

# Outline

## ★ Use-case Diagram

- ★ Actors
- ★ A use-case should be abstract
- ★ Generalization, Specialization
- ★ Inclusion, Extension

## ★ Sequence Diagram

- ★ First step of designing
- ★ An example
- ★ Components of a sequence diagram
- ★ Collaboration diagram
- ★ Conditional/asynchronous message, etc

# Introduction to Use-case Diagram

## ★ Role played

- ★ Initial statement of the requirements.
- ★ As detailed as you can.
- ★ External visible behavior of the system

## ★ Components

- ★ Actors: the roles that users can play
- ★ Use case: the interaction between actors and the system

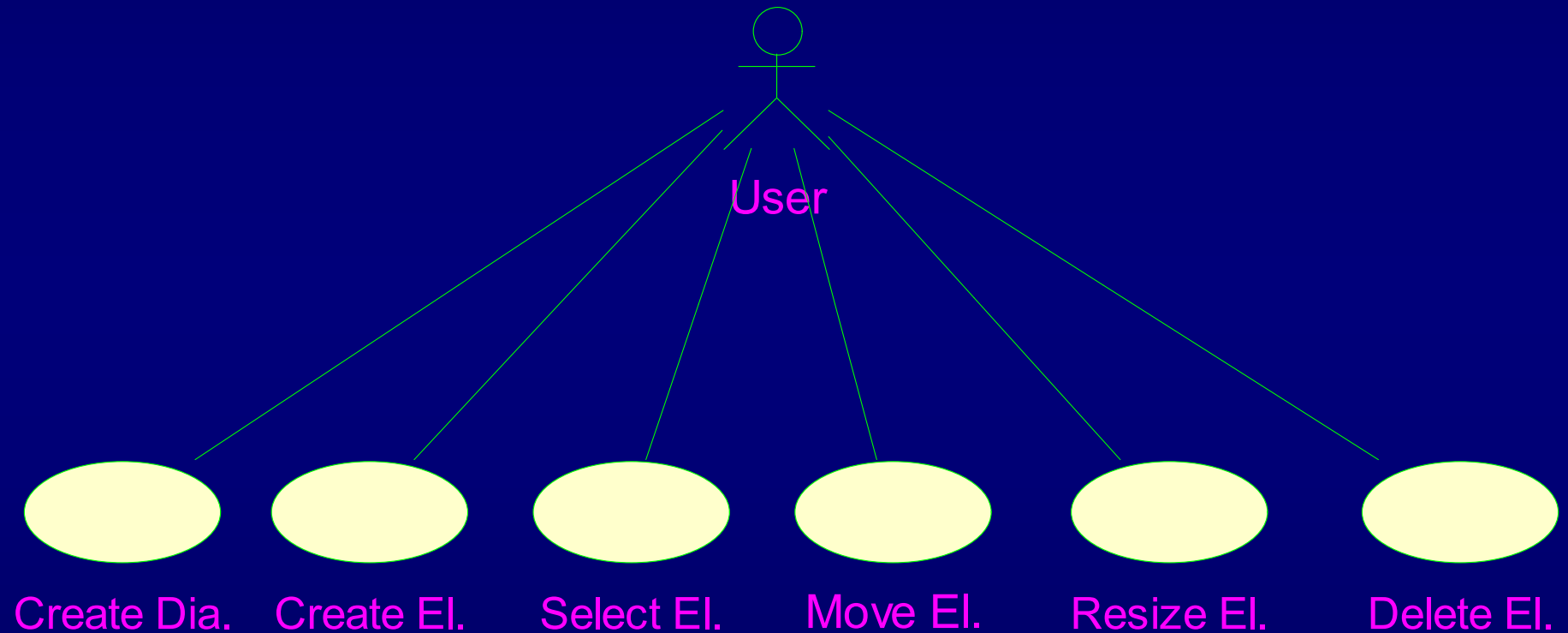
# Actors

- ✱ Usually: different groups of users
- ✱ No one-to-one correspondence between a single user and an actor
- ✱ Not necessarily be human
  - ✱ Human
  - ✱ Computer systems
  - ✱ Some devices

# Use case

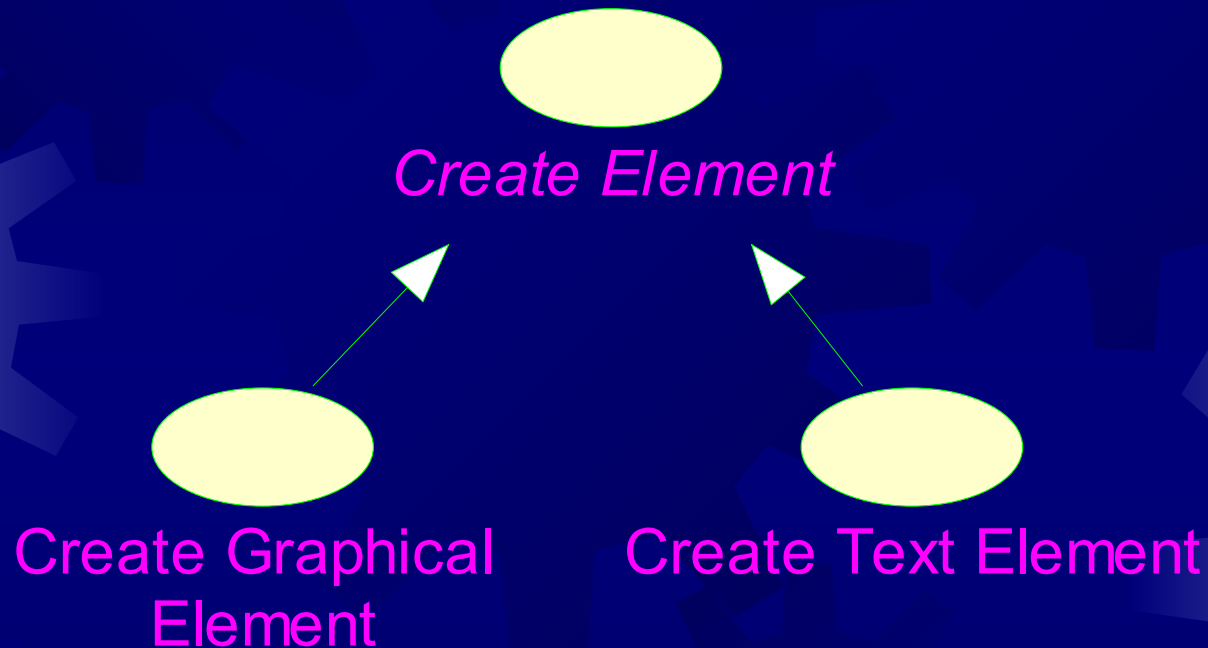
- ★ Should be abstract
  - ✱ Not every piece of operation can be viewed as a use case
  - ✱ Exceptions can NOT be viewed as separate use cases.
- ★ Definition: a description of a whole class of interactions that have the same overall intention.
- ★ Consists of
  - ✱ A basic course of events
  - ✱ Alternative courses
  - ✱ Exceptional courses

