

Übungen SW11: Eventhandling und innere Klassen

Themen: Kopplung und Kohäsion (P02), Event/Listener-Pattern (O12)
Zeitbedarf: ca. 180min.

Roland Gisler, Version 1.7.0 (FS25)

1 Event/Listener-Pattern mit `PropertyChangeListener`

1.1 Lernziele

- Funktionsweise des Event/Listener-Patterns von Java verstehen.
- Bereits vorhandene Event-Klassen (wieder-)verwenden.
- Sowohl Event-Quelle als auch Event-Empfänger implementieren können.

1.2 Grundlagen

In der Semesterwoche 04 haben wir ein einfaches Modell eines Autos mit Motor und Licht entworfen, um diese Komponenten über eine Schnittstelle **Switchable** ein- und ausschalten zu können. Dabei ging es um die Verwendung einer Schnittstelle und der Polymorphie. Diese Klassen nutzen wir als Grundlage für diese Übungen. Beachten Sie, dass Sie im Input `O12_IP_Events` einige Codefragmente zur Lösung dieser Aufgabe finden!

1.3 Aufgaben

Vorbereitung:

- a.) Machen Sie sich mit Ihrem eigenen Entwicklungsstand des Projektes von SW04 (**Switchable**, **Motor**, **Auto** und **Licht**) wieder vertraut. Sollten Sie dieses Projekt von BlueJ noch nicht in Ihr IDE-Projekt übernommen haben, machen Sie das jetzt!
- b.) Setzen Sie sich ganz bewusst mit Ihrem eigenen Code von damals auseinander und scheuen Sie sich nicht davor, Dinge zu verbessern, die Ihnen nun bewusster oder klarer geworden sind. Sie haben in den letzten Wochen sehr viel dazu gelernt!
- c.) Ein gutes Mittel zur Auseinandersetzung mit bestehendem Code kann auch die Ergänzung fehlender JavaDoc und von Unit-Tests sein (wenn nicht schon vorhanden), mit welchen Sie Ihr Verständnis bzw. die Erwartungen an den Code verifizieren können.

Hauptaufgaben:

- d.) Der Motor soll, wenn er seinen Status verändert (mindestens **EIN** und **AUS**), dies per **PropertyChangeEvent** (siehe Java API Dokumentation) mitteilen. Als Empfänger verwenden wir die Klasse **Auto** (oder als Variante: die Testanwendung, welche das Auto erzeugt und benutzt). Überlegen Sie sich welche Klassen in die Rollen als Event-Quelle und als Event-Listener schlüpfen.
- e.) Implementieren Sie als erstes den Event-Listener. Sie müssen dafür das **PropertyChangeListener**-Interface implementieren. Sie können als Event-Behandlung z.B. die Informationen aus dem Event mit **System.out.println(...)** ausgeben, oder besser: Verwenden Sie Logging! ☺

- f.) Für einen ersten Test können Sie die Listener-Methoden selber aufrufen. Dafür müssen Sie ein **PropertyChangeEvent**-Objekt erzeugen, was auch eine gute Übung ist.
- g.) Überlegen Sie sich, was wir nun auf der Event-Quelle (also dem **Motor**) alles machen müssen: Methoden zur Registrierung und Deregistrierung, eine Datenstruktur für die Listener, und eine Methode um einen Event an alle Listeners zu versenden. Verschaffen Sie sich mit Hilfe des Slides aus dem Input einen Überblick!
- h.) Als Datenstruktur verwenden Sie am besten eine **[Array]List<...>**. Achten Sie auf den korrekten Typparameter, was müssen Sie dort einsetzen?
- i.) Die Methoden zur Registrierung und Deregistrierung sind reine «Delegate»-Methoden und (beinahe) Einzeiler. Wenn wir sie professioneller implementieren, prüfen wir vor der Verwendung der Parameter, dass diese nicht **null** sind (wurde auf den Folien im Input aus Platzgründen weggelassen). Was könnte sonst (und wann und wo) passieren?
- j.) Zum Schluss implementieren wir noch die **firePropertyChangeEvent(...)**-Methode. Sie ist das unauffällige Herzstück des Ganzen: Über einen Iterator ruft sie auf allen registrierten Objekten die Methode für die Event-Behandlung auf!
- k.) Sind wir nun fertig? Wenn Sie den **Motor** ein- und ausschalten, empfangen Sie dann die erwarteten Events?

Tipp: Beim ersten Test funktioniert es sehr häufig nicht, weil man schlicht vergessen hat, einen Listener bei der Quelle zu registrieren... ☺

2 TemperaturVerlauf: Eigene Events implementieren

2.1 Lernziele

- Erste, einfache Applikation mit Eingabe, Verarbeitung und Ausgabe (EVA) auf Konsole.
- Implementation von eigenen Event-Klassen.
- Event-Listener (Behandlung) mit inneren [anonymen] Klassen implementieren.

