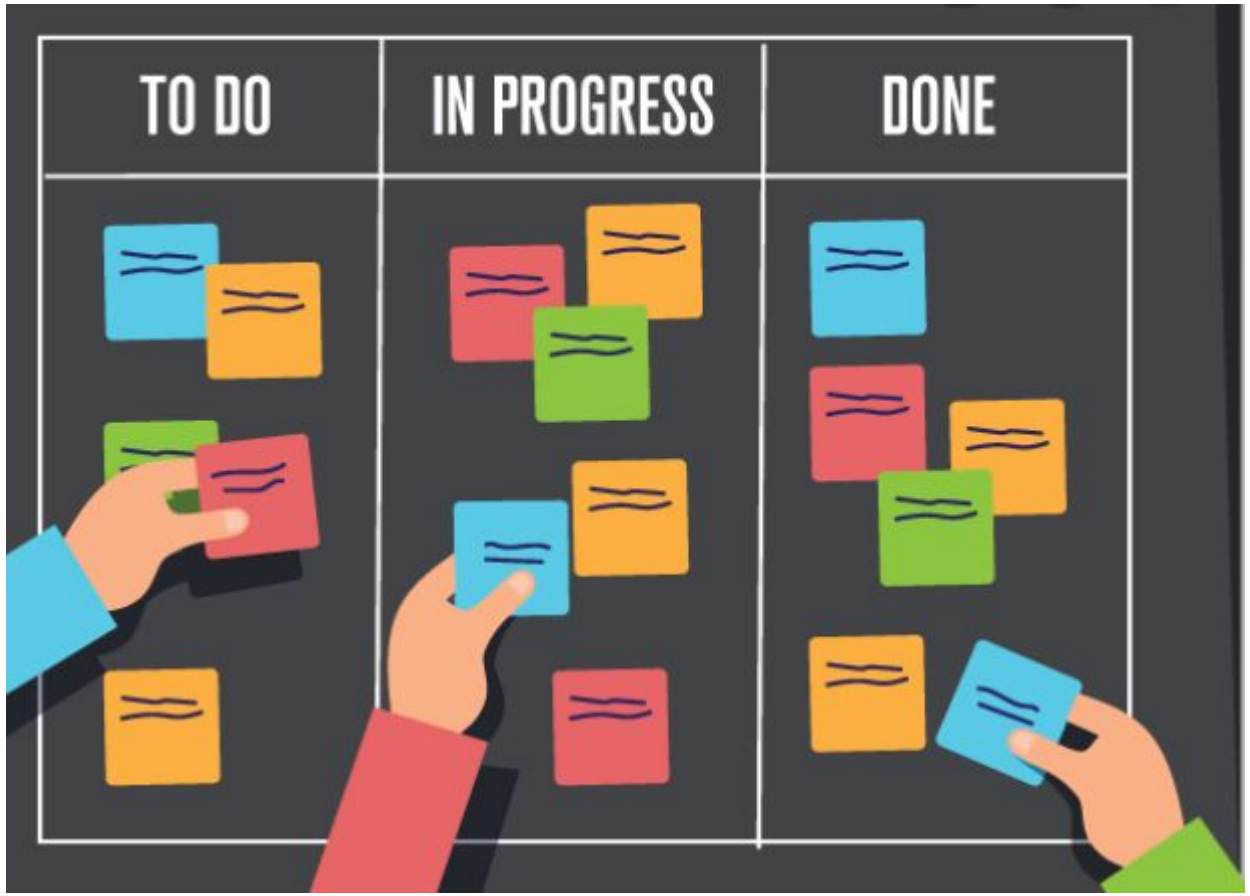
SCRUM

Kanban Board

A visual approach to project management. We can create Kanban boards to better manage the project.



Chapter#7

Modeling Ordered Interactions: Sequence Diagrams

Use cases allow your model to describe what your system must be able to do; **classes** allow your model to describe the different types of parts that make up your system's structure. With use cases and classes alone, you can't yet model how your system is actually doing to its job.

A sequence diagram describes the order in which the interactions (between system components (UML 2.0) / objects (UML 1.x)) take place.

