

State pattern (Zustandsmuster)

Hintergrund

Das Zustands-Entwurfsmuster ist eine Methode zur Modellierung von zustandsabhängigen Verhalten eines Objekts, resp. es soll sein äusseres Verhalten zur Laufzeit aufgrund seines Zustands ändern. Das Verhalten eines Objekts ändert sich entsprechend seines internen Zustands. Für alle möglichen Zustände wird eine definierte Schnittstelle bereitgestellt. Die Schnittstelle wird für jeden einzelnen Zustand durch eine separate Klasse implementiert.

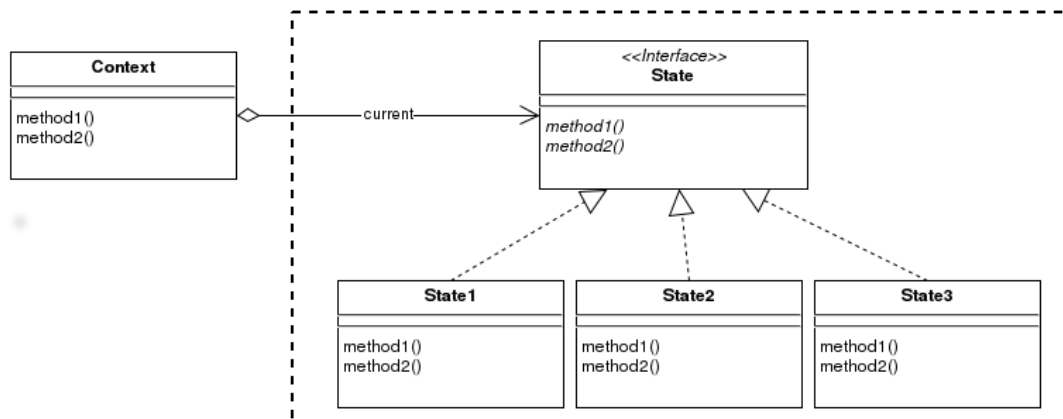


Abbildung 1: State pattern

Vor- und Nachteile

Weitere Zustände und Verhalten können einfach hinzugefügt werden. Die Weiterverwendung der Zustände und bessere Wartbarkeit sind ebenfalls Vorteile. Unleserlichkeit und Unübersichtlichkeit wird entgegen gewirkt, da umfangreiche if-then-else- und switch-Konstrukte vermieden werden.

Ein Nachteil ist, dass bei einfachen Zustand ein erhöhter Implementierungsaufwand besteht, denn alle Aktionen und Zustände müssen definiert werden. Dies führt zu einer erhöhten Klassenanzahl, da die Schnittstelle komplett implementiert werden muss.

Mögliche Implementation

Für eine Türe zeigt nachfolgende Implementation eine Anwendung des Zustandsmuster. Die Türe hat die folgenden Zustände und Aktionen:

- offen → schliessen
- geschlossen → abschliessen
- verschlossen → öffnen

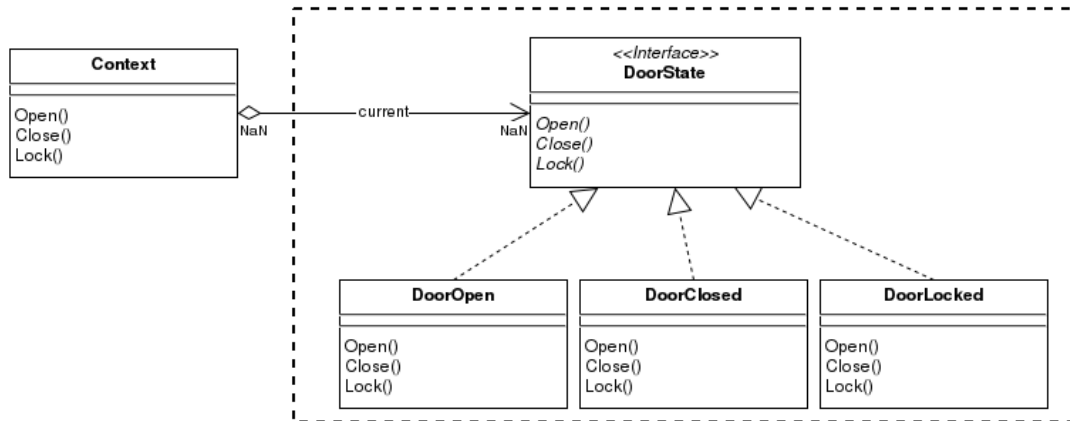


Abbildung 2: State pattern

Das Interface listet die möglichen Aktionen gemäss der obigen Aufzählung auf.

