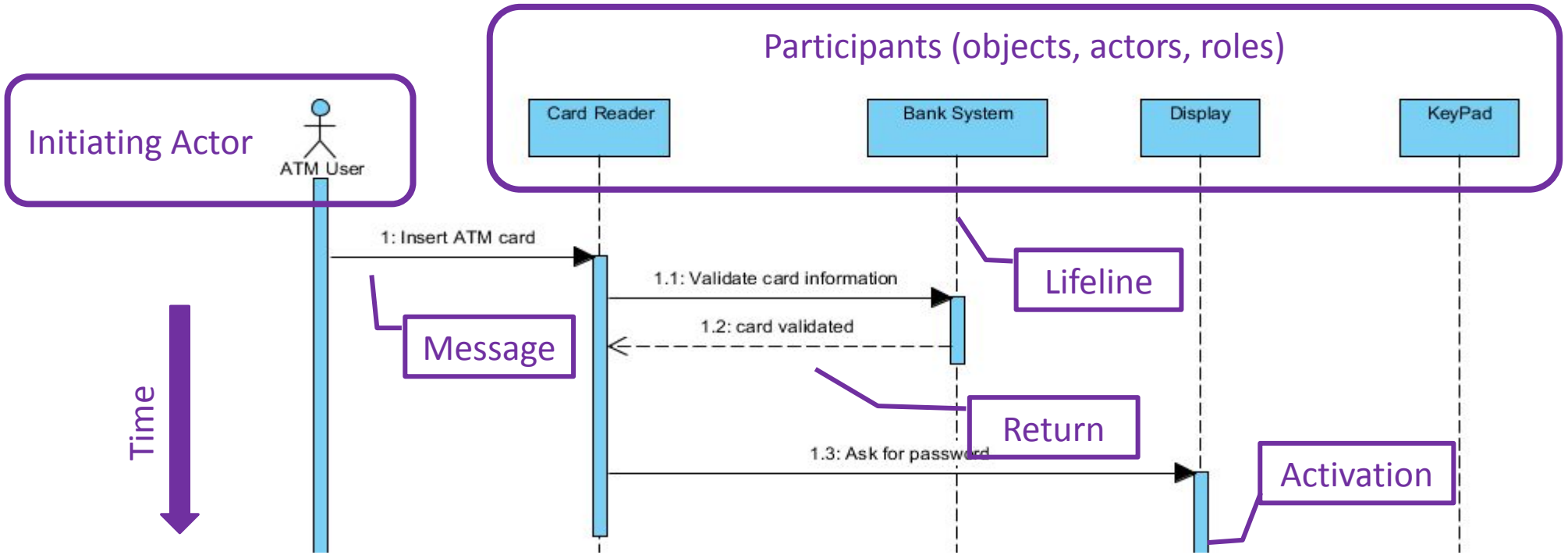
Lecture 11 – Sequence Diagrams

Learning Goals

- Describe the **purpose** of sequence diagrams
- Identify the **components** of a sequence diagram
- Discuss differences between sequence diagrams used during the **analysis phase** vs **design phase**
- Draw a sequence diagram for a given **use case scenario**
- Include **loops** and **conditionals** in a sequence diagram
- Convert a sequence diagram into a **class diagram**

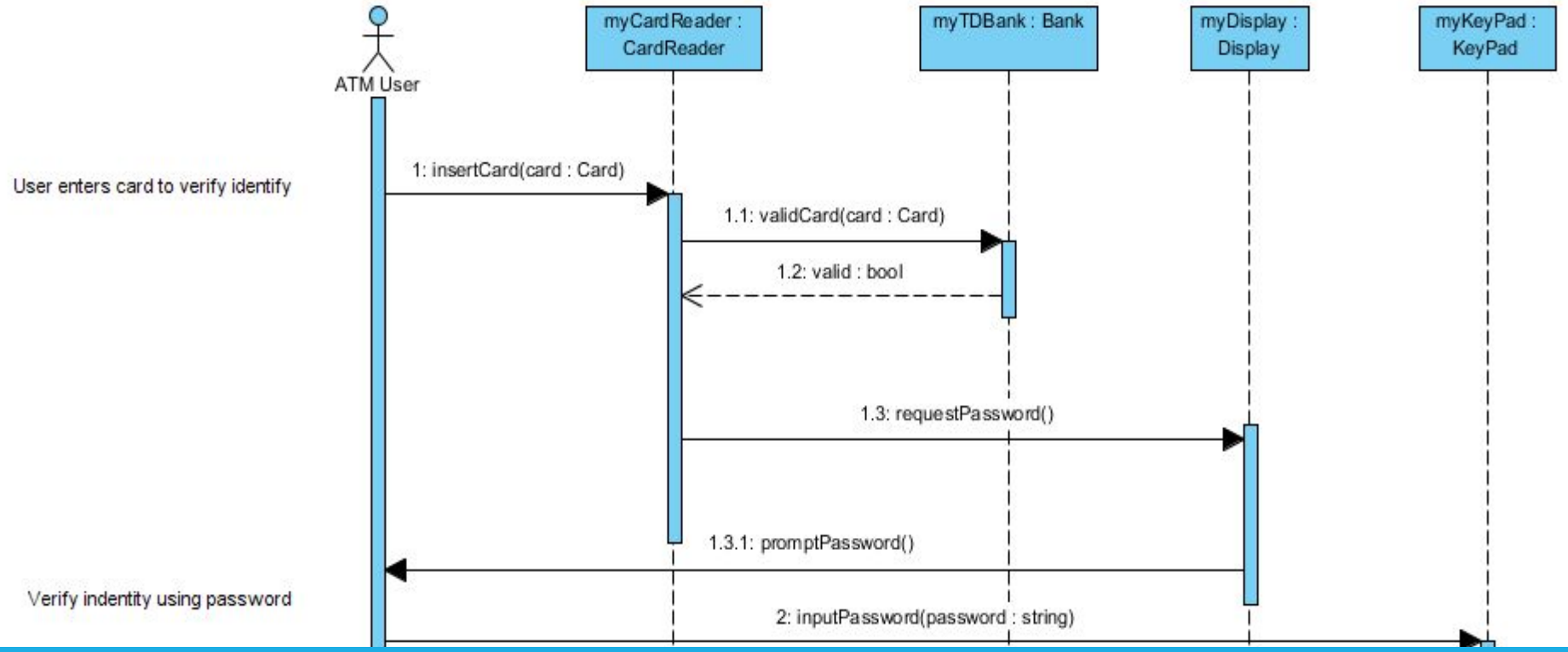
Sequence Diagrams

Sequence diagrams capture the step-by-step **sequence of events** in a single use-case **scenario**. They should capture the essence of user-interaction and document the response for **interesting** scenarios, involve only major objects/actors.



