State Design Pattern

Design Pattern research

1. **Intent of the State design pattern**

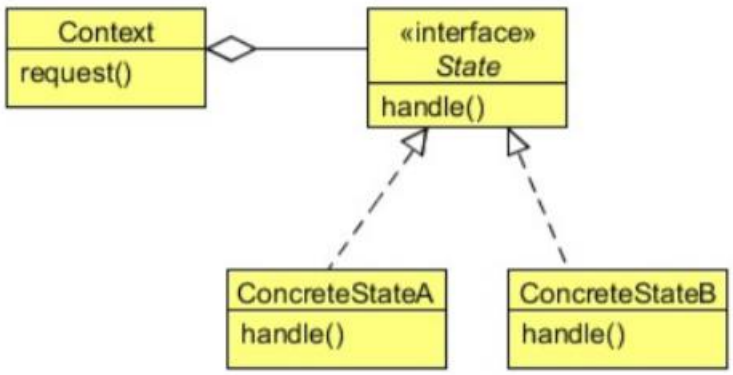
   - Allow an object to alter its behavior when its internal state changes
   - An object-oriented state machine implementation

2. **Explanation (more detailed)**

   - The State Design Pattern (behavioral) is used to model changes in the status or state of an object by delegating rules for such changes to individual objects representing each possible state
   - **SOLID Principles** which the DP supports:
     - ✓ Open-Closed principle
       State DP supports this principle, because it models the object's activity/protocol in a way which is opened for extension but closed for modification. If new specifications arise, the implementation can be extended with other ConcreteStates (ConcreteStateC, ConcreteStateD, …)
     - ✓ Single-Responsibility principle
       State DP helps this principle, because having multiple ConcreteState classes, each will be responsible only for that state's action (methods)
   - Three (3) components:

Context	State	ConcreteState subclasses
Defines the interface of interest to clients Maintains an instance of a *ConcreteState* subclass that defines the current state	Defines an interface for encapsulating the behavior associated with a particular state of the *Context*	Each subclass implements a behavior associated with a state of the *Context*

   - UML diagram/structure



   - Links to web sources for better explanatory:
     - https://sourcemaking.com/design_patterns/state
     - https://www.tutorialspoint.com/design_pattern/state_pattern.htm

3. Related patterns

- Strategy pattern
  The implementation of the State pattern builds on the Strategy pattern. The **difference** between State and Strategy is in the intent. With Strategy, the choice of algorithm is fairly stable. With State, a change in the state of the "context" object causes it to select from its "palette" of Strategy objects.
  One implementation of the Strategy pattern is similar with the State pattern.
  The **difference** is in binding times: strategy is a bind-once pattern, whereas State is more dynamic.

- Bridge pattern
  The structure of the Bridge pattern is almost identical with the structure of the Strategy pattern. The **difference** is that Bridge admits hierarchies of envelope classes, whereas State allows only one.
  The two patterns use the same structure to solve different problems: State allows an object's behavior to change along with its state, **while** Bridge's intent is to decouple an abstraction from its implementation so that the two can vary independently.

4. Common situations of use

Use the State pattern in either of the following cases:

- 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.

Examples of applications:

- This pattern can be observed in a **vending machine**. Vending machines have states based on the inventory, amount of currency deposited, the ability to make change, the item selected, etc. When currency is deposited and a selection is made, a vending machine will either deliver a product and no change, deliver a product and change, deliver no product due to insufficient currency on deposit, or deliver no product due to inventory depletion.

5. **Can be mistaken with**

   - **Strategy pattern**

Similarities

1. Class Diagrams of the Patterns
   Both patterns define a base state/strategy; sub-states/sub-strategies are children of the base class.
2. Both follow Open Closed Principle
3. Use of subclasses
   Both State & Strategy Pattern use subclasses to change the implementation via different derived implementations of States/Strategies.

Differences

1. Intent of the Pattern
   Intent or purpose of Strategy Pattern is to have a family of interchangeable algorithms which *can be chosen* based on the context and/or client needs. On the other hand, State Pattern's intent is *to manage states* of the object along with object's behavior which changes with its state.
2. Client Awareness of the strategy/state
   In a Strategy Pattern implementation the *strategy chosen is client-dependent* and hence the client is aware which strategy is being used. However, in State Pattern implementation *client interacts with the context to act on the object but does not decide on which State* to choose. The Object itself appears to change its State Class based on the interaction the Client has via the Context.
3. Reference back to the Context
   Every state in the State Pattern holds a reference back to the Context. However, each strategy does not hold the handle back to the Context in the Strategy Pattern.
4. Relation between individual states/individual strategies
   Different states in the State Pattern are related to one another, say as a successor or predecessor etc. This is because there is a *flow among the states like a Finite State Machine*. Strategy Pattern, however, just chooses *one of the strategies* from the multiple strategies available. There is no successor/predecessor relationship between strategies.
5. How vs. What & When
   Multiple strategies define multiple ways of *how* to do something. On the other hand, multiple states define *what* is to be done and based on the relationship between states *when* it is to be done.

Conclusion
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*.

The constructs for achieving these different goals are however very similar; both patterns are examples of composition with delegation.

6. Bibliography

Sources:

- Java Design Patterns, Reusable solutions to common problems, Rohit Joshi; http://enos.itcollege.ee/~jpoial/java/naited/Java-Design-Patterns.pdf
- Design Patterns, Explained simply; https://sourcemaking.com/design_patterns/state
- https://www.tutorialspoint.com/design_pattern/state_pattern.htm
- https://www.geeksforgeeks.org/state-design-pattern/
- https://en.wikipedia.org/wiki/State_pattern
- https://www.javabrahman.com/design-patterns/strategy-design-pattern-versus-state-design-pattern-analysis
- https://stackoverflow.com