Definition:

|  |
| --- |
| * An object's behavior depends on its state, and it must change its behavior at run-time depending on that state. * Operations have large, multipart conditional statements that depend on the object's state. This state is usually represented by one or more enumerated constants. Often, several operations will contain this same conditional structure. The State pattern puts each branch of the conditional in a separate class. This lets you treat the object's state as an object in its own right that can vary independently from other objects. |

*The****state pattern****is a behavioural software design pattern that allows an object to alter its behavioural  when its internal state changes.*

When to Use :

* An object's behavior depends on its state, and it must change its behavior at run-time depending on that state.
* Operations have large, multipart conditional statements that depend on the object's state. This state is usually represented by one or more enumerated constants. Often, several operations will contain this same conditional structure. The State pattern puts each branch of the conditional in a separate class. This lets you treat the object's state as an object in its own right that can vary independently from other objects.

Realworld Example :

Architecture :

Example :

Advantage :

*Some observations on their advantages:*

By using the *State* pattern the state-holding (context) class is relieved from knowledge of **what** state or type it is and what states or types that are available. This means that the class adheres to the open-closed design principle (OCP): the class is closed for changes in what states/types there are, but the states/types are open to extensions.

By using the *Strategy* pattern the algorithm-using (context) class is relieved from knowledge of **how** to perform a certain task (-- the "algorithm"). This case also creates an adherence to the OCP; the class is closed for changes regarding how to perform this task, but the design is very open to additions of other algorithms for solving this task.  
This likely also improves the context class' adherence to the single responsibility principle (SRP). Further the algorithm becomes easily available for reuse by other classes.

Disadvantage :

The state pattern is also known as **objects for states**. So, we can assume that more states need more codes, and the obvious side effect is difficult maintenance for us.

Question :

1. Difference between State and Strategy pattern

* States store reference to context object that contains them. Strategies do not.
* States are allowed to replace themselves (change the state of context object to themselves) while strategies are not.
* States are created by the context object itself , Strategies are passed to the context object as parameters.
* State provide the underlying implementation for everything the contest object does., In Strategies only handle the single specific task.
* The *State* pattern deals with **what** (state or type) an object is (in) -- it encapsulates state-dependent behavior, whereas
* the *Strategy* pattern deals with **how** an object performs a certain task -- it encapsulates an algorithm.

In layman's language,

in Strategy pattern, there are no states or all of them have same state. All one have is different ways of performing a task, like different doctors treat same disease of same patient with same state in different ways.

In state Pattern, subjectively there are states, like patient's current state(say high temperature or low temp), based on which next course of action(medicine prescription) will be decided.And one state can lead to other state, so there is state to state dependency( composition technically).

1. State Objects to be singleton?

- Yes. Always try to make state objects [singletons](https://howtodoinjava.com/design-patterns/creational/singleton-design-pattern-in-java/).

**Design participants**

* **State** – The interface define operations which each state must handle.
* **Concrete States** – The classes which contain the state specific behavior.
* **Context** – Defines an interface to client to interact. It maintains references to concrete state object which may be used to define current state of object. It delegates state-specific behavior to different State objects.