# The use case diagram of the diagram editor example



Scenario

# Generalization and Specialization



Abstract use case  
Explanation for a use case

# Inclusion of use case

- ★ Stereotype: specification of a relationship



# Realization of a use case

- ✦ Describe how a set of objects can interact with each other to implement the use case.
  - ✦ For designer: to build up an understanding of the objects, classes and interactions
  - ✦ The **first step** of design
  - ✦ Do not consider the GUI elements
- ✦ Notation: interaction diagrams
  - ✦ Sequence diagrams (more informative)
  - ✦ collaboration diagrams (more concise)



## Realization of the “create diagram” use case

- ✱ Multiple diagrams
  - ➔ We need a “diagram” class
- ✱ Only one active diagrams
  - ➔ We need a “DiagramEditor” object
- ✱ The “DiagramEditor” should have a link

```
graph LR; DE[:DiagramEditor] --- D1[:Diagram]; DE --- D2[:Diagram]; DE -- current --- D2;
```

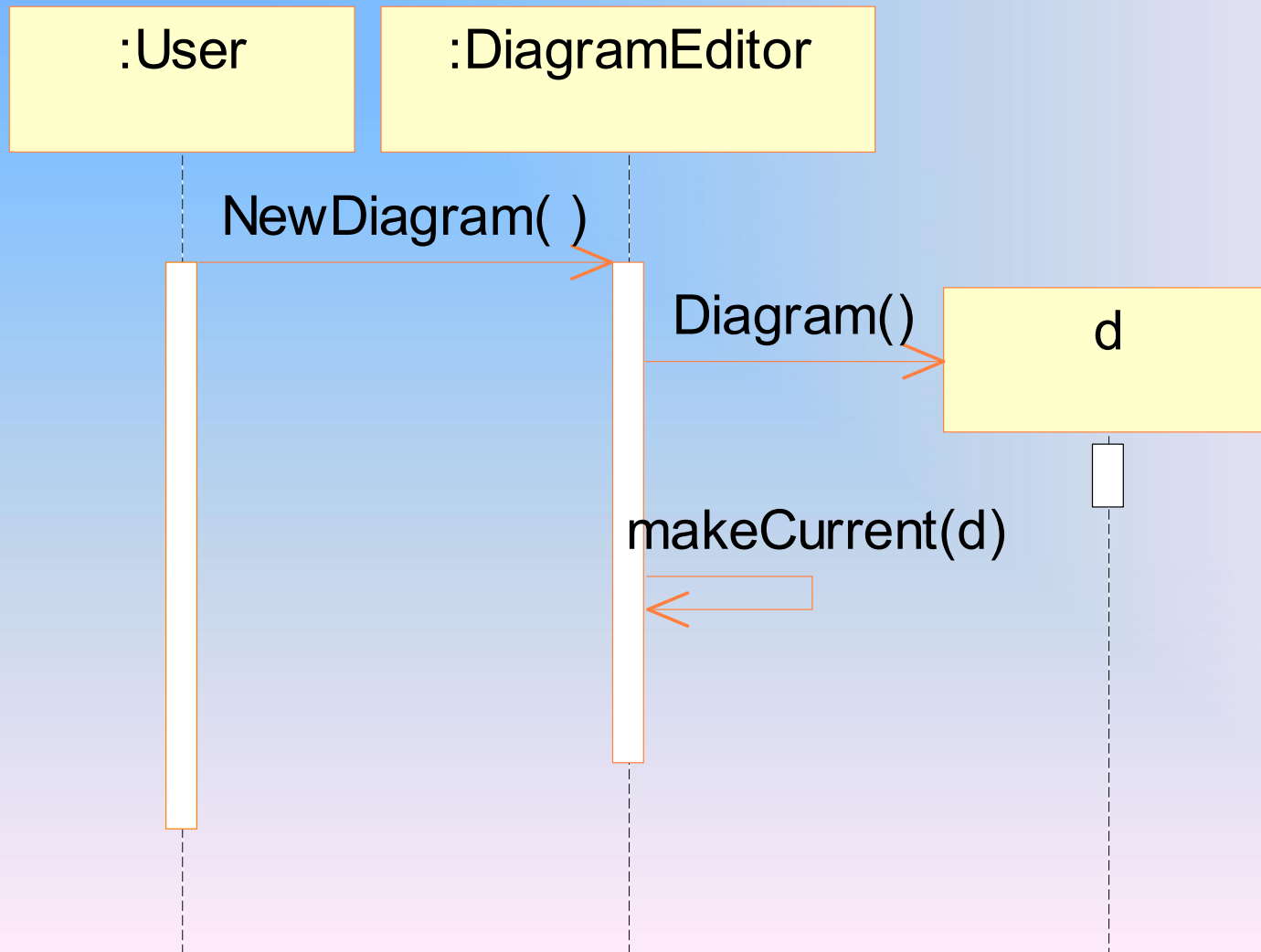
:DiagramEditor

:Diagram

current

:Diagram

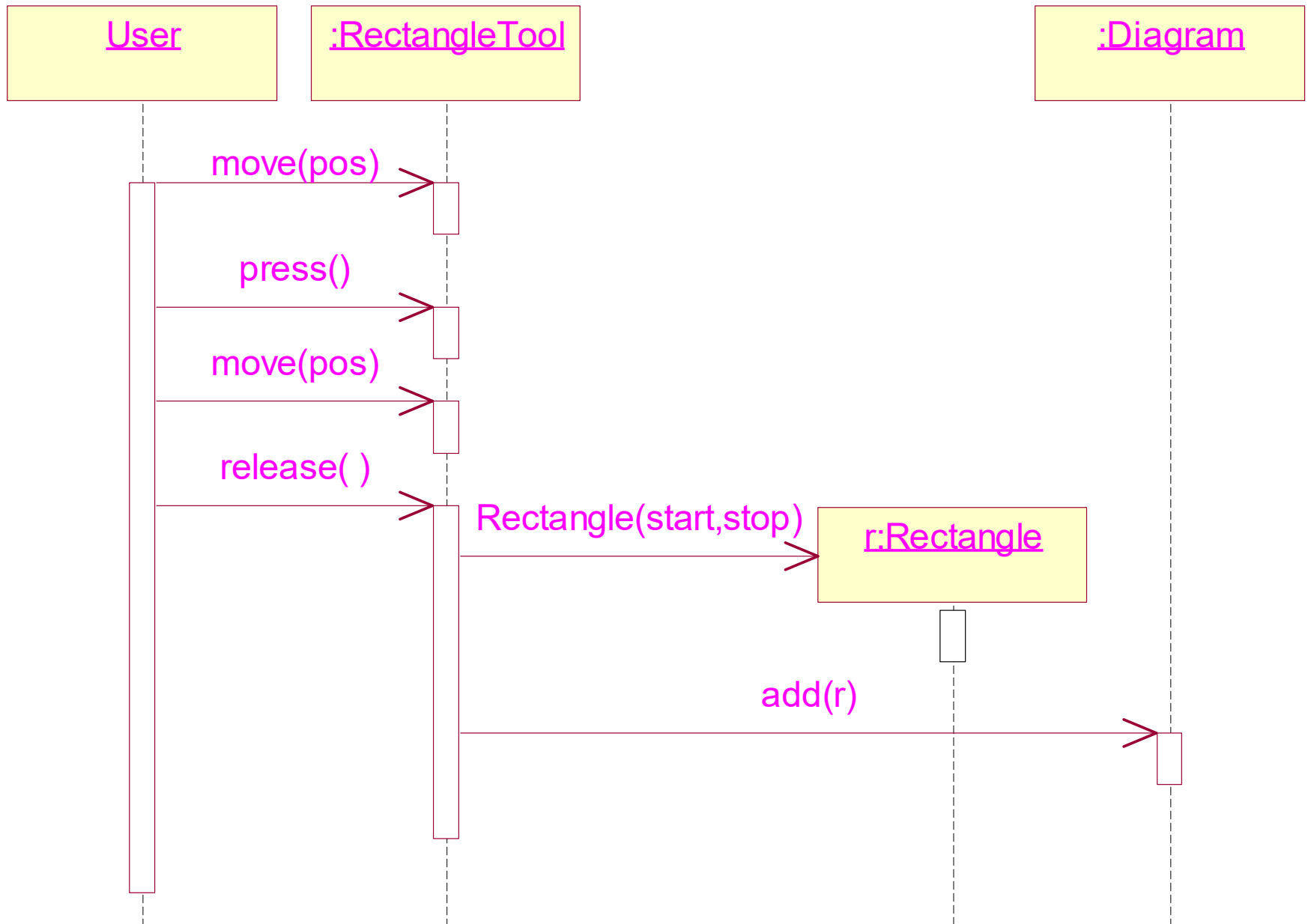
# A sequence diagram



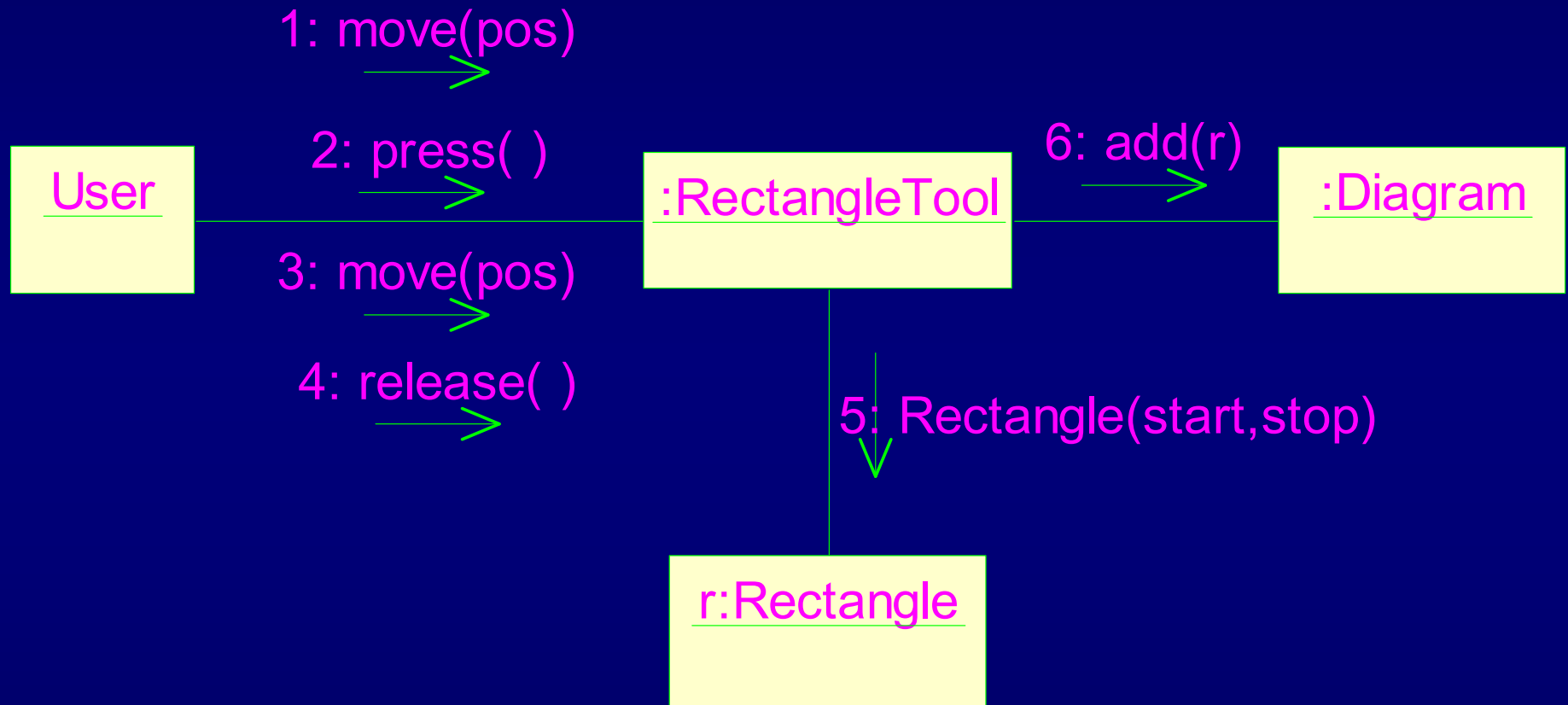
# Elements in a sequence diagram

- ✱ Objects are shown at the top.
- ✱ Time flows downward.
- ✱ Lifeline.
- ✱ Messages & message to oneself
- ✱ Activation: processing a message.
- ✱ Construction of a new object.

