Übung zur Vorlesung Software Engineering

Wintersemester 2024/25, Matthias Sokolowski, Max Granzow

TECHNISCHE UNIVERSITÄT DARMSTADT

Übungsblatt 08: Entwurfsmuster

Beachten Sie das Archiv mit Programmcode, welches in Moodle gemeinsam mit dem Übungsblatt veröffentlicht wird.

Abgabeformat: Das Übungsblatt beinhaltet zwei Bonusaufgaben, bei denen vorrangig Quelltext bearbeitet werden soll. Reichen Sie Ihren bearbeiteten Quelltext für die Bonusaufgaben über Moodle als **einzelnes ZIP-Archiv** ein, welches beide Projekte der Aufgaben enthält. Das in Moodle veröffentlichte Archiv dient als Referenz. Zusätzlich muss in Aufgabe 2a) ein Klassendiagramm gezeichnet werden, welches Sie zusätzlich als **PDF** in Moodle hochladen. Während der Bearbeitungszeit können Sie Ihre Lösungen beliebig oft aktualisieren. Wir bewerten die zuletzt hochgeladene Lösung.

Wichtig: Der abgegebene Quelltext **muss** die beiden vollständigen Projektordner umfassen, sodass die Abgabe durch eine Entwicklungsumgebung importiert werden kann.

Hinweis: Alle Mitglieder einer Übungsgruppe sollten in der Lage sein, die gleiche Abgabe einzusehen / zu editieren.

Achtung: Dieses Übungsblatt enthält 2 Bonusaufgaben.

Abgabetermin: 17.01.2025, 14:00
Besprechung der Lösungen in den Tutorien: Woche ab dem 20.01.2025
Veröffentlichung einer Lösungsskizze: 24.01.2025

Wir wünschen Ihnen frohe Feiertage und einen guten Rutsch ins neue Jahr!

Aufgabe 1: Beobachter Muster anwenden (Bonusaufgabe: 18 Punkte)

Im Krankenhaus soll ein Alarmierungssystem eingerichtet werden, welches die Vitalwerte der Patient:innen überwacht und das Personal informiert, wenn Patient:innen Hilfe brauchen. Die Vitalwerte werden von medizinischen Geräten (beispielsweise Herzschlag Messgeräten) überwacht. Die Anzeigegeräte (beispielsweise ein Pager) informieren das medizinische Personal über Änderungen der Vitalwerte.

Diese Funktionalität soll durch das Beobachter-Muster (Oberserver Pattern) umgesetzt werden. Die medizinischen Geräte sind dabei die Subjekte und die Anzeigegeräte die Beobachter.

Sie finden im Projekt `PatientMonitoring`, welches dem Übungsblatt beiliegt, das Java Paket `de.tu_darmstadt.patient_monitoring`, welches einen initialen Entwurf enthält. Entwickeln Sie Ihre Lösung auf der Basis dieses Code Gerüsts.

Die Klasse `Indicator` definiert das Interface für den Beobachter (Observer). In der Klasse `AlarmBell` ist bereits ein konkreter Observer implementiert, welcher einen Ton abspielt, wenn ein medizinisches Gerät eine Zustandsänderung signalisiert. Die Klasse `Dashboard` implementiert eine grafische Oberfläche, auf der alle überwachten Vitalwerte aus allen Räumen des Krankenhauses angezeigt werden. Die zu überwachenden medizinischen Geräte erben bereits von der abstrakten Subjektklasse `MedicalDevice`.

Wichtig: Schreiben Sie Ihren Code so, dass **keine** Nullreferenzzugriffe (`NullPointerException`) in den von Ihnen zu bearbeiteten Abschnitten auftreten können. Auch dann nicht, wenn ein Klient Ihre Implementierung fehlerhaft benutzt.

Kompilieren/Ausführen: Das Projekt kann mittels des Terminals (Eingabeauffordungen) übersetzt und ausgeführt werden. Navigieren Sie dazu in das Projektverzeichnis und führen den Befehl

Linux/macos	`./gradlew run`
Windows	`gradlew.bat run`

aus. Sie können das Projekt auch als Gradleprojekt in gängige Entwicklungsumgebungen (bspw. Intellij IDEA, eclipse) importieren. Nutzen Sie Java Version 17 oder höher.

