Unified Modeling Language (UML)

Analyst hat:
Where are we in the SDLC?

- ✧ Specification
  - Identify Problem
  - Gather requirements
  - **Analyze & Model requirements**
  - Select architecture/COTS
- ✧ Design and implementation
- ✧ Validation
- ✧ Evolution

Requirements for the Traditional and OO Approaches

Events, Use cases And event table

Things

Traditional Approach

Entity-relationship diagram (ERD)

Context diagram

DFD Fragments

Data flow definitions

Process descriptions

Other traditional models

Object-Oriented Approach

Class diagram

Use case diagrams

Use case descriptions

System sequence diagrams

Activity diagrams

State machine diagrams

Done

Now

Overview

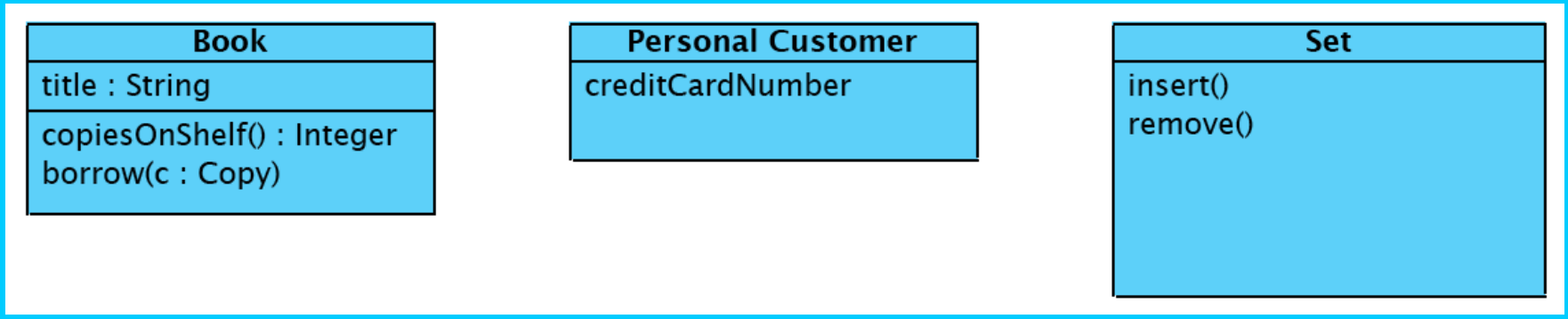
- ✧ Classes, attributes
- ✧ Generalization
- ✧ Aggregation and composition
- ✧ Association classes
- ✧ Interfaces
- ✧ Parameterized classes
- ✧ Interaction diagrams

Class Diagrams

- In a class diagram, classes of objects are represented by a square box.
- Variety of ways of identifying why such a box is needed
  - analysis + design
  - for now, let's assume we have done this and need to notate the result
- Classes obtain meaning by their associations with other classes
  - message sent from class to class
  - even class instances contained by others

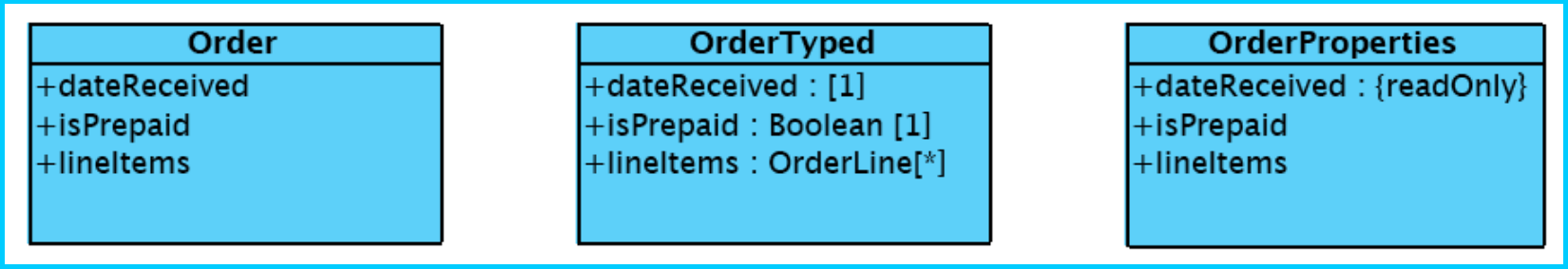
Class

- May or may not contain "features"
  - attributes
  - operations
- Some feature sets may remain empty
  - attributes, but no operations
  - operations, but no attributes



Class (continued)

- Features could have other attributes
  - visibilities (+, -, ~, #)
    - Public +
    - Package ~
    - Protected #
    - Private -
  - types
  - default values for attributes
  - additional properties
- Must take care: goal is notation + abstraction, not coding!



