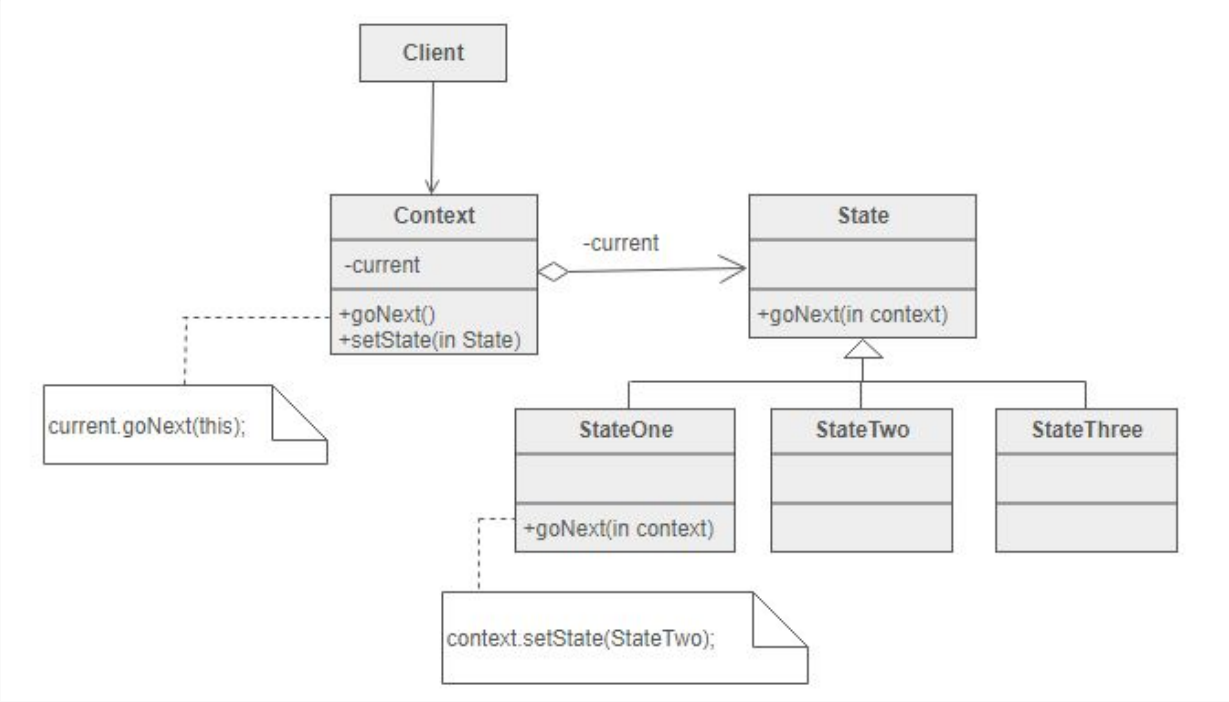
State Design Pattern

What is it?

- Behavioral pattern
- Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.
- Uses Polymorphism to define different behaviors for different states of an object.

How to implement

- Client
- Context
- State
- Derived States



Example

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

```java
class CeilingFanPullChain {
    // Context Class
    private State currentState;

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

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

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

Off.java

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

Low.java

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

High.java

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

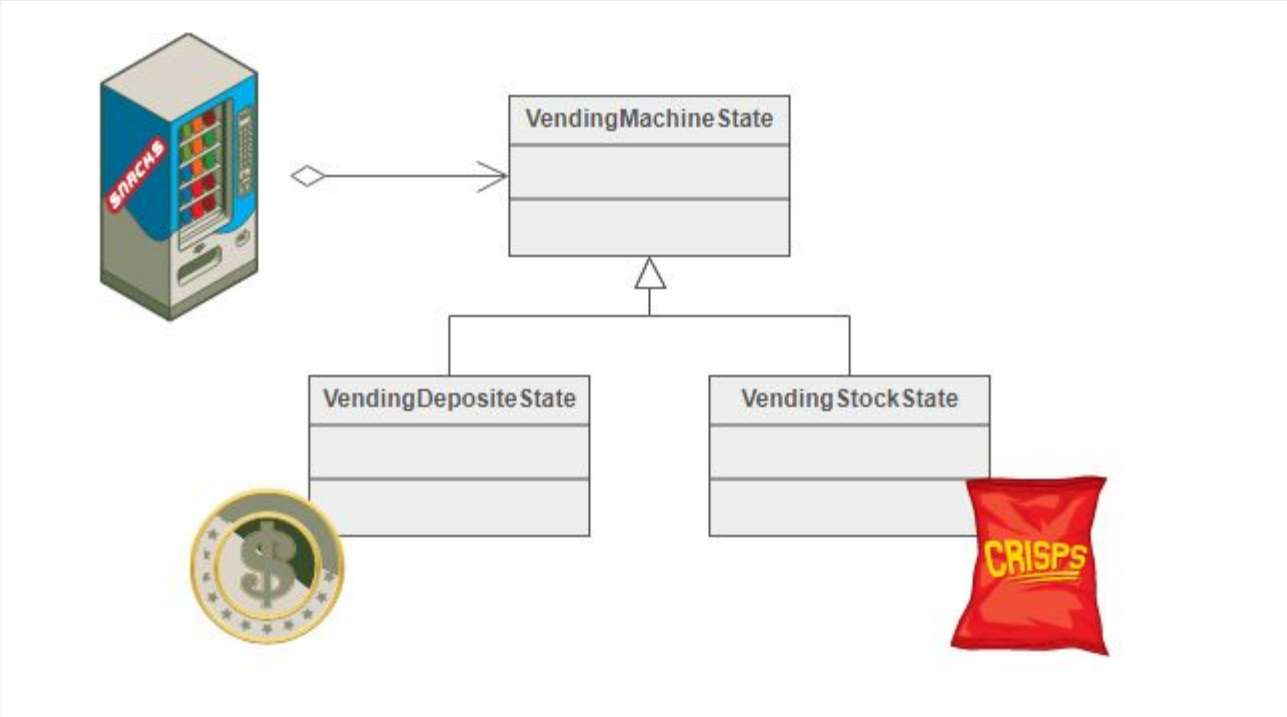
Medium.java

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

Client Implementation

```
1  public class StateDemo {
2      // Client class
3      public static void main(String[] args) {
4          CeilingFanPullChain chain = new CeilingFanPullChain();
5          while (true) {
6              System.out.print("Press ENTER");
7              getLine();
8              chain.pull();
9          }
10     }
11
12     static String getLine() {
13         BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
14         String line = null;
15         try {
16             line = in.readLine();
17         } catch (IOException ex) {
18             ex.printStackTrace();
19         }
20         return line;
21     }
22 }
23
```

Another Example



Checklist

- Identify an existing class, or create a new class, that will serve as the "state machine" from the client's perspective. That class is the "wrapper" class.
- The wrapper class maintains a "current" State object.
- Create a State base class that replicates the methods of the state machine interface. Each method takes one additional parameter: an instance of the wrapper class. The State base class specifies any useful "default" behavior.
- Create a State derived class for each domain state. These derived classes only override the methods they need to override.
- All client requests to the wrapper class are simply delegated to the current State object, and the wrapper object's this pointer is passed.
- The State methods change the "current" state in the wrapper object as appropriate.

Benefits and Drawbacks

- Polymorphic
- Easy to add more states (Flexible)
- Avoids tedious conditional branching

- Class Explosion
- Maintainability