2.2 Grundlagen

In dieser Aufgabe können wir vieles was wir bis letzte Woche gelernt haben, zu einer ersten, ganz einfachen Applikation verknüpfen: Wir wollen auf der Konsole (natürlich nur gültige) Temperaturwerte eingeben können, und diese als **Temperatur**-Objekte in der Klasse **TemperaturVerlauf** abspeichern. Die Klasse **TemperaturVerlauf** soll jedes Mal, wenn wir ein neues Minimum oder Maximum erreichen, über einen Event alle interessierten Listener sofort darüber informieren.

2.3 Aufgaben

Vorbereitung:

- a.) Erweitern und verknüpfen Sie die Aufgaben der letzten Wochen so weit miteinander, dass Sie Temperaturwerte eingeben können, und diese in der Klasse **TemperaturVerlauf** abgespeichert werden.
- b.) Wenn Sie das Programm mit der Eingabe mit «**exit**» beenden, soll eine kurze Statistik ausgegeben werden, welche im Minimum
 - die Anzahl Temperaturpunkte
 - die Durchschnittstemperatur
 - die minimale und die maximale Temperaturausgibt. Denken Sie daran: Ein grosser Vorteil der Objektorientierung ist, dass wenn die einzelnen Klassen gut vorbereitet sind (denken Sie z.B. an **toString()**), einiges einfacher wird.

Hauptaufgaben:

- c.) In diesem Beispiel wollen wir exemplarisch eigene Events definieren. Sie können selber entscheiden ob Sie eine oder zwei Eventklassen verwenden wollen. Überlegen Sie sich aber vorher, wie und wo sich das auswirkt.
Variante 1: Ein **TemperaturEvent** welcher per Attribut (Enum!) weiss, ob es sich um ein Maximum oder ein Minimum handelt.
Variante 2: Je ein **TemperaturMaxEvent** und ein **TemperaturMinEvent** welche die zwei Ereignisse getrennt signalisieren können.
- d.) Implementieren Sie die Eventklasse(n). Achten Sie darauf von **EventObject** zu erben! Einmal mehr macht es Sinn, die Klasse(n) sofort mit Unit Tests zu testen.
- e.) Implementieren Sie auf dem **TemperaturVerlauf** die Event-Quelle(n).
- f.) Ergänzen Sie Ihre Hauptanwendung mit den/dem entsprechenden Listener(n). Die Benutzer*In soll direkt nach der Eingabe eines Temperaturwertes informiert werden, wenn ein neues Minimum oder Maximum eingegeben wurde.
- g.) Je nach Variante die Sie gewählt haben, mussten Sie getrennte Listener-Methoden oder aber eine Methode (jedoch mit Fallunterscheidung) implementieren. Welche Variante finden Sie besser? Überlegen Sie die Vor- und Nachteile.
- h.) Wie haben Sie die Listener implementiert? Haben Sie an (anonyme) innere Klassen gedacht? Probieren Sie mit Hilfe des Inputs die verschiedenen Varianten aus!

3 Optionale Repetitionsaufgabe: Raumverwaltung – Teil 2 von 5

3.1 Lernziele

- Repetition bereit behandelter Themen zur Festigung und Vertiefung.

3.2 Grundlagen

Diese Zusatzaufgaben sind unabhängig von den restlichen Aufgaben dieser Woche. Sie baut aber auf der optionalen Repetitionsaufgabe der letzten Woche auf.

Optionale Aufgaben sind **nicht** Bestandteil der Besprechungen.

3.3 Aufgaben

- a.) Erstellen Sie eine Enumeration, mit welcher Stati für einen **freien**, einen **belegten** und einen **gesperrten** Raum dargestellt werden können.
- b.) Ergänzen Sie auf der Klasse **Raum** ein Attribut für den Raumstatus mit der Enumeration von a). Nur Klassen im selben Package sollen das Attribut setzen, aber alle anderen sollen es lesen können. Neue erstellte Räume sollen immer im Status '**frei**' sein.
- c.) Implementieren Sie eine Klasse **RaumVerwaltung** im selben Package wie die Klasse **Raum**, welche eine Datenstruktur verwendet, um eine Menge von **Raum**-Objekten zu speichern. Auf die Räume soll mittels der Raumnummer schnell zugegriffen werden können! Die Datenstruktur soll die Einträge implizit nach der Raumnummer sortieren.
- d.) Erzeugen Sie für erste Tests im Konstruktor der Klasse **RaumVerwaltung** einen Raum mit Nummer **603** und einer Platzanzahl von **12**. Legen Sie diesen Raum in der Datenstruktur von c) ab.
- e.) Ergänzen Sie auf der Klasse **RaumVerwaltung** eine Methode, welche anhand einer Raumnummer den entsprechenden Raum (als Objekt) zurückliefert. Überlegen Sie sich, was im Fehlerfall passieren soll, und denken Sie an die Datenkapselung. Testen Sie das gewünschte Verhalten Ihrer Klasse mit zwei Unit-Tests.