

## Class Diagram

## What is a Class Diagram?

- A diagram that shows a set of classes, interfaces, and collaborations and their relationships

## Why do we need Class Diagram?

- Focus on the conceptual and specification perspectives to avoid the premature implementation perspective all the time during your projects
- We can show the static structure of the things that exist, their internal structure, and their relationships to other things

## What are the main components of a Class Diagram?

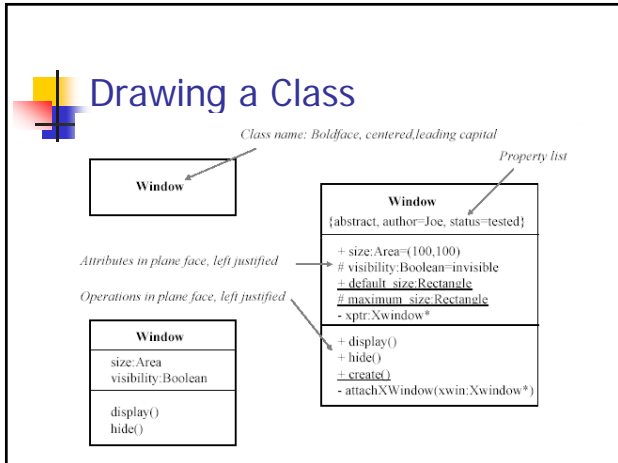
- Class
  - Class name
  - Attribute
  - Operation
- Relationships
  - Generalization
  - Association
  - Aggregation
  - Dependency

## Class - Semantic

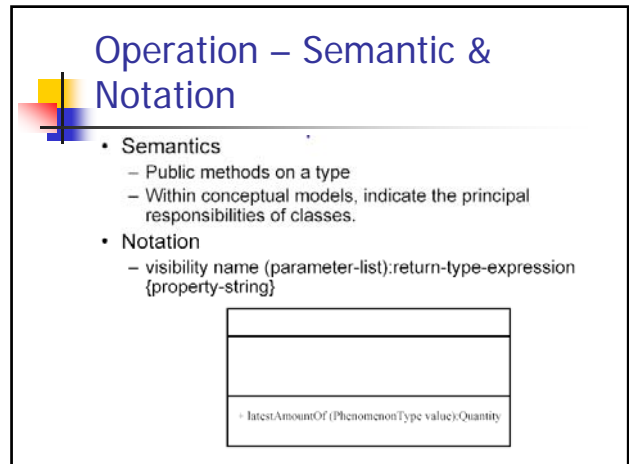
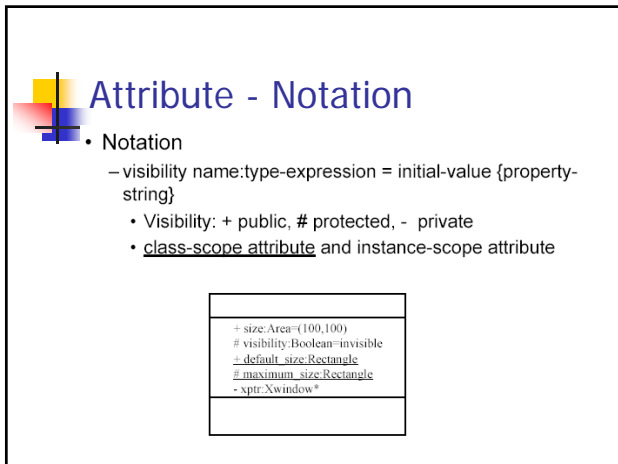
- A class is the descriptor for a set of objects with similar structure, behavior, and relationships
- Classes are declared in class diagrams and used in most other diagrams.
- The name of a class has scope within the package in which it is declared.
- The name must be unique among class names within its package

## Class - Notation

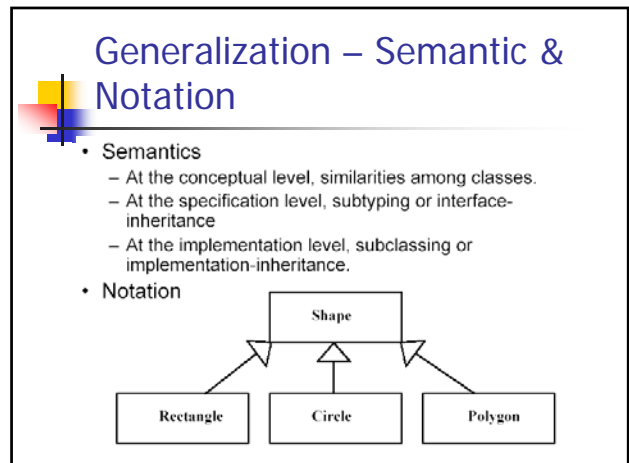




- ### Attributes - Semantic
- At the conceptual level, it is equivalent to a composition association
  - Attributes are always single valued
  - Any type used as an attribute has value rather than references



- ### Generalization
- A relationship between a general thing and a more specific kind of that thing
  - “is a kind of” relationship

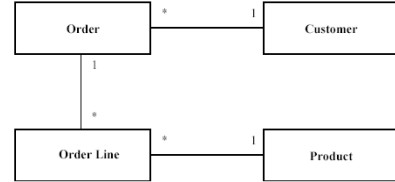


## Association

- A structural relationship that specifies that objects of one thing are connected to objects of other
- At the conceptual level, associations represent conceptual relationships between classes
- At the specification level, associations represent responsibilities
- At the implementation level, associations represent navigability.

## Association - Example

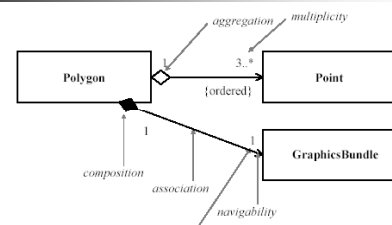
- Example
  - An Order has to come from a single Customer. A Customer may several Orders over time. Each of these Orders has several Order Lines, each of which refers to a single Product.



