Class Diagram

In software engineering, 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.

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

A UML class diagram is made up of:

- A set of classes and
- A set of relationships between classes

What is a Class

A description of a group of objects all with similar roles in the system, which consists of:

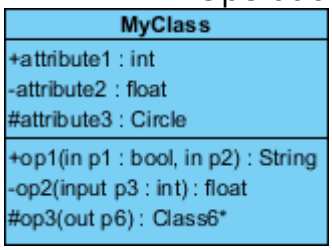
- **Structural features** (attributes) define what objects of the class "know"
    - Represent the state of an object of the class
    - Are descriptions of the structural or static features of a class
- **Behavioral features** (operations) define what objects of the class "can do"
    - Define the way in which objects may interact
    - Operations are descriptions of behavioral or dynamic features of a class

Class Notation

A class notation consists of three parts:

1. **Class Name**

2. The name of the class appears in the first partition.
3. **Class Attributes**
    - Attributes are shown in the second partition.
    - The attribute type is shown after the colon.
    - Attributes map onto member variables (data members) in code.
4. **Class Operations** (Methods)
    - Operations are shown in the third partition. They are services the class provides.
    - The return type of a method is shown after the colon at the end of the method signature.
    - The return type of method parameters is shown after the colon following the parameter name.
    - Operations map onto class methods in code



The graphical representation of the class - MyClass as shown above:

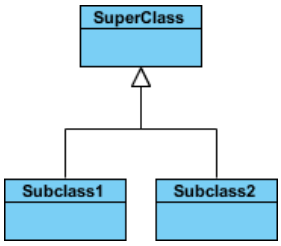
- MyClass has 3 attributes and 3 operations
- Parameter p3 of op2 is of type int
- op2 returns a float
- op3 returns a pointer (denoted by a *) to Class6

Class Relationships

A class may be involved in one or more relationships with other classes. A relationship can be one of the following types: (Refer to the figure on the right for the graphical representation of relationships).

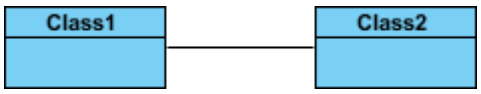
Relationship Type	Graphical Re

Inheritance (or Generalization):



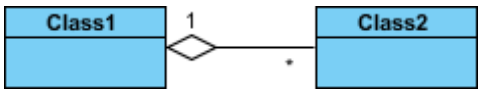
- Represents an "is-a" relationship.
- An abstract class name is shown in italics.
- SubClass1 and SubClass2 are specializations of Super Class.
- A solid line with a hollow arrowhead that point from the child to the parent class

Simple Association:



- A structural link between two peer classes.
- There is an association between Class1 and Class2
- A solid line connecting two classes

Aggregation:



A special type of association. It represents a "part of" relationship.

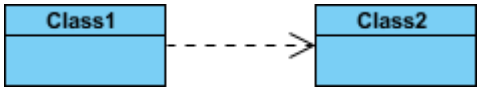
- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.
- A solid line with an unfilled diamond at the association end connected to the class of composite

Composition:

A special type of aggregation where parts are destroyed when the whole is destroyed.

- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.
- A solid line with a filled diamond at the association connected to the class of composite

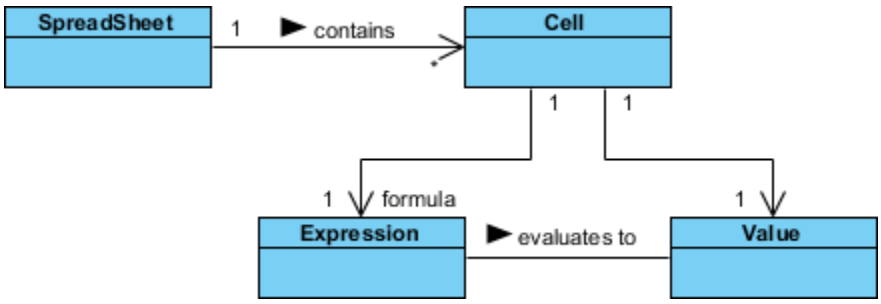
Dependency:



- Exists between two classes if the changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on Class2
- A dashed line with an open arrow

Relationship Names

- Names of relationships are written in the middle of the association line.
- Good relation names make sense when you read them out loud:
  - "Every spreadsheet **contains** some number of cells",
  - "an expression **evaluates to** a value"
- They often have a **small arrowhead to show the direction** in which direction to read the relationship, e.g., expressions evaluate to values, but values do not evaluate to expressions.



Relationship - Roles

- A role is a directional purpose of an association.
- Roles are written at the ends of an association line and describe the purpose played by that class in the relationship.
    - E.g., A cell is related to an expression. The nature of the relationship is that the expression is the **formula** of the cell.

Navigability

The arrows indicate whether, given one instance participating in a relationship, it is possible to determine the instances of the other class that are related to it.

The diagram above suggests that,

- Given a spreadsheet, we can locate all of the cells that it contains, but that
    - we cannot determine from a cell in what spreadsheet it is contained.
- Given a cell, we can obtain the related expression and value, but
    - given a value (or expression) we cannot find the cell of which those are attributes.

Visibility of Class attributes and Operations

In object-oriented design, there is a notation of visibility for attributes and operations. UML identifies four types of visibility: **public**, **protected**, **private**, and **package**.

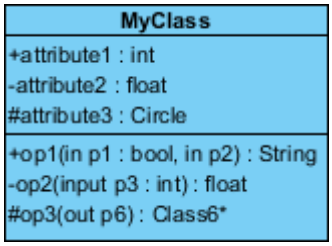
The +, -, # and ~ symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

- + denotes public attributes or operations
- - denotes private attributes or operations
- # denotes protected attributes or operations

- ~ denotes package attributes or operations

Access Right	public (+)	private (-)	protected (#)	Package (~)
Members of the same class	yes	yes	yes	yes
Members of derived classes	yes	no	yes	yes
Members of any other class	yes	no	no	in same package

Class Visibility Example



In the example above:

- attribute1 and op1 of MyClassName are public
- attribute3 and op3 are protected.
- attribute2 and op2 are private.

Access for each of these visibility types is shown below for members of different classes.

