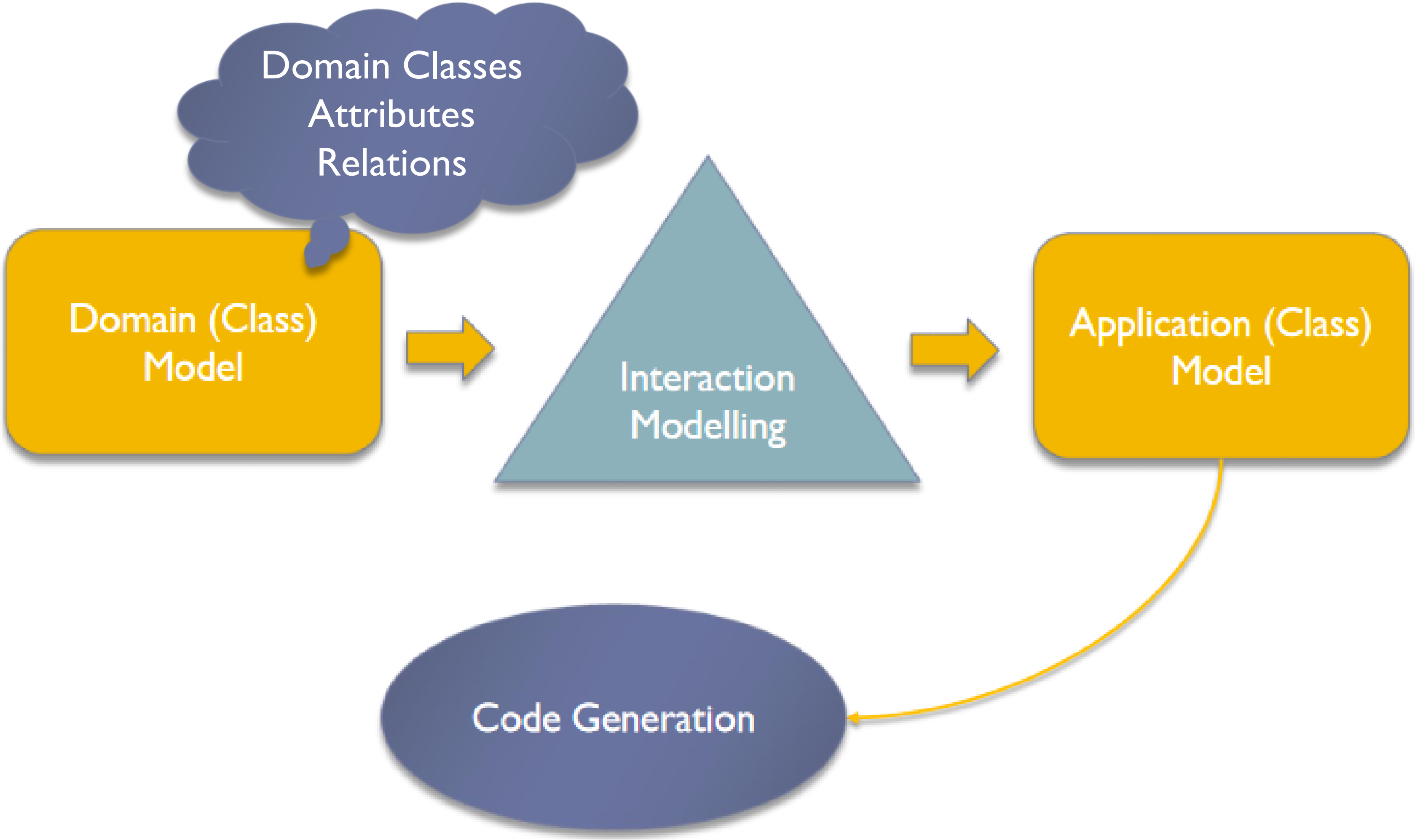
- **Class models**
  - Static structure of objects and their relationships
  - **Class diagrams**
    - Nodes are classes and arcs are relationships among classes

- **Interaction models**
  - Interactions can be modeled at **different levels of abstraction**
  - At a high level **use cases** describe *how a system interacts with outside actors*
  - Each **use case represents a functionality that a system provides to the user**
  - Use cases are helpful for **capturing informal requirements**
  - **Sequence diagrams** provide more details about *which operations need to be invoked in a specific scenario*

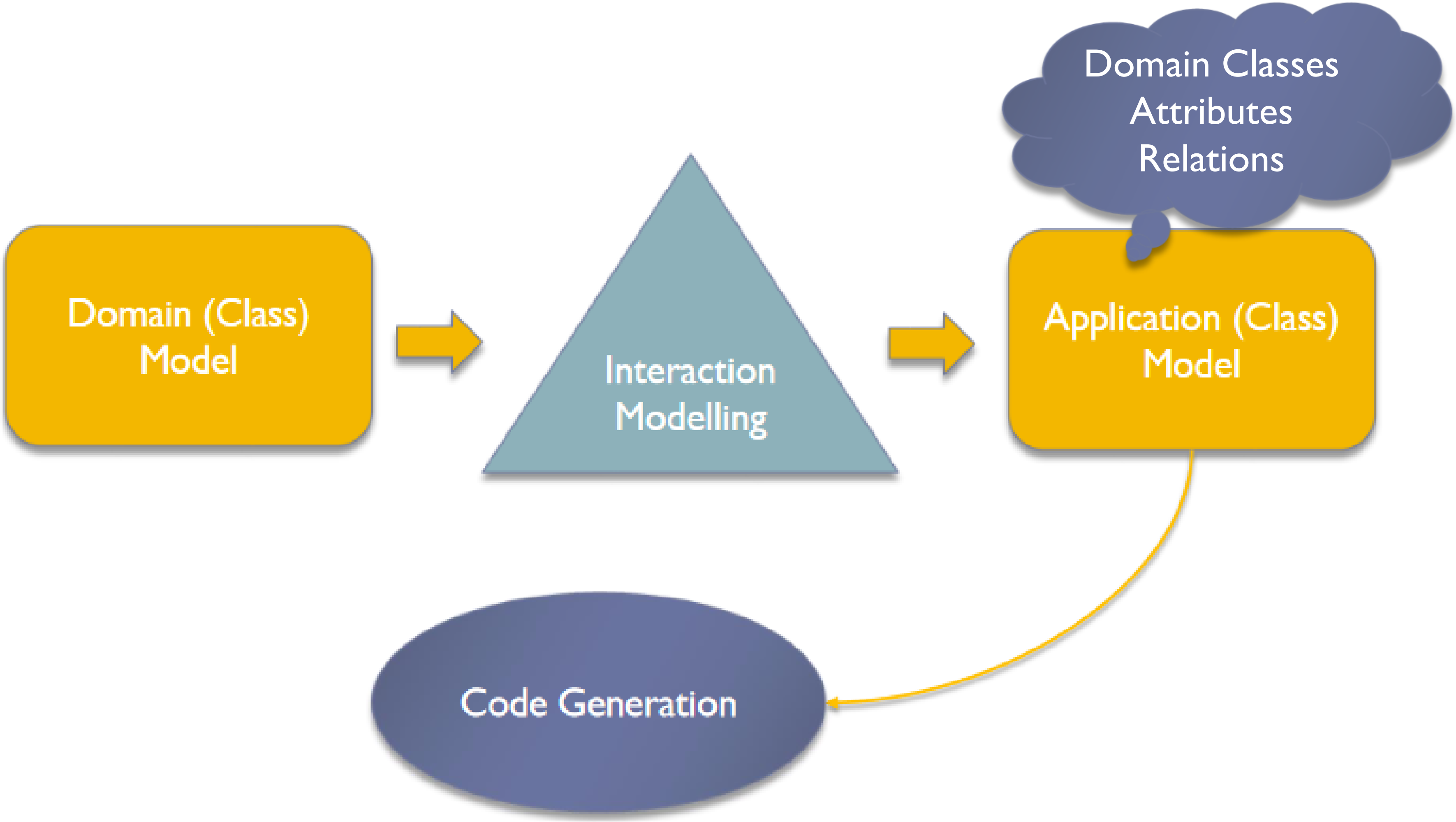
Software Development Methodology



Software Development Methodology



Software Development Methodology



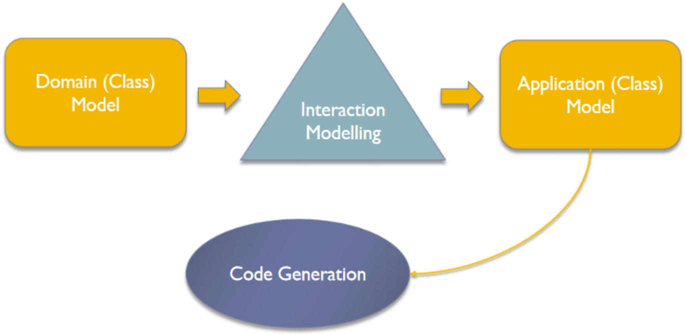
Class diagram

- The class diagram is a **central modeling technique** that **runs through nearly all object-oriented methods**.

- Describes the **types of objects** in the system and **various kinds of static relationships** which exist between them.

- In other words, Class Modeling focuses on static system structure in terms of **Classes** (**Class**, **Data Type**, **Interface** and **Signal items**), **Associations** and on characteristics of Classes (**Operations** and **Attributes**).