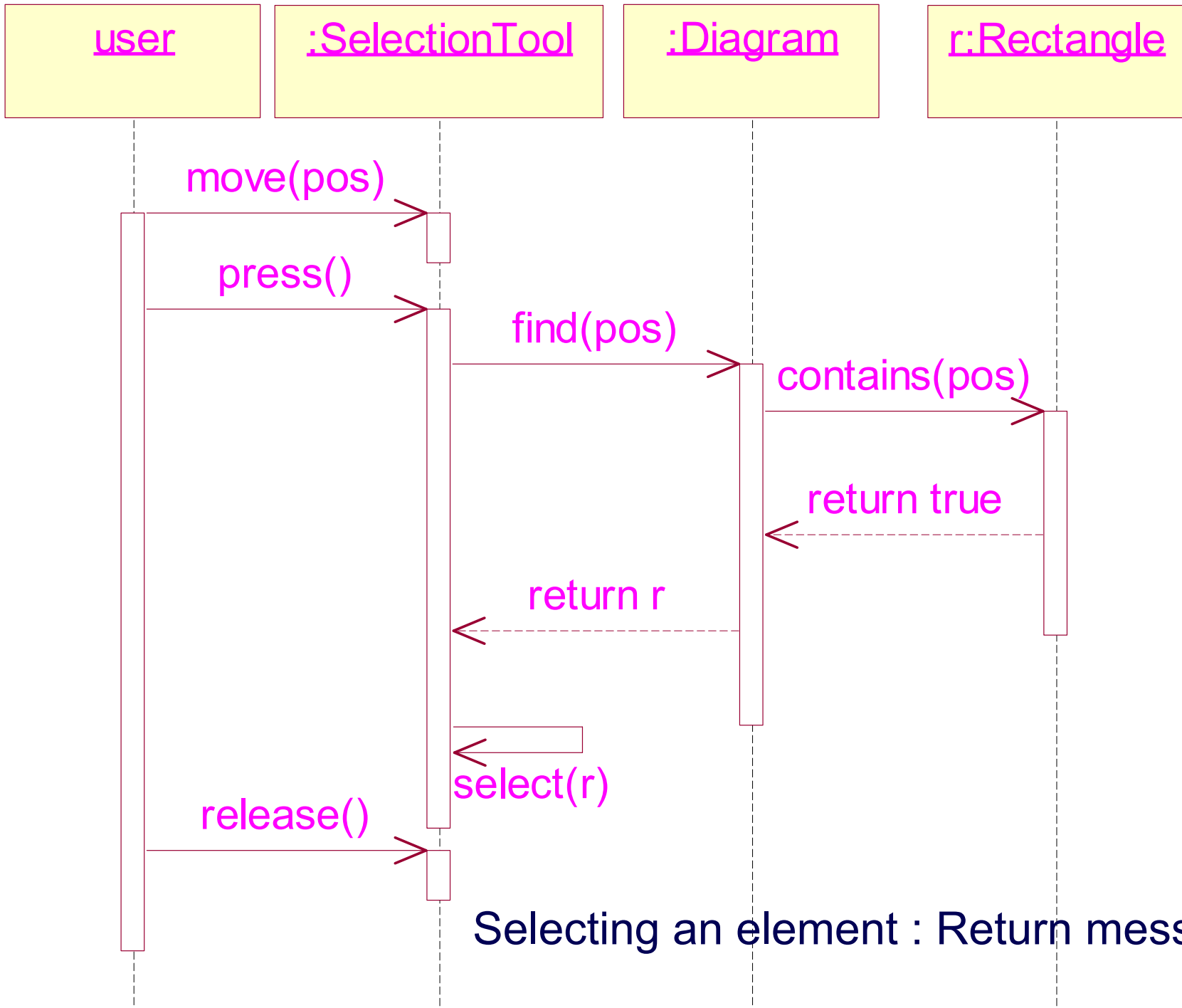
# Creating a rectangle element



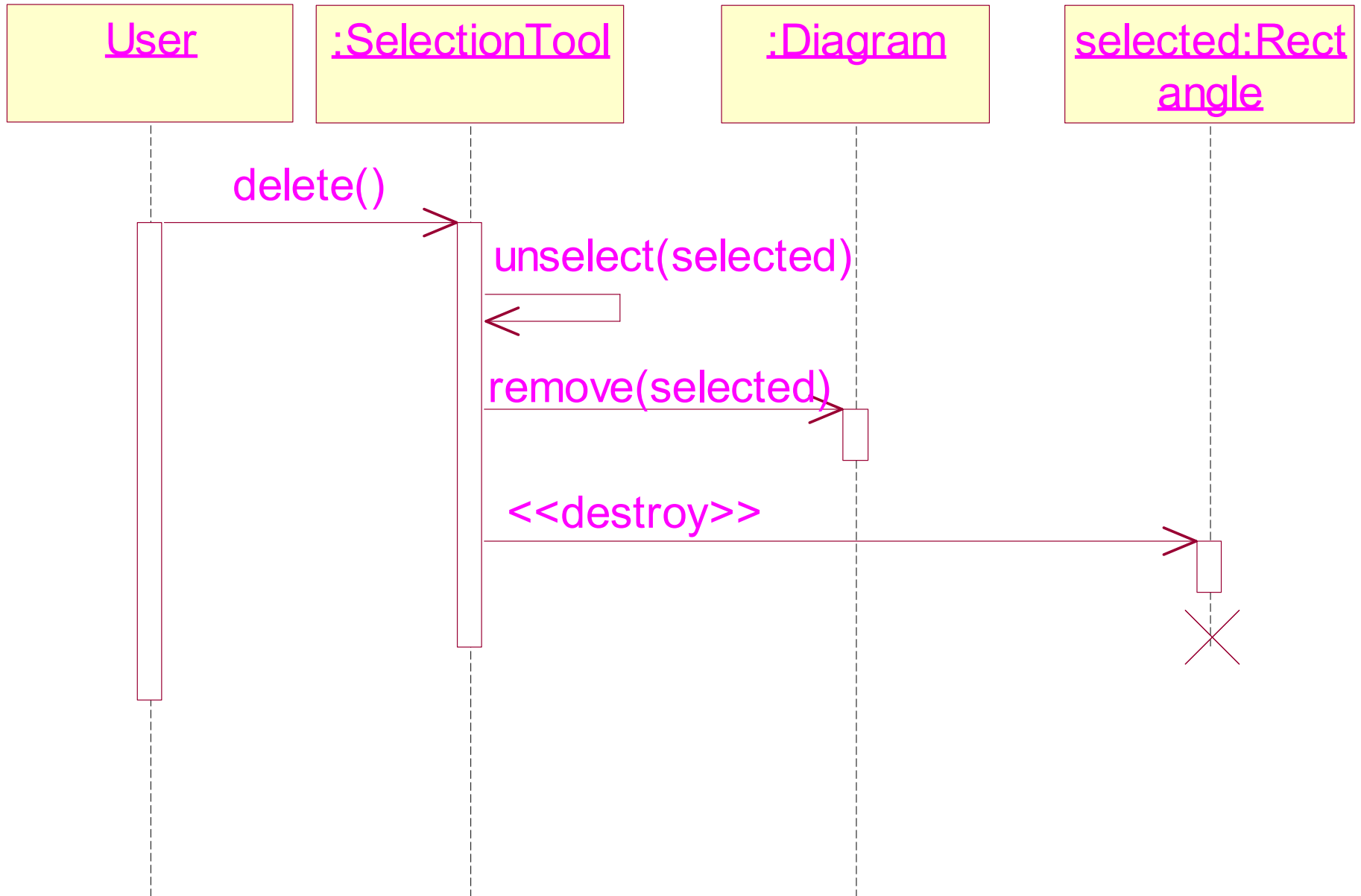
# A collaboration diagram



Order of the messages should be explicitly specified.