Multiplicity

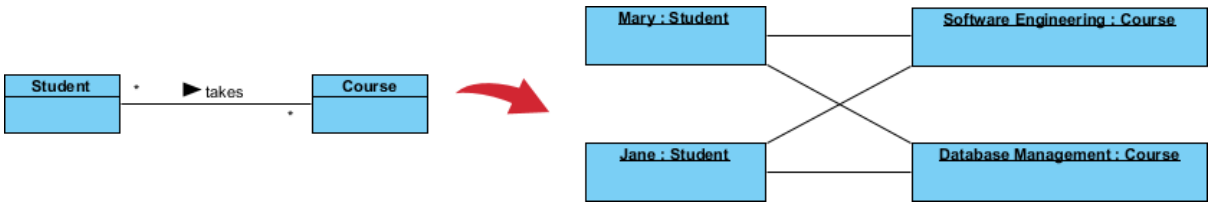
How many objects of each class take part in the relationships and multiplicity can be expressed as:

- Exactly one - 1
- Zero or one - 0..1
- Many - 0..* or *
- One or more - 1..*
- Exact Number - e.g. 3..4 or 6
- Or a complex relationship - e.g. 0..1, 3..4, 6.* would mean any number of objects other than 2 or 5

Multiplicity Example

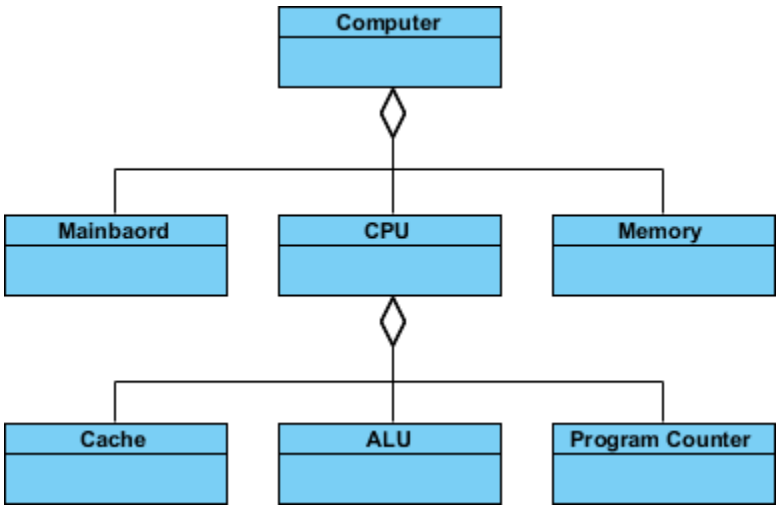
- Requirement: A Student can take many Courses and many Students can be enrolled in one Course.

- In the example below, the **class diagram** (on the left), describes the statement of the requirement above for the static model while the object diagram (on the right) shows the snapshot (an instance of the class diagram) of the course enrollment for the courses Software Engineering and Database Management respectively)



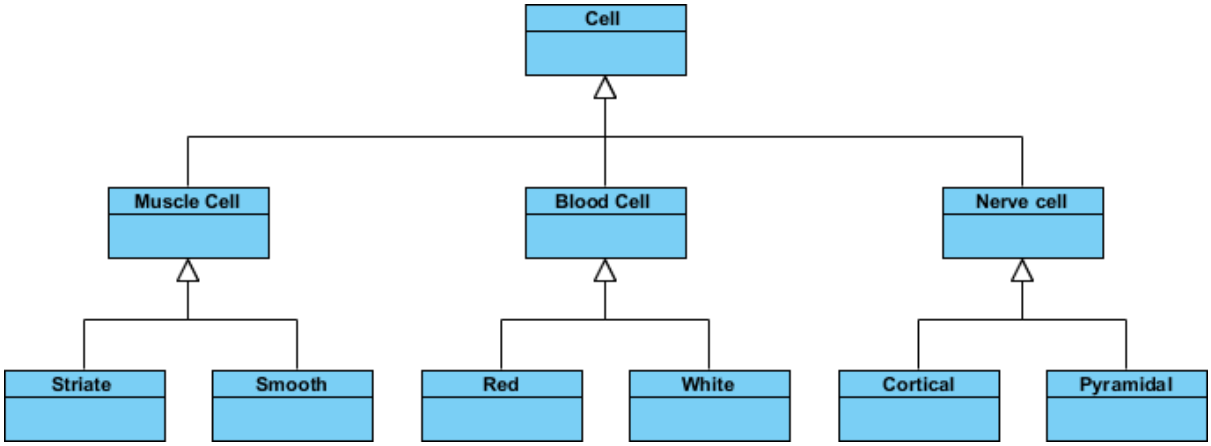
Aggregation Example - Computer and parts

- An aggregation is a special case of association denoting a "consists-of" hierarchy
- The aggregate is the parent class, the components are the children classes



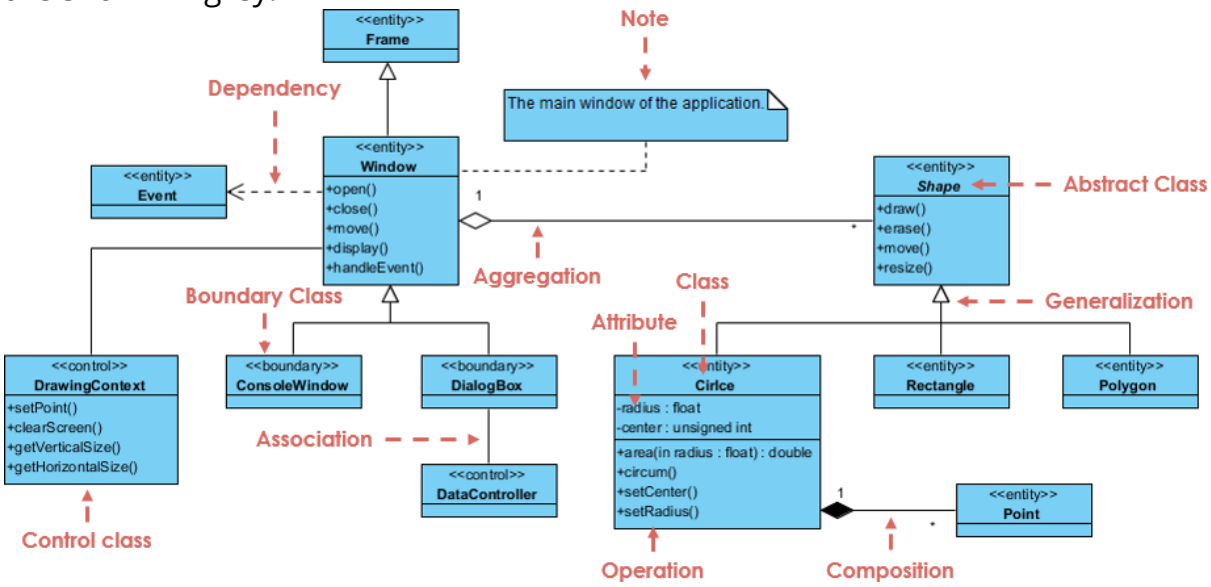
Inheritance Example - Cell Taxonomy

- Inheritance is another special case of an association denoting a "kind-of" hierarchy
- Inheritance simplifies the analysis model by introducing a taxonomy
- The child classes inherit the attributes and operations of the parent class.