Domain (class) model



Domain (class) model

Domain (class) model

- To answer **WHAT?** question, the domain model provides **classes with attributes and relations among them**

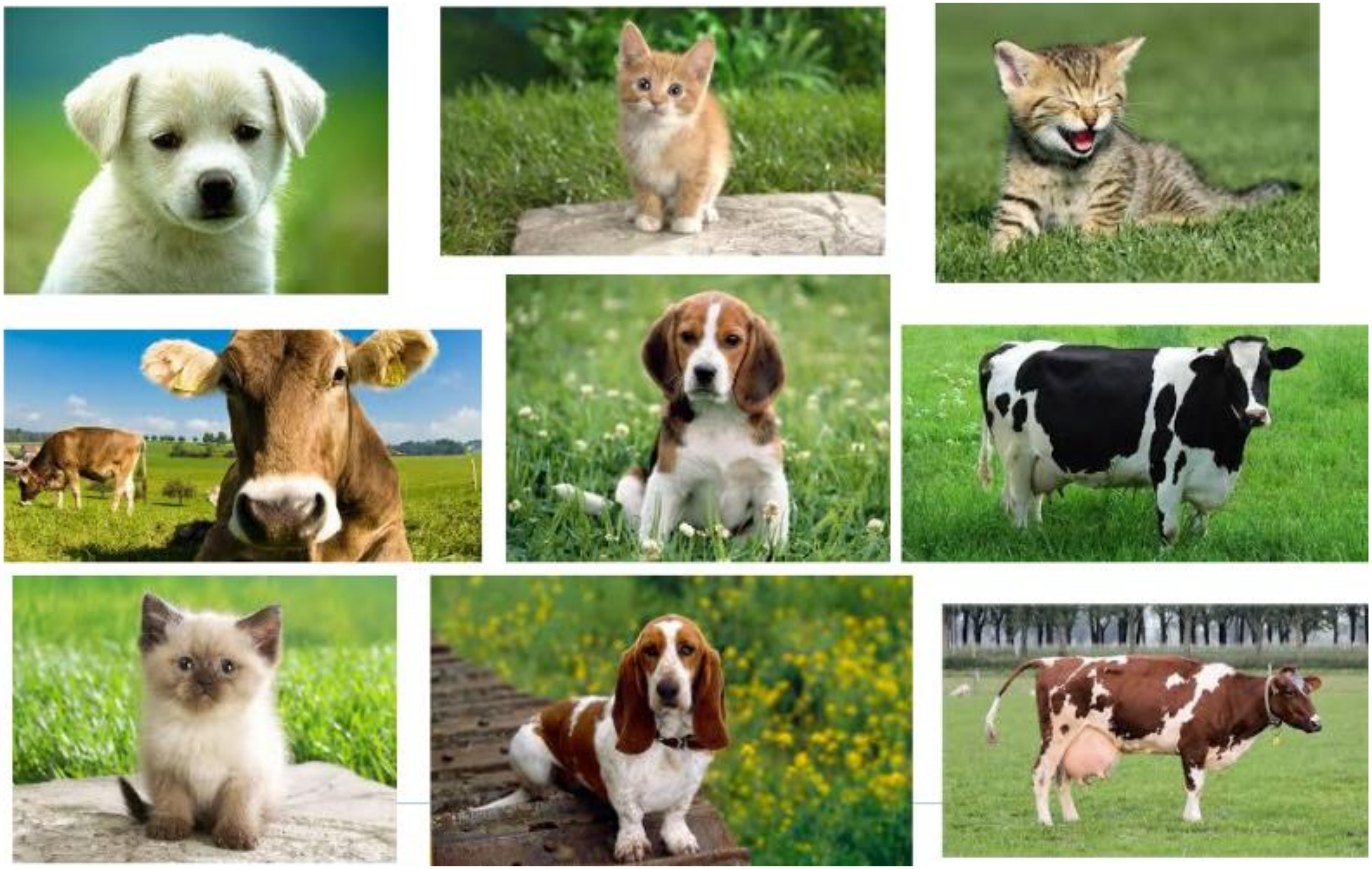
- **Operations are not specified**

Class diagrams

- **Classes**
  - A class describes **a group of objects with the same properties (attributes), behavior (operations), kinds of relationships and semantics**
  - Classes often appears as **nouns** in problem descriptions with users

- Objects
  - An object is a concept, abstraction or thing **with identity** that has a meaning for an application
  - An **object is an instance of a class**

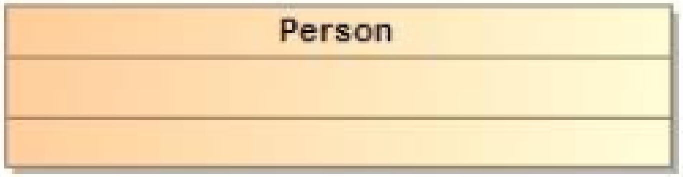
How many classes? And Instances?



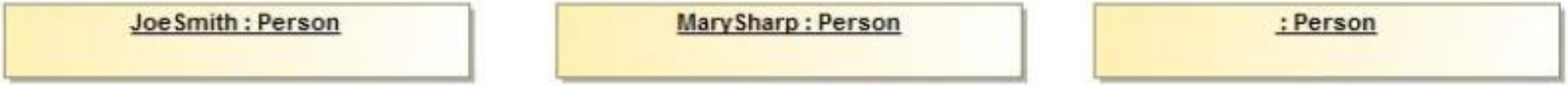
Is there only one correct answer?

Class diagrams

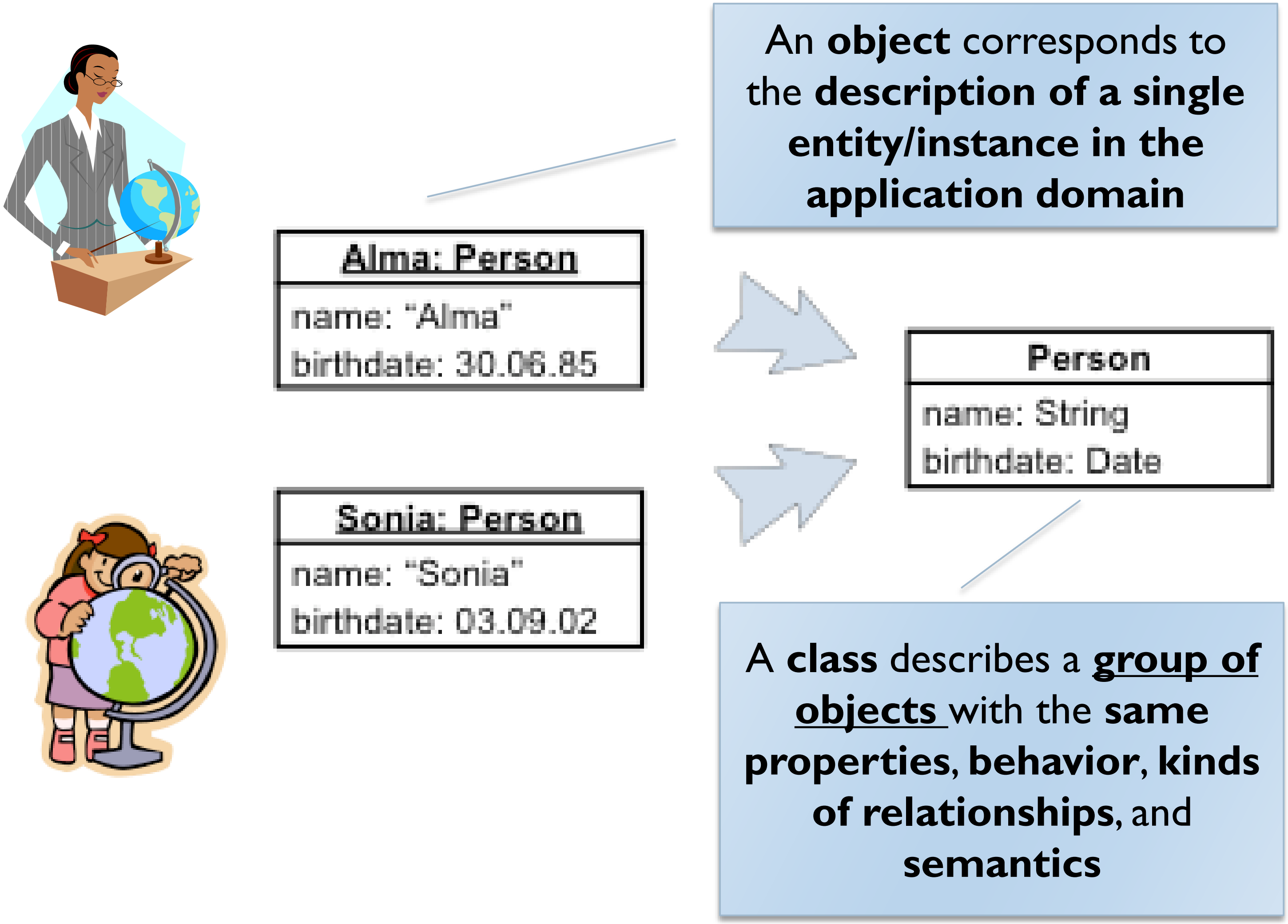
- Class
  - UML notation: box with a class name



- Object
  - UML notation: box with an object name followed by a colon and a class name. The object name and the class name are both underlined



Object versus Class



Alma: Person

name: "Alma"

birthdate: 30.06.85

Sonia: Person

name: "Sonia"

birthdate: 03.09.02

Person

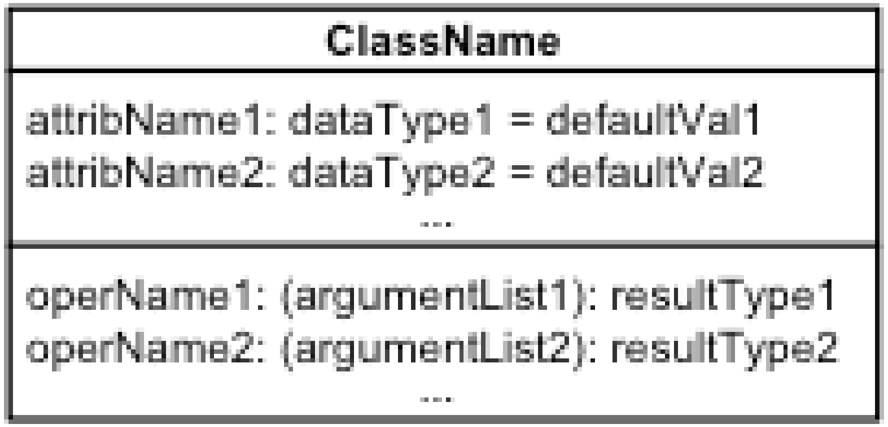
name: String

birthdate: Date

An **object** corresponds to the **description of a single entity/instance in the application domain**

