Dependency-Hell

Arndt Franzen (www.arndt-georg-franzen.de)

16. Juni 2022

Ein paar Worte vorab ...

Dies ist ein Zettel und kein detailliert ausgearbeitetes Dokument. Vieles ist im Telegrammstil verfasst und bedarf daher u.U. weiterer Erläuterungen. Es stellt lediglich einen Arbeitsstand dar.

1 Einleitung

Im Folgenden geht es um das Problem der sog. „Dependency-Hell“, wie es z.B. bei Maven-Projekten vorkommt. Hier verwenden wir nicht Maven; stattdessen stellen wir das mit Java-Bordmitteln nach.

2 Beispiel

Unser Beispiel besteht aus drei „Projekten“:

- *Verkehrssystem.* Das Verkehrssystem besteht aus einer Ampel und den möglichen Status. Es liegt in zwei Versionen vor. In Version 1 gibt es die Ampelstatus rot, gelb und grün; in Version 2 rot, rot-gelb, gelb und grün.
- *Fahrzeugsteuerung.* Die Fahrzeugsteuerung lässt ein Fahrzeug aufgrund von Ampelsignalen fahren, halten, anfahren und abbremsen. Sie verwendet *Verkehrssystem* in Version 1.
- *Verkehrsüberwachung.* Die Verkehrsüberwachung erhält „Signale“ von einer Ampel und „leitet diese weiter“. Sie hat Abhängigkeiten zum *Verkehrssystem* in Version 2 (sowie zur *Fahrzeugsteuerung*).

3 Quelltext

Natürlich wissen wir, dass man einige Sachen i.d.R. anders implementieren würde; aber hier geht es lediglich um ein Beispiel.

- Verkehrssystem V1

```
package test;

public enum AmpelStatus {
    ROT, GELB, GRUEN
}
```

- Verkehrssystem V2

```
package test;

public enum AmpelStatus {
    ROT, ROT_GELB, GELB, GRUEN
}
```

- Fahrzeugsteuerung

```
package test;

public class Fahrzeugsteuerung {
    public void ampelStatusChanged(AmpelStatus aktuellerAmpelStatus, AmpelStatus
        letzterAmpelStatus) {
```

```
        System.out.println("Ampel-Status: " + letzterAmpelStatus + " -> " +
            aktuellerAmpelStatus);
        if (letzterAmpelStatus == AmpelStatus.GELB && aktuellerAmpelStatus ==
            AmpelStatus.ROT) {
            System.out.println("Fahrzeugsteuerung: Halten");
        }
        else if (letzterAmpelStatus == AmpelStatus.ROT && aktuellerAmpelStatus ==
            AmpelStatus.GELB) {
            System.out.println("Fahrzeugsteuerung: Anfahren");
        }
        else if (letzterAmpelStatus == AmpelStatus.GELB && aktuellerAmpelStatus ==
            AmpelStatus.GRUEN) {
            System.out.println("Fahrzeugsteuerung: Fahren");
        }
        else if (letzterAmpelStatus == AmpelStatus.GRUEN && aktuellerAmpelStatus ==
            AmpelStatus.GELB) {
            System.out.println("Fahrzeugsteuerung: Abbremsen");
        }
    }
}
```

- Verkehrsüberwachung

```
package test;

public class Verkehrsueberwachung {
    public static void main(String[] args) {
        Fahrzeugsteuerung tmpFahrzeugsteuerung = new Fahrzeugsteuerung();
        for (int i=1; i<args.length; i++) {
            AmpelStatus tmpLetzterAmpelStatus = mapAmpelStatus(args[i-1]);
            AmpelStatus tmpAktuellerAmpelStatus = mapAmpelStatus(args[i]);
            tmpFahrzeugsteuerung.ampelStatusChanged(tmpAktuellerAmpelStatus,
                tmpLetzterAmpelStatus);
        }
    }

    private static AmpelStatus mapAmpelStatus(String aAmpelStatus) {
        if (aAmpelStatus.equals("0")) {
            return AmpelStatus.ROT;
        }
        else if (aAmpelStatus.equals("1")) {
            return AmpelStatus.ROT_GELB;
        }
        else if (aAmpelStatus.equals("2")) {
            return AmpelStatus.GELB;
        }
        else if (aAmpelStatus.equals("3")) {
            return AmpelStatus.GRUEN;
        }
        else {
            return null;
        }
    }
}
```

Die Ampelstatus werden durch die Übergabe an die `main`-Methode simuliert.

4 Laufzeitverhalten

Wir lassen das Programm zunächst mit *Verkehrssystem* in Version 1 laufen

```
java -cp verkehrsueberwachung;fahrzeugsteuerung;verkehrssystem-v1 test.
    Verkehrsueberwachung 2 0 1 3 2 0
```

und erhalten eine Exception

```
Ampel-Status: GELB -> ROT
Fahrzeugsteuerung: Halten
Exception in thread "main" java.lang.NoSuchFieldError: ROT_GELB
        at test.Verkehrsueberwachung.mapAmpelStatus(Verkehrsueberwachung.java:18)
        at test.Verkehrsueberwachung.main(Verkehrsueberwachung.java:8)
```

Hierbei handelt es sich um eine „guten Fehler“, da wir durch einen Programmabbruch auf den Fehler hingewiesen werden.

Jetzt lassen wir das Programm mit *Verkehrssystem* in Version 2 laufen

```
java -cp verkehrsueberwachung;fahrzeugsteuerung;verkehrssystem-v2 test.
    Verkehrsueberwachung 2 0 1 3 2 0
```

und erhalten die folgende Ausgabe

```
Ampel-Status: GELB -> ROT
Fahrzeugsteuerung: Halten
Ampel-Status: ROT -> ROT_GELB
Ampel-Status: ROT_GELB -> GRUEN
Ampel-Status: GRUEN -> GELB
Fahrzeugsteuerung: Abbremsen
Ampel-Status: GELB -> ROT
Fahrzeugsteuerung: Halten
```

Wir sehen, dass das Fahrzeug nie anfährt. Hierbei handelt es sich um eine „schlechten Fehler“, da das Programm nicht wie erwartet reagiert.

5 Bemerkung

Natürlich gibt es noch eine weitere „Fehlerquelle“ aufgrund von Versionsproblemen. Diese ist aber so offensichtlich, dass sie hier nur erwähnt wird: Fehler zur Compilezeit. Beispiel: *Verkehrsüberwachung* soll mit *Verkehrssystem* in der Version 1 kompiliert werden. Hierbei handelt es sich gem. der Definition aus 4 um einen „guten Fehler“.