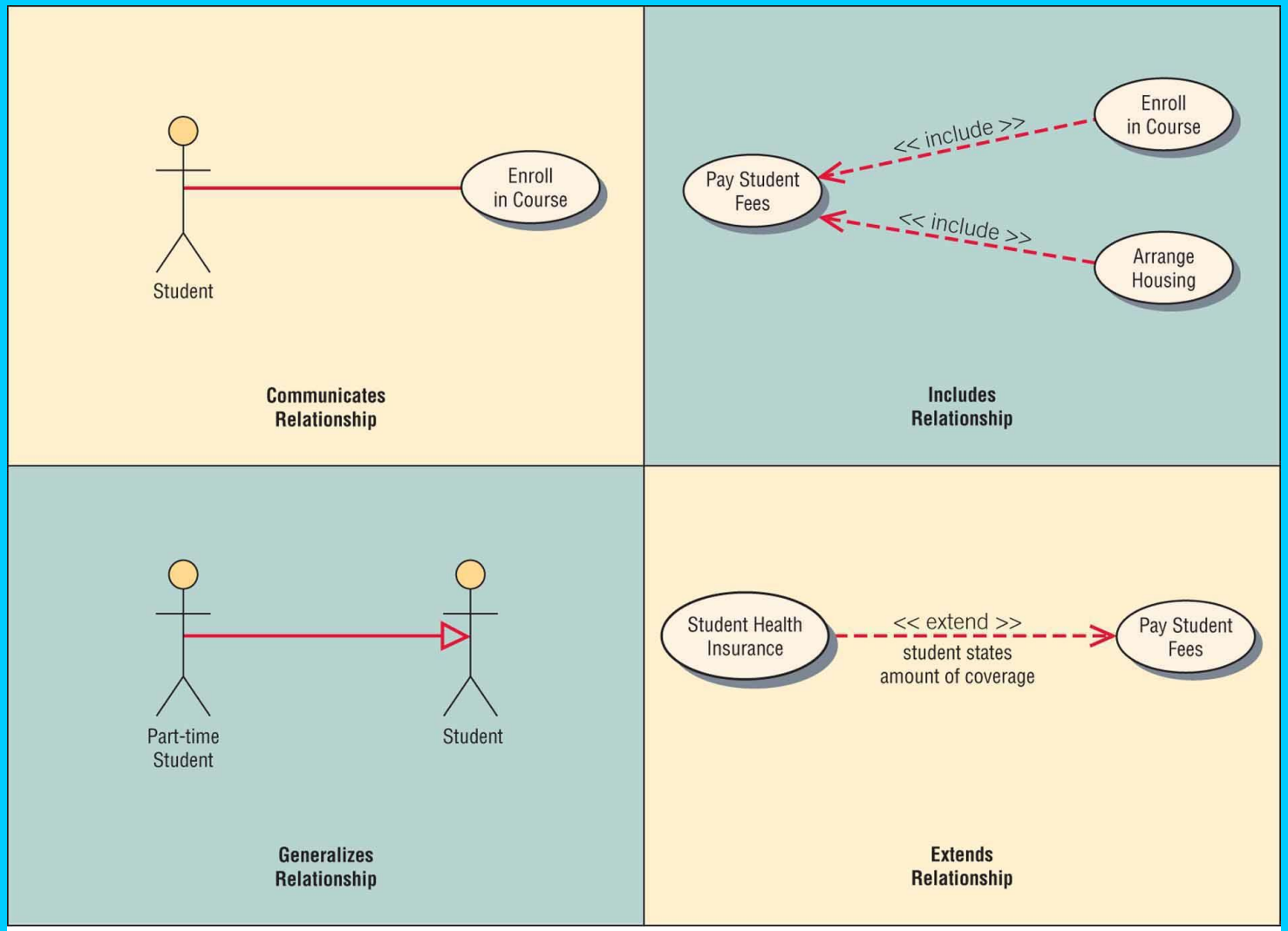
UML Relationships

- ✧ Behavioral Relationships (Already done)
  - Communicates
  - Includes
  - Extends
  - Generalizes
- ✧ Structural Relationships (New)
  - Generalizations
  - Associations
  - Aggregations
  - Dependencies

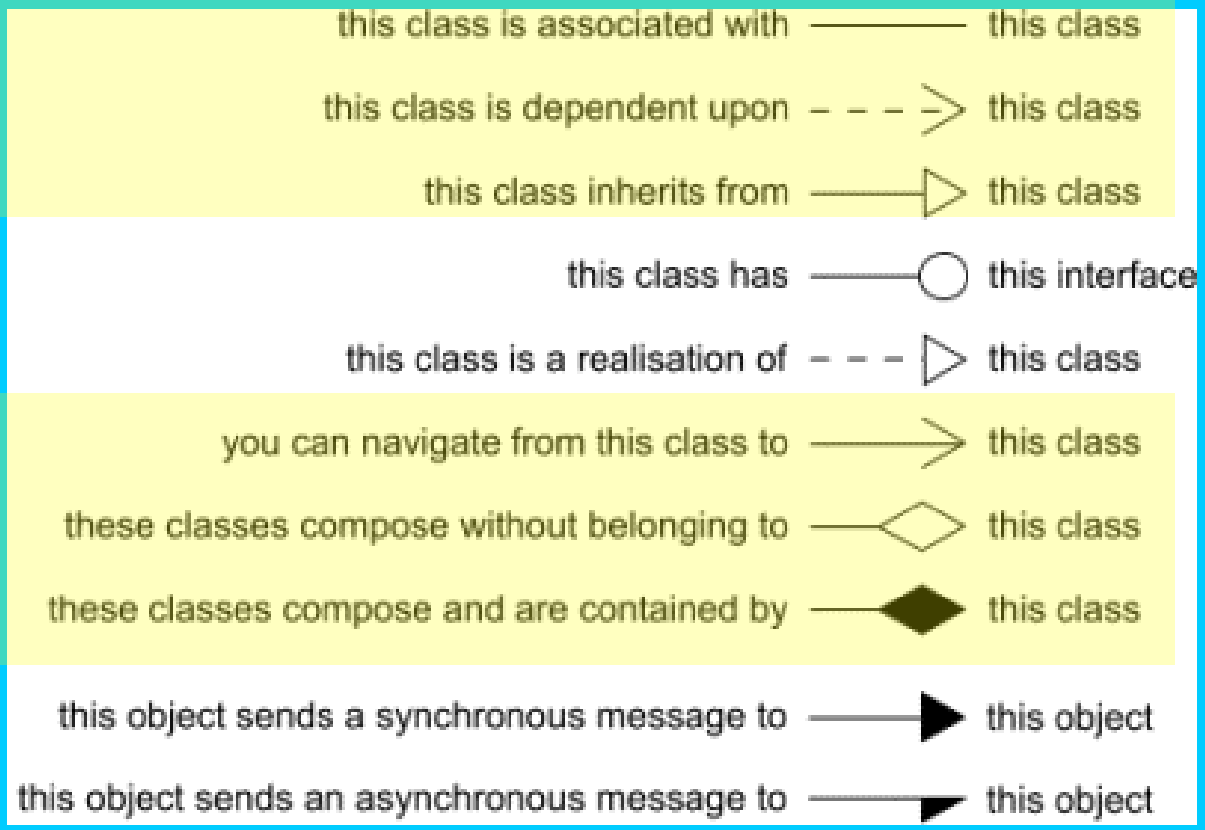
Recall: Behavioral Relationships Symbols

Relationship	Symbol	Meaning
Communicates		An actor is connected to a use case using a line with no arrowheads.
Includes	<< include >>	A use case contains a behavior that is common to more than one other use case. The arrow points to the common use case.
Extends	<< extend >>	A different use case handles exceptions from the basic use case. The arrow points from the extended to the basic use case.
Generalizes		One UML "thing" is more general than another "thing." The arrow points to the general "thing."

Recall: Behavioral Relationships Examples



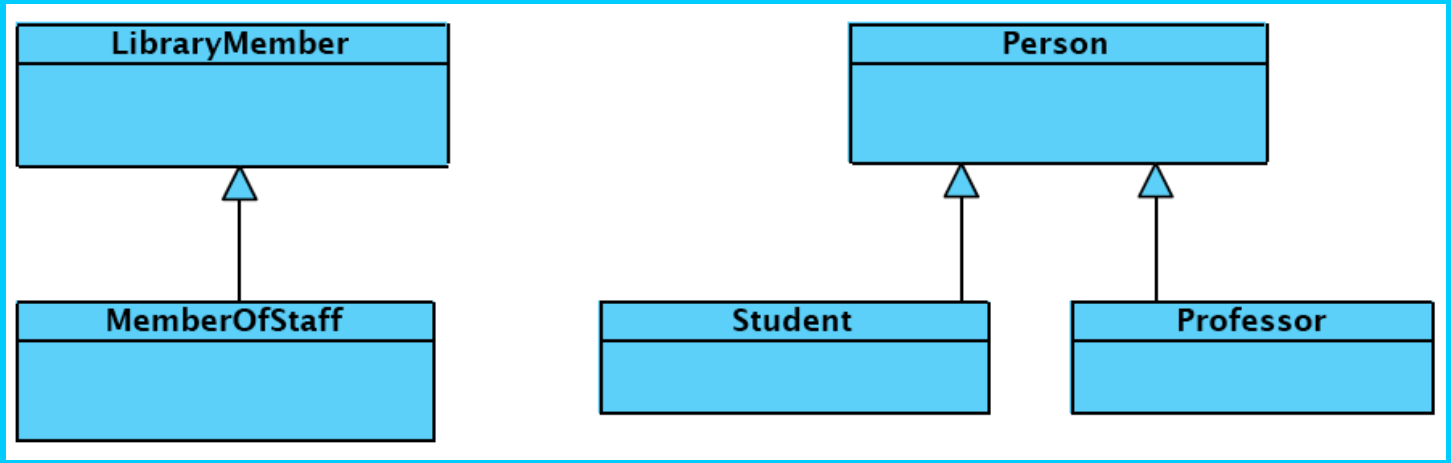
Structural Relationships



Generalization / subclassing

- Relationships of classes to each other via inheritance can be shown with an open arrow.
- Same diagram can mix subclassing with associations
- Sometimes we even want to keep software out of the discussion
  - Modeling classes and their relationships helps us understand problem domain.
  - Much harder to "keep software out" than it seems
  - In practice: keep software "as late as possible"

