

Software Engineering UML

Erik Fredericks // frederer@gvsu.edu

Adapted from materials provided by Byron DeVries, Jagadeesh Nandigam

Outline

What is modeling and why?

What is UML

Building blocks of UML

UML 2.x diagrams

- Structural diagrams
- Behavioral diagrams

But what ... is ... modeling (and why?)

A model is a **simplification** of reality

A model provides the **blueprints** of a system

A model may **describe** either structure or behavior of a system

- Structure – organization of the system
- Behavior – dynamics of the system

We build models to **better understand** the system we are developing

But what ... is ... modeling (and why?)

Through modeling, we achieve four aims:

- **Visualize** a system as it is or as we want it to be.
- **Specify** the structure or behavior of a system.
- Develop templates that can be used to **construct** a system.
- **Document** the decisions we have made.

What are some “models” outside of software engineering that accomplishes each aim?

Ok, well what is UML then?

UML is a language for visualizing, specifying, constructing, and documenting the artifacts of a (software) system.

Initial versions of UML were the unification efforts of (mostly) three prominent methods developed by the “three amigos”:

- Booch method by Grady Booch
- OMT (Object Modeling Technique) by James Rumbaugh
- OOSE (Object-Oriented Software Engineering) by Ivar Jacobson

The UML specification is now updated and managed by the Object Management Group (OMG): www.uml.org

Three Amigos (UML)



G. Booch



I. Jacobson



J. Rumbaugh

What is UML?

UML is **process independent**.

It is most suitable for **use-case driven**, **iterative**, and **incremental** development processes.

UML is not **limited to modeling software** and can be applied to modeling other (non-software) systems.

Building Blocks of UML

The vocabulary of UML consists of three kinds of building blocks:

Things:

- These are first-class citizens in a model (e.g., class in a class diagram)

Relationships:

- Tie “things” together

Diagrams:

- Graphical representation of interesting collections of things along with relationships.



**In a use case diagram
what are the things
and relationships?**

UML 2.x

UML 2.x (currently 2.5) defines two major kinds of diagrams:

- Structure Diagrams
- Behavior Diagrams

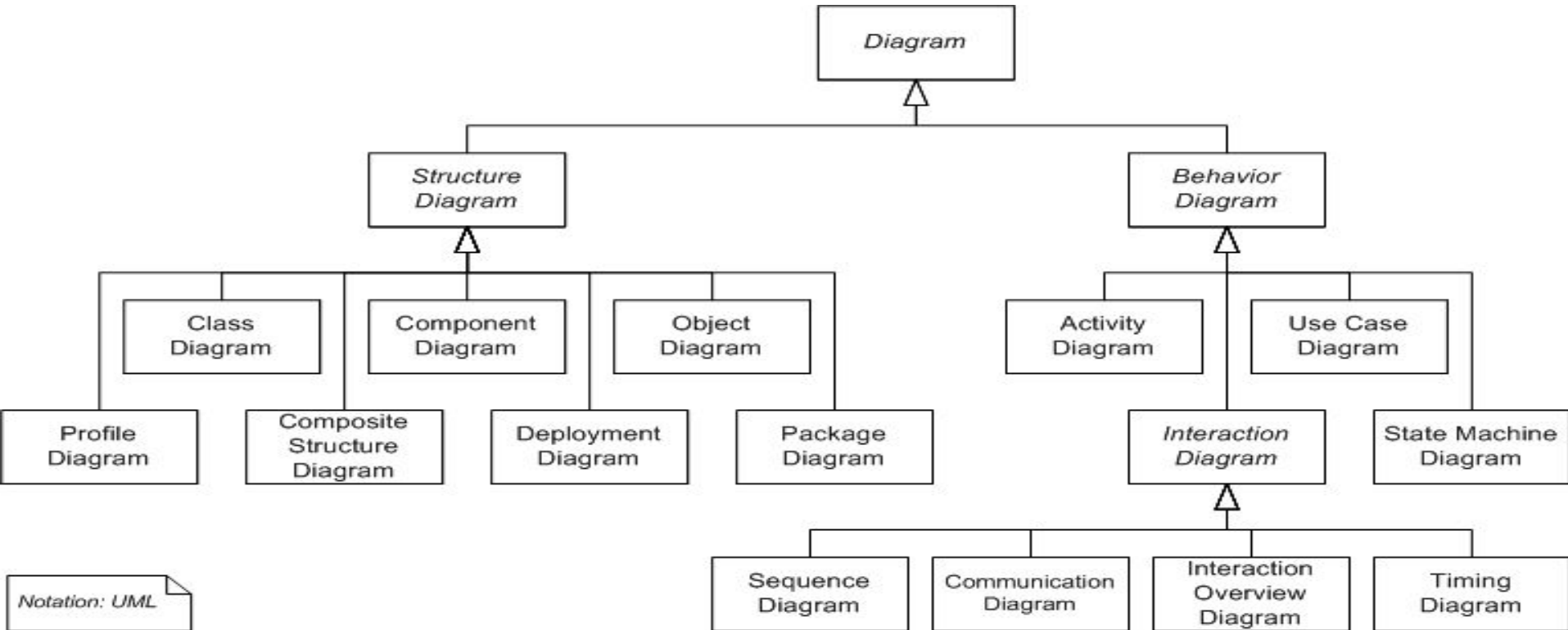
Structure diagrams

- Static structure of the system and its parts at different abstraction and implementation levels and how the parts are related to each other.

Behavior diagrams

- Dynamic behavior of objects (and other things) in a system.

UML 2.x Diagrams



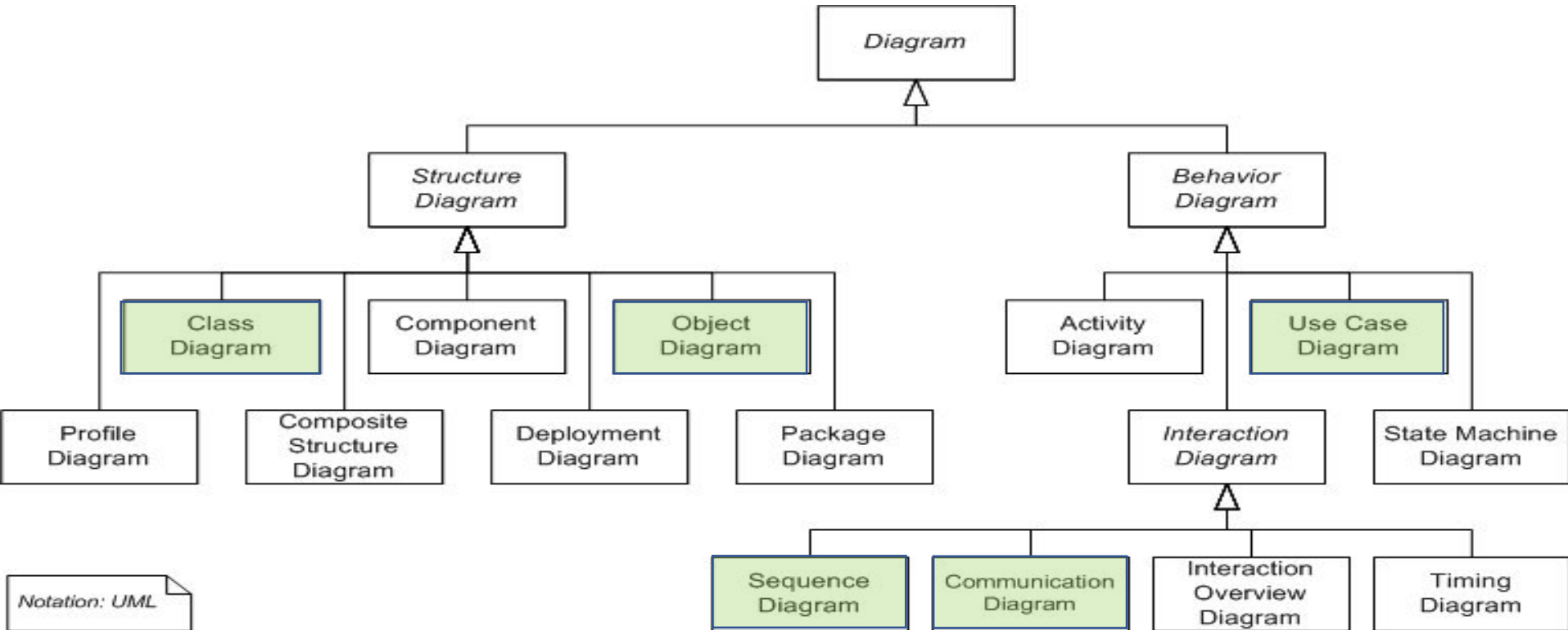
* https://en.wikipedia.org/wiki/Class_diagram

