3 Properties, Bindings und JavaFX-Collections

Dieses Kapitel beschäftigt sich mit wichtigem Basiswissen für das Verständnis der JavaFX-Elemente. Nachdem wieder die ersten Grundlagen durch das Lesen eines Abschnitts im Lehrbuch gelegt wurden, beschäftigen wir uns noch etwas intensiver mit dem Entwurfsmuster Observer (Beobachter) sowie mit Bindings in Form von Kopplungsnetzen.

3.1 Grundlagen

Zuerst sollten Sie sich einen ersten Eindruck über Properties, Bindings und JavaFX-Collections verschaffen.

Lesen Sie dazu den Abschnitt 4.2 des Lehrbuchs.

Zusammenfassung

Eine Property kapselt einen Wert wie z.B. einen String oder eine Zahl des Typs int oder double und ermöglicht, außer dem Lesen und Verändern des Werts auch Beobachter (oder Lauscher) an der Property anzumelden, die immer dann benachrichtigt werden, wenn sich der in der Property gekapselte Wert ändert. Eine spezielle Variante der Properties sind ReadOnlyProperties. Eine ReadOnlyProperty lässt sich über einen ReadOnlyWrapper beschaffen. Der im ReadOnlyWrapper gekapselte Wert kann über den ReadOnlyWrapper verändert werden. Über die ReadOnlyProperty kann der Wert aber nur gelesen und beobachtet, jedoch nicht verändert werden.
Mit Bindings können die Werte von Properties gekoppelt werden, so dass eine Wertänderung bei der einen Property eine entsprechende Wertänderung bei der anderen Property bewirkt. Es gibt unidirektionale und bidirektionale Bindings. Im unidirektionalen Fall kann man den Wert der einen Property ändern, worauf sich auch der Wert in der zweiten Property ändert. Der Versuch, den Wert der zweiten Property aber direkt zu ändern, löst eine Ausnahme aus. Bei der bidirektionalen Kopplung kann man den Wert der einen Property ändern, wodurch sich der Wert der anderen Property dann ebenfalls ändert.

Die JavaFX-Collections sind Collections wie Listen und HashMaps mit einer Property-Charakteristik. Das heißt, dass man an den Collections Beobachter anmelden kann, um sich über Änderungen wie das Hinzufügen, Ersetzen, Entfernen oder Vertauschen von Elementen benachrichtigen zu lassen.

Übungsaufgaben

3.1 Warum ist eine Property ein gutes Beispiel des Information-Hiding-Prinzips, das man u.a. in der objektorientierten Programmierung propagiert?

3.2 Im Text des Lehrbuchs wurde beschrieben, dass viele Eigenschaften der JavaFX-Elemente über Properties repräsentiert werden. Ein Beispiel für eine ReadOnlyProperty ist die Anzahl der Zeichen eines Texteingabefelds. Versuchen Sie zu begründen, warum wohl diese Property vom Typ ReadOnly ist!

3.2 Entwurfsmuster Observer

Entwurfsmuster

Zu Beginn des Abschnitts 4.2 wurde das Entwurfsmuster Observer (Beobachter) erwähnt. Entwurfsmuster sind auf Erfahrung basierende Vorschläge, wie bestimmte Aufgabenstellungen in Software umgesetzt werden können. Entwurfsmuster haben ein höheres Abstraktionsniveau als Klassenbibliotheken, da Entwurfsmuster keinen Programmcode darstellen, sondern nur beschreiben, wie Klassen und Schnittstellen in einer bestimmten Situation zusammenwirken sollen. Die konkreten Klassen und Schnittstellen mit ihren Methoden sind aber abhängig von der jeweiligen Anwendung. Die in der Beschreibung des Entwurfsmusters verwendeten Klassen und Schnittstellen sowie die verwendeten Methoden sind deshalb nur beispielhaft zu verstehen.

Entwufsmuster Beobachter (Observer)

Natürlich werden solche Entwurfsmuster mit Hilfe von Klassendiagrammen beschrieben. Das Klassendiagramm zum Entwurfsmuster Observer ist in Abbildung 3.1 zu sehen.

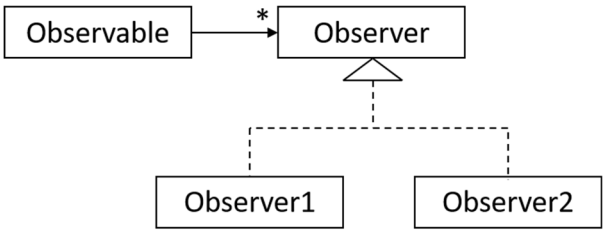

Abbildung 3.1: Klassendiagramm zum Entwurfsmuster Observer (Beobachter)

Observer ist eine Schnittstelle mit mindestens einer Methode, die beim Eintritt eines bestimmten Ereignisses aufgerufen werden soll. Observer1 und Observer2 sind stellvertretend zwei Klassen, welche die Observer-Schnittstelle implementieren und entsprechend Implementierungen für die Methode(n) der Observer-Schnittstelle beinhalten. Observable ist eine Klasse, die sich beobachten lässt (d.h. etwas Beobachtbares). Dazu können mehrere Objekte des Typs Observer an Observable angemeldet werden, wobei sich das Observable-Objekt alle angemeldeten Objekte merken muss, da es diese bei späterem Eintreten des interessierenden Ereignisses benachrichtigen muss.

