DUML translator design

Read Chapter 6
There are four modes in the DUML translator

- **Start**
- **Looking for a start tag**
- **Followed by W/S**
  - **Tag char after W/S**
  - **Possible Tag?**
    - **Followed by non-W/S**
  - **Matching tag char after non-W/S**
  - **Looking for an end tag**
    - **Followed by W/S**
What if we replace the switch with polymorphism?

- Remember the last approach using a switch statement?
- It wasn’t very OO—instead, it is a good “procedural” solution (you could do it in Pascal or C)
- Instead of using a separate `method` for each state, and then using a `switch statement` to control which one we execute ...
- We could use a separate `object` to represent each state, each with a different implementation of a `translateCharacter()` method ...
- ... And then use `polymorphism` to take the correct action based on the “current state”
First, imagine an interface for the “state” classes
Next, imagine a starting state
Next, let’s add a state for possible start tags
Next, let’s add a state looking for end tags
Next, let’s add a state for possible end tags
Finally, add a main class