Team Red CS1 / Task 12

State Pattern

Konzept

Problem:
Ein Objekt muss sich je nach Zustand unterschiedlich verhalten.
Mit dem Pattern wird vermieden, dass das unterschiedliche Verhalten mit langen if / else Anweisungen implementiert werden muss und trotzdem das Objekt ein einheitliches Interface bietet.

Beispiel:
Eine Datei hat verschiedene Stati: geöffnet, geschlossen, gedruckt, gesperrt und gelöscht. Falls jetzt versucht wird, die Datei zu löschen, ist das Verhalten je nach Status unterschiedlich. Wenn sie geöffnet ist, kann sie nicht gelöscht werden, etc.

Lösung:
Die verschiedenen Stati werden als einzelne Klassen mit einer gemeinsamen abstrakten Oberklasse oder einem Interface definiert. Darin wird jetzt das statusabhängige Verhalten implementiert.
Das eigentliche Objekt hat eine Referenz zum jeweiligen Status eine Methode um diesen zu ändern.

Vorteile / Nachteile

Vorteile:

- Saubere Trennung des unterschiedlichen Verhaltens
- Einfaches hinzufügen / entfernen von Stati
- Einheitliches Interface

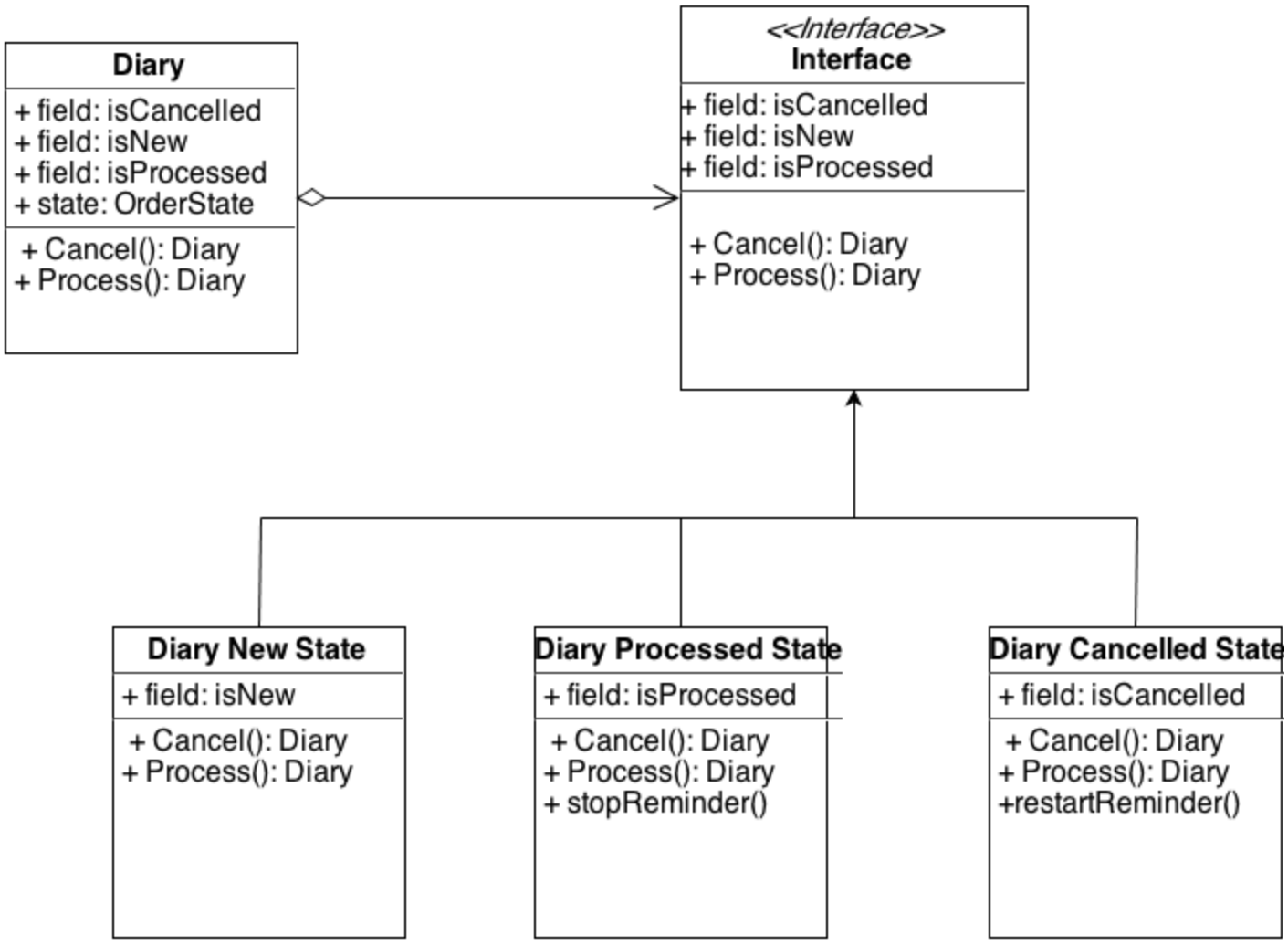
Nachteile:

- Bei trivialem Verhalten viel Overhead
- Gefahr von Doppelspurigkeiten, wenn zwei Stati sehr ähnlich sind.

Mögliche Implementation

Dateien auf Filesystem, Aufträge eines Onlineshops, Lieferung der Post, Transaktion einer Bank, etc.

Feature in Mobile Application, welches mit dem State Pattern umgesetzt wird: **Tagebuch**



```
package ch.bfh.red.app.view;

//Grundlage für konkrete Zustaende
public interface DiaryState {

	public void cancel();

	public void process();

}

public class DiaryNewState implements DiaryState{
	@Override
	public void cancel(){
		restartReminder();

	}
	@Override
	public void process(){
		showDiaryEditor();
	}

}

public class DiaryProcessedState implements DiaryState{
	@Override
	public void cancel(){

	}
	@Override
	public void process(){
		showDiary();
	}

}

public class DiaryCancelledState implements DiaryState{
	@Override
	public void cancel(){

	}
	@Override
```

```
        public void process(){
                restartReminder();
        }

}

//Diary ist der Kontext für die Zustaende
public class Diary implements DiaryState{

        DiaryState state;

        public DiaryModel(DiaryState state){
                this.state = state;
        }

        public void setDiaryState(DiaryState state){
                this.state = state;
        }

        @Override
        public void cancel(){
                state.cancel();

        }
        @Override
        public void process(){
                state.process();
        }

}
//Anwendungsbeispiel
Diary diary = new Diary(new DiaryNewState());

public void buttonClick(ClickEvent event) throws CommitException {
       if (event.getButton() == newEntry) {
           diary.process();
       } else if (event.getButton() == cancel) {
           diary.cancel();
       }
//       close();
   }
```