The order that interactions are placed down the page on a sequence diagram indicates the order in which those interactions will take place. Time on a sequence diagram is all about ordering, not duration.

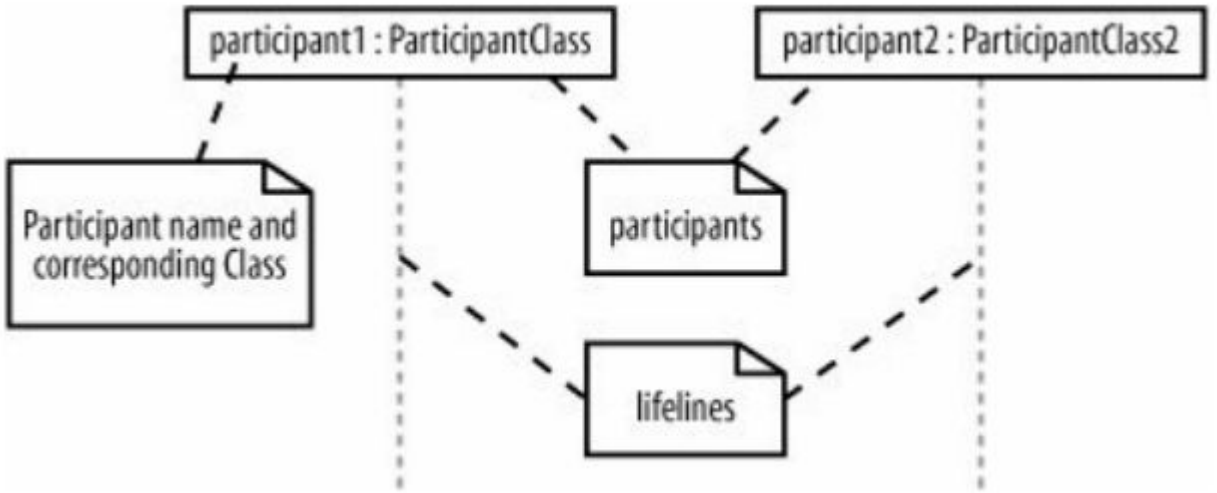
Modeling Ordered Interactions: Sequence Diagrams

Sequence diagrams are all about capturing the order of interactions between parts of your system.

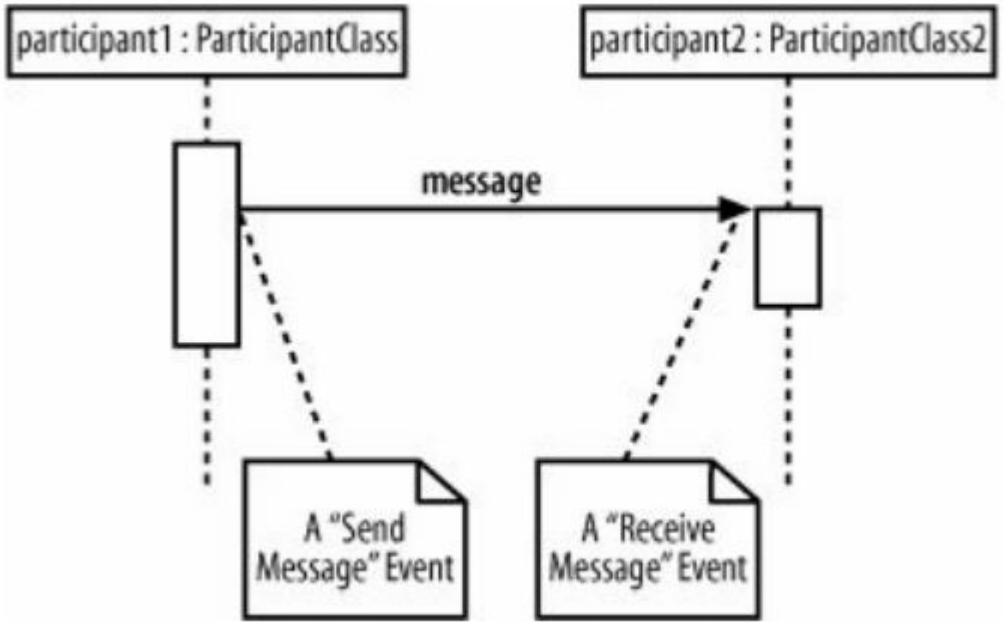
Using a sequence diagram, you can describe which interactions will be triggered when a particular use case is executed and in what order those interactions will occur.