Sequence Diagrams

- Participants are usually objects, can be labelled using name : class syntax.
- Messages can either be descriptive, or methods using the name (arguments) syntax



Sequence Diagrams: Analysis vs Design

Sequence diagrams are often drawn at two different times during the software-development cycle: during the **analysis** phase, and the **design** phase.

During **analysis phase**, useful for communication with the customer, **minimal** technical/implementation details. Helps to ensure **understanding** of the process. Often describes an existing (perhaps manual) system.

During **design phase**, includes more implementation **specifics**. Can be translated into software objects/class diagrams.

Example: Analysis Model for Elevator

Optional comments:

User Enters character command

Io Process Checks for a command

If Present IO Process reads Command and Validates it

If Command is Valid, IO passes command to Dispatcher

If Command is up or down request

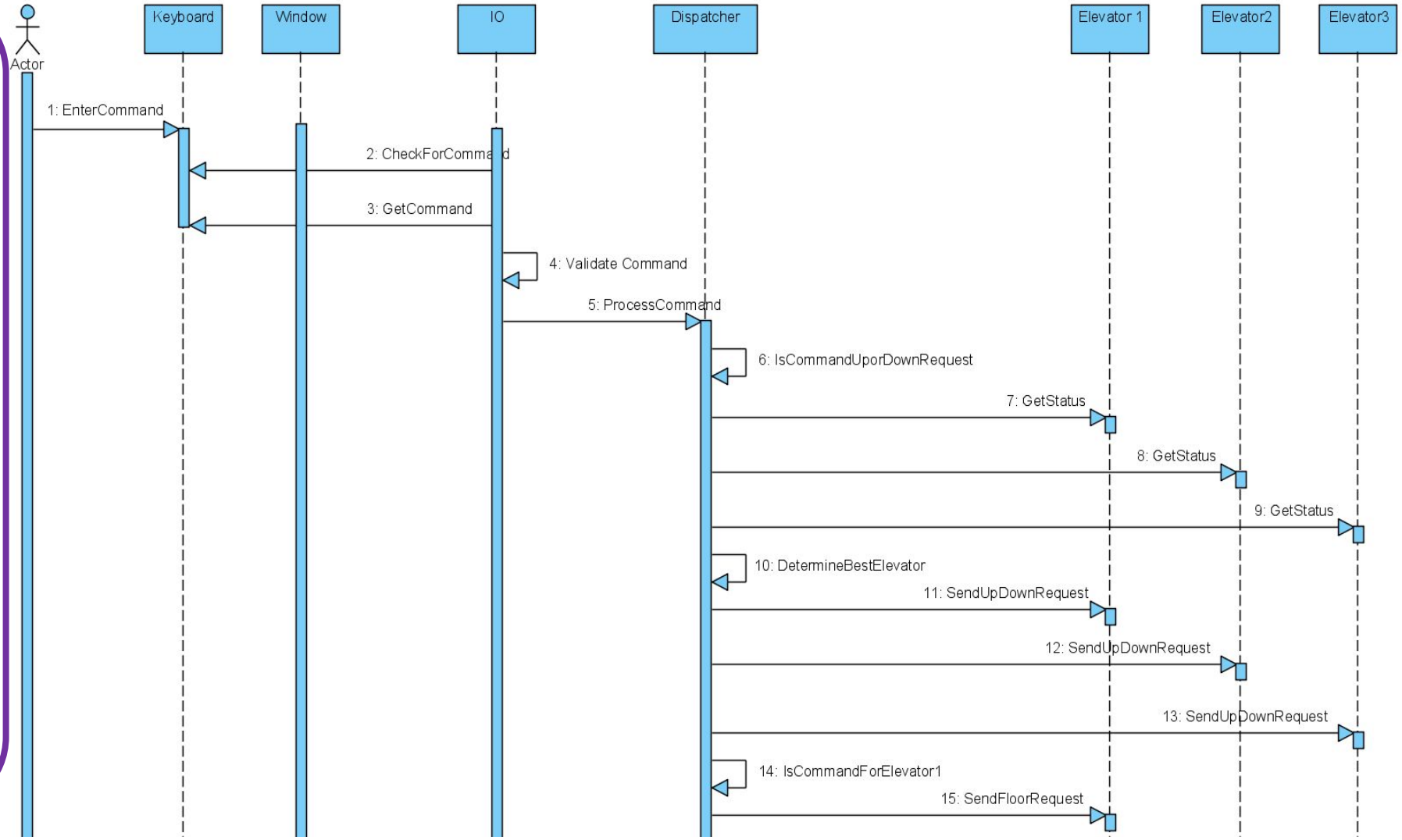
Read status of each elevator

Determine which elevator is best able to deal with it

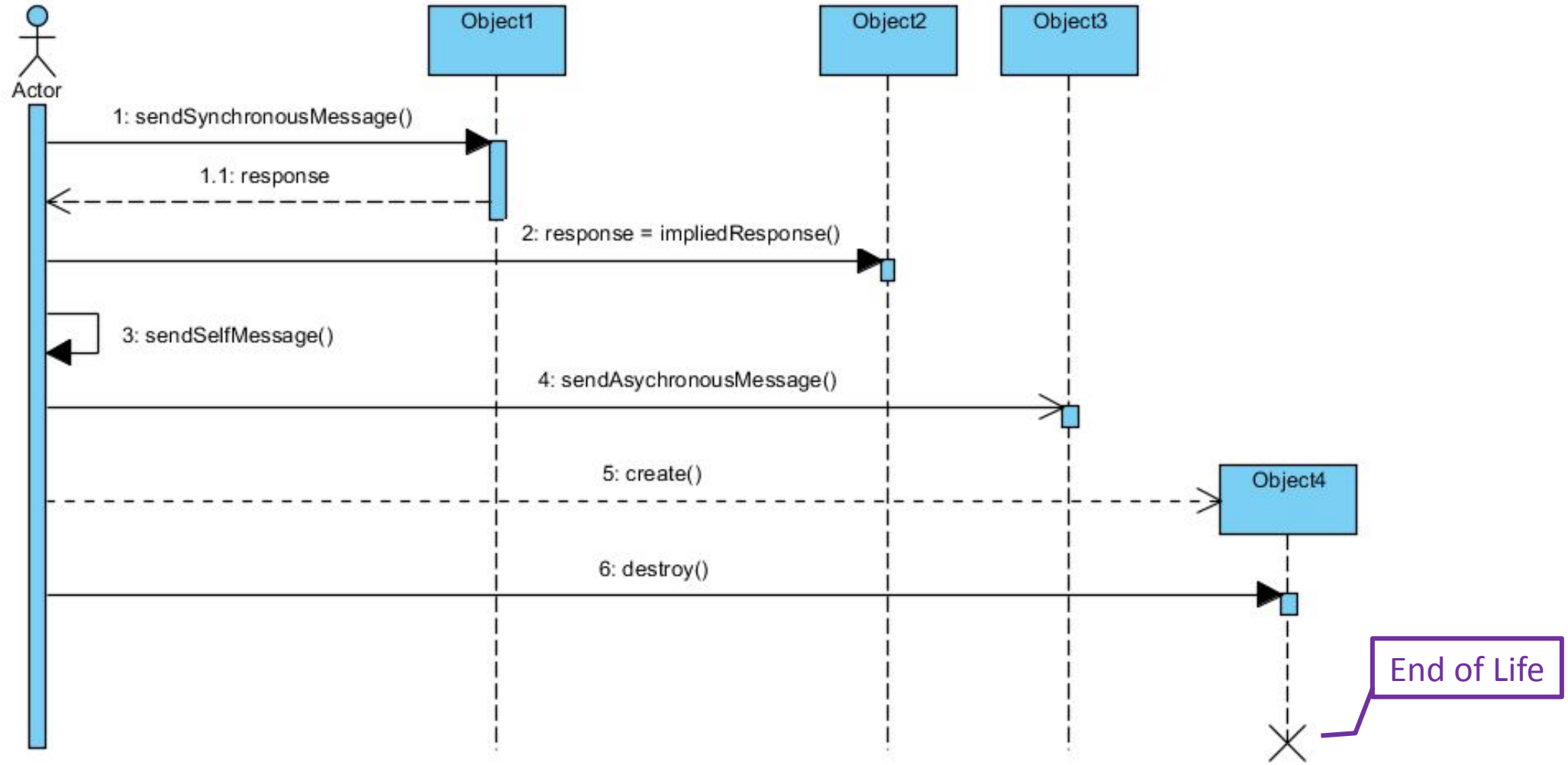
Send floor request to whichever elevator is best

If command is specific to one elevator

send floor request to that elevator



Sequence Diagrams: Message Types



Sequence Diagrams: Message Types

- **Synchronous**: This message requires a response before the interaction can continue. It's usually drawn using a line with a solid arrowhead pointing from one object to another.
- **Asynchronous**: This messages don't need a reply for interaction to continue. The arrowhead is usually open and there's no return message depicted.
- **Response**: This message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.
- **Self-message**: A message an object sends to itself, usually shown as a U shaped arrow pointing back to itself.
- **Create**: This is a message that creates a new object. Similar to a return message, it's depicted with a dashed line and an open arrowhead that points to the rectangle representing the object created.
- **Delete/Destroy**: This is a message that destroys an object. It can be shown by an arrow with an x at the end.
- **Lost/found:** These messages are for when the other participant is unknown (more details later)