Diagram types

We will only focus on the following most frequently used diagrams in UML:

- Class Diagrams
- Object Diagrams
- Use-Case Diagrams
- Sequence Diagrams
- Communication Diagrams

UML 2.x Diagrams



* https://en.wikipedia.org/wiki/Class_diagram

Relationships in UML

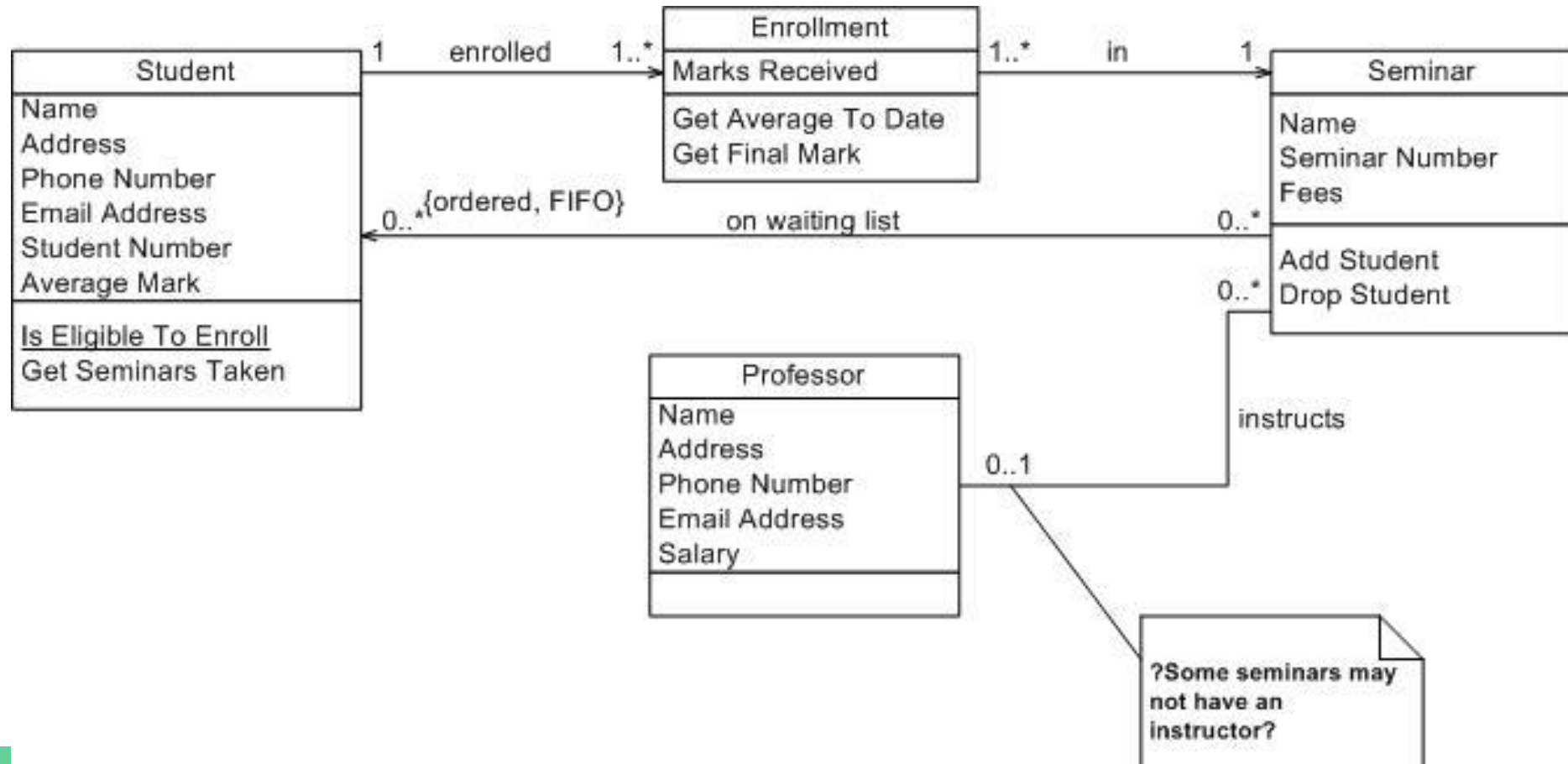
A relationship is a **connection** between things.

Graphically, a relationship is rendered as a path, with different kinds of lines to distinguish the different relationships.

Kinds of relationships in UML:

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

Let's contextualize in the form of **class diagrams**



But first ... what about those use cases we made?

Object identification!

Analyze use cases

- Nouns → suggest classes
- Verbs → suggest methods
- (Rough outline for first iteration)

Brainstorming!

- Initial list of classes (objects)
- Attributes, operations, relationships can be added in future iterations

Dependency relationships

Dependency indicates a “uses” relationship between two classes. In a class diagram, a dependency is rendered as a dashed directed line.



If a class A “uses” class B, then one or more of the following statements is generally true:

1. B is a local variable type in A
2. B is a return type in A
3. B is a parameter type in A
4. A uses methods in B

Generalization relationship

Generalization is a relationship between a general thing (called the **superclass** or **parent** class) and a more specific kind of that thing (called the **subclass** or **child** class).

Generalization is sometimes called an “**is-a**” relationship and is established through the process of inheritance.



A generalization relationship is rendered as a **solid directed line** with a **large open arrowhead** pointing to the **parent** class

Association Relationship

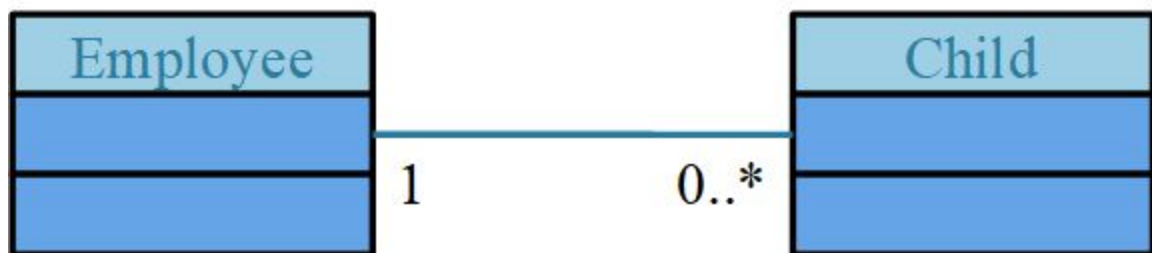
An association is a structural relationship that specifies that objects of one thing are connected to objects of another thing.





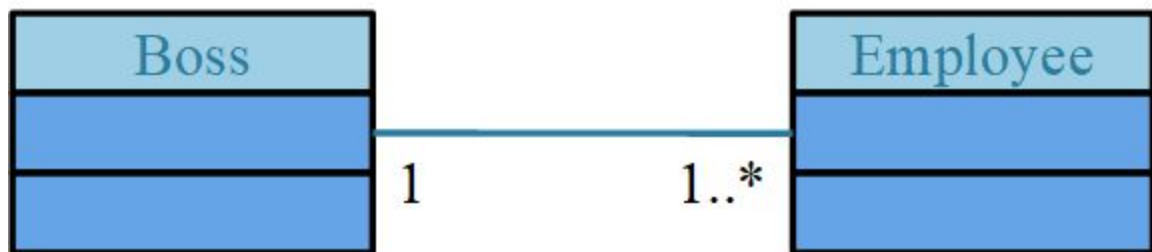
Exactly one:

A department has one and only one boss



Zero or more:

An employee has zero to many children



One or more:

A boss is responsible for one or more employees

Association Relationship: Directional

Association specifies a “has-a” or “whole/part” relationship between two classes, an association is rendered as a solid directed line.



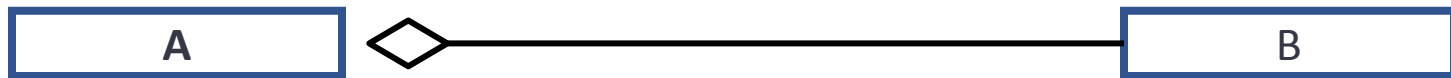
If a class A “has-a” class B, then the following statement is true:
B is a field variable type (instance or class variables) in A

Association Relationship: Aggregation, Composition

Aggregation and composition are rendered as a solid line with an open (aggregation) or filled (composition) diamond near the whole class.

