참고 자료

- https://refactoring.guru/design-patterns/state
- https://en.wikipedia.org/wiki/State_pattern

Intent

분류

- behavioral design pattern
  - object의 behavior을 캡슐화하고 요청을 object에 위임하는 패턴

정의

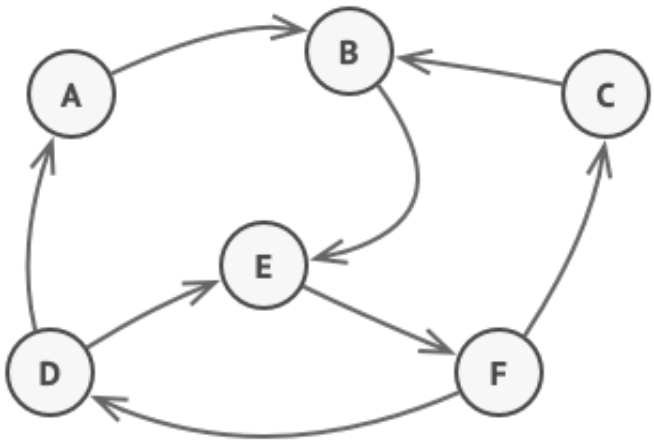
- object의 상태가 변할 때마다 행동을 바꾸도록 하는 디자인 패턴입니다.
  - 이 때문에 object의 class가 변경되는 것 처럼 보입니다.

Problem

상황 - 유한 상태 기계 (finite state machine)

State Pattern은 유한 상태 기계의 개념과 아주 밀접한 관련이 있습니다.

따라서 유한 상태 기계에 대해 먼저 설명드리겠습니다.



- 프로그램은 어떤 특정한 순간에(any given moment) 특정한 상태를 가집니다.
- 그 상태의 수는 유한합니다.
- 특정 상태에서 프로그램은 다르게 행동합니다.
- 프로그램은 한 상태에서 다른 상태로 즉시 변경될 수 있습니다.

- 현재 상태에 따라 변경될 수 있는 상태가 제한되어 있습니다. 또는 변경될 수 있는 상태가 없을 수 있습니다.

  이러한 switching 규칙을 전이(transition)라 부릅니다.
- transitions 또한 유한하며, 미리 정의되어 있습니다.

이러한 접근을 object에 적용할 수 있습니다. 그래서 이러한 접근을 적용한 object인 document프로그램을 소개해 드리겠습니다.

Document program

document클래스의 상태

1. Draft
2. Moderation
3. Published

document클래스의 method: publish

publish method는 각 상태별로 다르게 작동합니다.

1. Draft

   document의 상태를 moderation으로 전환합니다.
2. Moderation

   현 유저가 관리자일 때만 document를 public으로 바꿉니다.
3. Published

   아무것도 하지 않습니다.

안 좋은 접근

1. 객체의 필드에서 상태를 단순히 값의 집합으로 나타냅니다.
2. 조건문을 통해 현재 상태에 따라 다르게 행동하도록 합니다.

```
class Document is
    field state: string
    // ...
    method publish() is
        switch (state)
            "draft":
                state = "moderation"
                break
            "moderation":
                if (currentUser.role == 'admin')
                    state = "published"
                break
            "published":
                // Do nothing.
                break
    // ...
```

단점

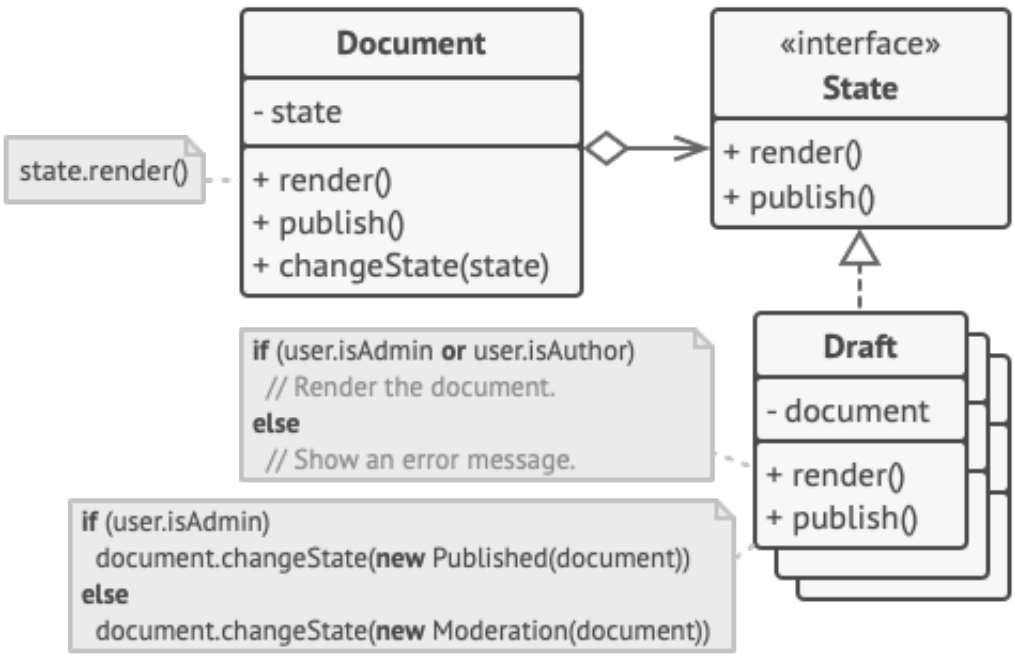
1. 상태와, 상태에 따른 행동을 추가시 모든 메서드 로직을 변경해야 할 수 있습니다.
2. 매우 복잡해집니다.

