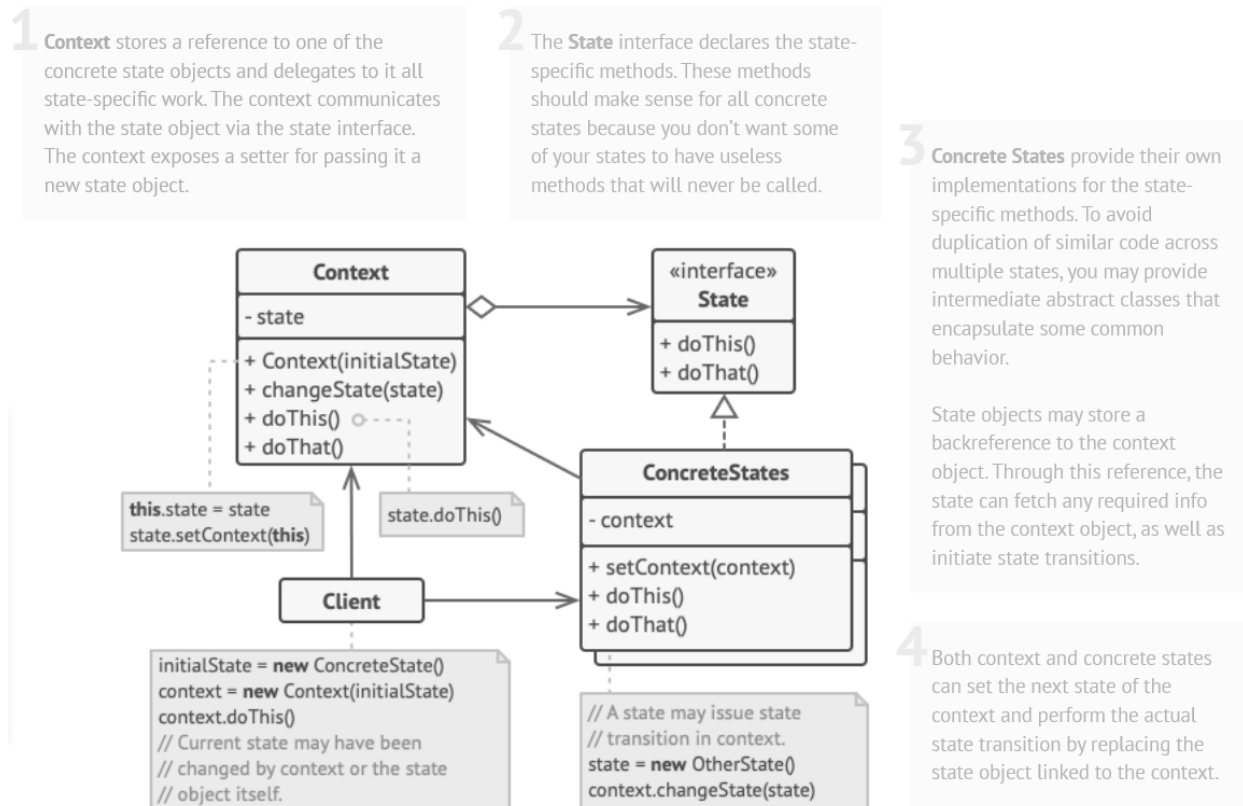# Intent

State is a behavioral design pattern that lets an object alter its behavior when its internal state changes. It appears as if the object changed its class.

Structure

1 **Context** stores a reference to one of the concrete state objects and delegates to it all state-specific work. The context communicates with the state object via the state interface. The context exposes a setter for passing it a new state object.

2 The **State** interface declares the state-specific methods. These methods should make sense for all concrete states because you don't want some of your states to have useless methods that will never be called.

3 **Concrete States** provide their own implementations for the state-specific methods. To avoid duplication of similar code across multiple states, you may provide intermediate abstract classes that encapsulate some common behavior.

State objects may store a backreference to the context object. Through this reference, the state can fetch any required info from the context object, as well as initiate state transitions.

4 Both context and concrete states can set the next state of the context and perform the actual state transition by replacing the state object linked to the context.


```
Context
- state
+ Context(initialState)
+ changeState(state)
+ doThis()
+ doThat()
```

```
«interface»
State
+ doThis()
+ doThat()
```

```
ConcreteStates
- context
+ setContext(context)
+ doThis()
+ doThat()
```

this.state = state
state.setContext(this)

state.doThis()

Client

initialState = **new** ConcreteState()
context = **new** Context(initialState)
context.doThis()
// Current state may have been
// changed by context or the state
// object itself.

// A state may issue state
// transition in context.
state = **new** OtherState()
context.changeState(state)

Applicability

• Use the State pattern when you have an object that behaves differently depending on its current state, the number of states is enormous, and the state-specific code changes frequently.

• Use the pattern when you have a class polluted with massive conditionals that alter how the class behaves according to the current values of the class's fields.

• Use State when you have a lot of duplicate code across similar states and transitions of a condition-based state machine.