## Association

- Role name
  - Each association has two roles; each role is a direction on the association
  - A role can be explicitly named with a label. If there is no label, you name a role after the target class
- Multiplicity
- Navigability
- Aggregation/Composition indicators
- Ordering

## Association

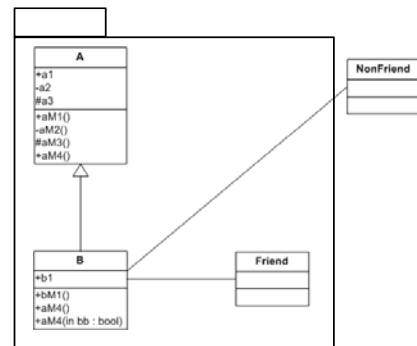


*An GraphicsBundle role whose source is Polygon and whose target is GraphicsBundle*

## Dependency

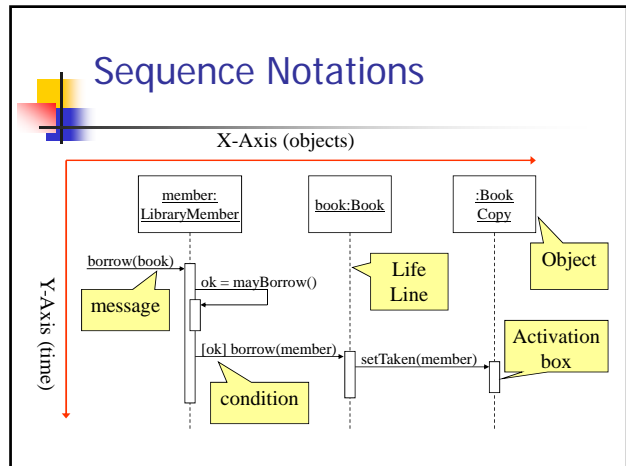
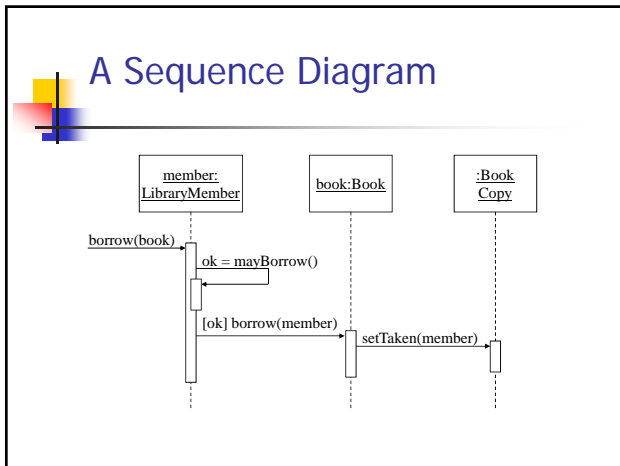
- A using relationship that states that a change in specification of one thing may affect another thing that uses it, but not necessarily the reverse
- One class uses another class as an argument in the signature of an operation

## Visibility Revisited



# Sequence Diagram

- ## Sequence Diagram
- Illustrates how objects interact with each other.
  - Emphasizes time ordering of messages.
  - Can model simple sequential flow, branching, iteration, recursion and concurrency



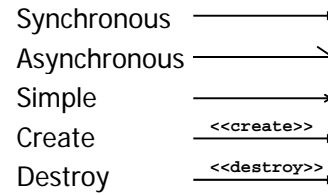
- ## Object
- Object naming:
    - syntax: `[instanceName][:className]`
    - Name classes consistently with your class diagram (same classes).
    - Include instance names when objects are referred to in messages or when several objects of the same type exist in the diagram.
  - The *Life-Line* represents the object's life during the interaction
- 

- ## Message
- An interaction between two objects is performed as a message sent from one object to another.
    - Most often implemented by a simple operation call.
    - Can be an actual message sent through some communication mechanism, either over the network or internally on a computer.
      - Inter-process communication (Signaling, ...)
      - Remote Procedure Call (RMI, CORBA, ...)

## Message (2)

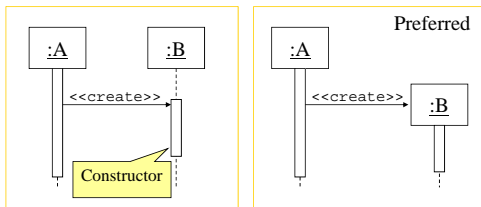
- A message is represented by an arrow between the life lines of two objects.
  - Self calls are also allowed
  - The time required by the receiver object to process the message is denoted by an *activation-box*.
- A message is labeled at minimum with the message name.
  - Arguments and control information (conditions, iteration) may be included.
  - Prefer using a brief textual description whenever an *actor* is the source or the target of a message.

## Message Types



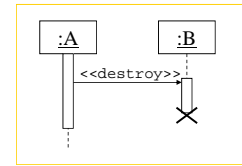
## Object Creation

- An object may create another object via a `<<create>>` message.



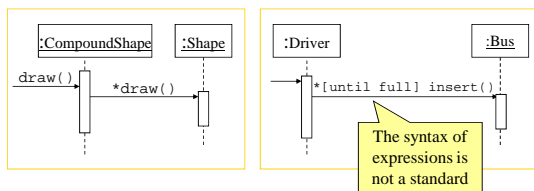
## Destroying Object

- An object may destroy another object via a `<<destroy>>` message.
  - An object may destroy itself.
  - Avoid modeling object destruction unless memory management is critical.



## Iteration in Sequence Diagram

- Iteration examples:



## Iteration of a set of messages