Das Abspielen des Alarmtons kann während der Entwicklung durch setzen der Umgebungsvariable `MUTED` verhindert werden (Beispielsweise in Linux `MUTED=1 ./gradlew run`). In diesem Fall wird der Alarmton dann durch die Ausgabe `<BEEP>` auf der Kommandozeile ersetzt.

Hinweis: Machen Sie sich vor dem Bearbeiten der folgenden Aufgaben mit dem Code-Gerüst eingängig vertraut. Identifizieren Sie dabei die bereits implementierten Komponenten und deren Zusammenhang zum Beobachter-Muster.

1a) Abstrakte Subjektklasse implementieren

Vervollständigen Sie die Implementierung der Klasse `MedicalDevice`. Ihre Implementierung muss alle für diese Klasse relevanten Elemente des Beobachter-Musters beinhalten (Siehe Folien zu Observer Pattern).

Da jedes Anzeigerät (`Indicator`) von mehreren medizinischen Geräten (`MedicalDevice`) Zustandsänderungen empfängt, soll das Subjekt beim Benachrichtigen der Beobachter sich selbst als Parameter übergeben.

Beachten Sie auch, dass das Hinzufügen und Entfernen von Beobachtern idempotent ist. Das bedeutet, dass wiederholtes Hinzufügen oder Entfernen des selben Beobachters keine Auswirkungen hat. Praktisch betrachtet bedeutet dies, dass nach wiederholtem Hinzufügen des selben Beobachters dieser bei einer Zustandsänderung nur einmalig benachrichtigt wird.

1b) Konkrete Subjekte erweitern

Identifizieren Sie alle konkreten Subjektklassen und erweitern Sie diese, sodass die hinterlegten Beobachter über alle Zustandsänderungen der Subjekte informiert werden.

1c) Hinzufügen eines weiteren konkreten Observers

Implementieren Sie einen weiteren konkreten Observer `Pager`. Legen Sie die Klasse dafür an geeigneter Stelle in der Paketstruktur an.

Pager werden an Ärzte/Ärztinnen ausgegeben, um diese schnell über Änderungen von Vitalwerten der Patient:innen zu informieren. Zur Einfachheit veranschaulichen wir diesen Vorgang durch Ausgaben auf der Kommandozeile.

Jeder Pager wird einem Arzt/einer Ärztin zugeordnet. Dafür wird sein/ihr Name als Parameter an den Konstruktor der Pager-Klasse übergeben. Wenn ein Pager über eine Zustandsänderung benachrichtigt wird, dann soll auf der Kommandozeile der Name des Arztes/der Ärztin und der Raum, in dem er/sie benötigt wird, ausgegeben werden.

Beispiel: „`Dr. House is called to Room 2.02` “.

Hinweis: Sie können mittels `System.out.print` und `System.out.println` auf die Standardausgabe der Kommandozeile schreiben. Der Raum, aus dem die Benachrichtigung erfolgt, kann mittels `getRoom` vom `MedicalDevice` abgefragt werden.

1d) Zustand an Beobachter signalisieren

In der aktuellen Implementierung des Alarmierungssystems werden alle Beobachter bei allen Zustandsänderungen benachrichtigt. Dies geschieht unabhängig von der Art oder Signifikanz der Änderung. Beispielsweise ist eine Schwankung der Sauerstoffsättigung zwischen 100% und 98% aus medizinischer Sicht unwichtig. Ein Abfall der Sauerstoffsättigung unter 90% hingegen ist kritisch und bedarf sofortiger Aufmerksamkeit. Dennoch erfolgt in beiden Fällen eine Zustandsänderung, welche des Alarmsignals im Beobachter `AlarmBell` auslöst und zur Benachrichtigung der Ärzte/Ärztinnen über Pager führt.

Um Anzeigegeräten die Möglichkeit zu geben, angemessen auf Zustandsänderungen reagieren zu können, soll die Signifikanz des aktuellen Zustands von medizinischen Geräten für Beobachter zugänglich gemacht werden. Fügen Sie

dafür eine Methode `isCritical()` der abstrakten Subjektklasse hinzu. Dies Methode gibt als Wahrheitswert zurück, ob die Patient:innenüberwachung einen medizinisch kritischen Zustand identifiziert hat, welcher der Aufmerksamkeit durch medizinisches Personal bedarf.