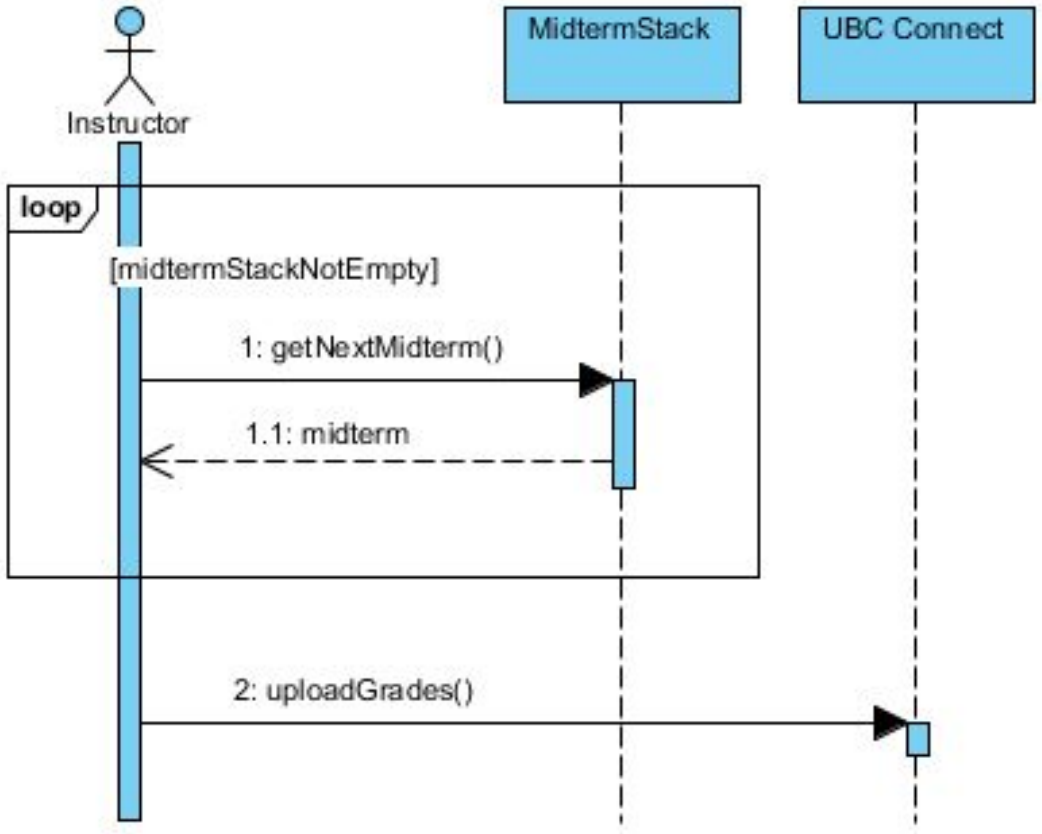
Sequence Diagrams: Decisions/Iterations

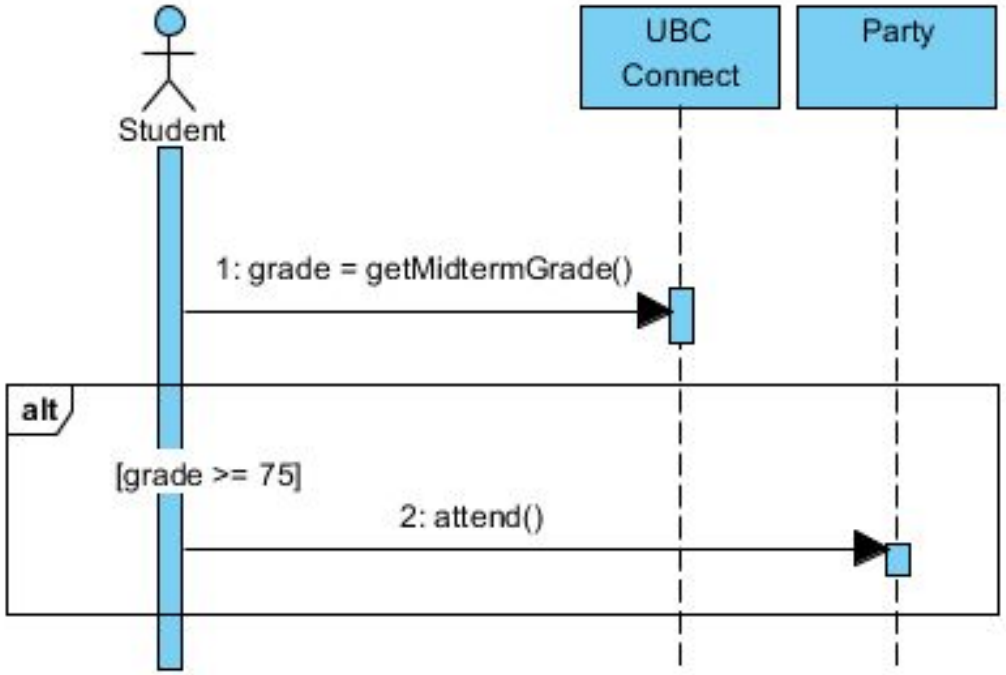
Sequence diagrams are ***not*** flowcharts. They are more useful when looking at **sequences** of events and **timings**.

When loops/decisions are important, we can include them **combination fragments**.

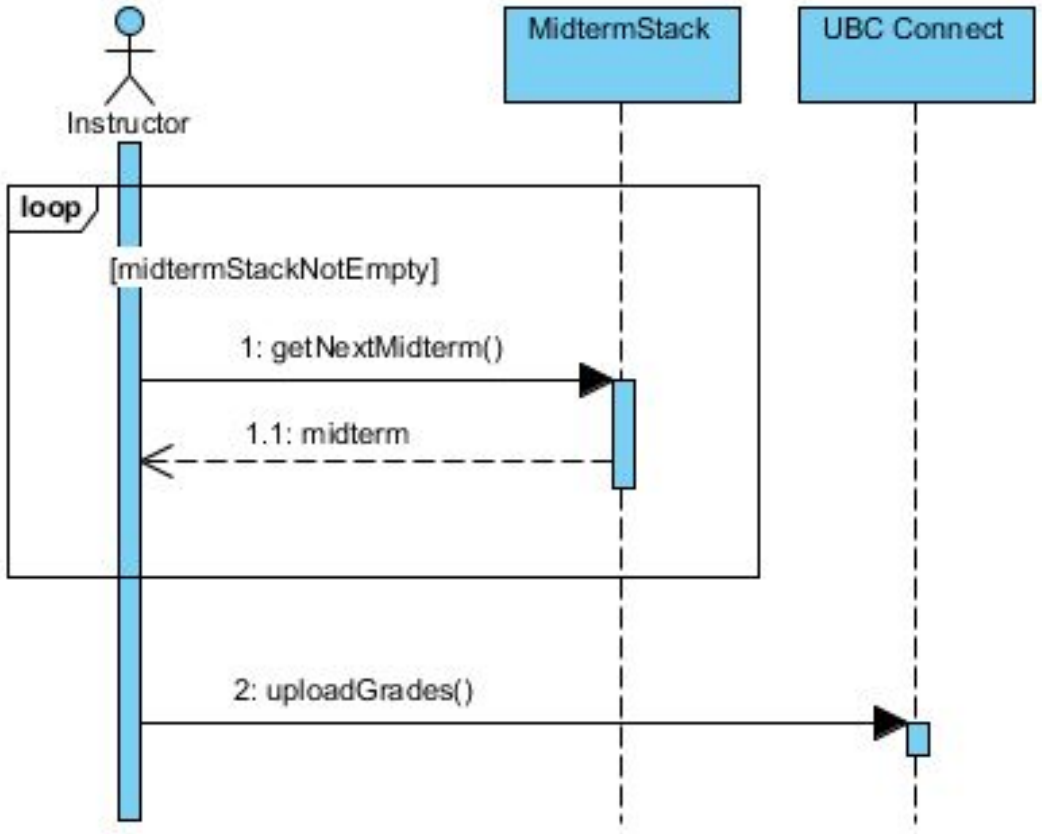
Opt	Optional fragment that executes if (Condition) is true.
Alt	Alternative fragment for mutually exclusive logic based on if-else type decision.
Loop	Loop fragment: A sequence of messages that repeats while some Condition is true. (**Note**: Can also be written as *loop(n)* to indicate looping n times)
Par	Two or more sequences that execute in parallel.
Region	Critical region. A sequence of statements that can only be executed by one thread at a time, the implication is that the designer will have to implement some form of mutual exclusion (i.e. a mutex with wait and signal) to protect the code.