Class Diagram - Diagram Tool Example

A class diagram may also have notes attached to classes or relationships. Notes are shown in grey.



In the example above:

We can interpret the meaning of the above class diagram by reading through the points as following.

1. Shape is an abstract class. It is shown in Italics.
2. Shape is a superclass. Circle, Rectangle and Polygon are derived from Shape. In other words, a Circle is-a Shape. This is a generalization / inheritance relationship.
3. There is an association between DialogBox and DataController.
4. Shape is part-of Window. This is an aggregation relationship. Shape can exist without Window.

5. Point is part-of Circle. This is a composition relationship. Point cannot exist without a Circle.
6. Window is dependent on Event. However, Event is not dependent on Window.
7. The attributes of Circle are radius and center. This is an entity class.
8. The method names of Circle are area(), circum(), setCenter() and setRadius().
9. The parameter radius in Circle is an in parameter of type float.
10. The method area() of class Circle returns a value of type double.
11. The attributes and method names of Rectangle are hidden. Some other classes in the diagram also have their attributes and method names hidden.

Dealing with Complex System - Multiple or Single Class Diagram?

Inevitably, if you are modeling a large system or a large business area, there will be numerous entities you must consider. Should we use multiple or a single class diagram for modeling the problem? The answer is:

- Instead of modeling every entity and its relationships on a single class diagram, it is better to use multiple class diagrams.
- Dividing a system into multiple class diagrams makes the system easier to understand, especially if each diagram is a graphical representation of a specific part of the system.

Perspectives of Class Diagram in Software Development Lifecycle

We can use class diagrams in different development phases of a **software development lifecycle** and typically by modeling class diagrams in three different perspectives (levels of detail) progressively as we move forward:

Conceptual perspective: The diagrams are interpreted as describing things in the real world. Thus, if you take the conceptual perspective you draw a diagram that represents the concepts in the domain under study. These concepts will

naturally relate to the classes that implement them. The conceptual perspective is **considered language-independent**.

Specification perspective: The diagrams are interpreted as describing software abstractions or components with specifications and interfaces but with no commitment to a particular implementation. Thus, if you take the specification perspective we are **looking at the interfaces of the software**, not the implementation.

Implementation perspective: The diagrams are interpreted as describing software implementations in a particular technology and **language**. Thus, if you take the implementation perspective we are **looking at the software implementation**.

2. Object Diagram

Object is an instance of a class in a particular moment in runtime that can have its own state and data values. Likewise a static **UML** object diagram is an instance of a **class diagram**; it shows a snapshot of the detailed state of a system at a point in time, thus an object diagram encompasses objects and their relationships which may be considered a special case of a class diagram or a **communication diagram**.

Purpose of Object Diagram

The use of object diagrams is fairly limited, mainly to show examples of data structures.

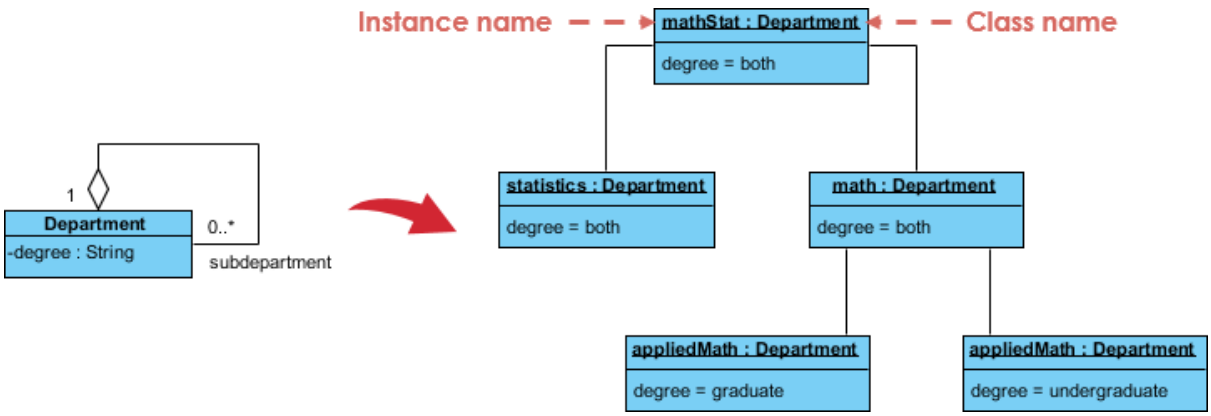
- During the analysis phase of a project, you might create a class diagram to describe the structure of a system and then create a set of object diagrams as test cases to verify the accuracy and completeness of the class diagram.
- Before you create a class diagram, you might create an object diagram to discover facts about specific model elements and their links, or to illustrate specific examples of the classifiers that are required.

Object Diagram at a Glance

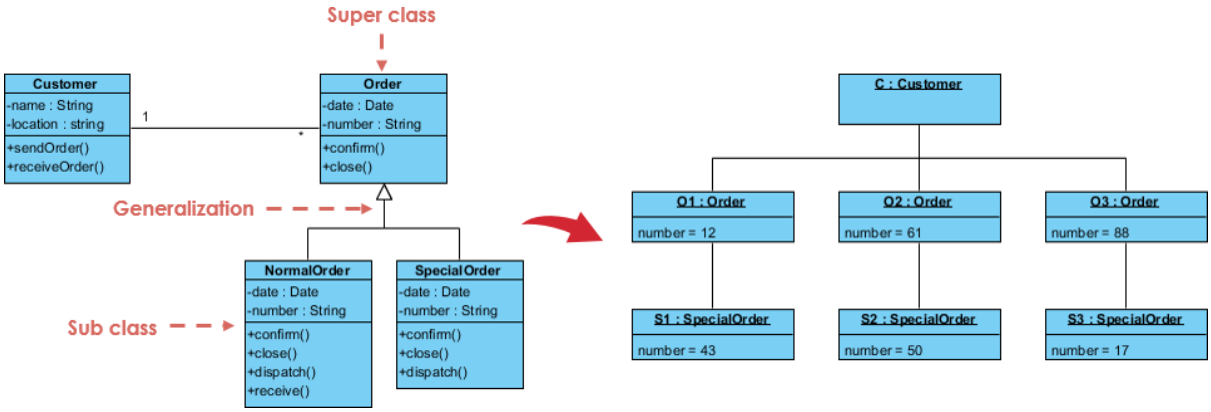
An object diagram shows this relation between the instantiated classes and the defined class, and the relation between these objects in the system. They are be useful to explain smaller portions of your system, when your system class diagram is very complex, and also sometimes modeling recursive relationship in diagram.