Implementieren Sie die Methode `isCritical()` in den konkreten Subjektklassen. Ein medizinisch kritischer Zustand liegt in folgenden Fällen vor:

- Die Herzfrequenz ist unterhalb von 40 Schlägen pro Minute.
- Die Herzfrequenz ist oberhalb von 140 Schlägen pro Minute.
- Eine Patient:in ruft nach Hilfe durch Betätigen das Ruf-Knopfs.
- Die Blutsauerstoffsättigung ist unter 90%.

Verwenden Sie die Methode `isCritical()` in den Anzeigegeräten `AlarmBell` und `Pager`, so dass nur dann alarmiert wird, wenn der Zustand kritisch ist.

1e) Verknüpfungen Herstellen

Eine einfache Simulation für die Patient:innenüberwachung ist in der Klasse `de.tu_darmstadt.patient_monitoring.Hospital` bereits teilweise implementiert.

Vervollständigen Sie die Klasse `Hospital`.

- Erzeugen Sie an geeigneter Stelle Pager für „Dr. Grey“ und „Dr. House“.
- Implementieren Sie die Methode `attachIndicatorsToDevices`, so dass alle Anzeigegeräte alle medizinischen Geräte beobachten.
- Benachrichtigen Sie in `attachIndicatorsToDevices` Anzeigegeräte direkt, nachdem sie einem Subjekt hinzugefügt wurden. Dadurch wird sichergestellt, dass alle Anzeigegeräte sofort über die Zustände der Subjekte informiert sind und somit medizinisch kritische Zustände nicht „übersehen“ werden können.

Wichtig: Nach Abschluss der Aufgabe sollte Ihr Code kompilierbar und ausführbar sein.

Hinweis: Wenn Sie die Klasse `Hospital` nach Bearbeiten der Aufgaben ausführen, könne Sie sowohl im GUI Dashboard als auch auf dem Terminal die Ereignisse im Krankenhaus verfolgen.

Aufgabe 2: Entwurfsmuster erarbeiten und anwenden (Bonusaufgabe: 12 Punkte)

In dieser Aufgabe wollen wir ein weiteres Entwurfsmuster kennenlernen, das nicht in der Vorlesung behandelt wurde. Das Strategie-Entwurfsmusters ermöglicht die Definition einer Familie von Algorithmen mit einer gemeinsamen Schnittstelle. Es ermöglicht dem Klienten, den konkret genutzten Algorithmus auszuwählen und auch zur Laufzeit zu ändern. Weitere Informationen dazu finden Sie in Moodle sowie im Buch *Design Patterns* von Gamma et al.

Kontext: Wenn neue Patient:innen im Krankenhaus stationiert werden, dann muss ihnen ein freies Bett in einem geeigneten Zimmer mit noch freien Betten zugewiesen werden. Der Algorithmus, mit dem Patient:innen auf Zimmer verteilt werden, soll dynamisch auswählbar sein. Dies soll durch Anwendung des Strategie-Entwurfsmusters umgesetzt werden.

Es sollen drei verschiedene Zuweisungsverfahren umgesetzt werden:

1. **Balanciert:** Neue Patient:innen sollen gleichmäßig zwischen Räumen verteilt werden. Dadurch soll die Ansteckungsgefahr zwischen Patient:innen reduziert werden, da weniger Patient:innen gleichzeitig in einem Raum sind. Eine neue Patient:in wird in dem Raum stationiert, in dem die wenigsten Betten bereits belegt sind.
2. **Auffüllen:** Um Betriebskosten zu reduzieren, sollen die Patient:innen in insgesamt möglichst wenigen Zimmern stationiert werden. Daher wird eine neue Patient:in in dem Raum stationiert, in dem die wenigsten Betten frei sind.

3. **Quarantäne:** Bei einer Quarantäne muss die Ansteckung zwischen Patient:innen um jeden Preis vermieden werden. Daher wird eine neue Patient:in alleine in einem noch leeren Raum stationiert. Sie darf in keinem Raum stationiert werden, in dem bereits andere Patient:innen stationiert sind.

Ist die Zuweisung uneindeutig, weil mehrere Zimmer gemäß eines Verfahrens zur Stationierung infrage kommen, kann eines dieser Zimmer beliebig ausgewählt werden.

Zusätzlich können Sie davon ausgehen, dass alle Zimmer die gleiche Anzahl Betten haben.

