PREVIOUSLY ON ASD…

- **Domain Modelling**
    - represent ideas / concepts from a domain
    - represent the relation between those domain concepts
- **Structured Modelling**
    - Representation of domain in software
    - Relationships: Composition, Aggregation, Association and Dependency
    - It's all about responsibility

ADVANCED SOFTWARE DESIGN
LECTURE 4
MODELLING BEHAVIOUR

Kiko Fernandez-Reyes

BEHAVIOURAL MODELLING

What is it?

Why is it important?

Which behaviour is modelled?

WHAT IS BEHAVIOUR?

BEHAVIOUR IS INTERACTION BETWEEN OBJECTS.

UML DIAGRAMS

INTERACTION DIAGRAMS

Visualise the **interactive behaviour** of the system.

Describe the **message flow** in the system.

Describe **structural organisation** of the objects.

Describe **interaction** among objects.

Different types of models capture different aspects of the interaction.

KINDS OF DIAGRAMS

Sequence Diagrams

Communication Diagrams

State Machine Diagrams

Interaction Overview Diagrams

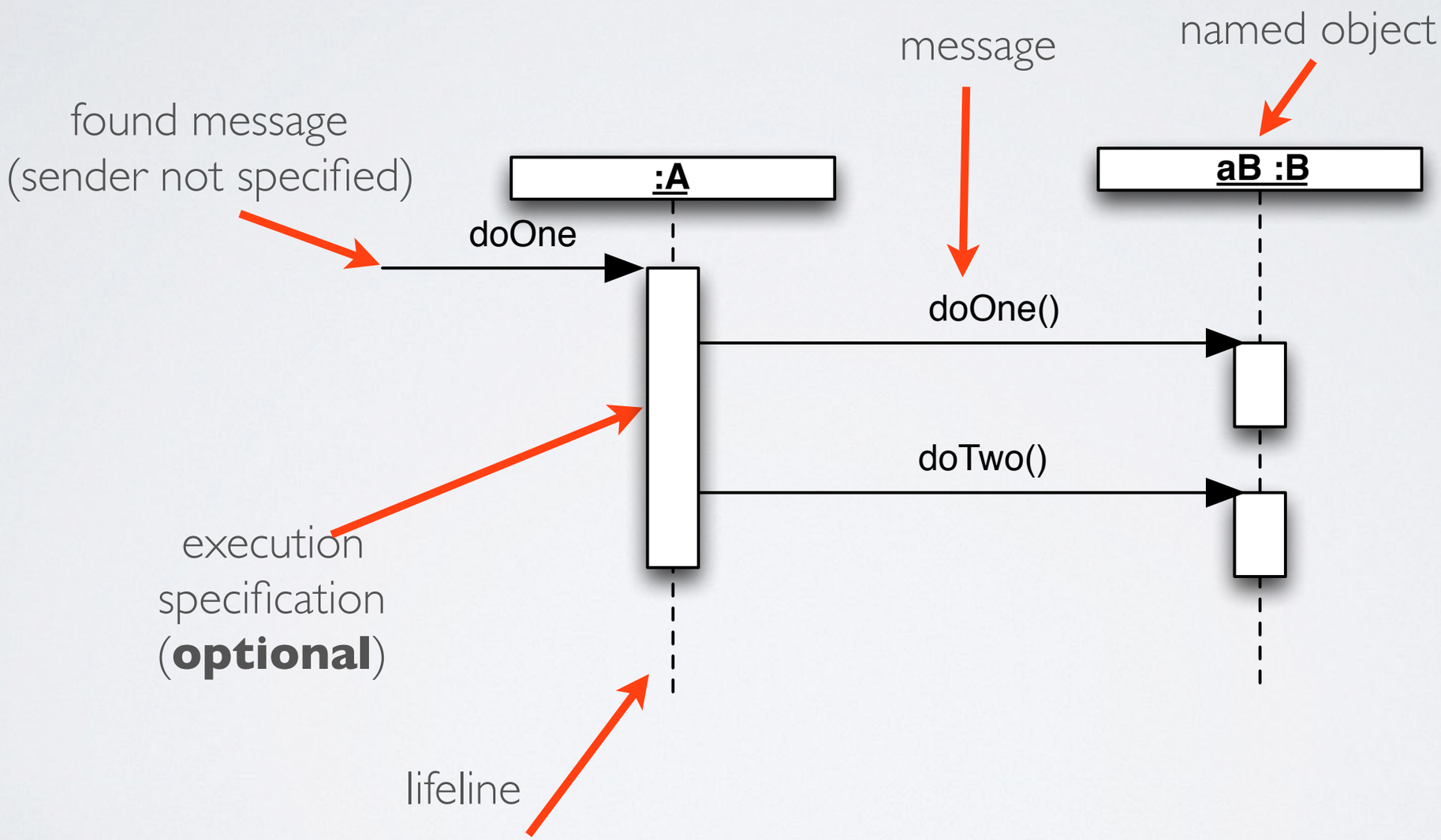
SEQUENCE DIAGRAMS

SEQUENCE DIAGRAM

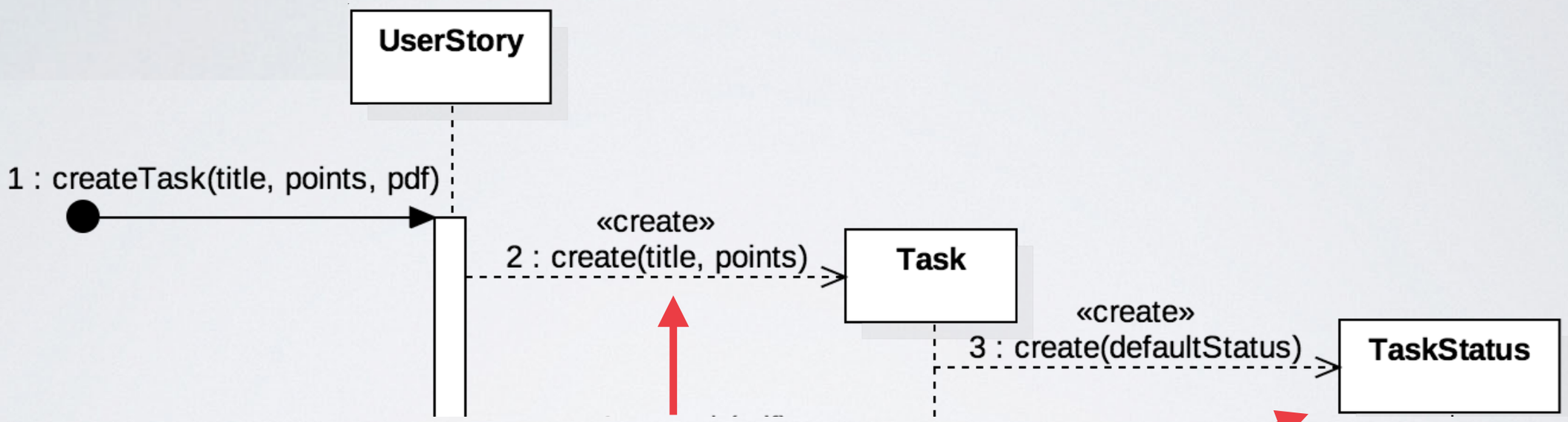
Defines sequence of events, in particular, order in which messages occur

