Problem 1: ObservableValueBase

Im Projekt 03_Observer2 finden Sie im Paket `patterns.observer.list` ein kleines Beispiel, in welchem das Observer-Pattern mit der Klasse `Observable` und dem Interface `Observer` implementiert ist.

Als konkrete Anwendung wird eine Observable Liste implementiert, in der Elemente vom Typ E eingefügt und gelöscht werden können. Bei jeder Änderung werden alle Observer benachrichtigt. Als Beispiele sind folgende Observer implementiert:

- `ConsoleObserver`: Gibt aus, dass sich die Liste geändert hat.
- `AdderObserver`: Gibt die Summe der in der Liste enthaltenen Zahlen aus (dieser Observer darf natürlich nur in einer Liste vom Typ `ObservableList<Integer>` registriert werden).

Im Hauptprogramm `ObserverTest` werden Observer registriert und die Liste danach geändert.

```
public static void main(String[] args) {
       ObservableList<Integer> list = new ObservableList<>();
       ConsoleObserver co = new ConsoleObserver();
       AdderObserver ao = new AdderObserver();
       list.addObserver(co);
       list.addObserver(ao);
       list.addValue(17);
}
```

Wird dieses Programm ausgeführt, so erscheint folgende Ausgabe:

```
ConsoleObserver: List has changed
AdderObserver: new sum is 17
```

Ihre Aufgabe besteht darin, die Klasse `ObservableList` als Erweiterung der Klasse `ObservableValueBase` zu implementieren.

Aufgaben, Vorgehen:

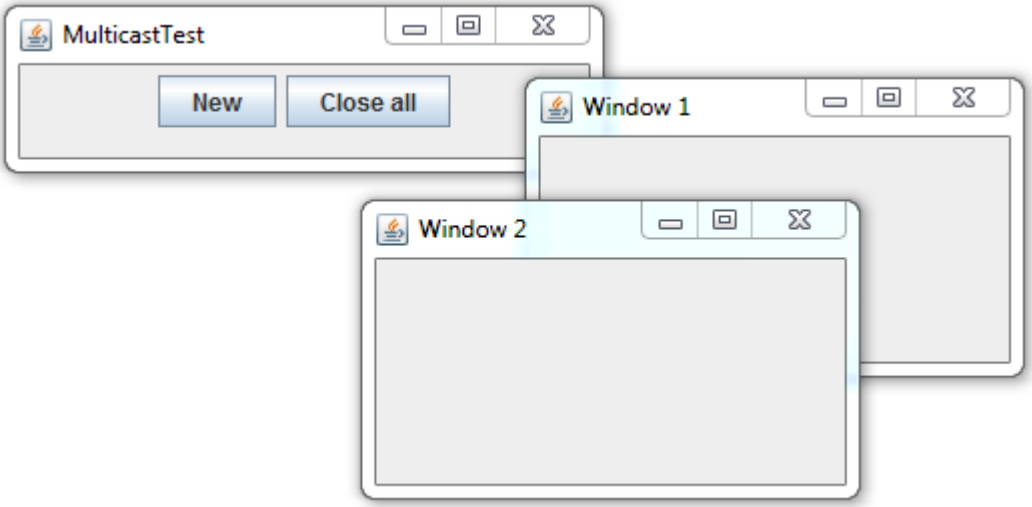
1. Problemanalyse (alleine oder zu zweit):
   - Studieren Sie die API Dokumentation von `javafx.beans.value.ObservableValueBase`.
   - Leiten Sie die Klasse `ObservableList` neu von der Klasse `ObservableValueBase<List<E>>` ab (die gegebene Klasse `Observable` können Sie zusammen mit dem gegebenen Interface `Observer` löschen). Korrigieren Sie die Fehler und implementieren Sie die fehlenden Methoden.
   - Verwenden Sie `javafx.beans.value.ChangedListener<List<Integer>>` als Basis-Klasse für den `AdderObserver`. Registrieren Sie vorerst nur diesen Observer in der Liste. Er wird bei jeder Änderung aufgerufen; d.h. falls der alte von `getValue` zurückgegeben Wert ungleich dem neuen von `getValue` zurückgegebenen Wert ist (Vergleich wird mit `equals` gemacht). Verhält sich Ihre Implementierung korrekt?
     Falls nein, erklären Sie warum nicht und korrigieren Sie Ihre Implementierung.
   - Implementieren Sie den `ConsoleObserver` als `javafx.beans.InvalidationListener` und registrieren Sie nur diesen Observer in der Liste. Dieser Observer sollte dann im Testlauf nur zweimal aufgerufen werden, nämlich nur, wenn seit dem letzten Aufruf die Listen-Werte einmal abgefragt wurden (passiert bei uns durch `printList(list)`). Verhält sich Ihre Implementierung korrekt?
     Falls nein, erklären Sie warum nicht und korrigieren Sie Ihre Implementierung.