Aggregation is a (conceptual) specialization of association, specifying a “whole-part” or “has-a” relationship between two objects.

- Distinguish “whole” from “part”
- Does not change the meaning of navigation across association
- Does not link the lifetimes of the whole and its parts



Association Relationship: Aggregation, Composition

Composition is a form of aggregation with strong ownership:

- Parts are created after the composite itself
- An object may be a part of only one composite at a time
- The whole is responsible for creation and destruction of its parts



Examples

Generalization (*is a kind of*):

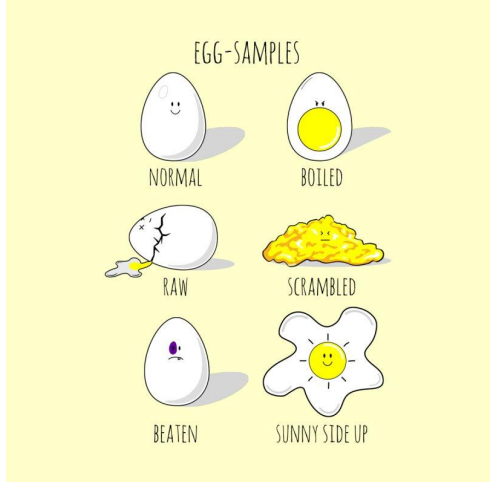
- Cardinal *is a kind of* Bird
- Truck *is a kind of* Land Vehicle (which *is a kind of* Vehicle)

Aggregation (**logical** *is a part of*):

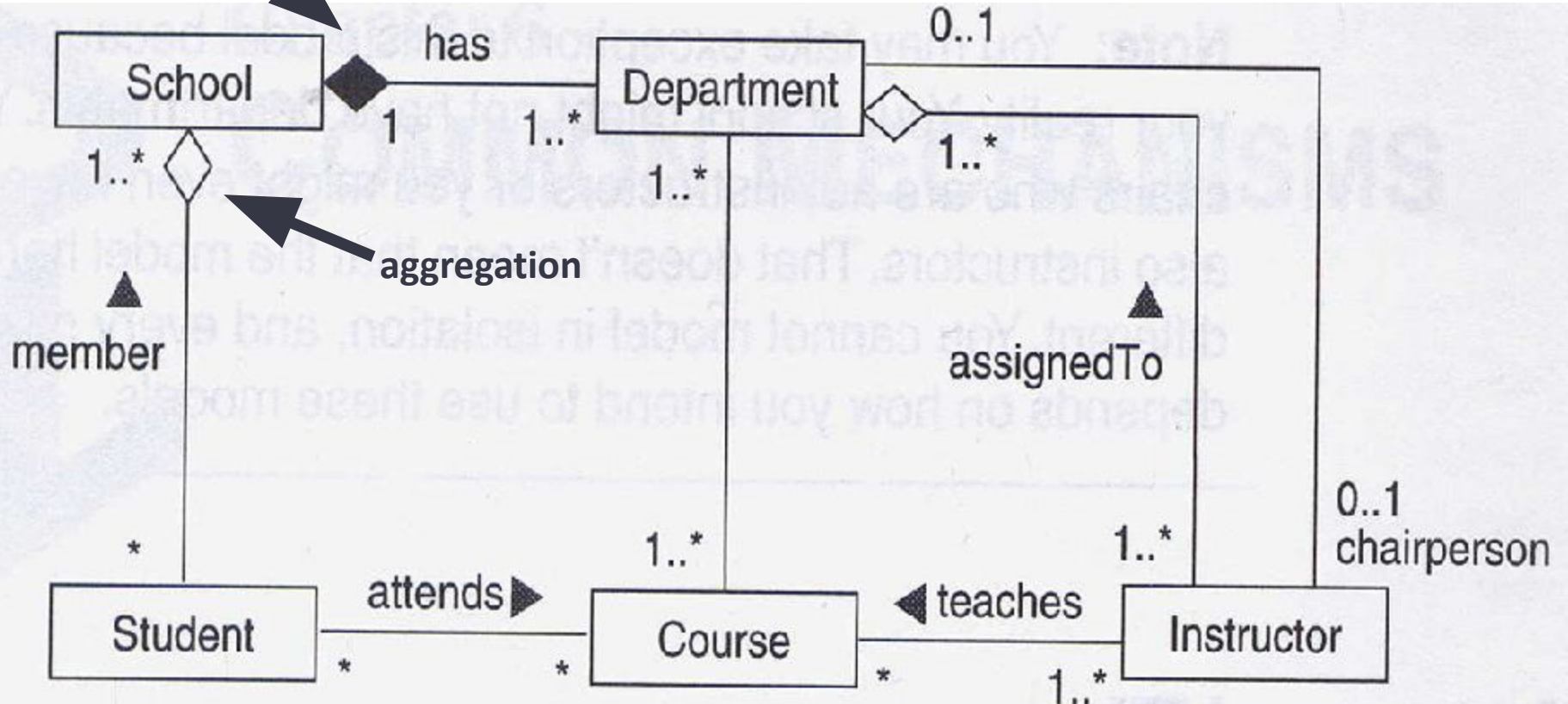
- Wheel instance *is a part of* a Vehicle instance

Composition (**physical** *is a part of*):

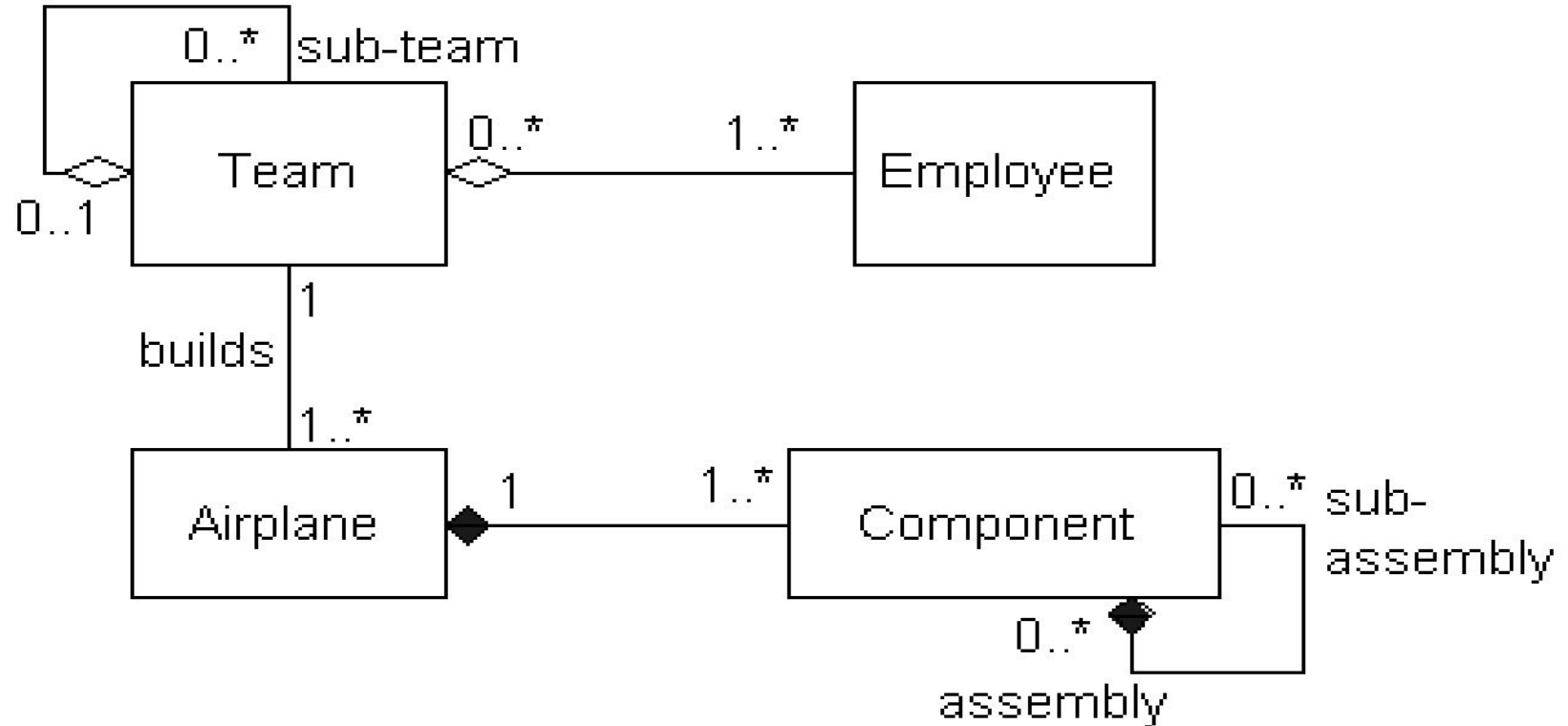
- Door instance *is a part of* only a single Vehicle instance