A sequence diagram is made up of a c**ollection of participants (the parts of your system)** that interact with each other during the sequence. Where a participant is placed on a sequence diagram is important.

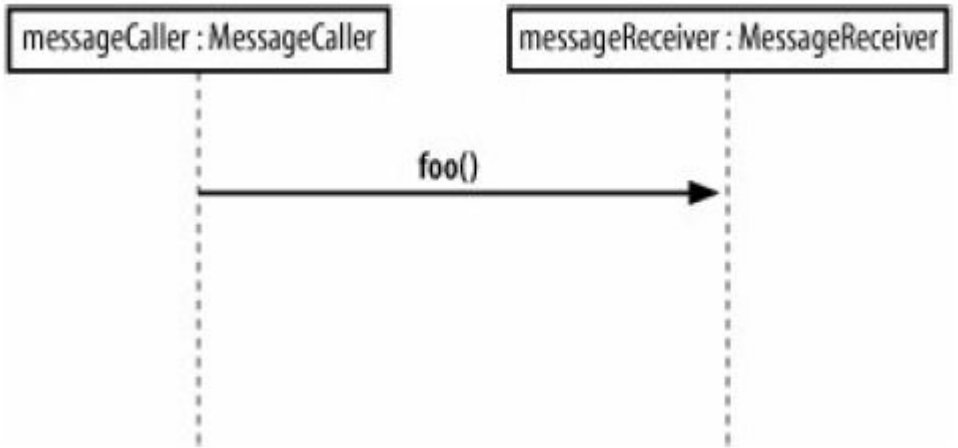
Participants in a Sequence Diagram



Messages

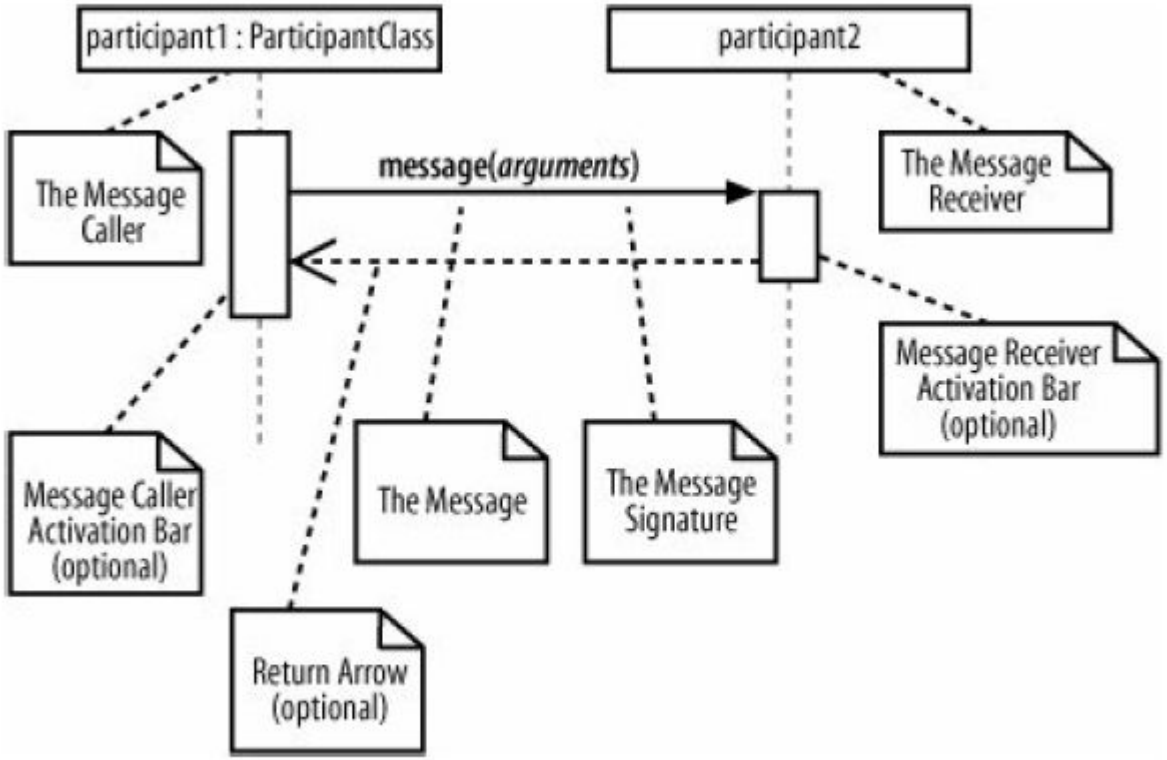

participant1 : ParticipantClass
participant2 : ParticipantClass2
message
A "Send Message" Event
A "Receive Message" Event

Sending Message(s)



Sending Message(s)

The smallest part of an interaction is an event. An interaction in a sequence diagram occurs when one participant decides to send a message to another participant



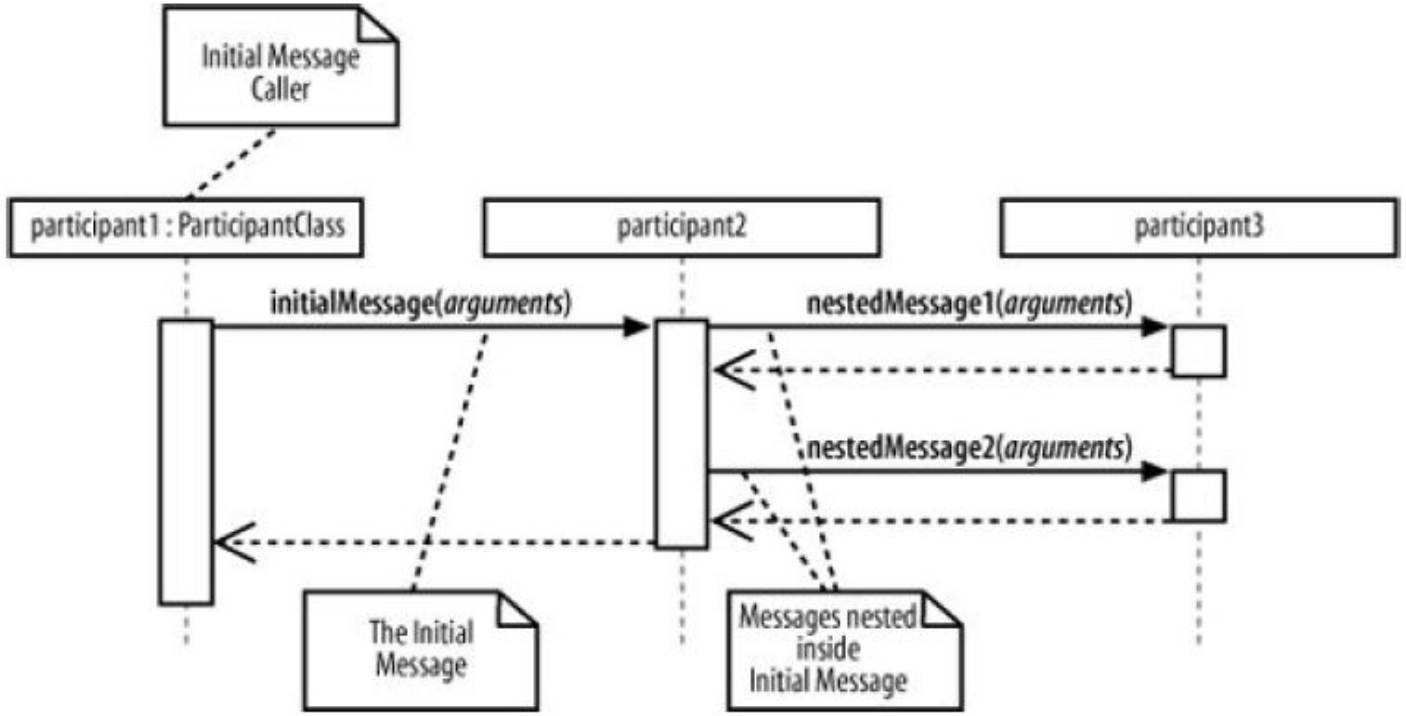
Sending Message(s)

```
public class MessageReceiver
{
   public void foo(  )
   {
      // Do something inside foo.
   }
}

public class MessageCaller
{
   private MessageReceiver messageReceiver;

   // Other Methods and Attributes of the class are declared here

   // The messageRecevier attribute is initialized elsewhere in
   // the class.

   public doSomething(String[] args)
   {
      // The MessageCaller invokes the foo(  ) method

      this.messageReceiver.foo(  ); // then waits for the method to return

      // before carrying on here with the rest of its work
   }
}
```