Beispiel: Zur Veranschaulichung der Verfahren betrachten wir das in Abbildung 1 dargestellte Beispiel und die Zuweisung von zwei neuen Patient:innen Alice und Bob. Alice kommt als Erste im Krankenhaus an und wird stationiert. Bob kommt anschließend an und wird nach Alice stationiert.

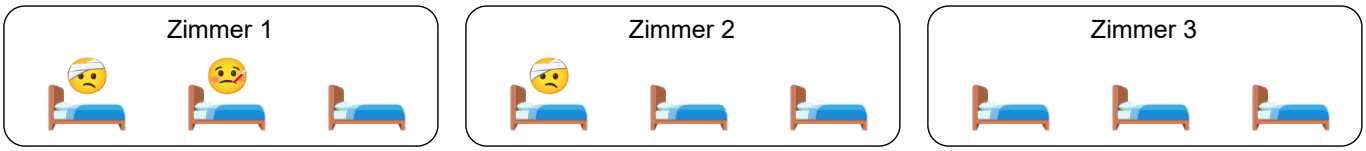

Abbildung 1: Krankenhaus mit drei Zimmern, die jeweils drei Betten enthalten. Insgesamt sind drei Betten bereits belegt.

1. **Balanciert:** Wenn die Verteilung balanciert erfolgen soll, dann wird Alice in Zimmer 3 stationiert, weil dieses Zimmer die meisten freien Betten bietet (3 freie Betten). Danach befinden sich in den Zimmern jeweils 2, 1 und 1 Patient:innen.

   Bob wird anschließend entweder Zimmer 2 oder Zimmer 3 zugeordnet, weil beide gleich viele freie Betten haben (2 freie Betten). Die Entscheidung zwischen diesen beiden Zimmern kann willkürlich erfolgen.

2. **Auffüllen:** Wenn bei der Verteilung Zimmer aufgefüllt werden sollen, dann wird Alice auf Zimmer 1 stationiert, weil dieses Zimmer die wenigsten freien Betten bietet (1 freies Bett).

   Bob wird anschließend auf Zimmer 2 stationiert, da Zimmer 1 bereits voll ist und Zimmer 3 mehr freie Betten hat als Zimmer 2.

3. **Quarantäne:** In einer Quarantäne Situation soll vermieden werden, dass sich Patient:innen gegenseitig anstecken. Daher wird Alice alleine in ein Zimmer mit keinen anderen Patient:innen stationiert. Das einzige ungenutzt Zimmer, welches für Alice in Frage kommt, ist Zimmer 3.

   Wenn Bob im Krankenhaus eintrifft, dann ist kein weiteres Zimmer unbelegt. Daher kann Bob nicht stationiert werden.

Hinweis: Nutzen Sie zur Implementierung der Aufgabe das bereitgestellte Code-Gerüst. Machen Sie sich mit allen Klassen des Code-Gerüsts einschließlich den darin implementierten Methoden vertraut.

Aufgabe:

Die Zuweisung von Patient:innen zu Räumen ist im beiliegenden Projekt in der Klasse `RoomAllocation` in der Methode `allocatePatient` bereits weitgehend realisiert. Allerdings ist die Auswahl des Raums bisher ausgelassen und mit `TODO` markiert. An dieser Stelle muss aus der Liste aller Räume `rooms` (Member der Klasse) ein geeigneter Raum ausgewählt und in der Variable `room` gespeichert werden.

Die Auswahl eines Raumes soll mittels einer Methode erfolgen, welche durch das Strategiemuster ausgetauscht werden kann. Die Liste der Räume soll als Parameter übergeben werden und der ausgewählte Raum durch den Rückgabewert signalisiert werden. Wenn die Zuweisung nicht möglich ist, dann soll `null` zurückgegeben werden. Dies ist beispielsweise dann notwendig, wenn alle Betten belegt sind.

2a) Adaptieren des Entwurfsmusters

Realisieren Sie die beschriebene Funktionalität mit Hilfe des Entwurfsmusters Strategie. **Erstellen** Sie dazu ein **Klassendiagramm**, das alle benötigten Bestandteile des Entwurfsmusters enthält. Modellieren Sie in diesem Klassendiagramm nur die Klassen und Beziehungen, welche an dem Entwurfsmuster Strategie beteiligt sind. Vergeben Sie für die am Entwurfsmuster beteiligten Elemente sinnvolle auf die Anwendung bezogene Bezeichner.

