State
State Intent

Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

(Behavioral)
The intent for the State pattern does not clearly specify the key (only?) indicator for the pattern.

If the behavior of an object is defined by a finite state machine then the State pattern is appropriate. No state machine \(\Rightarrow\) no need for this pattern.

Intent refers to internal state of object, i.e. object state is usually not visible to external client.

Object behavior is defined by state machines more often than you probably realize. The state machine is often obfuscated in convoluted logic.

Using a finite state machine to define behavior is a very important skill to develop even if you do not use the State design pattern to implement it. It provides a clear description of behavior.
A finite state machine consists of states, transitions, events, and actions.

- The object can exist in only one state at a time.
- In each state, the object’s behavior is different. Each method in the object may operate differently in each state.
- In any state, there is a set of events that can trigger the object to change to a new state. During the transition, the object may perform actions.
- The UML Statechart has even richer semantics defined (entry/exit actions, guard conditions, timeouts, composite states, orthogonal/concurrent states)
Every method with state-dependent behavior will need to conditionally decide what to do.

- Turn conditional into polymorphism.
- Separate the concerns for all state-dependent behavior into a class for each state.
- Delegate method execution to the state object. A state change becomes a delegate change.
The State pattern isolates the concerns of each state in state-specific objects.
There are many issues to consider when implementing a State pattern.

- How do you define state changes?
  - *Naïve approach* – explicitly code all of them
  - *Transition table* – state x event array

- Who has responsibility for state changes?
  - *ConcreteStates* – Easier to extend. Do they have the information needed? States are coupled.
  - *Context* – easier to understand

- When are ConcreteState objects constructed?
  - Create all states at construction of Context
  - Create/destroy states as needed at runtime
  - Can ConcreteState objects be shared by multiple Context instances?

- You need to have an event mechanism.