State Design Pattern

Team WXYZ

Qing Xu (20511062)
Wenyu Zhu (20539246)
Kehan Wang (20521116)
Qiuyun Luo (20543229)

Introduction:

The state pattern is a design pattern that allow an object encapsulates multiple behaviors based on its internal state. In State pattern, “state objects” represent various states and a “context object” whose behavior varies as its state object changes. A State pattern is also considered as a way for an object to change its behavior at runtime.

Purpose and Motivation

The pattern deal with the interaction (state change) among objects to achieve large functionality. Also, a start pattern also allows an object to alter it behavior at runtime.

Intended use case

An object that its behavior is based on a function of its state, and it must change its behavior at run-time depending on that state.

Vocabulary

1. "Context" class: a single interface to the client. It also maintains an instance of the Concrete State subclass that internally defines the implementation of the object's particular state.
2. State abstract base class (wrapper): This is considered to be an interface that encapsulates the object's behavior. This behavior is associated with the state of the object.
3. Concrete class : This is a subclass that implements the state interface. Concrete State implements the actual behavior associated with the object's particular state.

Advantages:

1. An object's behavior is the result of the function of its state, and the behavior gets changed at runtime depending on the state. So the concrete state have little independency on each other.
2. With State pattern, the benefits of implementing polymorphic behavior are evident, and it is also easier to add states to support additional behavior.
3. The State design pattern also improves Cohesion since state-specific behaviors are aggregated into the abstract state class.
4. State pattern improves the flexibility to extend the behavior of the application and overall improves code maintenance.

Disadvantages:

1. Class Explosion: Since every state needs to be defined with the help of the "wrapper" class, there is a chance that we might end up writing many more classes with a small functionality. Consider the case of finite state machines—if there are many states but each state is not too different from another state, we'd still need to write them as separate classes. This increases the amount of code we need to write, and it becomes difficult to review the structure of a state machine.
2. With the introduction of every new behavior, the Context class needs to be updated to deal with each behavior. This makes the Context behavior more brittle with every new behavior.

NFPs (non-functional properties):

Support:

1. Scalability: easier to add states to support additional behavior.
2. Dependability: One state object fails, others still work well.

Inhibit:

Efficiency: do not support asynchronous interaction. State object with small functionality.

Coupling reduce:

State pattern can reduce coupling in the program by aggregating state-dependent behavior into the abstract state class and the states are relatively independent from each other.

Example:

The State Design Pattern allows an object to change its behavior when its internal state changes. Vending machines have states based on the inventory, amount of currency deposited, the ability to make change, the item selected. It has four states: Idel, InsufficientDeposit, SufficientDeposit, Selected. In the state Idle, vending machine will wait for a customer to activate it. In the state InsufficientDeposit, vending machine will wait for enough deposit. Once it has enough deposit, vending machine goes to the next state, SufficientDeposit, which is waiting for user to select item. Once user selected item, vending machine goes to state Selected, delivers product and gives changes to user, and then turns back to Idle state.