A **class** describes a **group of objects** with the **same properties**, **behavior**, **kinds of relationships**, and **semantics**

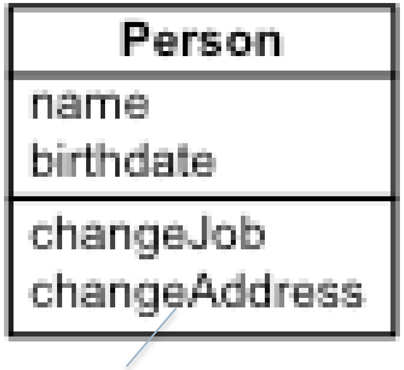
Class diagram



ClassName

attribName1: dataType1 = defaultVal1
attribName2: dataType2 = defaultVal2
...

operName1: (argumentList1): resultType1
operName2: (argumentList2): resultType2
...



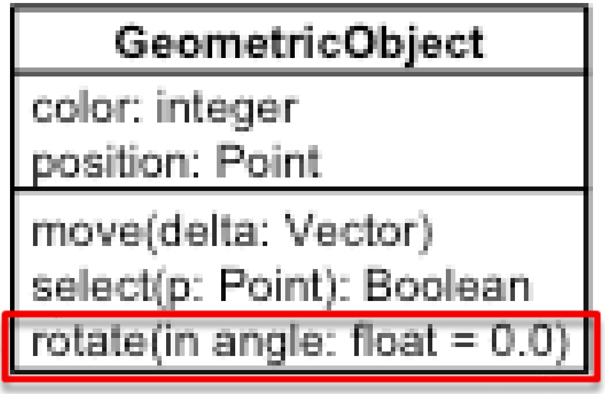
Person

name
birthdate

changeJob
changeAddress

Some details can be omitted

Richer specifications are also possible



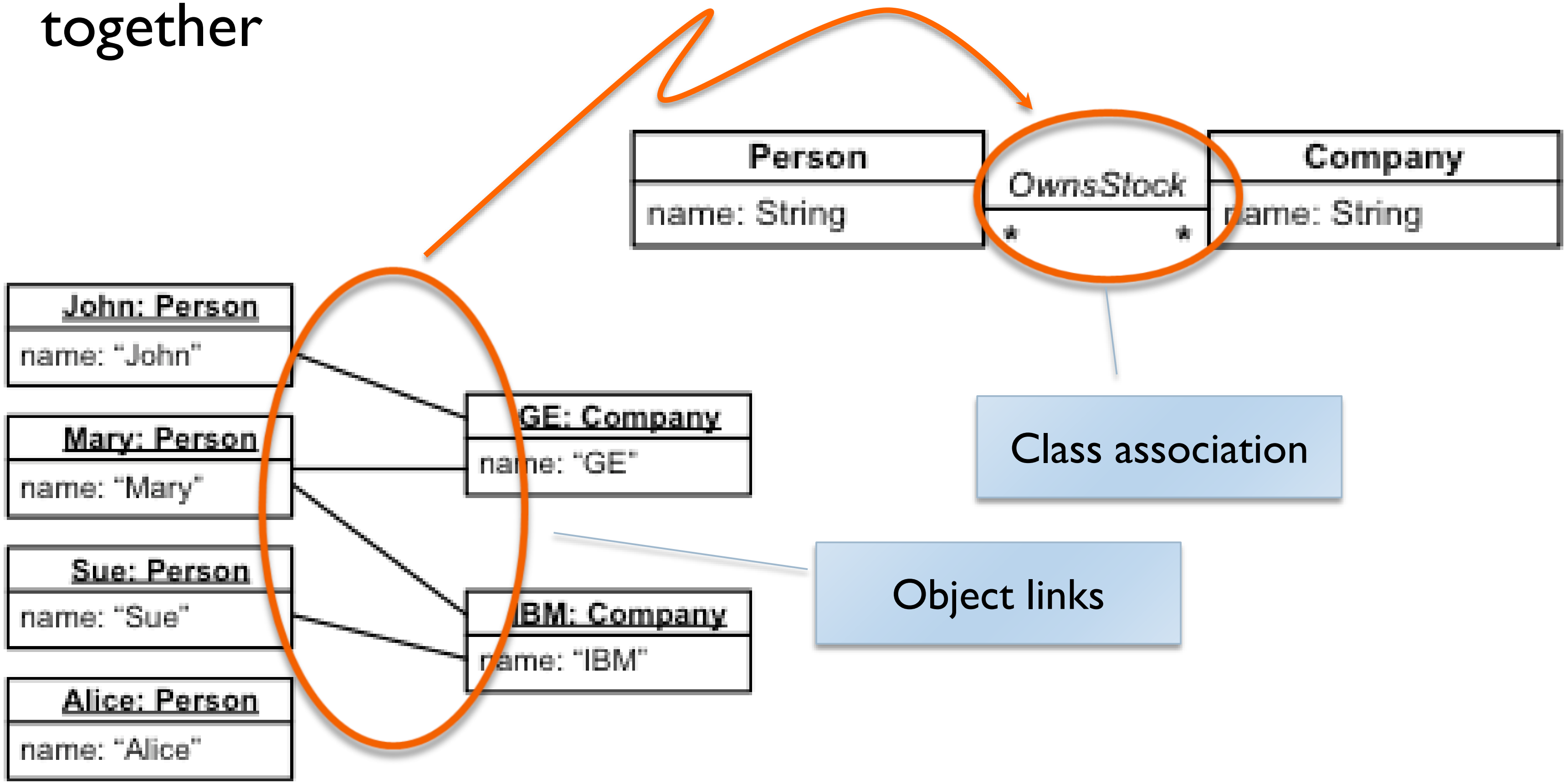
GeometricObject

color: integer
position: Point

move(delta: Vector)
select(p: Point): Boolean
rotate(in angle: float = 0.0)

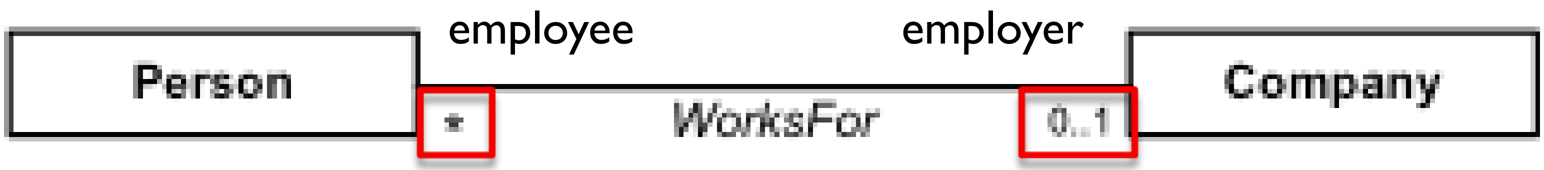
Class associations

- In the application domain, objects are usually linked together



Multiplicity and end names

An **association end name** specify the **role of a class** in the association

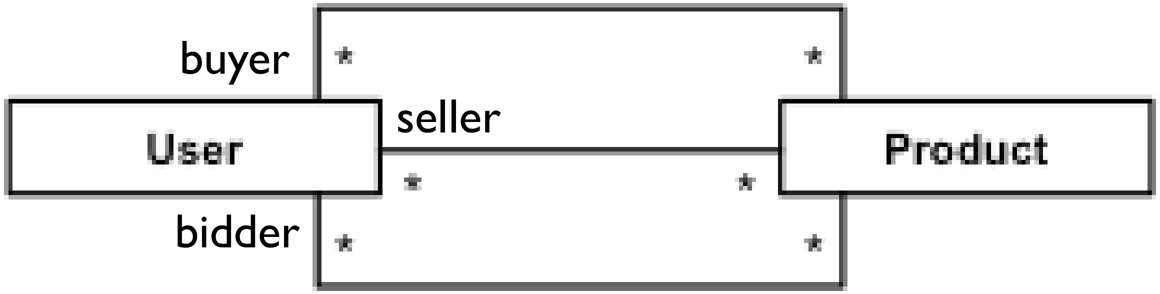


Multiplicity specifies the **number of objects that can be related** with respect to their underlying class

