1. **Politiker**

   Mit dem folgenden Zustandsdiagramm wird das Verhalten eines fiktiven Politikers beschrieben. Beachten Sie, dass es eine Parallelkomposition und mehrere hierarchische Zustände gibt.

   Eine Kantenbeschriftung `e/a` bedeutet, dass nach Erhalt des Ereignisses `e` eine Aktion `a` (hier ein internes Ereignis) ausgelöst wird. Weiterhin stellen wir Ihnen ein Programmfragment zur Verfügung, mit dem der Nutzer ein solches Objekt steuern kann. Die Grundidee ist dabei, dass der Nutzer eine Eingabe macht und das Objekt genau seinen aktuellen Zustand ausgibt. Der bereits implementierte Dialog in der hier zur Verfügung gestellten Klasse `Steuerung`, die noch ergänzt werden muss, hat folgendes Aussehen:

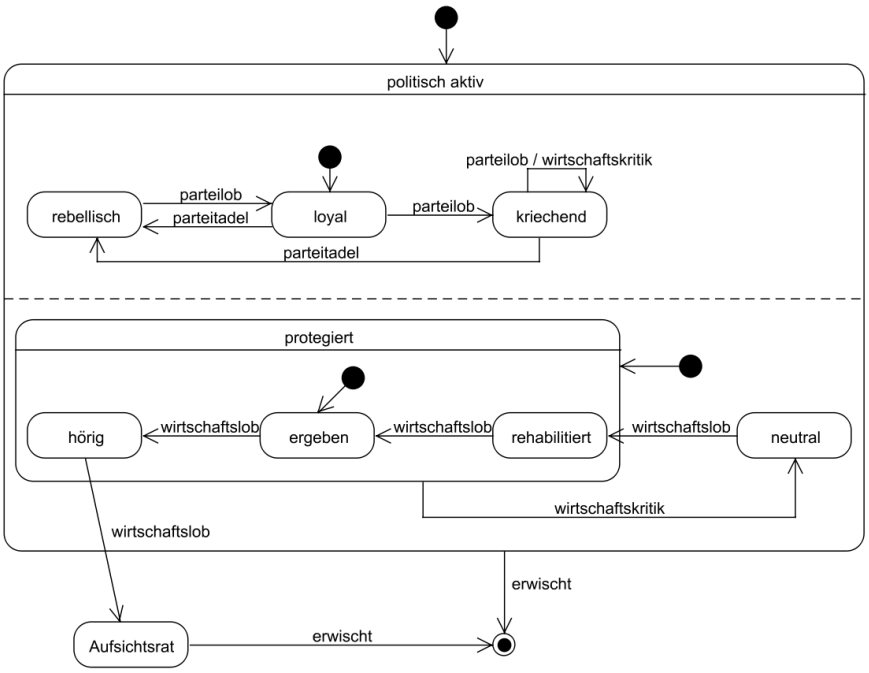
```
Welches naechste Ereignis?
(0) Lob von der eigenen Partei
(1) Tadel von der eigenen Partei
(2) Lob von der Wirtschaft
(3) Erwischt
1
Fiktiver Politiker befindet sich in (Teil)-Zustaenden:
POLITISCH_AKTIV REBELLISCH PROTEGIERT ERGEBEN
```

   Man sieht die Ausgabe, nachdem der Nutzer das Ereignis `parteitadel` ausgelöst hat und die genauen Informationen über den aktuellen Zustand ausgegeben werden.

```java
import java.util.Scanner;

public class Steuerung {
    private Context_Politiker politiker;

    public Steuerung(){
        int eingabe=-1;
        politiker=new Context_Politiker();
        while(politiker.aktiv()){
            System.out.println("Welches naechste Ereignis?\n"
                    +" (0) Lob von der eigenen Partei\n"
                    +" (1) Tadel von der eigenen Partei\n"
                    +" (2) Lob von der Wirtschaft\n"
                    +" (3) Erwischt");
            eingabe=new Scanner(System.in).nextInt();
            switch(eingabe){
            case 0:
                politiker.parteilob();
                break;
            case 1:
                politiker.parteitadel();
                break;
            case 2:
                politiker.wirtschaftslob();
                break;
            case 3:
                politiker.erwischt();
                break;
            }
            System.out.println("Fiktiver Politiker befindet sich in"
                    +" (Teil)-Zustaenden: " + politiker);
        }
    }

    public static void main(String[] args) {
        new Steuerung();
    }
}
```

Listing 1: `Steuerung.java`

Dieser Zustandsautomat zeigt alle möglichen Zustände und Unterzustände eines Politikers sowie deren Übergänge.




