DESIGN PATTERNS: **STATE**

Behavioral Pattern

1. Design Pattern Description

Allow an object to alter its behavior when its internal state changes at run-time.

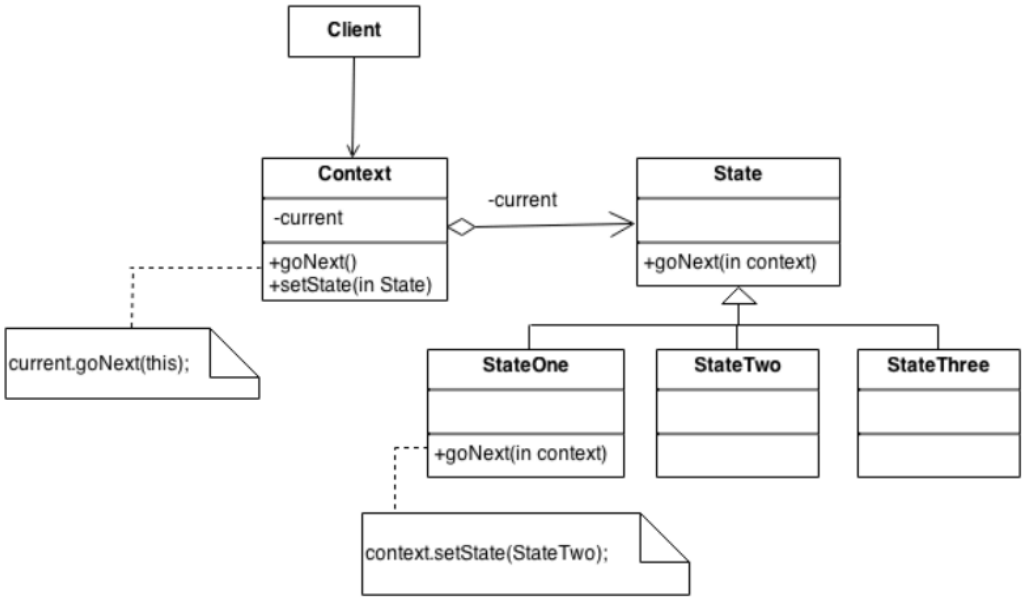
`The object will appear to change its class.`

- Context Class or Wrapper: → **single** interface to the world
- State Abstract + Derived Concrete State classes
- State-specific behavior defined in Concrete State classes
- "state" pointer in context class. Change it to change behavior.

Where to specify the state transitions? → 2 choices:

a) In the "**context" object**

b) In each **Concrete State class → ease adding new states**, but each concrete State class has knowledge about its siblings (**coupling**)



Other considerations

- Often **Singletons**. → I guess this applies specially when switching **"strategies"** than need to be initialized.
- **Flyweight** explains when and how State objects can be shared.

- **Interpreter** can use State to define parsing contexts.
- **Strategy** has 2 different implementations, the first is similar to State. The difference is in binding times (Strategy is a bind-once pattern, whereas State is more dynamic).
- Identical structure than **Bridge**, but State does not allow the wrapper to have hierarchy. But they 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.
- 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.

2. Design Pattern Example

Vending Machine

After depositing currency and selecting product, the vending machine can have the following **States**:

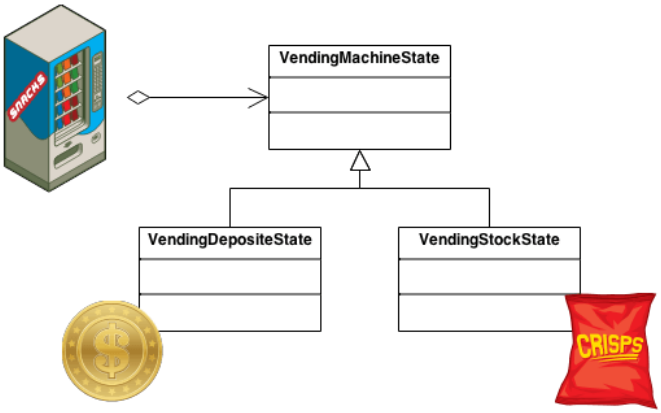
a) Deposit | Stock → Product & NO change

b) Deposit | Stock → Product & change

→ DeliverProduct()
ReturnChange() → with particular case of NO change

c) Deposit | Stock → No product (insufficient currency on deposit)

d) Deposit | Stock → No product (insufficient stock on inventory)



3-Speed Ceiling Fan state machine

```
interface State {
    void pull(CeilingFanPullChain wrapper);
}

class CeilingFanPullChain {
    private State currentState;

    public CeilingFanPullChain() {
        currentState = new Off();
    }

    public void set_state(State s) {
        currentState = s;
    }

    public void pull() {
        currentState.pull(this);
    }
}

class Off implements State {
    public void pull(CeilingFanPullChain wrapper) {
        wrapper.set_state(new Low());
        System.out.println("low speed");
    }
}

class Low implements State {
    public void pull(CeilingFanPullChain wrapper) {
        wrapper.set_state(new Medium());
        System.out.println("medium speed");
    }
}

class Medium implements State {
    public void pull(CeilingFanPullChain wrapper) {
        wrapper.set_state(new High());
        System.out.println("high speed");
    }
}

class High implements State {
    public void pull(CeilingFanPullChain wrapper) {
        wrapper.set_state(new Off());
        System.out.println("turning off");
    }
}

public class StateDemo {
    public static void main(String[] args) {
        CeilingFanPullChain chain = new CeilingFanPullChain();
        while (true) {
            System.out.print("Press ENTER");
            getLine();
            chain.pull();
        }
    }

    static String getLine() {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        String line = null;
        try {
            line = in.readLine();
```

```
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        return line;
    }
}
```

3. Existing Example in FEM-MAT-OO

Nope.

4. Design Proposal in FEM-MAT-OO

- ~~TopOpt + ShapeOpt → to trigger transition (treating optimizers as strategies)~~
- ~~To switch line-search strategies according to some criteria.~~ → STRATEGY
- Monitoring to coordinate refreshing.
-