Association Examples



Association examples

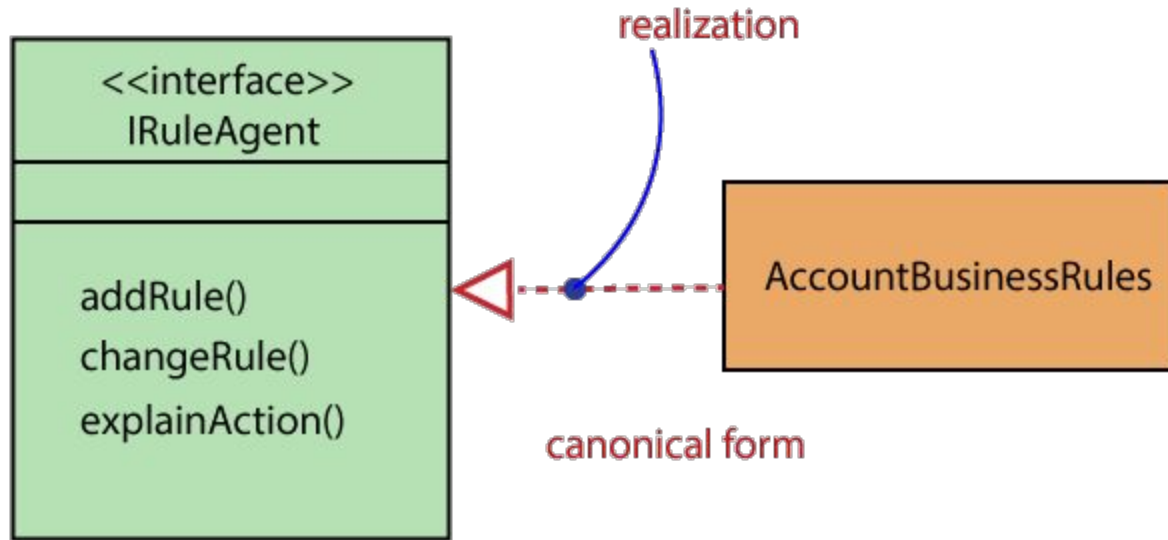


Realization

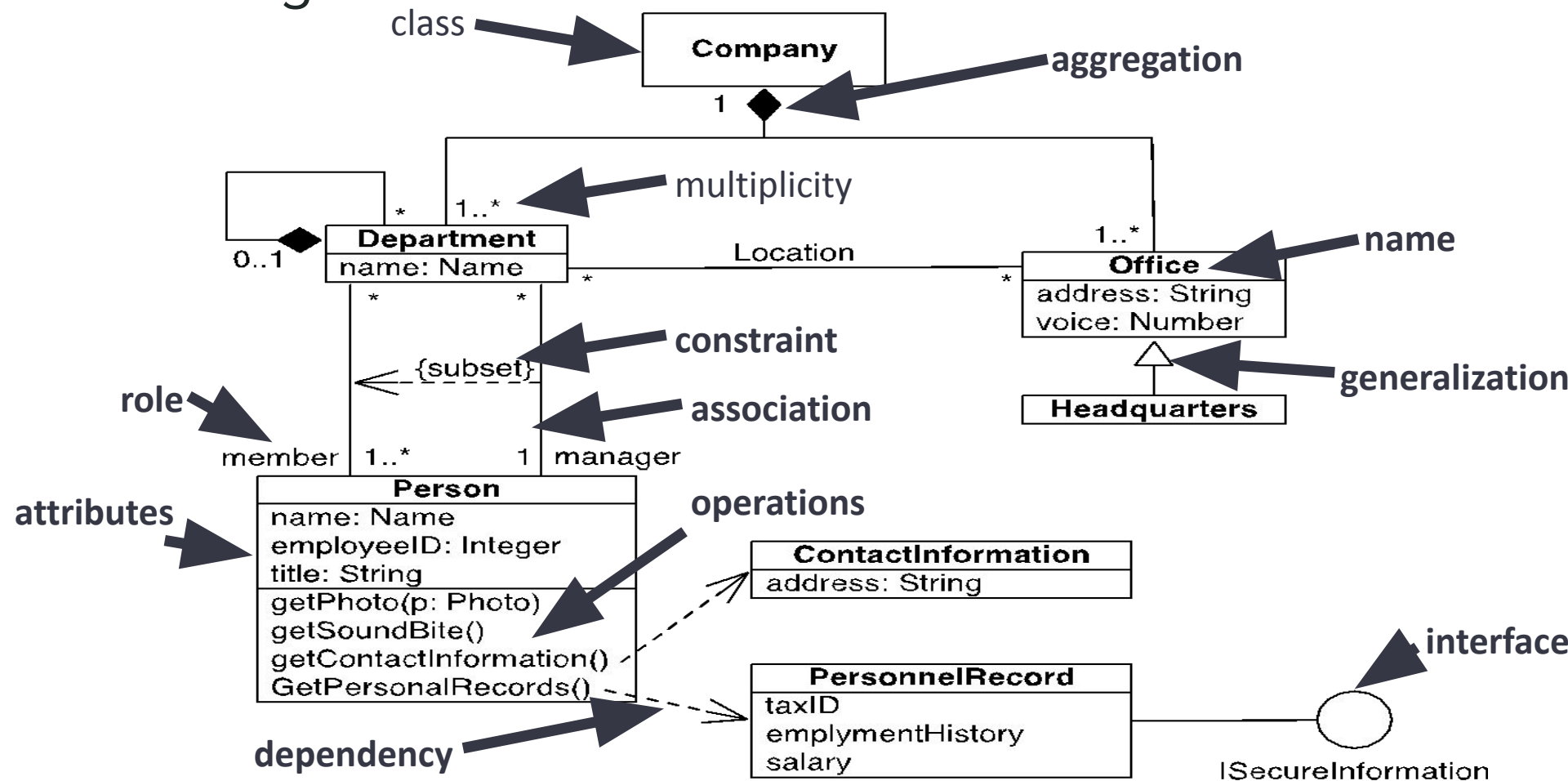
A realization is a relationship between two things where one thing (an **interface**) specifies a contract that another thing (a **class**) guarantees to carry out by implementing the operations specified in that contract



A realization relationship is rendered as a dashed directed line with an open arrowhead pointing to the interface



Class Diagrams



Class diagrams

When would we use class diagrams?

Would the type of software process we use change when we use class diagrams?

What are the pros / cons of generating class diagrams from existing code?



Object diagrams

An object diagram shows a set of **objects and their relationships** at a **point in time**.