Generalization / Subclassing

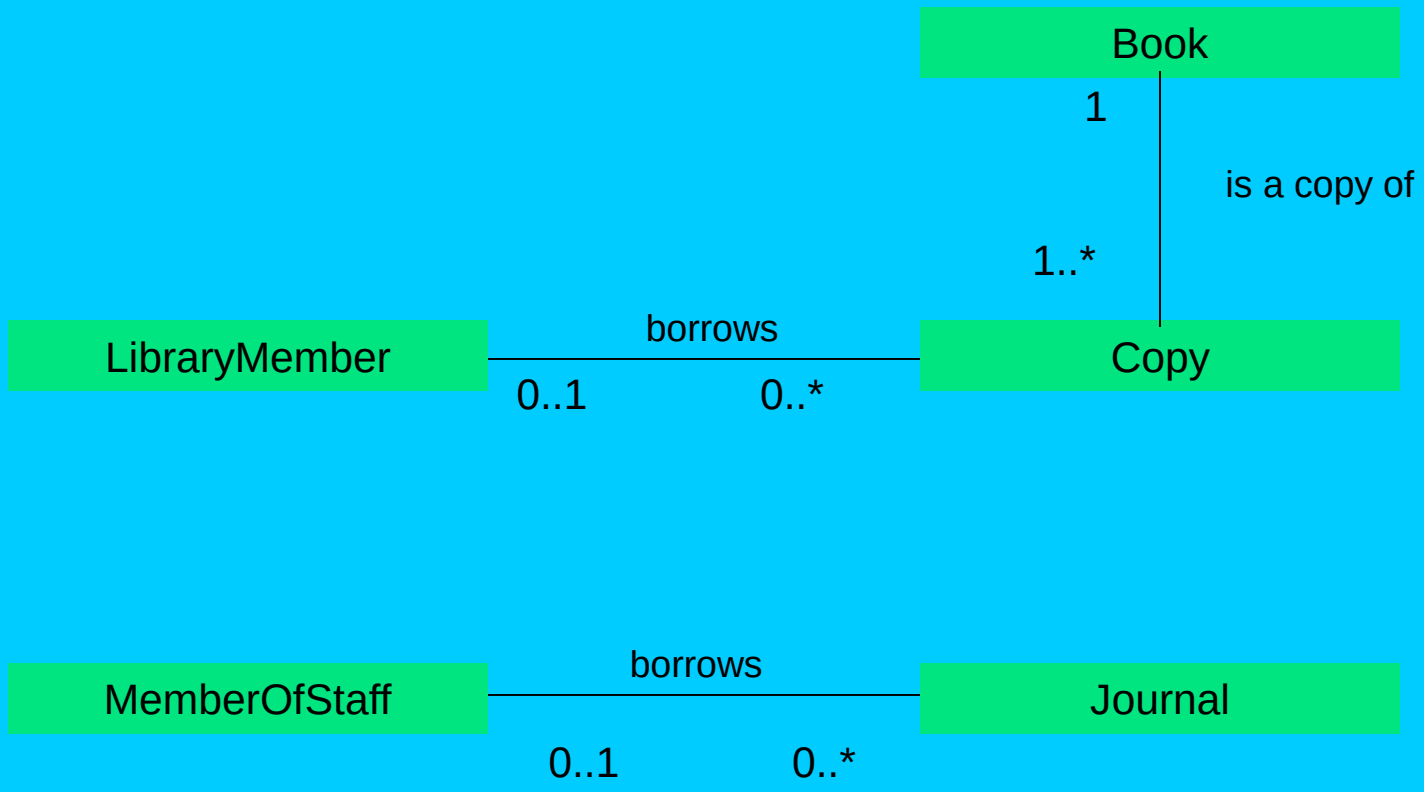


this class is associated with ——— this class

Associations

- ✧ We have previously seen associations
  - These are lines which connect classes
  - They indicate the roles classes play for each other within a diagram
- ✧ Consider a Library diagram
  - related Books, Copies, LibraryMembers, etc.

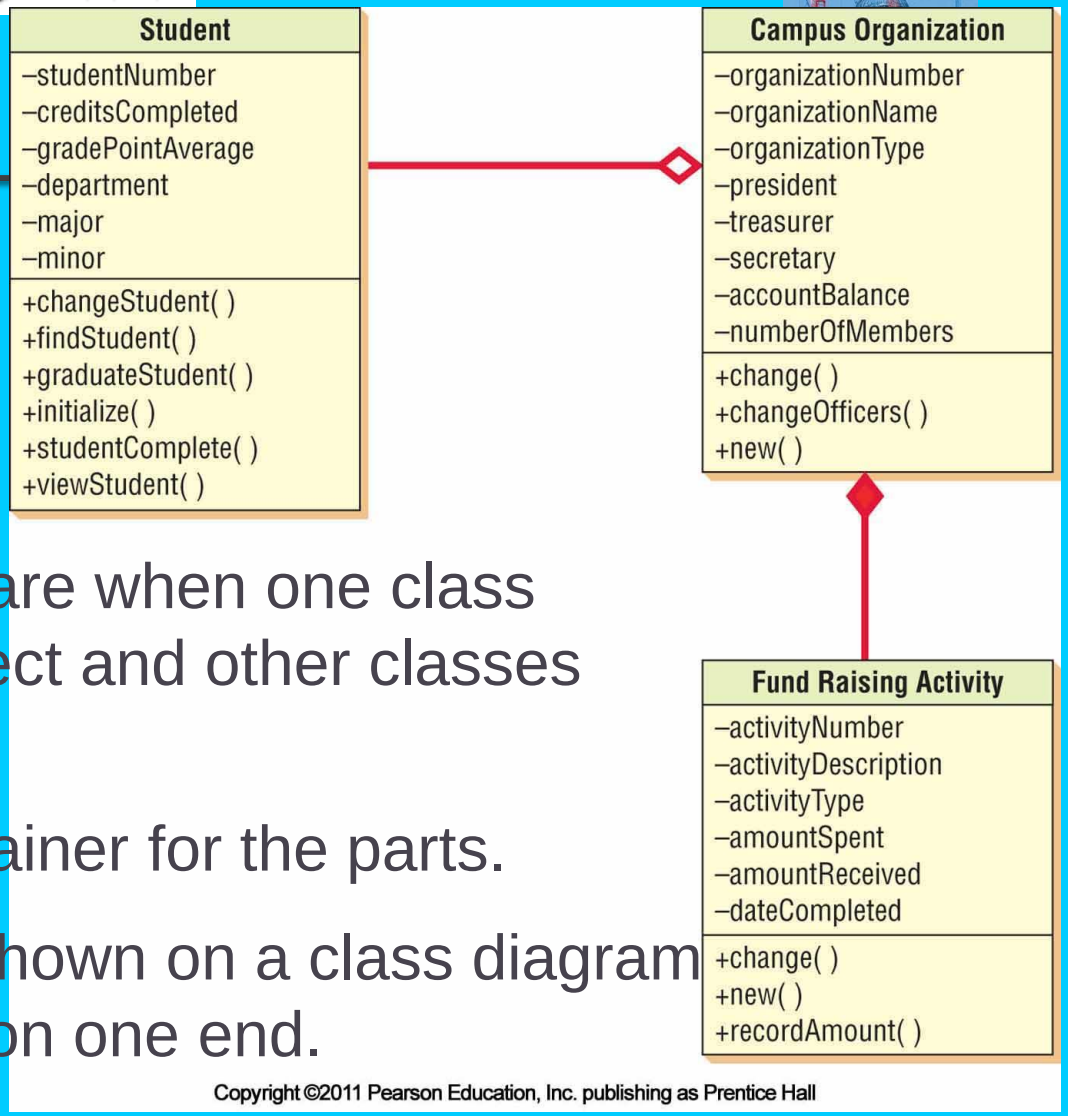
Example (for a library)



these classes compose without belonging to ◇ this class

these classes compose and are contained by ◆ this class

WHOLE/PART RELATIONSHIPS



- Whole/part relationships are when one class represents the whole object and other classes represent parts.
- The whole acts as a container for the parts.
- These relationships are shown on a class diagram by a line with a diamond on one end.
- The diamond is connected to the object that is the whole. (The campus organization has students)

WHOLE/PART RELATIONSHIPS

- **Collection**
  - A collection consists of a whole and its members.

