CE343 SOFTWARE ENGINEERING

Chapter 5 Software Design

Prepared by:

Ms. Sneha Padhiar

Assistant Professor,

U & P U. Patel Department of Computer Engineering

Unified Modelling Language (UML)

- Origin
  - In late 1980s and early 1990s different software development houses were using different notations
  - Developed in early 1990s to standardize the large number of object-oriented modelling notations
- Based Principally on
  - **OMT** [Rumbaugh 1991]
  - **Booch's methodology** [Booch 1991]
  - **OOSE** [Jacobson 1992]
  - **Odell's methodology** [Odell 1992]
  - **Shlaer and Mellor** [Shlaer 1992]

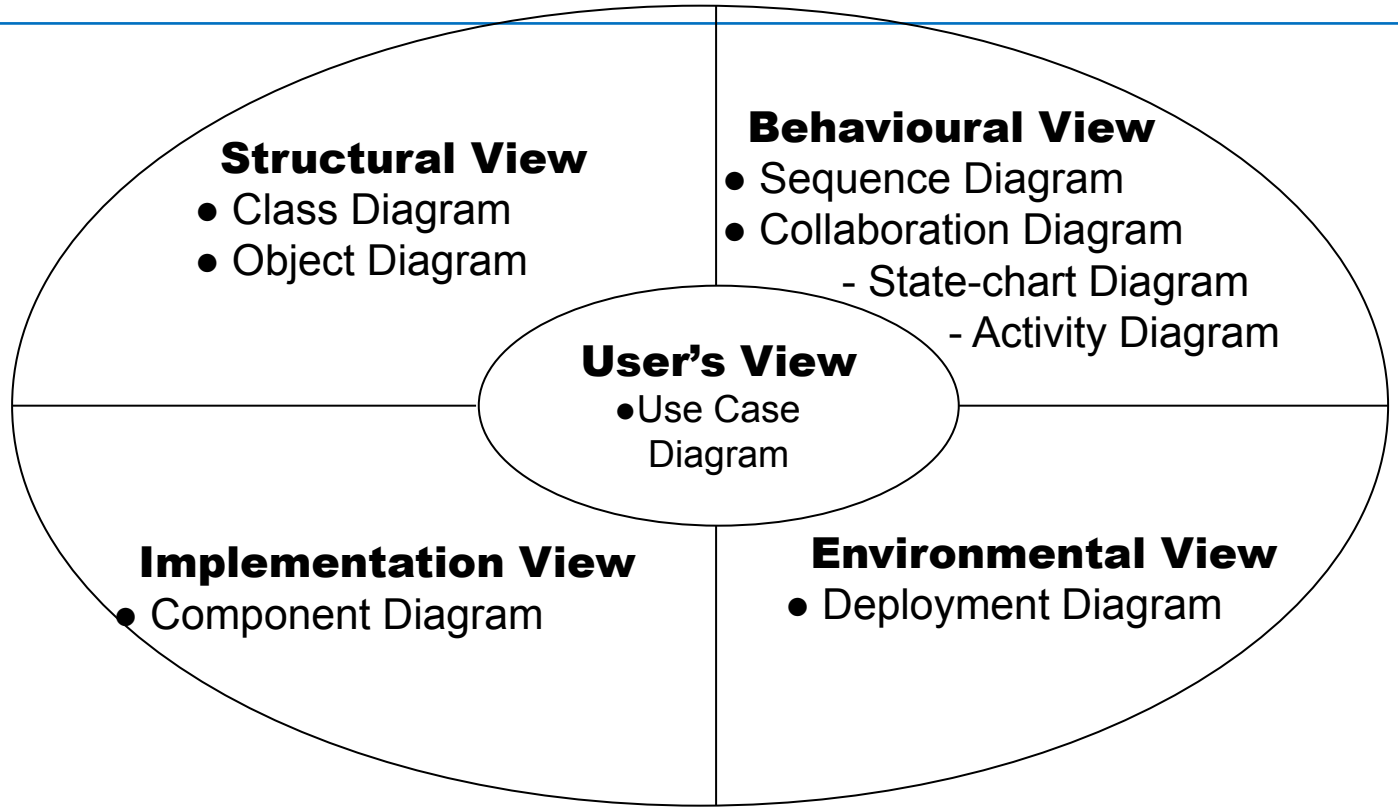
Why UML is required

- Model is required to capture only important aspects
- UML a graphical modelling tool, easy to understand and construct
- Helps in managing complexity

UML diagrams

- Nine diagrams to capture different views of a system
- Provide different perspectives of the software system
- Diagrams can be refined to get the actual implementation of the system

- Views of a system
  - **User's** view
  - **Structural** view
  - **Behavioral** view
  - **Implementation** view
  - **Environmental** view



Diagrams and views in UML

Are all views required?

- **NO**
  - Use case model, class diagram and one of the interaction diagram for a simple system
  - State chart diagram in case of many state changes
  - Deployment diagram in case of large number of hardware components

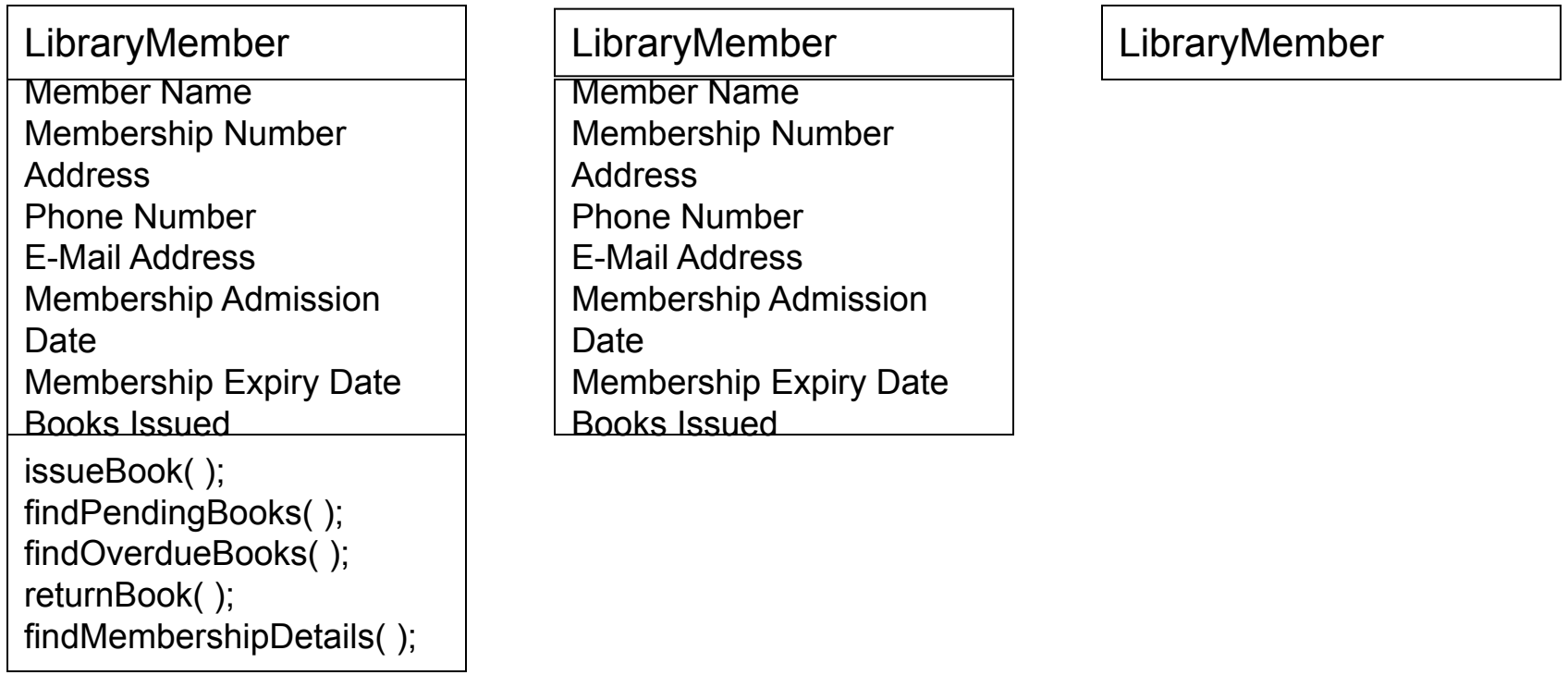
Class diagram

Class diagram

- Describes static structure of a system
- Main constituents are classes and their relationships:
  - **Generalization**
  - **Aggregation**
  - **Association**
  - Various kinds of **dependencies**

- Entities with common features, i.e. attributes and operations
- Classes are represented as solid outline rectangle with compartments
- Compartments for **name**, **attributes** & **operations**
- Attribute and operation compartment are optional for reuse purpose

Example of Class diagram

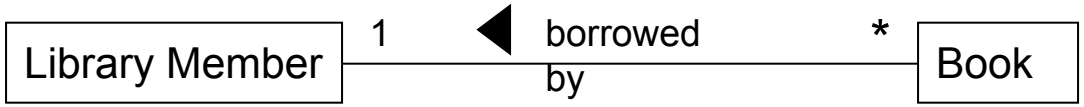


Different representations of the LibraryMember class

Association Relationship

- Enable objects to communicate with each other
- Usually binary but more classes can be involved
- Class can have relationship with itself (**recursive** association)
- Arrowhead used along with name, indicates direction of association
- Multiplicity indicates # of instances

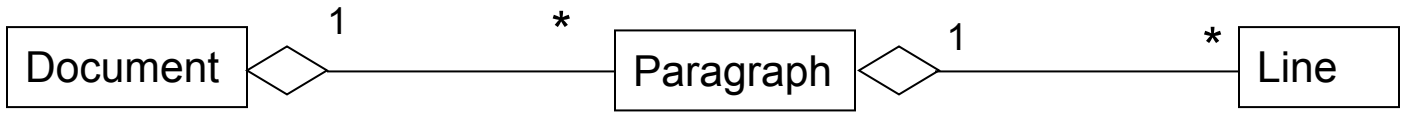
Association Relationship



Association between two classes

If two classes in a model need to communicate with each other, there must be a link between them, and that can be represented by an association (connector).