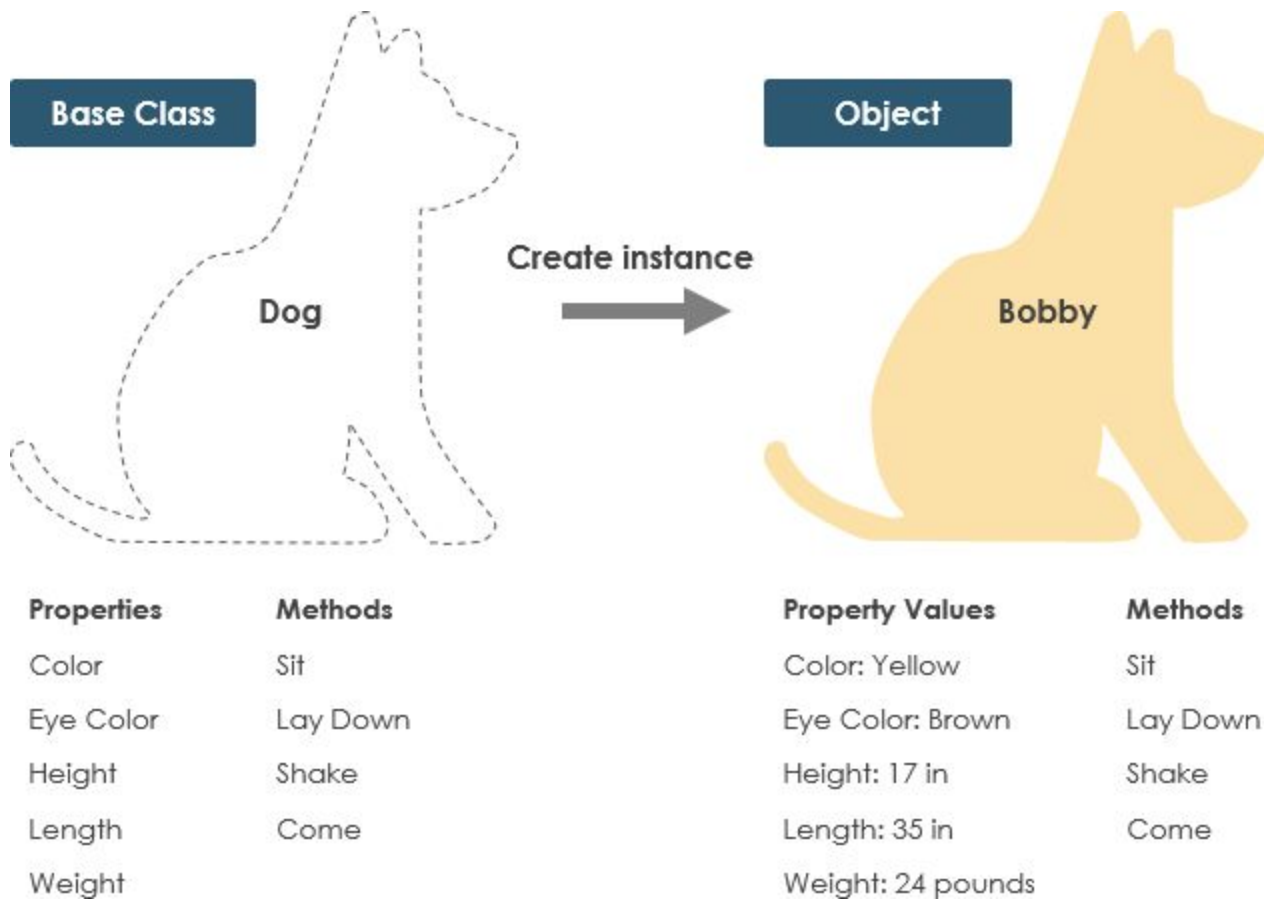
An object diagram covers a set of **instances of the things found in a class diagram**.

- It is essentially an instance of a class diagram

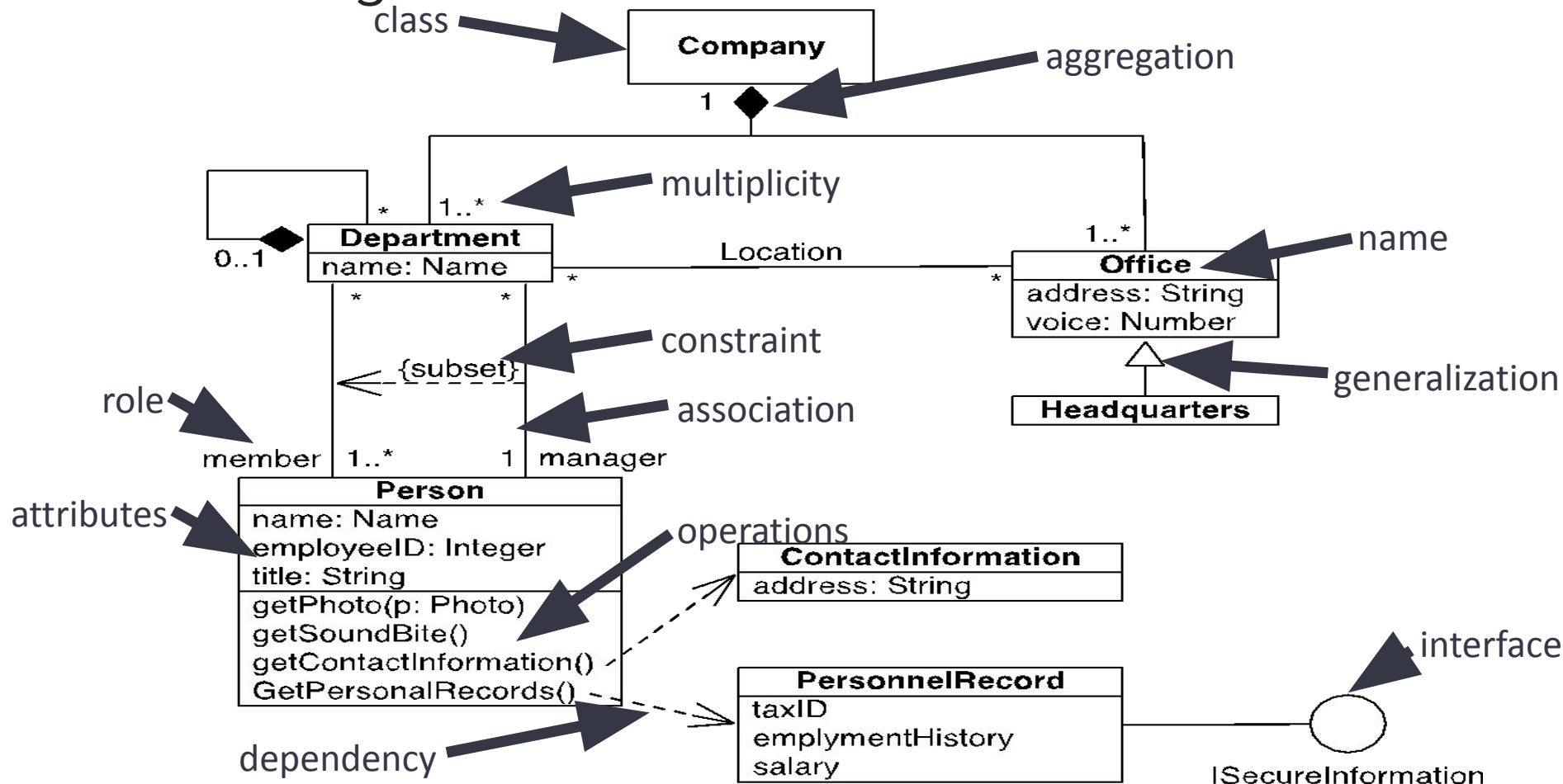
An object diagram expresses the **static part of an interaction**, consisting of objects that collaborate, but without any of the messages (i.e., method calls) passed among them.

An object diagram provides a **snapshot of the objects** in a system at a given **moment in time**.

