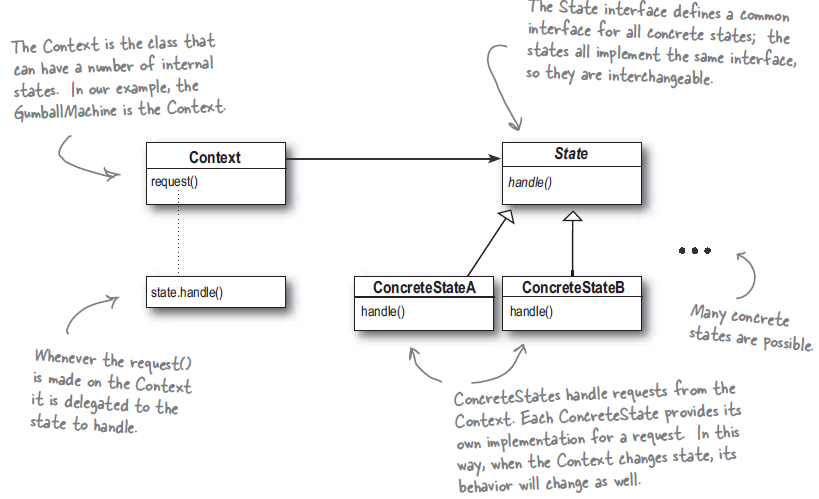
**The State Pattern:**

1. **Definition**: The State Pattern allows object to alter its behavior when its internal state changes. The object will appear to change its state.
2. **Class diagram**:



1. **Design notes**:
   1. State pattern contains a state interface that contains method for every action on the context.
   2. Every state should have concrete implementation of State interface. In this way Context class is closed for state modification but open to extension by adding new state. Also we’re localizing any changes needed in state itself.
   3. Context class should delegate work to the current state class and current state class handle the state transition. Context gets its behavior by delegating to current state object.
   4. When state transitions are fixed we can put the state transition logic in context otherwise state class should decide on next state.
   5. State interface can be a java interface or abstract class [if we want to specify a default behavior].
   6. States should be clearly separated out and put into their classes. That way we’re promoting One Class One Responsibility principle. It may increase no of concrete classes but its required for clarity.
2. Strategy vs State:
   1. Strategy is alternative to subclassing. With strategy we can change behavior of the object by composing with different object.
   2. State is alternative putting lot of conditions in the context class. By encapsulating behaviors in state object we can simply change the state object in context to change its behavior.
3. Java: javax.faces.lifecycle.LifeCycle#execute() (controlled by FacesServlet, the behavior is dependent on current phase (state) of JSF lifecycle)