Entwurfsmuser Beobachter bei der Ereignisbehandlung eines Buttons

Ganz konkret haben wir das Observer-Muster schon bei der Ereignisbehandlung und im Zusammenhang mit Properties kennen gelernt. Bei der Ereignisbehandlung sieht es durch die Methode setOnAction so aus, also ob zu einem Zeitpunkt höchstens ein Beobachter an einem Button angemeldet sein könnte (die Methode heißt ja nicht addOnAction). Ohne dies näher zu begründen, können aber auch hier mehrere Beobachter angemeldet werden, was man in der Praxis jedoch selten braucht. Mit den Klassen aus der Musterlösung zu Übungsaufgabe 2.6 sieht das Klassendiagramm für die Ereignisbehandlung damit so aus wie in Abbildung 3.2.

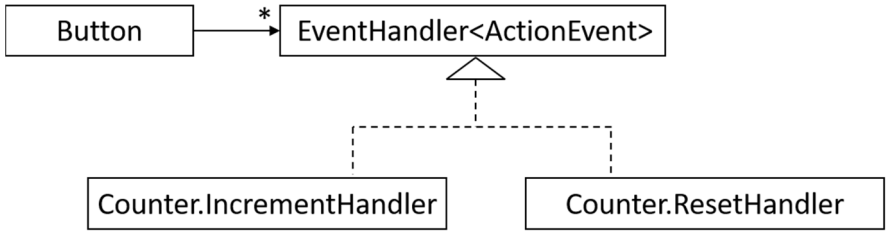

Abbildung 3.2: Klassendiagramm zur Ereignisbehandlung

Schließlich zeigt Abbildung 3.3 das entsprechende Klassendiagramm für das Property-Beispiel aus Listing 4.4 des Lehrbuchs.

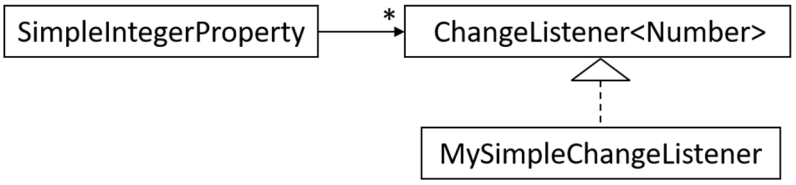

Abbildung 3.3: Klassendiagramm für Property

Zusammenfassung

Beim Entwurfsmuster Beobachter (Observer) können Beobachterobjekte an einem beobachtbaren Objekt angemeldet werden. Die Beobachter müssen eine bestimmte Schnittstelle implementieren. Tritt im beobachtbaren Objekt eine bestimmte Zustandsänderung ein, werden alle angemeldeten Beobachter durch Aufruf einer der Methoden der implementierten Schnittstelle über das Eintreten der Zustandsänderung informiert. Die Ereignisbehandlung in grafischen Benutzeroberflächen wird in der Regel nach diesem Entwurfsmuster gestaltet. Auch bei den Properties wird dieses Entwurfsmuster eingesetzt.

Übungsaufgaben

3.3 Geben Sie ein Klassendiagramm für das Programm aus Listing 4.2 an, das ebenfalls ein Beispiel für die Nutzung des Observer-Musters darstellt!

3.4 Schreiben Sie eine einfache eigene Implementierung einer Property mit einem Beispiel wie in Listing 4.4 des Lehrbuchs!

3.3 Kopplungsnetze

Vergleich mit Excel

Im Abschnitt 4.2.2 des Lehrbuchs wird das Thema Kopplungsnetze bereits angesprochen und mit der Tabellenkalkulation wie z.B. Excel verglichen. Der Wert eines Felds in Excel kann durch eine Formel von einem oder mehreren anderen Feldern abhängig sein. In JavaFX können Kopplungsnetze auf unterschiedliche Arten programmiert werden. Es gibt eine sogenannte High-Level- und eine Low-Level-Schnittstelle. Wir konzentrieren uns hier ausschließlich

High Level API und Low Level API

auf die High-Level-Schnittstelle, die wiederum zwei unterschiedliche Zugänge bietet. Wir betrachten zunächst die sogenannte Fluent API (Application Programming Interface) und beginnen mit einem Beispiel: Wenn num1 und num2 SimpleIntegerProperties sind, dann liefert der Ausdruck