2. Expertenrunde:
   Fassen Sie Ihre Beobachtungen zusammen und halten Sie fest, wie die Klasse `javafx.beans.value.ObservableValueBase` verwendet werden muss.

3. Präsentation:
   Sie haben 5 – 7 Minuten Zeit, um Ihre Resultate zu präsentieren. Eine kurze Diskussion mit der Klasse rundet Ihre Präsentation ab.
   Gliedern Sie Ihre Präsentation in die 3 Abschnitte:
   a) Problembeschreibung (anhand des konkreten Beispiels aus dieser Aufgabe)
      Denken Sie daran, dass Ihre Mitstudierenden die Aufgabe nicht studiert haben!
   b) Problemlösung
   c) Verallgemeinerung / Regel

Problem 2: Observer & Ressourcen

Im Projekt 03_Observer2 finden Sie im Paket `patterns.observer.memory` ein kleines Beispiel, das aus dem Buch *Core Java* entnommen ist. Das Programm erlaubt es, mehrere Fenster zu öffnen und diese über die Schaltfläche *Close all* alle wieder zu schliessen.



Die kleinen Fenster sind in der inneren Klasse `SimpleFrame` implementiert. Um das Problem zu verschärfen, wird in jeder Instanz der Klasse `SimpleFrame` ein Byte-Array der Grösse 256MB angelegt.

Führen Sie das Programm aus. Öffnen Sie mehrere Fenster und schliessen Sie diese mit *Close all* und wiederholen Sie diesen Vorgang mehrmals. Was können Sie dabei beobachten? Überlegen Sie sich, was das Problem dieses Programms ist.

Tipps:
- Geben Sie in der Methode `SimpleFrame.actionPerformed` einen Text aus, damit Sie sehen welche Action-Listeners ausgeführt werden (z.B. ein Text in dem die Fensternummer ausgegeben wird).
- Implementieren Sie in der Klasse `SimpleFrame` die Methode `public void finalize()`. Diese Methode wird aufgerufen, wenn ein Objekt nicht mehr erreichbar ist und vom Garbage-Collector weggeräumt werden kann. Mit der finalize-Methode wird diesem Objekt ein letzter Wunsch gewährt. Sie können so erkennen, welche Objekte vom Garbage-Collector weggeräumt werden.
- Mit dem Tool JConsole können Sie den von einem Java-Programm verwendeten Speicher verfolgen und auch explizit den Garbage-Collector aufrufen.

Aufgaben, Vorgehen:

1. Problemanalyse (alleine oder zu zweit):
   - Beschreiben Sie das Problem und wie es entsteht.
   - Wie kann das Problem behoben werden? Erarbeiten Sie einen Lösungsvorschlag.

2. Expertenrunde:
   Leiten Sie aus Ihren Beobachtungen eine Regel für das Observer-Pattern ab. Was muss beim Observer beachtet werden? Wann gibt es Probleme? Treten diese Probleme bei JavaFX auch auf?

3. Präsentation:
   Sie haben 5 – 7 Minuten Zeit, um Ihre Resultate zu präsentieren. Eine kurze Diskussion mit der Klasse rundet Ihre Präsentation ab.
   Gliedern Sie Ihre Präsentation in die 3 Abschnitte:
   a) Problembeschreibung (anhand des konkreten Beispiels aus dieser Aufgabe)
      Denken Sie daran, dass Ihre Mitstudierenden die Aufgabe *nicht* studiert haben!
   b) Problemlösung
   c) Verallgemeinerung / Regel

PS
In der aktuellen Version des Buches „Core Java“ ist dieses Problem behoben worden.

