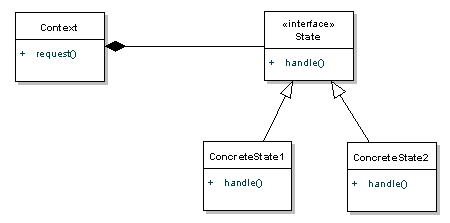
State is a pattern in which the object’s behavior is determined by its state. An object transitions from one state to another. A formalized construct which manages state and transitions is called state machine

**State** is a behavioral design pattern that allows an object to change the behavior when it’s internal state changes. Also, adding new states should not affect the behavior of existing states.

**GOF Definition**

Allows an object to alter its behavior when it’s internal state changes. The object will appear to change its class.

**Class Diagram**

**s**

The state pattern is set to solve two main problems:

* An object should change its behavior when it’s internal state changes.
* State-specific behavior should be defined independently. That is, adding new states should not affect the behavior of existing states.

**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.

#### State pattern vs strategy pattern

The structures of both patterns are similar, but the intents are different. The [strategy pattern](https://howtodoinjava.com/design-patterns/behavioral/strategy-design-pattern/) provides a better alternative to subclassing, while in state pattern – behavior is encapsulated in separate classes.

Yet both promote the composition and the delegation over inheritance.

Both design patterns are very similar, but their UML diagram is the same, with the idea behind them slightly different.

First, [strategy pattern](https://www.baeldung.com/java-strategy-pattern)**defines a family of interchangeable algorithms**. Generally, they achieve the same goal, but with a different implementation, for example, sorting or rendering algorithms.

**In state pattern, the behavior might change completely**, based on actual state.

Next, **in strategy, the client has to be aware of the possible strategies to use and change them explicitly,** whereas **in state pattern, each state is linked to another and create the flow as in Finite State Machine**.

#### State objects should be singletons?

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