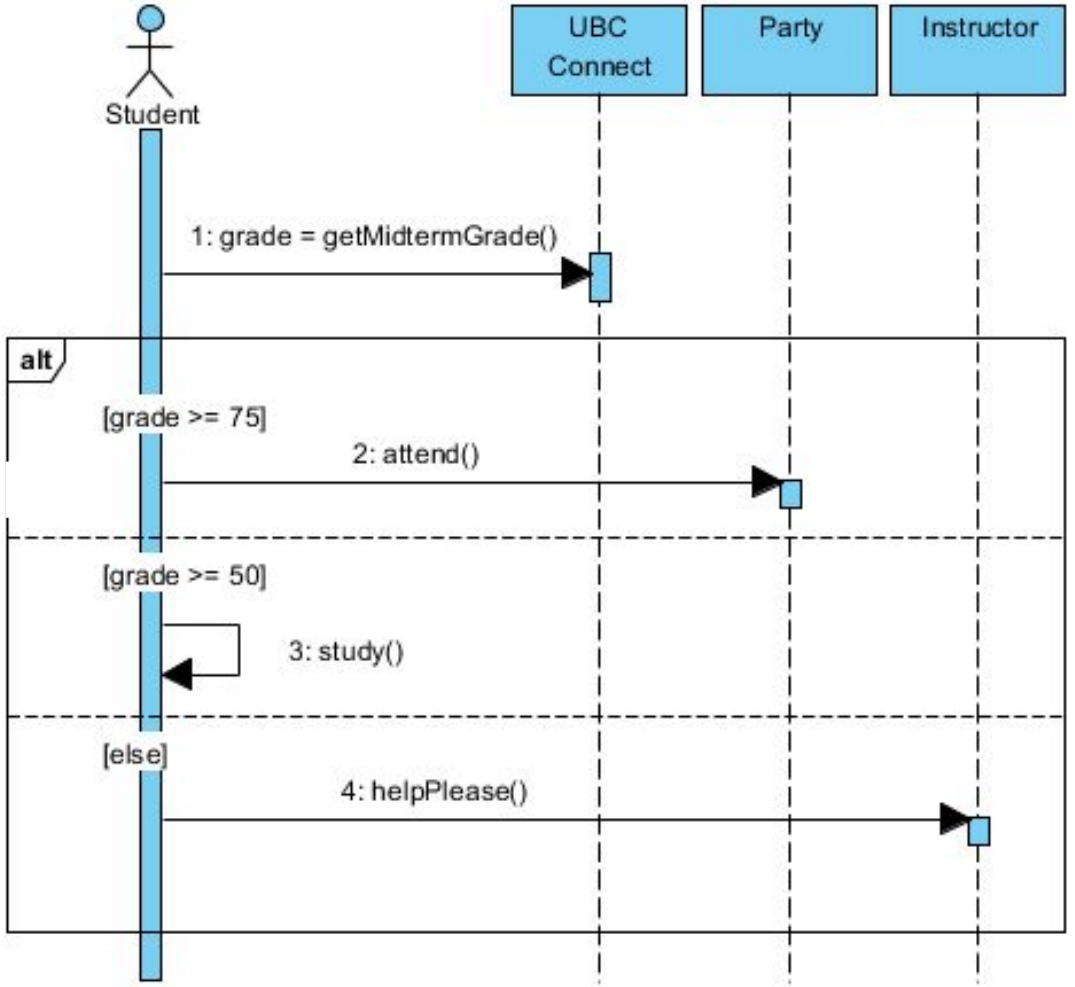
Sequence Diagrams: Decisions/Iterations





Sequence Diagrams: Decisions/Iterations





Class Example: Withdraw Cash

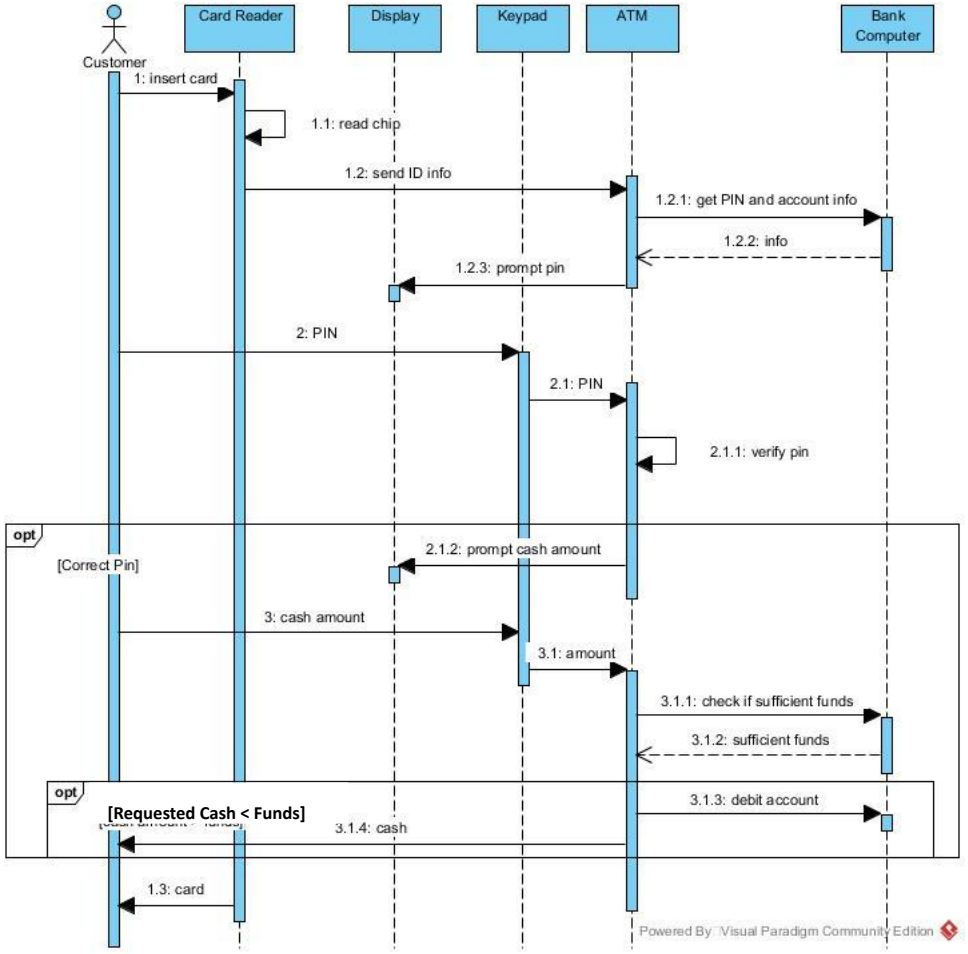
Create a Sequence Diagram for the following use-case scenario

Use Case Withdraw cash

- The user inserts their bank card into the card reader of the ATM
- The system reads the chip to identify user & account.
- The system contacts the bank to request the PIN number for the card and account details.
- The system prompts the user to enter their PIN.
- The user enters their PIN.
- The system prompts for the amount of the cash withdrawal.
- The user enters the amount of the cash withdrawal.
- The system checks with the banks central computer to ensure sufficient funds
- The cash is dispensed and the customer's account at the bank is debited
- The card is returned to the user

