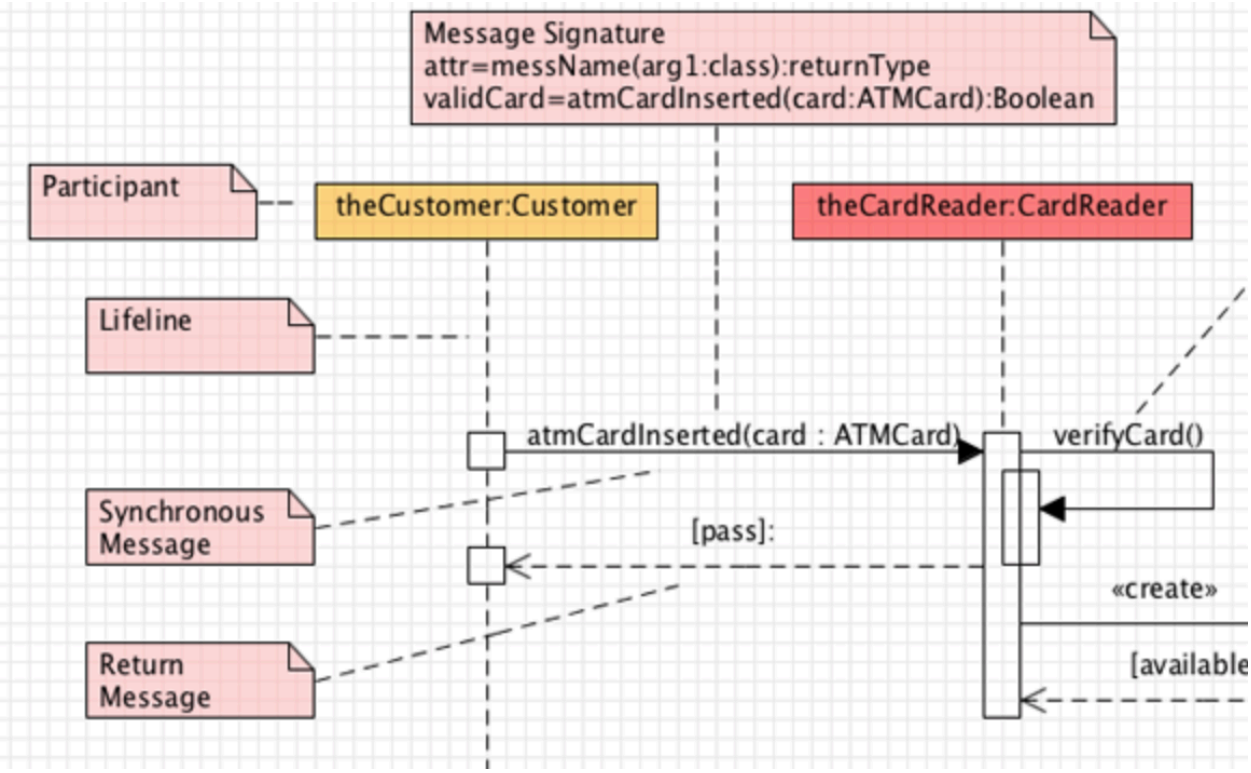
These notes are intended to give you additional information that is not included in the discussion slides - this includes stuff I wrote down on the board along with stuff I may have skipped (because I figured I had more important things to cover).

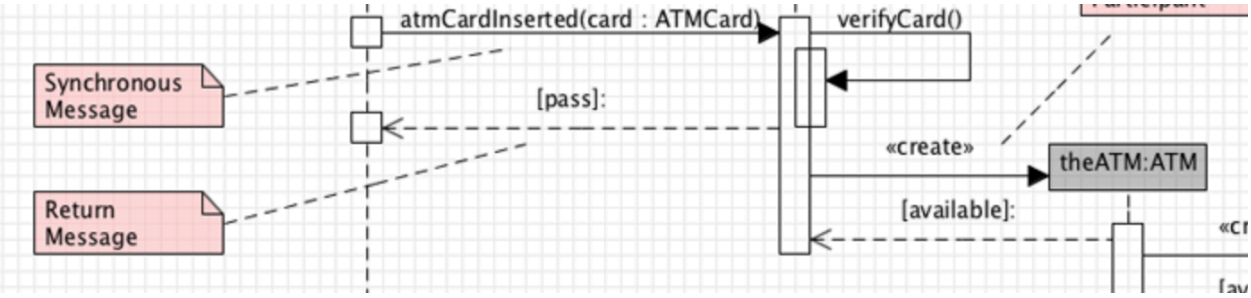
Sequence diagram

Models interactions in a program and shows the logical view of that. Focuses on showing the order of interactions between different parts of your program. In sequence diagrams, a *message* is an event that is sent from the caller to the receiver.