A natural refinement of Use Case diagrams

KEY INGREDIENTS

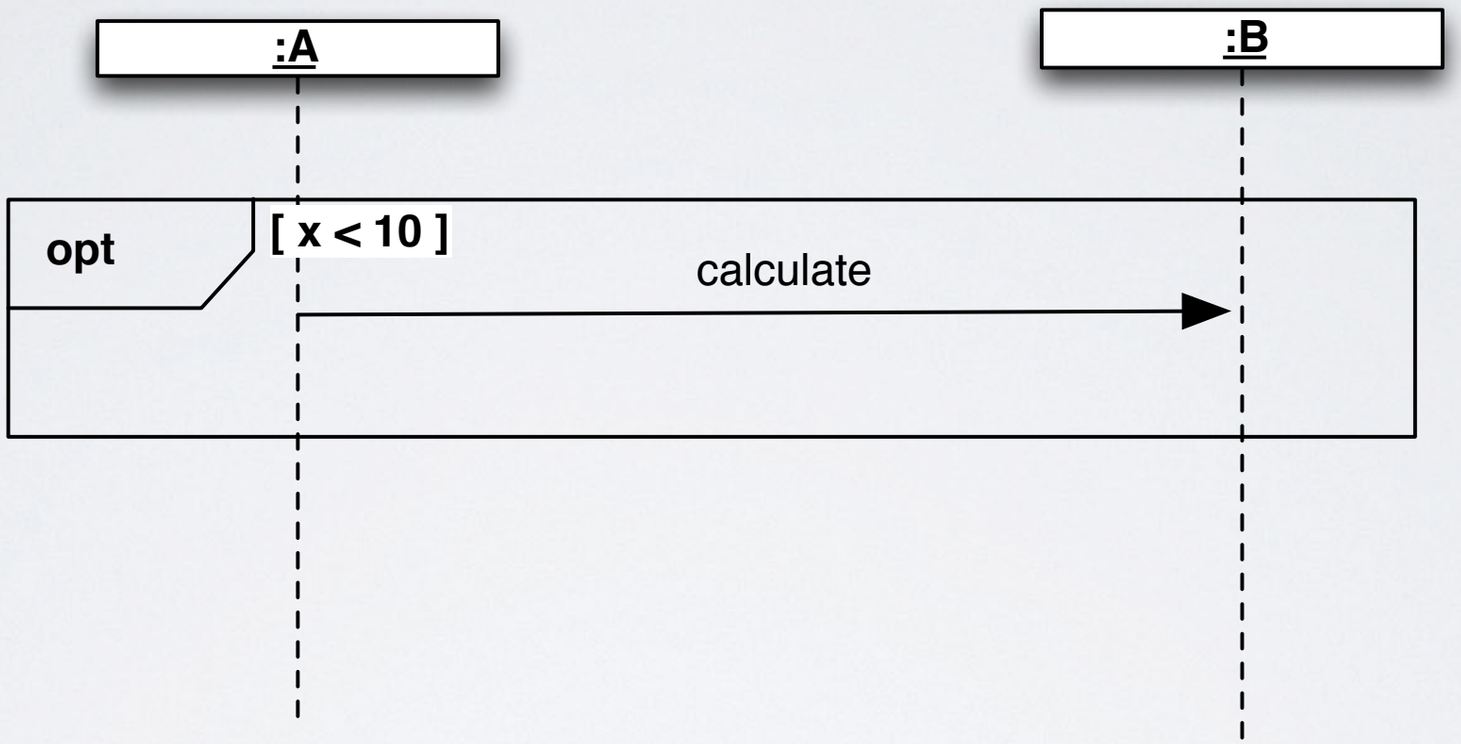


KEY INGREDIENTS

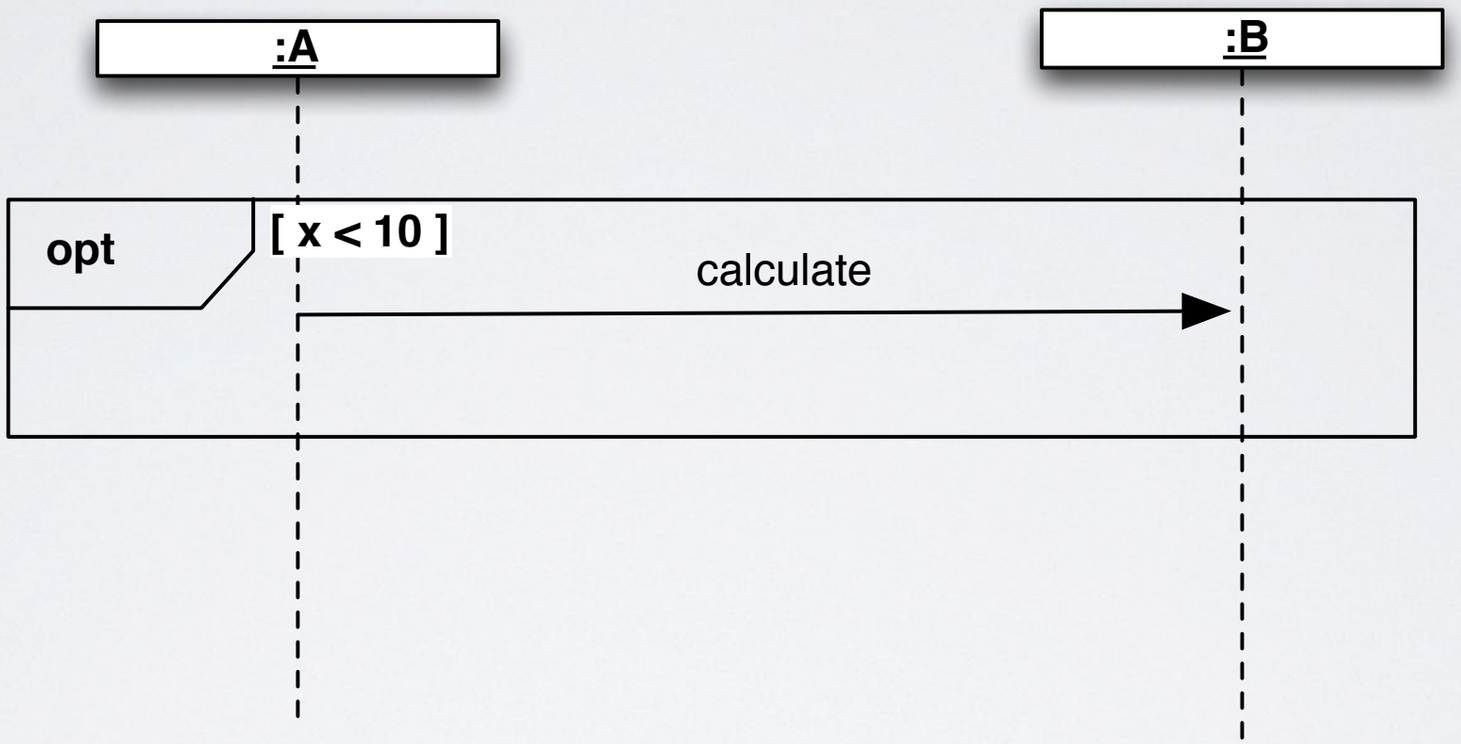


creation
of an object

FRAMES: OPTIONS

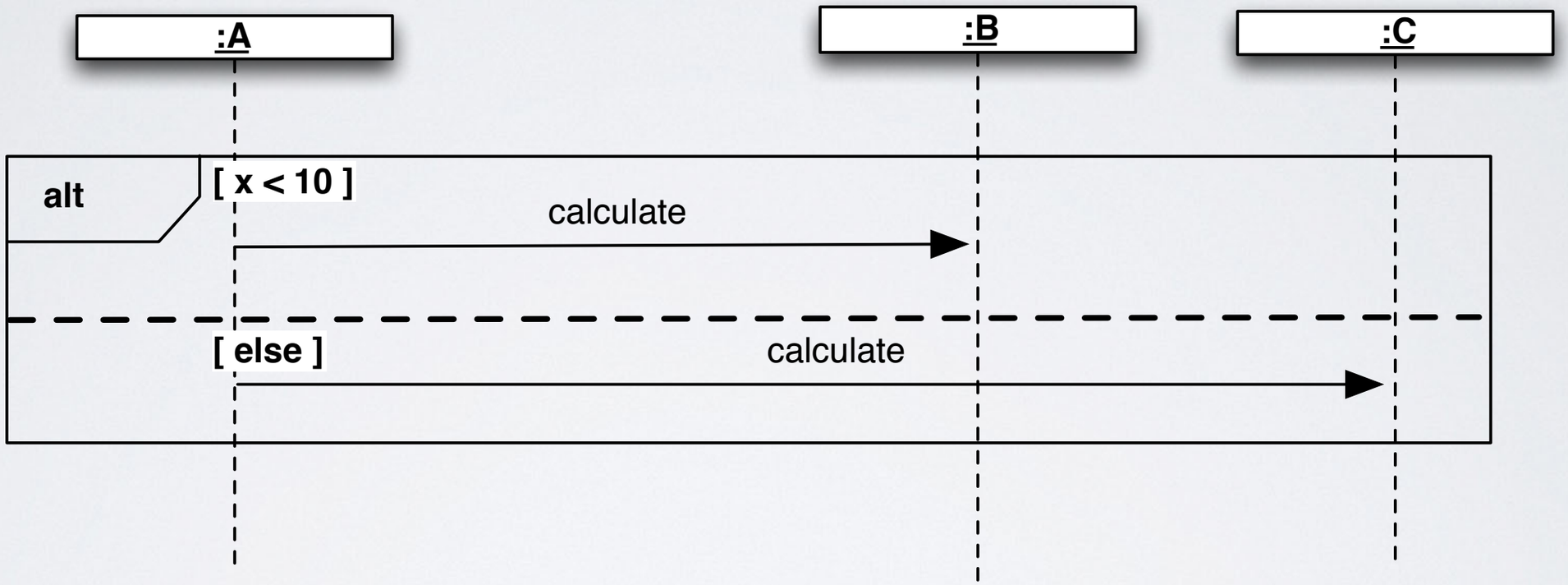


FRAMES: OPTIONS

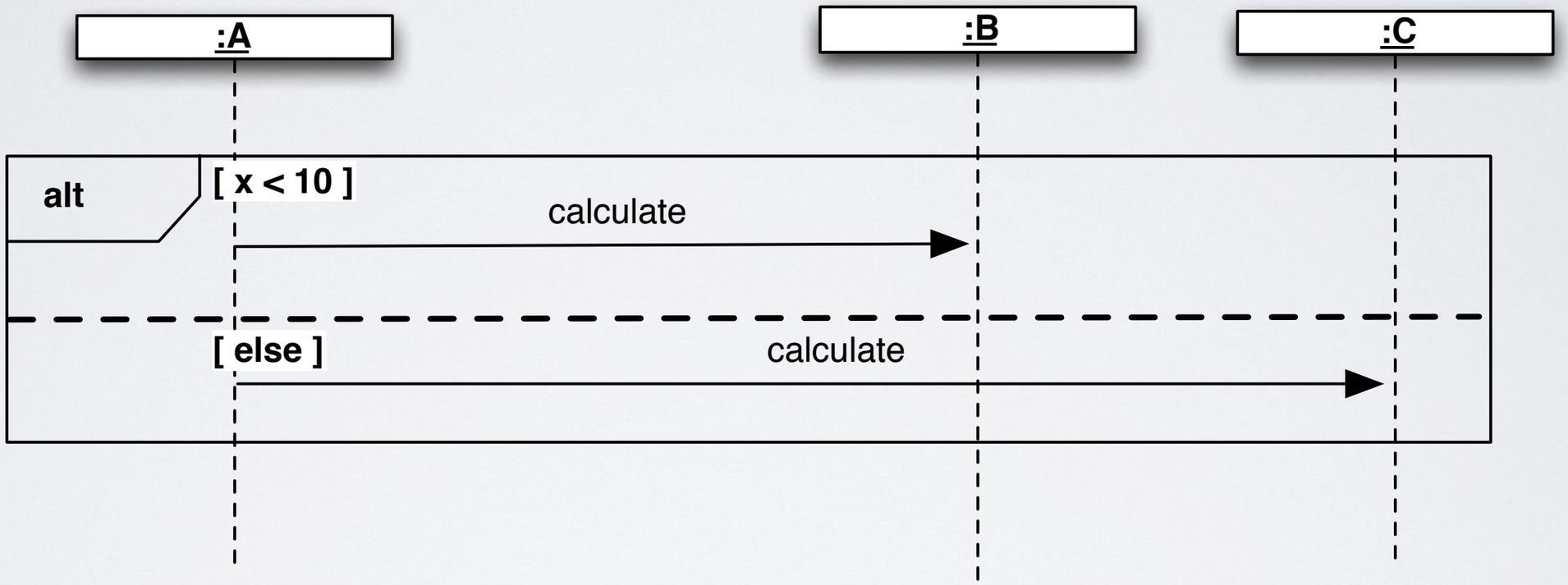


essentially an if statement

FRAMES: ALTERNATIVES

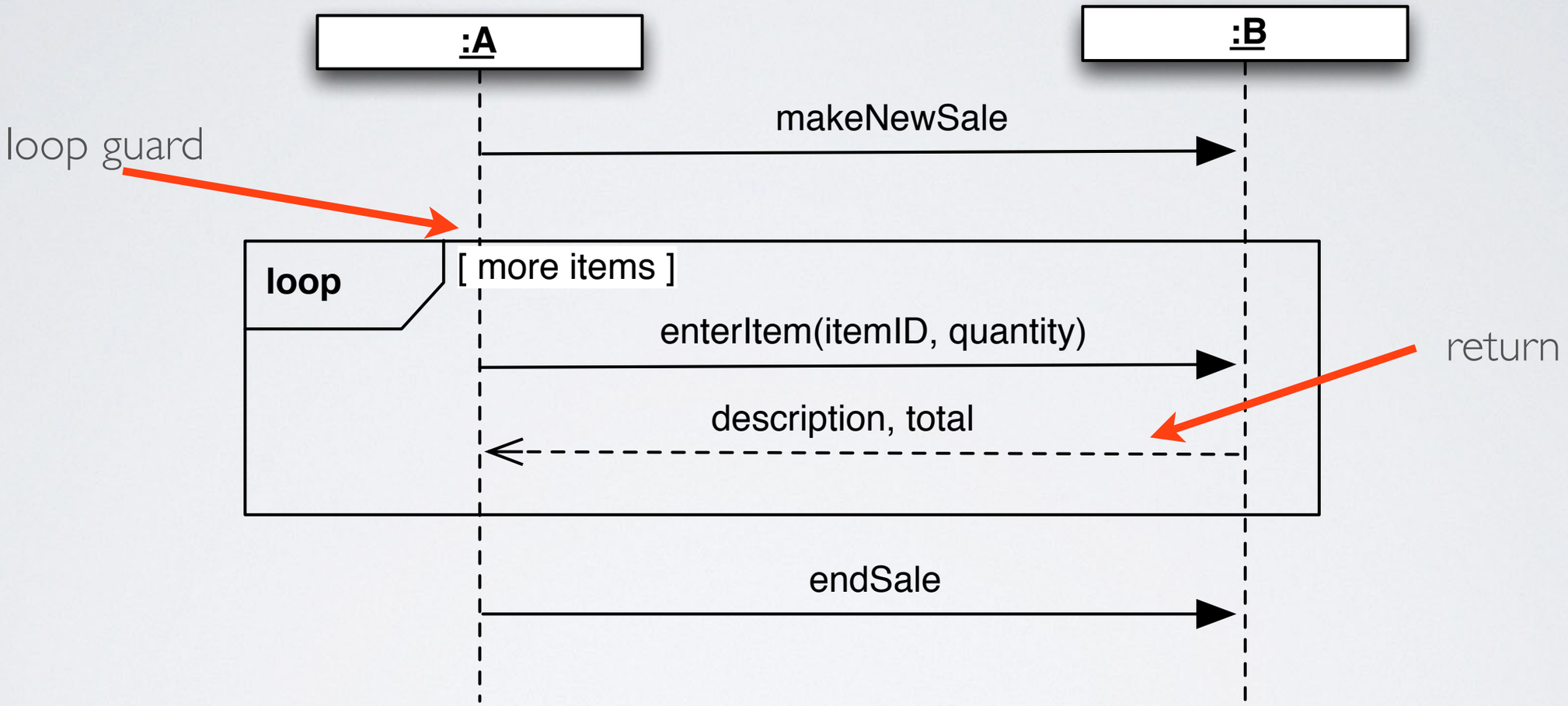


FRAMES: ALTERNATIVES



if-then-else

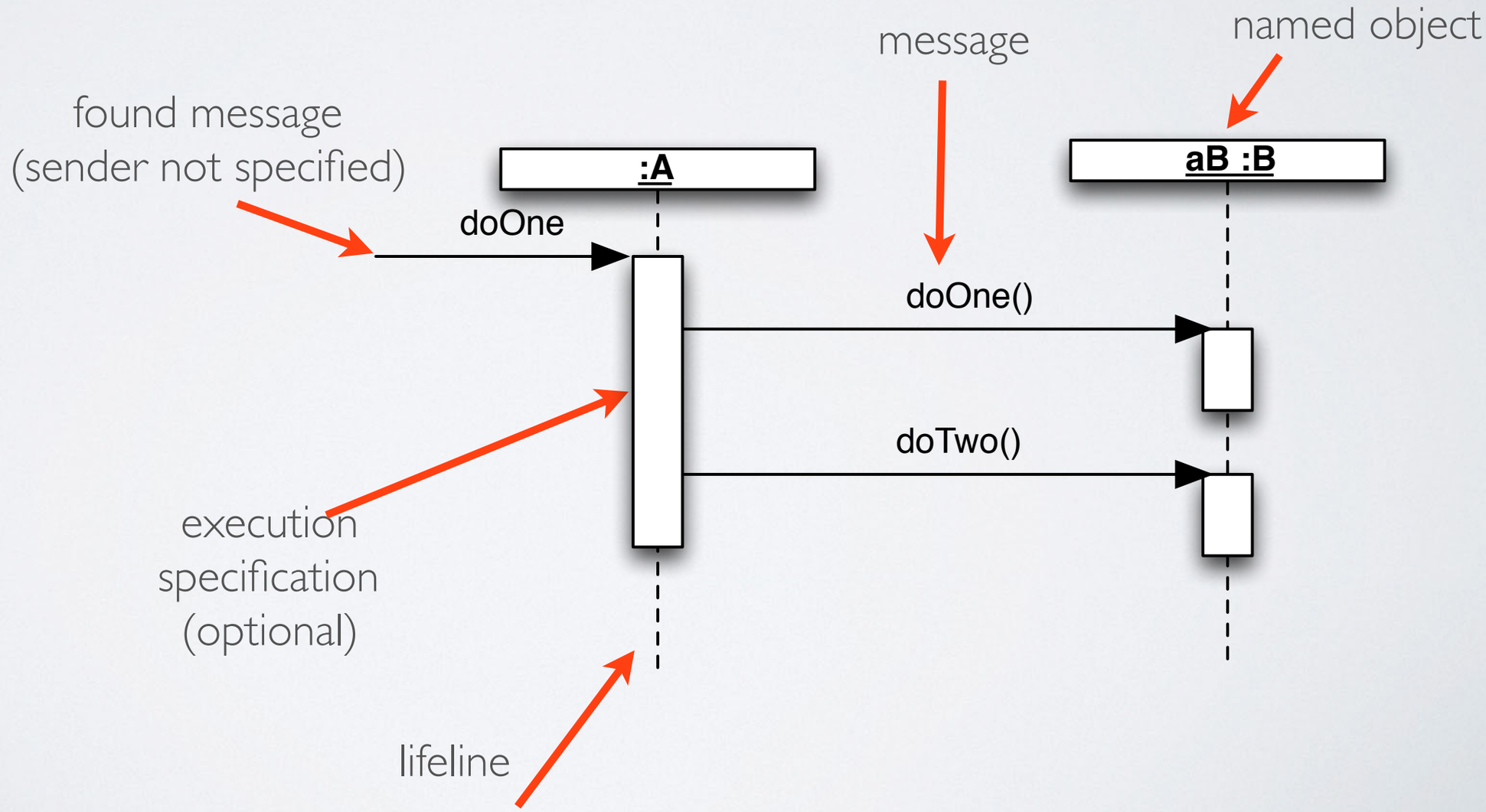
FRAMES: LOOPS



frames can be nested

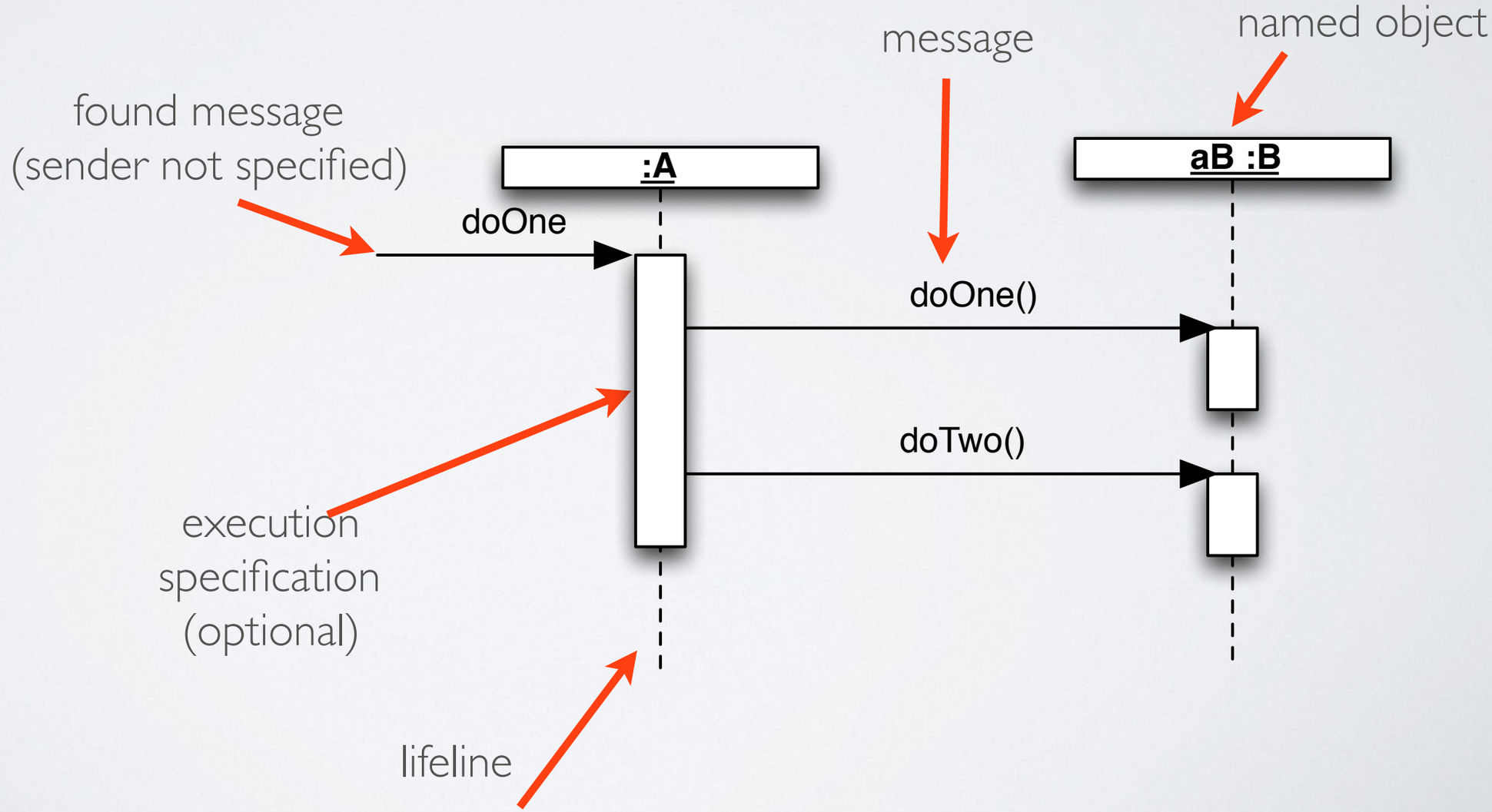
LET'S DRAW A FEW EXAMPLES

Take 5 min. to come up with the code that this sequence represents

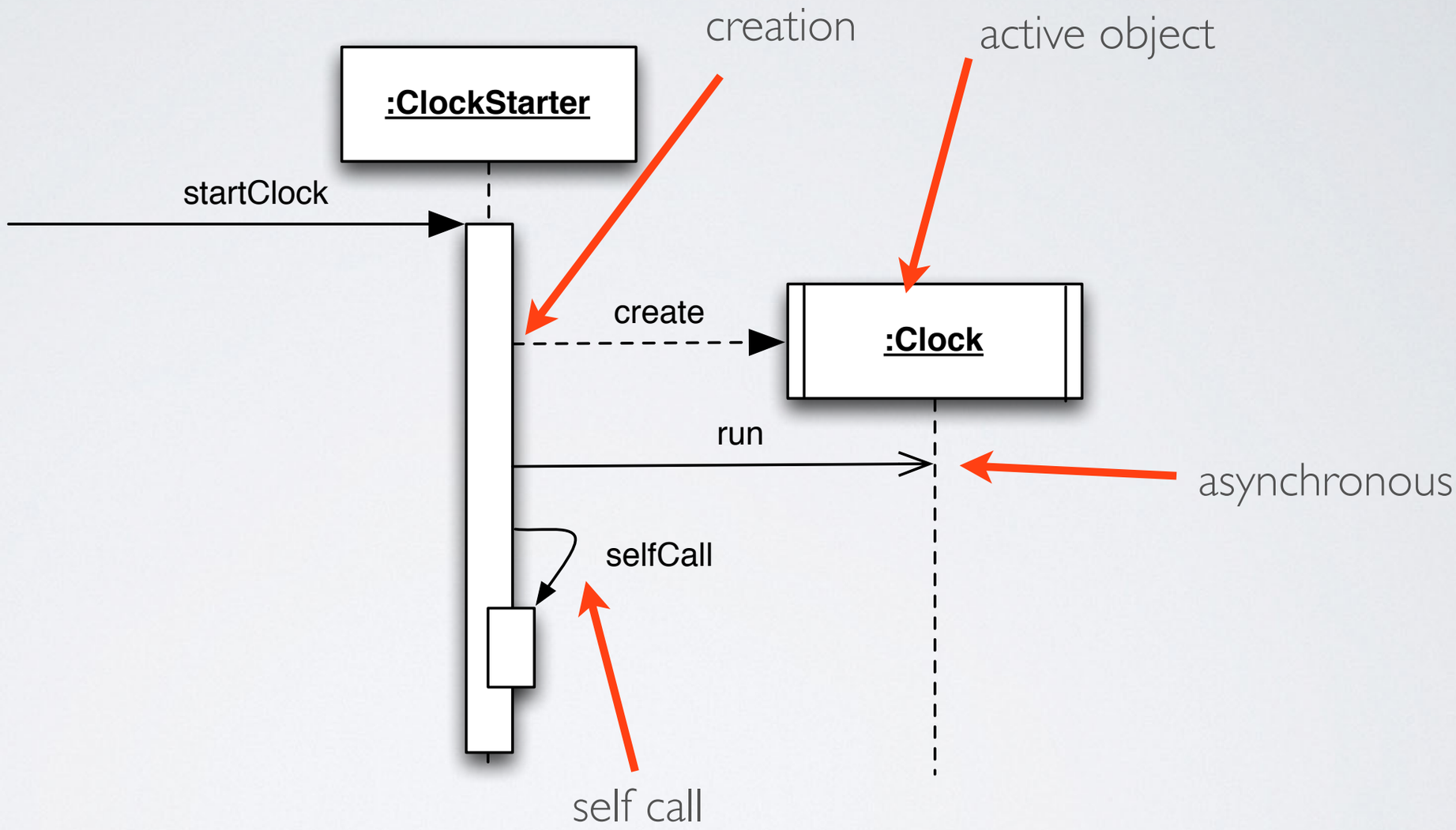


```
class A {
    private B aB;
    …
    public void doOne(){
      this.aB.doOne();
      this.aB.doTwo();
    }
}
```

Take 5 min. to come up with the code that this sequence represents