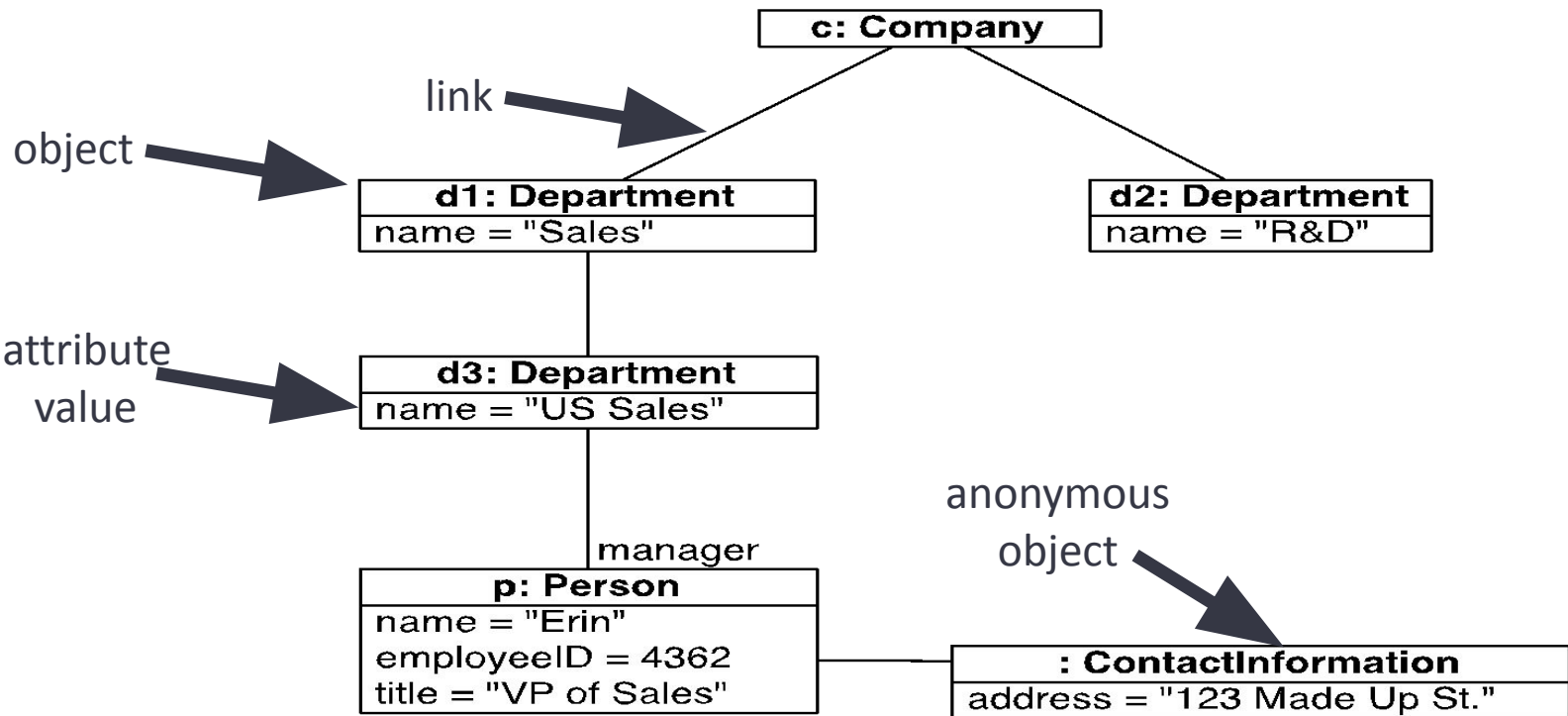
Typically used to model object structures



Class Diagrams



Object Diagram: Example



Object diagrams

When would we use object diagrams?

Would the type of software process we use change when we use object diagrams?

How many object diagrams are there for each class diagram?

Sequence diagrams

A sequence diagram emphasizes the **time ordering** of messages

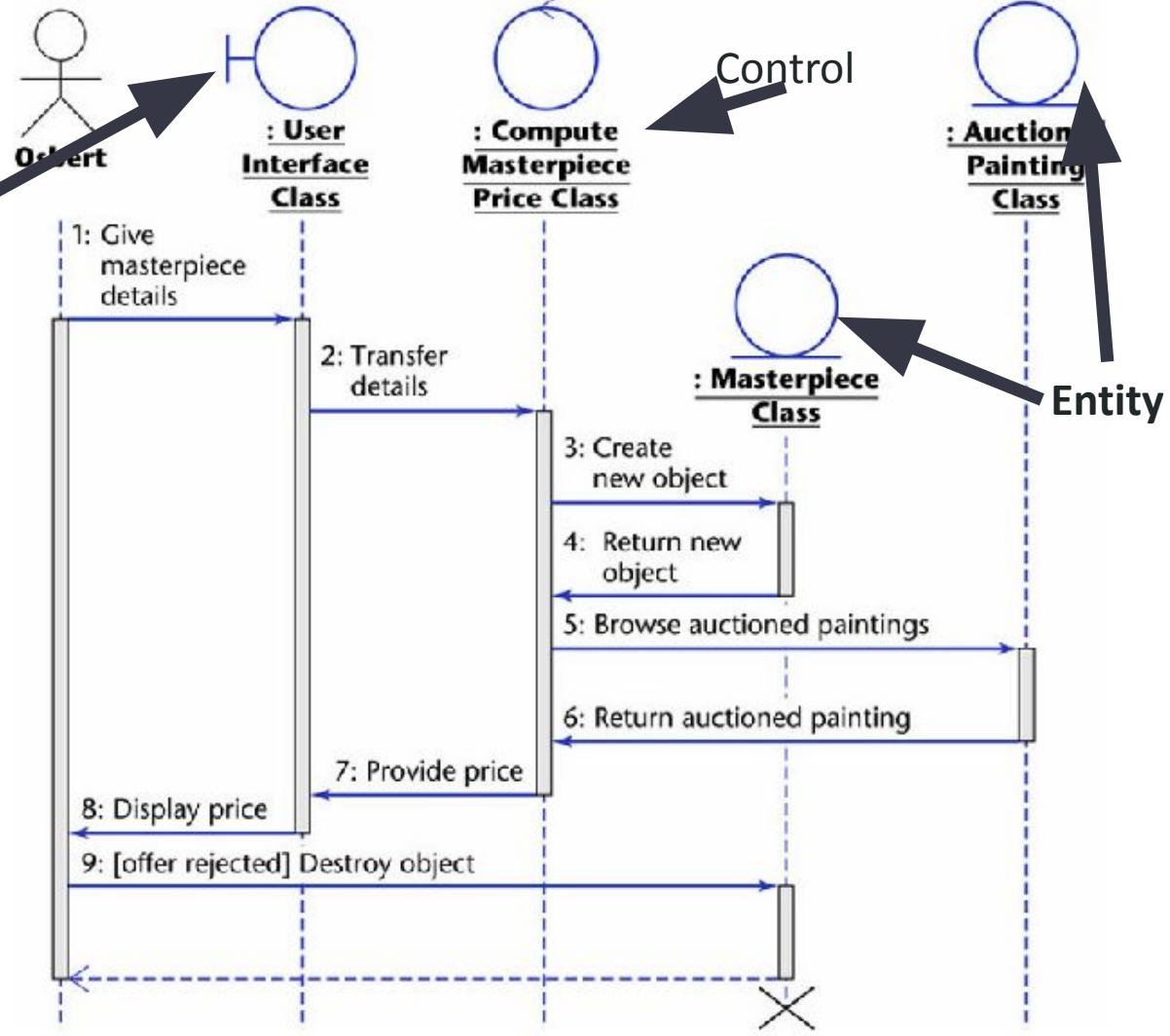
Graphically, a sequence diagram is a table that shows objects arranged along the x axis and messages, ordered in increasing time, along the y axis.

Two features that distinguish sequence diagrams from collaboration diagrams:

- **Object lifeline** – a vertical dashed line that represents the existence of an object over a period of time.
- **Focus of control** – a tall, thin rectangle that shows the period of time during which an object is performing an action.