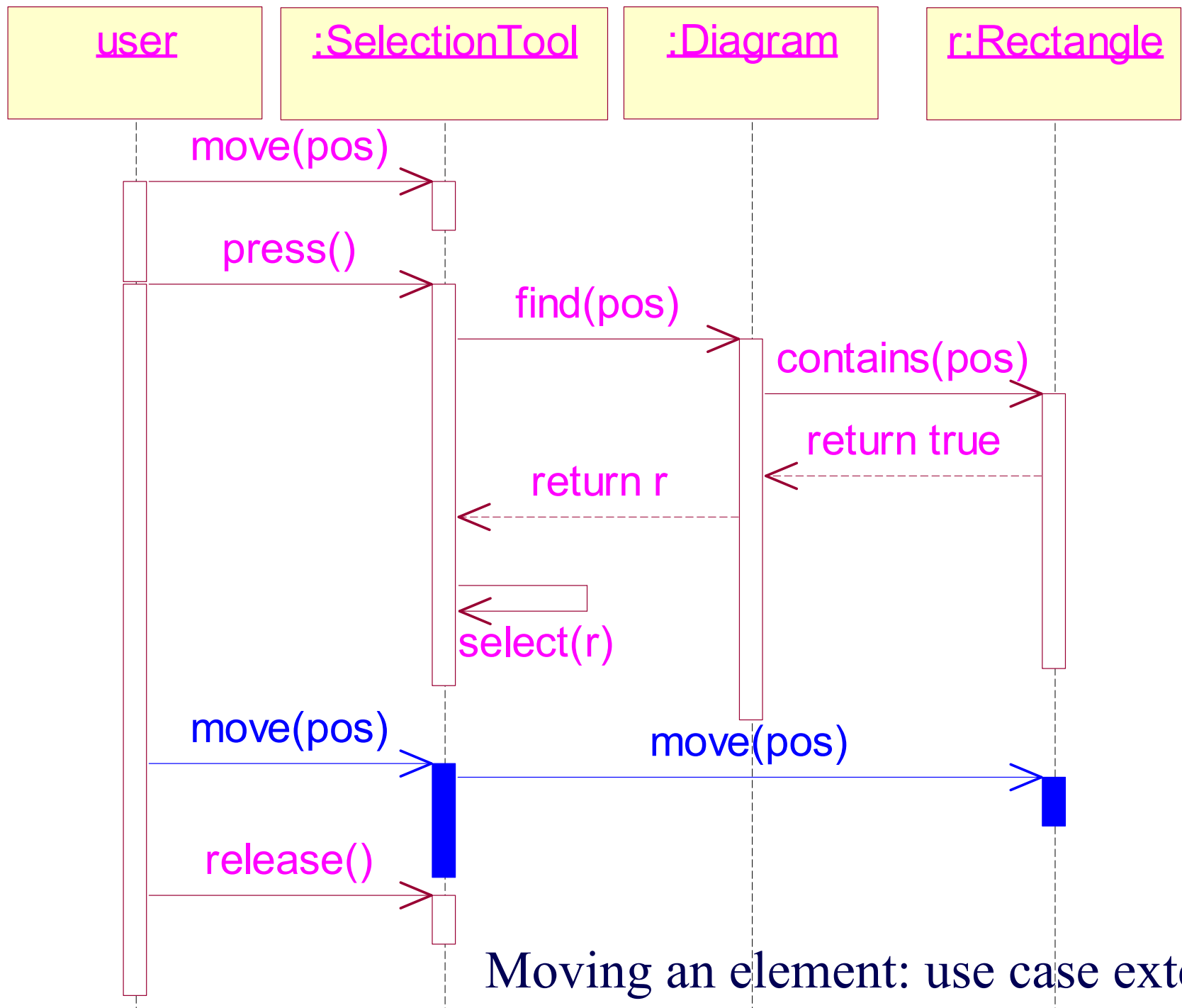


Selecting an element : Return messages



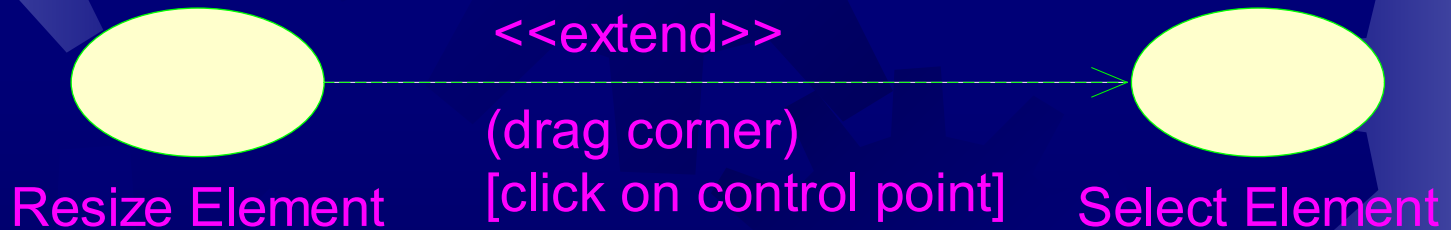
Deleting an element: Termination of objects