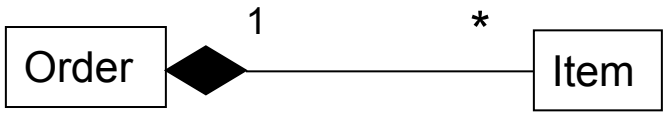
Aggregation Relationship



Representation of aggregation

aggregation object of one class "owns" object of another class.

Composition Relationship



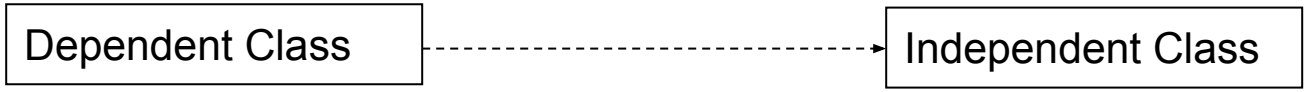
Representation of composition

composition object of one class "owns" object of another class.

Difference

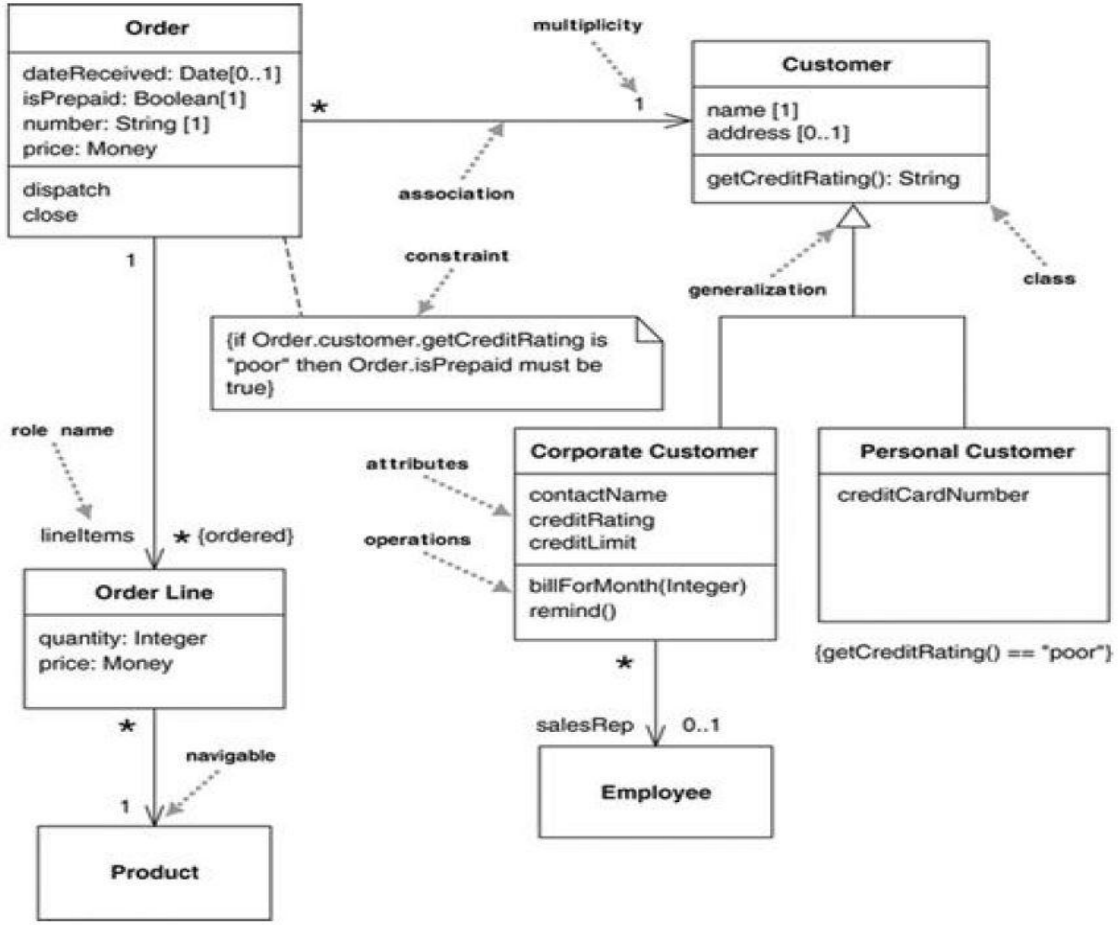
- **Aggregation** and **Composition** are subsets of association meaning they are **specific cases of association**.
- In both aggregation and composition object of one class "owns" object of another class. But there is a subtle difference:
- **Aggregation** implies a relationship where the child can exist independently of the parent. Example: Class (parent) and Student (child). Delete the Class and the Students still exist.
- **Composition** implies a relationship where the child cannot exist independent of the parent. Example: House (parent) and Room (child). Rooms don't exist separate to a House.

Class Dependency



Representation of dependence between class

Example


Order
dateReceived: Date[0..1]
isPrepaid: Boolean[1]
number: String [1]
price: Money
dispatch
close
multiplicity
Customer
name [1]
address [0..1]
getCreditRating(): String
association
constraint
generalization
class
{if Order.customer.getCreditRating is "poor" then Order.isPrepaid must be true}
role name
lineItems
{ordered}
attributes
operations
Corporate Customer
contactName
creditRating
creditLimit
billForMonth(Integer)
remind()
Personal Customer
creditCardNumber
{getCreditRating() == "poor"}
Order Line
quantity: Integer
price: Money
navigable
salesRep
0..1
Employee
Product

Sequence diagram

Interaction diagram

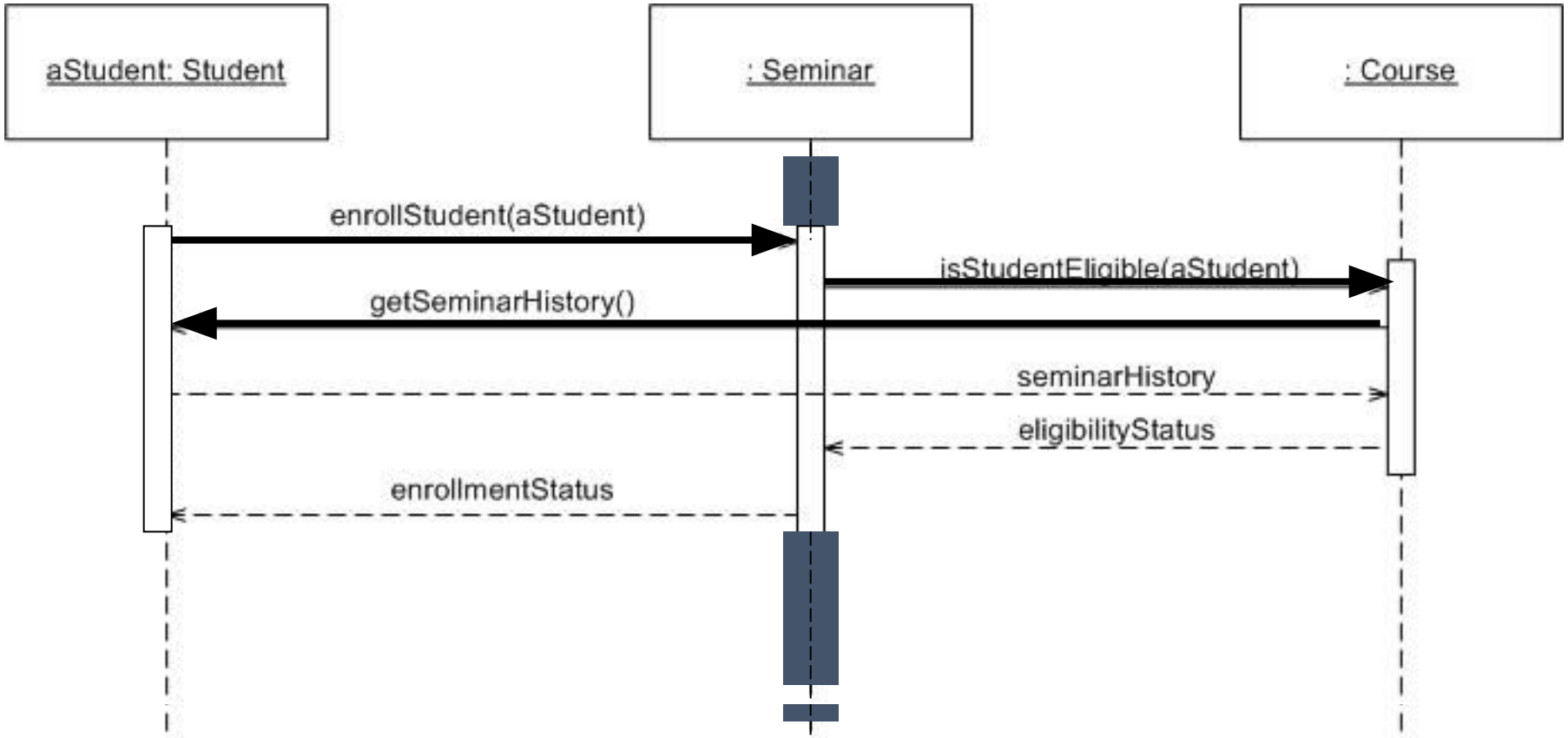
- Models how groups of objects collaborate to realize some behaviour
- Typically each interaction diagram realizes behaviour of a single use case
- Two kinds: **Sequence** & **Collaboration**
- Two diagrams are equivalent but portrays different perspective
- These diagram play a very important role in the design process