✧ **Aggregation "has"**

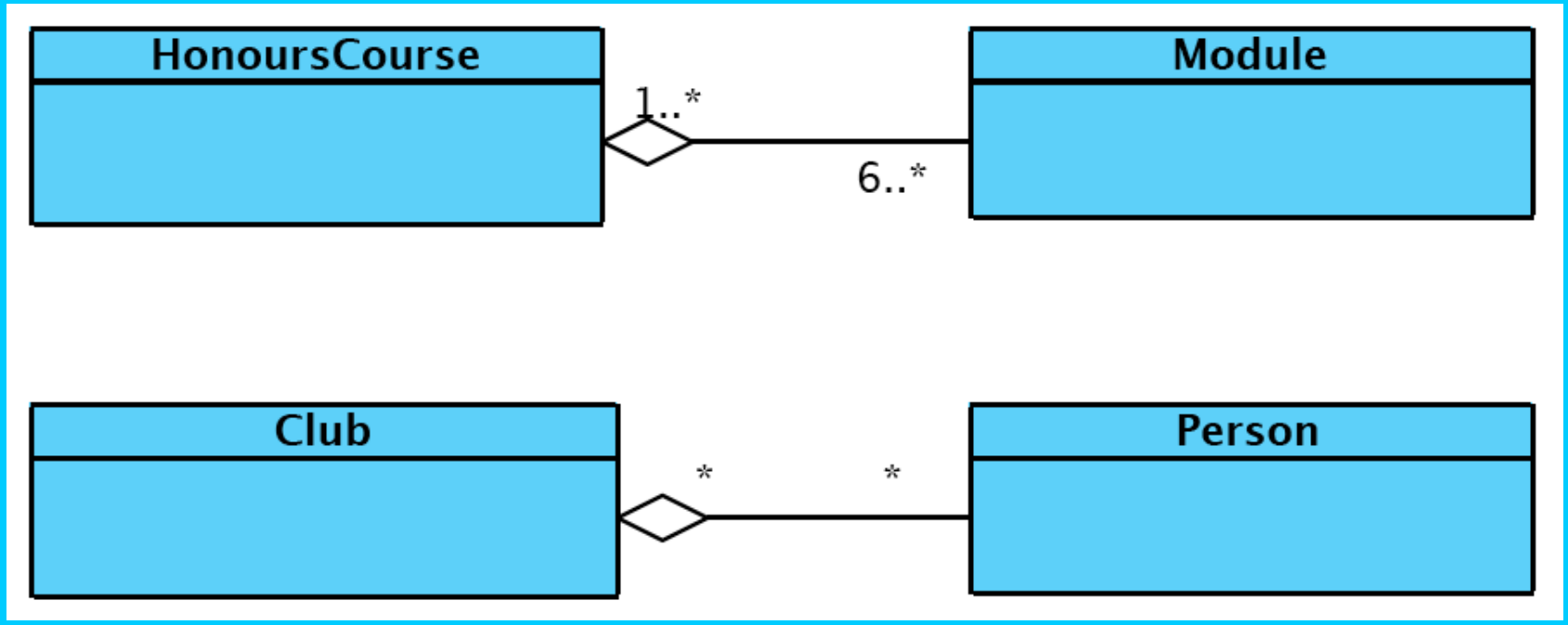
- An aggregation is often described as a "has a" relationship.

❖ **Composition "always has"**

- Composition, a whole/part relationship in which the whole has a responsibility for the part.
- It is a stronger relationship, and is usually shown with a filled-in diamond.
- Keywords for composition are one class "always contains" another class.

Aggregation (example)

"HonorsCourse has 6 or more Modules"

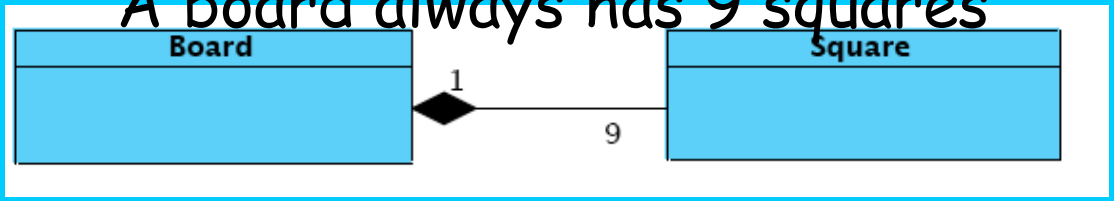


Composition

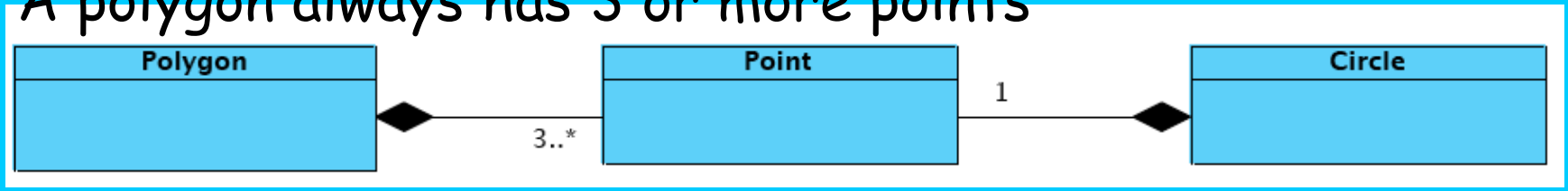
- ✧ Related to aggregation, but imposes some restrictions
  - These are actually "implementation restrictions" as opposed to modelling restrictions
- ✧ Notion of "strongly owned"
  - Some object N is owned by object M
  - If the whole M object is copied or deleted, all aggregated N objects are also copied or deleted

Composition (examples)

