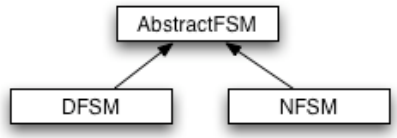
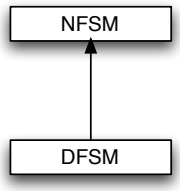
Topic

- ☐ 7th May 2007
  - ☐ Finite State Machines revisited
    - ☐ Adding a class for DFSMs
      - ☐ as sibling of NFSM, with new abstract superclass
        You can use the refactoring browser to help create the sibling class. It's important to look at each method in NFSM, and decided if it should be promoted to the superclass, or whether a new (and different) version should be created in the DFSM subclass



      - ☐ as subclass of NFSM. DFSMs are, after all, a specialization of NFSMs. I don't think that there is anything wrong with this, even "purists" would chose to pursue the above alternative. However, if you pursue this one, it's really important to cancel (= override with self shouldNotImplement) any methods that are not appropriate for the subclass.



    - ☐ Simple and clean way of writing the DFSM>>newFrom: aNFSM method
      - ☐ Can't really disregard efficiency when we changing the "big O" runtime.
  - ☐ Adding visualization to State Machines
    - ☐ As far as possible, don't change the code of the machine itself
      - ☐ Separate out the *View* — what's show on the GUI, from
      - ☐ the *Model* — the underlying logic of the FSM.
    - ☐ Since we have a working model, don't change it.
      - ☐ but we may have to refactor it a bit …
    - ☐ Add a class to view the whole machine, a class to view the states, and a class to view the transitions
      - ☐ Some of these can be existing views (Morphs)
    - ☐ Use the *Observer pattern* to notify views of changes in their model
      - ☐ Make each View object a *dependent* of the corresponding model object
        - ☐ modelObject **addDependent:** viewObject
      - ☐ Model objects send **self changed** when something interesting changes
      - ☐ view objects get an **update:** message
        - ☐ They can then ask their model for whatever they need to know
    - ☐ Break up **accept:** into pieces
      - ☐ so that each step can be visualized.