SOME OTHER FEATURES



GUIDELINES

GUIDELINES

- Novice modellers do not pay enough attention to interaction diagrams
- **Do dynamic object modelling with interaction diagrams**, not just static modelling with class diagrams.
- Without modelling behaviour, you cannot really understand the system.

GUIDELINES

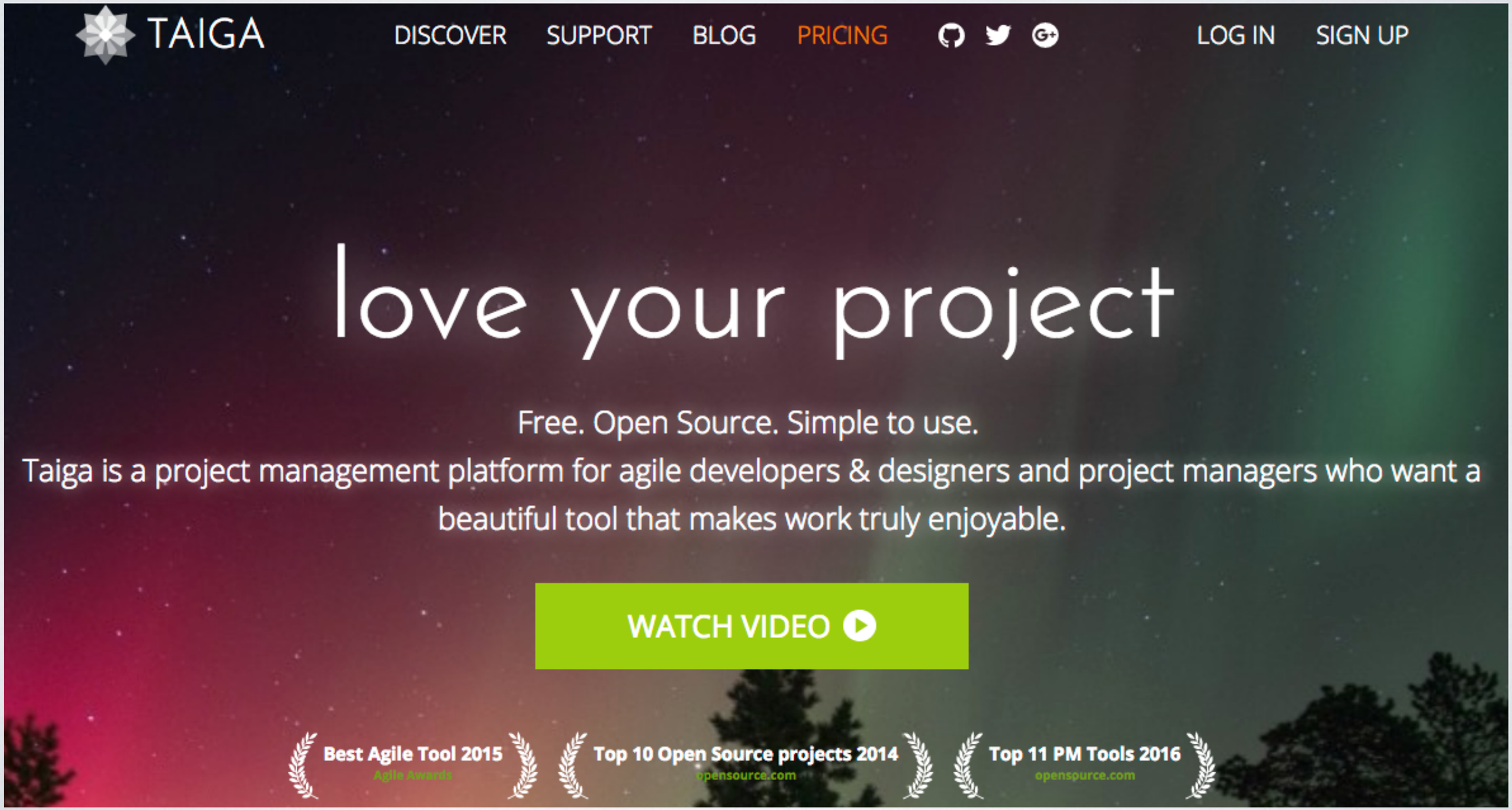
Walkthrough dynamic models against use cases.

Adjust static models whenever dynamic models expose weaknesses.

Ensure reuse among different dynamic models touching the same class – e.g., reusing a method for different scenarios

IN-CLASS EXERCISE

Group work



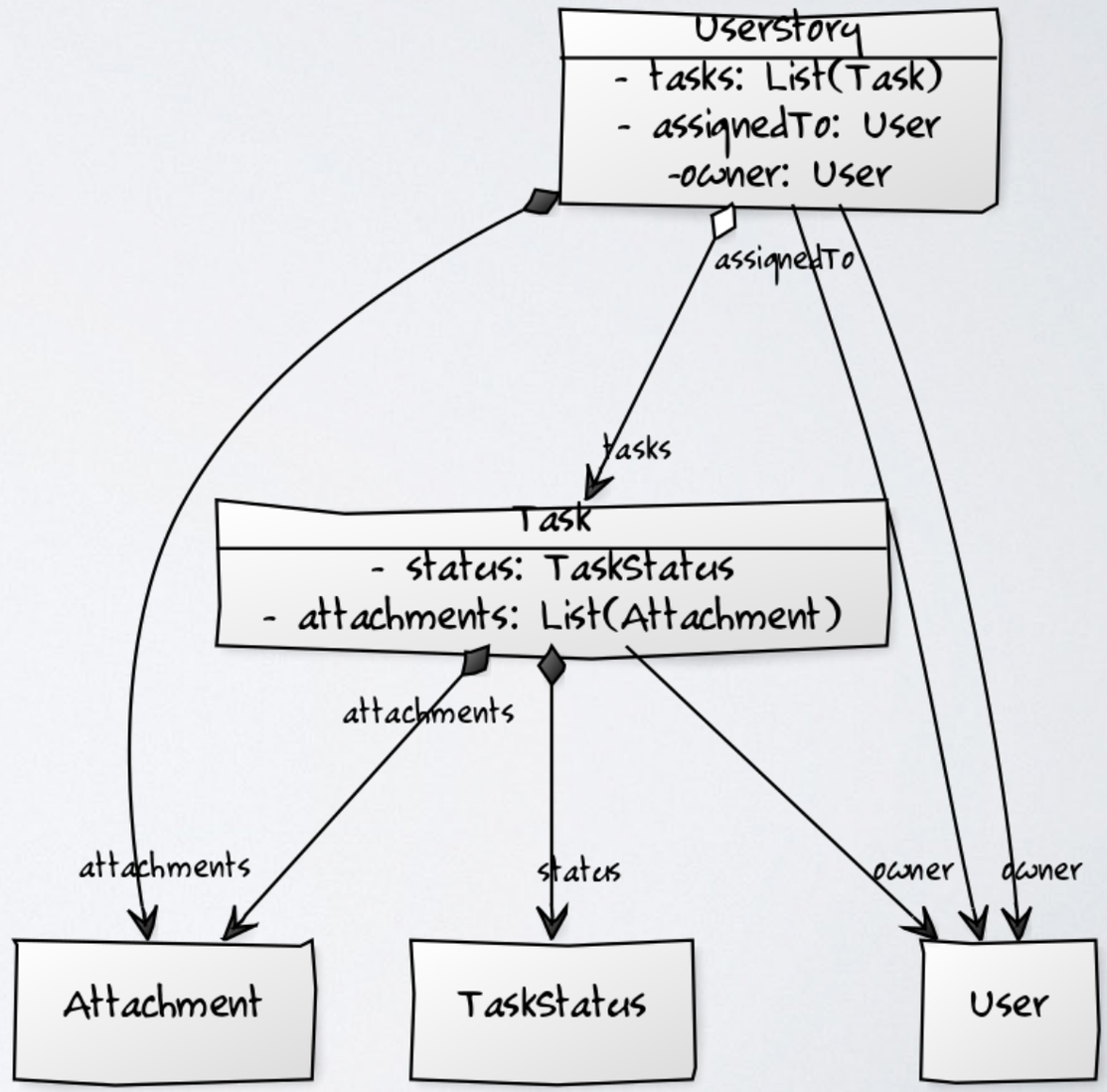
IN-CLASS EXERCISE

Draw sequence Diagram (SD) for creating a task that contains:

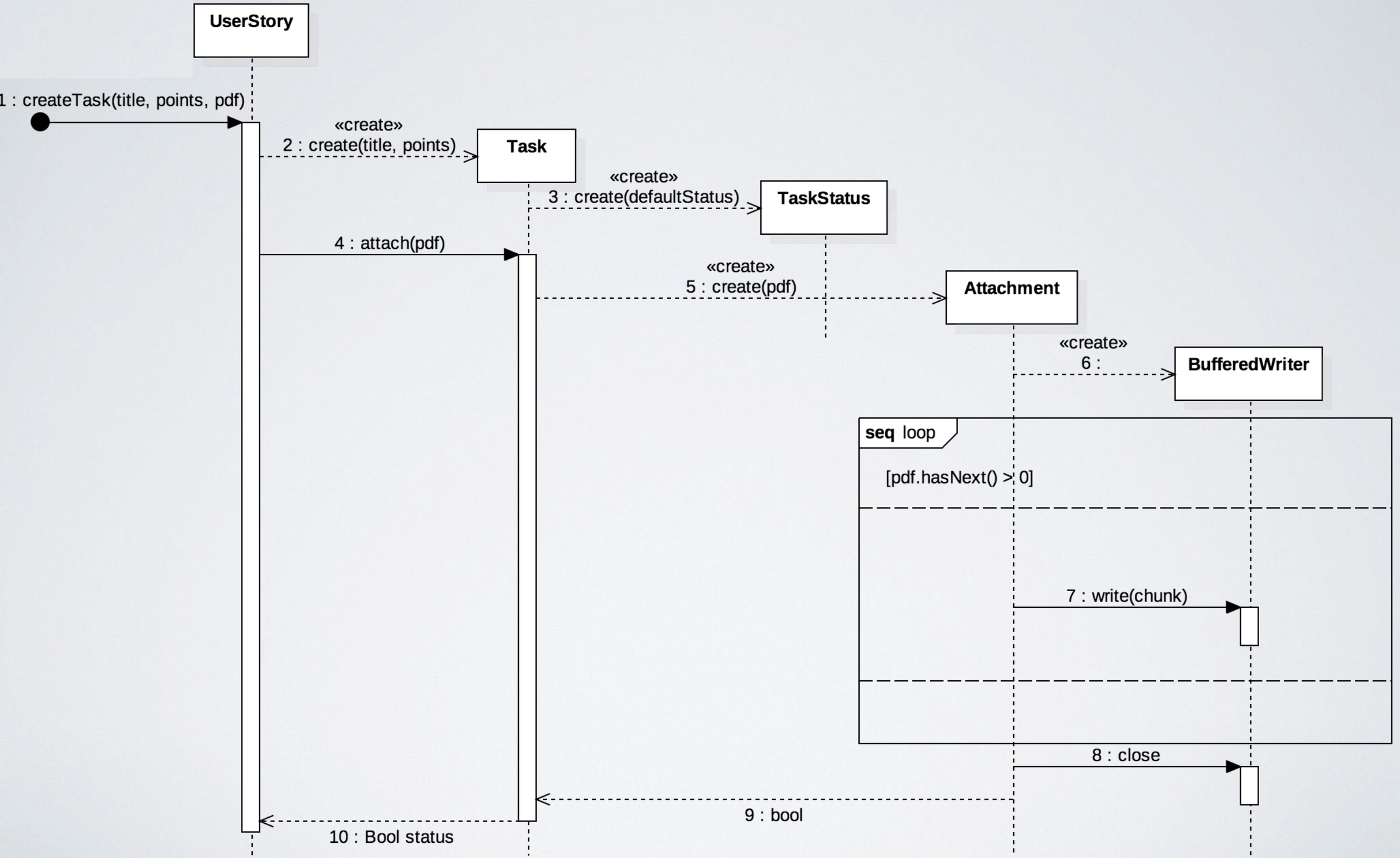
- title
- points
- pdf attachment

Hint:

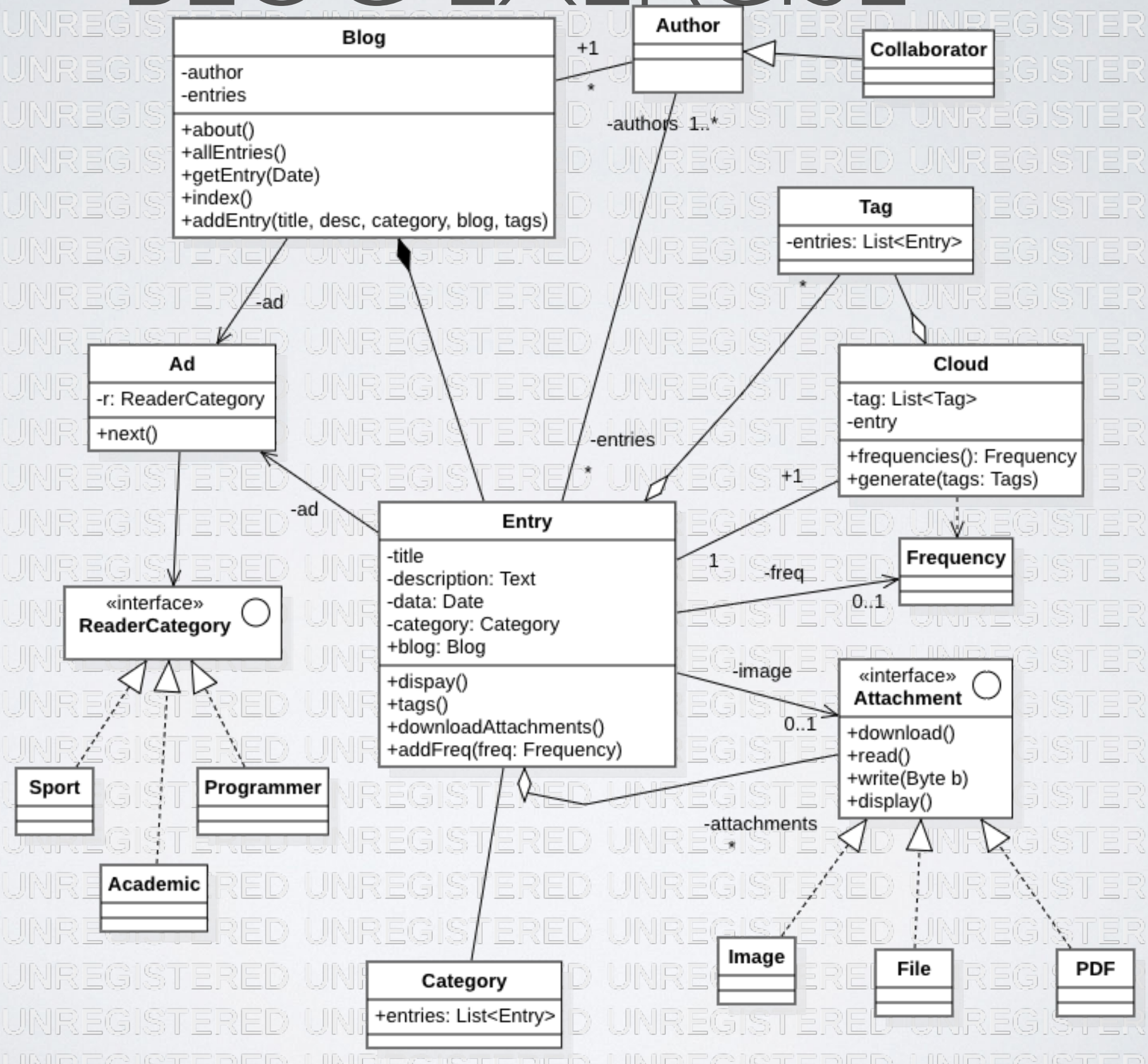
Create new classes if necessary
How would you upload attachment?



POSSIBLE SOLUTION

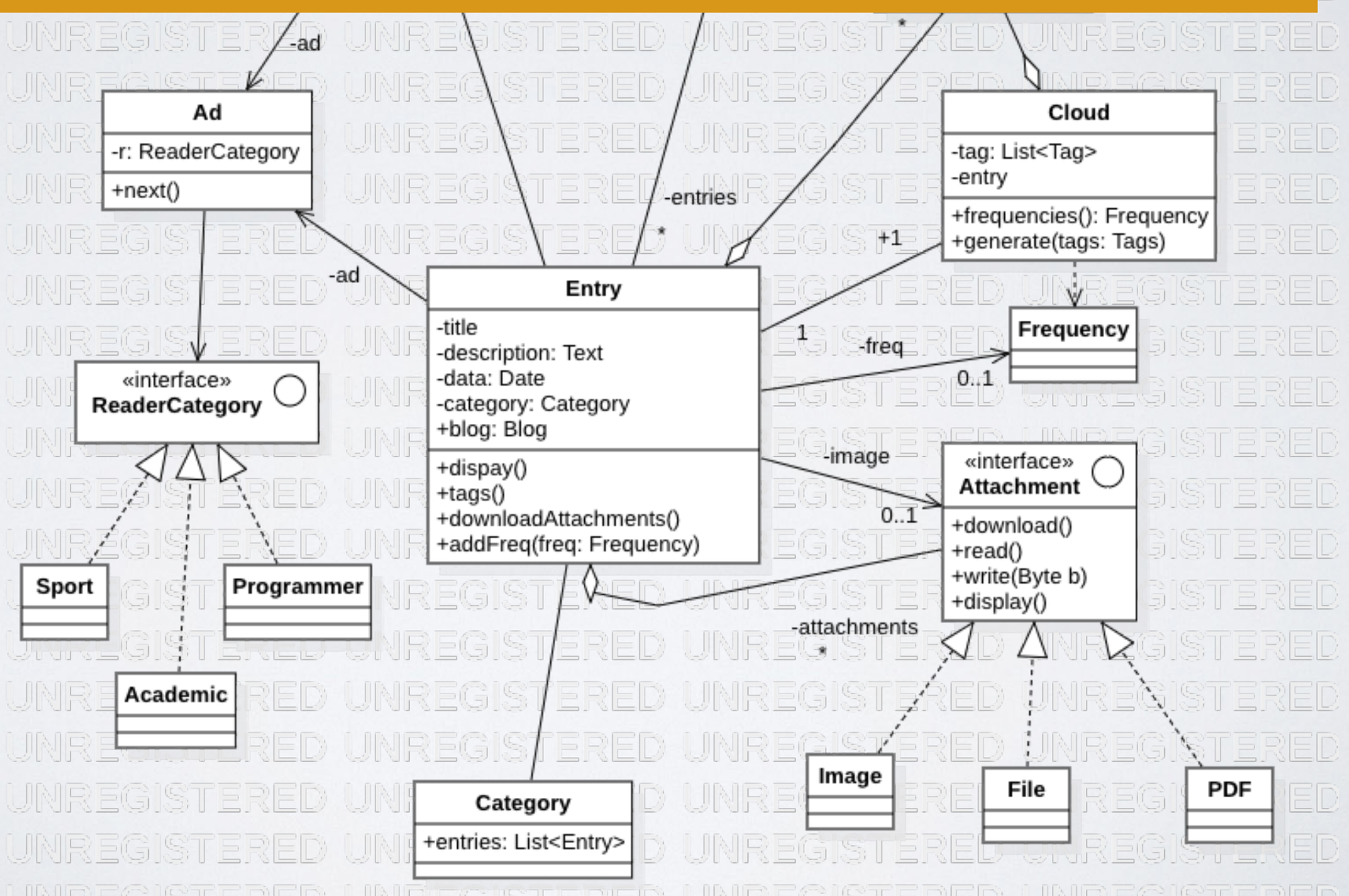


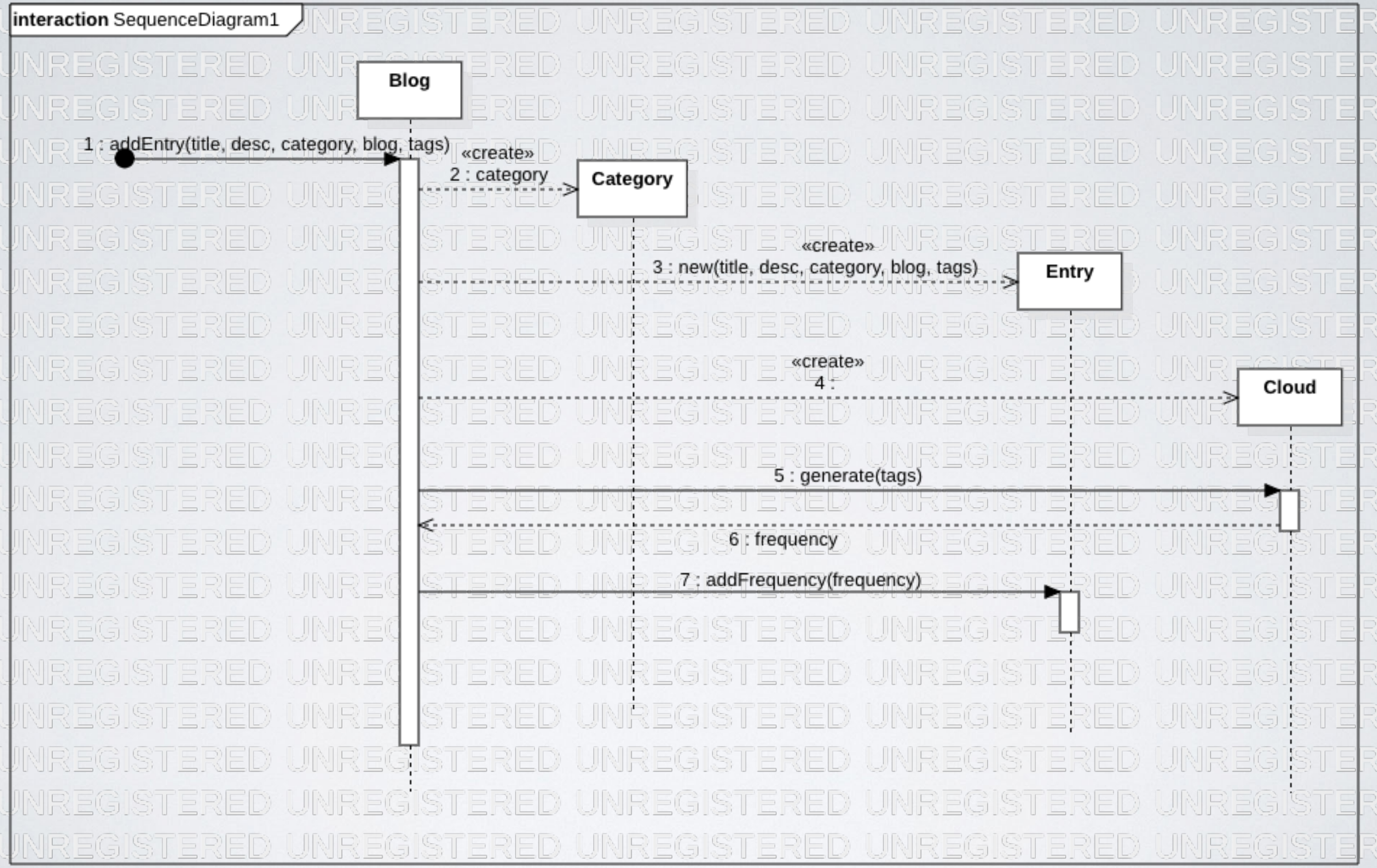
BLOG EXERCISE



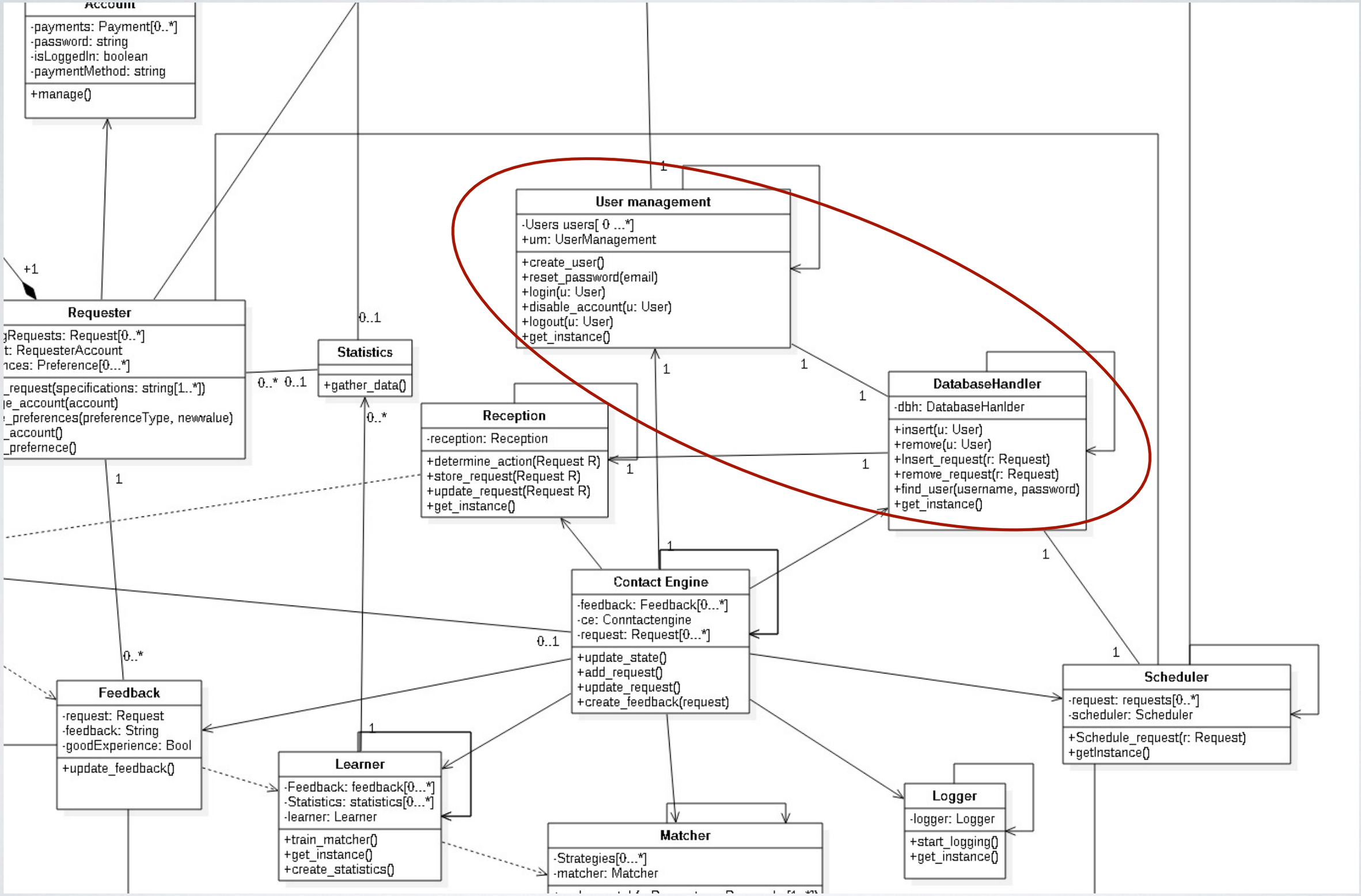
Draw sequence diagram starting from the `Blog` class, when someone calls method:

```
addEntry(title, desc, category, blog, tags)
          String  String  String    Blog  String
```

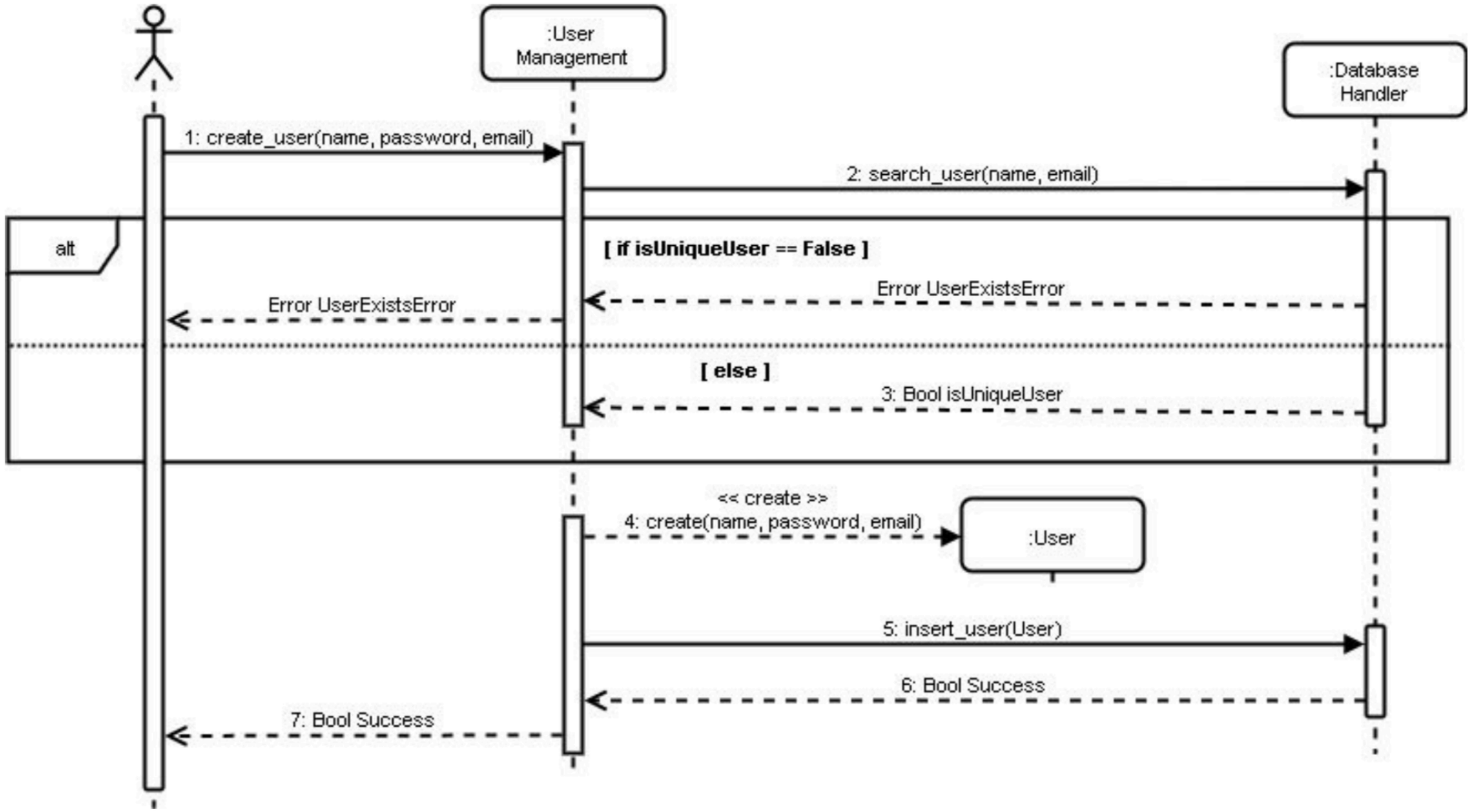




How would you create a user?



Creating a user:



OTHER BEHAVIOURAL MODELS

COMMUNICATION DIAGRAMS

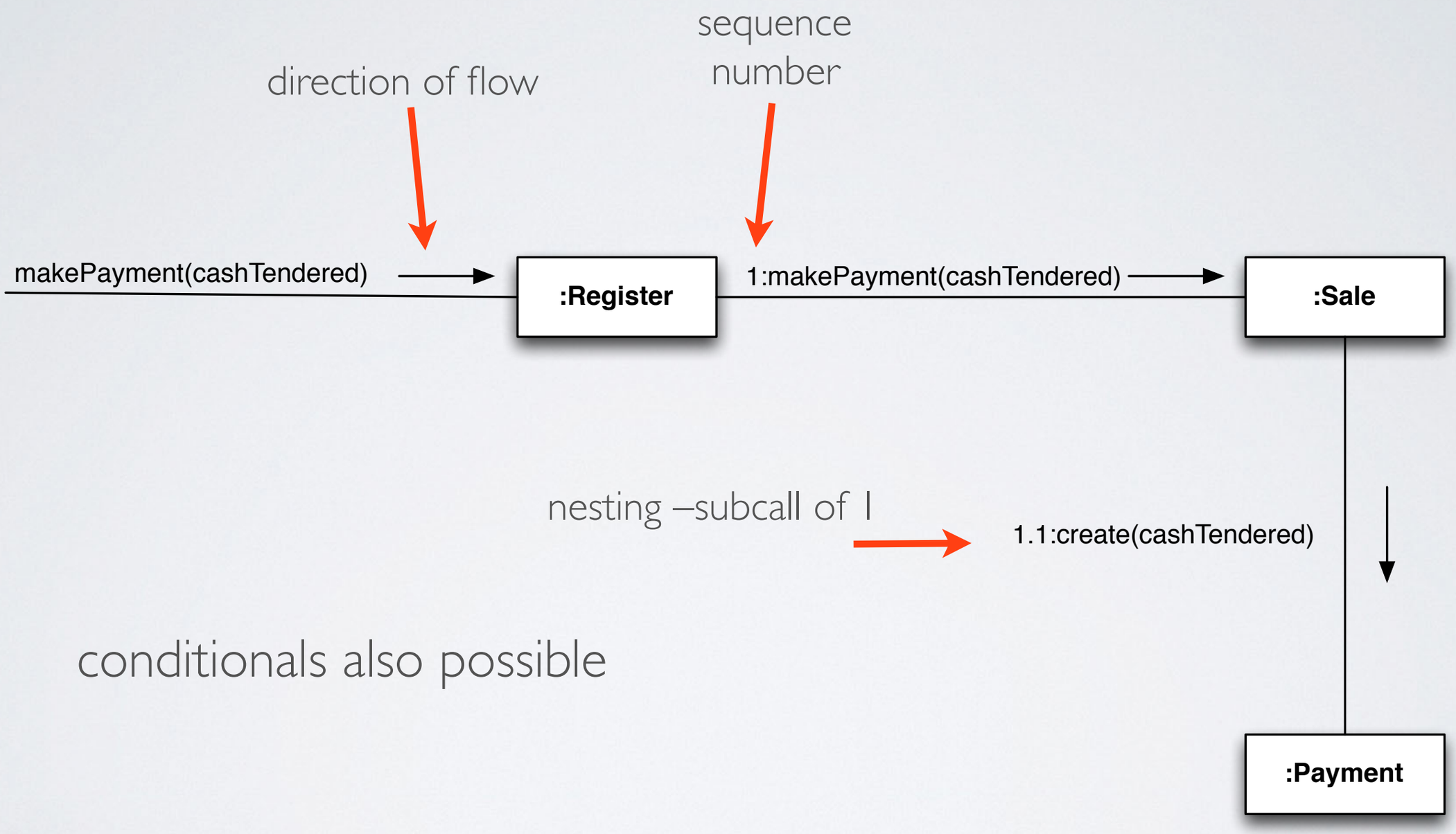
COMMUNICATION DIAGRAM

illustrate **object interactions in a graph** or network format

objects are placed anywhere in the diagram

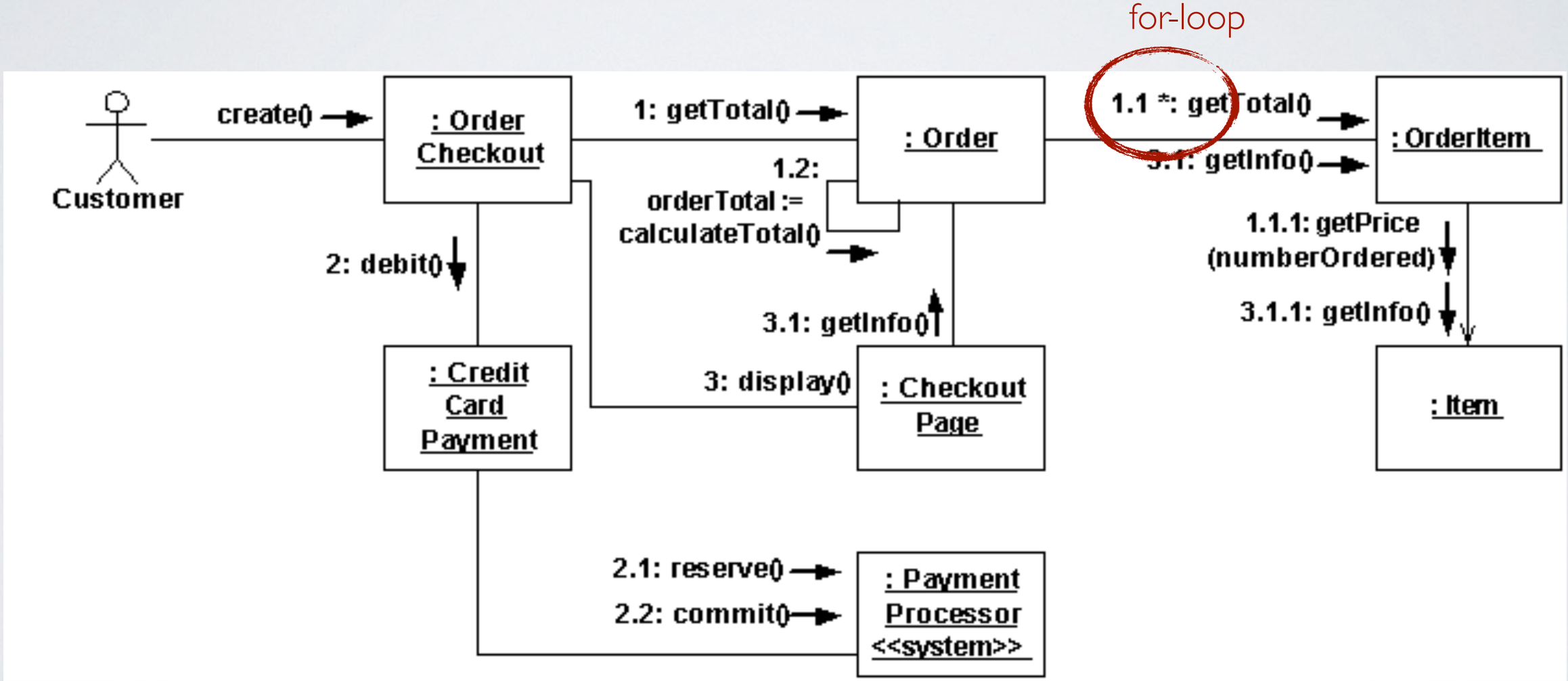
captures essence of **wall sketch**

EXAMPLE



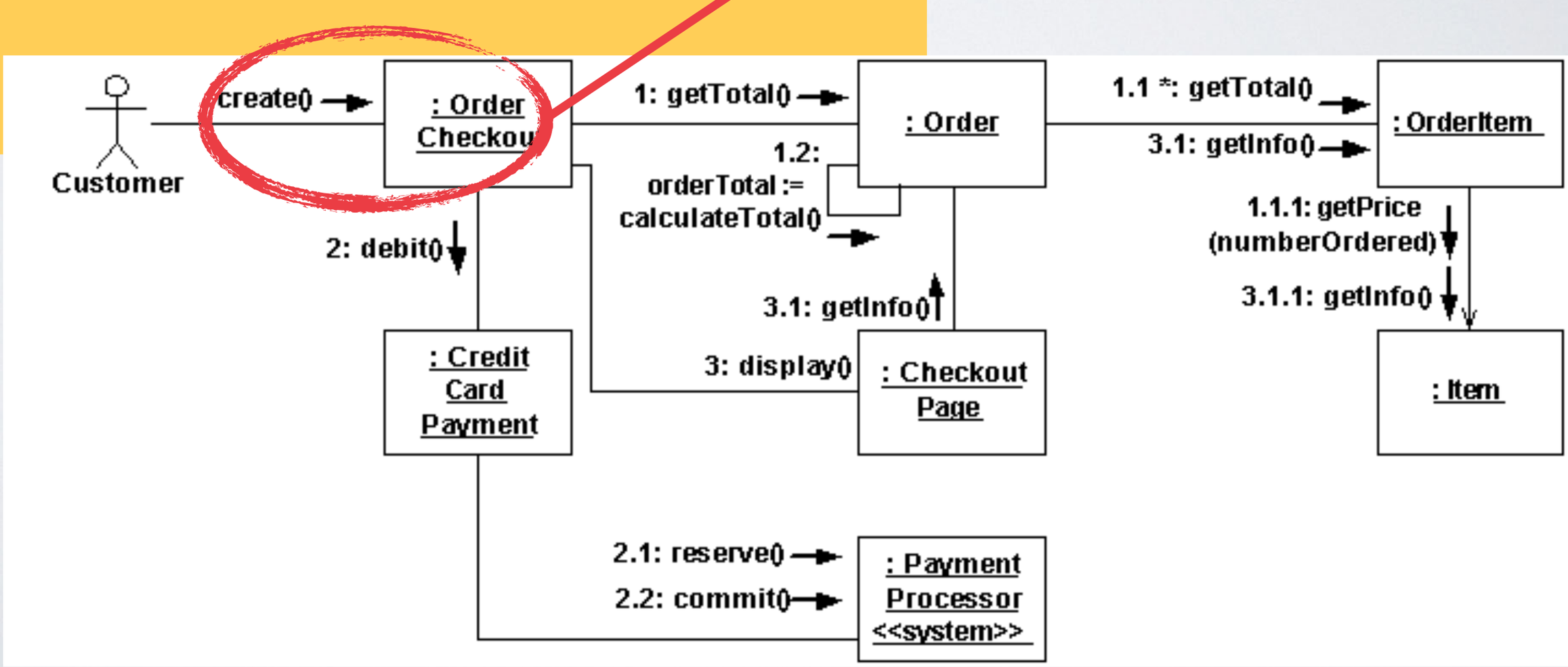
conditionals also possible

LARGER EXAMPLE



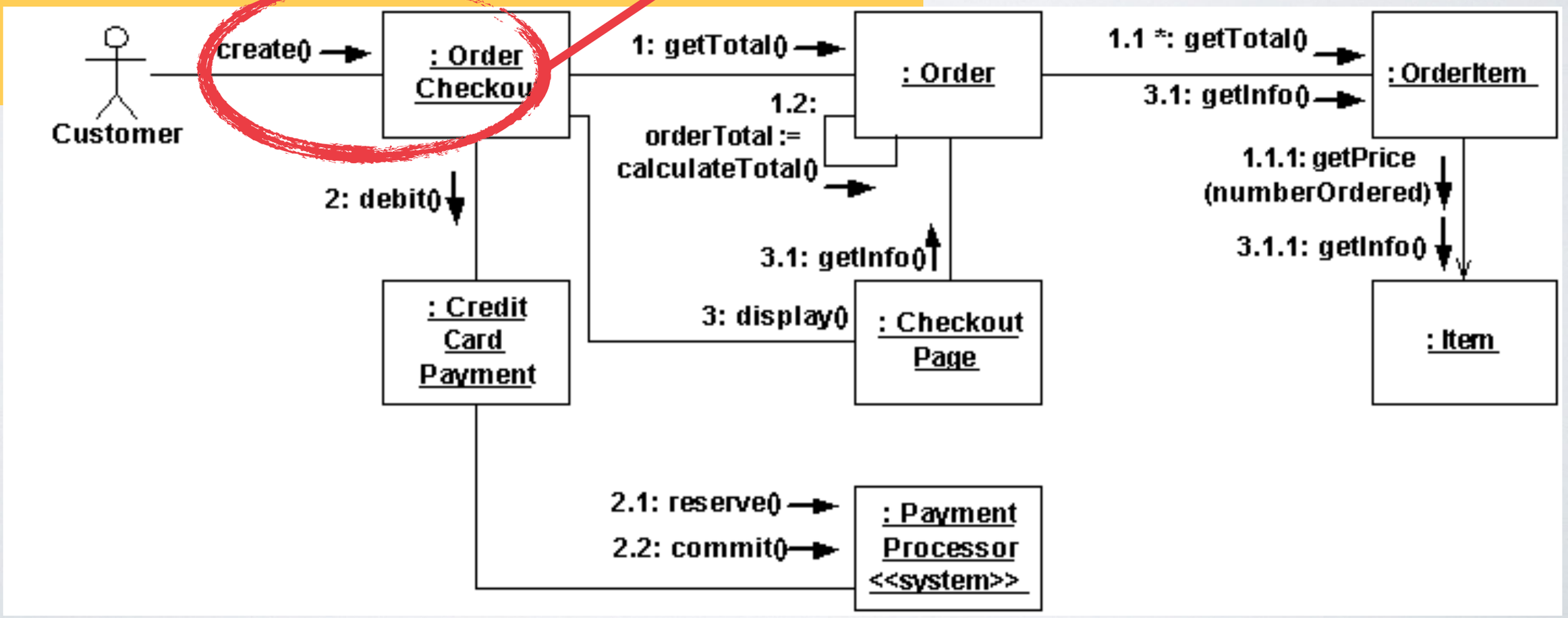
LET'S MAP THE DIAGRAM TO CODE

Write the code that represents the **create()** method

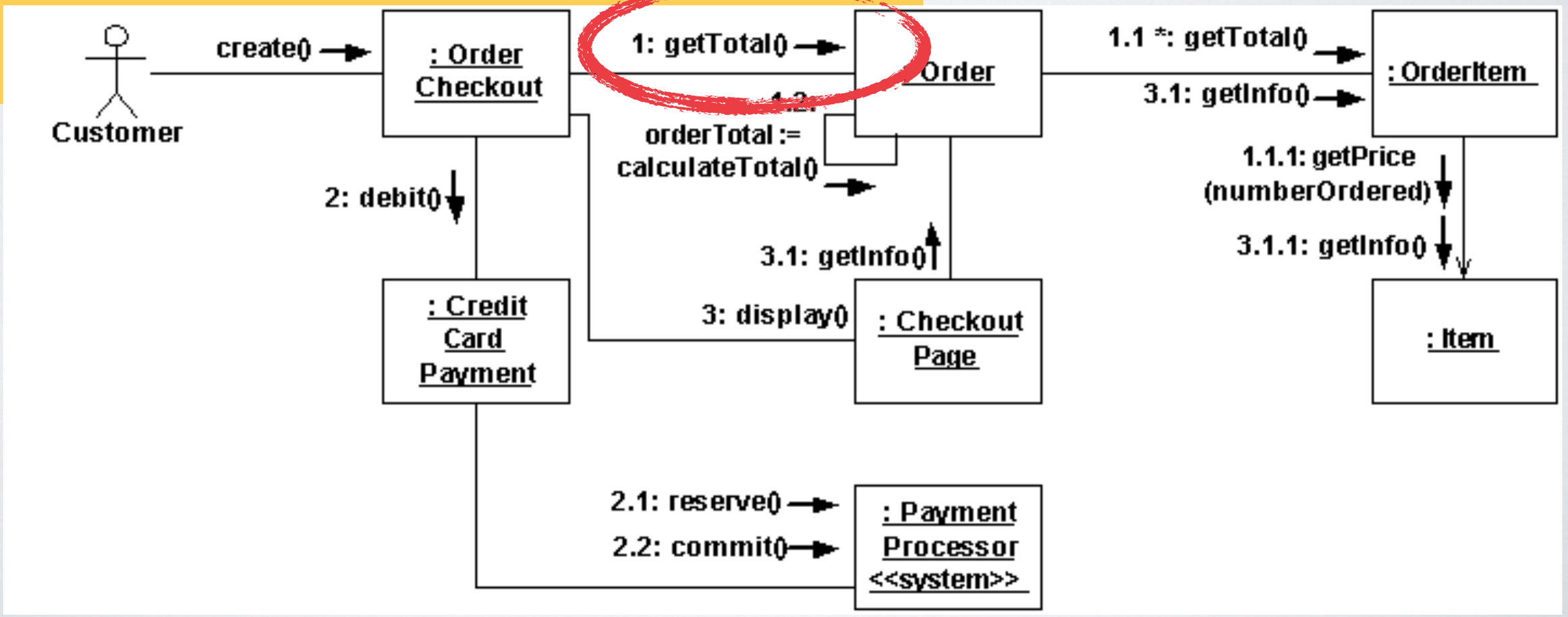


```
class OrderCheckout {
    private Order order = new Order();
    private CreditCardPayment p = new …
    …

  public OrderCheckout(){
  (1) this.order.getTotal();
  (2) this.p.debit();
  (3) this.checkoutPage.display();
    }
}
```

Write the code that represents the **create()** method

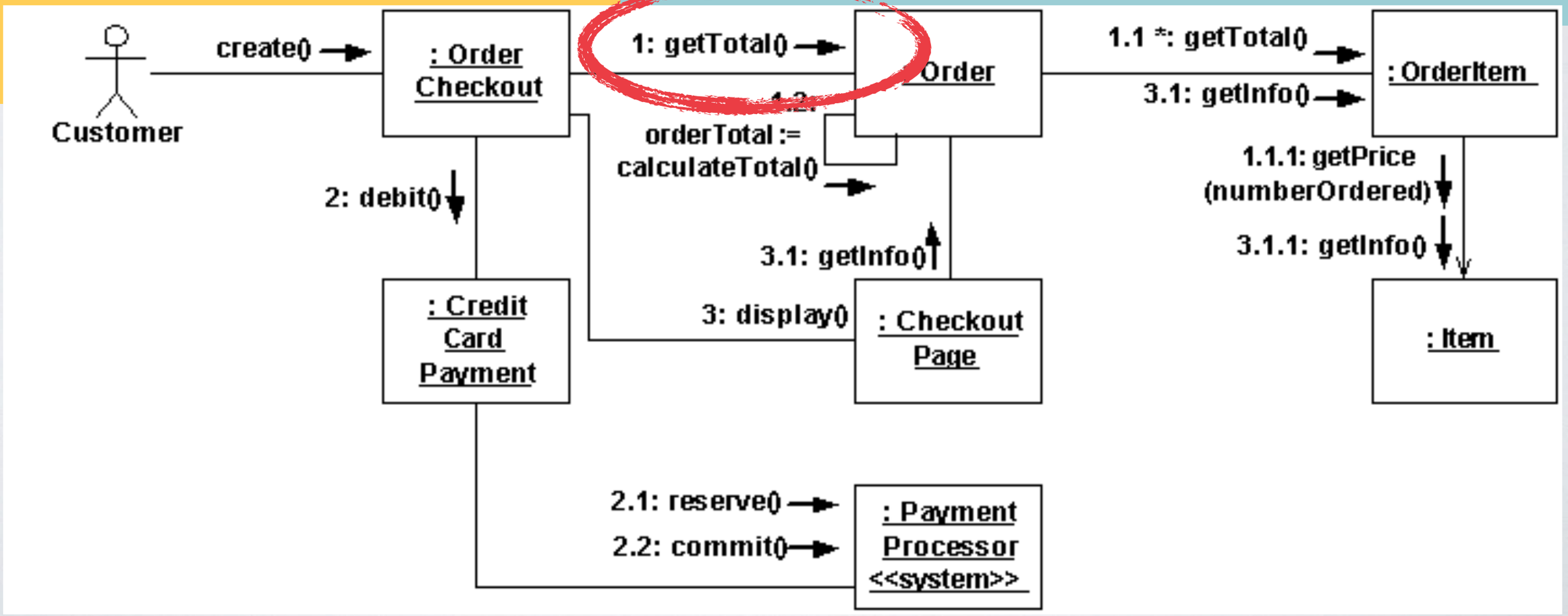


```
class OrderCheckout {
   private Order order = new Order();
   private CreditCardPayment p = new …
   …

  public OrderCheckout(){
(1) this.order.getTotal();
(2) this.p.debit();
(3) this.checkoutPage.display();
  }
}
```



```
class OrderCheckout {
    private Order order = new
    private CreditCardPayment
    …

   public OrderCheckout(){
  (1) this.order.getTotal();
  (2) this.p.debit();
  (3) this.checkoutPage.disp
    }
}
```

```
 class Order {

(1)public getTotal(){
(1.1)for(item: this.orderItems){
        item.getTotal();
      }
(1.2)orderTotal = calculateTotal()
    }
}
```



SEQUENCE VS COMMUNICATION DIAGRAMS

Sequence diagrams:

easier to see flow, just follow arrows (vs numbers)

large number of notation options

rigid structure – consumes horizontal space

Communication diagrams:

easier to sketch free form, more space efficient, and easier to modify on the fly

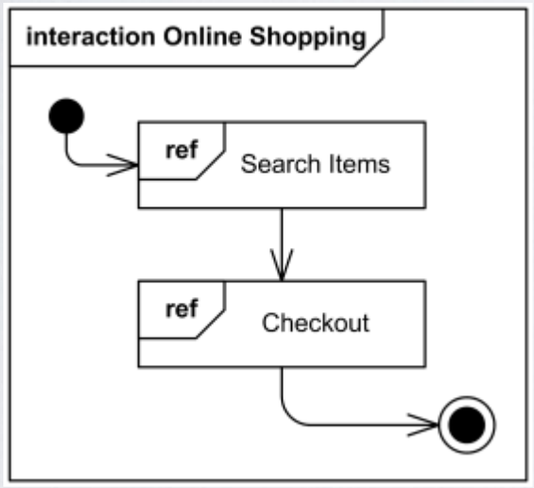
difficult to see sequence of messages

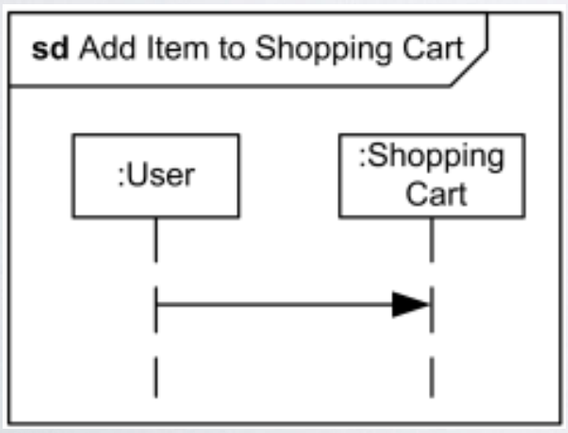
fewer notation options

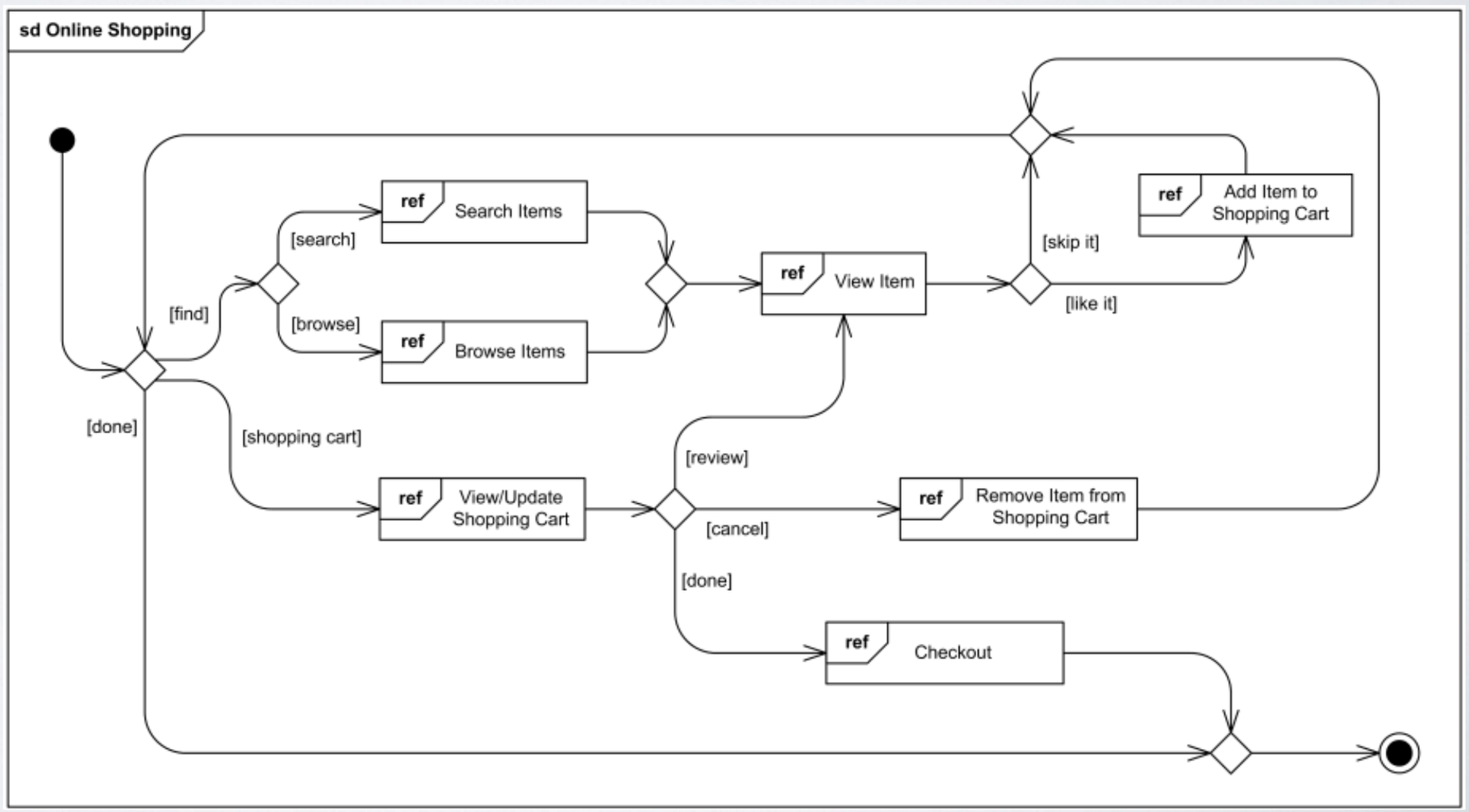
INTERACTION OVERVIEW DIAGRAMS

INTERACTION OVERVIEW DIAGRAM

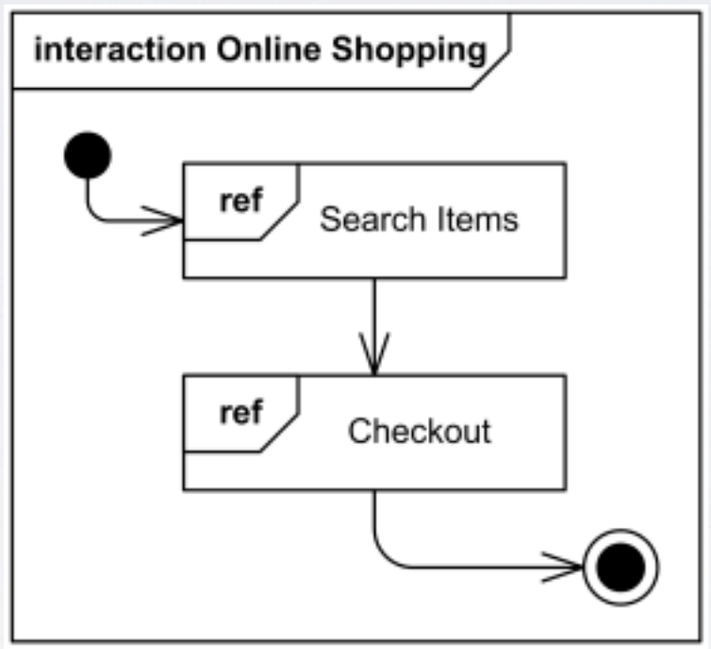
Provide a big-picture **overview** of how a set of interaction diagrams are related in terms of **logic and process flow**.