Fluent API

```
num1.add(num2)
```

ein NumberBinding-Objekt zurück, das auf mehrere Arten verwendet werden kann. Zum einen kann man natürlich über eine Methode den Wert auslesen, der sich als Summe der aktuellen Werte von num1 und num2 ergibt. Zum anderen kann man einen Listener an diesem NumberBinding anmelden, der immer dann benachrichtigt wird, wenn sich die Summe ändert, was immer dann der Fall ist, wenn sich der Wert von num1 oder num2 verändert hat. Schließlich kann man eine weitere Operationsverkettung auf das NumberBinding-Objekt anwenden, was dann eine neues NumberBinding-Objekt zurückliefert. Wenn also num3 und num4 auch SimpleIntegerProperties sind, dann liefert der Ausdruck

```
num1.add(num2).add(num3).add(num4)
```

ein NumberBinding-Objekt für die Summe der Werte der vier Properties. Außer add gibt es beispielsweise auch die Methoden subtract, multiply und divide mit offensichtlicher Bedeutung. Diese Methoden sind alle mehrfach überladen. Die Parameter dieser Methoden können primitive Datentypen sein, was bedeutet, dass in einem solchen Fall ein Ausdruck mit einer Konstanten definiert wird wie in folgendem Beispiel:

```
num1.multiply(2)
```

Die Parameter können aber auch beobachtbare Objekte sein. Hier kommen Properties in Frage wie z.B. in den oben gezeigten Beispielen. Wie oben erwähnt sind auch NumberBindings beobachtbar, so dass auch sie als Parameter vorkommen können. Ein Binding, das die Werte der vier Properties num1 bis num4 z.B. in der Form (num1+num2)*(num3+num4) verknüpft, kann deshalb so geschrieben werden:

```
num1.add(num2).multiply(num3.add(num4))
```

Methoden wie z.B. greaterThan oder lessThan liefern kein NumberBinding, sondern ein BooleanBinding zurück. BooleanBindings lassen sich über and, or und not koppeln. Weiterhin gibt es auch ein StringBinding, das zum Beispiel die Operation concat zur Konkatenation von Strings besitzt.

Statische Methoden der Klasse Bindings

Neben der Fluent API gibt es eine zweite Variante der High-Level-Schnittstelle zur Programmierung von Kopplungsnetzen. Diese besteht aus mehreren statischen Methoden der Klasse Bindings. Diese statischen Methoden sind beispielsweise add, subtract, multiply, divide, greaterThan, lessThan, and, or usw. Bei der Fluent API hatten zweistellige Operationen (wie z.B. eine Addition) nur einen Parameter für den zweiten Operanden, denn der erste Operand war das Objekt, auf das die Methode angewendet wurde. Bei

den statischen Methoden der Klasse Bindings werden für zweistellige Operationen beide Operanden als Parameter übergeben. Der obige Ausdruck (num1+num2)*(num3+num4) lässt sich unter Nutzung der statischen Methoden von Bindings somit auch so schreiben:

```
Bindings.multiply(Bindings.add(num1, num2),
                  Bindings.add(num3, num4))
```

Nach meiner persönlichen Auffassung ist diese Schreibweise übersichtlicher, wenn auch wegen der Nutzung statischer Methoden nicht wirklich objektorientiert. Theoretisch ist es möglich, beide Varianten zu vermischen, was ich nicht unbedingt empfehlen würde:

```
Bindings.multiply(num1.add(num2), num3.add(num4))
```

Ein Binding kann nicht nur in wieder anderen Bindings benutzt werden (wie oben demonstriert), sondern es ist auch möglich eine Property an ein solches Binding zu koppeln. Die Wirkung sollte offensichtlich sein. Im Folgenden sei num5 eine weitere SimpleIntegerProperty. Dann könnte man schreiben:

```
NumberBinding nb = num1.add(num2).multiply(num3.add(num4));
num5.bind(nb);
```

Eine bidirektionale Kopplung einer Property mit einem Binding ist natürlich nicht möglich. Wie sollen sich die Werte des Bindings ändern, wenn der Wert der Property verändert wird? Durch den Parametertyp von bindBidirectional wird eine bidirektionale Kopplung verhindert. Der folgende Ausdruck erzeugt einen Syntaxfehler:

```
num5.bindBidirectional(nb);
```

Zusammenfassung

Über Kopplungsnetze können in JavaFX die Werte mehrerer Properties durch Operationen verknüpft werden. Das Ergebnis einer solchen Verknüpfung ist ein NumberBinding, BooleanBinding oder StringBinding, von dem nicht nur der aktuelle Wert erfragt werden kann, sondern das beobachtbar ist (d.h. an das Listener angemeldet werden können). Kopplungsnetze lassen sich über eine High-Level- oder Low-Level-Schnittstelle programmieren. Die High-Level-Schnittstelle hat wiederum zwei Varianten: zum einen die Fluent API und zum anderen zahlreiche statische Methoden der Klasse Bindings. Es ist möglich, Properties an Bindings zu koppeln, allerdings nur unidirektional.

Übungsaufgaben

3.5 Geben Sie mit Hilfe der Fluent API ein BooleanBinding für diesen Ausdruck an: num1 > num2 * num3 && num2 > 0 (num1, num2 und num3 seien SimpleIntegerProperties)!

3.6 Geben Sie mit Hilfe statischer Methoden der Klasse Bindings ein BooleanBinding für diesen Ausdruck an: num1 > num2 * num3 && num2 > 0 (num1, num2 und num3 seien SimpleIntegerProperties)!