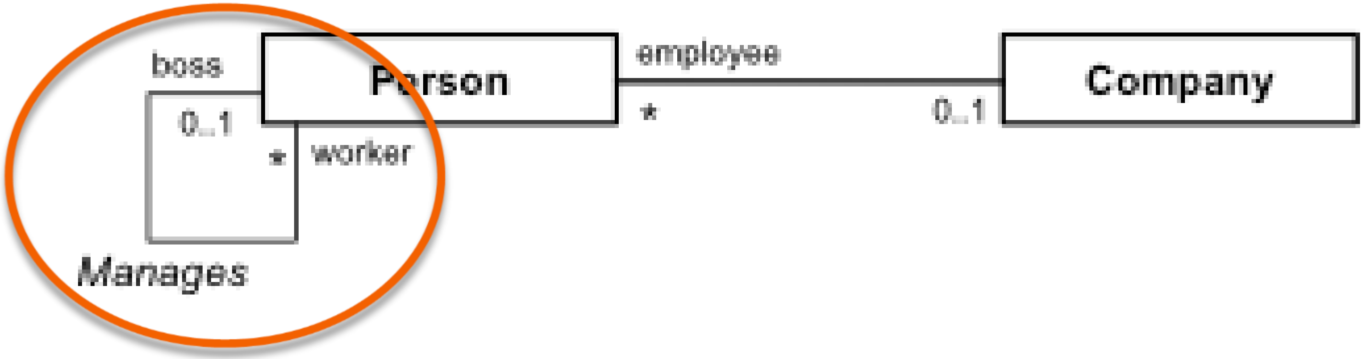
Examples

- 1 (default)
- * multiple
- 1..* one or more
- 3..5 three to five, inclusive

More about associations

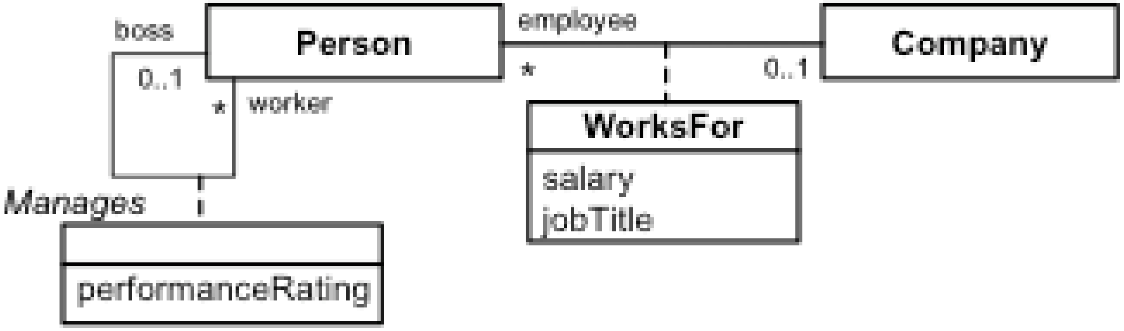


A pair of classes can have multiple associations

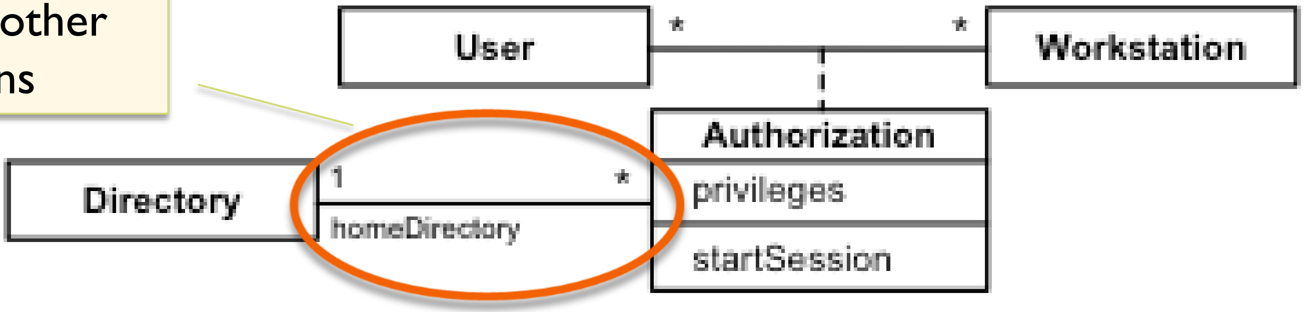


Association classes

Association may also have properties giving rise to association classes

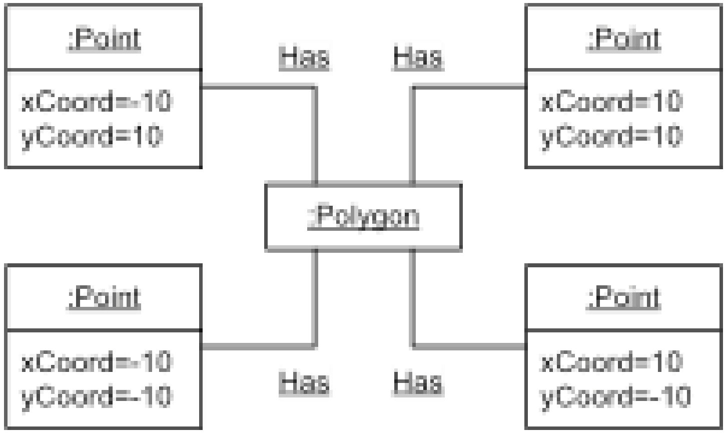


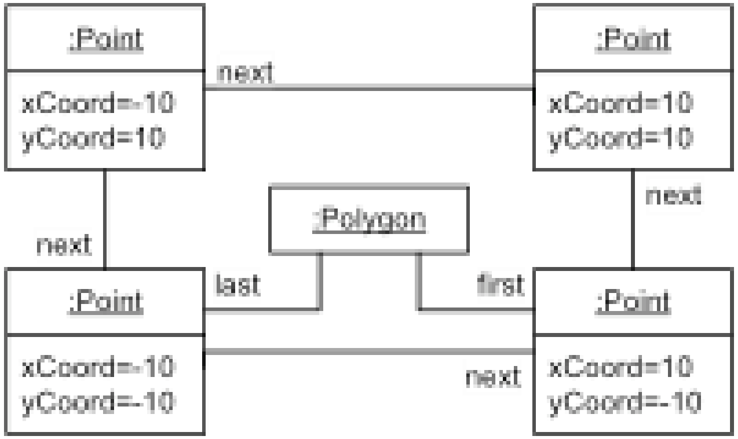
Association classes may participate in other associations



Exercise 1

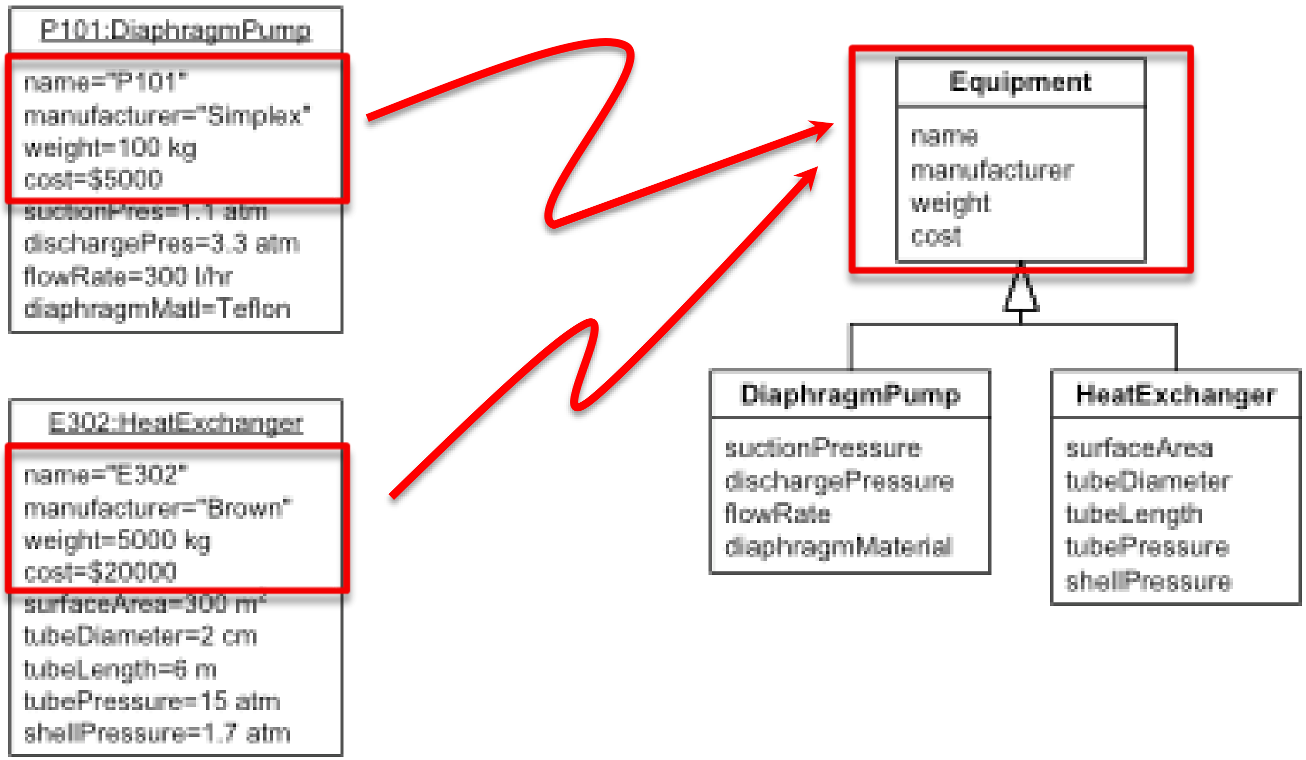
- Prepare a class diagram for the following object diagrams
  - Explain your decisions about multiplicity
  - Discuss about the differences of the resulting diagrams





Generalization

- Generalization is a relationship between classes providing an organized view of possible variations in both structure and behavior



P101:DiaphragmPump

name="P101"
manufacturer="Simplex"
weight=100 kg
cost=$5000
suctionPres=1.1 atm
dischargePres=3.3 atm
flowRate=300 l/hr
diaphragmMatl=Teflon

E302:HeatExchanger

name="E302"
manufacturer="Brown"
weight=5000 kg
cost=$20000
surfaceArea=300 m^2
tubeDiameter=2 cm
tubeLength=6 m
tubePressure=15 atm
shellPressure=1.7 atm