Problem 3: Nebeneffekte von Änderungen

Im Projekt 03_Observer2 finden Sie im Paket `patterns.observer.once` ein kleines Beispiel, in dem das Observer-Pattern mit der Klasse `Observable` und dem Interface `Observer` implementiert ist.
Als konkrete Anwendung wird ein `Sensor` implementiert, der einen Wert vom Typ `double` enthält. Dieser Wert kann mit der Methode `getValue` abgefragt und mit der Methode `setValue` verändert werden. Bei jeder Änderung werden alle Observer benachrichtigt. Als Beispiel ist folgender Observer implementiert:
- PrintObserver: Gibt den neuen Wert des Sensors auf der Konsole aus

Ihre Aufgabe ist es, einen weiteren Observer zu untersuchen. Dieser Observer meldet sich innerhalb der update-Operation vom Subjekt ab:

```
public class OnceObserver implements Observer {
	@Override
	public void update(Observable source){
		System.out.println("Once Observer called, bye...");
		source.removeObserver(this);
	}
}
```

Das Testprogramm registriert vier Observer und ändert den Wert des Sensors. Führen Sie dieses Programm aus.

```
public static void main(String[] args) {
	Sensor s = new Sensor(20);
	s.addObserver(new PrintObserver("Printer 1"));
	s.addObserver(new OnceObserver());
	s.addObserver(new PrintObserver("Printer 2"));
	s.addObserver(new PrintObserver("Printer 3"));
	s.setValue(22);
	s.setValue(30);
	s.setValue(25);
	s.setValue(22);
}
```

Aufgaben, Vorgehen:

1. Problemanalyse (alleine oder zu zweit):
   Überlegen Sie sich, was für ein Problem entsteht, wenn der `OnceObserver` ausgeführt wird, d.h. wenn sich ein Observer in einer update-Operation abmeldet.
   Tipp: Was passiert mit Iteratoren, deren Liste verändert wurde.

   Tritt das Problem auch auf, wenn
   a) der `OnceObserver` als letzter Observer im Sensor registriert wird?
   b) als vorletzter Observer im Sensor registriert wird?
   c) mehrere `OnceObserver` registriert werden?
   d) anstelle einer for-Schleife die forEach Methode (mit Lambda-Ausdruck) verwendet wird?

1. Expertenrunde:
   Wie kann das Problem behoben werden? Erarbeiten Sie einen Lösungsvorschlag, wie das Problem in der Klasse `Observable` korrigiert werden kann. Falls Sie keine Idee haben, dann schauen Sie doch in der Klasse `java.util.Observable` oder bei der Implementierung des Notifikationsmechanismus von JavaFX (`com.sun.javafx.binding.ExpressionHelper.Generic`) nach.

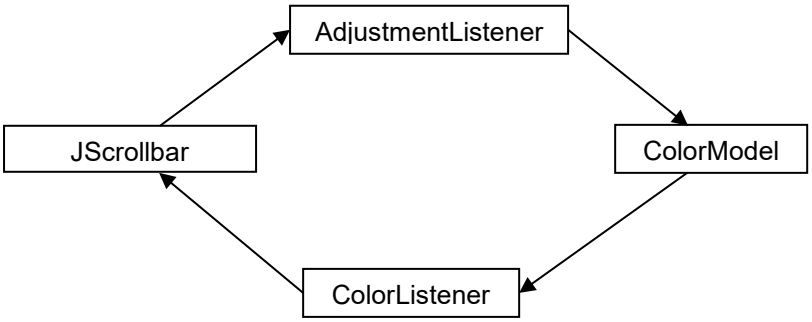
2. Präsentation:
   Sie haben 5 – 7 Minuten Zeit, um Ihre Resultate zu präsentieren. Eine kurze Diskussion mit der Klasse rundet Ihre Präsentation ab.
   Gliedern Sie Ihre Präsentation in die 3 Abschnitte:
   a) Problembeschreibung (anhand des konkreten Beispiels aus dieser Aufgabe)
      Denken Sie daran, dass Ihre Mitstudierenden die Aufgabe *nicht* studiert haben!
   b) Problemlösung
   c) Verallgemeinerung / Regel

Problem 4: Zyklische Abhängigkeit

In der ColorPicker-Applikation aus Übung 1 gibt es einen Scrollbar, mit dem man einen Farbkanal der eingestellten Farbe verändern kann. Natürlich existieren auch andere GUI-Elemente wie Menu-Items oder Eingabefelder, die uns aber im Moment nicht interessieren.

