Softwareentwicklung II

Sommersemester 2020

Übungen

Aufgabenblatt 6

Gruppe 67

8.6.2020

Aufgabe 6

Lösungvorschlag Kinoticketverkauf

Benutzen oder enthalten sich zwei Klassen gegenseitig führt das zu einer, möglichst zu vermeidenden, zyklischen Beziehung. Ein gute Möglichkeit zur Entkopplung von Subwerkzeugen von ihren Kontextwerkzeugen ist das Beobachtermuster. Der wesentliche Trick hierbei ist, dass von Interfaces und abstrakten Klassen keine Exemplare erzeugbar sind.

Das Kontextwerkzeug `KassenWerkzeug` (da es von der Startup-Klasse erzeugt wird, ist es gleichzeitig das Top-Level-Werkzeug) erzeugt Exemplare der Subwerkzeuge `DatumsAuswaehlWerkzeug`, `VorstellungsAuswaehlWerkzeug` und `PlatzVerkaufsWerkzeug`. Um Informationen über Veränderungen der Datums- oder der Vorstellungsauwahl an das übergeordnete Werkzeug weiter zu leiten, müssten die Subwerkzeuge mit diesem in eine zyklische Beziehung treten. Deshalb beobachtet das `KassenWerkzeug` lediglich die entsprechenden Subwerkzeuge: `DatumsAuswaehlWerkzeug` und `VorstellungsAuswaehlWerkzeug`. Dazu wird ein Interface `Beobachter` erzeugt, welches von `KassenWerkzeug` implementiert wird, ausserdem zwei abstracte Klassen `DatumsBeobachtbar` und `VorstellungsBeobachtbar` von denen die entsprechenden Klassen erben. Hierin werden alle Beobachter (hier nur der eine) in einer Objektsammlung (ein Set) aufgeführt; weitere können mittels einer geeigneten Methode hinzugefügt werden. Die einzigen Operationen des Interfaces sind `reagierenAufAenderung...()` für jedes zubeobachtenden Werkzeug. Diese werden jeweils innerhalb einer Methode `informiereUeberAenderung...()` der `Beobachtbar`-Klassen an jedem Beobachter des Sets innerhalb einer (erweiterten) for-Schleife aufgerufen (Abb: 1).
Das dynamische Zusammenspiel der Klassen in dem Beobachtungsmustern zeigt sich durch die Verfolgung einer Informationskette. Zunächst tritt eine Ereignis in einem der beiden beobachteten Subwerkzeuge auf, ausgelöst durch interaktive Aktionen der Datums- oder Vorstellungsauswahl. Durch das Drücken eines Auswahlbuttons wird die Methode `registriereUIAktionen` aufgerufen. Innerhalb dieser Methode wird jeweils die Methode `datumWurdeGeaendert()` (noch zu implementieren), bzw. `vorstellungWurdeAusgewaehlt()` aufgerufen. Diese wiederum ruft `informiereUeberAenderung...()` im jeweiligen Beobachtbar auf, welche die zugehörige Methode `reagiereAufAenderung...()` des Beobachters aufruft. Letzteres ist, wie gesagt, ein Interface, implementiert durch `KassenWerkzeug`. Innerhalb der jeweilige implementierenden Methoden erfolgt die Reaktion des Beobachters: `setzeTagesplanFuerAusgewaehltesDatum()`, bzw. `setzeAusgewaehlteVorstellung()`. Der Platzplan wird dann durch das `PlatzVerkaufsWerkzeug` aktualisiert; es wird von keinem Kontextwerkzeug beobachtet.