Equipment

name
manufacturer
weight
cost

DiaphragmPump

suctionPressure
dischargePressure
flowRate
diaphragmMaterial

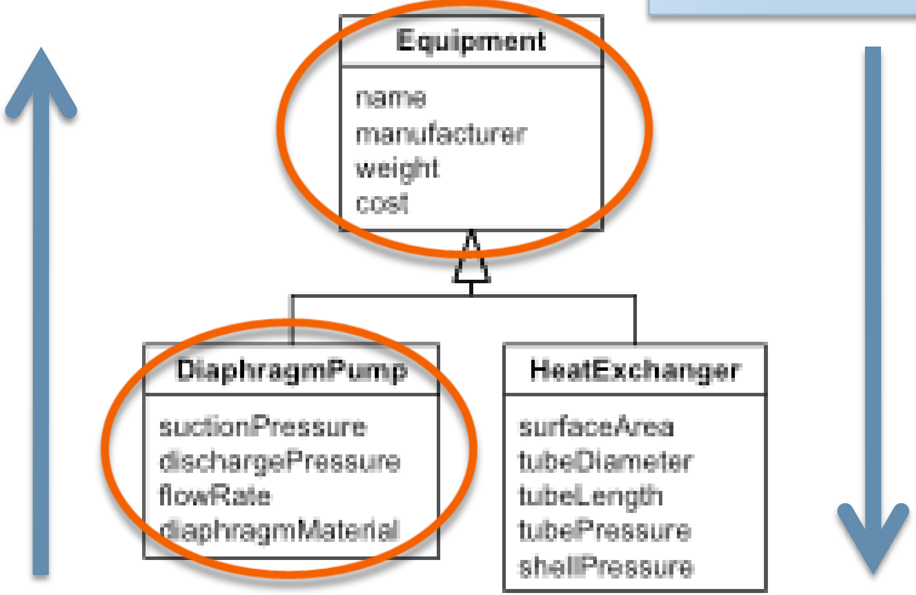
HeatExchanger

surfaceArea
tubeDiameter
tubeLength
tubePressure
shellPressure

Generalization and specialization

The **superclass** holds common properties and associations

Generalization

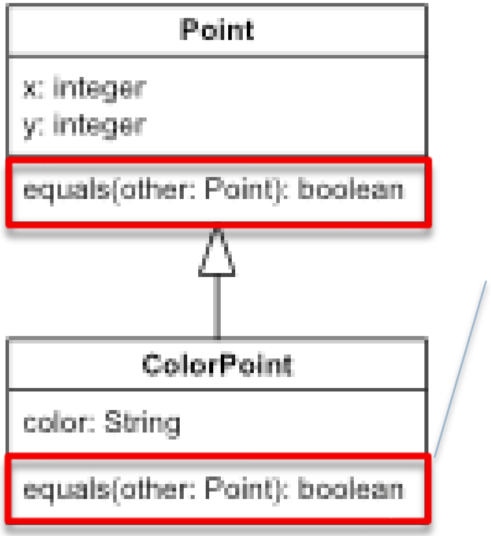


Specialization

Each **subclass** adds specific properties and associations

Inheritance and overriding

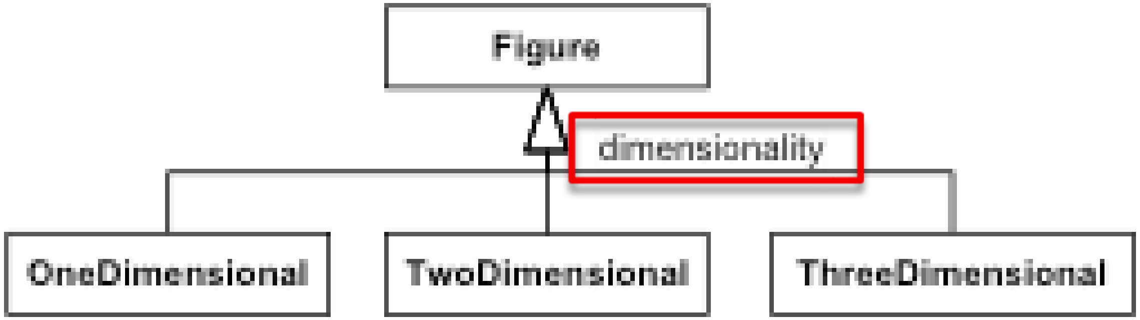
Inheritance is the mechanism for sharing attributes, operations, and associations via the generalization/specialization relationship



A subclass **overrides** a superclass operation when it lists the same name. The overriding operation is expected to refine and/or replace the overridden operation.

Generalization set name

- The **generalization set name** is an optional attribute that indicates the aspect being abstracted by a particular generalization

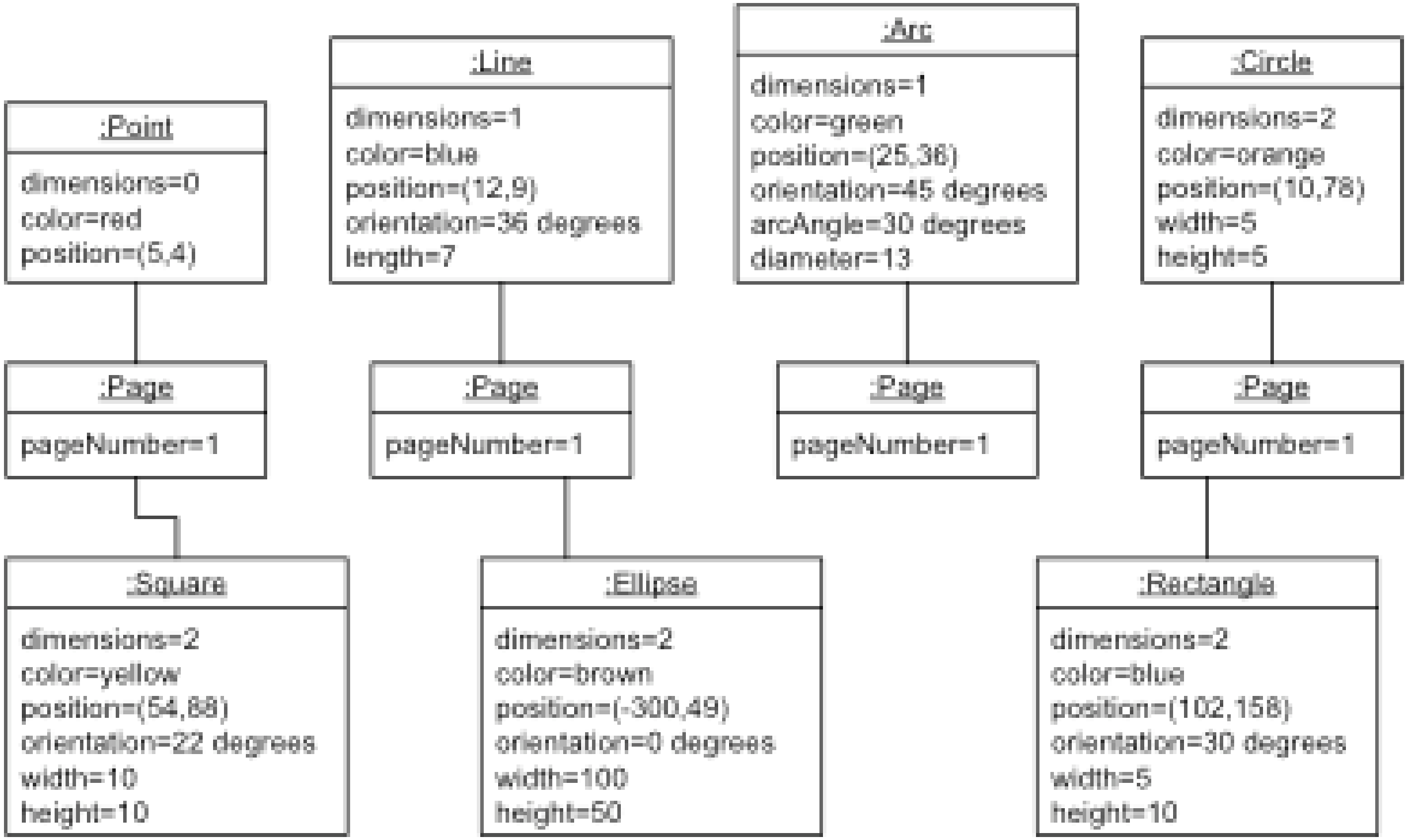


The class ***Vehicle*** can be specialized according to the following aspects:

- Means of propulsion:
  wind, fuel, animal, gravity
- Operating environment:
  land, air, water, outer space

Exercise 2

- Prepare a class diagram from the following object diagram
  - Identify generalization relationships



:Point
dimensions=0
color=red
position=(5,4)

:Line
dimensions=1
color=blue
position=(12,9)
orientation=36 degrees
length=7

:Arc
dimensions=1
color=green
position=(25,36)
orientation=45 degrees
arcAngle=30 degrees
diameter=13

:Circle
dimensions=2
color=orange
position=(10,78)
width=5
height=5

:Page
pageNumber=1

:Page
pageNumber=1

:Page
pageNumber=1

:Page
pageNumber=1

:Square
dimensions=2
color=yellow
position=(54,88)
orientation=22 degrees
width=10
height=10

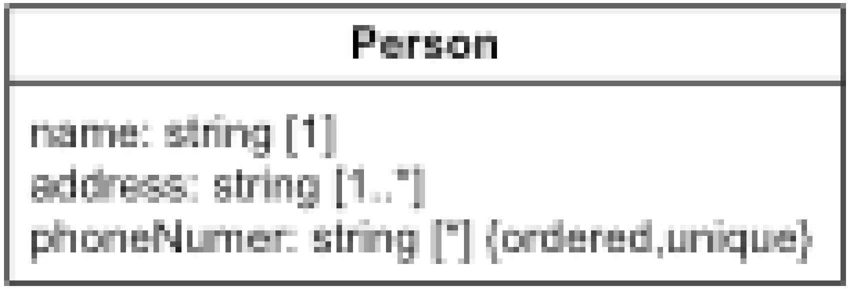
:Ellipse
dimensions=2
color=brown
position=(-300,49)
orientation=0 degrees
width=100
height=50