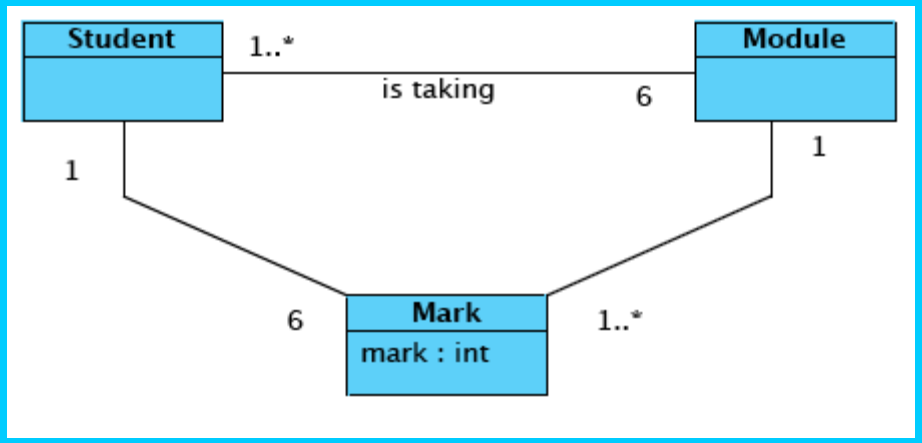
A board always has 9 squares



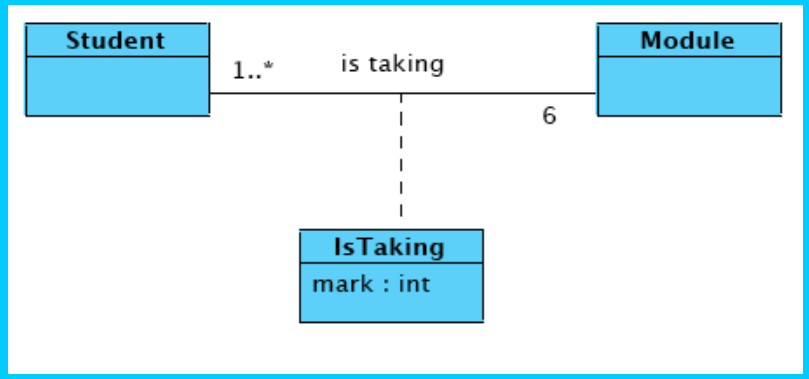
A polygon always has 3 or more points



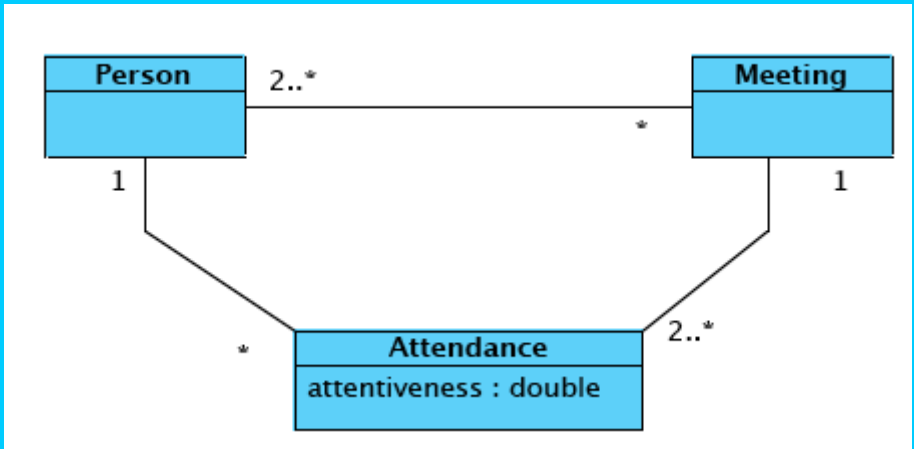
Without association classes…



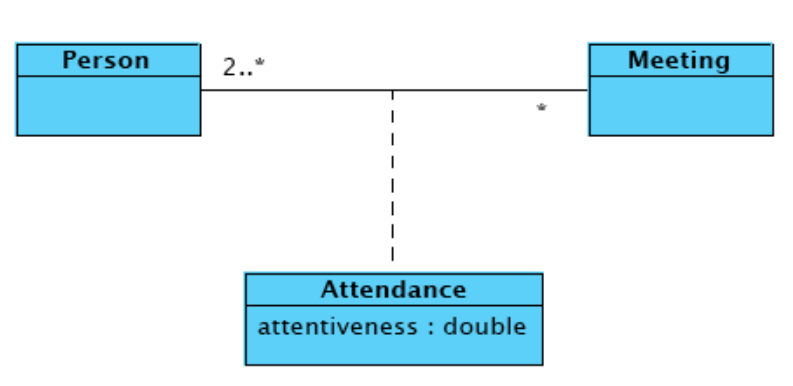
With association classes



Without association classes



With association classes



this class is dependent upon - - - > this class

Interfaces

- Java has an "interface" construct
  - specifies a list of operations
  - specifies a list of attributes
  - "implementation" of some Java interface must include code for it.
- Other languages have something similar
  - Multiple inheritance (C++)
  - Interfaces (C#)
  - Mixin classes (Python, Ruby)

```
public interface Attributed {
    void add(Attr newAttr);
    Attr find(String attrname);
    Attr remove(String attrname);
    java.util.Iterator<Attr> attrs();
}

class AttributedImpl
     implements Attributed,
          Iterable<Attr>
{
    public void add(Attr newAttr) {
        // ...
    }

    public Attr find(String name) {
        // ...
    }
    // etc. with bodies for "remove"
    // and "attrs"
}
```

C#

An abstract class cannot be instantiated.

```
abstract class GraphicObject
{
    abstract public int Area();
}
```

An abstract method is declared without any implementation.

An abstract class may contain abstract methods and accessors.

C#


```
class Square : GraphicObject
{
    int side = 0;

    public Square(int n)
    {
        side = n;
    }
    // Area method is required to avoid
    // a compile-time error.
    public override int Area()
    {
        return side * side;
    }
```