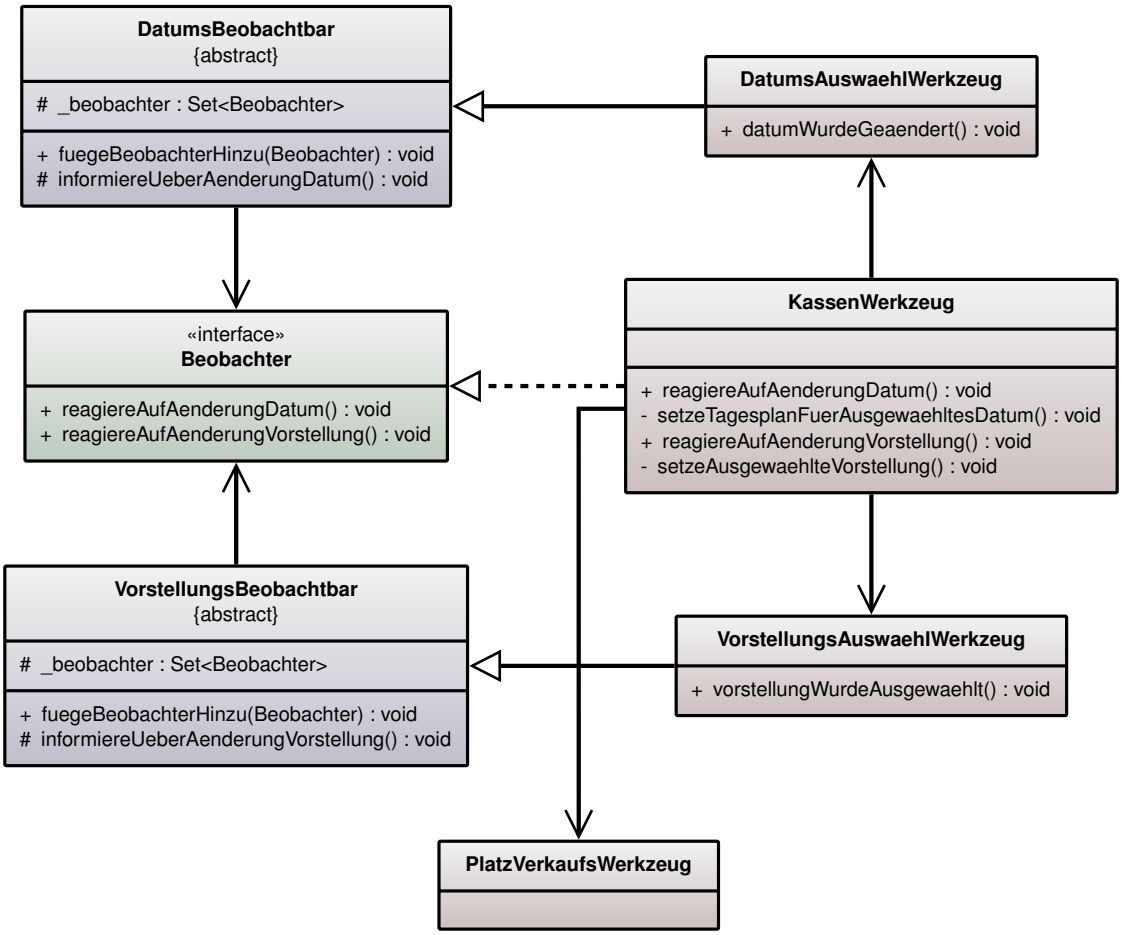

Abb. 1: UML-Klassendiagramm einiger Werkzeuge des Projektes Kinoticketverkauf.

In diesem Lösungsvorschlag erkennt das Kontextwerkzeug das aktivierte Subwerkzeug an den Methodennamen. Alternativ kann aber auch ein spezifischer Parameter von `informiereUeberAenderung()` an mehrere überladene `reagiereAufAenderung()`-Methoden übergeben werden, z. B. ein `Datum` oder eine `Vorstellung`. Eigentlich sollte es keinen Unterschied machen ob ähnliche Methoden ähnlich aber unterscheidbar benannt werden, oder ein Methode überladen wird.
Das `KassenWerkzeug` (der Beobachter) kann die beobachteten Werkzeuge (die Beobachtbaren) anhand der aufgerufenen Methode unterscheiden. Welche Methode aufgerufen wird, wird mittels statischer Polymorphie (über Type Matching) determiniert. Die Informationen über den neuen Zustand des sich geänderten Werkzeugs erhält der Beobachter (das `KassenWerkzeug`) anhand der als Parameter übergebenen Entität. Somit kann der Beobachter die neuen Werte direkt anhand der öffentlichen Methoden der Werkzeuge auslesen. Er tut dies innerhalb `setzeTagesplanFuerAusgewaehltesDatum` und `setzeAusgewaehlteVorstellung`.