:Rectangle
dimensions=2
color=blue
position=(102,158)
orientation=30 degrees
width=5
height=10

Class properties

Multiplicity

- [1] (default)
- [*] multiple
- [1..*] one or more
- [3..5] three to five, inclusive



Person

name: string [1]
address: string [1..*]
phoneNumer: string [*] {ordered,unique}

Visibility (depends on the programming language used for implementation)

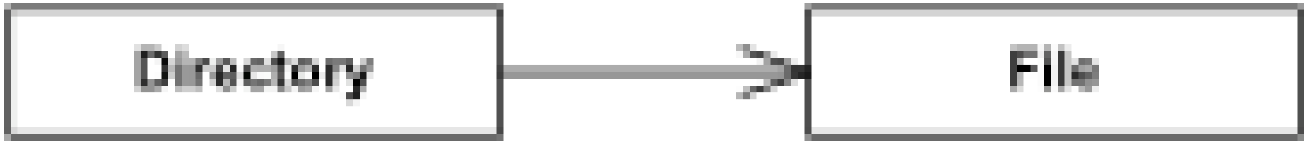
- + public
- # protected
- - private
- ~ package

Constraints for multi-valued attributes

- {ordered}
- {unique}

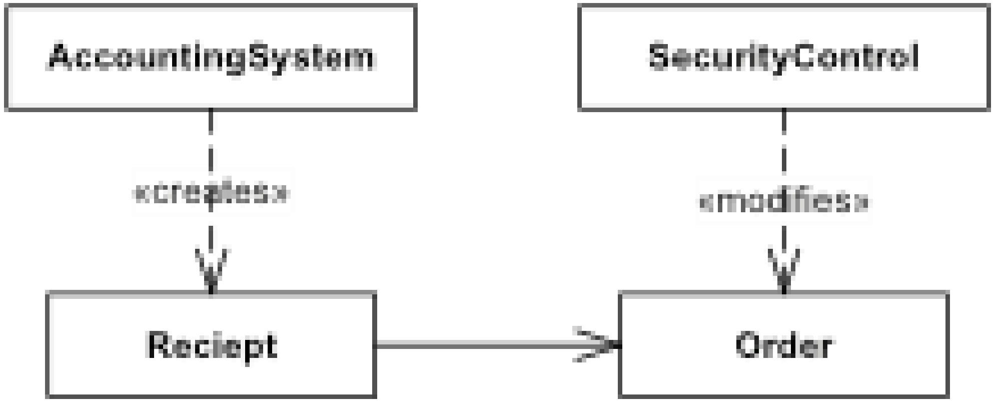
Navigation

- If an association is directed, messages can pass only on that direction
- If the association does not have directions, then it is a bidirectional association



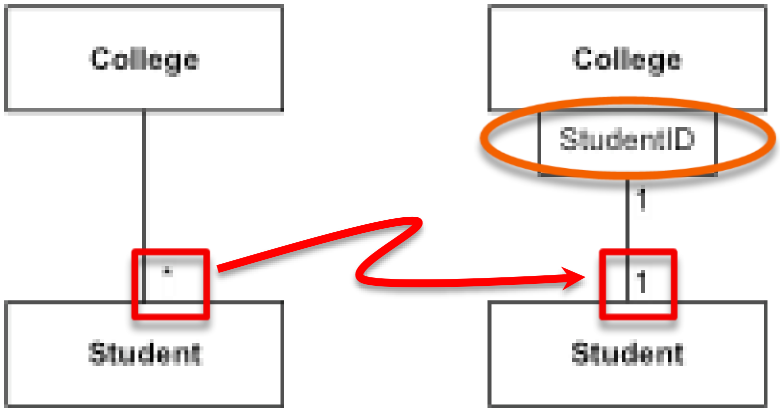
Dependencies

- A **dependency** is the most general relation between classes
- It indicates that an object affects another object



Qualifiers

- A **qualifier** is an attribute or list of attributes whose values serve to partition the set of objects associated with an object across an association

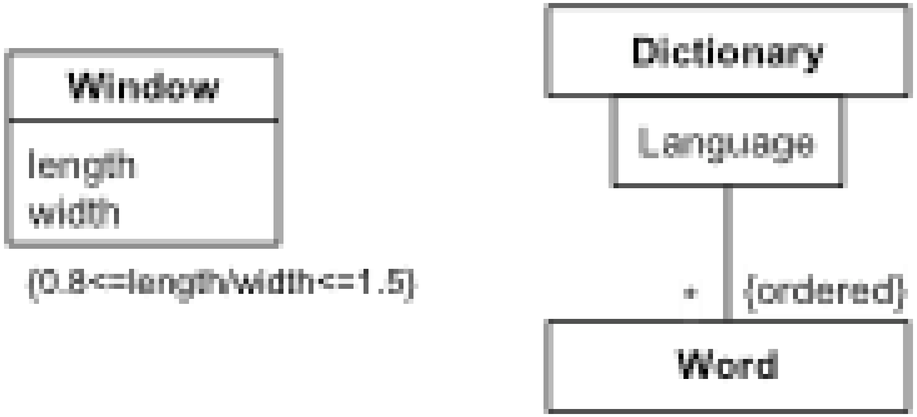


Qualifier

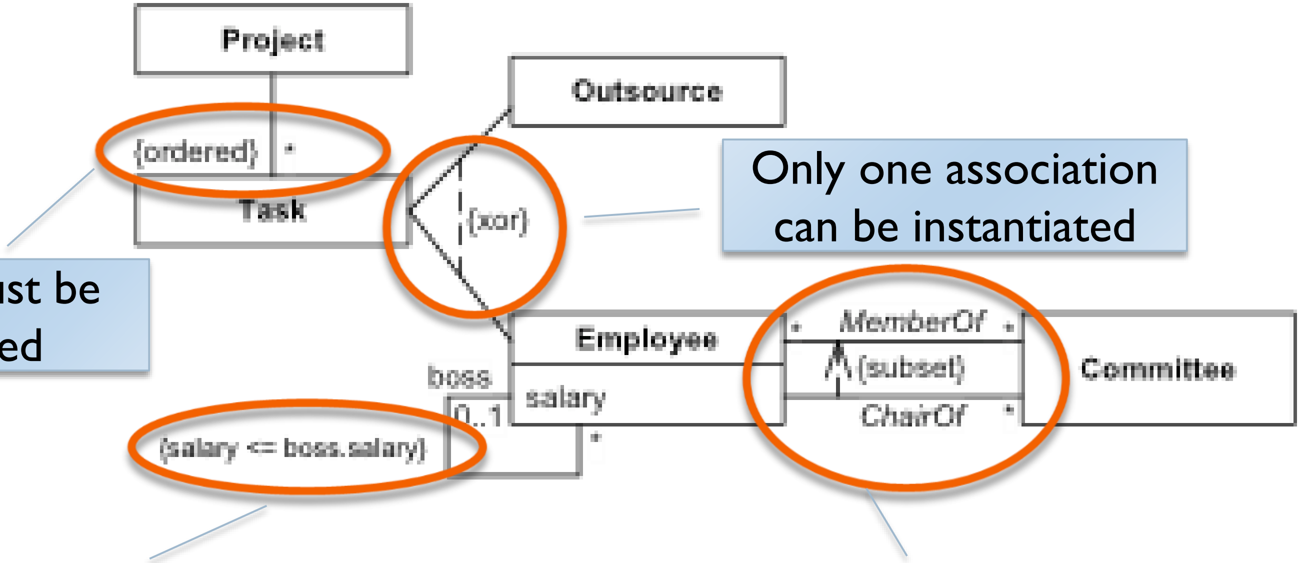
A qualifier selects among the target objects, reducing the effective multiplicity from "many" to "one"

Constraints

- Constrains are simple properties of associations, classes and many other things in UML
- Specify limitations that implementers need to satisfy



Examples of constraints



Only one association can be instantiated

Tasks must be ordered

Salary constraint must be enforced by the implementation

The *ChairOf* association is a subset of *MemberOf* association

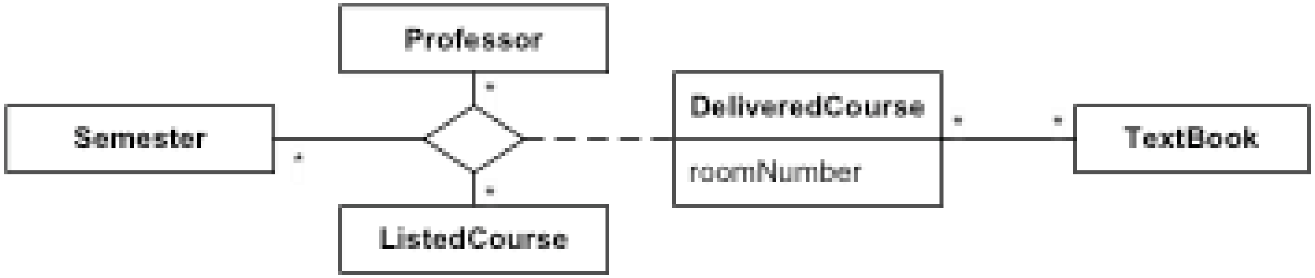
Constraints in UML

- Constraints can be applied to almost every element in UML diagrams, using:
  - natural language
  - mathematical notation
  - OCL (Object Constraint Language)