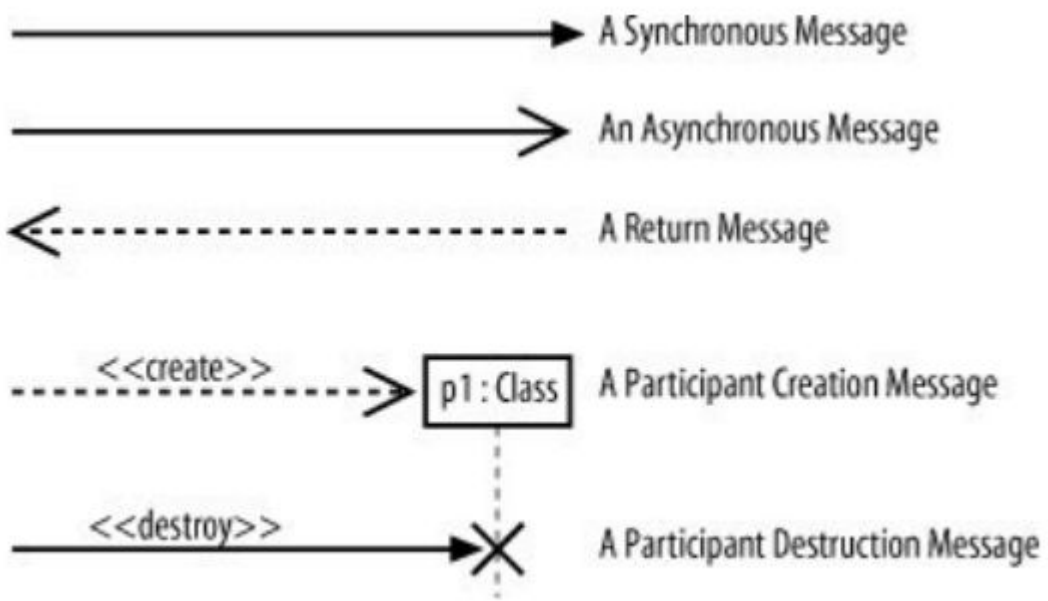
Sending Nested Message(s)

When a message from one participant results in one or more messages being sent by the receiving participant, those resulting messages are said to be nested within the triggering message.

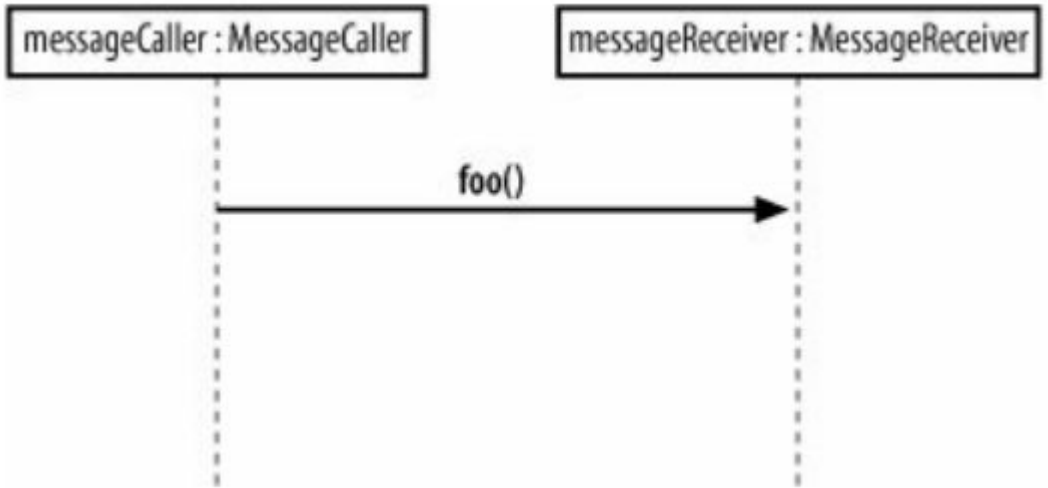
2 nested messages sent by participant2 to participant3.