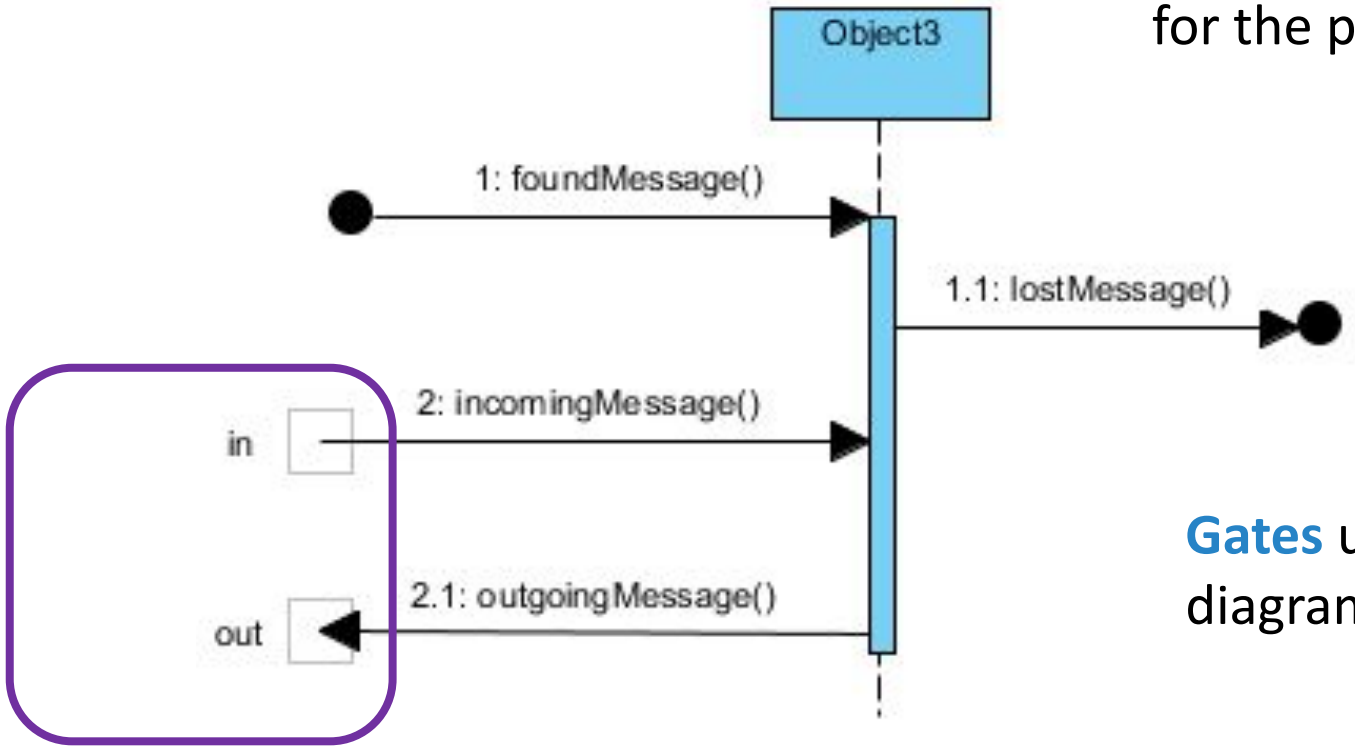
End

Class Example: Withdraw Cash



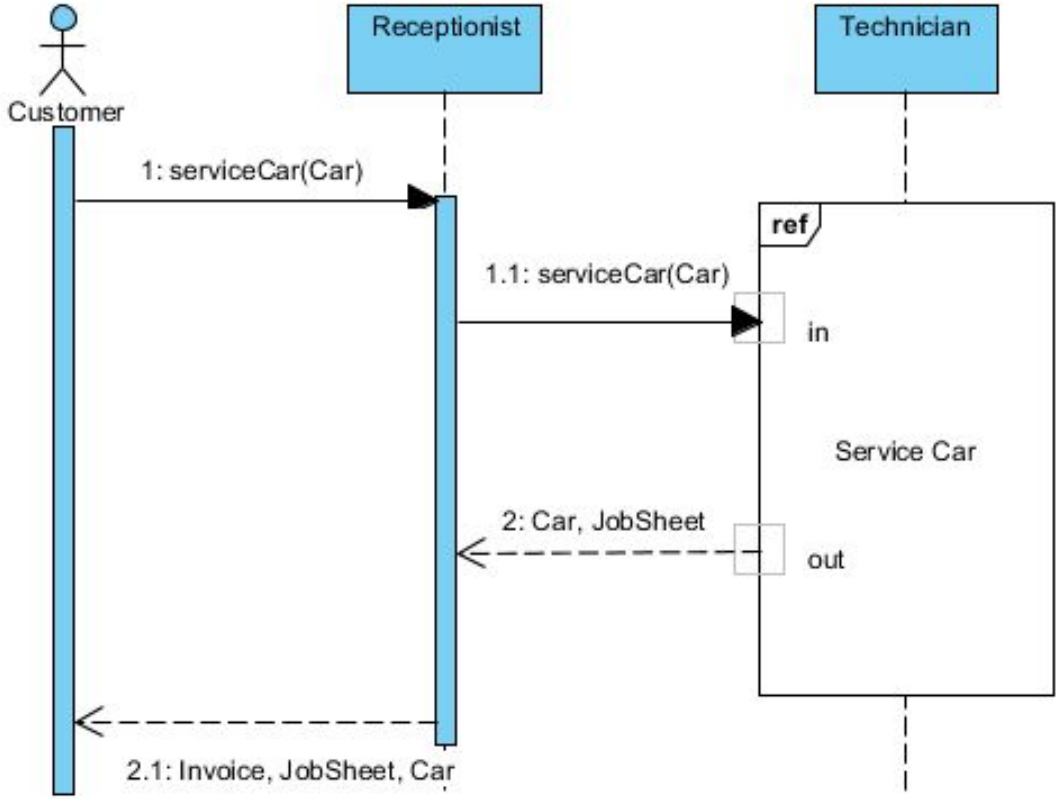
Sequence Diagrams: Incoming/Outgoing

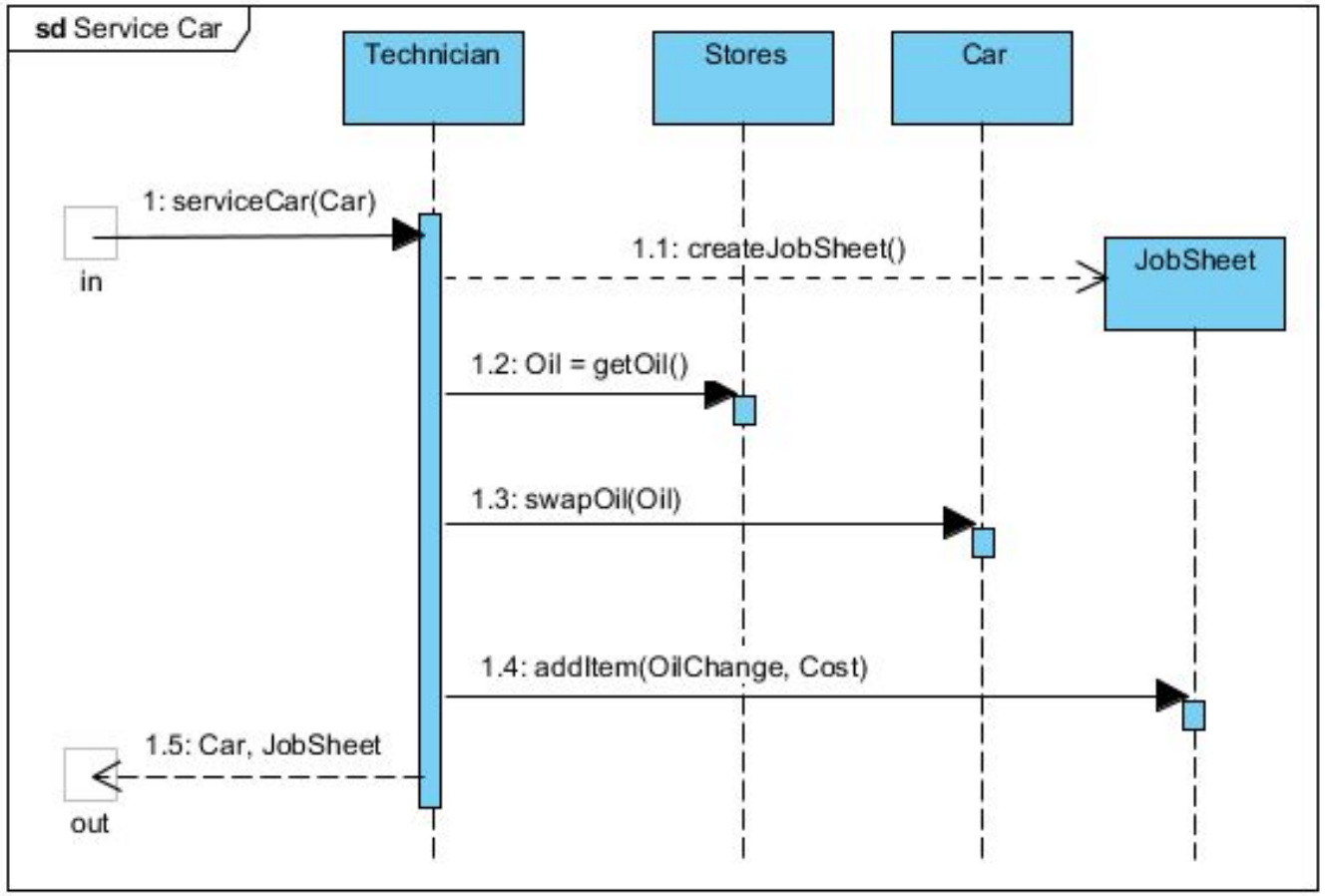
Lost and **found** messages are for when the other participant is unknown, or we don't care for the purposes of this diagram.



Gates usually connect to other sequence diagrams.