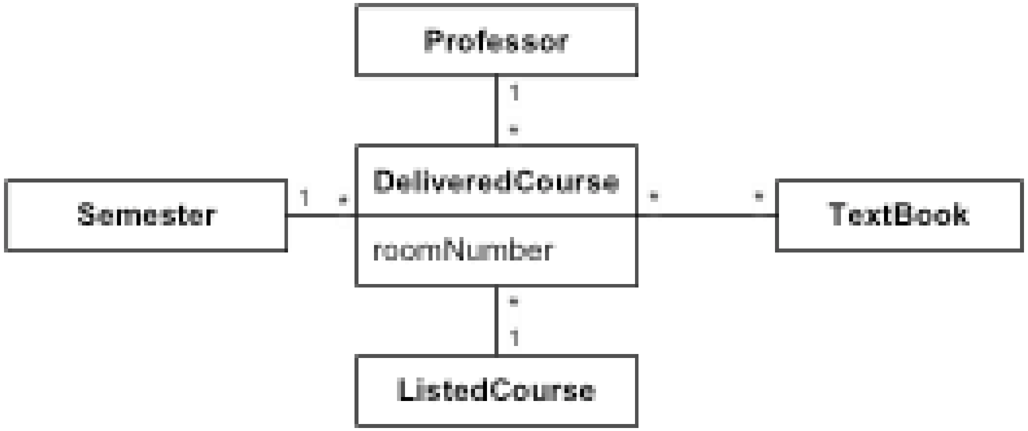
- Constraints can be used for expressing:
  - Invariants
    - interest > 3%
  - Preconditions
    - before loan() takes place, salary > 5,000$
  - Postconditions
    - after loan() takes place, dayCollect = 1 or 10

N-ary associations

- Occasionally, you will find some associations among three or more classes
  - Try to decompose those n-ary associations into binary associations

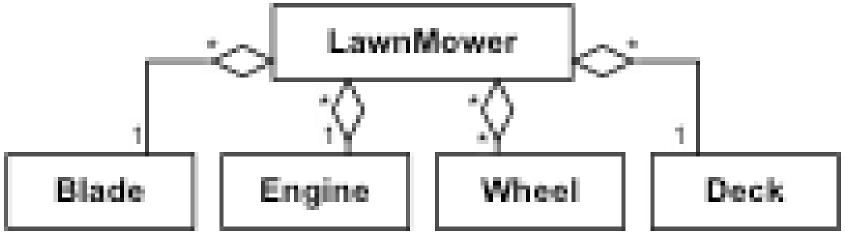


The meaning of the new model might be slightly different (for instance, you must be careful about multiplicity)

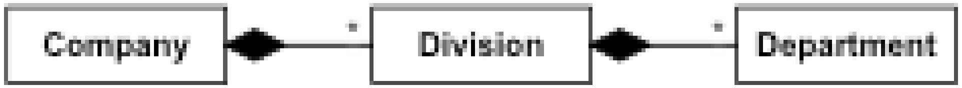


Aggregation and composition

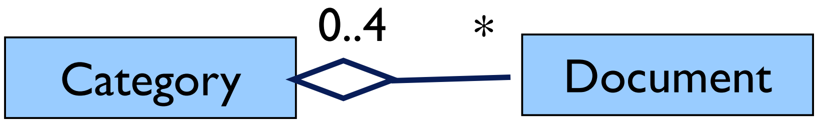
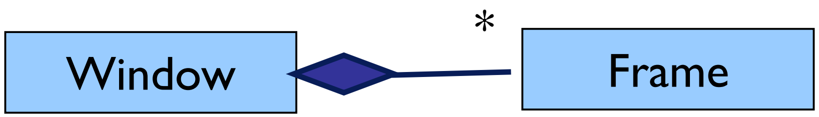
- Aggregation is a special form of association
  - Underlines the fact that an object is made of constituent parts



- Composition is a more restrictive form of aggregation
  - Two additional constraints
    - A constituent part can belong to at most one assembly
    - The part has a coincident lifetime as the assembly



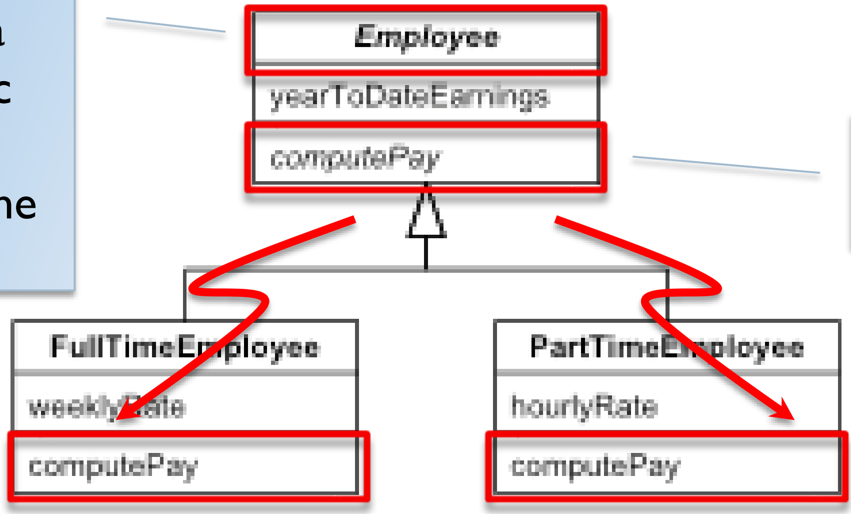
Association versus Composition

Aggregation	**Composition**
Part can be shared by several wholes 	Part is always a part of a single whole 
Parts can live independently (i.e., whole cardinality can be 0..*)	Parts exist only as part of the whole (e.g. when a Window is destroyed all other widgets are also destroyed)
Whole is not solely responsible for the object	Whole is responsible and should create/destroy the objects

Abstract classes

- An **abstract class** is a class that has no direct instances
  - It may define common properties
  - It may define some operation signatures called **abstract operations**

An abstract class is specified with a name in an italic font (or using {abstract} near the class name)

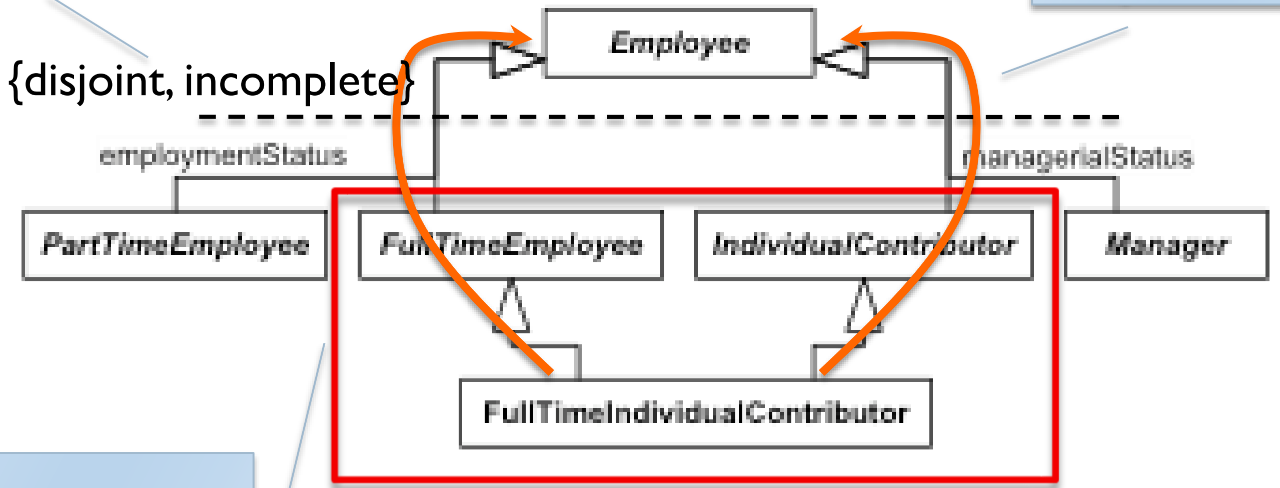


Abstract operation (use an italic font)

Multiple inheritance

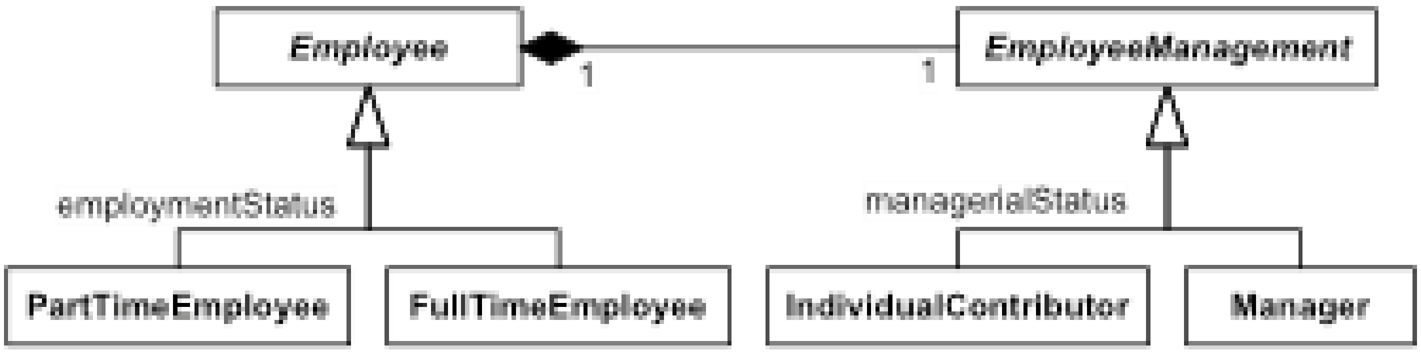
The "diamond problem" will not be present if the classes are disjoint