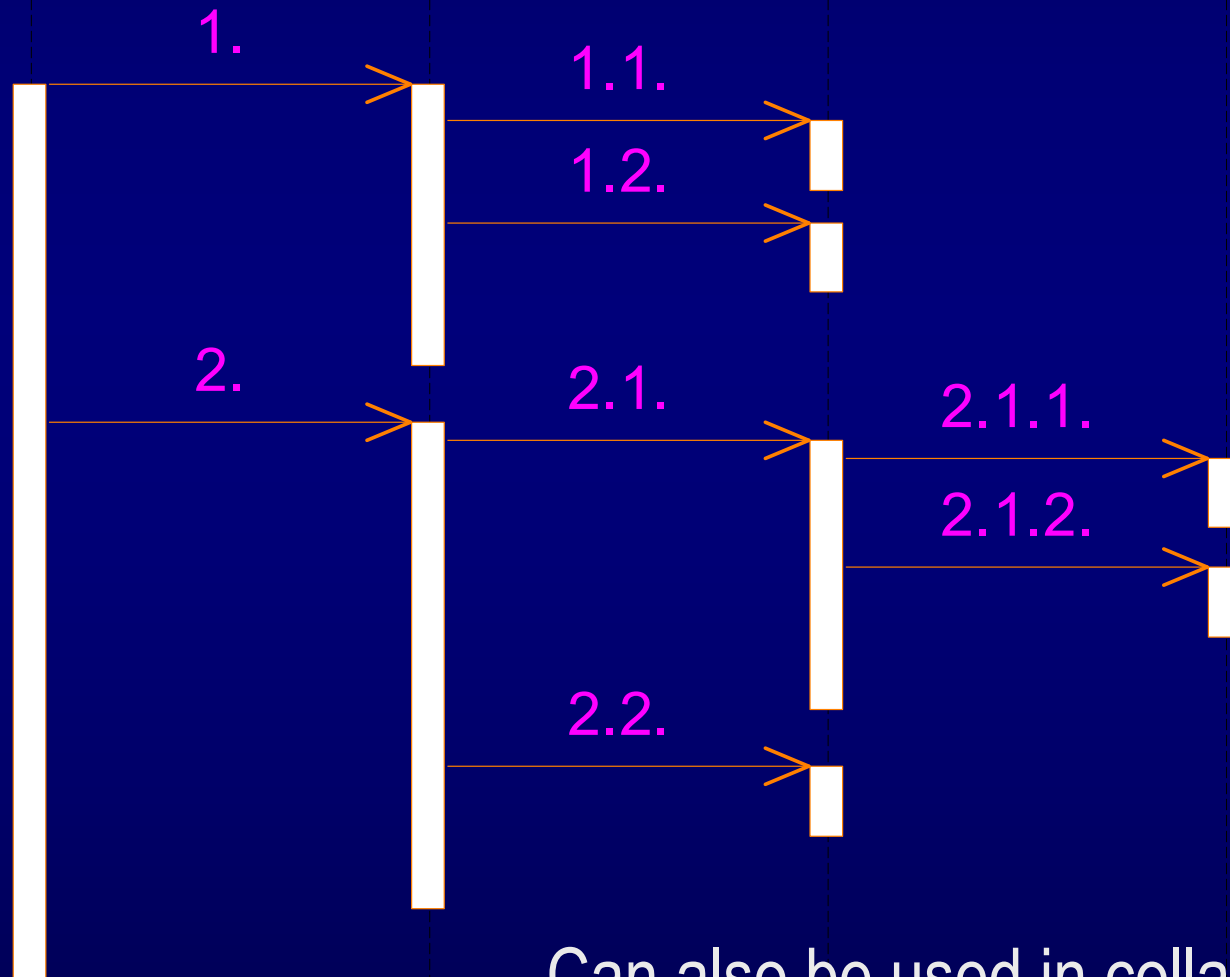


Moving an element: use case extension

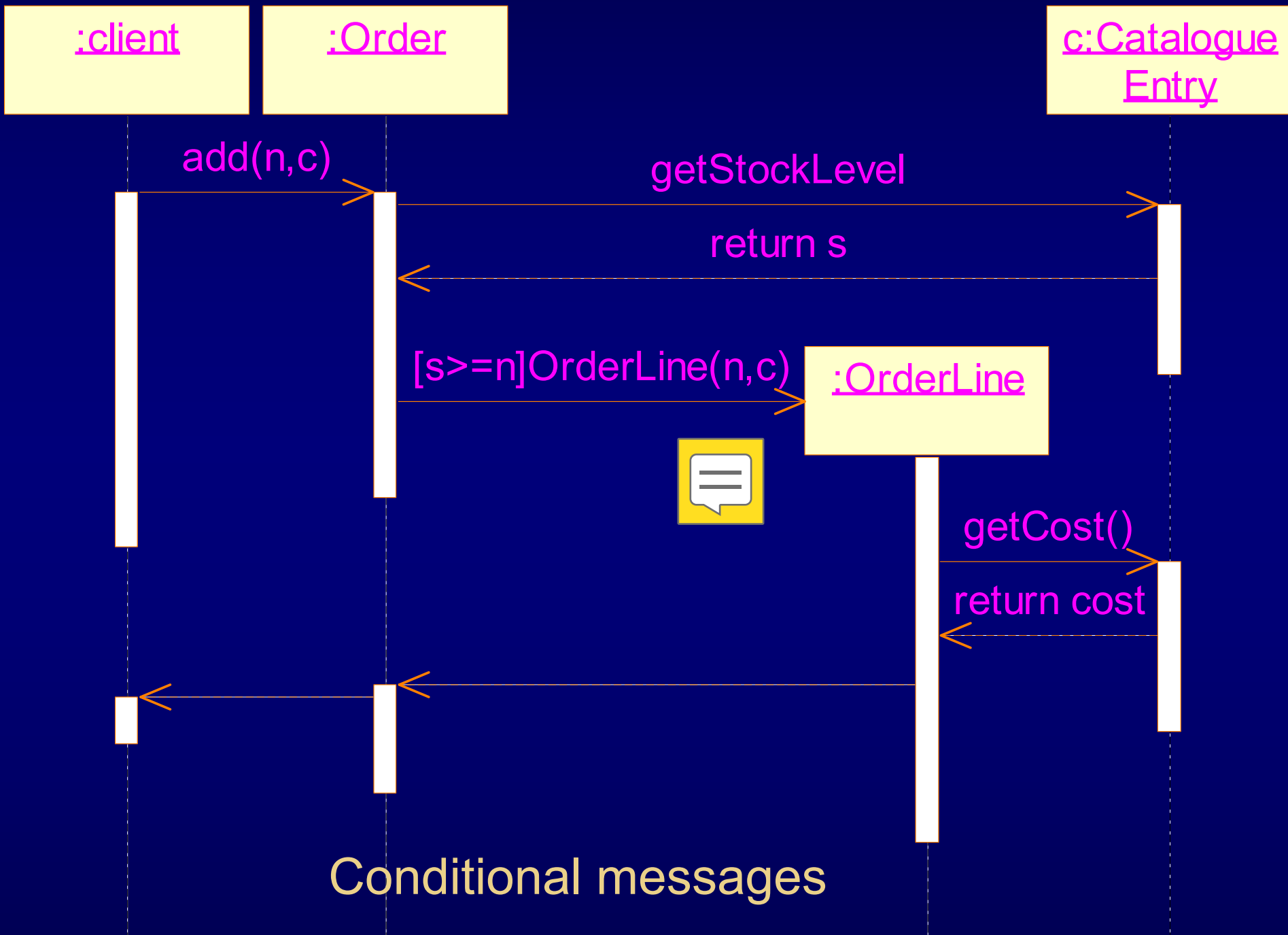