The best way to illustrate what an object diagram look like is to show the object diagram derived from the corresponding class diagram.

The following Order Management System shows their relationships. This small class diagram shows that a university Department can contain lots of other Departments and the object diagram below instantiates the class diagram, replacing it by a concrete example.



Class to Object Diagram Example - Order System



Basic Object Diagram Symbols and Notations

Object Names:

- Every object is actually symbolized like a rectangle, that offers the name from the objec and its class underlined as well as divided with a colon.

Object Attributes:

- Similar to classes, you are able to list object attributes inside a separate compartment. However, unlike classes, object attributes should have values assigned for them.

Links:

- Links tend to be instances associated with associations. You can draw a link while using the lines utilized in class diagrams.

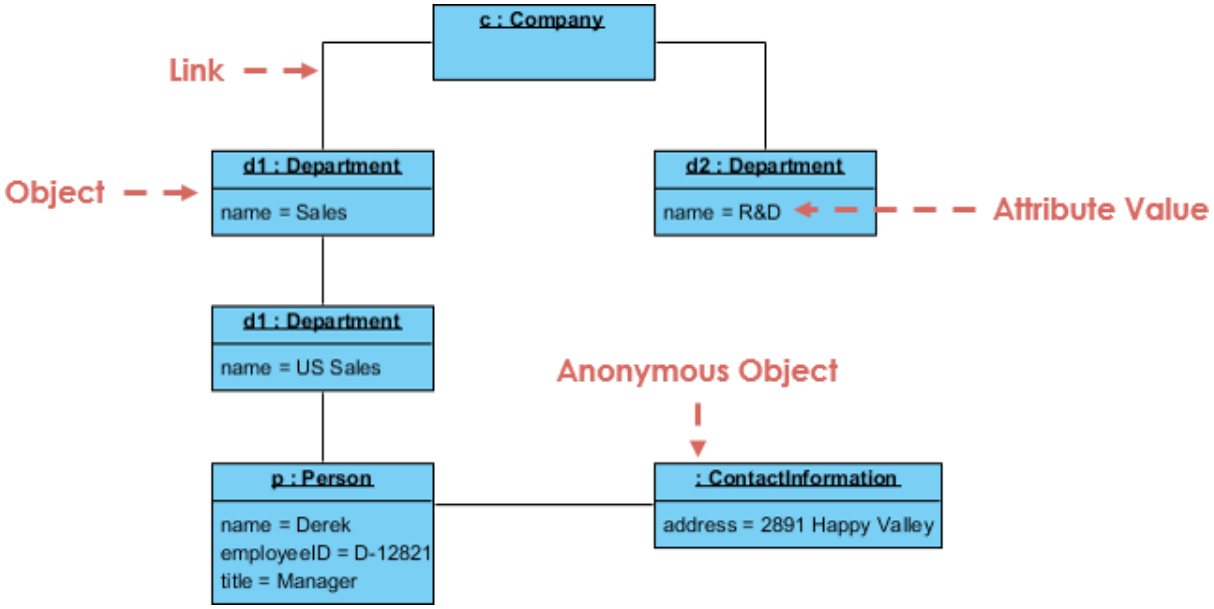
Class Diagram vs. Object Diagram

In UML, object diagrams provide a snapshot of the instances in a system and the relationships between the instances. By instantiating the model elements in a class diagram, you can explore the behavior of a system at a point in time.

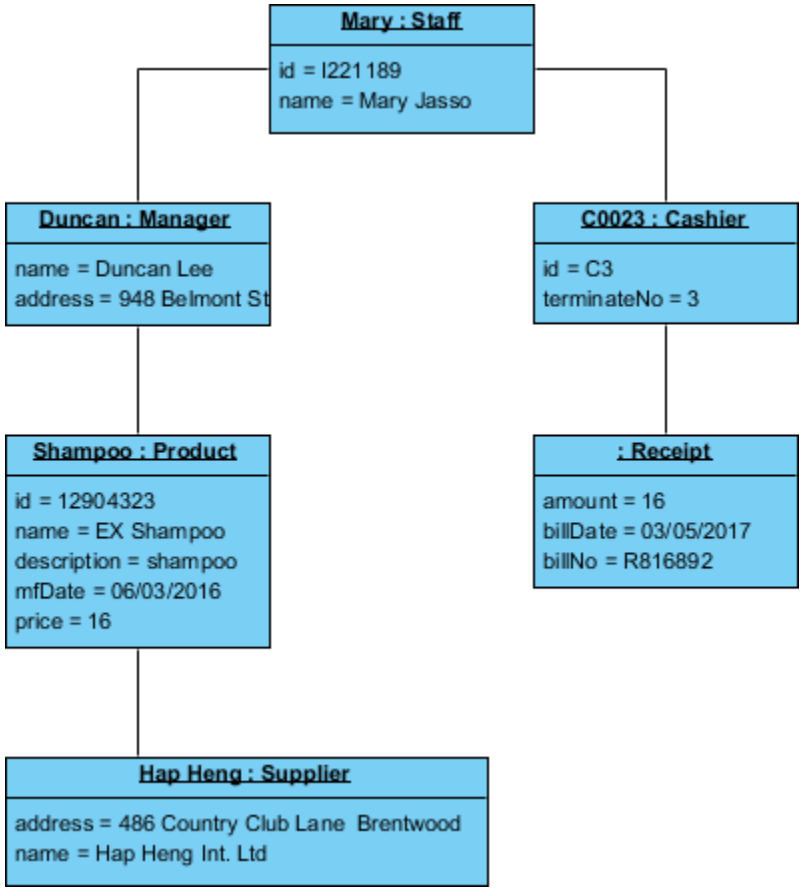
- An object diagram is a UML structural diagram that shows the instances of the classifiers in models.
- Object diagrams use notation that is similar to that used in class diagrams.
- Class diagrams show the actual classifiers and their relationships in a system
- Object diagrams show specific instances of those classifiers and the links between those instances at a point in time.
- You can create object diagrams by instantiating the classifiers in class, deployment, component, and use-case diagrams.