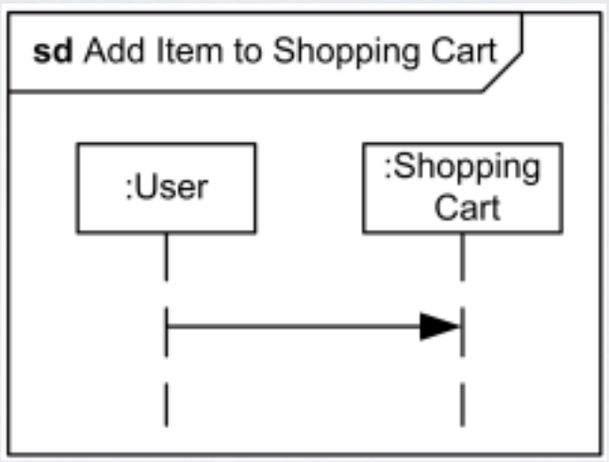


FRAMES

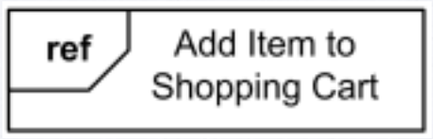


Encloses some kind of interaction diagram.
Name describe relevant component.

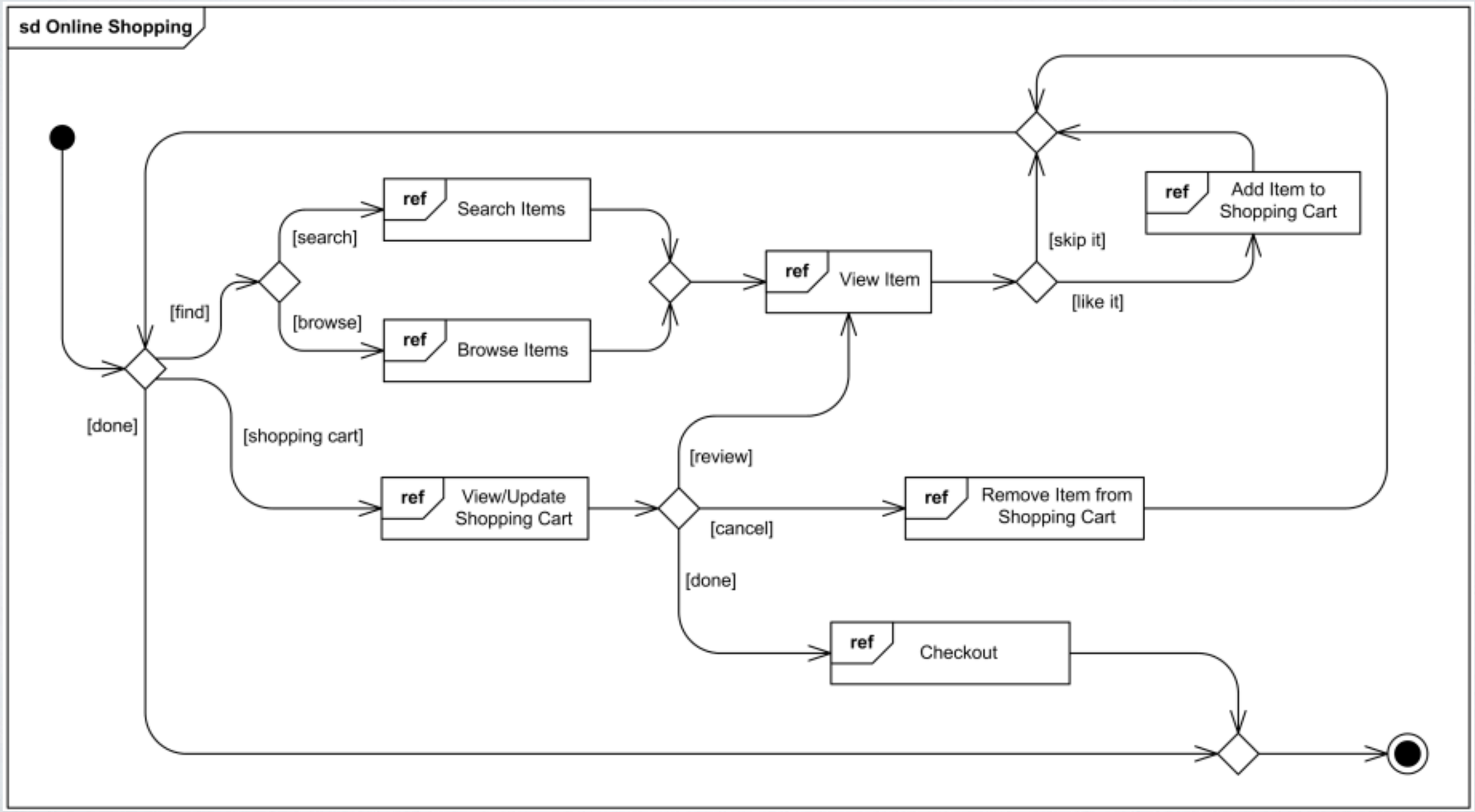
INDIRECTION



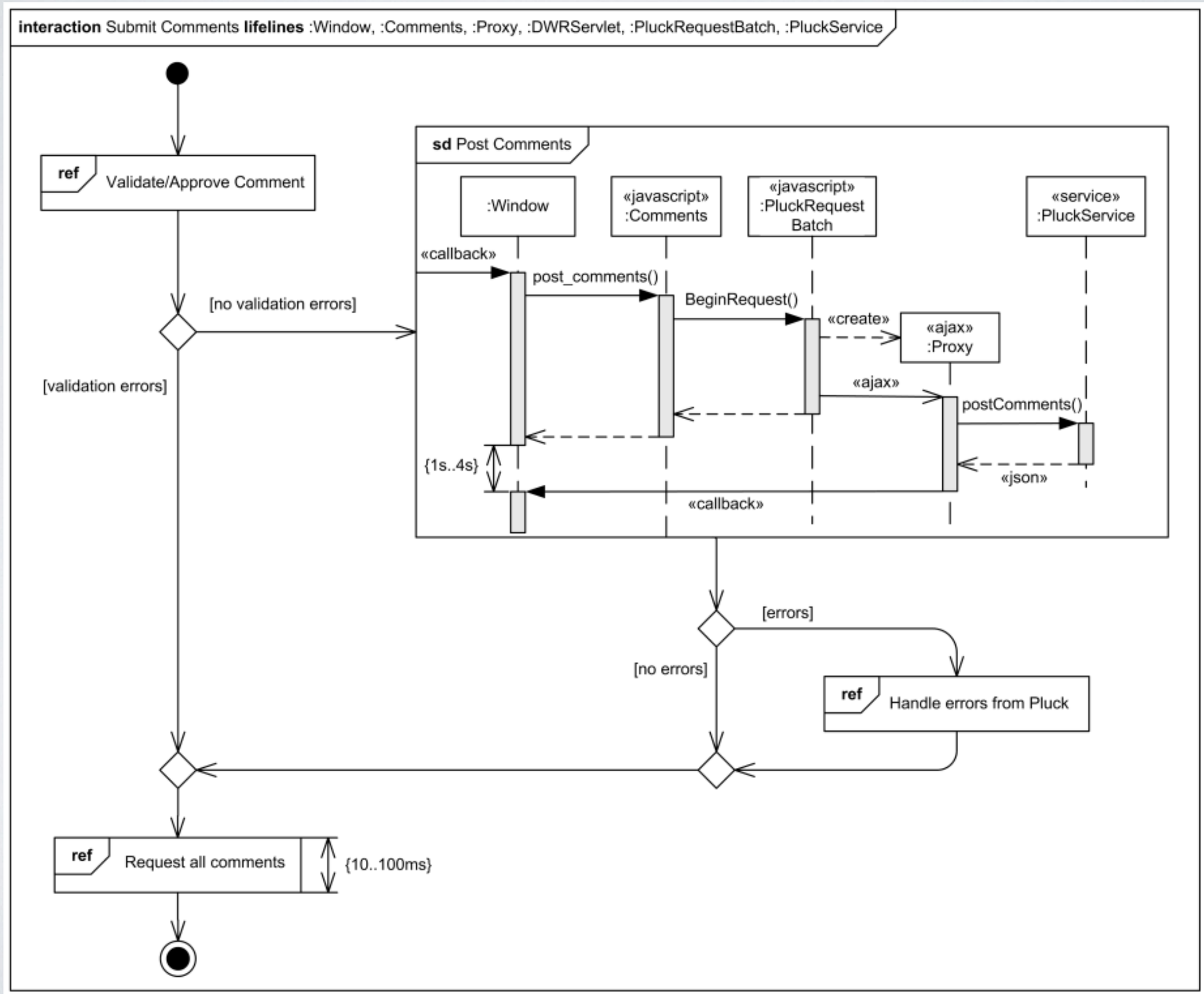
An interaction defined elsewhere
can be referred to inside an
Interaction Overview diagram.



EXAMPLE 1


sd Online Shopping
ref
Search Items
[search]
[find]
[browse]
ref
Browse Items
ref
View Item
[skip it]
[like it]
ref
Add Item to
Shopping Cart
[done]
[shopping cart]
ref
View/Update
Shopping Cart
[review]
[cancel]
ref
Remove Item from
Shopping Cart
[done]
ref
Checkout

EXAMPLE 2


interaction Submit Comments lifelines :Window, :Comments, :Proxy, :DWRServlet, :PluckRequestBatch, :PluckService
ref
Validate/Approve Comment
[no validation errors]
[validation errors]
sd Post Comments
:Window
«javascript»
:Comments
«javascript»
:PluckRequest
Batch
«service»
:PluckService
«callback»
post_comments()
BeginRequest()
«create»
«ajax»
:Proxy
«ajax»
postComments()
«json»
{1s..4s}
«callback»
[errors]
[no errors]
ref
Handle errors from Pluck
ref
Request all comments
{10..100ms}

STATE MACHINE DIAGRAMS

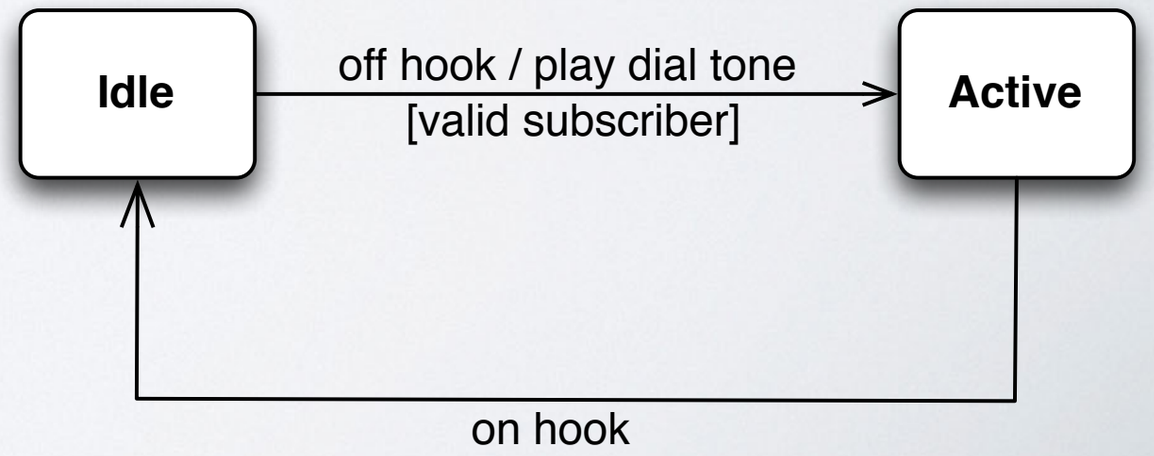
STATE MACHINE DIAGRAM

illustrates the **behaviour** of an object in **reaction** to an event based on interesting events and states of an object

