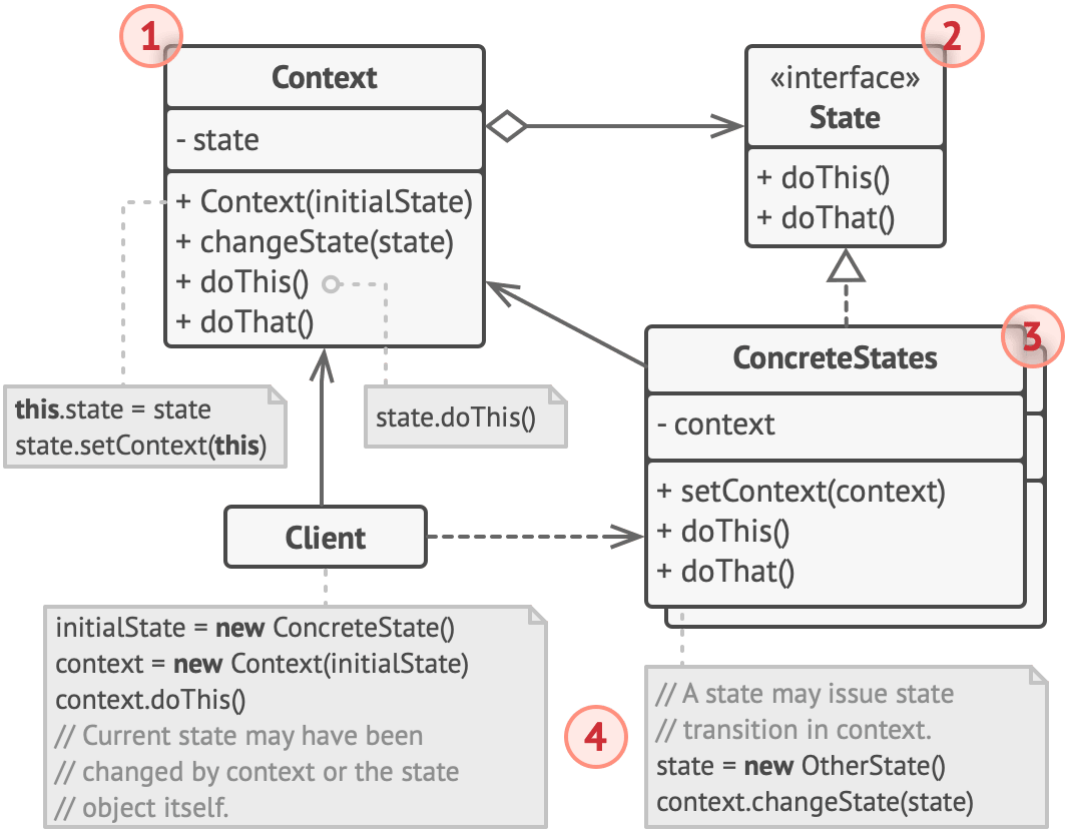
State pattern, the particular states may be aware of each other and initiate transitions from one state to another, whereas strategies almost never know about each other.

Real-World Analogy

The buttons and switches in your smartphone behave differently depending on the current state of the device:

- When the phone is unlocked, pressing buttons leads to executing various functions.
- When the phone is locked, pressing any button leads to the unlock screen.
- When the phone's charge is low, pressing any button shows the charging screen.

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.

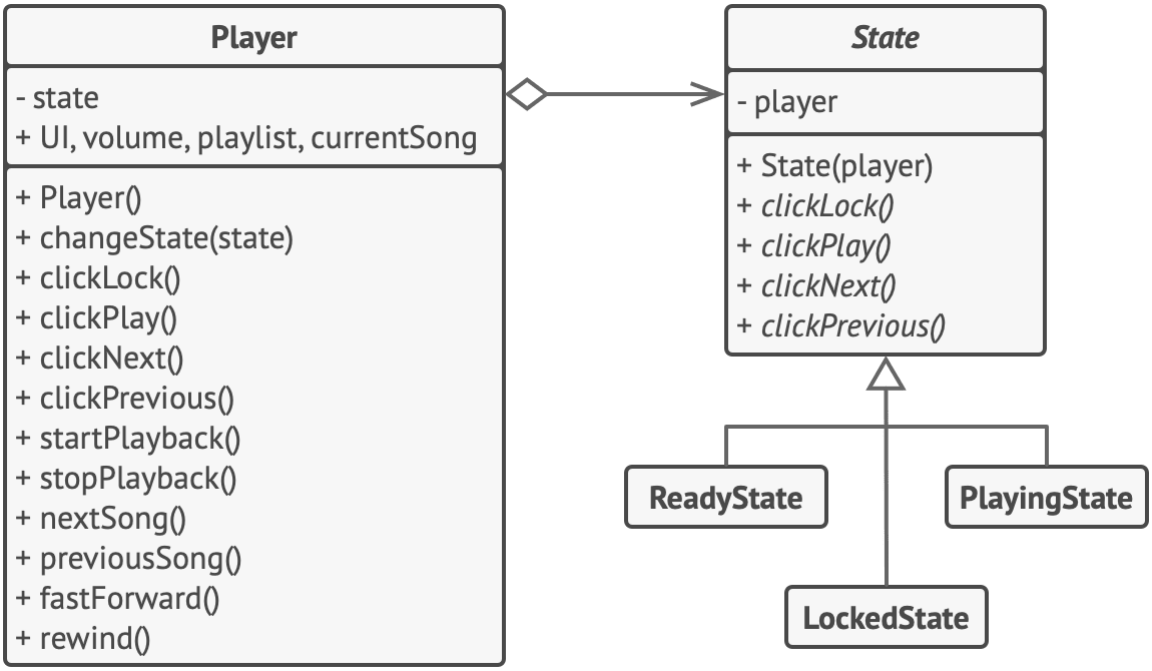
3. **Concrete States** provide their own implementations for the state-specific methods. To avoid duplication of similar code across multiple states, you may provide intermediate abstract classes that encapsulate some common behavior.

   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.

Pseudocode

In this example, the **State** pattern lets the same controls of the media player behave differently, depending on the current playback state.



Example of changing object behavior with state objects.

The main object of the player is always linked to a state object that performs most of the work for the player. Some actions replace the current state object of the player with another, which changes the way the player reacts to user interactions.