Object Diagrams - Learn by Examples

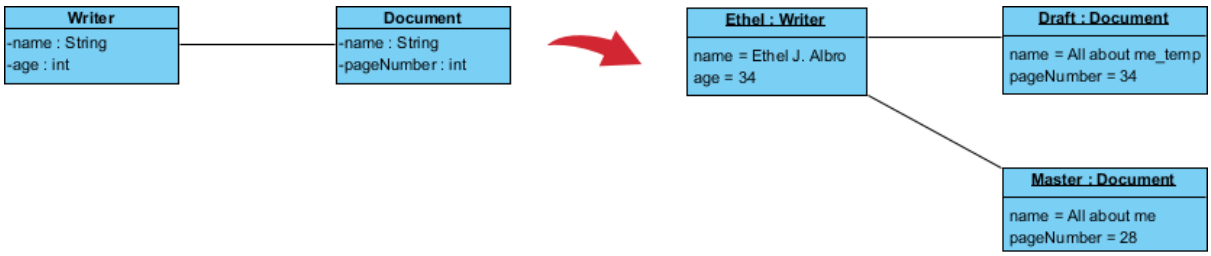
Object Diagram Example I - Company Structure



Object Diagram Example II - POS



Object Diagram Example III - Writer



Class Diagram Object Diagram

Steps for Modeling Object Structures

A communication diagram (called collaboration in previous version of UML) without messages is also known as an object diagram, and the relationships between objects are called links. An object diagram must be a valid instantiation of a static class diagram. Objects must have classes and links between objects must be instances of associations between classes. This can be used as a quick consistency check. To do this we can develop an object diagram by using the following steps:

- Identify the **mechanism** you'd like to model. A mechanism represents some **functions** or **behaviors** of the part of the system you are **modeling that results from the interaction of a society of classes, interfaces, and other things**.
- For each mechanism, identify the classes, interfaces, and other elements that participate in this collaboration; identify the relationships among these things, as well.
- Consider one scenario that walks through this mechanism. Freeze that scenario at a moment in time, and render each object that participates in the mechanism.
- Expose the state and attribute values of each such object, as necessary, to understand the scenario.

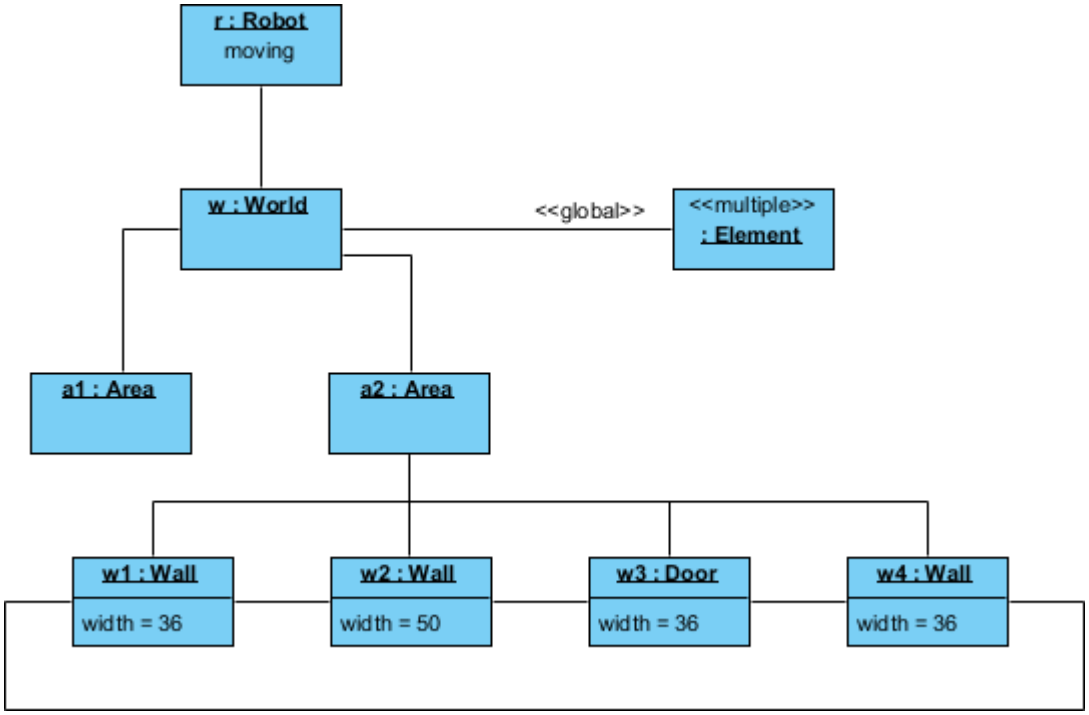
Object Diagram Example IV - Robot Moving Behavior

Similarly, expose the links among these objects, representing instances of associations among them.

As this figure indicates, one object represents the robot itself (r, an instance of Robot), and r is currently in the state marked moving. This object has a link to w, an instance of World, which represents an abstraction of the robot's world

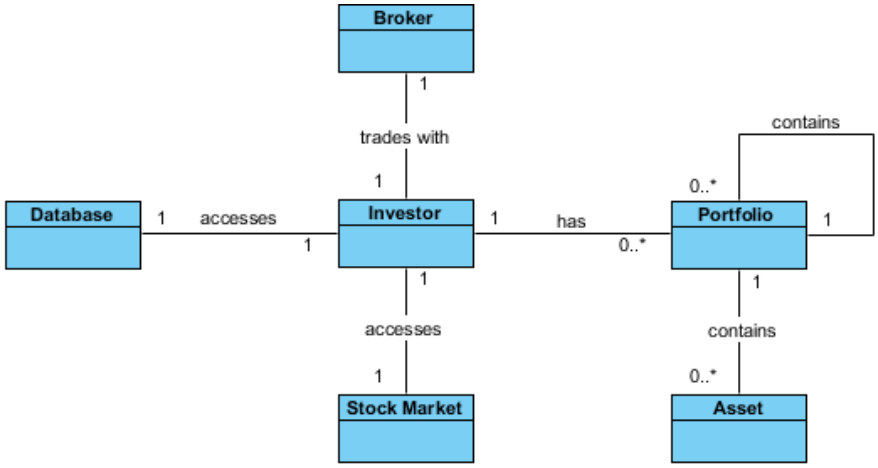
model. This object has a link to a multiple objects that consists of instances of Element, which represent entities that the robot has identified but not yet assigned in its world view. These elements are marked as part of the robot's global state.

At this moment in time, w is linked to two instances of Area. One of them (a2) is shown with its own links to three Wall and one Door object. Each of these walls is marked with its current width, and each is shown linked to its neighboring walls. As this object diagram suggests, the robot has recognized this enclosed area, which has walls on three sides and a door on the fourth.

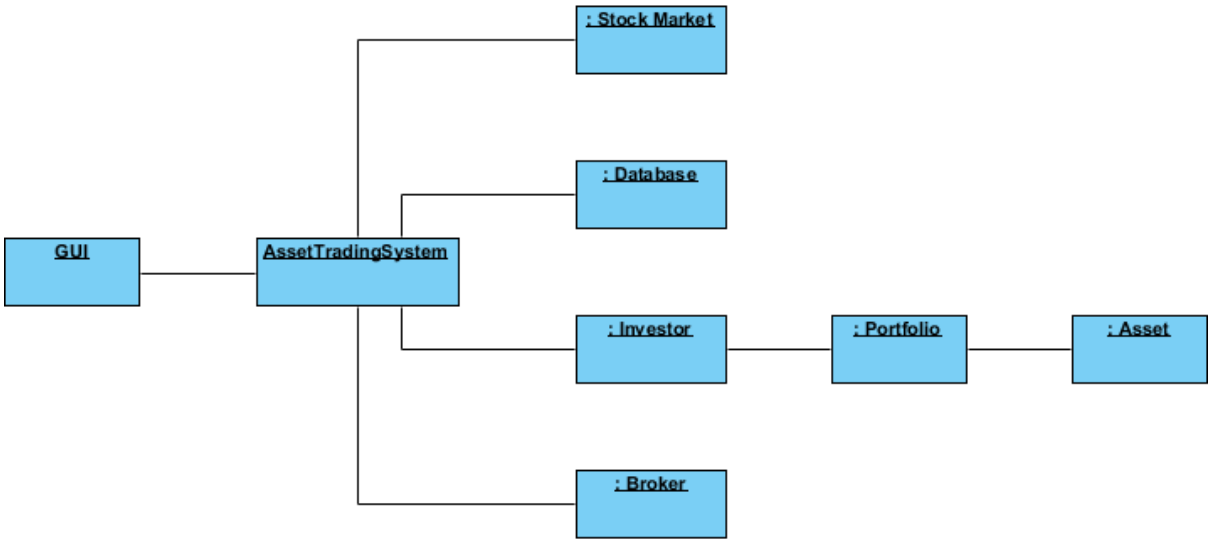


Object Diagram Example V - Deriving an Object Structure Similar to Communication Diagram

Besides showing the objects' state at a particular point in time, an object diagram can also be used to represent the occurrences of interactions between classes in runtime. The result looks a bit like a communication diagram. The figure below provides an example of such a class diagram and its corresponding object diagram:



Class Diagram



Object Diagram

Communication Diagram

UML communication diagrams, like the **sequence diagrams** - a kind of interaction diagram, shows how objects interact. A communication diagram is an extension of object diagram that shows the objects along with the messages that travel from one to another. In addition to the associations among objects, communication diagram shows the messages the objects send each other.

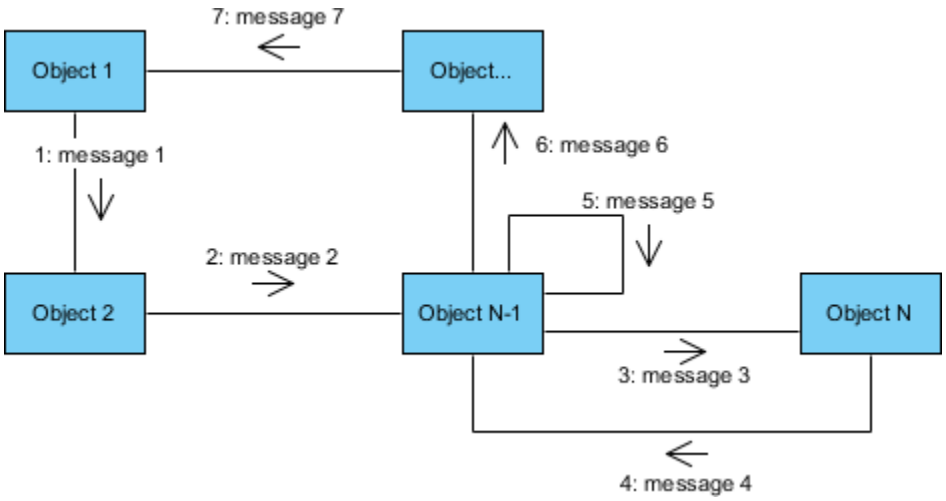
Purpose of Communication Diagram

- Model message passing between objects or roles that deliver the functionalities of use cases and operations
- Model mechanisms within the architectural design of the system
- Capture interactions that show the passed messages between objects and roles within the collaboration scenario
- Model alternative scenarios within use cases or operations that involve the collaboration of different objects and interactions
- Support the identification of objects (hence classes), and their attributes (parameters of message) and operations (messages) that participate in use cases

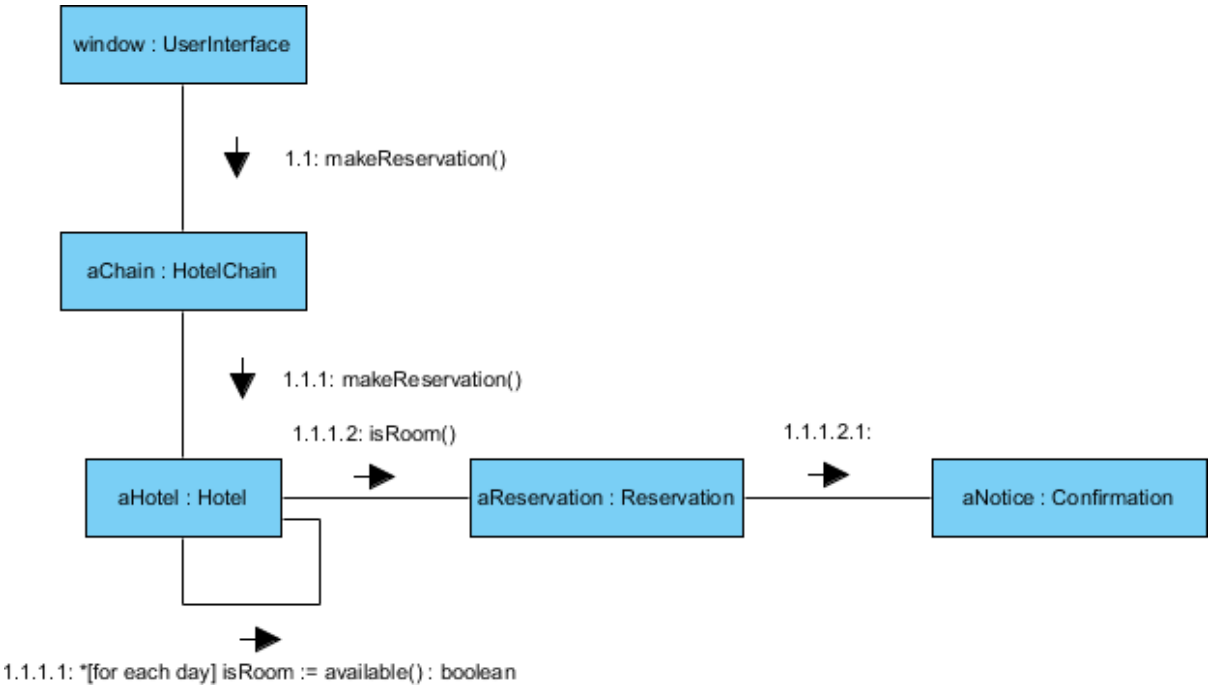
Communication Diagram at a Glance

In the example of the notation for a communication diagram, objects (actors in use cases) are represented by rectangles. In the example (generic communication diagram):