Sequence Diagrams

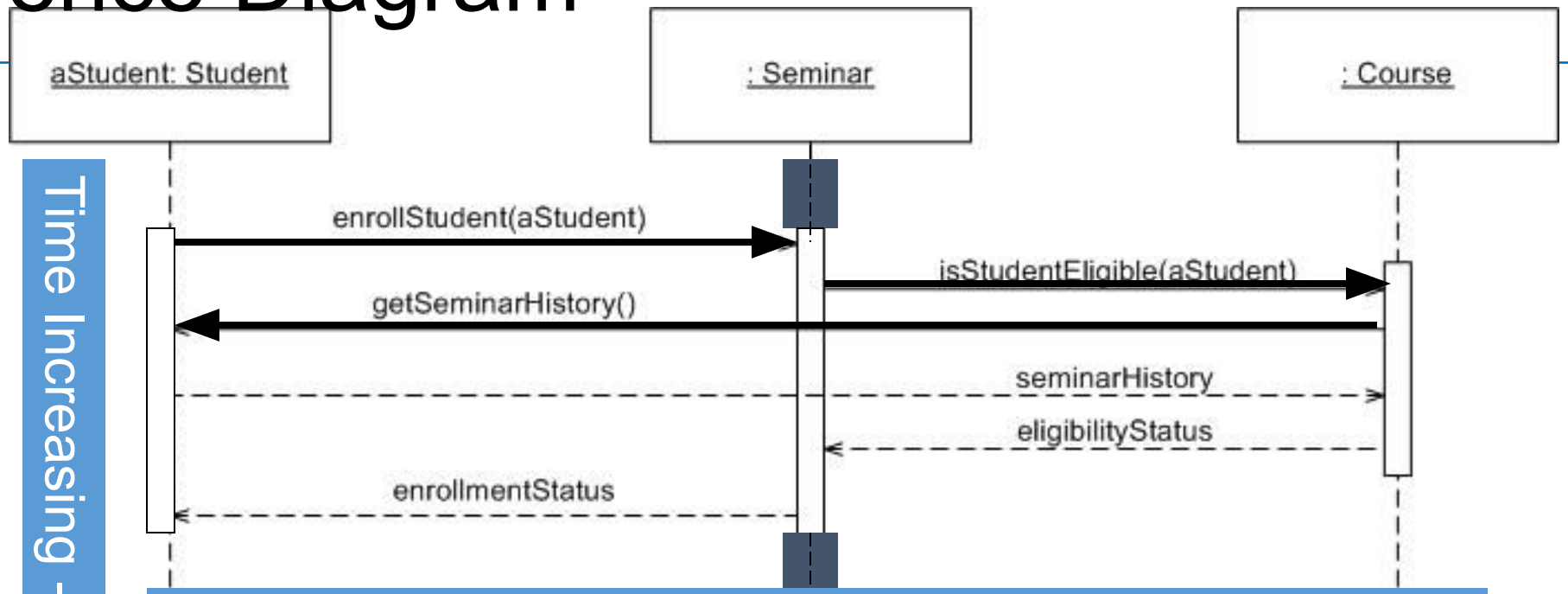
- Describe the flow of messages, events, actions between objects
- Show concurrent processes and activations
- Show time sequences that are not easily depicted in other diagrams
- Typically used during analysis and design to document and understand the logical flow of your system

Emphasis on time ordering!