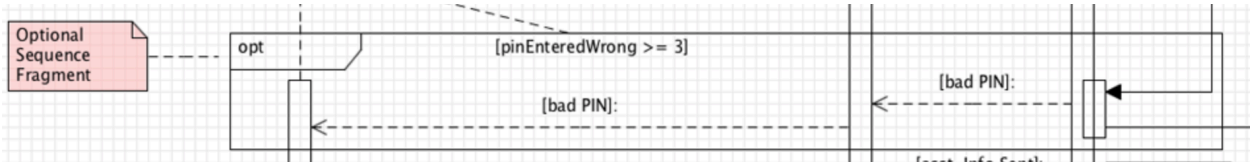


Asynchronus messages are those where the caller keeps sending messages without waiting for a response. They do not have a filled in arrow.

<<create>> message is used to create a new participant. <<delete>> message is used to destroy a participant.

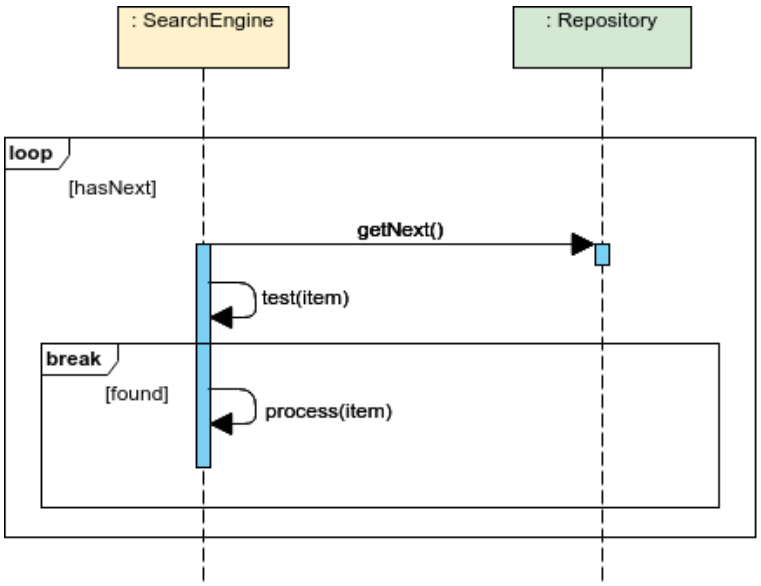


Sequence fragments are denoted by boxes that surround a small group of interactions. The top left corner contains the type of fragment. **Opt (Optional)** is only activated with a guard condition.

