WIKIPEDIA

State pattern

The **state pattern** is a behavioral software design pattern that allows an object to alter its behavior when its internal state changes. This pattern is close to the concept of finite-state machines. The state pattern can be interpreted as a strategy pattern, which is able to switch a strategy through invocations of methods defined in the pattern's interface.

The state pattern is used in computer programming to encapsulate varying behavior for the same object, based on its internal state. This can be a cleaner way for an object to change its behavior at runtime without resorting to conditional statements and thus improve maintainability.[1]:395

Contents

Overview

The state design pattern is one of the twenty-three design patterns designed by the Gang of Four that describe how to solve recurring design problems. Such problems cover the design of flexible and reusable object-oriented software, such as objects that are easy to implement, change, test, and reuse.[2]

The state pattern is set to solve two main problems:[3]

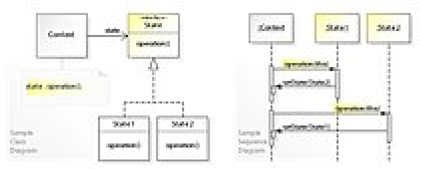
- An object should change its behavior when its internal state changes.
- State-specific behavior should be defined independently. That is, adding new states should not affect the behavior of existing states.

Implementing state-specific behavior directly within a class is inflexible because it commits the class to a particular behavior and makes it impossible to add a new state or change the behavior of an existing state later independently from (without changing) the class. In this, the pattern describes two solutions:

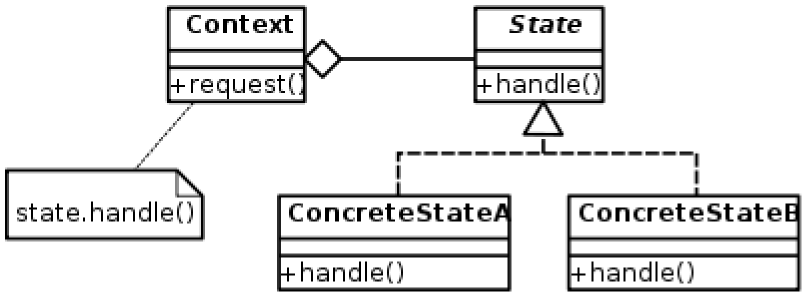
- Define separate (state) objects that encapsulate state-specific behavior for each state. That is, define an interface (state) for performing state-specific behavior, and define classes that implement the interface for each state.
- A class delegates state-specific behavior to its current state object instead of implementing state-specific behavior directly.

This makes a class independent of how state-specific behavior is implemented. New states can be added by defining new state classes. A class can change its behavior at run-time by changing its current state object.

Structure


A sample UML class and sequence diagram for the State design pattern.[4]

In the above Unified Modeling Language (UML) class diagram, the `Context` class doesn't implement state-specific behavior directly. Instead, `Context` refers to the `State` interface for performing state-specific behavior (`state.operation()`), which makes `Context` independent of how state-specific behavior is implemented. The `State1` and `State2` classes implement the `State` interface, that is, implement (encapsulate) the state-specific behavior for each state. The UML sequence diagram shows the run-time interactions:



State in UML[1]

The `Context` object delegates state-specific behavior to different `State` objects. First, `Context` calls `operation(this)` on its current (initial) state object (`State1`), which performs the operation and calls `setState(State2)` on `Context` to change context's current state to `State2`. The next time, `Context` again calls `operation(this)` on its current state object (`State2`), which performs the operation and changes context's current state to `State1`.

Example

Java

The state interface and two implementations. The state's method has a reference to the context object and is able to change its state.

```java
interface State {
    void writeName(StateContext context, String name);
}

class LowerCaseState implements State {
    @Override
    public void writeName(StateContext context, String name) {
        System.out.println(name.toLowerCase());
        context.setState(new MultipleUpperCaseState());
    }
}

class MultipleUpperCaseState implements State {
    /* Counter local to this state */
    private int count = 0;

    @Override
    public void writeName(StateContext context, String name) {
        System.out.println(name.toUpperCase());
        /* Change state after StateMultipleUpperCase's writeName() gets invoked twice */
        if(++count > 1) {
            context.setState(new LowerCaseState());
        }
    }
}
```

The context class has a state variable that it instantiates in an initial state, in this case `LowerCaseState`. In its method, it uses the corresponding methods of the state object.

```
class StateContext {
    private State state;

    public StateContext() {
        state = new LowerCaseState();
    }

    /**
     * Set the current state.
     * Normally only called by classes implementing the State interface.
     * @param newState the new state of this context
     */
    void setState(State newState) {
        state = newState;
    }

    public void writeName(String name) {
        state.writeName(this, name);
    }
}
```

The demonstration below shows the usage:

```
public class StateDemo {
    public static void main(String[] args) {
        var context = new StateContext();

        context.writeName("Monday");
        context.writeName("Tuesday");
        context.writeName("Wednesday");
        context.writeName("Thursday");
        context.writeName("Friday");
        context.writeName("Saturday");
        context.writeName("Sunday");
    }
}
```

With the above code, the output of `main()` from `StateDemo` is:

```
monday
TUESDAY
WEDNESDAY
thursday
FRIDAY
SATURDAY
sunday
```

References

1. Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. ISBN 0-201-63361-2.
2. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley. pp. 305ff. ISBN 0-201-63361-2.
3. "The State design pattern - Problem, Solution, and Applicability" (http://w3sdesign.com/?gr=b08&ugr=proble). *w3sDesign.com*. Retrieved 2017-08-12.
4. "The State design pattern – Structure and Collaboration" (http://w3sdesign.com/?gr=b08&ugr=struct). *w3sDesign.com*. Retrieved 2017-08-12.

Retrieved from "https://en.wikipedia.org/w/index.php?title=State_pattern&oldid=888177646"