Sequence Diagram

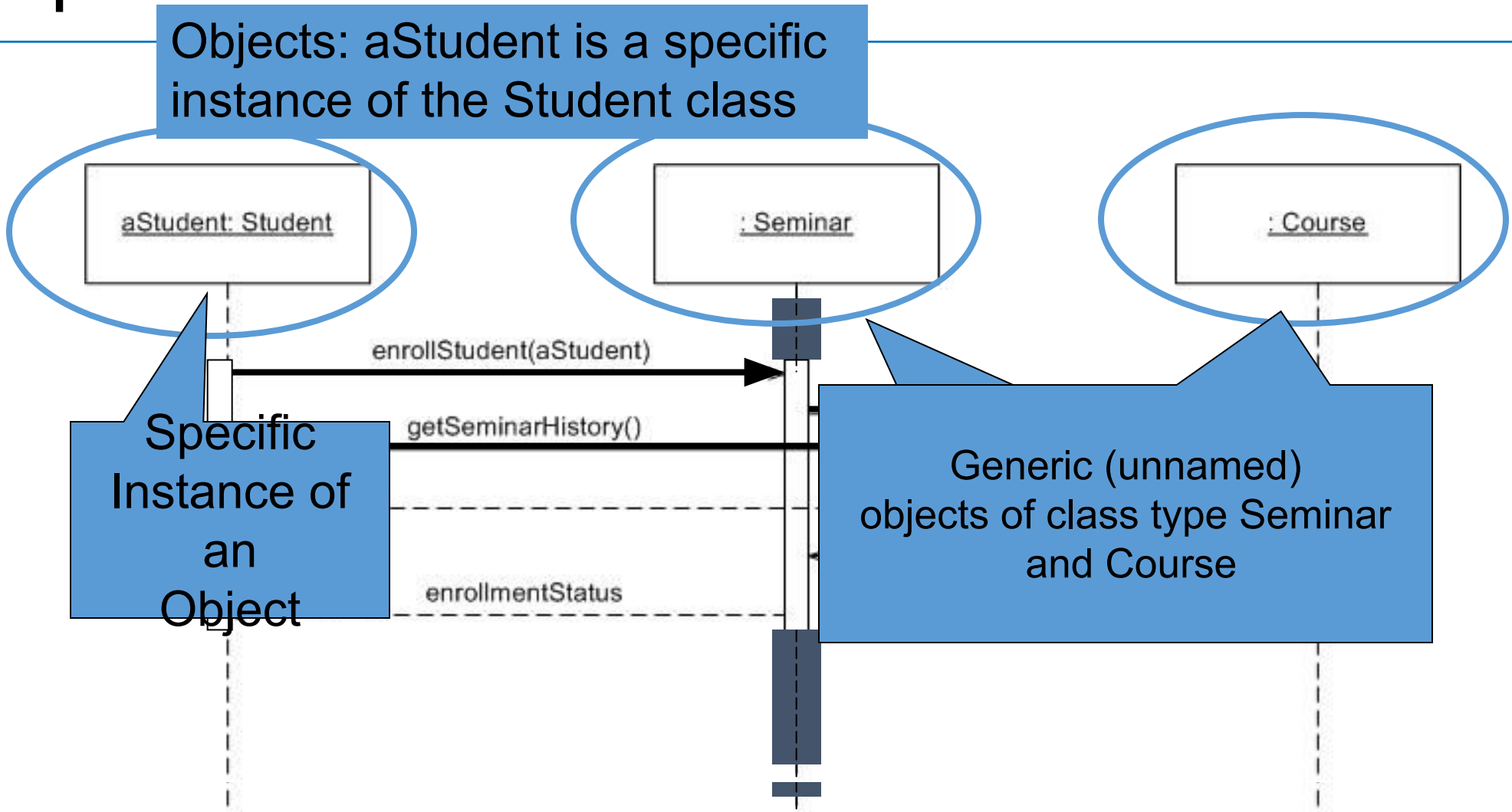


Sequence Diagram

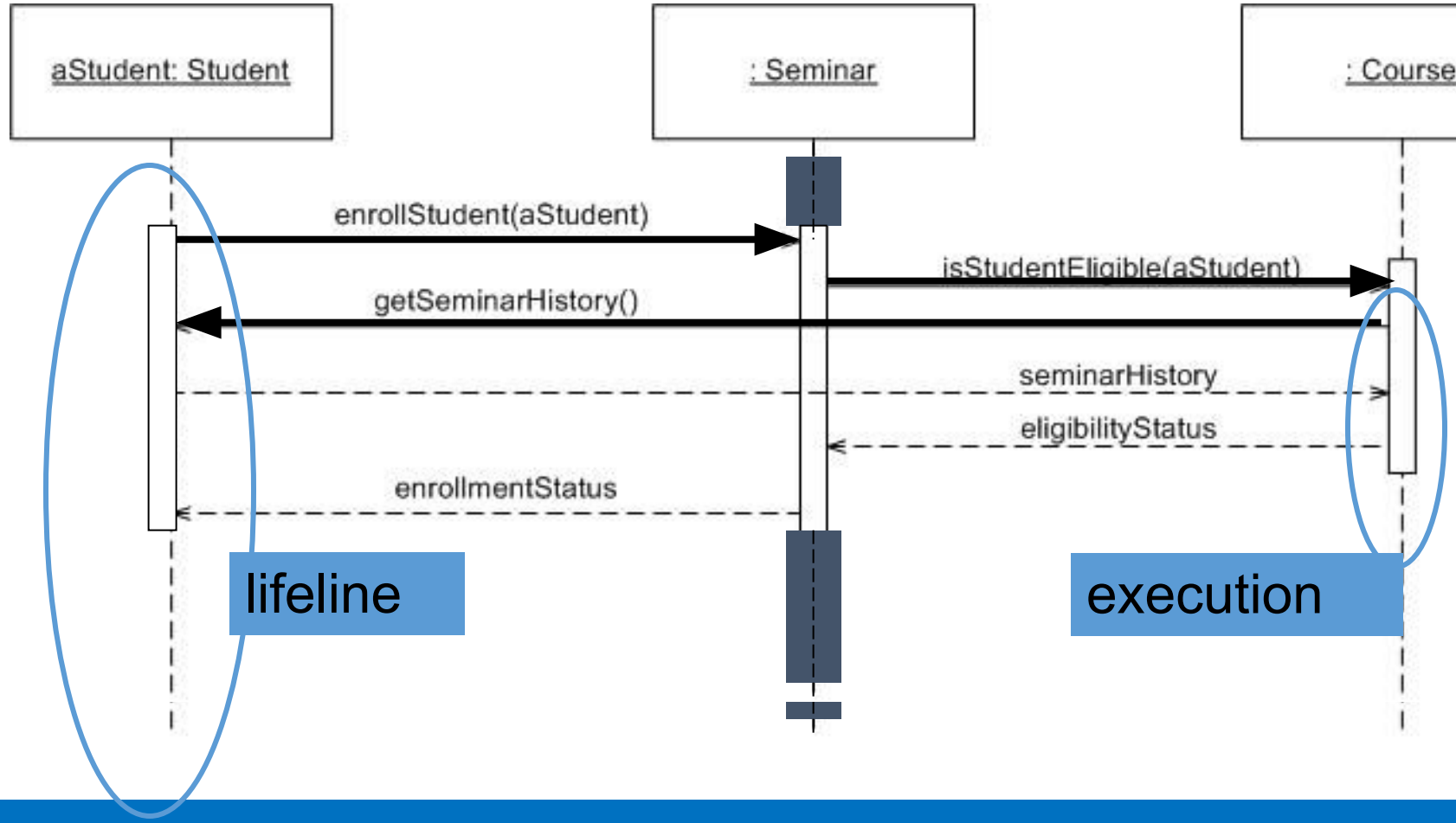


All lines should be horizontal to indicate instantaneous actions. Additionally if ActivityA happens before ActivityB, ActivityA must be above activity A

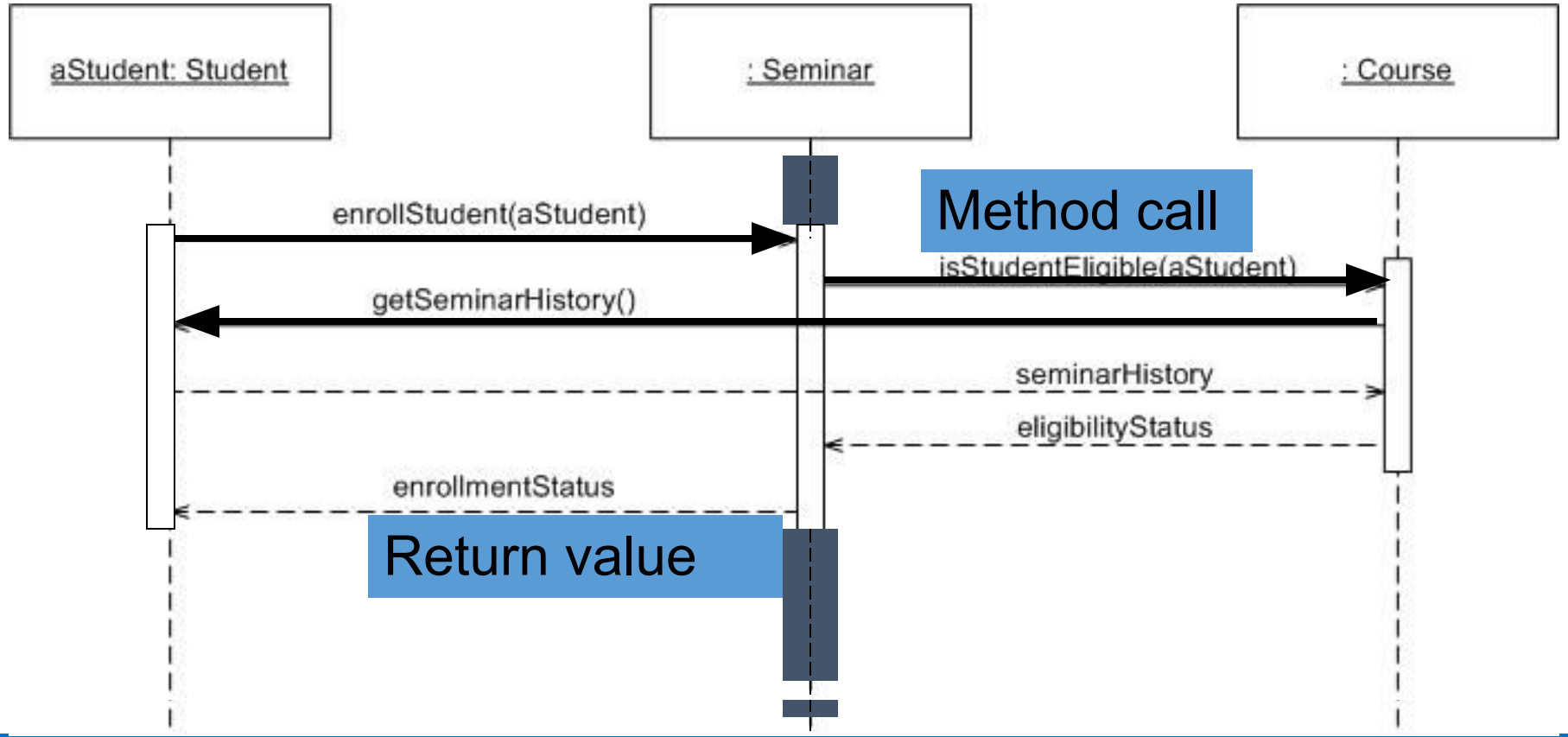
Components


Objects: aStudent is a specific instance of the Student class
aStudent: Student
: Seminar
: Course
enrollStudent(aStudent)
getSeminarHistory()
enrollmentStatus
Specific Instance of an Object
Generic (unnamed) objects of class type Seminar and Course