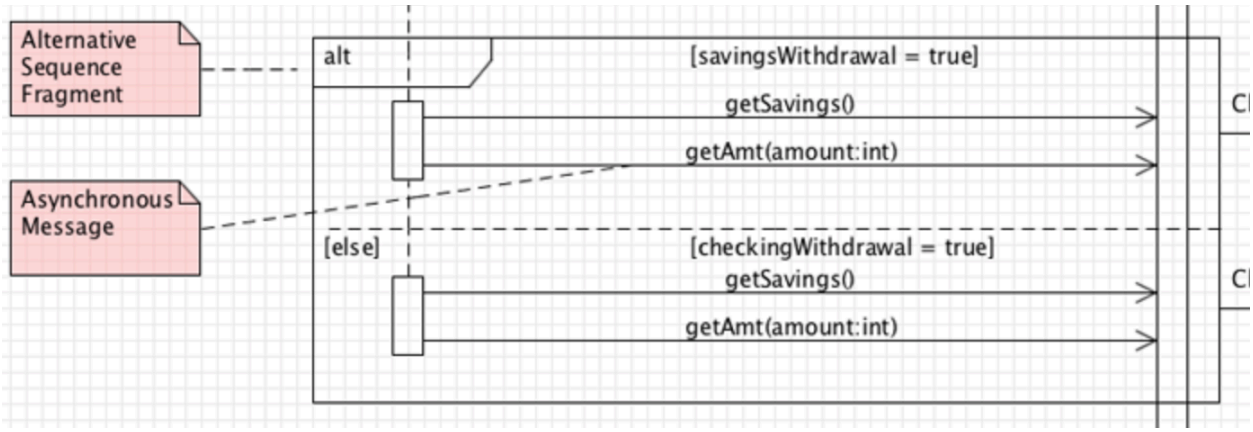


Ref (Reference) points to a different sequence fragment to deal with less space and decrease code duplication.

Loop is a sequence fragment that continues to execute the interactions until the guard condition is false.

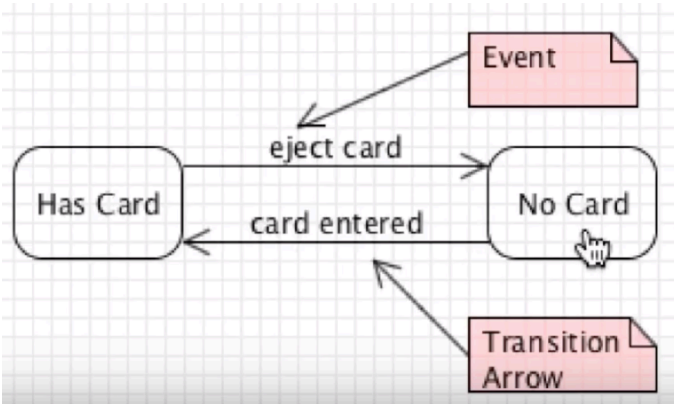


Alt (Alternative) is an if-else sequence fragment with a corresponding guard condition.

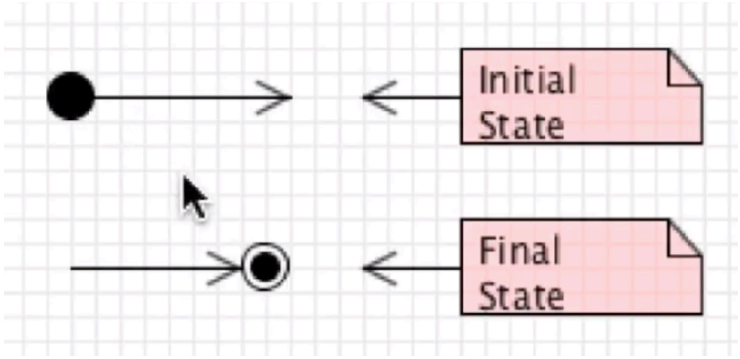


Statechart diagram

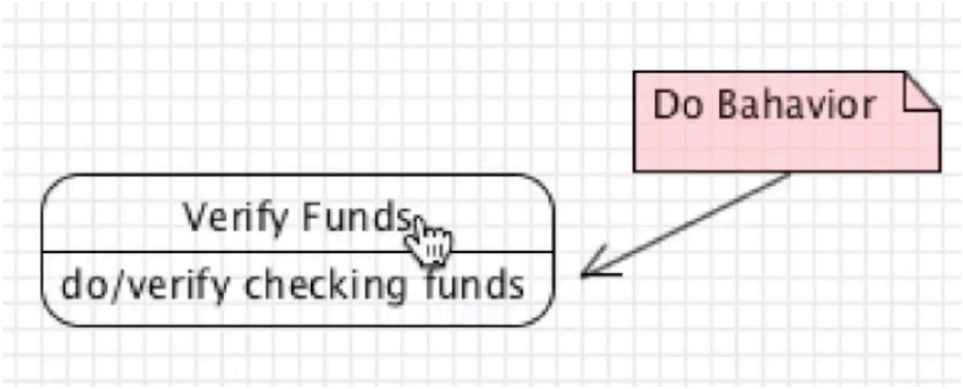
Models the changing states of objects and events that cause these state changes.