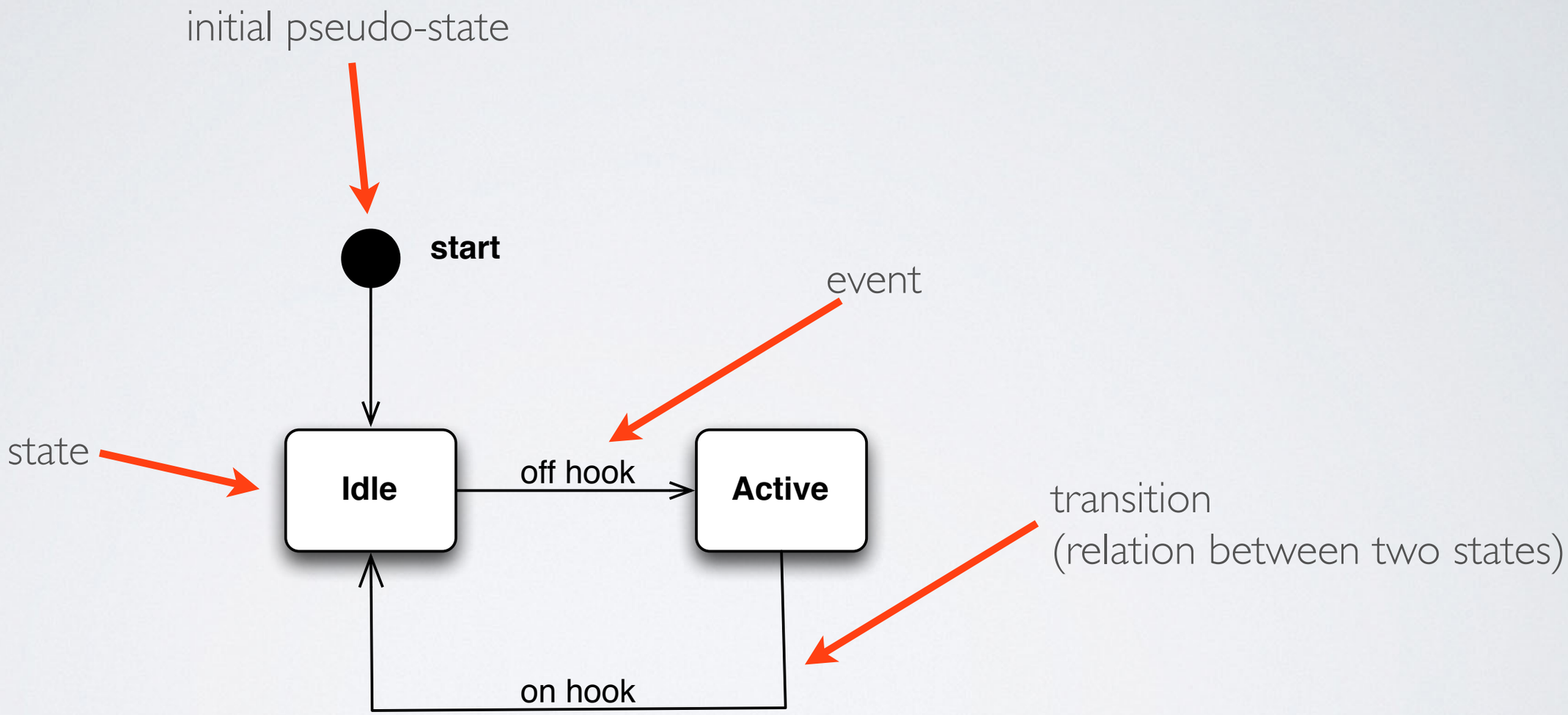
show the lifecycle of an object

use only for *state-dependent* objects

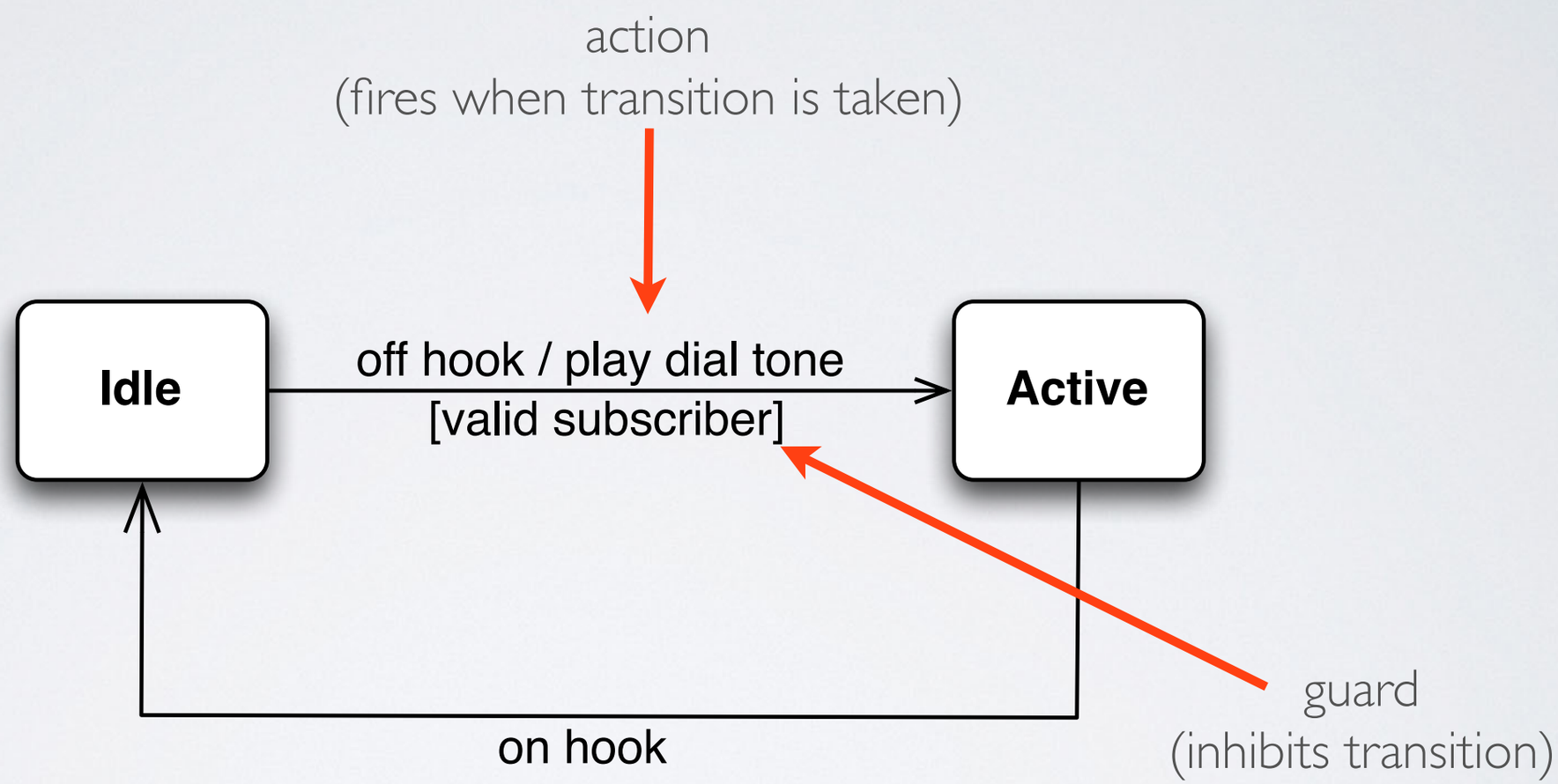
common in process control, device control, **protocol handlers**, and telecommunications



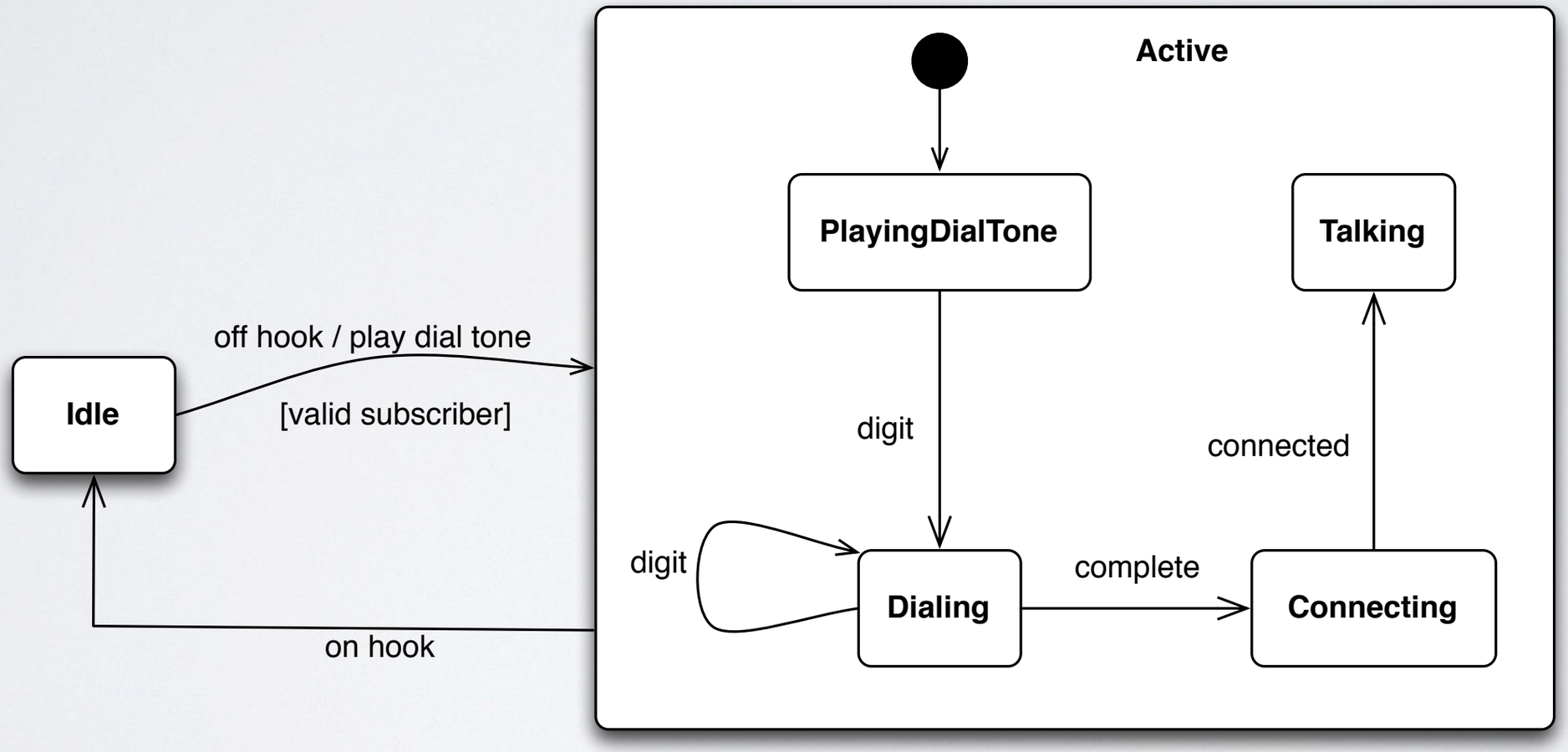
EXAMPLE


initial pseudo-state
start
event
state
Idle
off hook
Active
transition
(relation between two states)
on hook

GUARDS AND ACTIONS



NESTED STATES



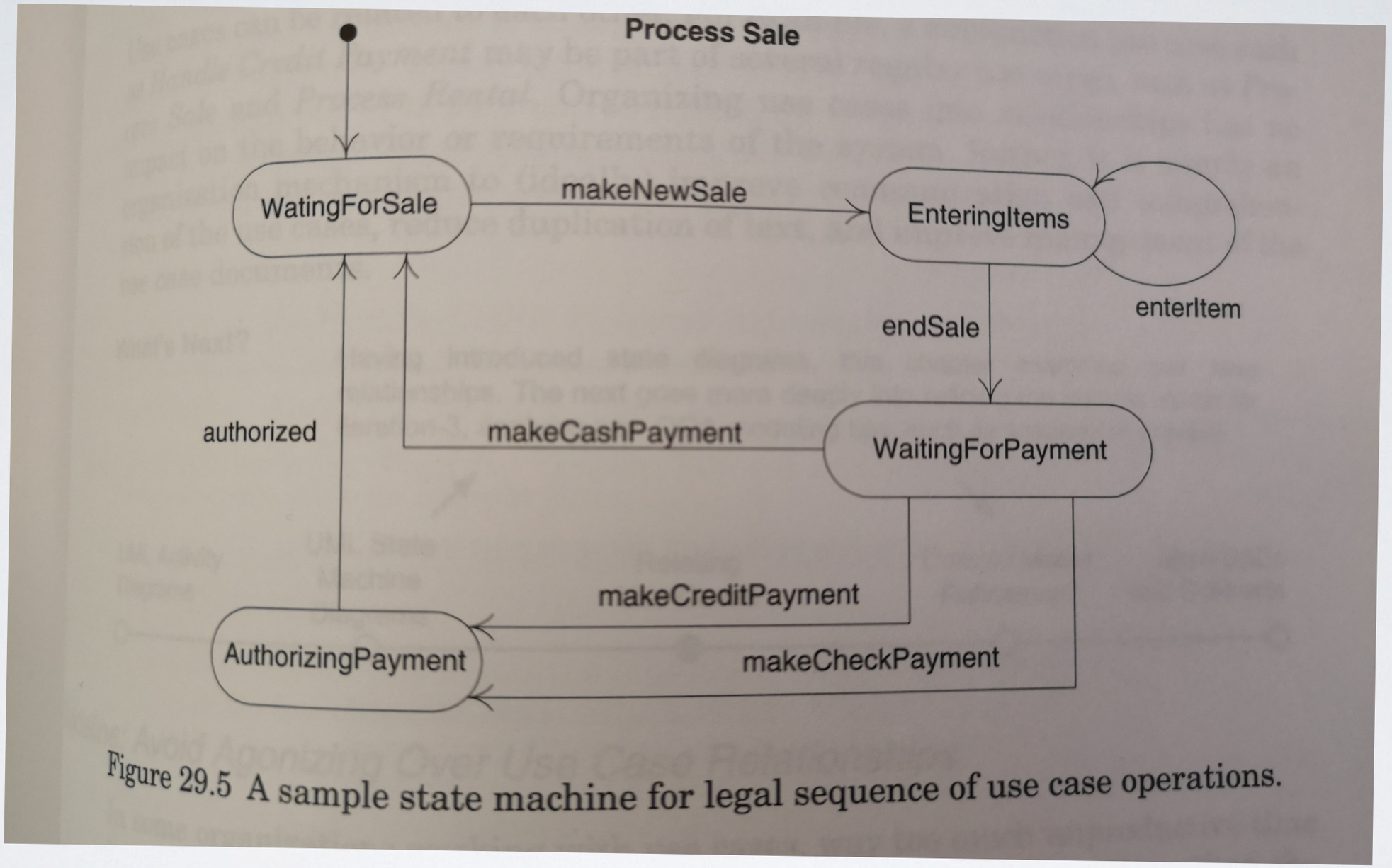
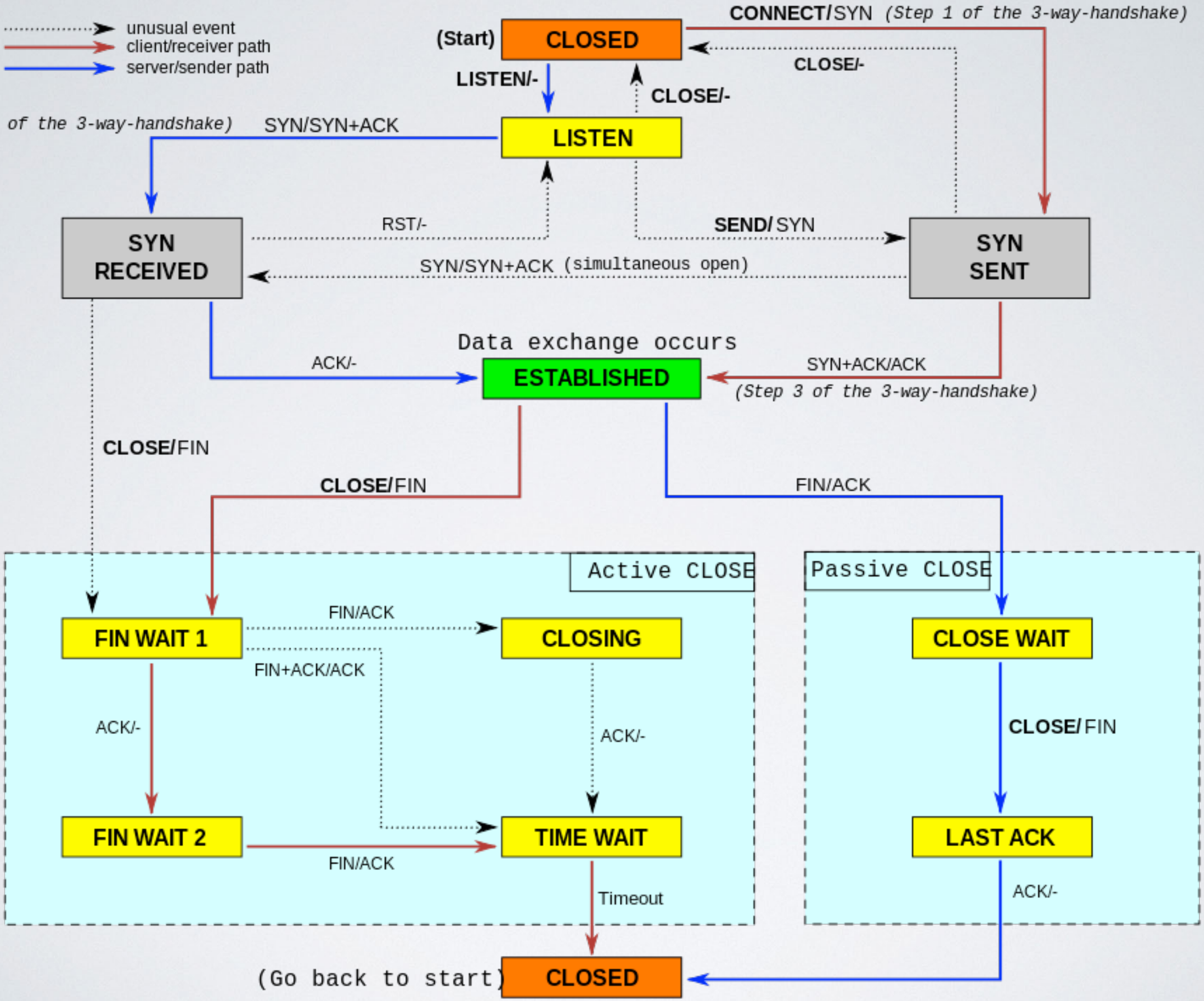


Figure 29.5 A sample state machine for legal sequence of use case operations.

Applying UML and Patterns, Craig Larman, p. 491

EXAMPLE

- TCP Protocol


unusual event
client/receiver path
server/sender path
(Start)
CLOSED
CONNECT/SYN (Step 1 of the 3-way-handshake)
CLOSE/-
LISTEN/-
CLOSE/-
(Step 2 of the 3-way-handshake)
SYN/SYN+ACK
LISTEN
RST/-
SEND/SYN
SYN RECEIVED
SYN/SYN+ACK (simultaneous open)
SYN SENT
Data exchange occurs
ACK/-
ESTABLISHED
SYN+ACK/ACK
(Step 3 of the 3-way-handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK
Active CLOSE
Passive CLOSE
FIN WAIT 1
FIN/ACK
CLOSING
FIN+ACK/ACK
CLOSE WAIT
ACK/-
ACK/-
CLOSE/FIN
FIN WAIT 2
TIME WAIT
LAST ACK
FIN/ACK
Timeout
ACK/-
(Go back to start)
CLOSED

CONCLUSION

- Behaviour modelling for
  - expressing dynamic interactions
    - runtime behaviour
- Most important / common diagrams:
  - **Sequence**
  - Communication

NEXT LECTURE

GRASP principles:

- General guidelines for **designing flexible software**