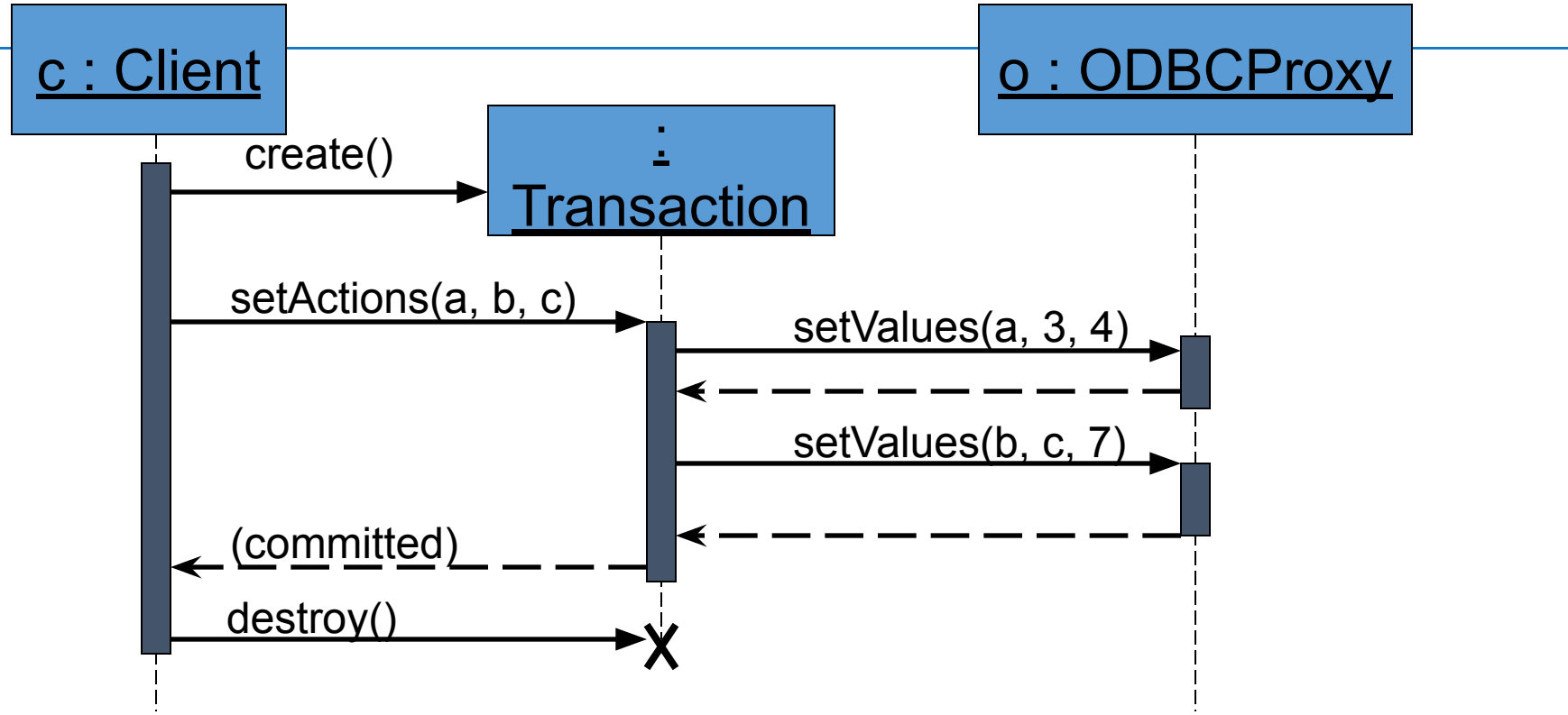
Components


aStudent: Student
: Seminar
: Course
enrollStudent(aStudent)
isStudentEligible(aStudent)
getSeminarHistory()
seminarHistory
eligibilityStatus
enrollmentStatus
lifeline
execution

Components


aStudent: Student
: Seminar
: Course
enrollStudent(aStudent)
Method call
isStudentEligible(aStudent)
getSeminarHistory()
seminarHistory
eligibilityStatus
enrollmentStatus
Return value

Components



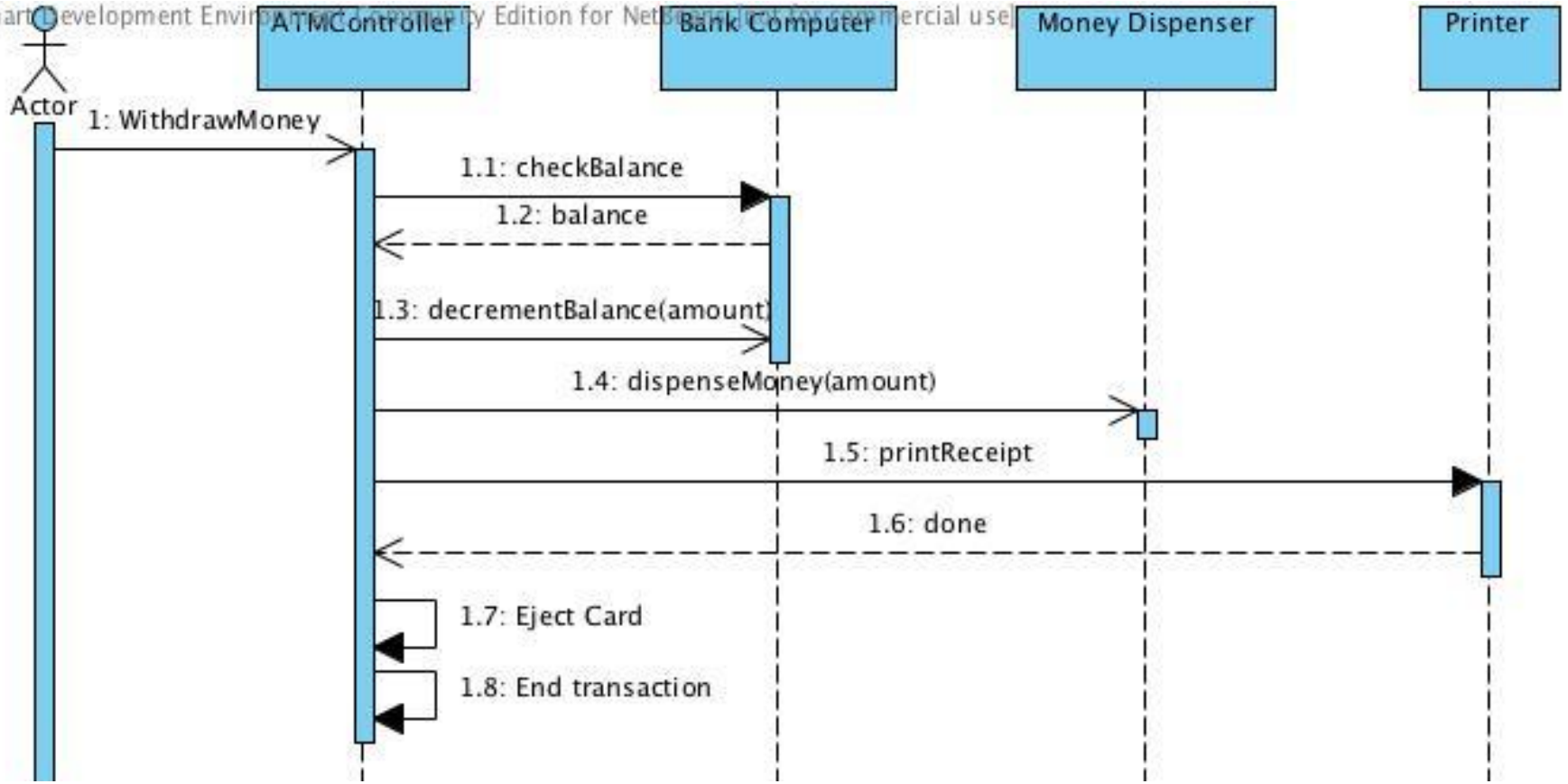
create()
destroy()