Be careful with the "diamond problem": some overlapping may cause conflicts



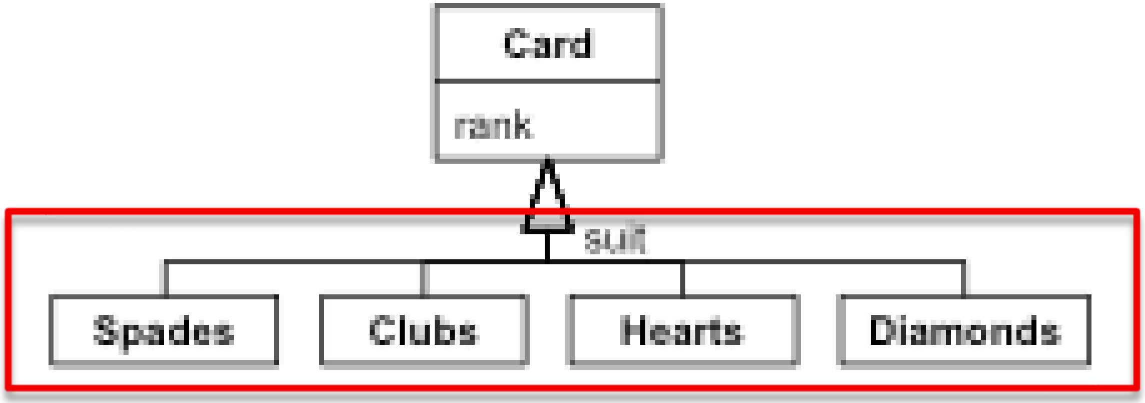
A class can specialize more than one superclasses

Delegation as an alternative to multiple inheritance

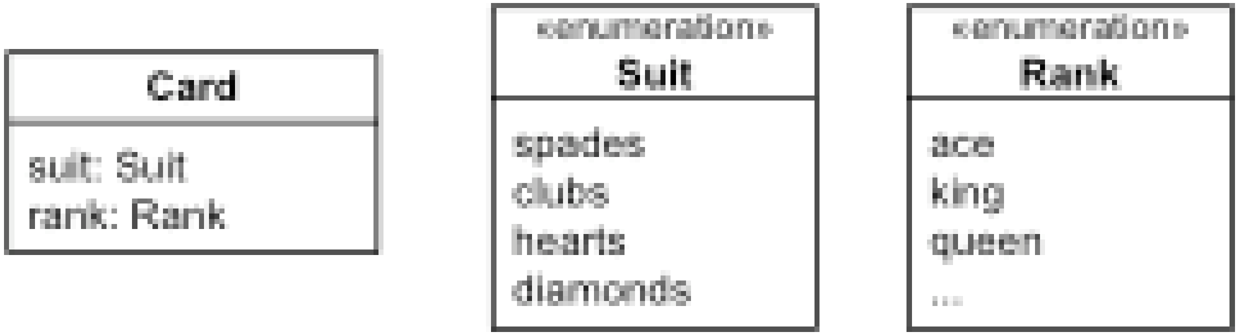


Note: Consider using interfaces…

Enumerations

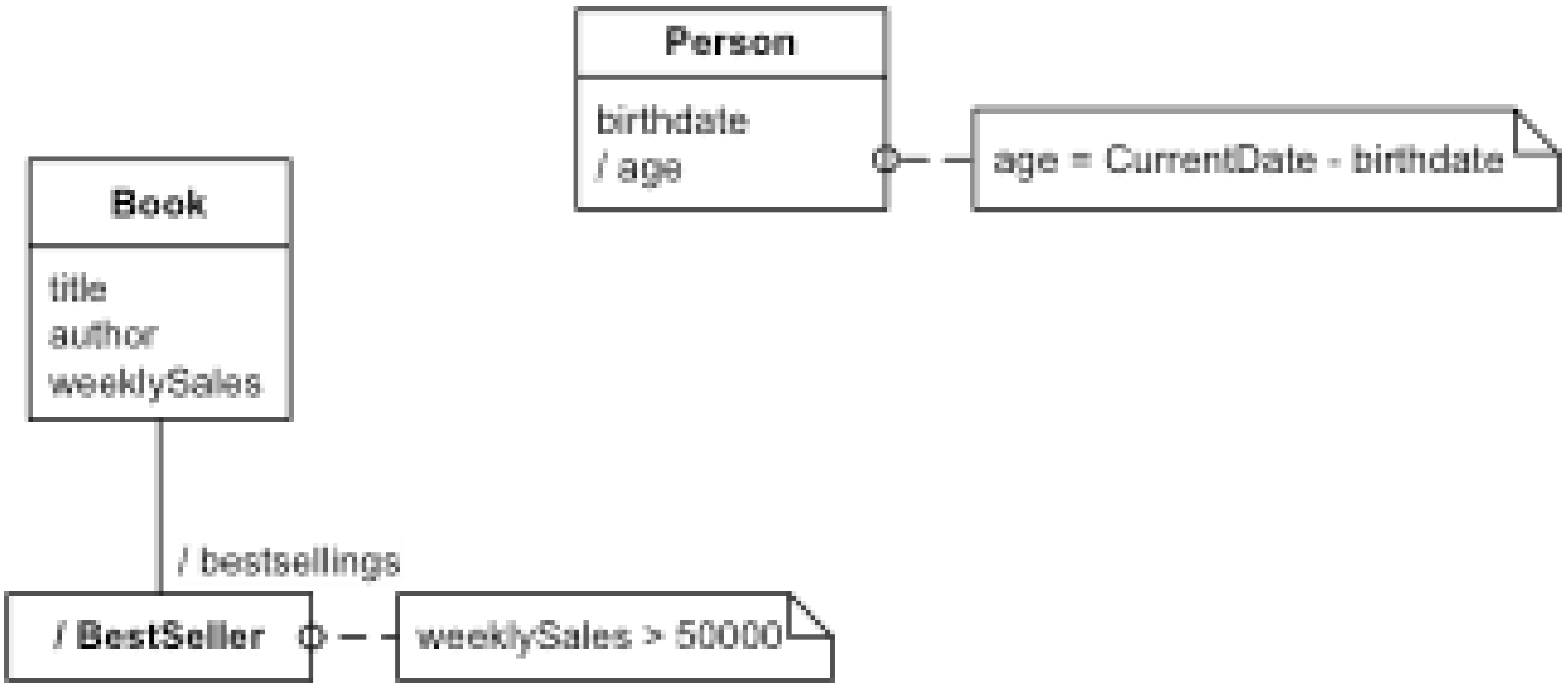


An **enumeration** is a data type that has **a finite set of values**. You should avoid modeling enumerations as generalization hierarchies.



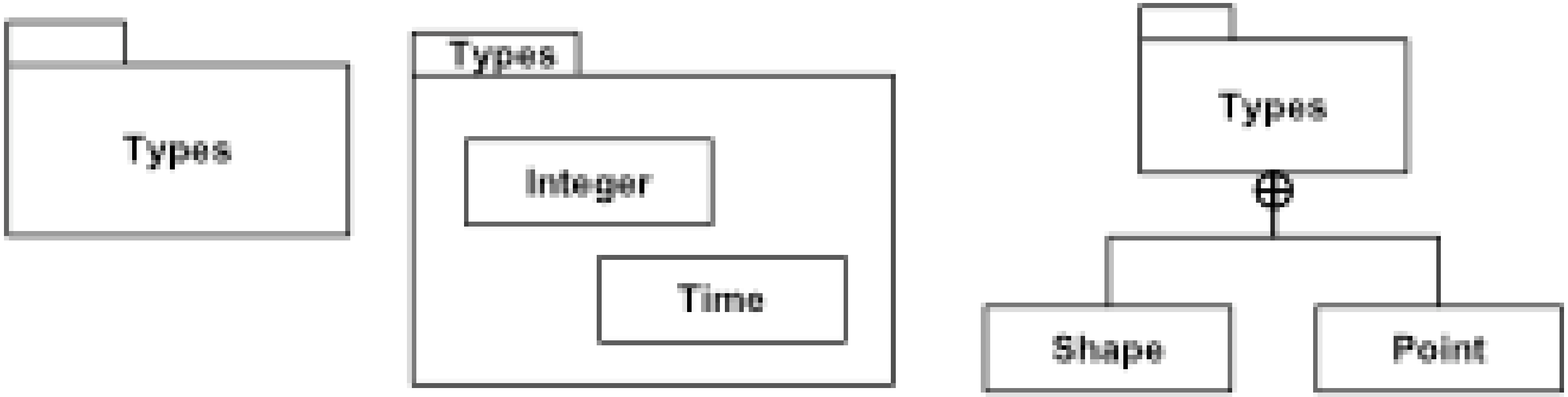
Derived data

- A **derived element** is a function of one or more elements, which in turn can be derived



Packages

- A **package** is a group of elements (classes, associations, generalizations and nested packages) with a common theme
  - A model can be partitioned in packages to make it easier to understand



Key takeaways

- The class model is the graphical representation of the structure of the system and also the relation between the objects and classes in the system.
- Each object in the system has data structure and behaviour.
- Object sharing the same features are grouped to a class.
- Objects are the proper nouns and classes are common nouns identified in the problem statement provided for the development of the application.
- The relationship between the objects is the link and the group of links with the same structure is termed as an association.
- The class model focuses on the factors that are essential from the applications point of view.

https://binaryterms.com/class-model.html

Some reading (to be updated)

- **Textbook**
- Michael Blaha and James Rumbaugh. Object-Oriented Modeling and Design with UML (2nd Edition), Prentice Hall, 2004
- Kurt Jensen and Lars M. Kristensen. Coloured Petri Nets. Springer 2009.
- **Links**
- Fully elaborated ATM example in UML by Russell Bjork
- UML 2.2 Stencil for Visio
- Story-driven Modeling by Albert Zündorf.
- Woped
- Workflow course by Wil van der Aalst
- CPN Tools home page