Achten Sie auf korrekte und vollständige Notation.

2b) Anwenden des Entwurfsmusters

Implementieren Sie Ihr Design aus Teilaufgabe a) in Java. Definieren Sie das Strategieinterface und die konkreten Strategieklassen. Implementieren Sie alle konkreten Strategieklassen entsprechend der oben beschriebenen Spezifikation der Zuweisungsverfahren.

Erweitern Sie die Klasse `RoomAllocation`. Fügen Sie der Klasse `RoomAllocation` eine Member-Variable hinzu, welche die Strategie speichert. Initialisieren Sie diese mit einem Zuweisungsverfahren Ihrer Wahl, so dass `RoomAllocation` nach Initialisierung sofort einsatzbereit ist.

Vervollständigen Sie die Methode `allocatePatient`.

Fügen Sie der Klasse einen Setter mit dem Bezeichner `setStrategy` hinzu, durch den die Strategie zur Laufzeit ausgetauscht werden kann.

Ihre Implementierung für diesen Aufgabenteil muss vollständig im Package `de.tu_darmstadt.smart_health.room_allocation` in dem Projekt, welches dem Übungsblatt beiliegt, erfolgen. Nehmen Sie für diesen Aufgabenteil keine Änderungen an Klassen außerhalb dieses Packages vor.

Wichtig: Nach Abschluss der Aufgabe muss Ihr Code kompilierbar und ausführbar sein.

Hinweis: Überprüfen Sie Ihre Implementierung auf Korrektheit und achten Sie bei Ihrer Implementierung auf übliche Fehler. Außerdem muss Ihre Implementierung mit Ihrem im Klassendiagramm dokumentierten Design übereinstimmen.

Aufgabe 3: Entwurfsmuster erkennen

Im Folgenden sind Ausschnitte für die Klassen `EditorKit` und `AbstractCollection` aus der Java API Spezifikation, welche unter der Adresse https://docs.oracle.com/en/java/javase/19/docs/api/index.html verfügbar ist, abgebildet. Nennen Sie zunächst das auf dem Ausschnitt, unter Berücksichtigung der auf dem Aufgabenzettel genannten Informationen, erkennbare Entwurfsmuster und ordnen Sie anschließend die Elemente der Java API den im Folgenden aufgelisteten Bestandteilen (Participants) des Entwurfmusters zu. Hierzu ist es ggf. erforderlich, die vollständige Spezifikation der Klasse und weiterer verwendeter Typen zu betrachten. **Nennen Sie genau ein Beispiel je Bestandteil!**

Bestandteile der Entwurfsmuster:

- Template Method Pattern:
  - Abstrakte Klasse
  - Eine oder mehrere Schablonen-Methoden (template methods)
  - Eine oder mehrere Operationen, die in den Schablonen-Methoden verwendet werden
  - Konkrete Klasse
- Abstract Factory Design Pattern:
  - Abstrakte Fabrik
  - Konkrete Implementierungen der abstrakten Fabrik
  - Methoden zum Erstellen von Produkten
  - Abstrakte Produkte
  - Konkrete Implementierungen der Produkte

Ausschnitte aus Klassen der Java API Spezifikation:

a) Abstract Class `javax.swing.text.EditorKit`
Definiert die benötigte Funktionalität zum Editieren von Texten in unterschiedlichen Formaten wie z.B. HTML oder RTF. Instanzen dieser Klasse finden zum Beispiel Anwendung in der Swing-Komponente `JEditorPane`.

Modifier and Type	Method	Description
`Object`	`clone()`	Creates a copy of the editor kit.
`abstract Caret`	`createCaret()`	Fetches a caret that can navigate through views produced by the associated ViewFactory.
`abstract Document`	`createDefaultDocument()`	Creates an uninitialized text storage model that is appropriate for this type of editor.

b) Abstract Class `java.util.AbstractCollection`
Implementiert die gemeinsame Grundfunktionalität von Realisierungen des Interfaces `java.util.Collection` wie die `java.util.ArrayList`.

`boolean`	`contains(Object o)`	Returns `true` if this collection contains the specified element.
`abstract Iterator<E>`	`iterator()`	Returns an iterator over the elements contained in this collection.
`boolean`	`remove(Object o)`	Removes a single instance of the specified element from this collection, if it is present (optional operation).