Ein alternativer Lösungsansatz benutzt nur eine Beobachtbar-Klasse mit zwei unterschiedlichen `informiereUeberAenderung`-Methoden, welche dann die (überladene) `reagiereAufAenderung`-Methode mit der jeweils passenden Signatur im Interface aufruft (Abb: 2). Diese Implementation ist zwar etwas schlanker, dafür aber kryptischer und unflexibler. Die Liste der Beobachter ist jetzt für beide Subwerkzeuge identisch. Soll zu einem späteren

Zeitpunkt, aus was für einem Grund auch immer, ein oder mehrere weiterer Beobachter hinzu kommen, welche nicht beide Subwerkzeuge beobachten, muss ein weiteres Set hinzugefügt werden. Vermutlich ist Schlankheit vs. Flexibilität eine häufig vorkommende Entscheidung in der Softwarentwicklung.
Wir werden uns vermutlich für diese zweite Lösung entscheiden.

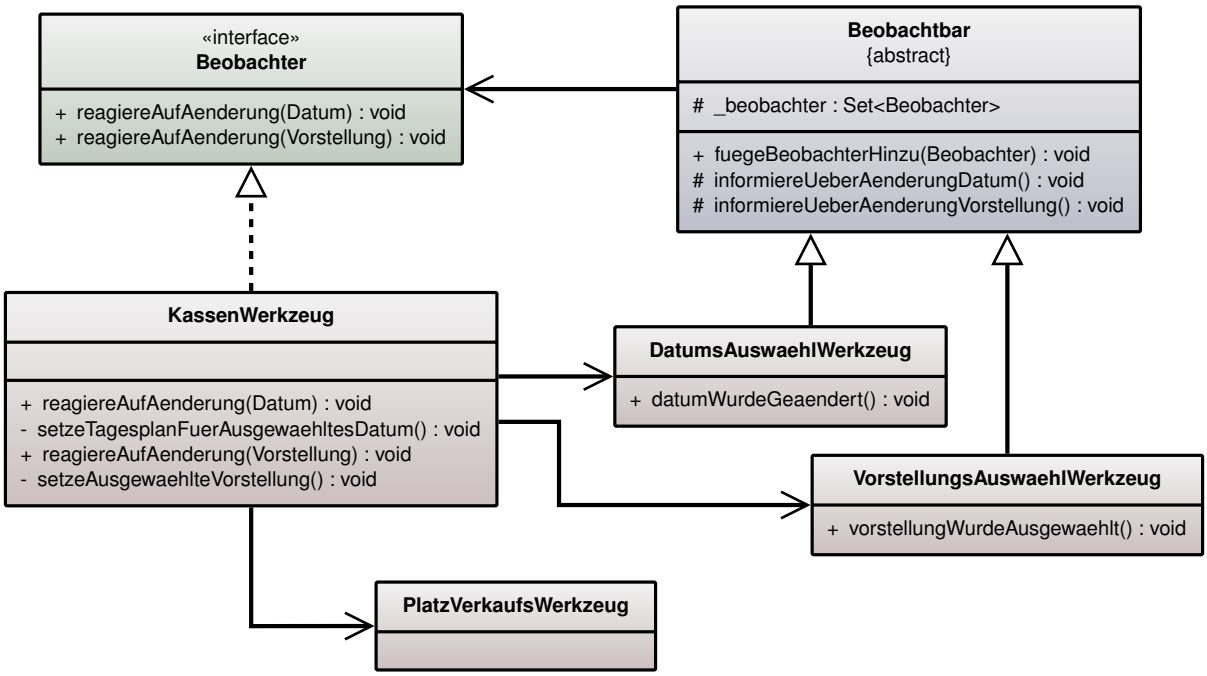

Abb. 2: UML-Klassendiagramm des alternativen Lösungsansatzes.

