**Sequence Diagrams**

Sequence diagrams to figure out what happens in the software.

A sequence diagram captures the behavior of a single scenario. The diagram shows a number of example objects and the messages that are passed between these objects within the use case.

Sequence diagrams show the interaction by showing each participant with a lifeline that runs vertically down the page and the ordering of messages by reading down the page.

Each lifeline has an activation bar that shows when the participant is active in the interaction. This corresponds to one of the participant's methods being on the stack.

Naming often is useful to correlate participants on the diagram.

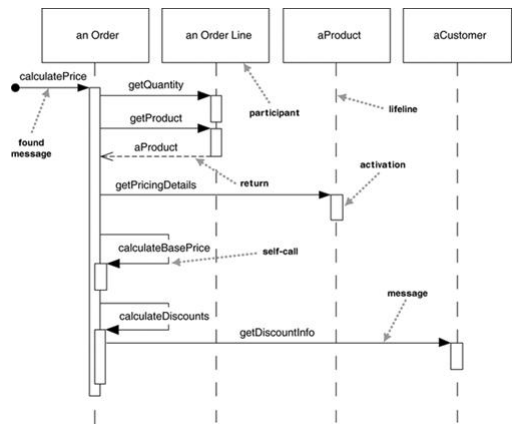
2 main types of designs:

centralized control - with one participant pretty much doing all the processing and other participants there to supply data.

Distributed control - in which the processing is split among many participants, each one doing a little bit of the algorithm.



^^^This example is distributed



^^^This example is centralised – actor messaging each object

When to use Sequence Diagrams:

You should use sequence diagrams when you want to look at the behaviour of several objects within a single use case. Sequence diagrams are good at showing collaborations among the objects; they are not so good at precise definition of the behaviour.

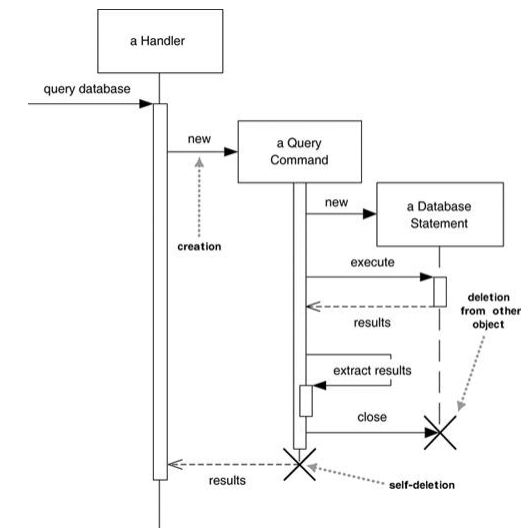
If you want to explore multiple alternative interactions quickly, you may be better off with CRC cards, as that avoids a lot of drawing and erasing. It's often handy to have a CRC card session to explore design alternatives and then use sequence diagrams to capture any interactions that you want to refer to later.

Activity = chapter 11

Object deletion:

Deletion of a participant is indicated by big X. A message arrow going into the X indicates one participant explicitly deleting another; an X at the end of a lifeline shows a participant deleting itself.

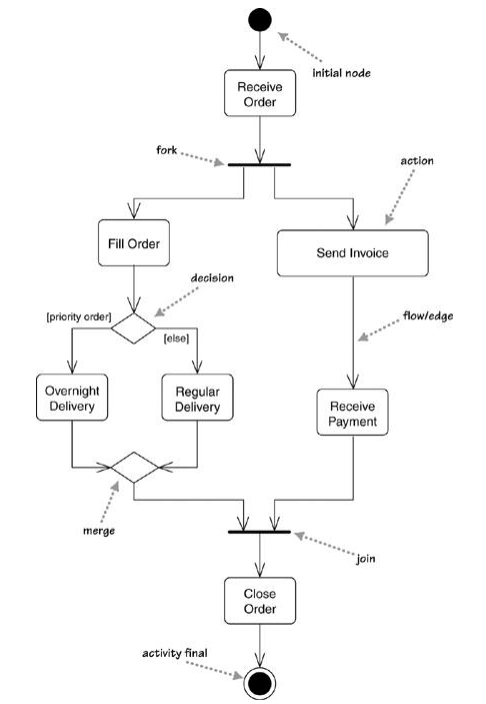
In a garbage-collected environment, you don't delete objects directly, but it's still worth using the X to indicate when an object is no longer needed and is ready to be collected. It's also appropriate for close operations, indicating that the object isn't usable any more.