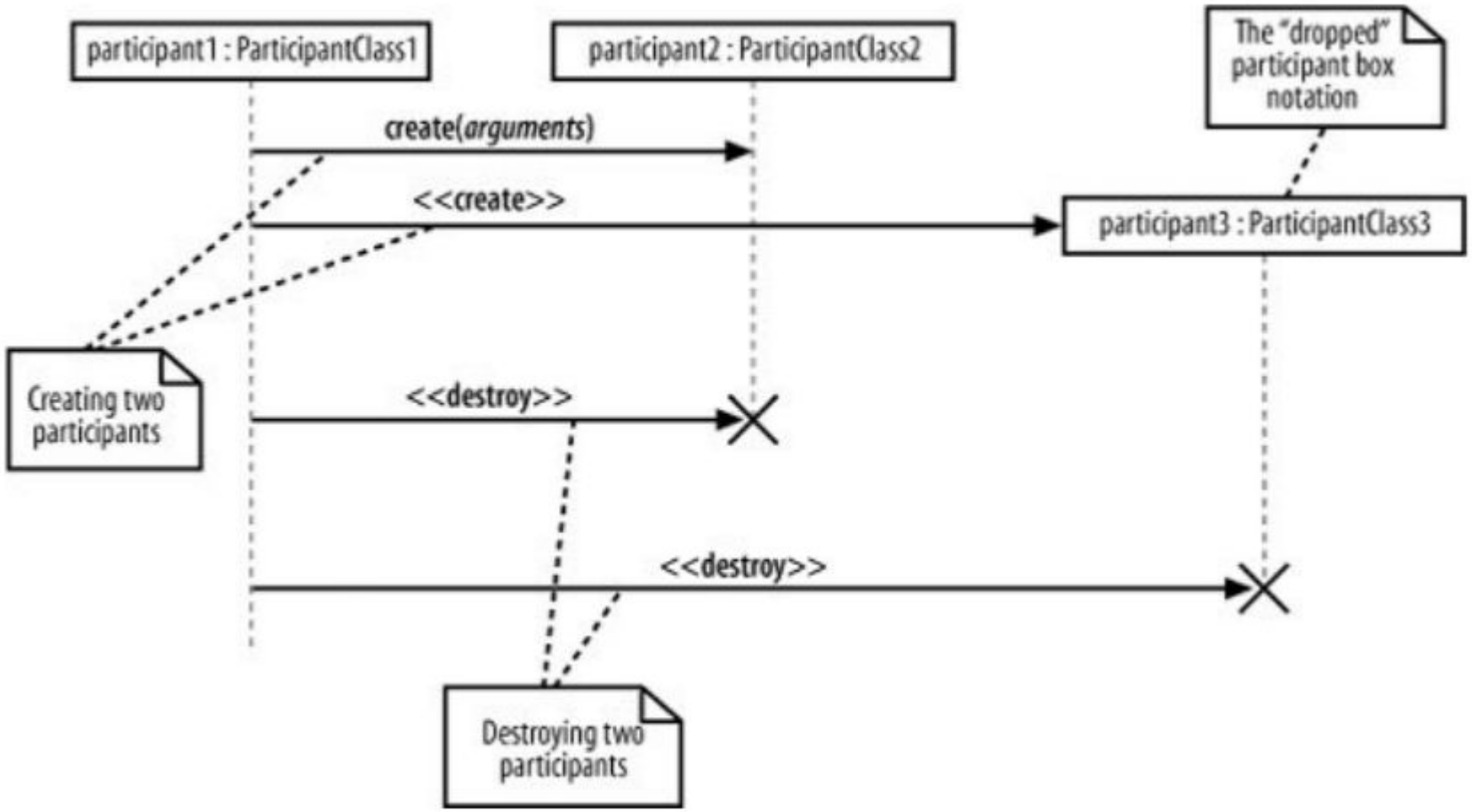
Types of Message Arrows



Synchronous Message



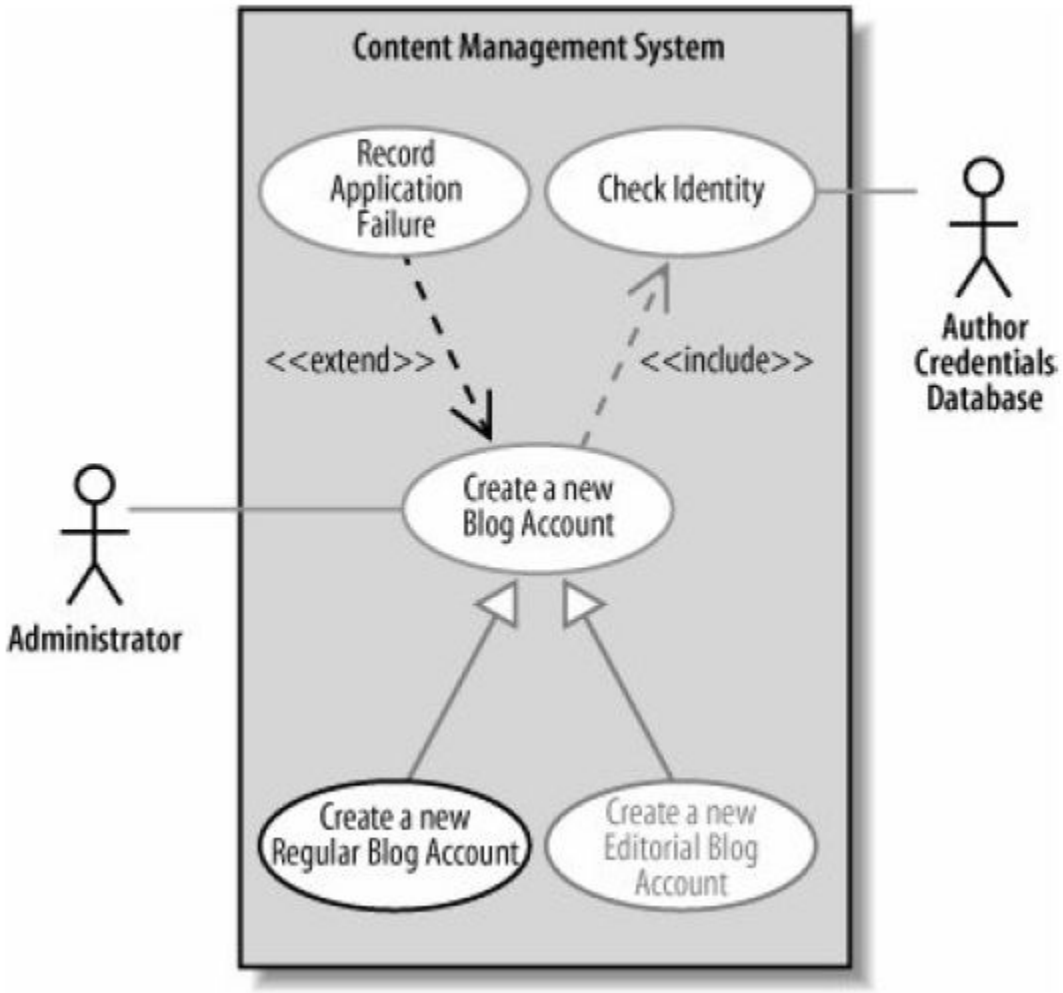
Participant Creation and Destruction Messages



Example: Bringing a Use Case to Life with a Sequence Diagram

Step	Action
1	The Administrator asks the system to create a new blog account.
2	The Administrator selects the regular blog account type.
3	The Administrator enters the author's details.
4	The author's details are checked using the Author Credentials Database.
5	The new regular blog account is created.
6	A summary of the new blog account's details are emailed to the author.

Figure 7-13. The Create a new Regular Blog Account use case diagram



Sequence Diagram For the Previous Use Case