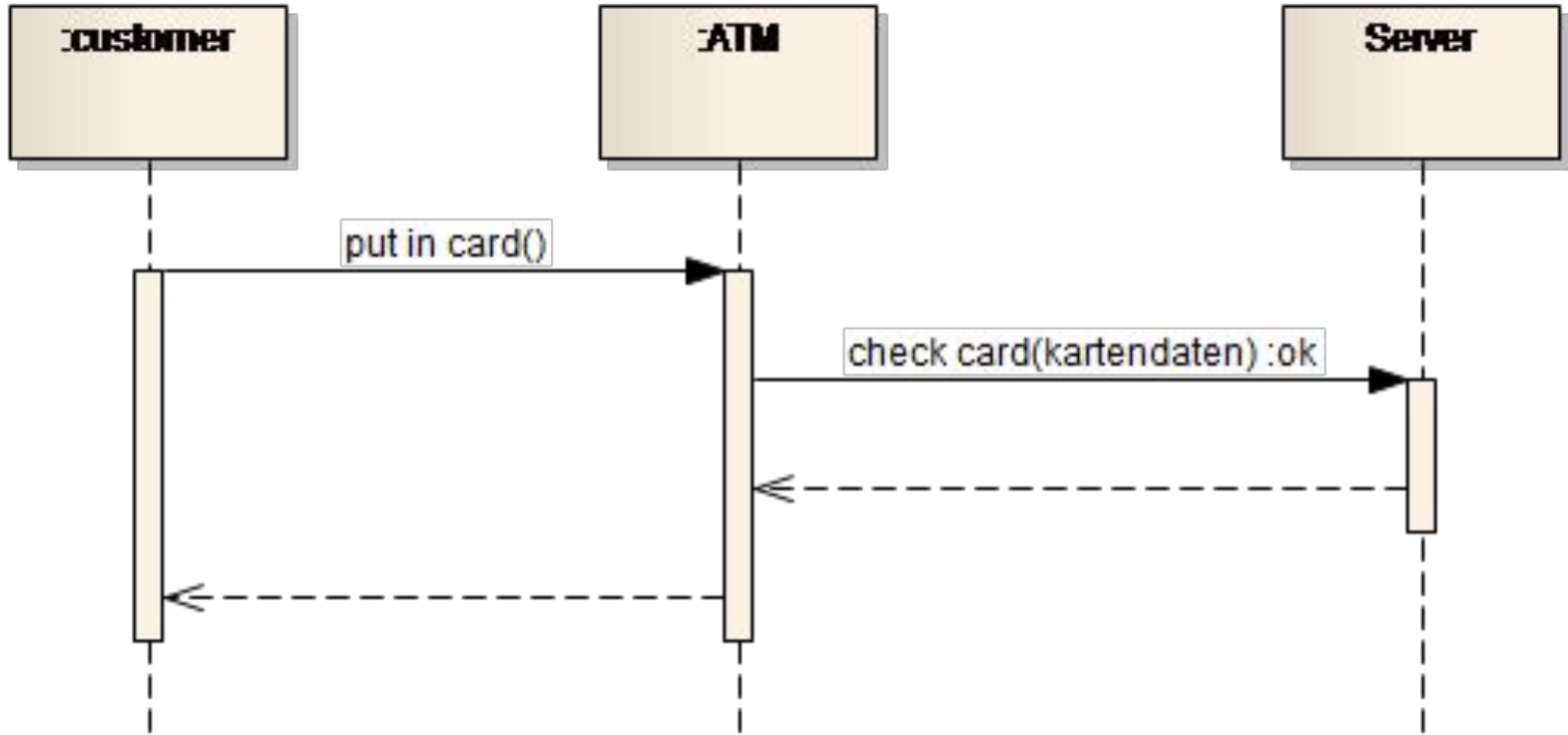
Sequence Diagram: Example Art Auction


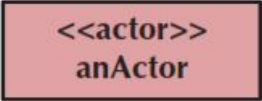
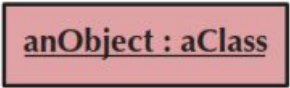

Boundary


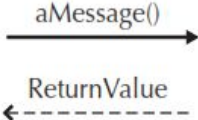
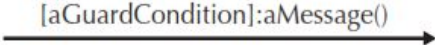

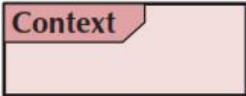


http://www.utm.mx/~caff/doc/OpenUPWeb/openup/guidances/guidelines/entity_control_boundary_pattern_C4047897.html

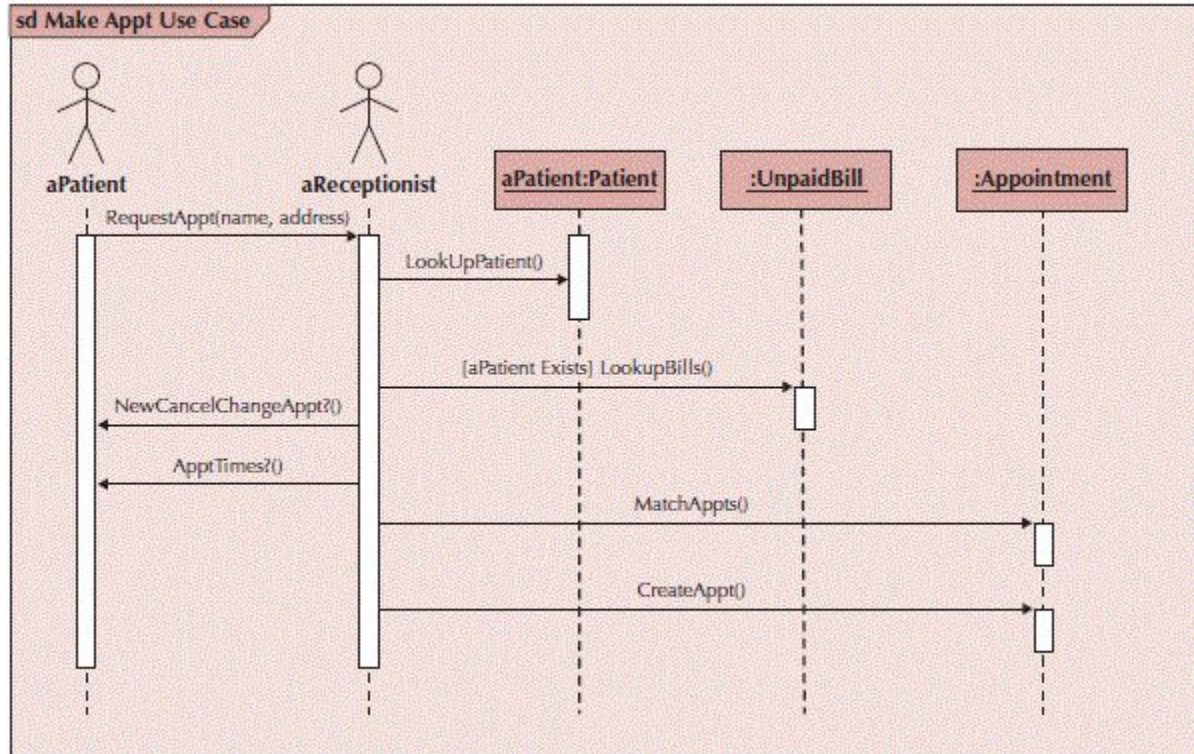
Takeaway: **communication between objects**



Term and Definition	Symbol
<p>An actor:</p> <ul style="list-style-type: none"> ■ Is a person or system that derives benefit from and is external to the system. ■ Participates in a sequence by sending and/or receiving messages. ■ Is placed across the top of the diagram. ■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with <<actor>> in it (alternative). 	 <p>anActor</p> 
<p>An object:</p> <ul style="list-style-type: none"> ■ Participates in a sequence by sending and/or receiving messages. ■ Is placed across the top of the diagram. 	
<p>A lifeline:</p> <ul style="list-style-type: none"> ■ Denotes the life of an object during a sequence. ■ Contains an X at the point at which the class no longer interacts. 	

<p>An execution occurrence:</p> <ul style="list-style-type: none">■ Is a long narrow rectangle placed atop a lifeline.■ Denotes when an object is sending or receiving messages.	
<p>A message:</p> <ul style="list-style-type: none">■ Conveys information from one object to another one.■ A operation call is labeled with the message being sent and a solid arrow, whereas a return is labeled with the value being returned and shown as a dashed arrow.	 <pre>graph LR; A -- "aMessage()" --> B; B -.- "ReturnValue" --> A;</pre>
<p>A guard condition:</p> <ul style="list-style-type: none">■ Represents a test that must be met for the message to be sent.	 <pre>graph LR; A -- "[aGuardCondition]:aMessage()" --> B;</pre>
<p>For object destruction:</p> <ul style="list-style-type: none">■ An X is placed at the end of an object's lifeline to show that it is going out of existence.	
<p>A frame:</p> <ul style="list-style-type: none">■ Indicates the context of the sequence diagram.	

Takeaway: **sequential interaction**



A PROCESS (for building sequence diagrams)

Set the context

Identify actors and objects that interact in the use-case scenario

Set the lifeline for each object

Add messages by drawing arrows

- Shows how they are passed from one object to another
- Include any parameters in parentheses
- Obvious return values are excluded

A PROCESS (for building sequence diagrams)

Add execution occurrence to each object's lifeline

Validate the sequence diagram

- Ensures that it depicts all of the steps in the process



Communication diagram

A communication (previously aka collaboration) diagram emphasizes the organization of objects that participate in an interaction

Two features that distinguish communication diagrams from sequence diagrams:

- **Path** – To show how one object is linked to another, a path stereotype can be attached to the far end of a link
- **Sequence number** – To indicate the time order of a message, each message is prefixed with a number

Sequence and communication diagrams are semantically equivalent

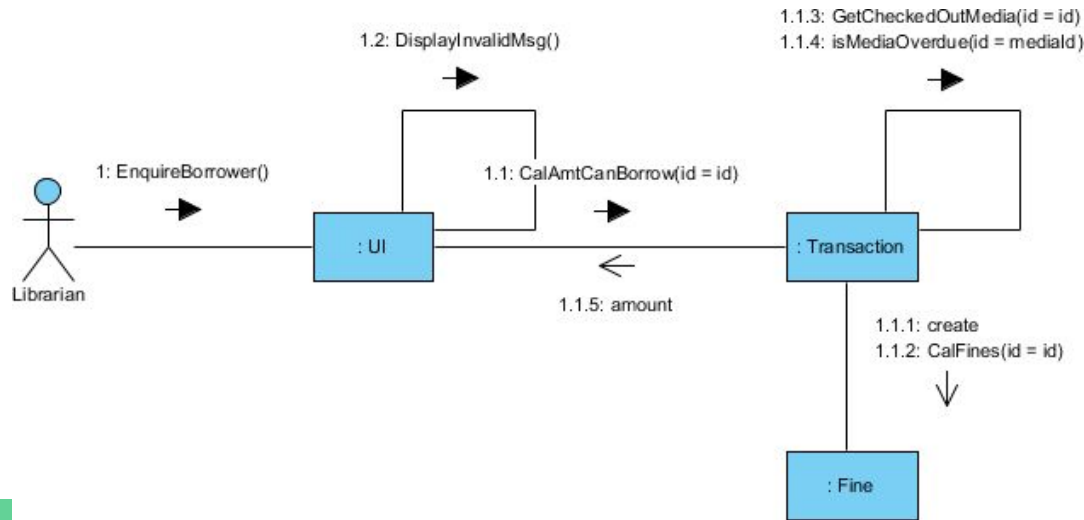
You can take a diagram in one form and convert it to the other without any loss of information

Communication diagrams

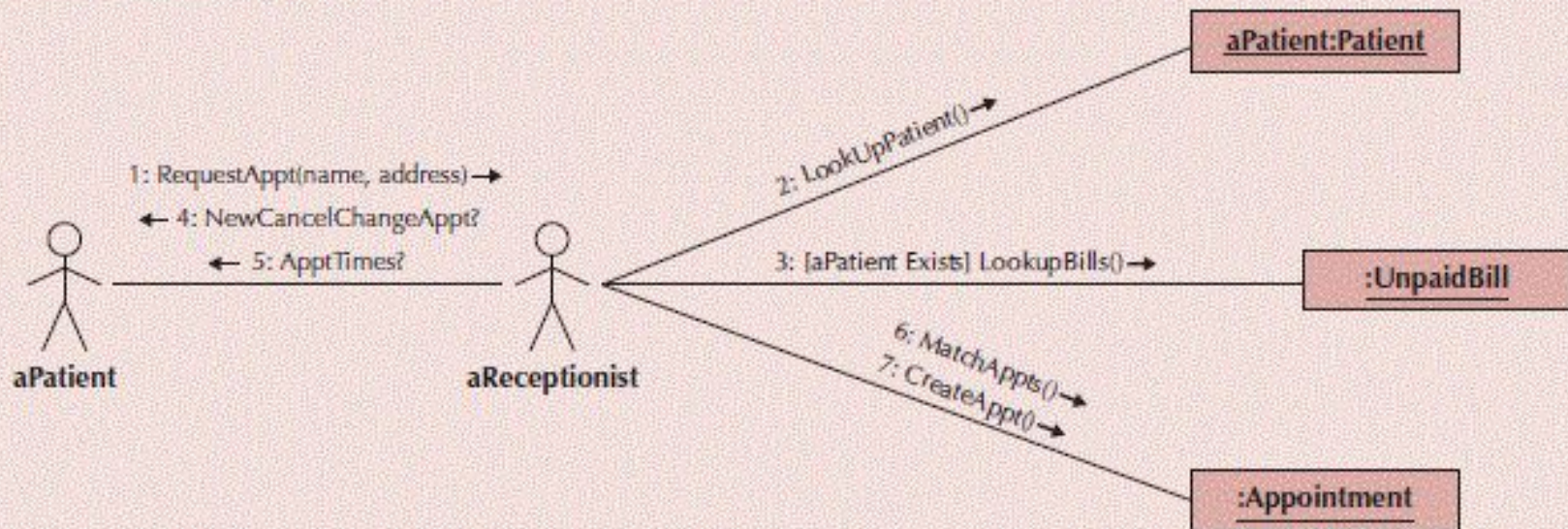
Depict the dependencies among the objects

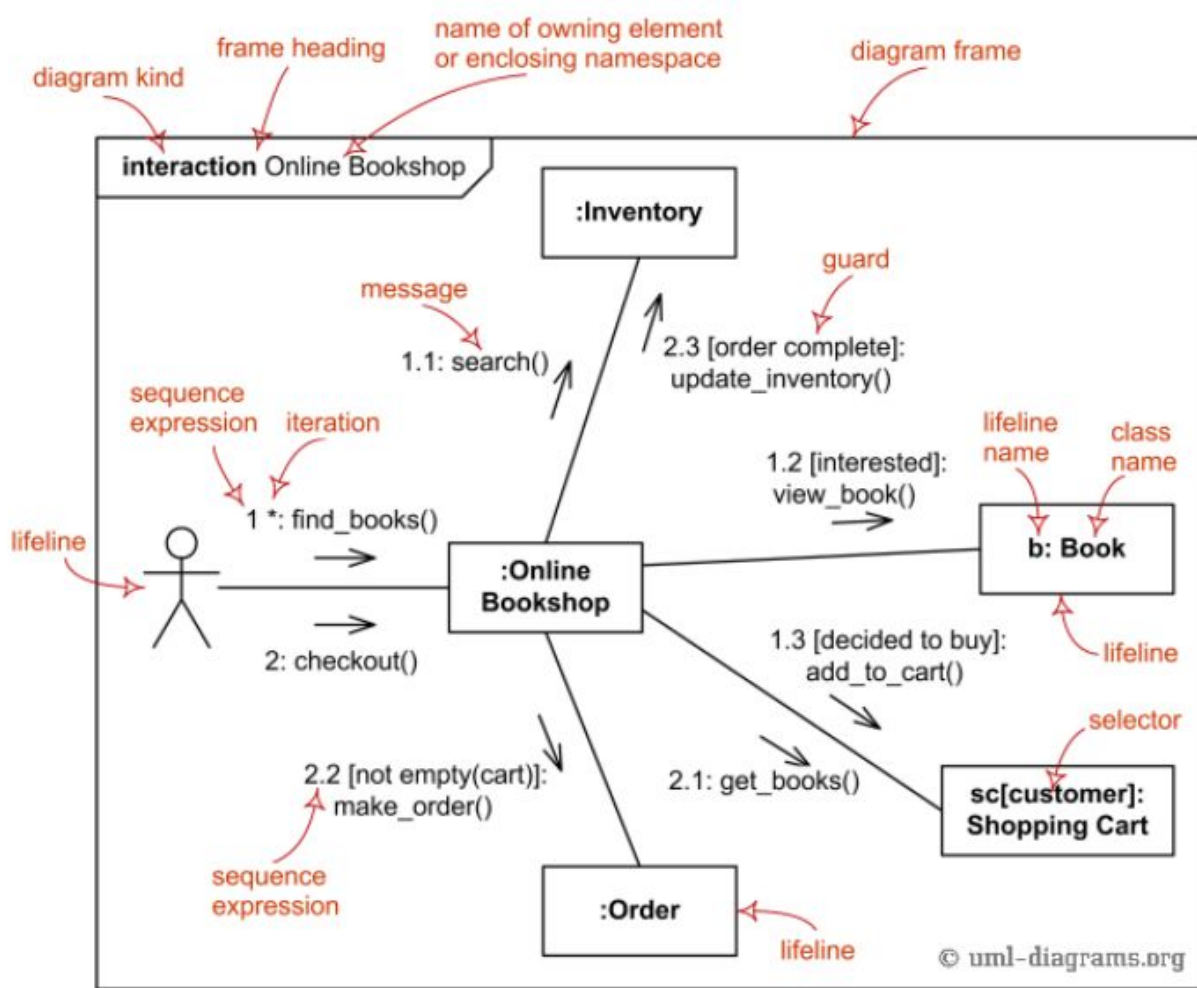
An object diagram that shows message passing relationships

Emphasize the flow through a set of objects



sd Make Appt Use Case





The major elements of UML communication diagram.

Building communications diagrams

Set the context

Identify objects, actors and associations between them

Lay out the diagram

Add the messages

Validate the model