9/3/2014

Design Patterns: State

DEUTSCHES
KREBSFORSCHUNGSZENTRUM
IN DER HELMHOLTZ-GEMEINSCHAFT

50 Jahre – Forschen für
ein Leben ohne Krebs

Design Pattern

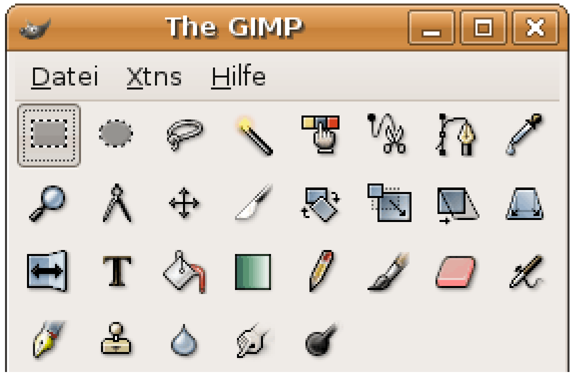
- A pattern is a recurring solution to a standard problem, in a context
- Guidelines for implementing software
- Approved designs to solve architectural problems

- Different types:
  - Creational
    - Deal with initializing and configuring classes and objects
  - Structural
    - Deal with decoupling interface and implementation of classes and objects
    - Composition of classes or objects
  - Behavioral
    - Deal with dynamic interactions among societies of classes and objects
    - How they distribute responsibility
  - Concurrency
    - Deal with multi-threaded programming paradigm

The State Pattern

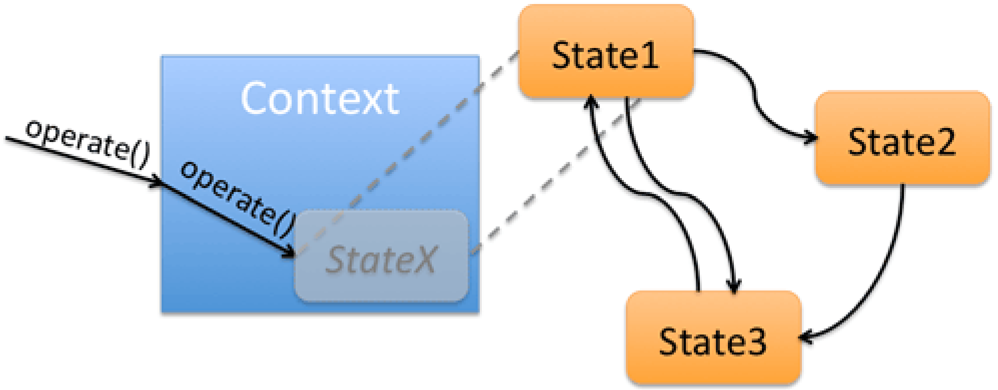
- A behavioral pattern
- Encapsulate varying behavior for the same routine based on an object's state object
- Avoid large monolithic conditional statements

- Usage (examples):
  - Drawing tools
  - State machines (Parser)
  - (Network-) connections (e.g. TCP)



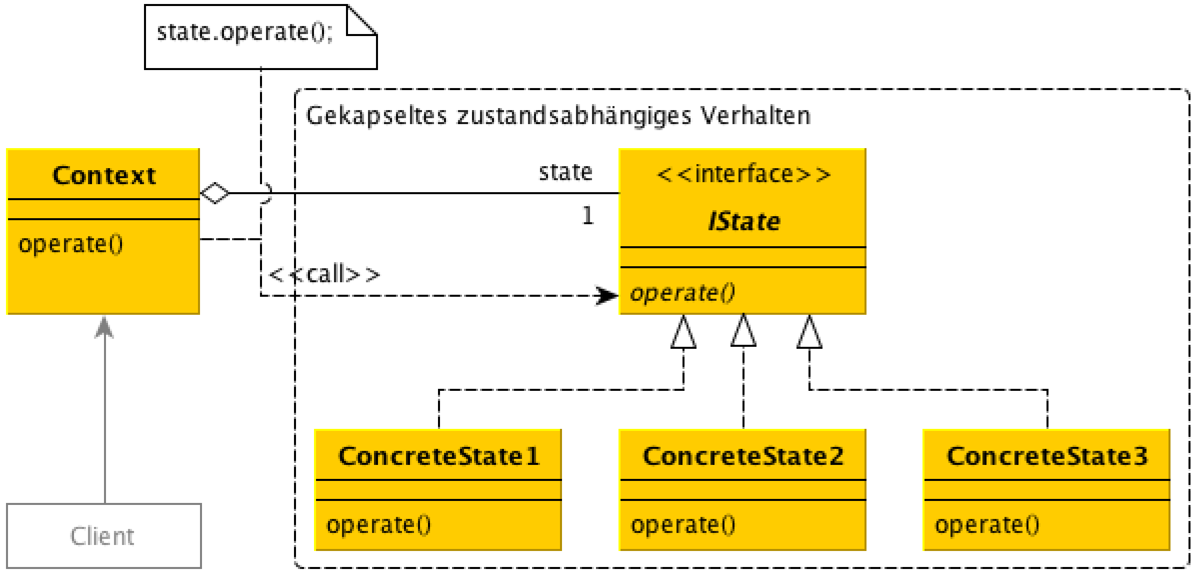
$$\begin{tikzcd}
\text{Start} \arrow[r] & Z_0 \arrow[r, "a"] \arrow[d, "b"] & Z_1 \arrow[dl, "b"] \arrow[d, "a"] \\
 & Z_3 \arrow[loop below, "a,b"] & Z_2 \arrow[loop below, "a,b"]
\end{tikzcd}$$