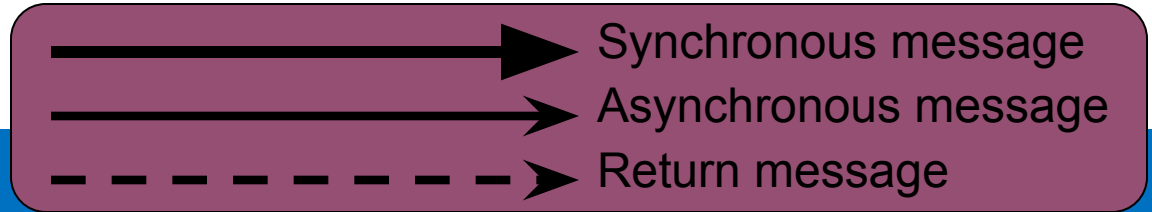
Synchronous message

Asynchronous message

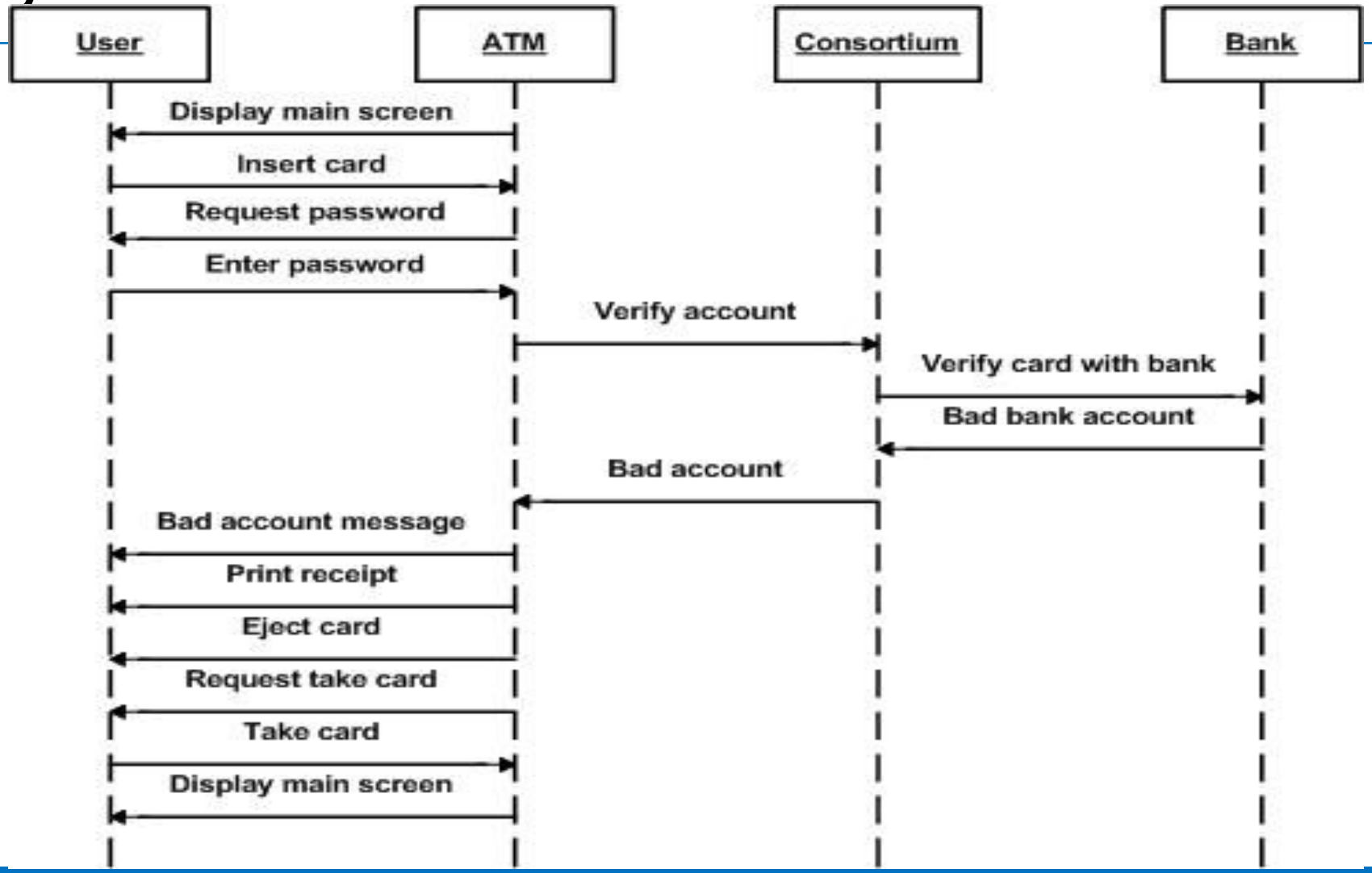
Return message

Async Message Example

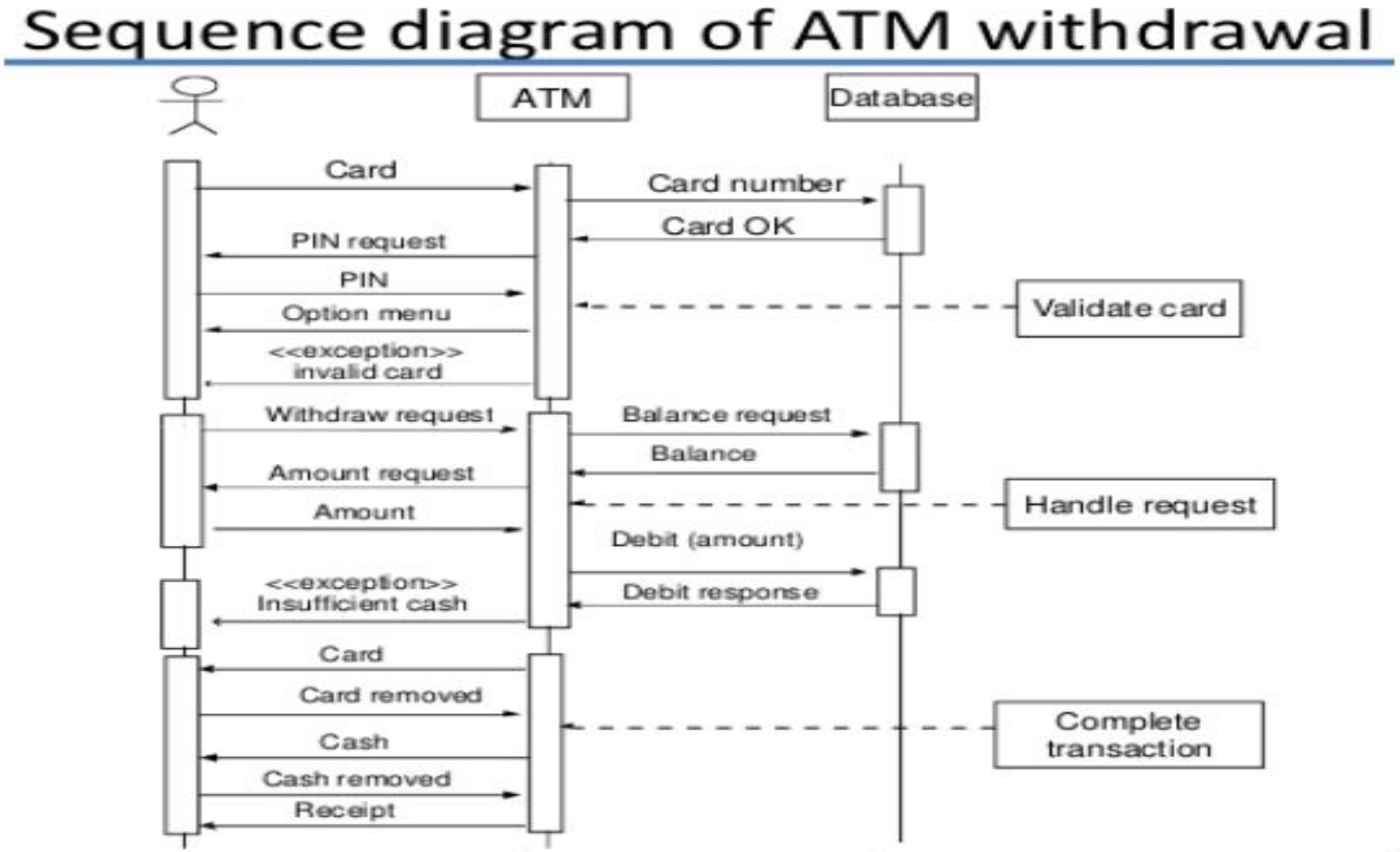




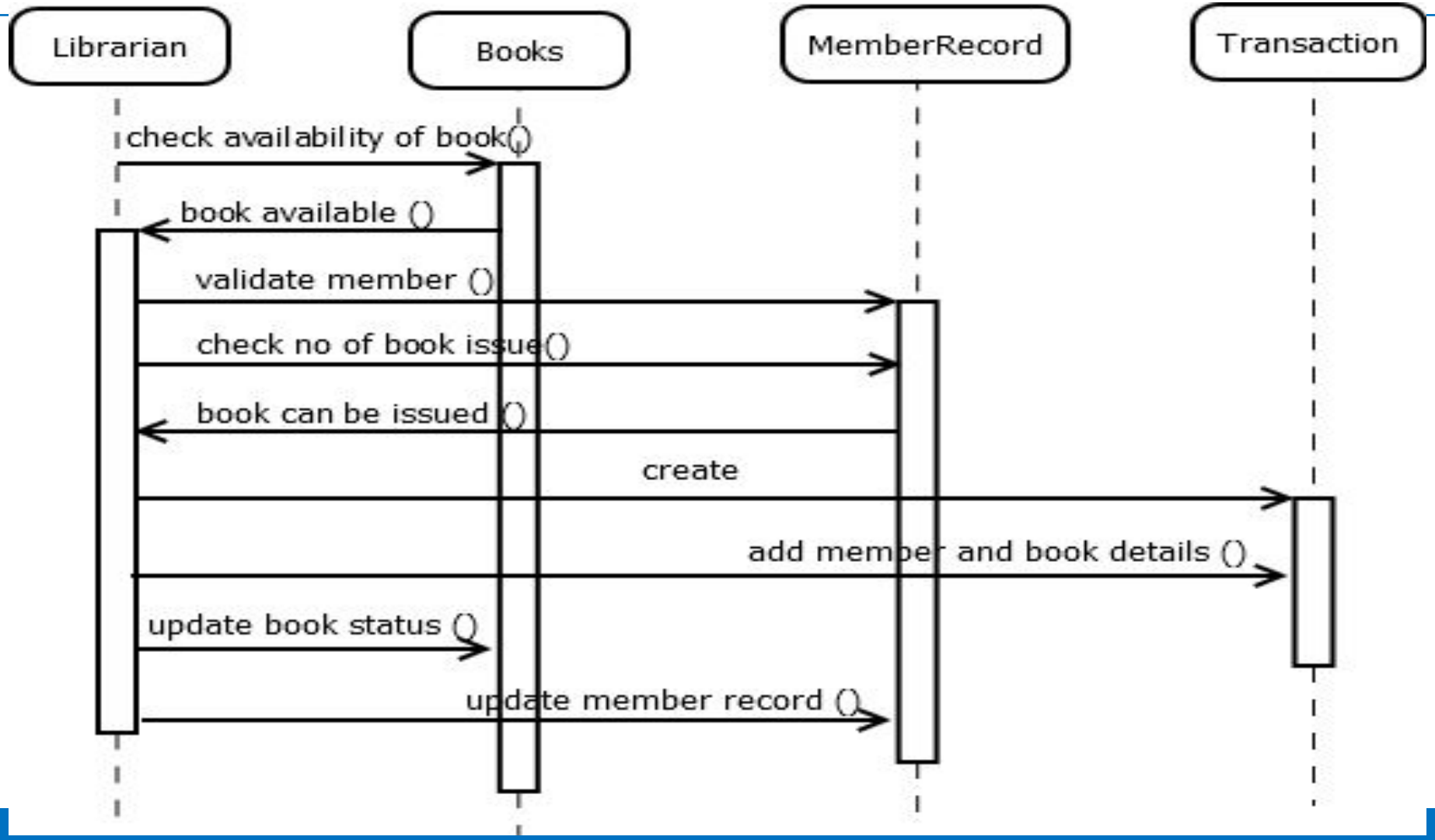
ATM System


User
ATM
Consortium
Bank
Display main screen
Insert card
Request password
Enter password
Verify account
Verify card with bank
Bad bank account
Bad account
Bad account message
Print receipt
