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 entsprechenden seines internen Zustand. Für alle möglichen Zustände wird eine definierte Schnittstelle bereitgestellt. Die Schnittstelle wird für jeden einzelnen Zustand durch eine separate Klasse implementiert.

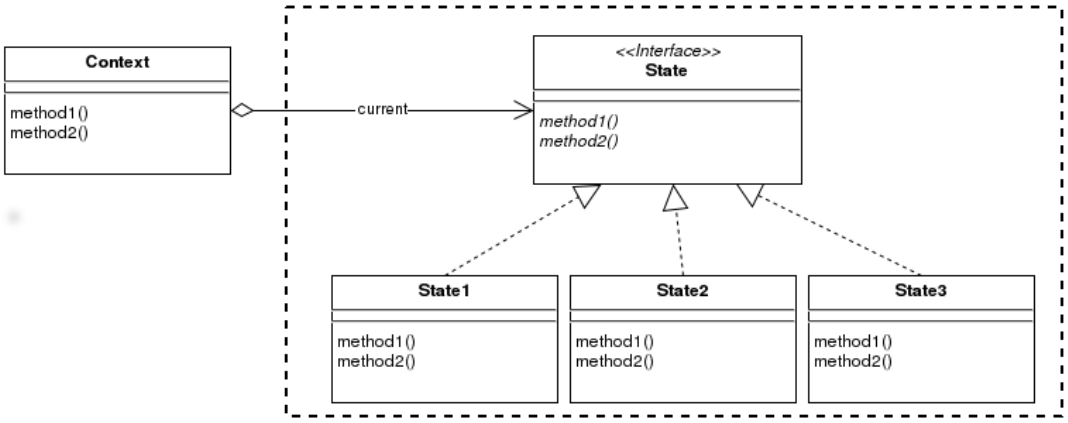

Abbildung 1: State pattern

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, das 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 Inplementation eine Anwendung des Zustandsmuster. Die Türe hat die folgenden Zustände und Aktionen:

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

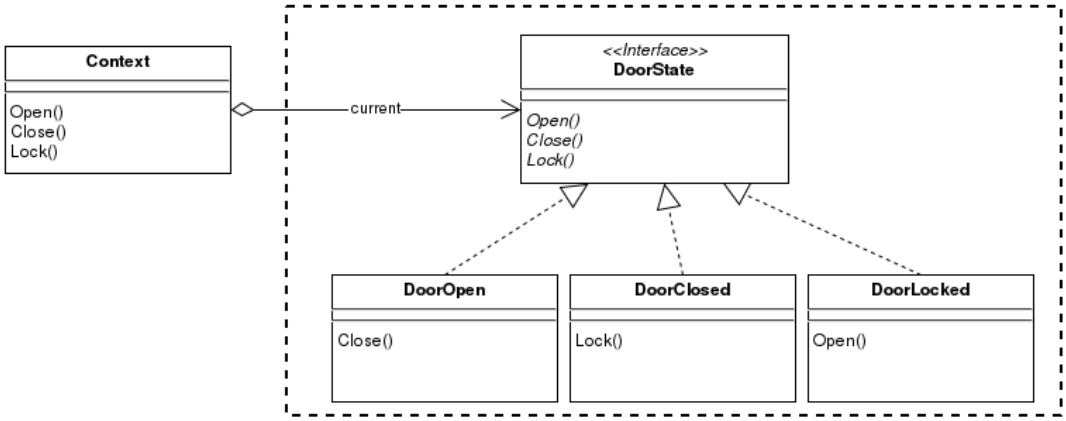

Abbildung 2: State pattern

Listing 1: Interface

```
// State of the door
public interface DoorState {
    public String action();
}
```

Analog Abbildung 2 gibt es drei Zustände und darausfolgende Aktionen.

Listing 2: Zustand: Türe geöffnet

```
//Concrete State: DoorOpen
public class DoorOpen implements DoorState {
    @Override
    public String action() {
        return "Close the door";
    }
}
```

Listing 3: Zustand: Türe geschlossen

```
//Concrete State: DoorClosed
public class DoorClosed implements DoorState {
    @Override
    public String action() {
        return "Lock the door";
    }
}
```

Listing 4: Zustand: Türe verschlossen

```
//Concrete State: DoorLocked
public class DoorLocked implements DoorState {
    @Override
    public String action() {
        return "Open the door";
    }
}
```

Die Kontext-Klasse hat eine Verbindung zum Interface.

Listing 5: Context

```
// Context
public class Door implements DoorState {
    DoorState doorState;

    public Door(DoorState doorState) {
        this.doorState = doorState;
    }

    public void setDoorState(DoorState doorState) {
        this.doorState = doorState;
    }

    @Override
    public String action() {
        return doorState.action();
    }
}
```

Die State-Klasse demonstriert die Verwendung.

Listing 6: State-Klasse

```
public class State {
    public static void main(String[] args) {
        Door door = new Door(new DoorOpen());
        System.out.println("Door is open: " + door.action());

        door.setDoorState(new DoorClosed());
        System.out.println("Door is closed: " + door.action());

        door.setDoorState(new DoorLocked());
        System.out.println("Door is locked: " + door.action());
    }
}
```

Jenach Zustand wird eine andere Aktion ausgeführt.

Listing 7: Context

```
Door is open: Close the door
Door is closed: Lock the door
Door is locked: Open the door
```

Unsere Implementation