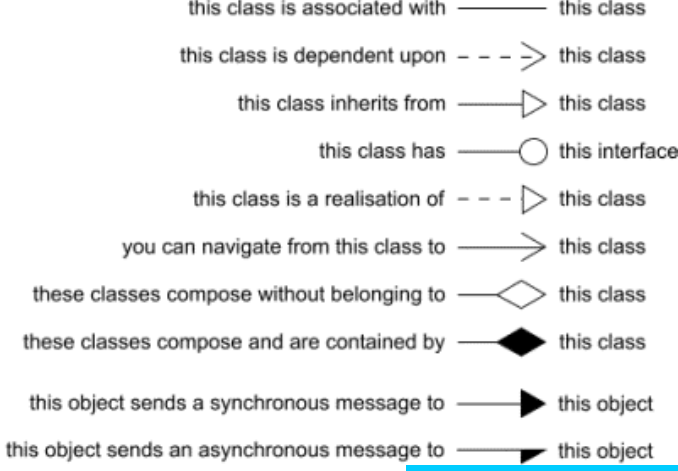
C#

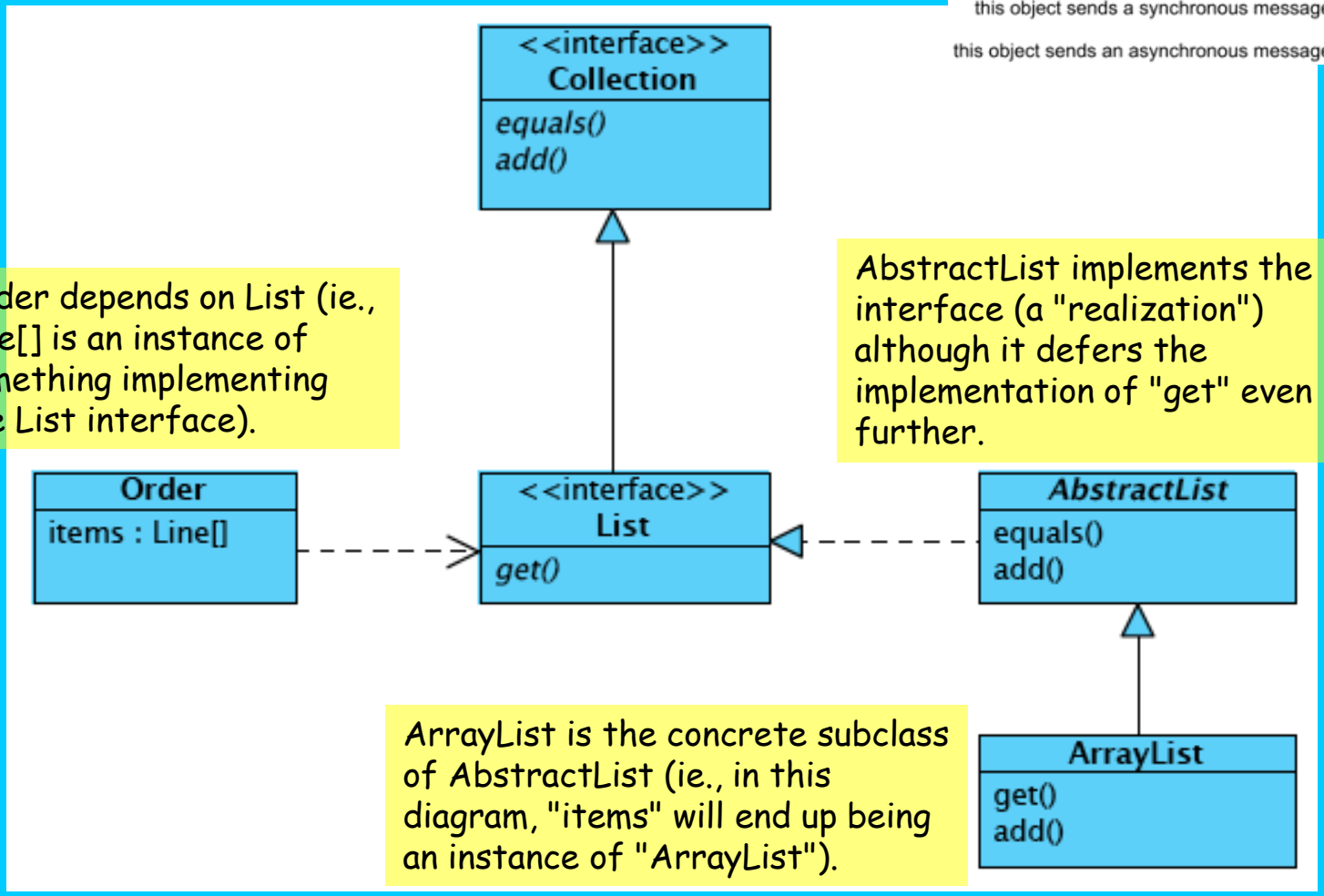

```
    static void Main()
    {
        Square sq = new Square(12);
        Console.WriteLine("Area of the square = {0}", sq.Area());
    }
}
```

Abstract SuperClass C++

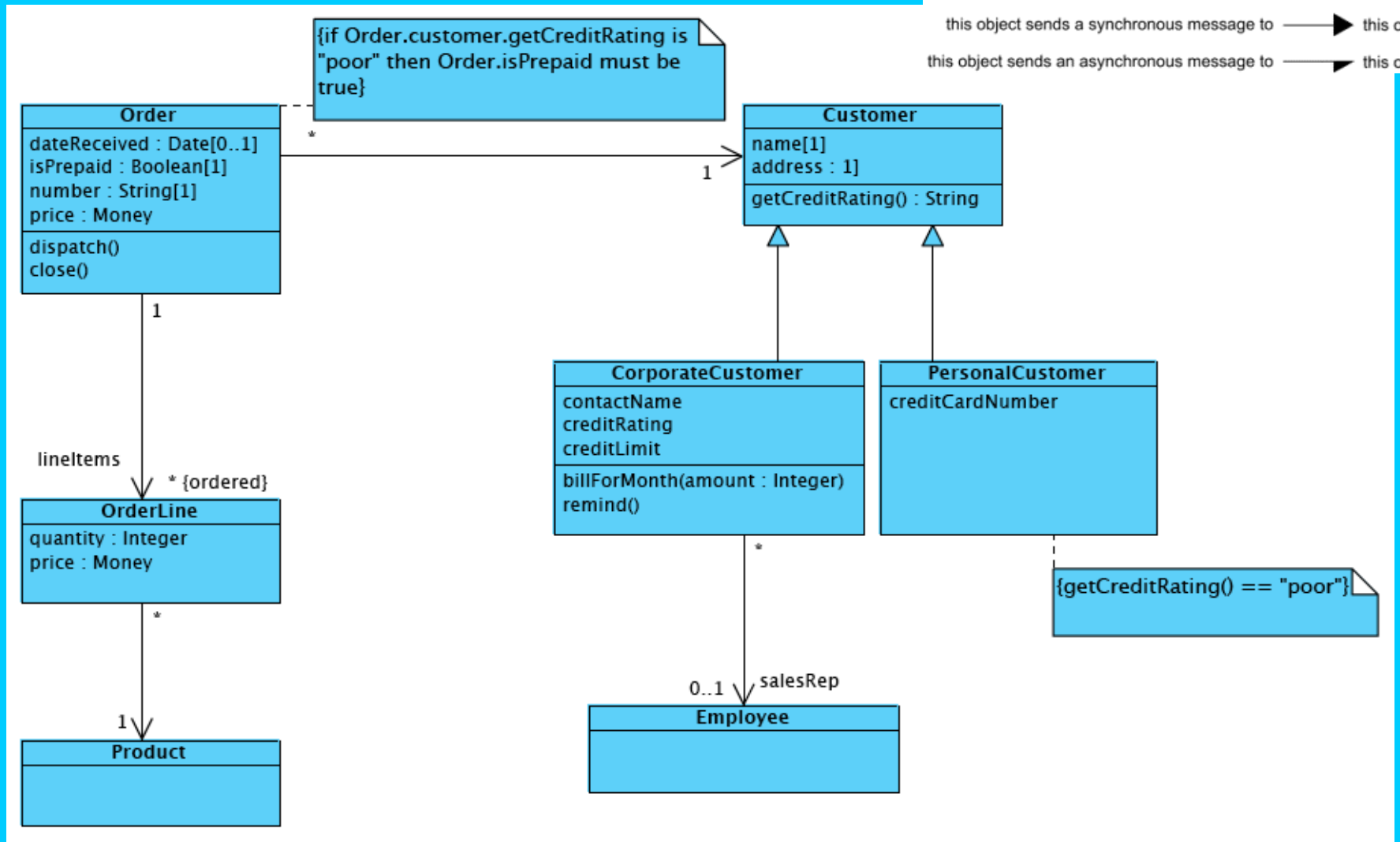

```
class AbstractClass {
    public: virtual void AbstractMemberFunction() = 0;
        // Pure virtual function makes
        // this class Abstract class.

    virtual void NonAbstractMemberFunction1();
        // Virtual function.

    void NonAbstractMemberFunction2();
};
```

Interface (example)