Eject card
Request take card
Take card
Display main screen

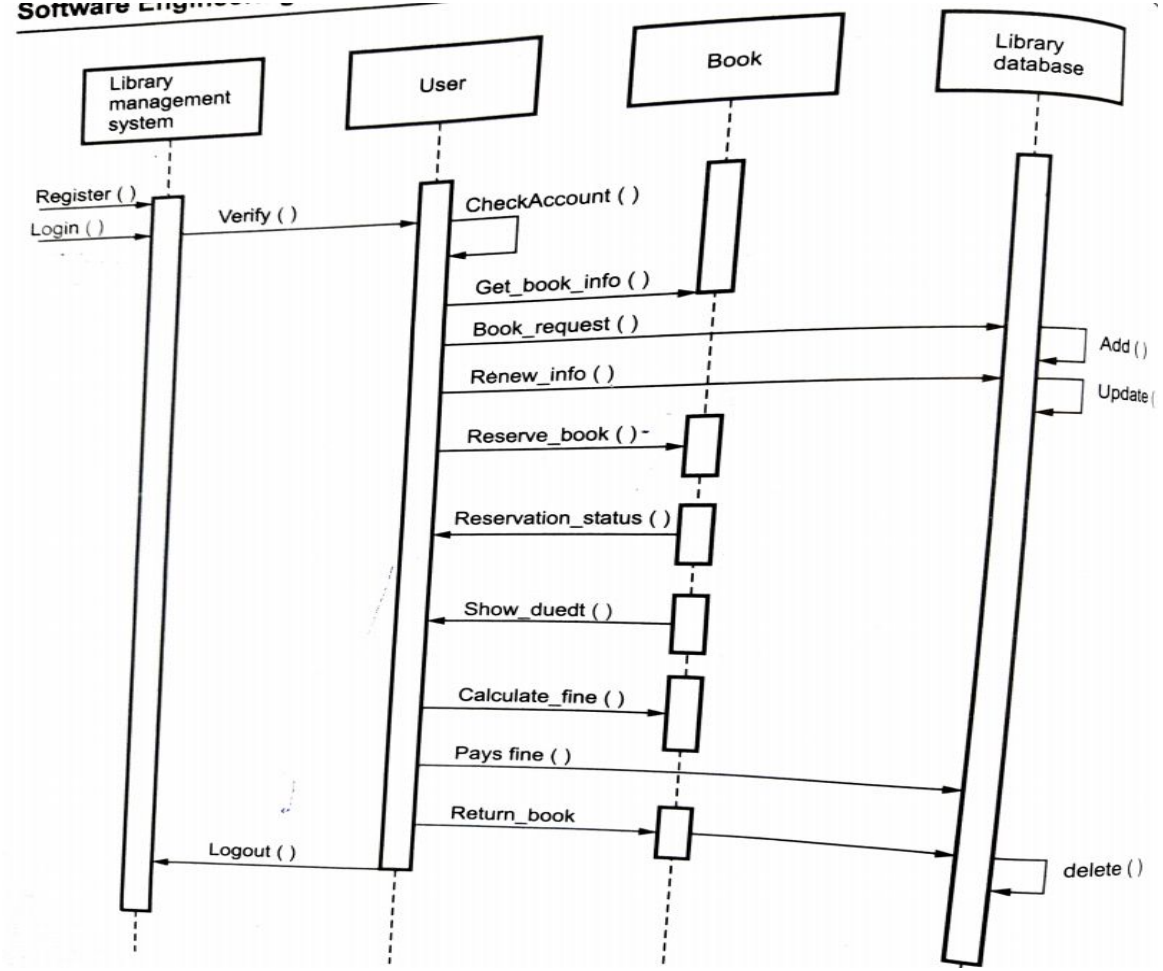
ATM


Sequence diagram of ATM withdrawal
ATM
Database
Card
Card number
Card OK
PIN request
PIN
Option menu
Validate card
<<exception>>
invalid card
Withdraw request
Balance request
Balance
Amount request
Handle request
Amount
Debit (amount)
<<exception>>
Insufficient cash
Debit response
Card
Card removed
Complete
transaction
Cash
Cash removed
Receipt