*Hinweis:* Dem Endzustand können Sie z.B. den Namen Ruhestand oder Ende geben (in der Implementierung und ggf. bei Aufgabenteil (a).

(a) Welchen Zustand erreicht der Automat, nachdem die folgenden Ereignisse aufgetreten sind? (getrennt betrachten)

i. `parteilob.parteitadel`

**Lösung:**

`POLITISCH_AKTIV REBELLISCH PROTEGIERT ERGEBEN`

ii. `parteilob.wirtschaftslob.wirtschaftslob`

**Lösung:**

`POLITISCH_AKTIV KRIECHEND AUFSICHTSRAT`

iii. `parteilob.witschaftslob.parteilob.wirtschaftslob`

**Lösung:**

`POLITISCH_AKTIV KRIECHEND PROTEGIERT REHABILITIERT`

iv. `parteilob.witschaftslob.erwischt`

**Lösung:**

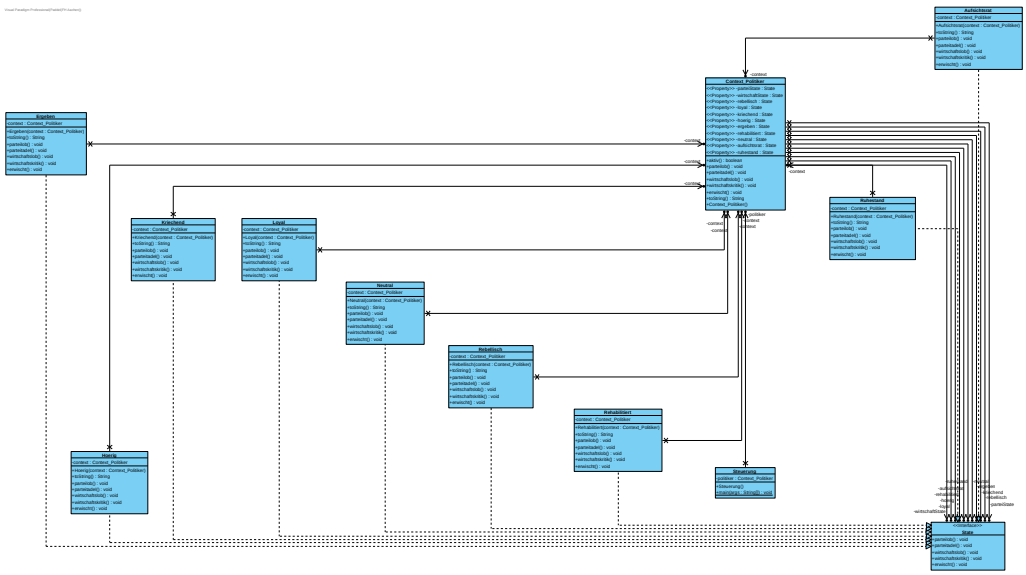
`RUHESTAND`

(b) Ihre Aufgabe besteht darin, den Zustandsautomaten präzise unter Nutzung und Ergänzung der Klasse `Steuerung` (Quellcode siehe oben) zu *implementieren*.

*Recherchieren Sie das State-Pattern* und setzen Sie den obigen Zustandsautomaten damit um. Bedenken Sie dabei, dass ein konkreter Zustand durchaus auch der Context für weitere Unterzustände sein kann. Erzeugen Sie im Anschluss an die Implementierung mit Hilfe von Visual Paradigm ein *Klassendiagramm* aus ihrem Code (Sie können natürlich auch mit dem Klassendiagramm beginnen und daraus Code erzeugen!). Überprüfen Sie mit Ihrem Code Ihre Ergebnisse aus Aufgabenteil (a).

Weitere Informationen zur *Code Generation* mit Visual Paradigm finden Sie hier.

**Lösung:**



Der Code befindet sich im Anhang.

Eine gute Erklärung zum State-Pattern ist hier zu finden.

P.S. Machen Sie sich Gedanken darüber, wie eine Aktion wie z.B. `wirtschaftskritik` aus dem Unterzustand `politisch aktiv` an `protegiert` weitergeleitet werden kann.