Wird der Wert des Scrollbars verändert, so wird ein Event an alle im Scrollbar registrierten Listener verschickt. Einer dieser Listener wird daraufhin die Farbe im ColorModel nachführen.

Das ColorModel wiederum verschickt einen ColorEvent an alle registrierten ColorListeners. Einer dieser ColorListener setzt dann wiederum den Wert des Scrollbars entsprechend der ausgewählten Farbe.



Im Projekt 03_Observer2 finden Sie im Paket `patterns.observer.cycle` ein kleines Beispiel, das dieses Problem illustriert. Das `ColorModel` sowie die Klasse `RedScrollbar` erweitern die Klasse `Observable`. Das Testprogramm erzeugt ein `ColorModel` und einen `RedScrollbar`, der den Rot-Wert der eingestellten Farbe darstellt. Im `Scrollbar` wird ein `Listener` registriert, der die Farbe im Modell setzt. Im Programm wird dann zuerst die Farbe auf grau gesetzt und danach wird der Wert im Scrollbar auf 44 gesetzt. Der Wert des Scrollbars und der Rotanteil der eingestellten Farbe sollten dabei immer identisch sein.

```
model.setColor(Color.gray);
System.out.println("model red value = " + model.getColor().getRed());
System.out.println("scrollbar value = " + sb.getValue());
sb.setValue(44);
System.out.println("model red value = " + model.getColor().getRed());
System.out.println("scrollbar value = " + sb.getValue());
```

Aufgaben, Vorgehen:

1. Problemanalyse (alleine oder zu zweit):
   - Beschreiben Sie, was für ein Problem diese ColorPicker-Applikation hat. Zeichnen Sie ein Sequenzdiagramm, welches das Problem verdeutlicht.
   - Wie kann das Problem behoben werden? Erarbeiten Sie zwei Lösungsvorschläge.

2. Expertenrunde:
   Leiten Sie aus Ihren Beobachtungen eine Regel für das Observer-Pattern ab. Was muss beachtet werden? Wann gibt es Probleme?

3. Präsentation:
   Sie haben 5 – 7 Minuten Zeit, um Ihre Resultate zu präsentieren. Eine kurze Diskussion mit der Klasse rundet Ihre Präsentation ab.
   Gliedern Sie Ihre Präsentation in die 3 Abschnitte:
   a) Problembeschreibung (anhand des konkreten Beispiels aus dieser Aufgabe)
      Denken Sie daran, dass Ihre Mitstudierenden die Aufgabe *nicht* studiert haben!
   b) Problemlösung
   c) Verallgemeinerung / Regel

Problem 5: Was ist das Modell, wie wird es verändert?

Für die Implementation des Colorpicker ist folgender Vorschlag eingebracht worden: Anstelle der Farbe vom Typ Color werden im Modell die drei Farbkanäle rot, grün und blau separat gespeichert. Wird eine Farbe gesetzt, so müssen die drei Methoden *setRed*, *setGreen* und *setBlue* aufgerufen werden. Der Vorteil dieser Lösung ist, dass die Scrollbars und die Textfelder, die ja nur je einen Farbkanal darstellen, diesen direkt ändern können und auch nur auf Änderungen ihres Farbkanal-Modells achten müssen.

Die Farbfläche muss dann natürlich auf drei Modelle reagieren, um die Mischung der drei Farbkanäle korrekt wiederzugeben. Umgekehrt müssen RadioButtons, die z.B. Gelb als Voreinstellung anbieten, alle drei Modelle verändern.

Im Projekt 03_Observer2 finden Sie im Paket `patterns.observer.multimodel` ein kleines Beispiel, in dem dieser Ansatz realisiert worden ist.

Aufgaben, Vorgehen:

1. Problemanalyse (alleine oder zu zweit):
   Überlegen Sie sich, was für ein Problem dieser Ansatz der drei Modelle hat.
   In der Klasse ColorTest wird folgendes Szenario durchgespielt: Die Farbe wird mit Schwarz initialisiert und dann auf Weiss gesetzt. Zusätzlich ist ein Rot-Allergie-Listener registriert, der die Farbe Grau einstellt, sobald die Farbe Rot gewählt worden ist.
   Welche Farbe ist am Schluss eingestellt?

