Hausaufgabe 10

Tim Wende

December 12, 2021

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.

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
   `POLITISCH_AKTIV REBELLISCH PROTEGIERT ERGEBEN`

   ii. parteilob.wirtschaftslob.wirtschaftslob
   `AUFSICHTSRAT`
   alternativ[1]:
   `POLITISCH_AKTIV KRIECHEND AUFSICHTSRAT`

   iii. parteilob.wirtschaftslob.wirtschaftslob
   `POLITISCH_AKTIV KRIECHEND PROTEGIERT REHABILITIERT`

   iv. parteilob.witschaftslob.erwischt
   `RUHEZUSTAND`

[1] Siehe Code (vorab: ungenaue Definition von parallelen inneren Zuständen)

b. Ihre Aufgabe besteht darin, den Zustandsautomaten präzise unter Nutzung und Ergänzung der Klasse Steuerung (Quellcode siehe ~~oben~~ unten) 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.

Vorab: Hier eine gekürzte übersichtliche Version:

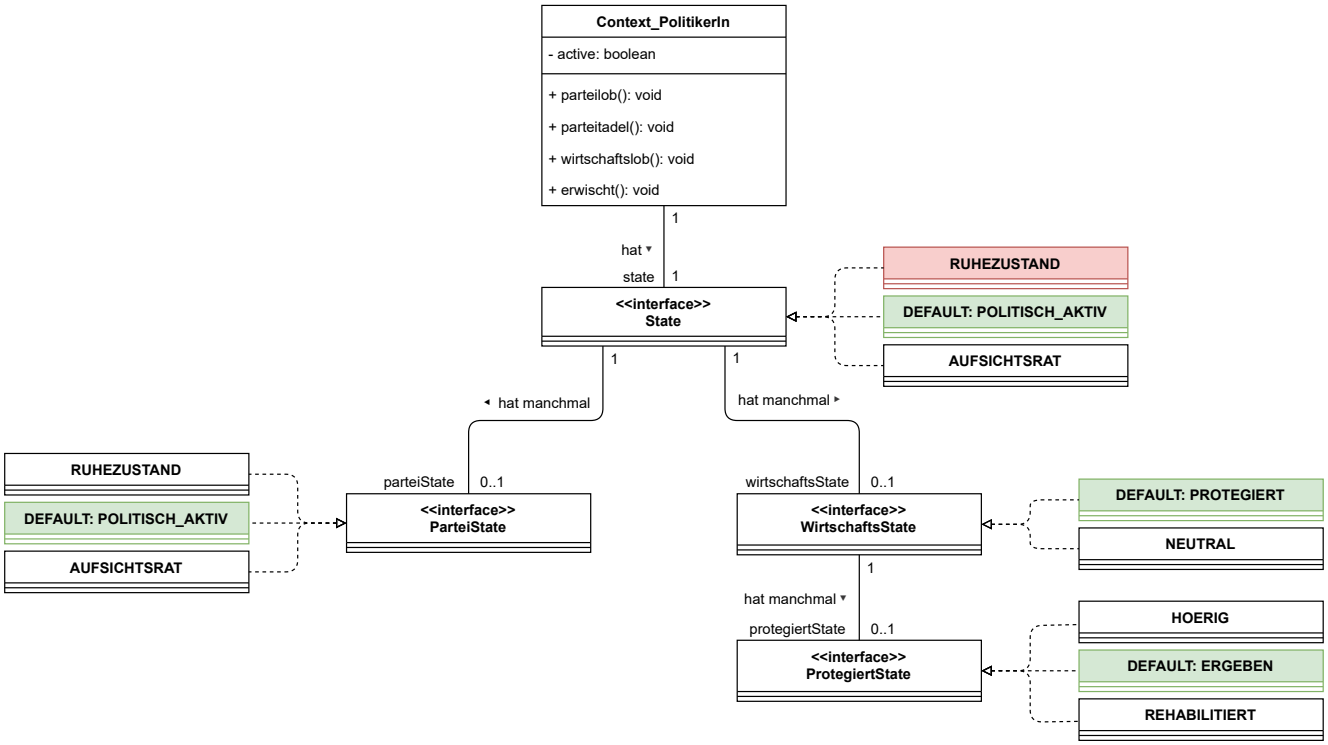

Figure 1: `class_diagram`

- Die im Klassendiagramm dargestellten Interfaces sind offensichtlich Implementierungen dieser. Daher gehören die durch Rollen implizit gegebenen Attribute auch zu den Interfaces
- Die grün dargestellten Klassen sind die default Klassen
- Die rot dargestellten Klassen sind Endzustände

Hier die vollständige Version aus Visual Paradigm:

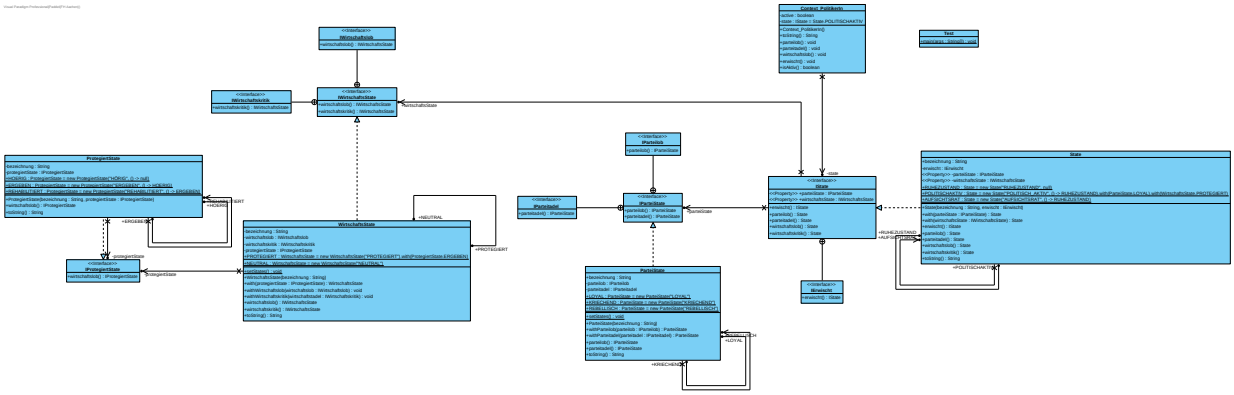

Figure 2: `class_diagram_vp`

Hier der gegebene Zustandsautomat dazu:
(Wer auch immer den Pfeil von `hörig` zu `Aufsichtsrat` erstellt hat :/)

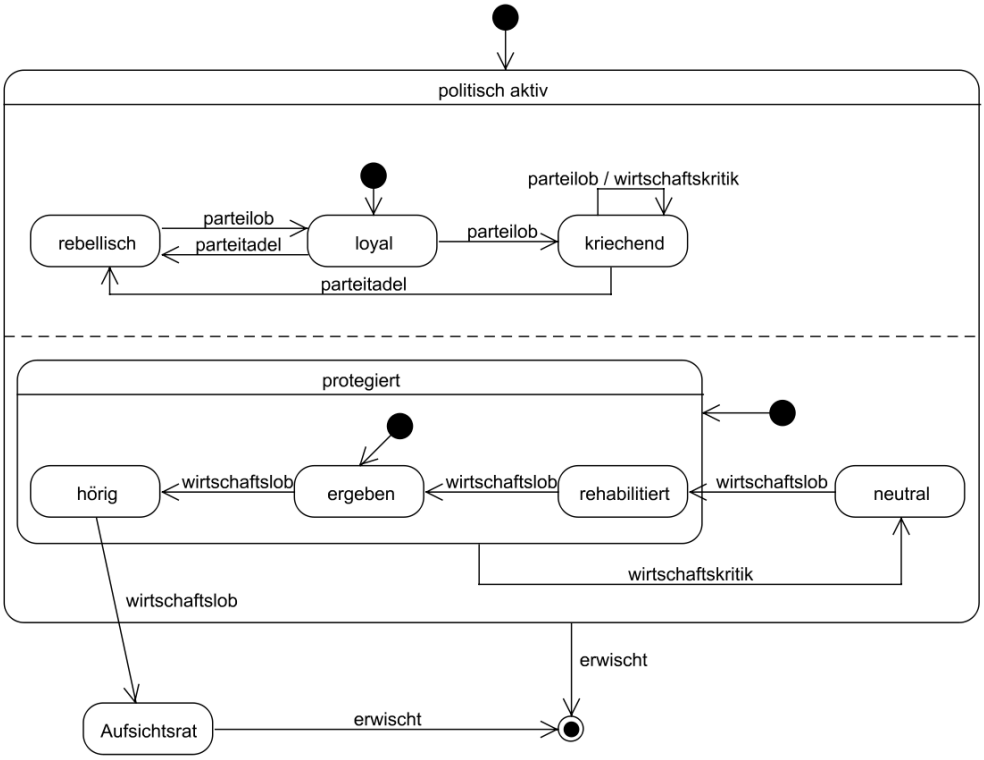

Figure 3: `politiker_zustandsautomat`

Hier die leicht veränderte Steuerung:

```
import java.util.Scanner;

public class Steuerung {
	private Context_PolitikerIn politiker;

	public Steuerung(){
		int eingabe=-1;
		politiker=new Context_PolitikerIn();
		while(politiker.isAktiv()){
			System.out.println("Welches nste 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)-Zusten: " + politiker);
		}
	}

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

Und hier meine Testklasse:

```
import java.util.Scanner;

public class Test {
	public static void eingabe() {
		int eingabe;
		Scanner sc = new Scanner(System.in);
		Context_PolitikerIn politikerIn = new Context_PolitikerIn();

		while (politikerIn.isAktiv()) {
			System.out.println("Welches nchste Ereignis?\n" + " (0) Lob von der eigenen
					+ " (1) Tadel von der eigenen Partei\n"
					+ " (2) Lob von der Wirtschaft\n" + " (3) Erwischt\n" + " (*
			eingabe = sc.nextInt();
			switch (eingabe) {
			case 0:
				politikerIn.parteilob();
				break;
			case 1:
				politikerIn.parteitadel();
				break;
			case 2:
				politikerIn.wirtschaftslob();
				break;
			case 3:
				politikerIn.erwischt();
				break;
			default:
				return;
			}
			System.out.println("Fiktiver Politiker befindet sich in" + " (Teil)-Zustnder
		}

		sc.close();
	}

	public static void a_i() {
		Context_PolitikerIn politikerIn = new Context_PolitikerIn();
		politikerIn.parteilob();
		politikerIn.parteitadel();
		System.out.println(politikerIn);
	}

	public static void a_ii() {
		Context_PolitikerIn politikerIn = new Context_PolitikerIn();
		politikerIn.parteilob();
		politikerIn.wirtschaftslob();
		politikerIn.wirtschaftslob();
		System.out.println(politikerIn);
	}

	public static void a_iii() {
		Context_PolitikerIn politikerIn = new Context_PolitikerIn();
		politikerIn.parteilob();
		politikerIn.wirtschaftslob();
		politikerIn.parteilob();
		politikerIn.wirtschaftslob();
		System.out.println(politikerIn);
	}
```

```
        public static void a_iv() {
                Context_PolitikerIn politikerIn = new Context_PolitikerIn();
                politikerIn.parteilob();
                politikerIn.wirtschaftslob();
                politikerIn.erwischt();
                System.out.println(politikerIn);
        }

        public static void main(String[] args) {
//                a_i();
//                a_ii();
//                a_iii();
//                a_iv();

                eingabe();
        }
}
```

Auch in dieser Hausaufgabe entfallen alle weiteren Klassen, jedoch hier noch ein Paar Kommentare:

- Die Klassen mit dem Präfix `I` sind offensichtlich Interfaces. So weit, so bekannt, jedoch sind die Klassen ohne das eben besagte `I` Gerüst Klassen, welche genutzt werden, um nicht jede Klasse einzeln implementieren zu müssen. So nutze ich beispielsweise `State` drei mal (bzw. drei Objekte der Klasse `State`) anstatt die Klassen `RUHEZUSTAND`, `POLITISCHAKTIV`, `AUFSICHTSRAT` ... Diese sind dann als static Attribute in den Klassen zu finden:

  ```
  public static final State RUHEZUSTAND = new State("RUHEZUSTAND", null);
  public static final State POLITISCHAKTIV = new State("POLITISCH_AKTIV", () -> RUHEZUSTAND)
      .with(ParteiState.LOYAL).with(WirtschaftsState.PROTEGIERT);
  public static final State AUFSICHTSRAT = new State("AUFSICHTSRAT", () -> RUHEZUSTAND);
  ```

  Selbes Spiel wie in der letzten Hausaufgabe: Da die Objekte `final` sind, können diese `public sein`. Wenn man mit mehreren PolitikerInnen parallel arbeitet, muss man copy ctors für diese DEFAULT Attribute nutzen. Da diese jedoch dann in einem anderen Kontext existieren würden bzw dieser Fall gar nicht vorgesehen ist, habe ich diese nicht implementiert.
- Ich bin davon ausgegangen, dass ein/e PolitikerIn nicht in mehreren Zuständen sein kann, da die initial Zustände in POLITISCH_AKTIV keine parallelen Aubläufe darstellen. Demensprechend habe ich die Zustände wie hier zu sehen implementiert. Sollte der Zustand bei `ii.` `POLITISCH_AKTIV KRIECHEND AUFSICHTSRAT` sein, müsste man mehrere Zustände bei dem/der PolitikerIn ermöglichen. Dies ist natürlich dann nicht realitätsnah, nicht wahr liebe (leider) nicht ganz fiktive CDU Politiker.
- Das Interface `ParteiState` könne von `Parteilob` erben. Von dort aus könne `State` weiter die Methoden erben. Dies würde Code-dopplung verhindern und dank Javadoc die Klassen leesbarer machen. Dies hat jedoch nicht auf Anhieb funktioniert und Java wehrt sich irgendwie dagegen. Apropos: Auf Javadoc musste leider aus Zeit-Gründen verzichtet werden.
- Ich habe mal für jede Methode ein eigenes Interface erstellt, dies hilft mir beim Erstellen der DEFAULT Klassen. Sollte dies nicht gewollt sein, müsste man eigene (private inner) Klassen erstellen. Mehr dazu in der Ersten Anmerkung.

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:

In meiner Lösung; gar nicht. Man könnte den `State` durch die verschiedenen Methoden durchreichen, oder den Kontekt `public` machen. Alternativ reagiert man auf die resultierenden Umstände, wie in meinem Code.