Implementation

Für die Implementation wurde der zweite Lösungansatz umgesetzt und verbessert (Abb: 3). Jetzt werden spezifische Enum-Objekte (`DATUMSAENDERUNG` oder `VORSTELLUNGSAUSWAHL`) als Parameter an die Methode `informiereUeberAenderung` des Beobachtbars von den beiden Subwerkzeugen übergeben und an die Methode `reagiereAufAenderung` der implementierenden Klasse der Beobachters, also dem `KassenWerkzeug`, weitergereicht. Letzter erzeugt dann die zu dem Parameter (abhängig von dem auslösenden Ereignis) passende Reaktion: `setzeTagesplanFuerAusgewaeltesDatum` oder `setzeAusgewaehlteVorstellung`.

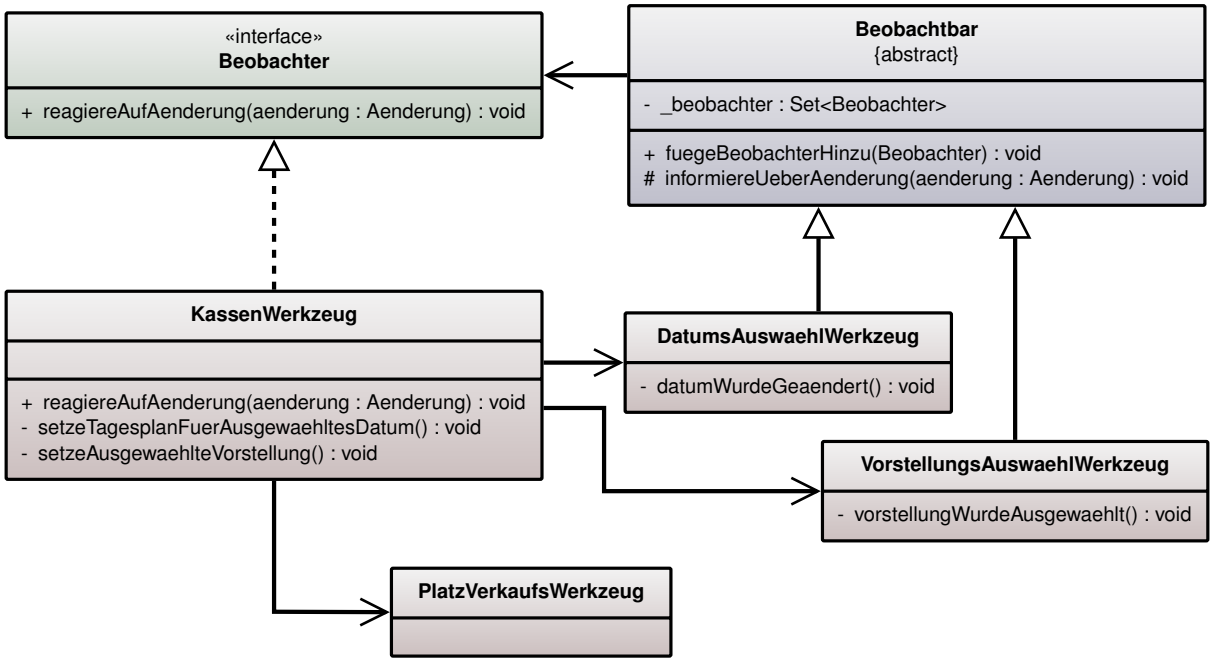

Abb. 3: UML-Klassendiagramm der Implementation.

Neu implementiert wurden die Klassen `Beobachter`, `Beobachtbar` (befinden sich beide im `startup`-Packet) und die Enum-Klasse `Aenderung`. Verändert, bzw neu implementiert wurden die Methoden:

- `reagiereAufAenderung` im `KassenWerkzeug`
- `vorstellungWurdeAusgewaehlt` im `VorstellungsAuswaehlWerkzeug`
- `datumWurdeGeandert` im `DatumAuswaehlWerkzeug`
- `weiterButtonWurdeGedrueckt` im `DatumAuswaehlWerkzeug`
- `zurueckButtonWurdeGedrueckt` im `DatumAuswaehlWerkzeug`

Das Hinzufügen vom `KassenWerkzeug` als Beobachter erfolgte in dessen Konstruktor. Die Anwendung funktioniert wie gefordert und alle Testklassen laufen.