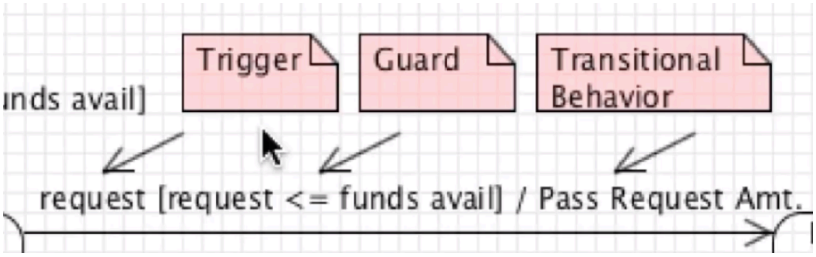
They have an initial and a final state depicted like so:



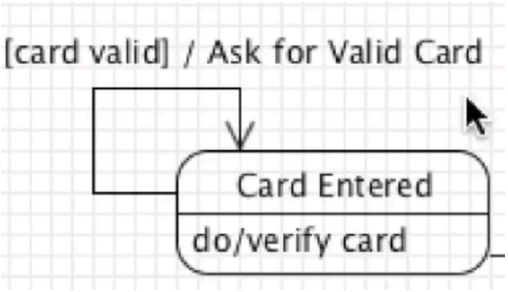
You may also document what's going on in your system by using the *do* keyword:



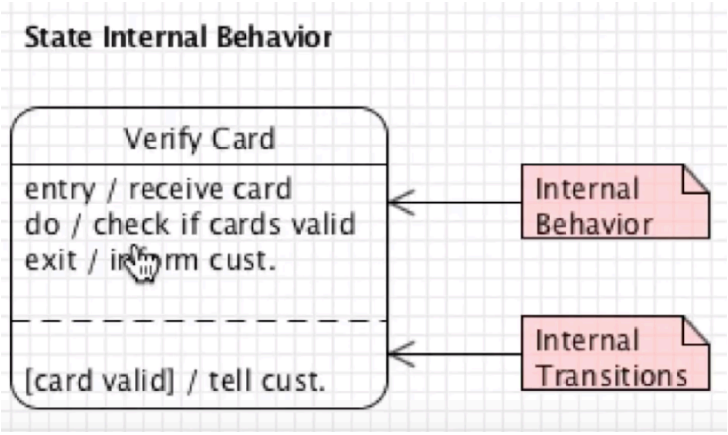
It's also useful to state a guard condition for transitions along with transitional behavior which may indicate message passing.



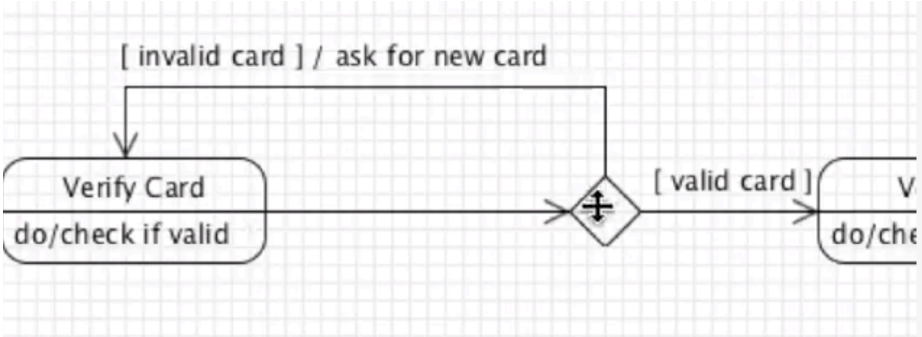
A small example to verify a card. Here if the card isn't valid, we return to the same state.



State internal behavior gives a flow-chart style logical description of what goes on within a state including the exit transition.