# Extension of use case



# Hierarchical numbering of messages




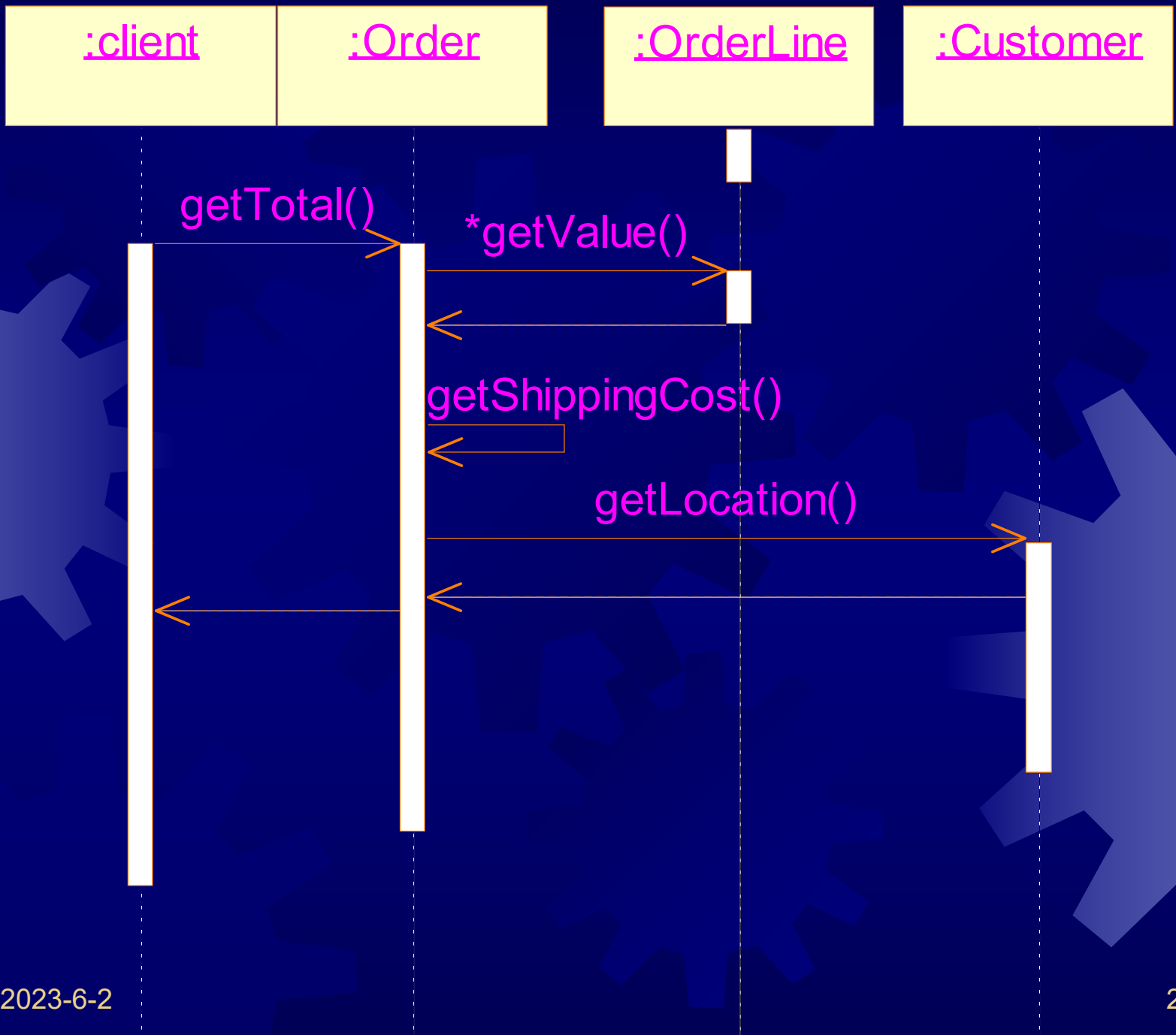
Can also be used in collaboration diagram