2. Expertenrunde:
   Wie kann das Problem behoben werden?
   Was für Schlüsse können Sie aus Ihren Beobachtungen ziehen?

3. Präsentation:
   Sie haben 5 – 7 Minuten Zeit, um Ihre Resultate zu präsentieren. Eine kurze Diskussion mit der Klasse rundet Ihre Präsentation ab.
   Gliedern Sie Ihre Präsentation in die 3 Abschnitte:
   a) Problembeschreibung (anhand des konkreten Beispiels aus dieser Aufgabe)
      Denken Sie daran, dass Ihre Mitstudierenden die Aufgabe *nicht* studiert haben!
   b) Problemlösung
   c) Verallgemeinerung / Regel

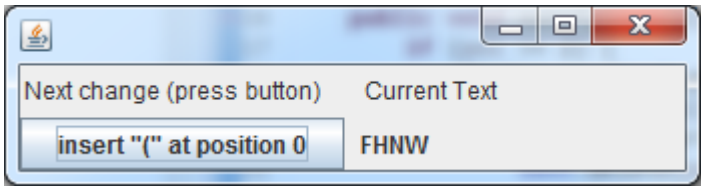
Problem 6: Kausalität von Änderungen

Bei diesem Problem betrachten wir einen einfachen Texteditor, dessen Modell mit Hilfe der Klasse `java.lang.StringBuilder` implementiert ist. Im Modell können Zeichen eingefügt und gelöscht werden. Bei jeder Änderung benachrichtigt das Modell die registrierten Observer mit der Methode `notifyInsert(int pos, char ch)` bzw. `notifyDelete(int from, int len)`.

Wir nehmen nun an, dass folgende konkrete Listener existieren:

- Ein TextListener der den Text in einem Fenster darstellt. Dieser Listener führt seine Sicht nur auf Grund der Modelländerungen nach, braucht also keine Referenz auf das Modell
- Ein CorrectionListener, der einfache Textkorrekturen vornimmt. Konkret prüft dieser Listener beim Einfügen einer schliessenden Klammer, ob die Buchstabenfolge „(“ , „c” und „)” eingegeben wurde und ersetzt diese dann durch das Zeichen „©“. Dazu löscht dieser Listener die Zeichenfolge "(c)" und fügt danach das neue Zeichen "©" ein.

Im Projekt 03_Observer2 finden Sie im Paket `patterns.observer.copyright` ein kleines Beispiel, das dieses Problem illustriert. Das Textmodell ist bereits mit dem Text „FHNW“ initialisiert, und mit jedem Klick auf die Schaltfläche wird ein neues Zeichen in das Modell eingefügt. Rechts von der Schaltfläche wird der Inhalt des TextListeners angezeigt.



Aufgaben, Vorgehen:

1. Problemanalyse (alleine oder zu zweit):
   - Beschreiben Sie das Problem dieser Konfiguration und überlegen Sie sich, wie dieses Problem zustande kommt (z.B. mit Hilfe eines Sequenzdiagrammes).
   - Vertauschen Sie die Reihenfolge der beiden Listener. Tritt dann das Problem weiterhin auf? Warum bzw. warum nicht?

2. Expertenrunde:
   Wie kann das Problem behoben werden?
   Eine offensichtliche Lösung wäre, dass der Korrekturlistener nach dem Darstellungslistener eingefügt wird, aber es könnte ja auch sein, dass mehrere Korrekturlistener registriert werden müssen. Es ist also eine generellere Lösung gesucht!

3. Präsentation:
   Sie haben 5 – 7 Minuten Zeit, um Ihre Resultate zu präsentieren. Eine kurze Diskussion mit der Klasse rundet Ihre Präsentation ab.
   Gliedern Sie Ihre Präsentation in die 3 Abschnitte:
   a) Problembeschreibung (anhand des konkreten Beispiels aus dieser Aufgabe)
      Denken Sie daran, dass Ihre Mitstudierenden die Aufgabe *nicht* studiert haben!
   b) Problemlösung
   c) Verallgemeinerung / Regel