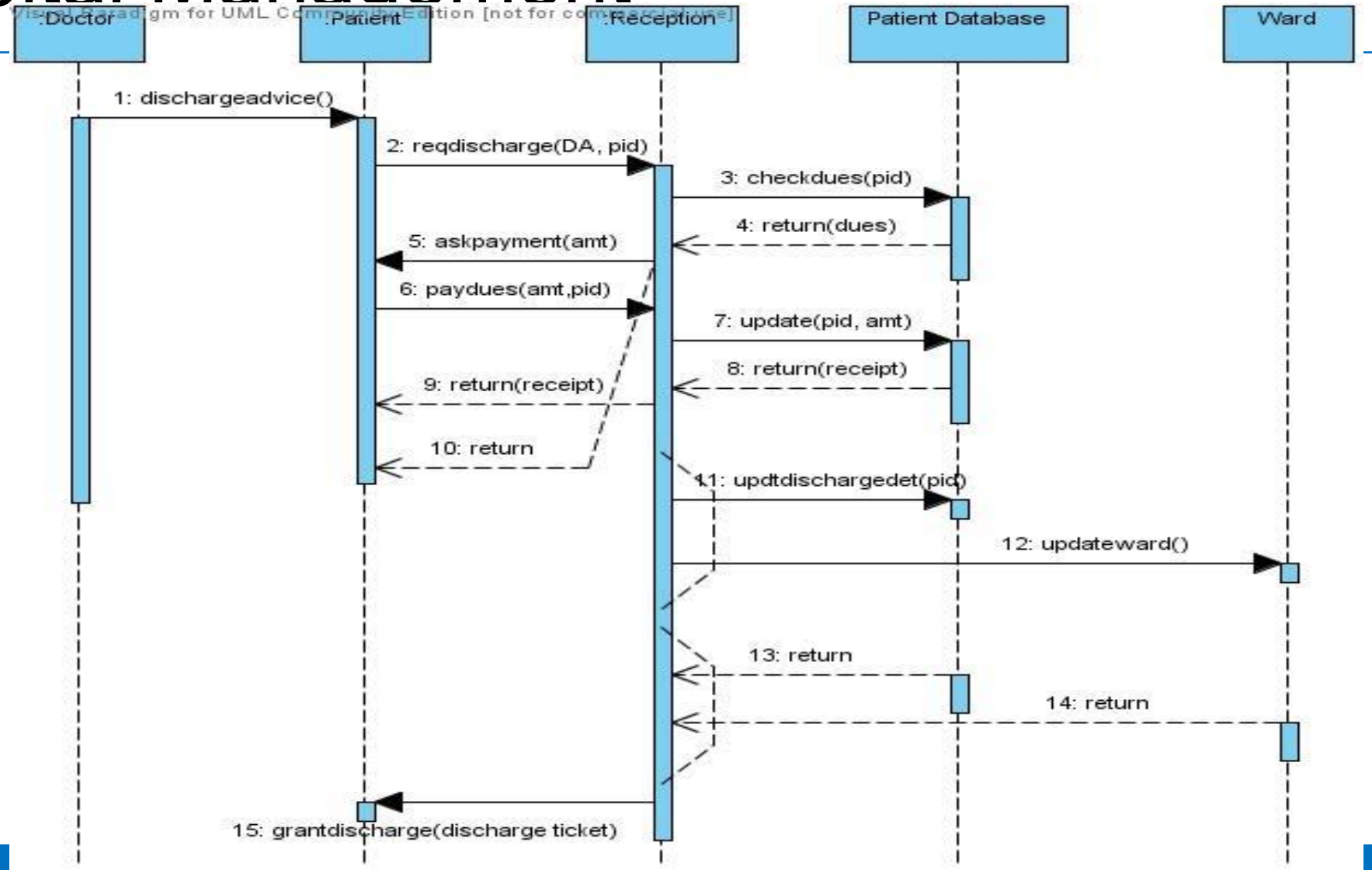
Library Management



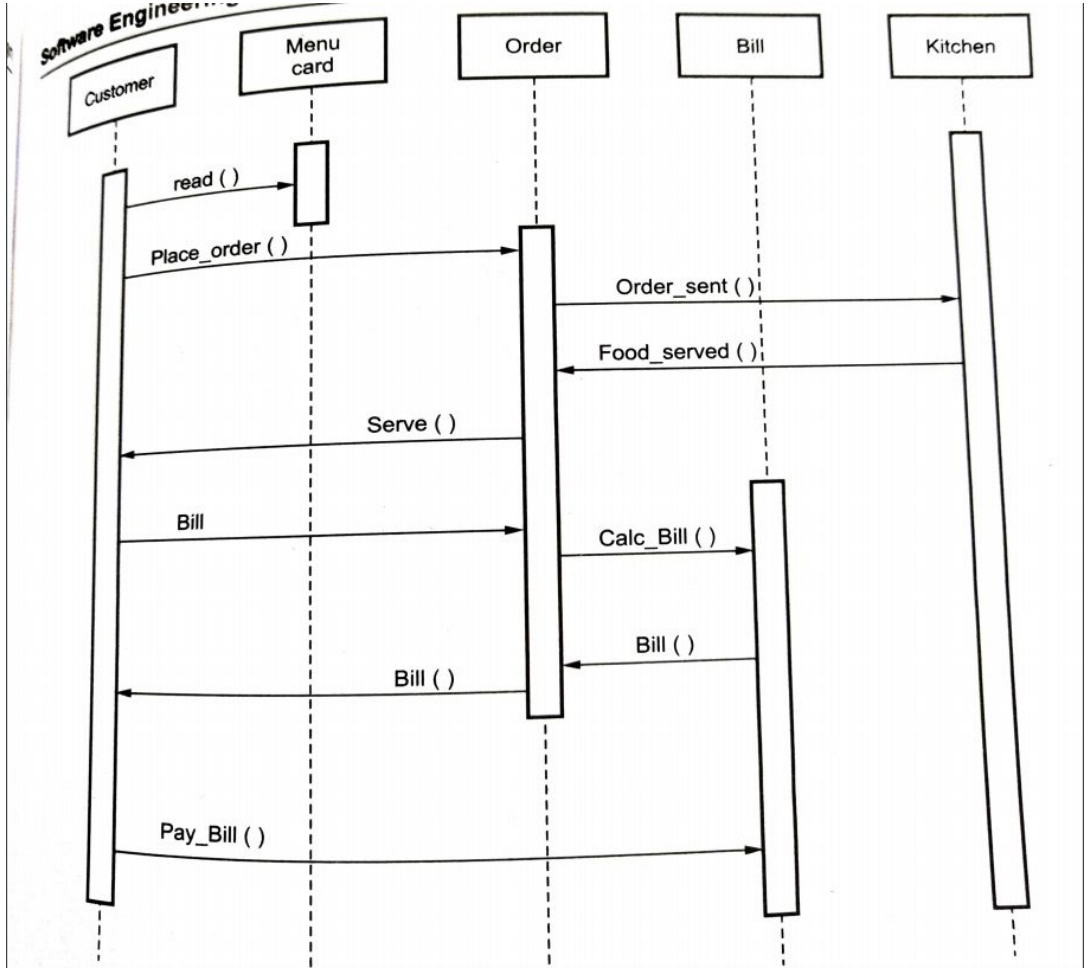
Library Management



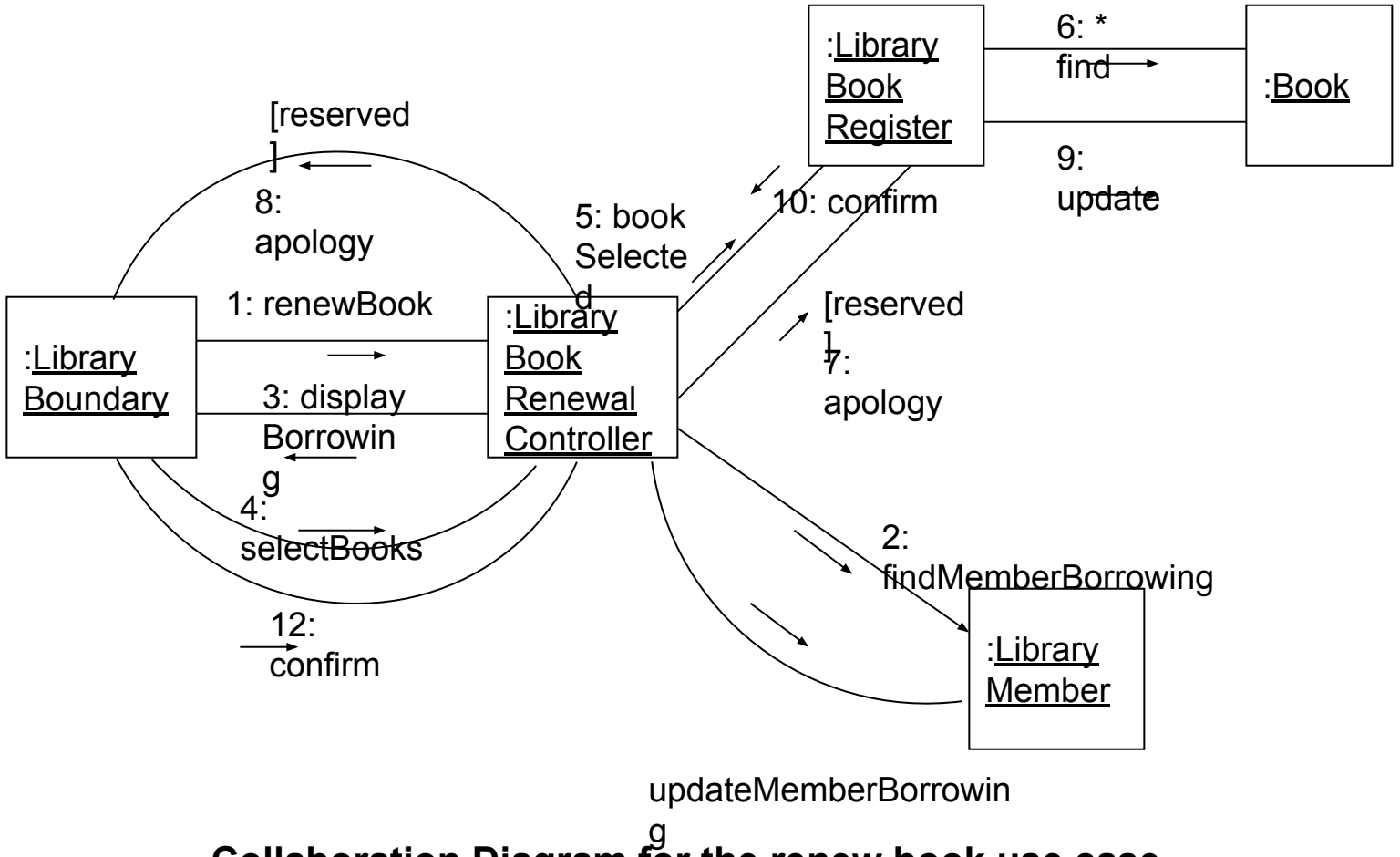
Hospital Management



Food ordering system



Example of Collaboration diagram



Collaboration Diagram for the renew book use case

Summary

- Sequence diagrams model object interactions with an emphasis on time ordering
- Method call lines
  - Must be horizontal!
  - Vertical height matters! “Lower equals Later”
  - Label the lines
- Lifeline – dotted vertical line
- Execution bar – bar around lifeline when code is running
- Arrows
  - Synchronous call (you’re waiting for a return value) – triangle arrow-head
  - Asynchronous call (not waiting for a return) – open arrow-head
  - Return call – dashed line