The State Pattern



```
class Context
{
  private IState state;

  public void operate()
  {   state.operate();   }
}
```

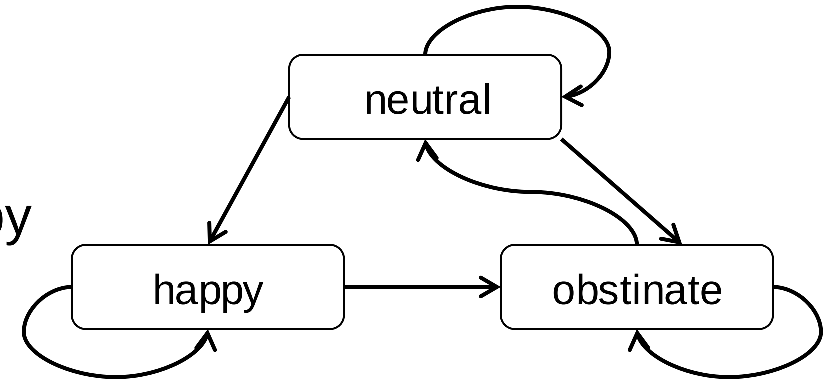
```
interface IState
{
  public void operate();
}
```



Source: http://www.philipphauer.de/study/se/design-pattern.php

Example

- Modeling your girl/boy friend:
  - Interact with him/her: talk, kiss, annoy
  - State (Mood): neutral, obstinate, happy

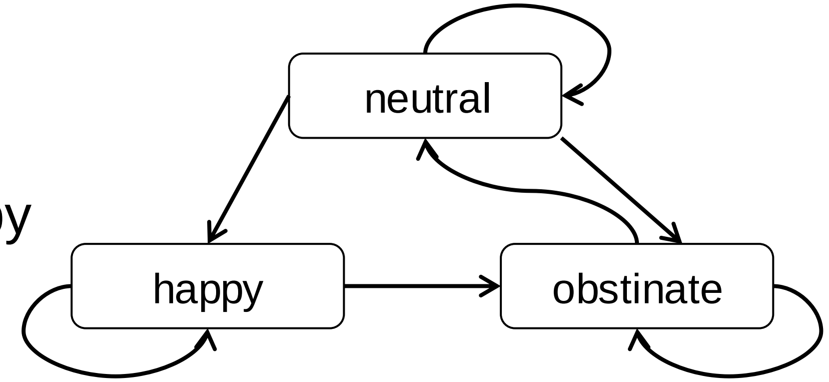


```
class Girlfriend{
    // Mood states
    private static final int NEUTRAL = 0;
    private static final int OBSTINATE = 1;
    private static final int HAPPY = 2;

    private int currentMood;

    //State-dependent behavior
    public void talk(){
        if (currentMood  == NEUTRAL){ System.out.println("Fününününü.");}
        else if (currentMood  ==  OBSTINATE){ System.out.println("Driving home now! I do not
                                               want to talk to you!!");}
        else if (currentMood  == HAPPY){ System.out.println("Hihi, Fünüüüüüüünü!");}
    }
    public void kiss(){
        if (currentMood  == NEUTRAL){
           …
    }
…
}
```

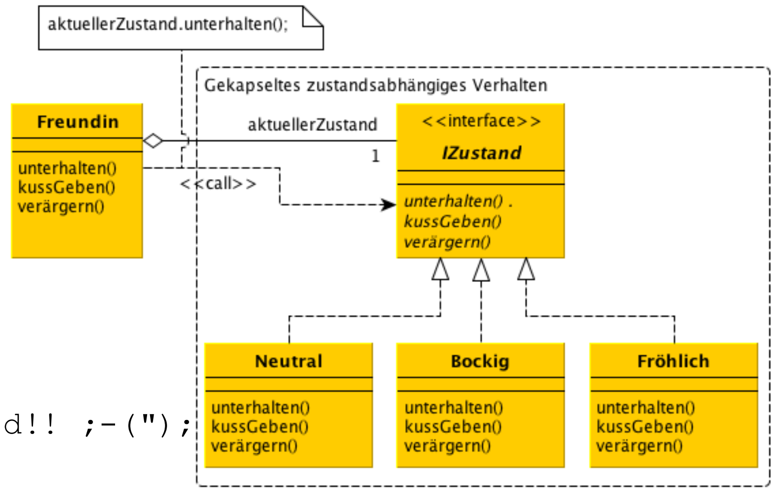
Example: http://www.philipphauer.de/study/se/design-pattern.php

Example

- Modeling your girl/boy friend:
  - Interact with him/her: talk, kiss, annoy
  - State (Mood): neutral, obstinate, happy



```
class Neutral {
    public void talk() {
        //NEUTRAL-dependent behavior
        System.out.println("Fününününü.");
    }
    public void kiss() {
        //NEUTRAL-dependent behavior
       System.out.println("Hihi :-)");
    }
    public void obstinate() {
        //NEUTRAL-dependent behavior
        System.out.println("You're kidding! I'm pissed!! ;-(");
    }
}
class Obstinate{
    public void talk() {…}
}
…
```



Example: http://www.philipphauer.de/study/se/design-pattern.php

Pos and Cons

+ Extensibility & Change robust
+ Increased cohesion → Intuitive & Comprehensible
+ Explicite state transition

– Increased number of classes
– Less compact

- Excursion: State vs Strategy Pattern
    - Same structure, different intention

Sources

- Design Patterns. Elements of Reusable Object-Oriented Software, Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29
- http://www.philipphauer.de/study/se/design-pattern.php