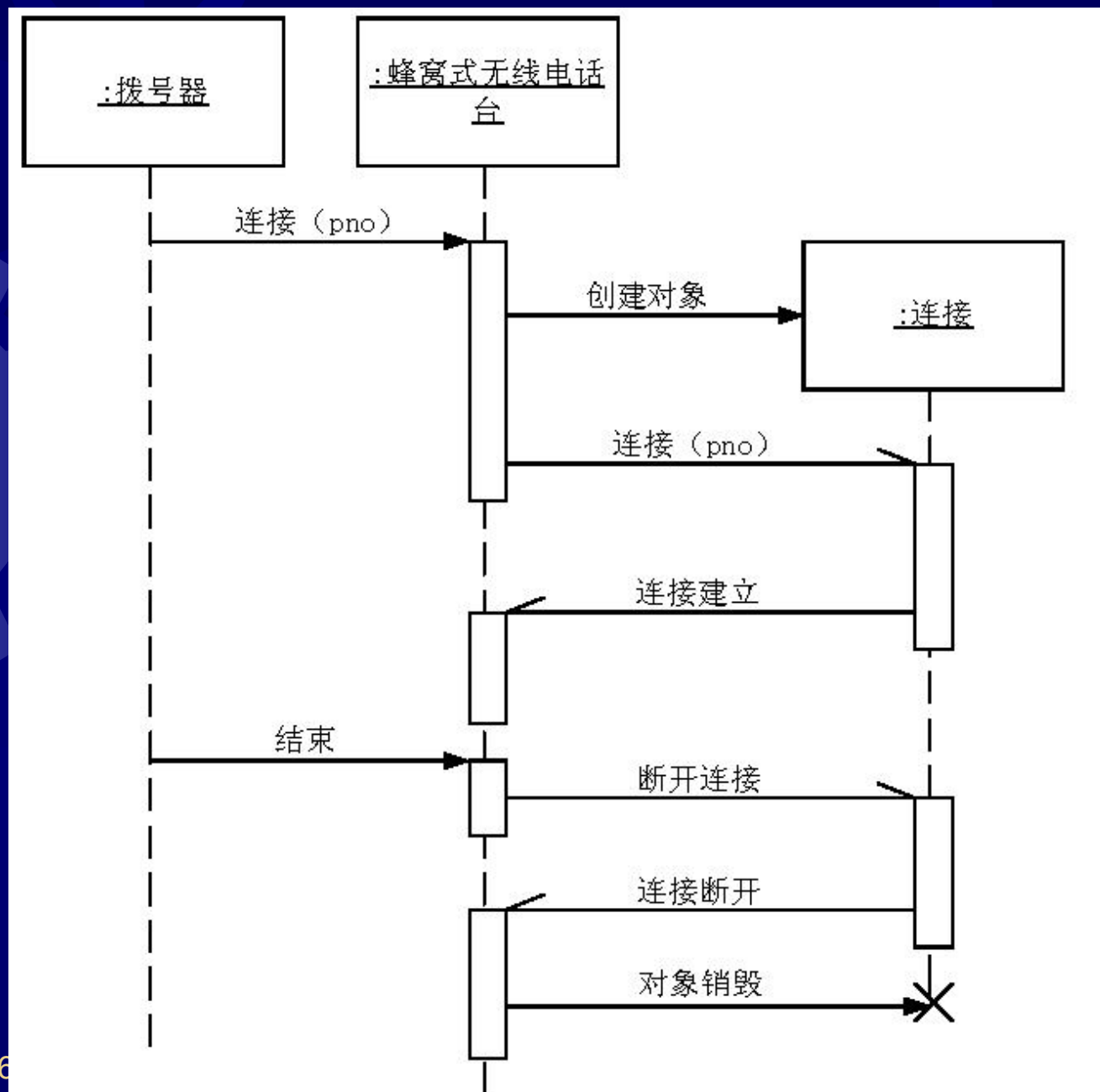
Conditional messages

# Message to oneself

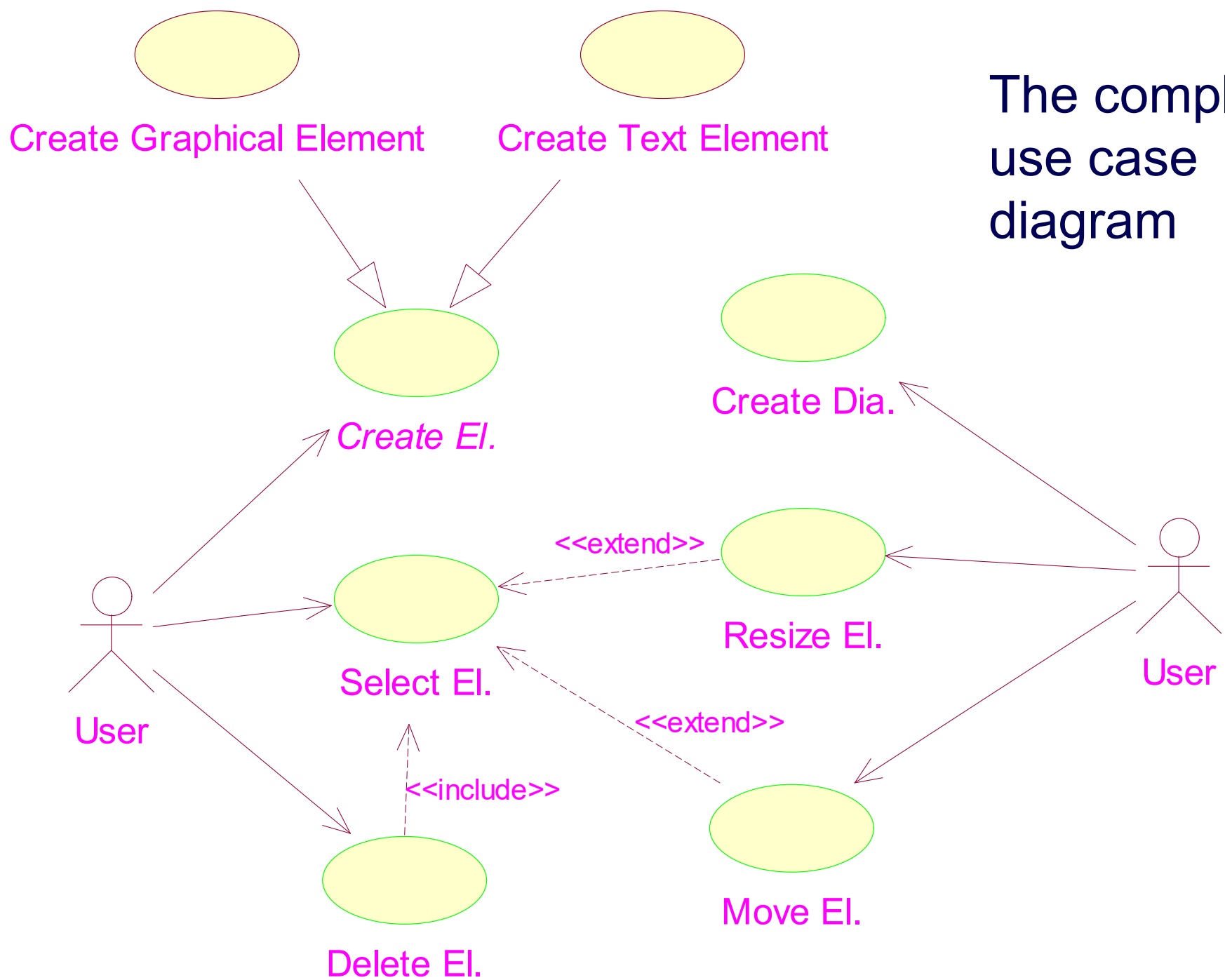
- ✱ It represents implementation details.
- ✱ But when the return value of the message is used in the further messages, it should be made explicit. 
- ✱ It causes **recursive** activity



# Asynchronous message and multiple threads



# The complete use case diagram





# Generalization between actors