Listing 1: Interface

```

1 // State of the door
2 public interface DoorState {
3     public String close();
4     public String lock();
5     public String open();
6 }

```

Analog Abbildung 2 gibt es drei Zustände und darausfolgende Aktionen.

Listing 2: Zustand: Türe geöffnet

```

1 //Concrete State: DoorOpen
2 public class DoorOpen implements DoorState {
3     public String close() {
4         return "Can close the door.";
5     }
6
7     @Override
8     public String lock() {
9         return "Can not lock the door.";
10    }
11
12    @Override
13    public String open() {
14        return "Door is already open.";
15    }
16 }

```

Listing 3: Zustand: Türe geschlossen

```

1 //Concrete State: DoorClosed
2 public class DoorClosed implements DoorState {
3     @Override
4     public String lock() {
5         return "Lock the door.";
6     }
7
8     @Override

```

```

9     public String close() {
10         return "Door is already closed.";
11     }
12
13     @Override
14     public String open() {
15         return "Open the door.";
16     }
17 }

```

Listing 4: Zustand: Türe verschlossen

```

1 //Concrete State: DoorLocked
2 public class DoorLocked implements DoorState {
3     @Override
4     public String open() {
5         return "Can open the door. ";
6     }
7
8     @Override
9     public String close() {
10        return "Door is closed.";
11    }
12
13    @Override
14    public String lock() {
15        return "Door is already locked.";
16    }
17 }

```

Die Kontext-Klasse hat eine Verbindung zum Interface.

Listing 5: Context

```

1 // Context
2 public class Door implements DoorState {
3     DoorState doorState;
4
5     public Door(DoorState doorState) {
6         this.doorState = doorState;
7     }
8
9     public void setDoorState(DoorState doorState) {
10        this.doorState = doorState;
11    }
12
13    @Override
14    public String close() {
15        return doorState.close();
16    }
17
18    @Override
19    public String lock() {
20        return doorState.lock();
21    }
22
23    @Override
24    public String open() {

```

```

25         return doorState.open();
26     }
27 }

```

Die State-Klasse demonstriert die Verwendung.

Listing 6: State-Klasse

```

1 public class State {
2     public static void main(String[] args) {
3         Door door = new Door(new DoorOpen());
4         System.out.println("Door is open:");
5         System.out.println("- " + door.open());
6         System.out.println("- " + door.close());
7         System.out.println("- " + door.lock());
8
9         door.setDoorState(new DoorClosed());
10        System.out.println("Door is closed:");
11        System.out.println("- " + door.open());
12        System.out.println("- " + door.close());
13        System.out.println("- " + door.lock());
14
15        door.setDoorState(new DoorLocked());
16        System.out.println("Door is locked:");
17        System.out.println("- " + door.open());
18        System.out.println("- " + door.close());
19        System.out.println("- " + door.lock());
20    }
21 }

```

Je nach Zustand der Türe lassen sich natürlich nicht alle Aktion ausgeführt. Die nachfolgende Ausgabe zeigt dies.

Listing 7: Output

```

1 Door is open:
2 - Door is already open.
3 - Can close the door.
4 - Can not lock the door.
5 Door is closed:
6 - Open the door.
7 - Door is already closed.
8 - Lock the door.
9 Door is locked:
10 - Can open the door.
11 - Door is closed.
12 - Door is already locked.

```

Neben der Implementierung als Klassen, was vorheriges Beispiel zeigt, lässt sich das State-Pattern auch als Enumeration umsetzen.

Unsere Implementation