Solution

핵심

state들을 value가 아닌 class들로 각각 선언합니다. 그리고 state에 관한 behavior은 state에게 위임하는 것이지요.

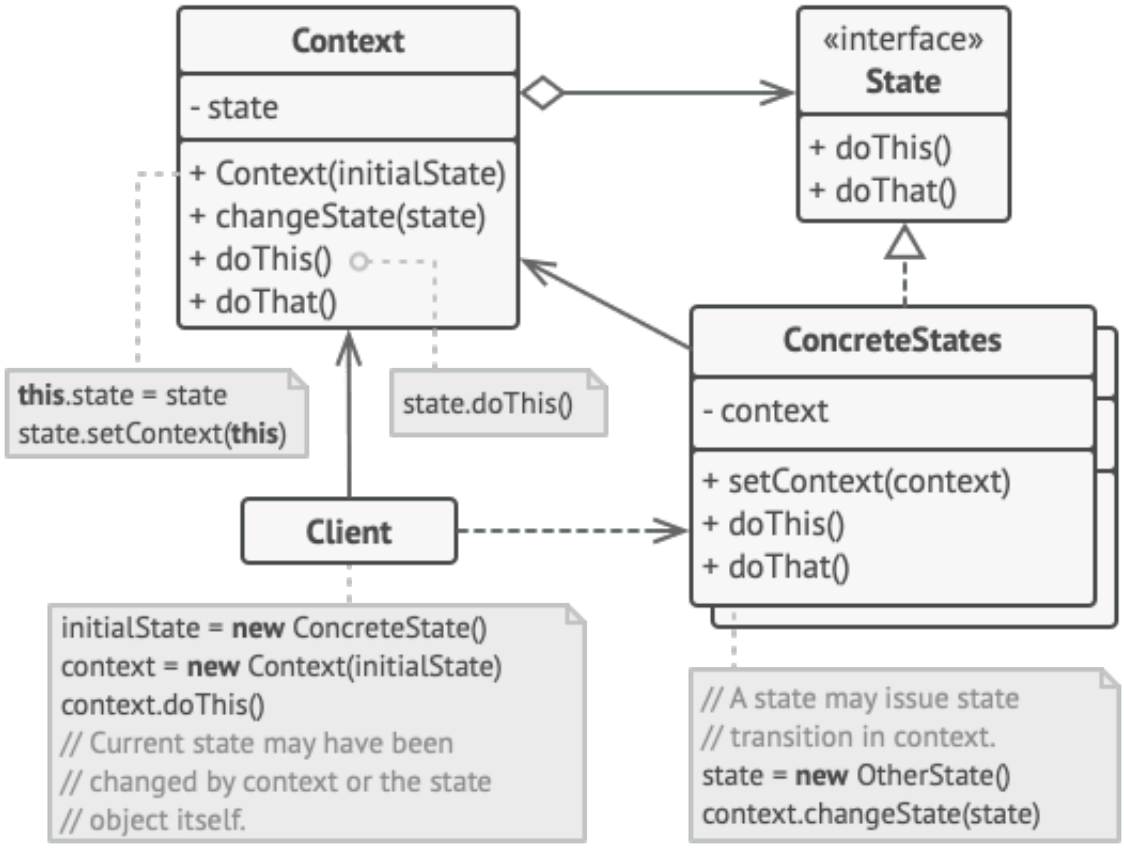
적용



1. Document객체를 context 또는 origin object라 부르겠습니다. 이 객체는 현재 상태를 나타내주는 state objects를 reference로 저장합니다. 그리고 상태에 관련된 일을 이 state object에게 넘겨줍니다.
2. 상태를 전이하기 위해선, 현재 상태의 객체를 다른 객체로 바꿔줍니다.

   이 작업에서 특정 state에 종속되면 안되니 상태 객체들을 추상화한 interface를 선언합니다.

구조



1.Context

- concrete state objects를 reference로 둡니다.
- 상태에 관한 작업을 concrete state objects에게 넘깁니다.
- state interface를 통해 concrete state와 소통합니다.
- 새로운 상태 object를 설정하기 위해 setter를 public으로 선언합니다.

2. State

- 상태에 특정한 method를 선언합니다. (state-specific method)

3. Concrete States

- 자기 상태만의 로직을 수행합니다.
- context에 대한 back reference를 둡니다.
    - context object에게 데이터를 얻고, state를 전이시키기 위함입니다.

예제 - wiki

context의 상태에 따라 대문자가, 소문자가 출력되는 예제입니다.

초기 상태는 소문자 출력상태이며, 소문자 출력시 대문자 출력하는 상태로 변환되고, 대문자 출력 상태에서 2번 출력시 소문자 출력 상태로 변환됩니다.

그리고 이 예제에선 state method에 context object를 전달함으로써 context를 조작할 수 있게 하였습니다.

1. State

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

2. LoswerCaseState

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

3. 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());
        }
    }
}
```

4. StateContext

```
class StateContext {
    private State state;

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

    void setState(State newState) {
        state = newState;
    }

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

5. StateDemo

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

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

출력 결과

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