- The objects are Object1, Object2, Object..., ObjectN-1 ..., and ObjectN.
- Messages passed between objects are represented by labeled arrows that start with the sending object (actor) and end with the receiving object.
- The sample messages passed between objects are labeled 1: message1, 2: message2, 3: message3, etc., where the numerical prefix to the message name indicates its order in the sequence.
- Object1 first sends Object2 the message message1, Object2 in turn sends ObjectN-1 the message message2, and so on.
- Messages that objects send to themselves are indicated as loops (e.g., message message5).



Communication Example - Hotel Reservation



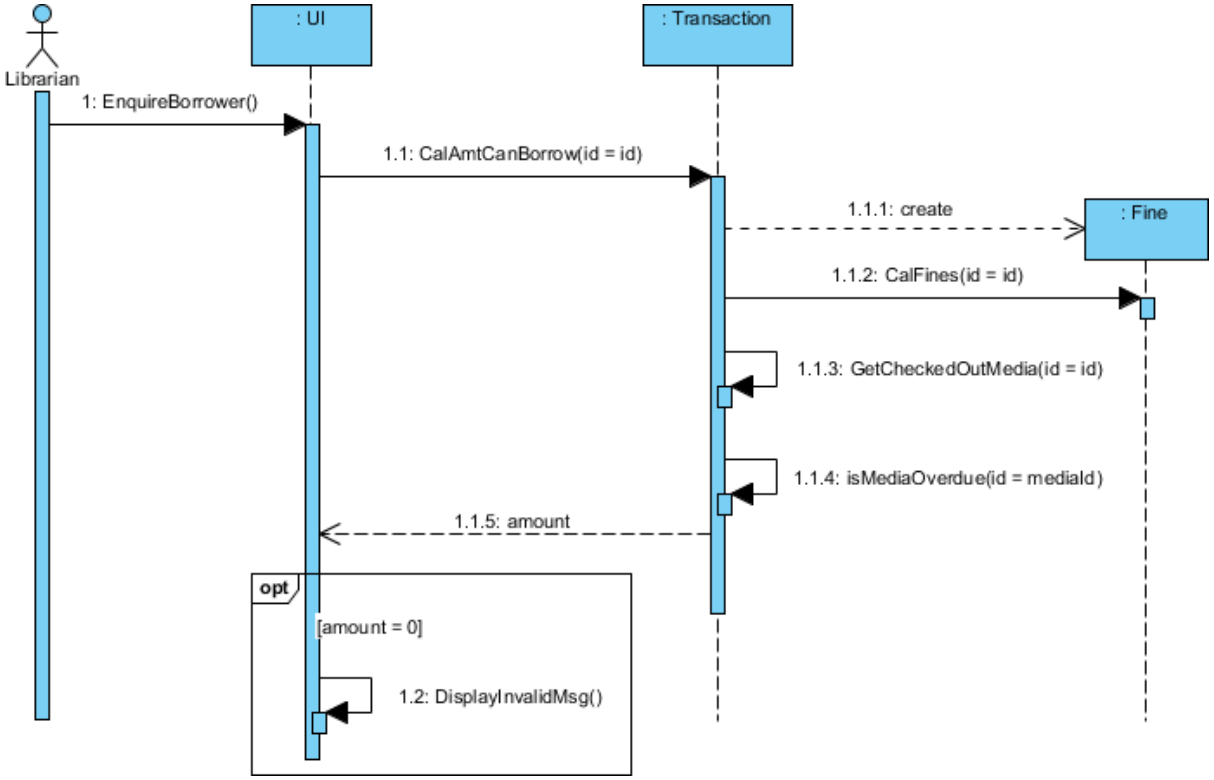
- Each message in a communication diagram has a sequence number.
- The top-level message is numbered 1.
- Messages sent during the same call have the same decimal prefix, but suffixes of 1, 2, etc. according to when they occur.

Communication Diagram vs Sequence Diagram

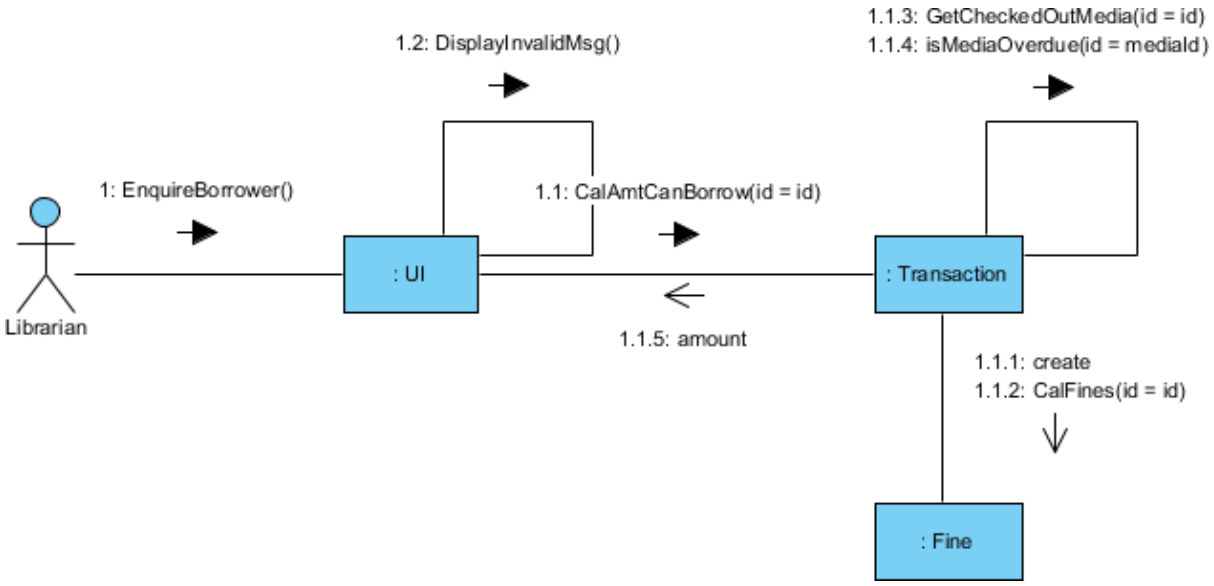
The communication diagram and the sequence diagram are similar. They're semantically equivalent, that is, the present the same information, and you can turn a communication to a sequence diagram and vice versa. The main distinction between them is that the communication diagram arranged elements according to space, the sequence diagram is according to time.

Of the two types of interaction diagrams, sequence diagrams seem to be used far more than communication diagrams. So, why would you use communication diagrams? First of all, they are very useful for visualizing the relationship between objects collaborating to perform a particular task. This is difficult to determine from a sequence diagram. In addition, communication diagrams can also help you determine the accuracy of your static model (i.e., class diagrams).

Example - Sequence diagram vs Communication (Library Item Overdue)



If you open this sequence diagram in Visual Paradigm you can automatically generate the communication diagram shown in figure below:



Note: If you compare the two diagrams, you'll see they both contain objects and messages. It also becomes clear that it's much easier to determine the time ordering of messages by looking at the sequence diagram and it's easier to see the relationships between objects by looking at the communication diagram.

Communication Diagram Elements

Objects participating in a collaboration come in two flavors: supplier and client.

- **Supplier objects** are the objects that supply the method that is being called, and therefore receive the message.
- **Client objects** call methods on supplier objects, and therefore send messages.

Links

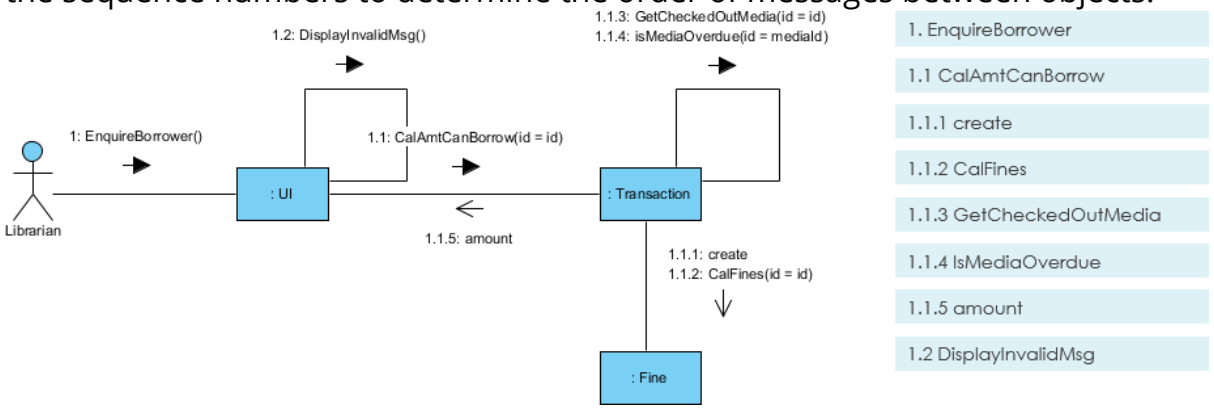
- The connecting lines drawn between objects in a communication diagram are links.
- These links are what set communication diagrams apart from sequence diagrams. They enable you to see the relationships between objects.
- Each link represents a relationship between objects and symbolizes the ability of objects to send messages to each other.
- If an object sends messages to itself, the link carrying these messages is represented as a loop icon. This loop can be seen on both the UI object and the Transaction object.

Messages in communication diagrams are shown as arrows pointing from the Client object to the Supplier object. Typically, messages represent a client

invoking an operation on a supplier object. They can be modeled along with the objects in the following manner:

- Message icons have one or more messages associated with them.
- Messages are composed of message text prefixed by a sequence number.
- This sequence number indicates the time-ordering of the message.

For example, in the communication diagram in the figure below, you can follow the sequence numbers to determine the order of messages between objects:



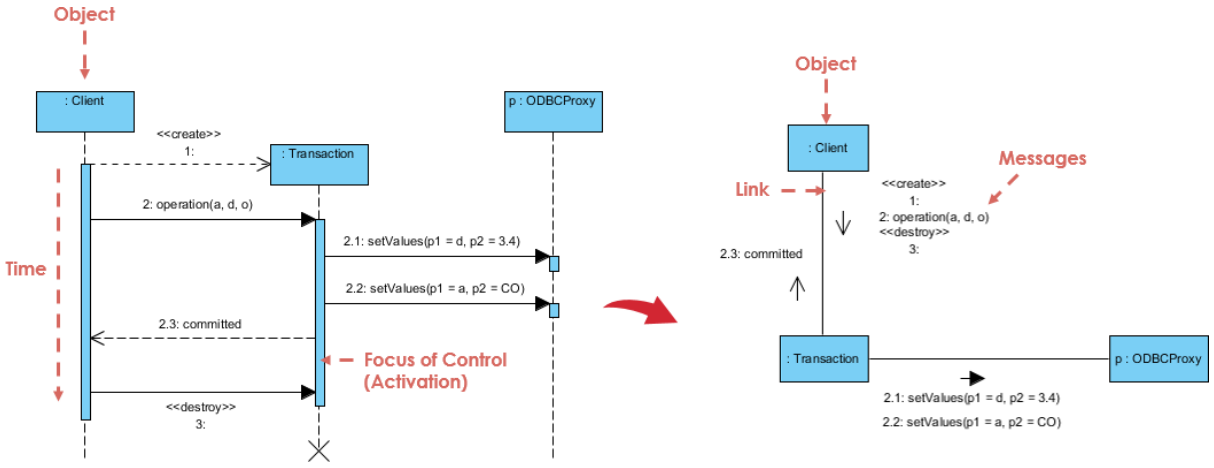
Understanding the Numbering of Messages in Communication Diagram

- The first message in a communication diagram is always numbered 1, the second is 2, and so on.
- You can indicate that a message is nested under a parent message by adding a decimal point and incremental digits to the parent's sequence number.

For example:

Based on the example above, the "CalAmtCanBorrow" message is the first nested message under "EnquireBorrower" and is given the sequence number 1.1. The second nested message under "EnquireBorrower" is "DisplayInvalidMsg", so it's given a sequence number of 1.2.

Example - From Sequence Diagram to Communication Diagram



Note that:

- **Focus of control**: also called **execution occurrence/activation**. It shows as tall, thin rectangle on a lifeline that represents **the period** during which **an element is performing an operation**.
- The top and the bottom of the rectangle are aligned with the initiation and the completion time respectively.
- In communication diagram focus of control is explicit and thus, can be represented by the message nest numbering.