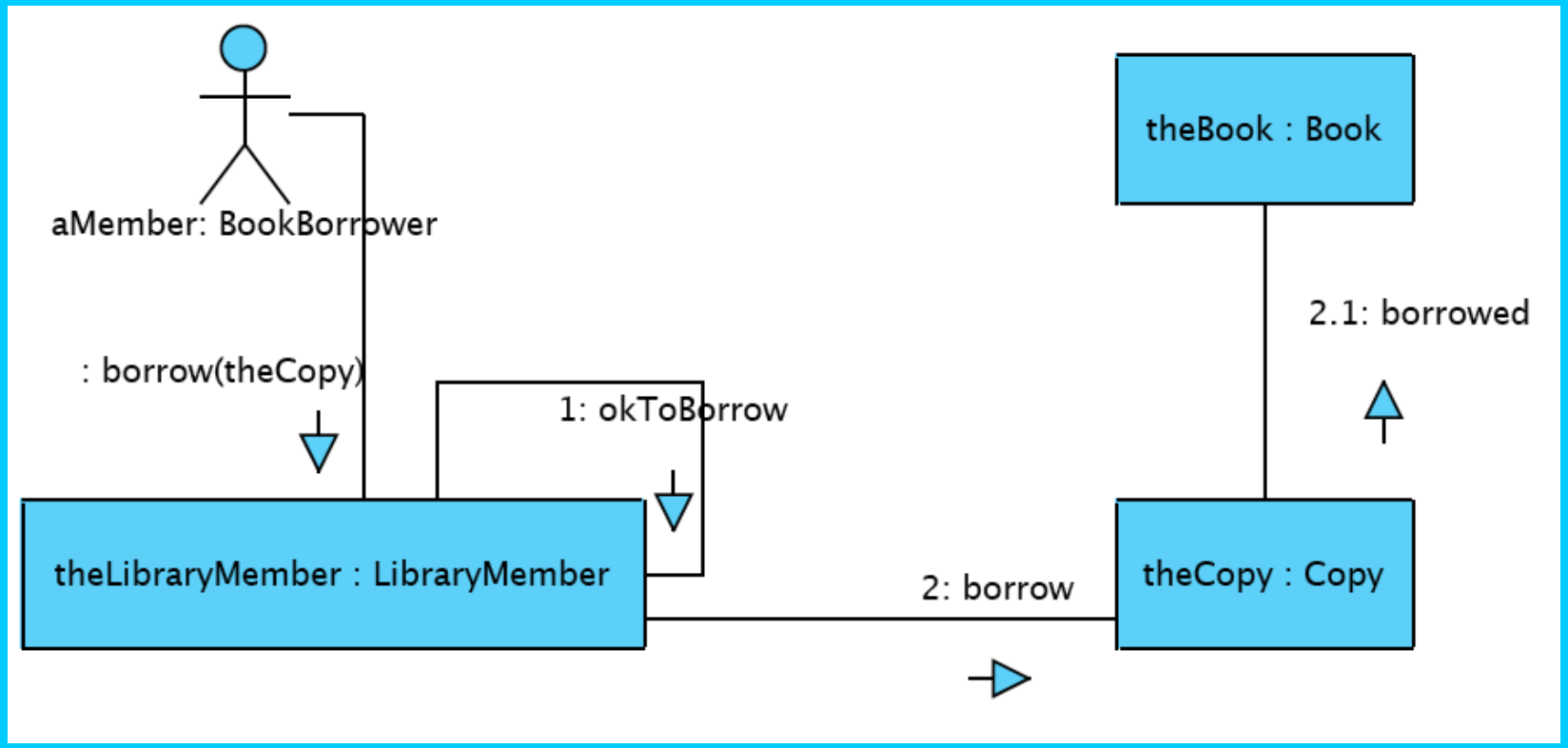
Simple class diagram



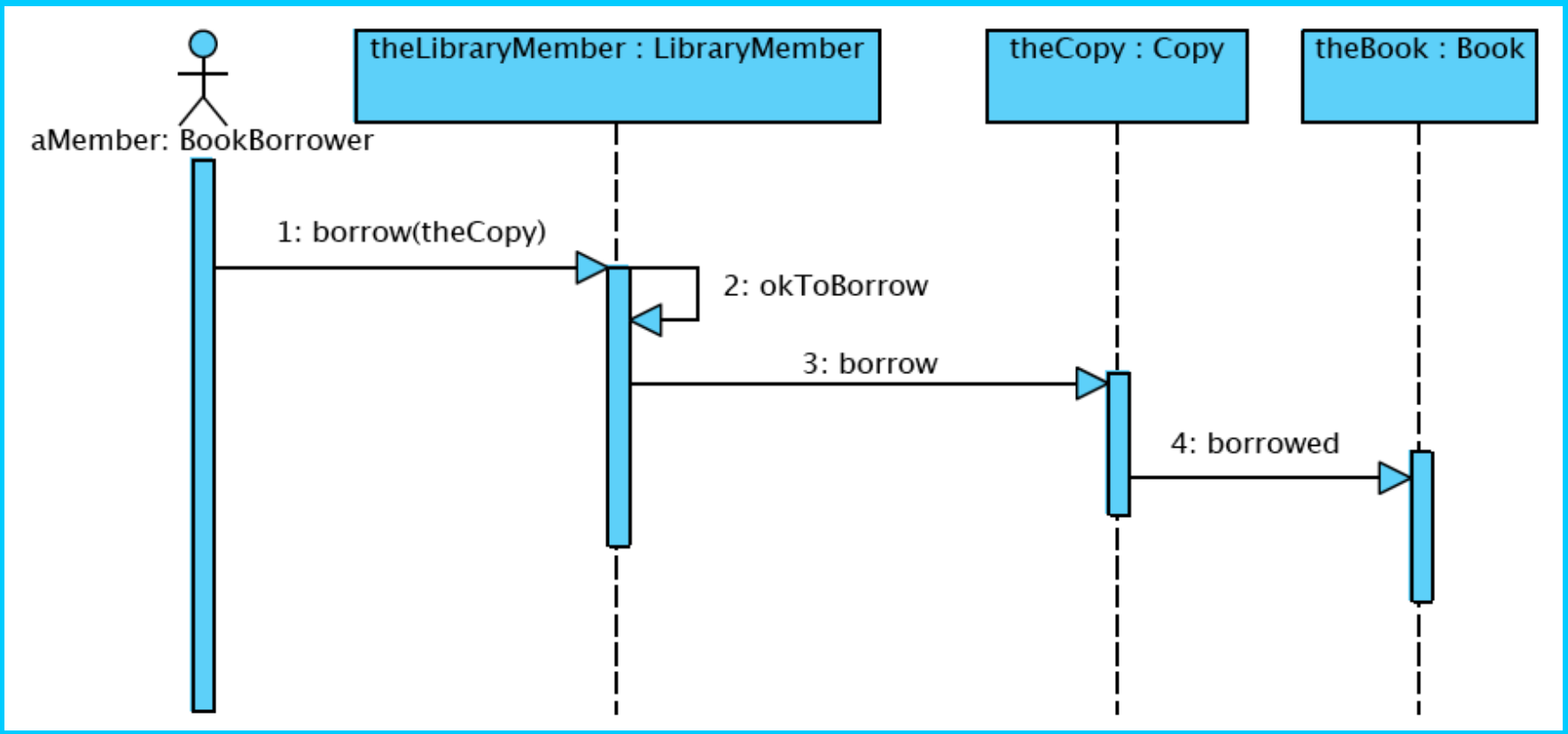
Interaction diagrams

- Three main types (in order of increasing detail)
  - Collaboration diagram
  - Communication diagram
  - Sequence diagram
- Captures dynamic behavior of system
  - Several such diagrams are needed to capture all dynamic behavior
  - Each diagram style emphasizes different aspects of that behavior