**Activity Diagrams:**

Activity diagrams are a technique to describe procedural logic, business process, and work flow. In many ways, they play a role similar to flowcharts, but the principal difference between them and flowchart notation is that they support parallel behaviour.

Activity diagrams tell you what happens, but they do not tell you who does what. In programming, this means that the diagram does not convey which class is responsible for each action.



We begin at the initial node action and then do the action Receive Order. Once that is done, we encounter a fork. A fork has one incoming flow and several outgoing concurrent flows.

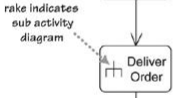
Activity refers to a sequence of actions, so the diagram shows an activity that's made up of actions.

When you have parallelism, you'll need to synchronize. We show this with the join before the Close Order action. With a join, the outgoing flow is taken only when all the incoming flows reach the join.

A decision, called branch in UML 1, has a single incoming flow and several guarded out-bound flows. Each outbound flow has a guard: a Boolean condition placed inside square brackets. Each time you reach a decision, you can take only one of the outbound flows, so the guards should be mutually exclusive. Using [else] as a guard indicates that the [else] flow should be used if all the other guards on the decision are false.

A merge has multiple input flows and a single output. A merge marks the end of conditional behavior started by a decision. – In UML2 use only a single incoming and outgoing flow to an action.

Actions can be implemented either as subactivities or as methods on classes. You can show a subactivity by using the rake symbol.

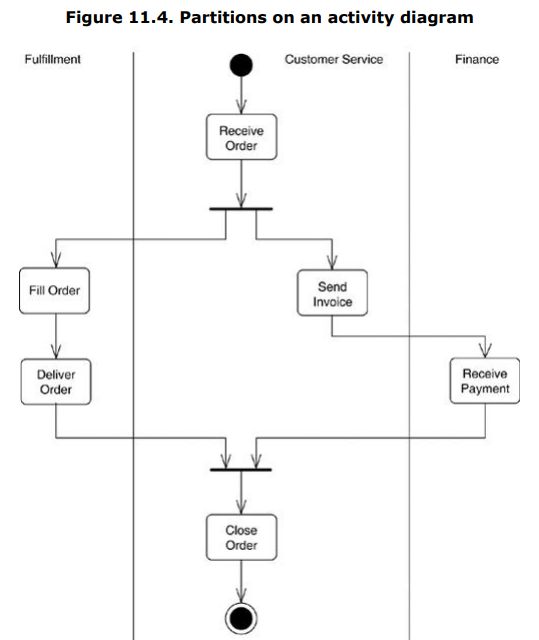


When to use Activity Diagrams:

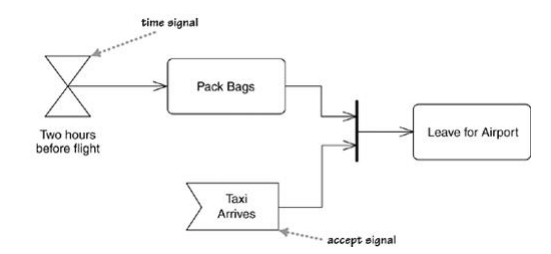
The great strength of activity diagrams lies in the fact that they support and encourage parallel behaviour

The main strength of doing this may come with people using UML as a programming language. In this case, activity diagrams represent an important technique to represent behavioral logic.

If you want to show who does what, you can divide an activity diagram into partitions, which show which actions one class or organization unit carries out:



A time signal occurs because of the passage of time. A signal indicates that the activity receives an event from an outside process.



A flow final indicates the end of one particular flow, without terminating the whole activity. Allows sections to act as filters, destroying rejected tokens. The rest of the activity can continue