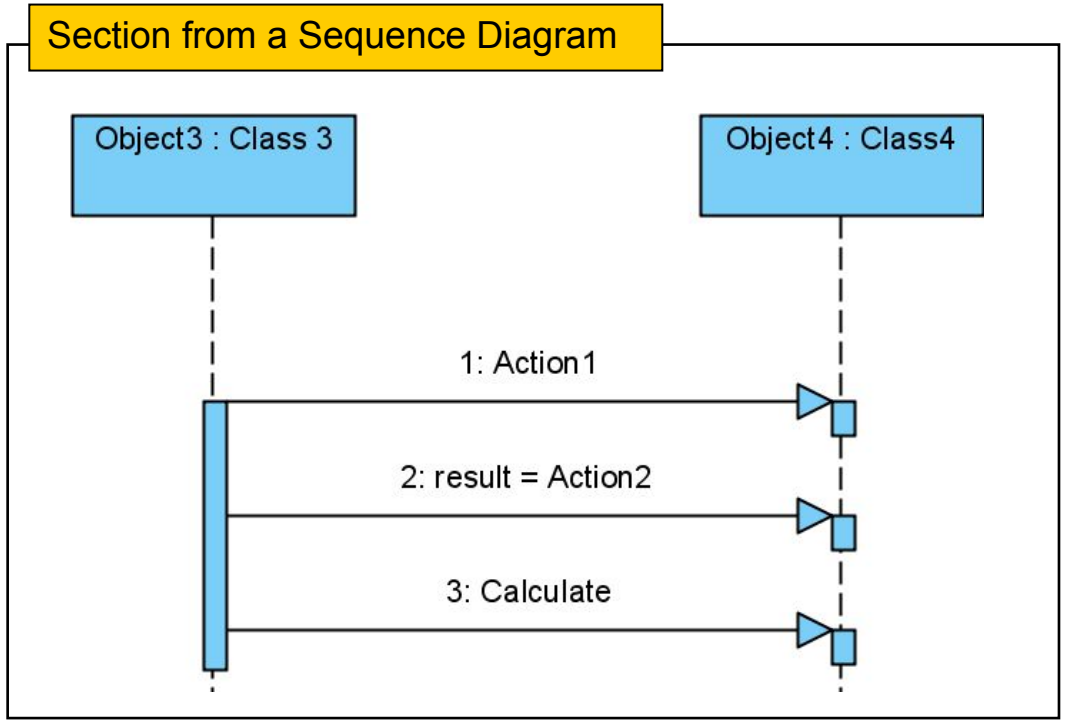
Sequence Diagrams: Hierarchies

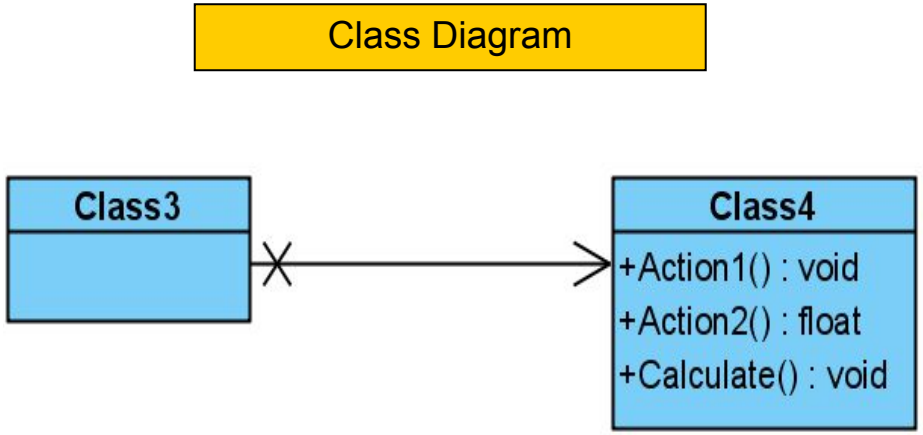




https://knowhow.visual-paradigm.com/uml/sequence-diagram-gate/

Sequence Diagrams to Class Diagrams

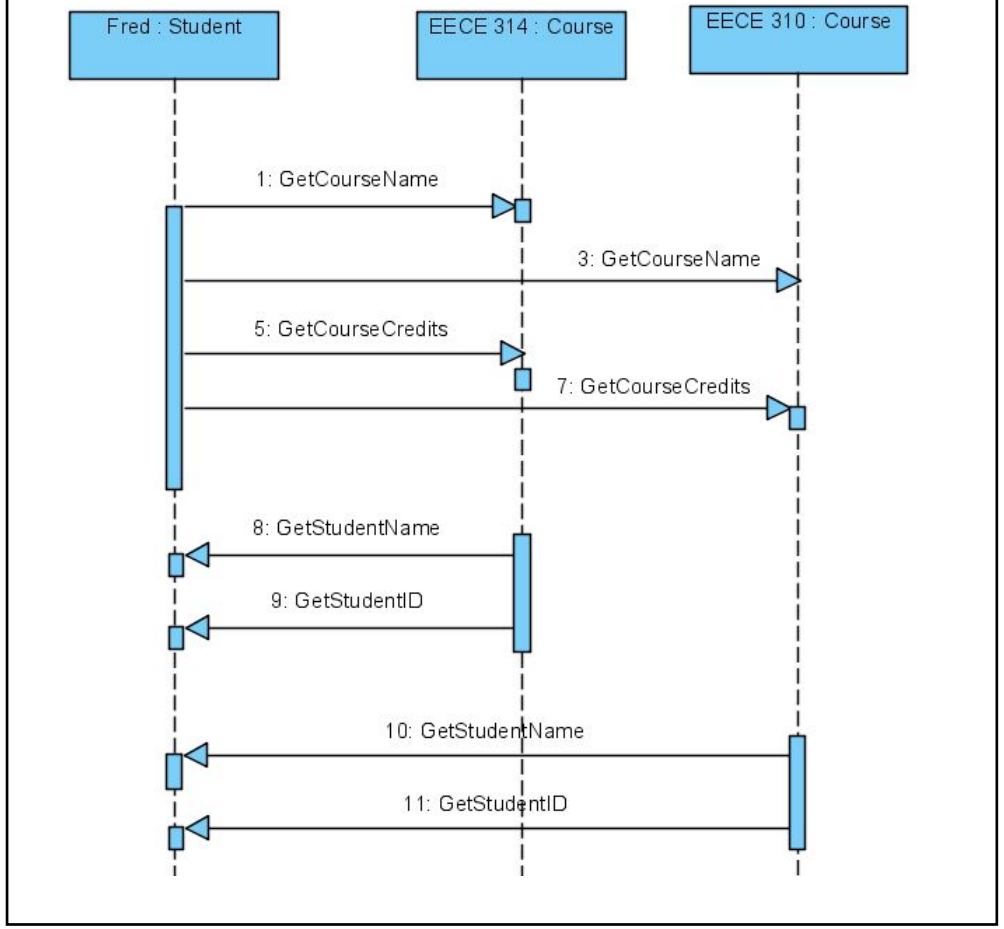




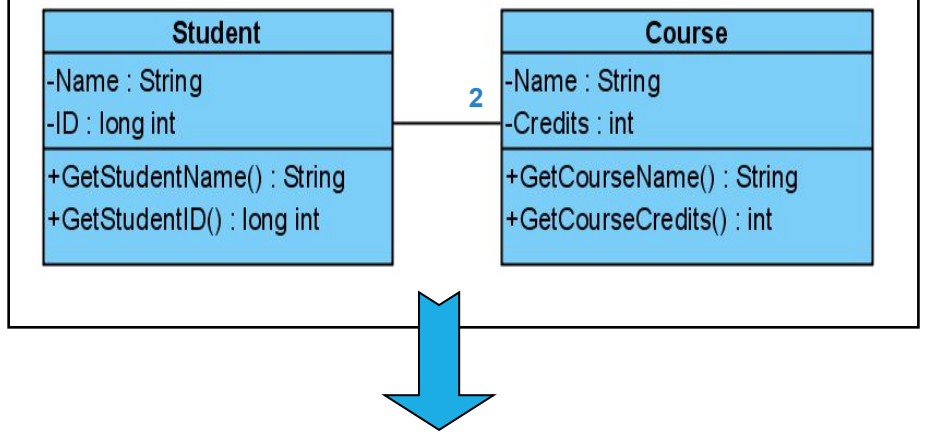
Sequence diagrams and Class Diagrams must be **consistent**. Messages usually correspond to operations, direction of arrows corresponds to direction of **association**.

Sequence Diagram to Class Diagram

Section from a Sequence Diagram



Maps to this Class Diagram




```
class Student {
    Course *course1 ;   // points at runtime to EECE314
    Course *course2 ;   // points at runtime to EECE310
    . . .
} ;
```

```
class Course {
    Student *theStudent ;   // points to 'fred' at runtime
     . . .
} ;
```