**Name**: State

**Category**: Behavioral

**Description**: A state allows an object to have different behaviors at different times. The state pattern strongly resembles a Finite-State Machine as there are a finite number of states and a state can only transition to a state that it is programmed to go to. This transition can happen when something specific happens in that state. Also, when in a specific state, the object may have options that the other states may not have. This method is used a lot in modern software for several reasons. To put this into perspective, imagine launching a game. The game starts in the state of “title\_screen”, and this state can go to the “play” state after the user clicks the start button. The user can also use the other options available in the “title\_screen” state such as the credits. After clicking start, the user loses all the options while the game was in the “title\_screen” state, but now has the options to move around in game, use items, etc. This method also is used for how OS’s work, such as the separation between the login screen and the rest of the functionalities.

**When to Use**: This method should be used if an object needs to have specific behavior at specific times. For example, music players will change their state to control the music, whether it is to skip, rewind, pause, or play the audio. The transition conditions for these states are whether the user clicks the corresponding button to activate the state.

**Advantages**:

* Allows for an object to be able to do different methods depending on its state.
* It is simple to add new states without causing problems with other states.

**Disadvantages**:

* Improper implementation of this pattern can clutter up the code.