Aufgabe 2: Klassendiagramme, STATE-Pattern, Zustandsdiagramme

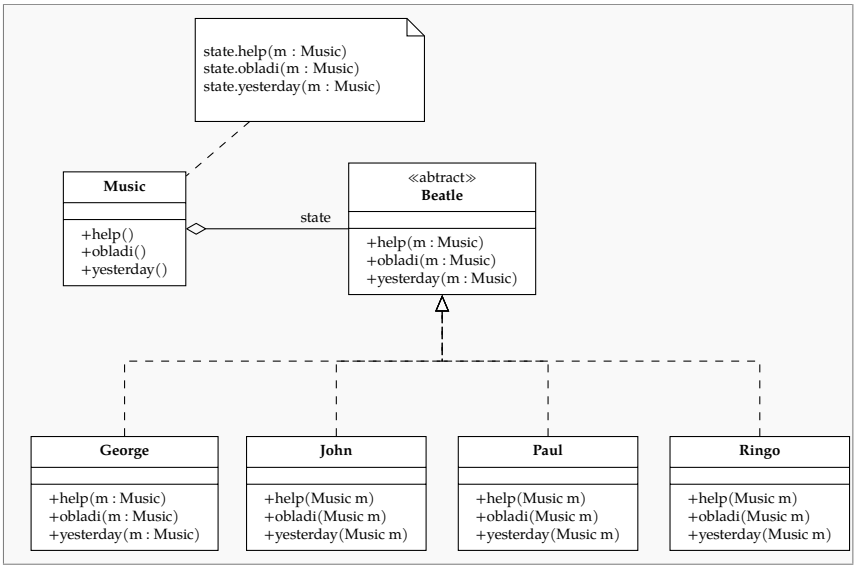
Gegeben sei das Java-Programm:

```java
public class Music {
  private Beatle state;

  public Music() {
    state = new Paul();
  }

  public void help() {
    this.state.help(this);
  }

  public void obladi() {
    this.state.obladi(this);
  }

  public void yesterday() {
    this.state.yesterday(this);
  }

  public void setBeatle(Beatle b) {
    state = b;
  }
}

abstract class Beatle {
  public abstract void help(Music m);

  public abstract void obladi(Music m);

  public abstract void yesterday(Music m);
}

class George extends Beatle {
  public void help(Music m) {
    System.out.println("help");
    m.setBeatle(new John());
  }

  public void obladi(Music m) {
  }

  public void yesterday(Music m) {
    System.out.println("yesterday");
    m.setBeatle(new Paul());
  }
}

class John extends Beatle {
  public void help(Music m) {
    System.out.println("help");
    m.setBeatle(new Paul());
  }

  public void obladi(Music m) {
    System.out.println("obladi");
    m.setBeatle(new Ringo());
```

```
59    }
60
61    public void yesterday(Music m) {
62    }
63  }
64
65  class Paul extends Beatle {
66    public void help(Music m) {
67      System.out.println("help");
68      m.setBeatle(new George());
69    }
70
71    public void obladi(Music m) {
72    }
73
74    public void yesterday(Music m) {
75      System.out.println("yesterday");
76    }
77  }
78
79  class Ringo extends Beatle {
80    public void help(Music m) {
81      System.out.println("help");
82      m.setBeatle(new John());
83    }
84
85    public void obladi(Music m) {
86    }
87
88    public void yesterday(Music m) {
89      System.out.println("yesterday");
90      m.setBeatle(new Paul());
91    }
92  }
```

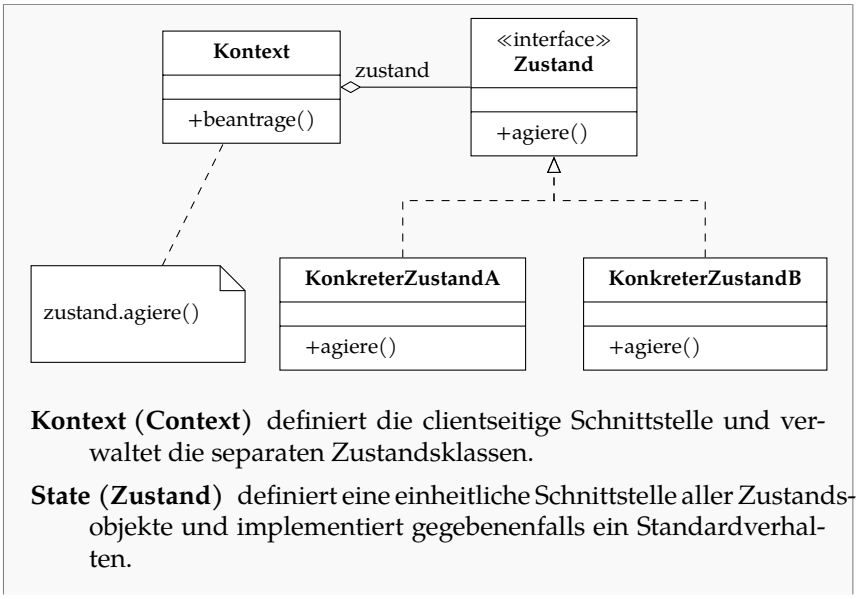
github: raw

(a) Zeichnen Sie ein UML-Klassendiagramm, das die statische Struktur des Programms modelliert. Instanzvariablen mit einem Klassentyp sollen durch gerichtete Assoziationen mit Rollennamen und passender Multiplizität am gerichteten Assoziationsende modelliert werden. Alle aus dem Programmcode ersichtlichen statischen Informationen sollen in dem Klassendiagramm dargestellt werden.



Das Programm implementiert ein Zustandsdiagramm, das das Verhalten von Objekten der Klasse `Music` beschreibt. Für die Implementierung wurde das Design-Pattern `STATE` angewendet.

(c) Geben Sie die statische Struktur des STATE-Patterns an und erläutern Sie, welche Rollen aus dem Entwurfsmuster den Klassen des gegebenen Programms dabei zufallen und welche Operationen aus dem Entwurfsmuster durch (ggf. mehrere) Methoden in unserem Beispielprogramm implementiert werden. Es ist von den z. B. im Design-Pattern-Katalog von Gamma et al. verwendeten Namen auszugehen, das heißt von Klassen mit Namen `Context`, `State`, `ConcreteStateA`, `ConcreteStateB` und von Operationen mit Namen `request` und `handle`.



Kontext (Context) definiert die clientseitige Schnittstelle und verwaltet die separaten Zustandsklassen.

State (Zustand) definiert eine einheitliche Schnittstelle aller Zustandsobjekte und implementiert gegebenenfalls ein Standardverhalten.

KontreterZustand (**ConcreteState**) implementiert das Verhalten, das mit dem Zustand des Kontextobjektes verbunden ist.

(d) Zeichnen Sie das UML-Zustandsdiagramm (mit Anfangszustand), das von dem Programm implementiert wird. Dabei muss - gemäß der UML-Notation - unterscheidbar sein, was Ereignisse und was Aktionen sind. In dem Diagramm kann zur Vereinfachung statt `System.out.println ("x")` einfach `"x"` geschrieben werden.