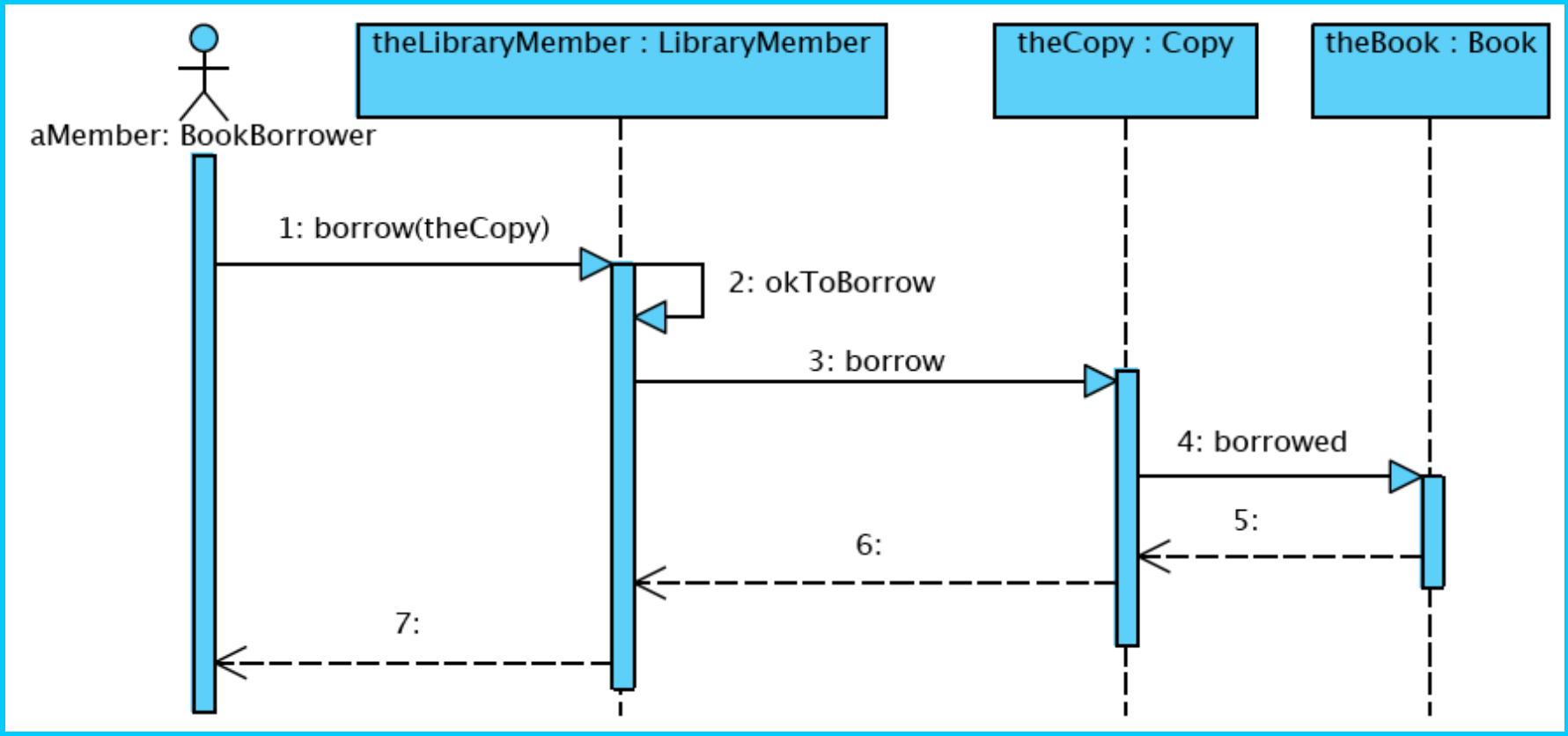
Communication diagram



Sequence diagram



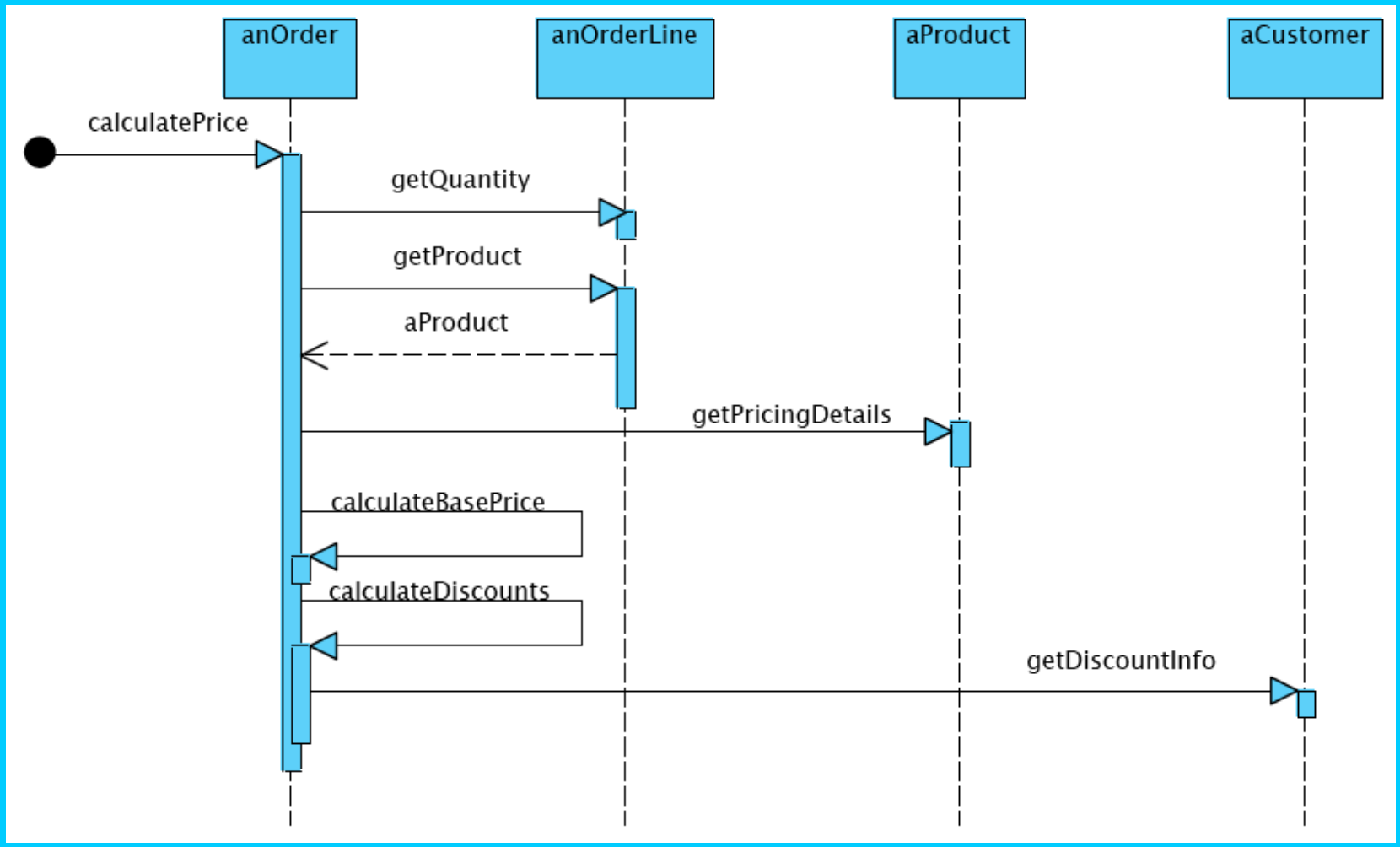
Sequence diagram (w/returns)



Variations

- If class of object is obvious, then no need to mention class
- Some UML tools help show activations and related nesting
  - Shaded areas of bar: active; Clear areas of bar: nested
  - Helps demonstrate absence (or presence) of concurrency…
- Object construction and destruction
  - Helpful if object is intended to be a connection, a resource, etc.
  - Not necessarily the same as "object deletion" (ie., garbage collection)

Sequence diagram



When to use these diagrams?

- Sequence diagrams are:
  - more expressive
  - easier to read
- Choosing between one and the other:
  - often personal preference
  - most people seem to prefer sequence diagrams
- More rationally:
  - Communication diagrams should be chosen when links between objects need emphasis
  - These are also easier to draw and alter on a whiteboard
  - Sometimes used as a substitute for CRC cards