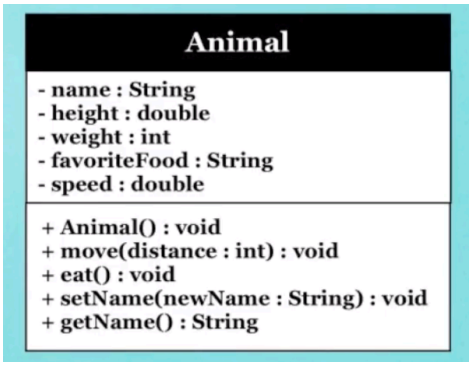


Here's how you will model boolean conditions:

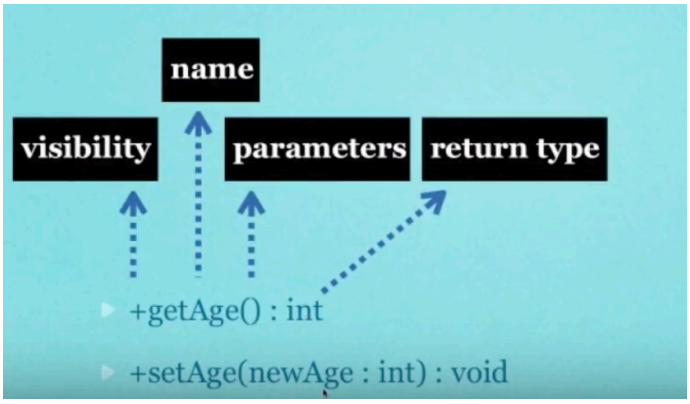


Class diagram

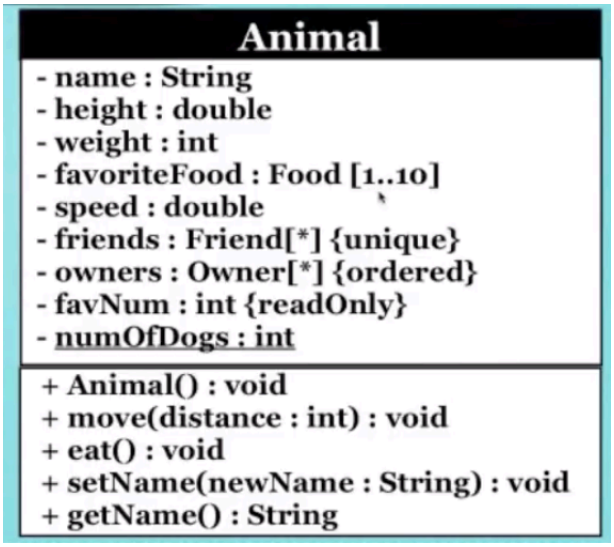
Refer to the slides it's all there. But here are some more examples:



Here's the syntax for methods:

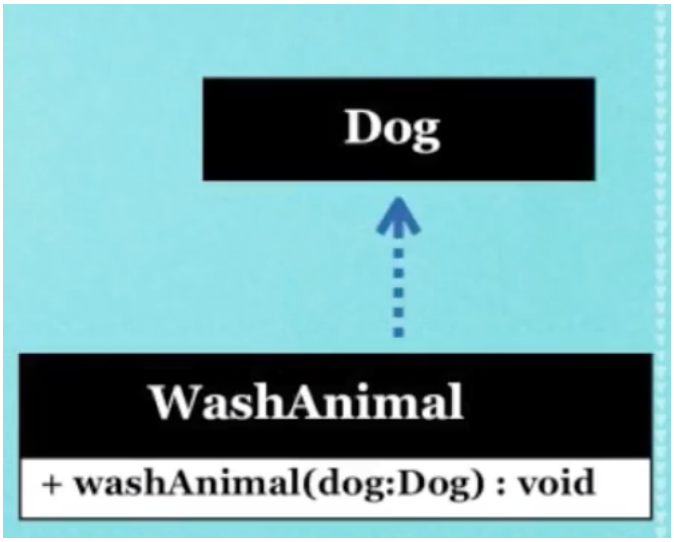


Here's what you would do for multiplicity:



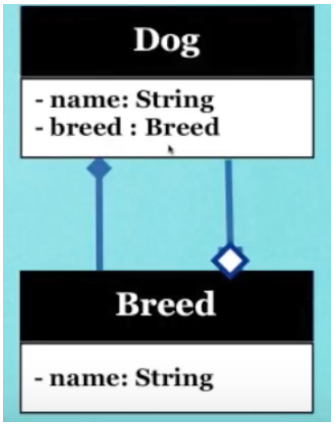
The underlined attribute is *static.*

Dependence - loosely coupled:



WashAnimal uses Dog.

Aggregation & Composition:



Constraints: