

Gr. 1, E. Pitzer

Name \_\_\_\_\_ Aufwand in h \_\_\_\_\_

Gr. 2, F. Gruber-Leitner

Punkte \_\_\_\_\_ Kurzzeichen Tutor / Übungsleiter \_\_\_\_\_ / \_\_\_\_\_

---

Wie angekündigt wollen wir in den restlichen Übungen – also von der vorliegenden 6. bis zur 8. Übung – anhand *einer* Aufgabenstellung verschiedene Möglichkeiten der Java-Klassenbibliothek ausloten. Damit Sie die weiteren Übungen meistern können, sollten Sie von Anfang an mitmachen.

## CaaS – Campina as a Service: Online-Menübestellung

### Kurzbeschreibung

Unsere Haus-und-Hof-Mensa möchte für die Studierenden ein besonderes Service anbieten: Um die Wartezeit auf das Mittagsmenü zu minimieren, kann bereits am Vormittag das Wunschmenü online reserviert werden und kommt dann pünktlich heiß auf den Tisch.

Dafür wird die Erstellung einer Software in Auftrag gegeben, die im Wesentlichen folgende Anforderungen erfüllen soll:

### Menüverwaltung

- Wartung der aktuellen Speisekarte
- Benutzerverwaltung (hinzufügen, aktualisieren, sperren von Benutzern)
- Übersicht über getätigte Bestellungen

### Online-Bestellung

- Login mit Benutzername und Passwort
- Bestellung eines Menüs für den aktuellen Tag

### Menüverwaltung

Die Menüverwaltung steht dem Restaurantbetreiber zur Verfügung und ermöglicht ihm, seine aktuelle Speisekarte zu warten. Die Speisekarte besteht aus verschiedenen Bereichen („Vegetarische Gerichte“, „Aus der Pfanne und vom Grill“, „Fischspezialitäten“ etc.). Für die Online-Bestellung werden nur die Hauptspeisen eingetragen, alle anderen Speisen (Desserts, etc.) können ohnehin rasch serviert werden und brauchen nicht berücksichtigt zu werden. Jede Speise besteht aus der Beschreibung („Wiener Schnitzel mit Petersilkartoffeln“) und einem Preis. Um auch vorübergehende Angebote abbilden zu können, sollen Speisen optional mit einem Beginn- und Endedatum versehen werden können.

Möchten Studierende das Online-Service nutzen, müssen sie sich beim Restaurantbetreiber persönlich melden. Dieser kann dann in der Menüverwaltung neue Benutzer anlegen (Vorname, Nachname, Benutzername, Passwort). Wenn ein Benutzer mehrmals ein Menü bestellt, ohne es abzuholen, so möchte der Betreiber den Benutzer auch wieder sperren können.

In einer Übersichtsanzeige sieht das Restaurant alle Bestellungen des aktuellen Tages (gereiht nach der Uhrzeit, zu der die Benutzer das Menü serviert bekommen wollen) und kann so punktgenau kochen.

## Online-Bestellungsplattform

Nach der Anmeldung mit den erhaltenen Zugangsdaten erscheint die aktuelle Speisekarte. Der Benutzer kann aus der Speisekarte eine Hauptspeise auswählen und muss eine Uhrzeit angeben, zu der die Speise fertig sein soll. Optional kann er auch Sonderwünsche bekannt geben – einen „Gruß an die Küche“ quasi.

Die Online-Plattform ist allerdings jeden Tag nur bis 11:00 geöffnet, anschließend werden keine Bestellungen mehr angenommen.

### Ausbaustufe 1: CaaS-FX

**(24 Punkte)**

Der Restaurantbetreiber möchte vor der endgültigen Auftragserteilung der Online-Plattform zunächst einen Prototyp für die Menüverwaltung sehen.

Entwickeln Sie daher mit Hilfe von JavaFx für die Menüverwaltung einen funktionstüchtigen, ausbaufähigen Benutzeroberflächen-Prototyp. Versuchen Sie eine möglichst intuitiv zu verwendende Benutzeroberfläche zu entwerfen. Setzen Sie dafür Ihr in UEN erworbenes Wissen ein. Trennen Sie den Entwurfs- vom Implementierungsprozess, indem Sie Ihre Benutzeroberfläche zunächst mit Mockups (Grobentwurf der Benutzeroberfläche) modellieren. Fügen Sie die Mockups zu Ihrer Systemdokumentation hinzu.

Trennen Sie den Code zur Realisierung der grafischen Benutzeroberfläche vom Code zur Repräsentation der Daten Ihrer Anwendung (Benutzer, Menüs, Bestellungen etc.). Durch diese Maßnahme wird Ihre Anwendung einfach erweiterbar, was Ihnen in der nächsten Ausbaustufe zugutekommen sollte. Die Verwendung von FXML und Werkzeugen zum Design der grafischen Benutzeroberfläche (JavaFX Scene Builder) ist nicht erlaubt.

Ihr Prototyp soll es ermöglichen, alle Fenster und Dialoge der Benutzeroberfläche zu öffnen (und diese auch wieder zu schließen). Die Anwendung muss die Eingaben aber noch nicht über die Programmlaufzeit hinaus speichern können. Als Ersatz dafür können Sie mit hart codierten Daten arbeiten. Der Benutzeroberflächen-Prototyp muss folgende Funktionen abdecken:

- Benutzer hinzufügen/sperren, Benutzerdaten aktualisieren
- Speisekarte: Bereiche erstellen/löschen
- Speisekarte: Hauptspeisen hinzufügen/löschen
- Anzeige der heutigen Bestellungen

Beachten Sie, dass die Implementierung der anderen Systemkomponenten erst in weiteren Ausbaustufen gefordert ist. Die genauen technischen Anforderungen an diese Komponenten werden im weiteren Verlauf der Übung bekannt gegeben.

## 1 Campina as a Service

### 1.1 Lösungsidee

Folgend ist die Dokumentation für die Aufgabenstellung Campina as a Service angeführt.

Dieser Prototyp soll alle enthaltenen View Teile in Tab anzeigen, was aber in Zukunft geändert werden könnte. Da es sich nicht zeitlich ausgehen wird, soll auf das css und Design im Allgemeinen verzichtet werden.

Bei dieser Aufgabenstellung soll in Bezug auf die Softwarearchitektur wie folgt vorgegangen werden.

1. Das Datenmodell soll über POJOs abgebildet werden, die in den Views nicht direkt verwendet werden sollten es sei den, dass ein Model Overhead wäre.
2. Der Zugriff auf die Datenkomponenten einer Entität innerhalb der Views soll über eigene Models abgebildet werden.
3. Die einzelnen View Teile sollen über eigene Klassen abgebildet werden, die die Views aufbauen und die View Logik (Buisiness Logic) beinhalten.
4. Eigene Kontroller Implementierungen reagieren auf die Action Events der Buttons, die Operationen auf den Entitäten anwenden.
5. In der nächsten Stufe sollen die Datenzugriffe über DAOs abgebildet werden, die in den Kontroller Implementierungen verwendet werden.
6. Die Referenzen auf Nodes sollen, wenn möglich, in einem Kontext zusammengefasst werden, so wie alle Ressourcen, die über Instanz Grenzen hinaus verwenden werden.

Folgende Abbildung zeigt den prinzipiellen Aufbau aller Formulare.

Da die Entitäten nur wenige Skalare Datenkomponenten enthalten, abgesehen von den 1-n, ... Beziehungen, soll hier eine einfache Form angewendet werden.

Diese Views können in Zukunft auch noch abgeändert und die List Daten anders dargestellt werden. (Siehe Order Tabelle)

Das Diagramm zeigt den Aufbau eines Formulars auf einem Gitternetz. Oben befindet sich ein Dropdown-Menü mit der Aufschrift 'Select Entity' und einem nach unten gerichteten Pfeil. Darunter befindet sich ein Textfeld mit der Beschriftung 'Field Label' und dem Inhalt 'Entity Field'. Unten links ist ein Button mit der Aufschrift 'Action' zu sehen. Rechts neben dem Button befindet sich ein Teil eines weiteren Elements, das als 'Entity Field' beschriftet ist.

Abbildung 1: Formularaufbau

Um den folgenden Aufwand der Formulare zu verringern und die Konsistenz der Formulare zu gewährleisten, soll eine Komponente entworfen werden, die in der Lage ist deklarative Klassen oder Methoden Annotationen bzw. deren enthaltenen Informationen zu verarbeiten und ein Formular zu generieren sowie es zu verwalten.

Um zu Vermeiden das Referenzen durch die Klassen gezogen werden, soll eine Kontext implementiert werden, der in der Lage ist die Nötigen Informationen und Referenzen zu verwalten, damit sie in den Implementierungen verwendet werden können.

Die Implementierungen werden in Javadoc und in der folgenden Dokumentation beschrieben.

## Übung 3

### 1.2 Source-Code(Implementation View API)

Folgend ist der Source der View API Implementierungen der Übung angeführt.

#### 1.2.1 AbstractViewModel.java

Folgende abstrakte Klasse stellt die Basisklasse für alle View Models dar, die eine Entität verwalten.

Listing 1: AbstractViewModel.java

```

1 package at.fh.ooe.swe4.fx.campina.view.api;
2
3 import java.io.Serializable;
4 import java.util.Objects;
5
6 import at.fh.ooe.swe4.fx.campina.jpa.api.AbstractEntity;
7
8 /**
9  * This model is the base for all entity amanging models.
10  *
11  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
12  * @date Jun 5, 2015
13  * @param <I>
14  *         the type of the entity id used in this model
15  * @param <T>
16  *         the entity type the current instance is managing
17  */
18 public abstract class AbstractViewModel<I extends Serializable, T extends AbstractEntity<I>> {
19
20     private I id;
21     private T entity;
22
23     /**
24      * Initially calls reset()
25      *
26      * @see AbstractViewModel#reset()
27      */
28     public AbstractViewModel() {
29         super();
30         reset();
31     }
32
33     /**
34      * Initializes this instance with the given entity
35      *
36      * @param entity
37      *         the entity this models backs
38      * @throws NullPointerException
39      *         if the given entity is null
40      */
41     public AbstractViewModel(final T entity) {
42         prepare(entity);
43     }
44
45     /**
46      * Resets this model by setting a new entity.<br>
47      * This method shall call prepare right away so that this model is correctly
48      * initialized.
49      */
50     public abstract void reset();
51
52     /**

```

# Übung 3

```

53     * Prepares this model with the given entity
54     *
55     * @param entity
56     *         the entity this model backs.
57     */
58     public void prepare(T entity) {
59         Objects.requireNonNull(entity);
60
61         setEntity(entity);
62         setId(entity.getId());
63     }
64
65     /**
66     * The model id retrieved from backed entity
67     *
68     * @return the model id
69     */
70     public I getId() {
71         return id;
72     }
73
74     /**
75     * Sets the model id, shall be equal to backed entity id.
76     */
77     public void setId(I id) {
78         this.id = id;
79     }
80
81     /**
82     * Gets the backed entity
83     *
84     * @return the backed entity
85     */
86     public T getEntity() {
87         return entity;
88     }
89
90     /**
91     * Sets the backed entity
92     *
93     * @param entity
94     *         the entity this model backs
95     */
96     public void setEntity(T entity) {
97         this.entity = entity;
98     }
99
100     @Override
101     public int hashCode() {
102         final int prime = 31;
103         int result = 1;
104         result = prime * result + ((id == null) ? 0 : id.hashCode());
105         return result;
106     }
107
108     @Override
109     public boolean equals(Object obj) {
110         if (this == obj) {
111             return true;
112         }
113         if (obj == null) {
114             return false;
115         }

```

## Übung 3

```

116     if (getClass() != obj.getClass()) {
117         return false;
118     }
119     AbstractViewModel<I, T> other = (AbstractViewModel<I, T>) obj;
120     if (id == null) {
121         if (other.id != null) {
122             return false;
123         }
124     } else if (!id.equals(other.id)) {
125         return false;
126     }
127     return true;
128 }
129
130 }

```

### 1.2.2 IdHolder.java

Folgendes Interface markiert eine Klasse oder ein anderes Interface als eine Instanz, die eine Id zur Verfügung stellt.

Listing 2: IdHolder.java

```

1  package at.fh.ooe.swe4.fx.campina.view.api;
2
3  /**
4   * Marks an id holder.
5   *
6   * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
7   * @date Jun 3, 2015
8   * @param <T>
9   *         the type of the hold id
10  */
11  public interface IdHolder<T> {
12
13      /**
14       * Teh id of this id holder
15       *
16       * @return the id
17       */
18      public T getId();
19
20  }

```

### 1.2.3 ViewHandler.java

Folgendes Interface spezifiziert einen View Handler, der einen Teilbaum einer Scene verwaltet.

Listing 3: ViewHandler.java

```
1 package at.fh.ooe.swe4.fx.campina.view.api;
2
3 import javafx.scene.Scene;
4
5 /**
6  * Specifies an handler for a view or view part.
7  *
8  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
9  * @date Jun 3, 2015
10  * @param <N>
11  *         the type of the component to create and manage
12  */
13 public interface ViewHandler<N> extends IdHolder<String> {
14
15     /**
16      * Creates the scene part.
17      *
18      * @return the root node of this scene part.
19      */
20     public N createNode();
21
22     /**
23      * Initializes this part right after the {@link Scene} holds this part.
24      */
25     public void initHandler();
26 }
```

## Übung 3

### 1.2.4 EventHandlerFactory.java

Folgendes Interface spezifiziert eine Event Handler Factory die Events registriert und die registrierten Events in Form einer Map<EventType, List<Event>> liefert.

Listing 4: EventHandlerFactory.java

```

1 package at.fh.ooe.swe4.fx.campina.view.api;
2
3 import java.io.Serializable;
4 import java.util.List;
5 import java.util.Map;
6
7 import javafx.event.Event;
8 import javafx.event.EventHandler;
9 import javafx.event.EventType;
10
11 /**
12  * Specifies an EventHandlerFactory which provides a mpa of event handlers
13  * mapped to their event Type.
14  *
15  * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
16  * @date Jun 5, 2015
17  * @param <T>
18  *         the key type
19  */
20 public interface EventHandlerFactory<K> extends Serializable {
21
22     /**
23      * registers the events for an given id which should point to an
24      * EventTarget.
25      *
26      * @param id
27      *         the event target id
28      * @return the map containing the registered events
29      */
30     public <T extends Event> Map<EventType, List<EventHandler>> registerEventHandler(final K id);
31 }

```



## Übung 3

### 1.3 Source-Code(Implementation View Annotation)

Folgend ist der Source der Annotationen der Übung angeführt.

#### 1.3.1 FormField.java

Folgende Annotation markiert eine Methode als ein Formularfeld.

Diese Annotation wird dazu verwendet um die Form zu erstellen und zu verwalten.

Listing 5: FormField.java

```

1 package at.fh.ooe.swe4.fx.campina.view.annotation;
2
3 import java.lang.annotation.ElementType;
4 import java.lang.annotation.Retention;
5 import java.lang.annotation.RetentionPolicy;
6 import java.lang.annotation.Target;
7
8 import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;
9
10 /**
11  * This annotation marks an method as an form field and connects the form field
12  * to the model method.<br>
13  * This annotation must be placed on valid java bean getter method.
14  *
15  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
16  * @date Jun 5, 2015
17  */
18 @Retention(RetentionPolicy.RUNTIME)
19 @Target(value = {
20     ElementType.METHOD
21 })
22 public @interface FormField {
23
24     /**
25      * @return the form field id
26      */
27     String id();
28
29     /**
30      * @return the backed form field type
31      */
32     FieldType type();
33
34     /**
35      * @return the form field label
36      */
37     String label();
38
39     /**
40      * @return Message displayed if required but not set. Default ""
41      */
42     String requiredMessage() default "";
43
44     /**
45      * @return Flag which marks an form field as required. Default false
46      */
47     boolean required() default false;
48
49     /**
50      * @return the ordinal of this form field in the form
51      */

```

## Übung 3

```
52  int ordinal() default 1;
53
54  /**
55   * @return additional value class. Must be used when form field of type
56   *       {@link FormFieldType#SELECT}.
57   */
58  Class<?> valueClass() default Object.class;
59
60  /**
61   * An event handler factory which can be used to provide event listener for
62   * the generate form field
63   */
64  EventHandlerFactories eventHandlerFactories() default @EventHandlerFactories();
65 }
```

## Übung 3

### 1.3.2 SelectFormField.java

Folgende Annotation markiert eine Methode als einen Data Provider für ein Formularfeld vom Typ Select.

Listing 6: SelectFormField.java

```

1 package at.fh.ooe.swe4.fx.campina.view.annotation;
2
3 import java.lang.annotation.ElementType;
4 import java.lang.annotation.Retention;
5 import java.lang.annotation.RetentionPolicy;
6 import java.lang.annotation.Target;
7
8 import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;
9 import javafx.util.StringConverter;
10
11 /**
12  * This annotation marks an valid java bean getter method as an data provider
13  * for an select field.
14  *
15  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
16  * @date Jun 5, 2015
17  */
18 @Retention(RetentionPolicy.RUNTIME)
19 @Target(ElementType.METHOD)
20 public @interface SelectFormField {
21
22     /**
23      * @return property name of the form field with type
24      *         {@link FormFieldType#SELECT}.<br>
25      *         E.g.: target="day" -> getDays
26      */
27     String target();
28
29     /**
30      * @return An converter used for the backed form field of type
31      *         {@link FormFieldType#SELECT}.
32      */
33     Class<? extends StringConverter> converter() default StringConverter.class;
34 }

```

## Übung 3

### 1.3.3 EventHandlerFactories.java

Folgende Annotation kann in der Annotation `@FormField(...)` verwendet werden um Factories für EventHandler für dieses Formularfeld zu definieren.

Listing 7: EventHandlerFactories.java

```

1 package at.fh.ooe.swe4.fx.campina.view.annotation;
2
3 import java.lang.annotation.ElementType;
4 import java.lang.annotation.Retention;
5 import java.lang.annotation.RetentionPolicy;
6 import java.lang.annotation.Target;
7
8 import at.fh.ooe.swe4.fx.campina.view.api.EventHandlerFactory;
9
10 /**
11  * This annotation allows to define event listener factories for form field
12  * annotated methods.
13  *
14  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
15  * @date Jun 5, 2015
16  */
17 @Retention(RetentionPolicy.RUNTIME)
18 @Target(value = ElementType.ANNOTATION_TYPE)
19 public @interface EventHandlerFactories {
20
21     /**
22      * @return the array of event handler factories.
23      */
24     Class<? extends EventHandlerFactory>[] value() default {};
25 }

```

## Übung 3

### 1.4 Source-Code(Implementation View Component Buidler)

Folgend ist der Source der Komponenten Builder von JavaFX Komponenten der Übung angeführt. Dieser Ansatz ist in Zukunft in Frage zu stellen, da man sich fragen muss ob das Erstellen der Komponenten so ausgelagert Sinn macht.

#### 1.4.1 AbstractFxComponentBuilder.java

Folgende abstrakte Klasse stellt die Basisklasse aller Builder Implementierungen dar und zwingt die abgeleiteten Klassen für einen konkreten Typ der Builder zu sein.

Listing 8: AbstractFxComponentBuilder.java

```

1  package at.fh.ooe.swe4.fx.campina.component.builder.api;
2
3  import java.util.List;
4  import java.util.Map;
5  import java.util.Map.Entry;
6  import java.util.Objects;
7
8  import javafx.event.EventHandler;
9  import javafx.event.EventType;
10
11 /**
12  * This is the base builder for all component buidlers.
13  *
14  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
15  * @date Jun 5, 2015
16  * @param <T>
17  *         the Component this builder is for
18  * @param <B>
19  *         the actual buidler type
20  */
21 @SuppressWarnings({
22     "unchecked", "rawtypes" })
23 public abstract class AbstractFxComponentBuilder<T, B extends AbstractFxComponentBuilder<T, B>> {
24
25     public Map<EventType, List<EventHandler>> events;
26
27     /**
28      *
29      */
30     public AbstractFxComponentBuilder() {
31     }
32
33     /**
34      * Registers the event handlers
35      *
36      * @param handlers
37      *         the handlers to register
38      * @return the current instance
39      */
40     public B registerEventHandlers(
41         final Map<EventType, List<EventHandler>> handlers) {
42         checkIfStarted();
43         Objects.requireNonNull(handlers, "Cannot add event handlers because provided map is null");
44
45         for (Entry<EventType, List<EventHandler>> entry : handlers.entrySet()) {
46             events.put(entry.getKey(), entry.getValue());
47         }
48     }

```

## Übung 3

```

49     return (B) this;
50 }
51
52 /**
53  * Ensures that the handler has been properly started.
54  */
55 protected abstract void checkIfStarted();
56
57 }

```

### 1.4.2 DuplicateKeyException.java

Folgende Klasse stellt die Exception dar um anzuzeigen ob sich doppelte Ids in der Scene befinden. Wird aber nur in den Buidler Implementierungen verwendet, da es mir schwer fiel alle Ids in allen Handlern zu überprüfen um sicherzustellen dass keine Duplikate bezüglich den Ids auftreten.

Listing 9: DuplicateKeyException.java

```

1 package at.fh.ooe.swe4.fx.campina.component.builder.exception;
2
3 public class DuplicateKeyException extends RuntimeException {
4
5     private static final long serialVersionUID = 99858106221010346L;
6
7     public DuplicateKeyException() {
8         // TODO Auto-generated constructor stub
9     }
10
11     public DuplicateKeyException(String message) {
12         super(message);
13         // TODO Auto-generated constructor stub
14     }
15
16     public DuplicateKeyException(Throwable cause) {
17         super(cause);
18         // TODO Auto-generated constructor stub
19     }
20
21     public DuplicateKeyException(String message, Throwable cause) {
22         super(message, cause);
23         // TODO Auto-generated constructor stub
24     }
25
26     public DuplicateKeyException(String message, Throwable cause,
27         boolean enableSuppression, boolean writableStackTrace) {
28         super(message, cause, enableSuppression, writableStackTrace);
29         // TODO Auto-generated constructor stub
30     }
31
32 }

```

## Übung 3

### 1.4.3 MenuBarBuilder.java

Folgende Klasse stellt den Builder für MenuBar dar.

Listing 10: MenuBarBuidler.java

```

1 package at.fh.ooe.swe4.fx.campina.component.builder.impl;
2
3 import java.util.HashSet;
4 import java.util.Objects;
5 import java.util.Set;
6
7 import javafx.scene.control.Menu;
8 import javafx.scene.control.MenuBar;
9 import at.fh.ooe.swe4.fx.campina.component.builder.api.AbstractFxComponentBuilder;
10
11 /**
12  * This is the buidler for menu bars
13  *
14  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
15  * @date Jun 5, 2015
16  */
17 public class MenuBarBuilder extends AbstractFxComponentBuilder<MenuBar, MenuBarBuilder> {
18
19     private Set<Menu> menus;
20
21     /**
22      * Starts the buidler
23      *
24      * @return the current instance
25      */
26     public MenuBarBuilder start() {
27         if (menus != null) {
28             throw new IllegalStateException("Builder already initialized");
29         }
30         menus = new HashSet<>();
31         return this;
32     }
33
34     /**
35      * Ads an menu to this builder
36      *
37      * @param menu
38      *         the menu for the menu bar
39      * @return the current instance
40      */
41     public MenuBarBuilder addMenu(final Menu menu) {
42         checkIfStarted();
43         Objects.requireNonNull(menu, "Cannto add null menu to menu bar");
44
45         menus.add(menu);
46         return this;
47     }
48
49     /**
50      * Buidls anew menu bar isntance with the ormerly set menus.
51      *
52      * @return the menu bar instance
53      */
54     public MenuBar build() {
55         checkIfStarted();
56         final MenuBar bar = new MenuBar();
57         bar.getMenus()
58             .addAll(menus);

```

## Übung 3

```
59     return bar;
60 }
61
62 /**
63  * Ends this builder. Will need to be restarted if reused.
64  *
65  * @return the current instance
66  */
67 public MenuBarBuilder end() {
68     checkIfStarted();
69     menus = null;
70     return this;
71 }
72
73 @Override
74 protected void checkIfStarted() {
75     if (menus == null) {
76         throw new IllegalStateException("Buuilder not inititalized");
77     }
78 }
79 }
```



## Übung 3

### 1.4.4 MenuBuilder.java

Folgende Klasse stellt den Builder für Menu dar.

Listing 11: MenuBuidler.java

```

1 package at.fh.ooe.swe4.fx.campina.component.builder.impl;
2
3 import java.util.ArrayList;
4 import java.util.HashMap;
5 import java.util.List;
6 import java.util.Map.Entry;
7 import java.util.Objects;
8
9 import javafx.event.EventHandler;
10 import javafx.event.EventType;
11 import javafx.scene.control.Menu;
12 import javafx.scene.control.MenuItem;
13 import at.fh.ooe.swe4.fx.campina.component.builder.api.AbstractFxComponentBuilder;
14
15 /**
16  * This is the buidler for {@link Menu}
17  *
18  * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
19  * @date Jun 5, 2015
20  */
21 public class MenuBuilder extends AbstractFxComponentBuilder<Menu, MenuBuilder> {
22
23     public List<MenuItem> items = null;
24
25     /**
26      * Starts the builder
27      *
28      * @return the current instance
29      */
30     public MenuBuilder start() {
31         if (items != null) {
32             throw new IllegalStateException("Builder already initialized");
33         }
34
35         items = new ArrayList<>();
36         events = new HashMap<>();
37         return this;
38     }
39
40     /**
41      * Ends the builder. Needs to be restarted i reused.
42      *
43      * @return the current instance
44      */
45     public MenuBuilder end() {
46         checkIfStarted();
47         items = null;
48         return this;
49     }
50
51     /**
52      * Builds the menu instance.
53      *
54      * @param id
55      *         the id of the menu instance
56      * @param label
57      *         the label of the menu instance
58      * @return the menu instance

```

## Übung 3

```

59  */
60  public Menu build(final String id, final String label) {
61      checkIfStarted();
62      Objects.requireNonNull(label, "Cannot create menu with null id");
63      Objects.requireNonNull(label, "Cannot create menu with null label");
64
65      final Menu menu = new Menu(label);
66      menu.setId(id);
67      for (MenuItem item : items) {
68          menu.getItems()
69              .add(item);
70          item.setId(new StringBuilder(item.getParentMenu()
71              .getId()).append(":")
72              .append(item.getId())
73              .toString());
74      }
75
76      for (Entry<EventType, List<EventHandler>> event : events.entrySet()) {
77          for (EventHandler handler : event.getValue()) {
78              menu.addEventHandler(event.getKey(), handler);
79          }
80      }
81      return menu;
82  }
83
84  /**
85   * Adds an item to this builder
86   *
87   * @param item
88   *         the item to be added
89   * @return the current instance
90   */
91  public MenuBuilder addItem(final MenuItem item) {
92      checkIfStarted();
93      Objects.requireNonNull(item, "Cannot add nul item");
94
95      items.add(item);
96      return this;
97  }
98
99  @Override
100  protected void checkIfStarted() {
101      if (items == null) {
102          throw new IllegalStateException("Builder not initialized");
103      }
104  }
105  }

```

## Übung 3

### 1.4.5 MenuItemBuilder.java

Folgende Klasse stellt den Builder für MenuItem dar.

Listing 12: MenuItemBuidler.java

```

1 package at.fh.ooe.swe4.fx.campina.component.builder.impl;
2
3 import java.util.HashMap;
4 import java.util.HashSet;
5 import java.util.List;
6 import java.util.Map.Entry;
7 import java.util.Objects;
8 import java.util.Set;
9
10 import javafx.event.EventHandler;
11 import javafx.event.EventType;
12 import javafx.scene.control.MenuItem;
13 import at.fh.ooe.swe4.fx.campina.component.builder.api.AbstractFxComponentBuilder;
14 import at.fh.ooe.swe4.fx.campina.component.builder.exception.DuplicateKeyException;
15
16 public class MenuItemBuilder extends AbstractFxComponentBuilder<MenuItem, MenuItemBuilder> {
17
18     private Set<String> ids;
19
20     public MenuItemBuilder start() {
21         ids = new HashSet<String>();
22         events = new HashMap<>();
23         return this;
24     }
25
26     public MenuItemBuilder removeFromContext(final String id) {
27         checkIfStarted();
28         Objects.requireNonNull(id,
29             "Cannot remove item from context with null id");
30
31         ids.remove(id);
32         return this;
33     }
34
35     public MenuItem build(final String id, final String label) {
36         checkIfStarted();
37         Objects.requireNonNull(id, "Cannot create item for null id");
38         Objects.requireNonNull(label, "Cannot create item for null label");
39
40         if (!ids.add(id)) {
41             throw new DuplicateKeyException(
42                 "Builder has already built an item with this id");
43         }
44
45         final MenuItem item = new MenuItem(label);
46         item.setId(id);
47         for (Entry<EventType, List<EventHandler>> event : events.entrySet()) {
48             for (EventHandler handler : event.getValue()) {
49                 item.addEventHandler(event.getKey(), handler);
50             }
51         }
52         return item;
53     }
54
55     public MenuItemBuilder end() {
56         ids = null;
57         events = null;
58         return this;

```

## Übung 3

```
59     }
60
61     @Override
62     protected void checkIfStarted() {
63         if (ids == null) {
64             throw new IllegalStateException("Builder not inititalized");
65         }
66     }
67 }
```

## Übung 3

### 1.5 Source-Code(Implementation Form Handler)

Folgend ist der Source des Form Handling angeführt.

#### 1.5.1 FormUtils.java

Folgende Klasse stellt Utilities für das Form Handling zur Verfügung.

Listing 13: FormUtils.java

```

1  package at.fh.ooe.swe4.fx.campina.view.form;
2
3  import java.math.BigDecimal;
4  import java.math.RoundingMode;
5  import java.text.NumberFormat;
6  import java.time.Instant;
7  import java.time.LocalDate;
8  import java.time.ZoneId;
9  import java.util.Calendar;
10 import java.util.Date;
11 import java.util.Objects;
12
13 import javafx.scene.Node;
14 import javafx.scene.control.ChoiceBox;
15 import javafx.scene.control.DatePicker;
16 import javafx.scene.control.TextField;
17 import javafx.util.converter.BigDecimalStringConverter;
18
19 /**
20  * Utility class for handling form fields.
21  *
22  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
23  * @date Jun 5, 2015
24  */
25 public class FormUtils {
26
27     private FormUtils() {
28     }
29
30     /**
31      * Enumeration whcih specifies the supported JavaFX components.
32      *
33      * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
34      * @date Jun 5, 2015
35      */
36     public enum FormFieldType {
37         INPUT_TEXT(String.class, TextField.class),
38         DATE_PICKER(Calendar.class, DatePicker.class),
39         DECIMAL(BigDecimal.class, TextField.class),
40         SELECT(Object.class, ChoiceBox.class);
41
42         public final Class<?> valueClass;
43         public final Class<? extends Node> nodeClass;
44
45         /**
46          * @param valueClass
47          * @param nodeClass
48          */
49         private FormFieldType(Class<?> valueClass, Class<? extends Node> nodeClass) {
50             this.valueClass = valueClass;
51             this.nodeClass = nodeClass;
52         }

```

# Übung 3

```

53 }
54
55 /**
56  * Interface which marks an Validator
57  *
58  * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
59  * @date Jun 5, 2015
60  * @param <T>
61  *         the {@link Node} type this validator is for
62  */
63 public static interface FormFieldValidator<T extends Node> {
64     /**
65      * Answers the question if the current instance is valid.
66      *
67      * @param type
68      * @param inst
69      * @return true if valid, false otherwise
70      */
71     public boolean valid(FormFieldType type, T inst);
72 }
73
74 /**
75  * A required validator for Form Fields.
76  *
77  * @author Thomas Herzog <thomas.hertzog@students.fh-hagenberg.at>
78  * @date Jun 5, 2015
79  * @param <T>
80  *         the {@link Node} type fo the to validate node
81  */
82 public static class RequiredFormFieldValidator<T extends Node> implements FormFieldValidator<T>
83     ↪ {
84
85     @Override
86     public boolean valid(FormFieldType type, T node) {
87         final Object value = getFormFieldValue(type, node);
88         if (value == null) {
89             return Boolean.FALSE;
90         }
91         if (value instanceof String) {
92             return !((String) value).trim()
93                 .isEmpty();
94         }
95         return Boolean.TRUE;
96     }
97 }
98
99 /**
100  * Creates a node regarding the given {@link FormFieldType}
101  *
102  * @param type
103  *         the type of the form field
104  * @return the new corresponding node instance
105  */
106 public static Node create(final FormFieldType type) {
107     Objects.requireNonNull(type);
108
109     switch (type) {
110         case INPUT_TEXT:
111             return new TextField();
112         case DATE_PICKER:
113             return new DatePicker();
114         case DECIMAL:
115             return new TextField();

```

## Übung 3

```

115     case SELECT:
116         return new ChoiceBox<>();
117     default:
118         throw new IllegalArgumentException("FormFieldType: '" + type.name() + "' is not managed
119             ↪ here");
120     }
121 }
122
123 /**
124  * Gets the node value.
125  *
126  * @param type
127  *         the type of the form field
128  * @param node
129  *         the node to get value from
130  *
131  * @return the node's value
132  * @throws NullPointerException
133  *         if the given node is null
134  * @throws IllegalArgumentException
135  *         if the node is of an unmanaged type
136  */
137 public static Object getFormFieldValue(FormFieldType type, final Node node) {
138     Objects.requireNonNull(node, "Need node to get its value");
139
140     switch (type) {
141     case INPUT_TEXT:
142         return ((TextField) node).getText();
143     case DATE_PICKER:
144         final DatePicker picker = (DatePicker) node;
145         final Calendar cal;
146         LocalDate localDate = picker.getValue();
147         if (localDate != null) {
148             Instant instant = Instant.from(localDate.atStartOfDay(ZoneId.systemDefault()));
149             cal = Calendar.getInstance();
150             cal.setTime(Date.from(instant));
151         } else {
152             cal = null;
153         }
154         return cal;
155     case DECIMAL:
156         final String value = ((TextField) node).getText();
157         if ((value != null) && (!value.trim().isEmpty())) {
158             return new BigDecimal(value);
159         }
160         return (BigDecimal) null;
161     case SELECT:
162         final ChoiceBox<?> box = (ChoiceBox<?>) node;
163         return box.getSelectionModel()
164             .getSelectedItem();
165     default:
166         break;
167     }
168     throw new IllegalArgumentException("Nod of type '" + node.getClass()
169         .getName() + "' unknown");
170 }
171
172 /**
173  * Resets the node value by setting it to null.
174  *
175  * @param type
176  *         the form field type

```

## Übung 3

```

177  * @param node
178  *         the node to be reset
179  * @see FormFieldType#setFormValue(Node, Object)
180  */
181  public static void resetFormValue(final FormFieldType type, final Node node) {
182      setFormValue(type, node, null);
183  }
184
185  /**
186   * Sets a value on the given node.
187   *
188   * @param type
189   *         the form field type
190   * @param node
191   *         the node to set value on
192   * @param value
193   *         the value to be set. (Model held type)
194   * @throws NullPointerException
195   *         if the given node is null
196   * @throws IllegalArgumentException
197   *         if the node is of an unmanaged type
198   */
199  public static void setFormValue(final FormFieldType type, final Node node, final Object value) {
200      Objects.requireNonNull(node, "Need node to get its value");
201
202      switch (type) {
203
204          case INPUT_TEXT:
205              ((TextField) node).setText((String) value);
206              return;
207
208          case DATE_PICKER:
209              final LocalDate localDate;
210              if (value != null) {
211                  Date date = new Date();
212                  Instant instant = date.toInstant();
213                  localDate = instant.atZone(ZoneId.systemDefault())
214                      .toLocalDate();
215              } else {
216                  localDate = null;
217              }
218              ((DatePicker) node).setValue(localDate);
219              return;
220          case DECIMAL:
221              String formatted = null;
222              if (value != null) {
223                  final NumberFormat nf = NumberFormat.getCurrencyInstance();
224                  nf.setMaximumFractionDigits(2);
225                  nf.setMinimumFractionDigits(1);
226                  nf.setRoundingMode(RoundingMode.UNNECESSARY);
227                  nf.setMinimumIntegerDigits(1);
228                  nf.setMaximumFractionDigits(3);
229                  formatted = nf.format(((BigDecimal) value).doubleValue());
230                  formatted = new BigDecimalStringConverter().toString((BigDecimal) value);
231              }
232              ((TextField) node).setText(formatted);
233              return;
234          case SELECT:
235              final ChoiceBox<Object> box = (ChoiceBox<Object>) node;
236              box.getSelectionModel()
237                  .select(box.getItems()
238                      .indexOf(value));
239              return;

```



Übung 3

---

```
240     default:
241     }
242     throw new IllegalArgumentException("Nod of type '" + node.getClass()
243                                     .getName() + "' unknown");
244 }
245 }
```

## Übung 3

### 1.5.2 FormHandler.java

Dieser Handler erstellt das Formular und verwaltet dieses Formular.

Listing 14: FormHandler.java

```

1 package at.fh.ooe.swe4.fx.campina.view.form;
2
3 import java.lang.reflect.Method;
4 import java.util.ArrayList;
5 import java.util.Collections;
6 import java.util.Comparator;
7 import java.util.HashMap;
8 import java.util.List;
9 import java.util.Map;
10 import java.util.Objects;
11
12 import javafx.collections.ObservableList;
13 import javafx.scene.Node;
14 import javafx.scene.control.ChoiceBox;
15 import javafx.scene.layout.ColumnConstraints;
16 import javafx.scene.layout.GridPane;
17 import javafx.scene.text.Text;
18 import javafx.util.StringConverter;
19 import at.fh.ooe.swe4.fx.campina.view.annotation.FormField;
20 import at.fh.ooe.swe4.fx.campina.view.annotation.SelectFormField;
21 import at.fh.ooe.swe4.fx.campina.view.api.AbstractViewModel;
22 import at.fh.ooe.swe4.fx.campina.view.api.FormContext;
23 import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;
24 import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldValidator;
25 import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.RequiredFormFieldValidator;
26
27 /**
28  * This class is a for field handler which handles the form fields related to
29  * the model class the handler is for. <br>
30  * This implementation uses fluent api.
31  *
32  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
33  * @date May 31, 2015
34  * @param <T>
35  */
36 public class FormHandler<T extends AbstractViewModel> {
37
38     /**
39      * Model which holds the resolved form field information.
40      *
41      * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
42      * @date May 31, 2015
43      */
44     private static class FormFieldResolvedModel {
45
46         public final String id;
47         public final String globalPrefix;
48         public final String methodGetterName;
49         public final String methodSetterName;
50         public final FormField field;
51         private final Map<Object, Object> additionalData = new HashMap<>();
52
53         /**
54          * Used for lightweight instance for searching in set
55          *
56          * @param globalPrefix
57          * @param methodName
58          */

```

# Übung 3

```

59     public FormFieldResolvedModel(String globalPrefix, String methodName) {
60         super();
61         this.globalPrefix = globalPrefix;
62         this.methodGetterName = methodName;
63         this.methodSetterName = methodName.replace("get", "set");
64         this.id = "field-" + methodName.substring(3, methodName.length())
65             .toLowerCase();
66
67         this.field = null;
68     }
69
70     /**
71      * @param globalPrefix
72      * @param methodName
73      * @param field
74      */
75     public FormFieldResolvedModel(String globalPrefix, String methodName, FormField field) {
76         super();
77         Objects.requireNonNull(globalPrefix);
78         Objects.requireNonNull(methodName);
79         Objects.requireNonNull(field);
80
81         if (!methodName.startsWith("get")) {
82             throw new IllegalArgumentException("Method must be a valid java bean getter");
83         }
84         this.globalPrefix = globalPrefix;
85         this.methodGetterName = methodName;
86         this.methodSetterName = methodName.replace("get", "set");
87         this.id = "field-" + methodName.substring(3, methodName.length())
88             .toLowerCase();
89
90         ;
91         this.field = field;
92     }
93
94     public String toMessageId() {
95         Objects.requireNonNull(id);
96
97         return globalPrefix + "-message-" + id;
98     }
99
100    public String toLabelId() {
101        return globalPrefix + "-label-" + id;
102    }
103
104    public String toNodeId() {
105        return globalPrefix + "-node-" + id;
106    }
107
108    public <T> void putData(final Object
109        key, T instance) {
110        Objects.requireNonNull(key);
111        Objects.requireNonNull(instance);
112
113        additionalData.put(key, instance);
114    }
115
116    public <T> T getData(final Object key) {
117        Objects.requireNonNull(key);
118
119        return (T) additionalData.get(key);
120    }
121
122    @Override

```

## Übung 3

```

122     public int hashCode() {
123         final int prime = 31;
124         int result = 1;
125         result = prime * result + ((id == null) ? 0 : id.hashCode());
126         return result;
127     }
128
129     @Override
130     public boolean equals(Object obj) {
131         if (this == obj)
132             return true;
133         if (obj == null)
134             return false;
135         if (getClass() != obj.getClass())
136             return false;
137         FormFieldResolvedModel other = (FormFieldResolvedModel) obj;
138         if (id == null) {
139             if (other.id != null)
140                 return false;
141         } else if (!id.equals(other.id))
142             return false;
143         return true;
144     }
145 }
146
147 private boolean started = Boolean.FALSE;
148
149 /**
150  * Empty default constructor
151  */
152 public FormHandler() {
153     super();
154 }
155
156 /**
157  * Initializes this builder.
158  *
159  * @return the current instance
160  * @throws NullPointerException
161  *         if the given model class is null
162  * @throws IllegalStateException
163  *         if the builder is already started
164  */
165 public FormHandler<T> init() {
166     if (started) {
167         throw new IllegalStateException("Handler needs to be end before restarted");
168     }
169
170     started = Boolean.TRUE;
171
172     return this;
173 }
174
175 /**
176  * Ands this builder.
177  *
178  * @return the current instance
179  * @throws IllegalStateException
180  *         if the builder hasn't been started before
181  */
182 public FormHandler<T> end() {
183     checkIfStarted();
184 }

```

# Übung 3

```

185     this.started = Boolean.FALSE;
186     return this;
187 }
188
189 /**
190  * Generates the grid which holds the form and its fields.
191  *
192  * @param ctx
193  *         the backing form context which gets set on each form field
194  *         node (label, field, message)
195  * @return the generated grid
196  * @throws NullPointerException
197  *         if the given id is null
198  * @throws IllegalStateException
199  *         if the builder hasn't been started before
200  */
201 public GridPane generateFormGrid(FormContext<T> ctx) {
202     checkIfStarted();
203     Objects.requireNonNull(ctx, "Form Context must be given");
204
205     // column constraints, should be replaced by css
206     final ColumnConstraints labelColConst = new ColumnConstraints(2);
207     labelColConst.setPrefWidth(150);
208     final ColumnConstraints valueColConst = new ColumnConstraints(2);
209     valueColConst.setPrefWidth(250);
210     final ColumnConstraints messageColConst = new ColumnConstraints(2);
211     valueColConst.setPrefWidth(250);
212
213     // the grid which holds the form
214     final GridPane gridPane = new GridPane();
215     gridPane.setId(ctx.id + "-form");
216
217     gridPane.setHgap(10);
218     gridPane.setVgap(10);
219     gridPane.setUserData(ctx);
220     gridPane.getColumnConstraints()
221         .add(labelColConst);
222     gridPane.getColumnConstraints()
223         .add(valueColConst);
224     gridPane.getColumnConstraints()
225         .add(messageColConst);
226
227     // the form fields defined in the model
228     final List<FormFieldResolvedModel> models = createResolvedModels(ctx);
229
230     // generate form fields
231     for (int i = 0; i < models.size(); i++) {
232         final FormFieldResolvedModel model = models.get(i);
233
234         // form field label
235         final Text labelText = new Text(model.field.label());
236         labelText.setId(model.toLabelId());
237         labelText.setUserData(ctx);
238
239         // form field message
240         final Text messageText = new Text("");
241         messageText.setId(model.toMessageId());
242         messageText.setVisible(Boolean.FALSE);
243         messageText.setUserData(ctx);
244
245         // form field
246         final Node node = FormUtils.create(model.field.type());
247         node.setId(model.toNodeId());

```

# Übung 3

```

248     node.setUserData(ctx);
249     // Handling for select type
250     if (model.field.type()
251         .equals(FormFieldType.SELECT)) {
252         final ObservableList<Object> list = model.getData(model.id);
253         final SelectFormField select = model.getData(SelectFormField.class);
254         StringConverter<Object> converter = null;
255         if (!select.converter()
256             .equals(StringConverter.class)) {
257             try {
258                 converter = select.converter()
259                     .newInstance();
260             } catch (Throwable e) {
261                 throw new IllegalStateException("Could not instantiate select string converter '" +
262                     ↪ select.converter()
263                         .getName() + "'", e);
264             }
265         }
266         final ChoiceBox<Object> box = (ChoiceBox<Object>) node;
267         box.setItems(list);
268         if (converter != null) {
269             box.setConverter(converter);
270         }
271     }
272     // TODO: register form field events
273
274     // set form fields on grid
275     gridPane.add(labelText, 0, i);
276     gridPane.add(node, 1, i);
277     gridPane.add(messageText, 2, i);
278 }
279
280 return gridPane;
281 }
282
283 /**
284  * Fills the model with the form field set values.
285  *
286  * @param ctx
287  *      the form context
288  *
289  * @return the current instance
290  * @throws NullPointerException
291  *      if either the scene or model is null
292  * @throws IllegalStateException
293  *      if the model defines form fields which could not be found on
294  *      the given scene.<br>
295  *      If the reflective invocation of the models setter fails<br>
296  *      If the builder hasn't been started yet
297  */
298 public FormHandler<T> fillModel(FormContext<T> ctx) {
299     checkIfStarted();
300     Objects.requireNonNull(ctx, "Need form context to search for form fields");
301
302     final List<FormFieldResolvedModel> models = createResolvedModels(ctx);
303     for (FormFieldResolvedModel fieldModel : models) {
304         final Node node = ctx.scene.lookup("#" + fieldModel.toNodeId());
305         if (node == null) {
306             throw new IllegalStateException("Scene does not contain form field with id");
307         }
308         try {
309             final Method method;

```

# Übung 3

```

310     Class<?> valueClass = fieldModel.field.valueClass();
311     if (valueClass.equals(Object.class)) {
312         valueClass = fieldModel.field.type().valueClass;
313     }
314     method = ctx.model.getClass()
315         .getMethod(fieldModel.methodSetterName, valueClass);
316     method.invoke(ctx.model, FormUtils.getFormFieldValue(fieldModel.field.type(), node));
317 } catch (Throwable e) {
318     throw new IllegalStateException("Could not set model value", e);
319 }
320 }
321 return this;
322 }
323
324 /**
325  * Fills the form with the model provided set values.
326  *
327  * @param ctx
328  *         the form context
329  *
330  * @return the current instance
331  * @throws NullPointerException
332  *         if either the scene or model is null
333  * @throws IllegalStateException
334  *         if the model defines form fields which could not be found on
335  *         the given scene.<br>
336  *         If the reflective invocation of the models getter fails<br>
337  *         If the builder hasn't been started yet
338  */
339 public FormHandler<T> fillForm(FormContext<T> ctx) {
340     checkIfStarted();
341     Objects.requireNonNull(ctx, "Need context to search for form fields");
342
343     final List<FormFieldResolvedModel> models = createResolvedModels(ctx);
344     for (FormFieldResolvedModel fieldModel : models) {
345         final Node node = ctx.scene.lookup("#" + fieldModel.toNodeId());
346         if (node == null) {
347             throw new IllegalStateException("Scene does not contain form field with id");
348         }
349         try {
350             if (FormFieldType.SELECT.equals(fieldModel.field.type())) {
351                 final ChoiceBox<Object> box = (ChoiceBox<Object>) node;
352                 box.setItems(fieldModel.getData(fieldModel.id));
353             }
354             final Object value = ctx.model.getClass()
355                 .getMethod(fieldModel.methodGetterName)
356                 .invoke(ctx.model);
357             FormUtils.setFormValue(fieldModel.field.type(), node, value);
358         } catch (Throwable e) {
359             throw new IllegalStateException("Could not set model value", e);
360         }
361     }
362     return this;
363 }
364
365 /**
366  * Resets the form by searching all form elements in the given scene. It
367  * sets all values to null.
368  *
369  * @param ctx
370  *         the form context
371  *
372  * @return the current instance

```

# Übung 3

```

373  * @throws NullPointerException
374  *         if the context is null
375  * @throws IllegalStateException
376  *         if a model has a for field defined but it could not be found
377  *         on the scene
378  */
379  public FormHandler<T> resetForm(FormContext<T> ctx) {
380      checkIfStarted();
381      Objects.requireNonNull(ctx.scene, "Need to scene to search for form fields");
382
383      final List<FormFieldResolvedModel> models = createResolvedModels(ctx);
384      for (FormFieldResolvedModel fieldModel : models) {
385          final Node node = ctx.scene.lookup("#" + fieldModel.toNodeId());
386          if (node == null) {
387              throw new IllegalStateException("Scene does not contain form field with id");
388          }
389          FormUtils.resetFormValue(fieldModel.field.type(), node);
390          final Text messageNode = (Text) ctx.scene.lookup("#" + fieldModel.toMessageId());
391          messageNode.setVisible(Boolean.FALSE);
392      }
393      return this;
394  }
395
396  /**
397   * Validates the form backed by the given {@link FormContext}. <br>
398   * Manipulates the context valid flag during the validation
399   *
400   * @param ctx
401   *         the form context
402   * @return the current instance
403   * @throws NullPointerException
404   *         if the context is null
405   * @throws IllegalStateException
406   *         if the builder hasn't been start yet
407   */
408  public FormHandler<T> validateForm(final FormContext<T> ctx) {
409      checkIfStarted();
410      Objects.requireNonNull(ctx, "Need form context to validate form");
411
412      // expected valid
413      ctx.valid = Boolean.TRUE;
414      // required validator
415      final FormFieldValidator<Node> requiredValidator = new RequiredFormFieldValidator<Node>();
416      // the form field models
417      final List<FormFieldResolvedModel> models = createResolvedModels(ctx);
418      for (FormFieldResolvedModel fieldModel : models) {
419          final Node node = ctx.scene.lookup("#" + fieldModel.toNodeId());
420          if (node == null) {
421              throw new IllegalStateException("Scene does not contain form field with id");
422          }
423          final Text messageNode = (Text) ctx.scene.lookup("#" + fieldModel.toMessageId());
424          // need type validation
425          if (FormFieldType.DECIMAL.equals(fieldModel.field.type())) {
426              try {
427                  FormUtils.getFormFieldValue(FormFieldType.DECIMAL, node);
428                  messageNode.setVisible(Boolean.FALSE);
429                  messageNode.setText("");
430              } catch (NumberFormatException e) {
431                  ctx.valid = Boolean.FALSE;
432                  messageNode.setVisible(Boolean.TRUE);
433                  messageNode.setText("Keine gültige Nummer");
434                  continue;
435              }
436          }
437      }
438  }

```



## Übung 3

```

436     }
437     // need required validation
438     if (fieldModel.field.required()) {
439         // is invalid
440         if (!requiredValidator.valid(fieldModel.field.type(), node)) {
441             ctx.valid = Boolean.FALSE;
442             messageNode.setVisible(Boolean.TRUE);
443             messageNode.setText(fieldModel.field.requiredMessage());
444         }
445         // reset if valid
446         else {
447             messageNode.setText("");
448         }
449     }
450 }
451 return this;
452 }
453
454 /**
455  * Throws exception if builder is not started when this method gets called.
456  *
457  * @throws IllegalStateException
458  *         if the builder hasn't been started yet
459  */
460 private void checkIfStarted() {
461     if (!started) {
462         throw new IllegalStateException("Builder not started");
463     }
464 }
465
466 /**
467  * Creates a list of resolved field models for the given model class.
468  *
469  * @param ctx
470  *        The context which holds a model to generate the field models
471  *        from
472  * @return the resolved form fields represented by the created model
473  * @throws NullPointerException
474  *         if the context is null
475  */
476 private List<FormFieldResolvedModel> createResolvedModels(FormContext<T> ctx) {
477     Objects.requireNonNull(ctx, "Context must not be null");
478
479     final T model = ctx.model;
480     final List<FormFieldResolvedModel> models = new ArrayList<>();
481     final Method[] methods = model.getClass()
482         .getDeclaredMethods();
483
484     // form field annotations
485     for (Method method : methods) {
486         final FormField field = method.getAnnotation(FormField.class);
487         FormFieldResolvedModel fieldModel;
488         final String methodName = method.getName();
489         // ensure valid java bean getter
490         if (field != null) {
491             if (!methodName.startsWith("get")) {
492                 throw new IllegalStateException("FormField annotated method must be a valid getter
493                     ↪ method '" + methodName
494                     + "'");
495             }
496             fieldModel = new FormFieldResolvedModel(ctx.id, methodName, field);
497             models.add(fieldModel);
498         }
499     }
500 }

```

## Übung 3

```

498     }
499
500     // additional annotations
501     for (Method method : methods) {
502
503         final SelectFormField select = method.getAnnotation(SelectFormField.class);
504         FormFieldResolvedModel fieldModel;
505
506         if (select != null) {
507             int index = -1;
508             final String target = select.target();
509             final String targetGetter = new StringBuilder("get").append(target.substring(0, 1)
510                 .toUpperCase())
511                 .append(target.substring(1, target.length()))
512                 .toString();
513             if ((index = models.indexOf(new FormFieldResolvedModel(ctx.id, targetGetter))) == -1) {
514                 throw new IllegalStateException("SelectFormField target '" + targetGetter + "' field
515                     ↪ model '" + model.getClass()
516                                     .getName() + "' not found");
517             }
518             fieldModel = models.get(index);
519             try {
520                 final ObservableList<Object> dataList = (ObservableList<Object>) method.invoke(model);
521                 fieldModel.putData(fieldModel.id, dataList);
522                 fieldModel.putData(select.annotationType(), select);
523             } catch (Throwable e) {
524                 throw new IllegalStateException("Cannot retrieve select data (? extends Iterable<?>)",
525                     ↪ e);
526             }
527         }
528     }
529
530     // Sort models by annotation provided ordinal
531     Collections.sort(models, new Comparator<FormFieldResolvedModel>() {
532         @Override
533         public int compare(FormFieldResolvedModel o1, FormFieldResolvedModel o2) {
534             return Integer.valueOf(o1.field.ordinal())
535                 .compareTo(o2.field.ordinal());
536         }
537     });
538     return models;
539 }

```

## Übung 3

### 1.6 Source-Code(Implementation Data Model)

Folgend ist der Source der Daten Modelle angeführt.

Dieses Model kann dann in weiterer Folge zu JPA Entitäten umgewandelt werden.

#### 1.6.1 AbstractEntity.java

Folgende abstrakte Klasse stellt die Basis Entität für alle Entitäten dar und implementiert bereits `_getId()`; `setId()`;, wobei `_getId()` dafür gedacht ist um die private Datenkomponente id zu erhalten sondern um die Subklassen dazu zu zwingen, dass sie `getId()` überschreiben und ein `@Id` Mapping definieren. (JPA relevant `@Id`, `@EmbeddedId`)

Listing 15: AbstractEntity.java

```

1 package at.fh.ooe.swe4.fx.campina.jpa.api;
2
3 import java.io.Serializable;
4
5 public abstract class AbstractEntity<I extends Serializable> implements Serializable {
6
7     private static final long serialVersionUID = 1095329951571671581L;
8
9     private I id;
10
11     public AbstractEntity() {
12     }
13
14     public AbstractEntity(I id) {
15         super();
16         this.id = id;
17     }
18
19     public abstract I getId();
20
21     protected I _getId() {
22         return id;
23     }
24
25     public void setId(I id) {
26         this.id = id;
27     }
28
29     @Override
30     public int hashCode() {
31         final int prime = 31;
32         int result = 1;
33         result = prime * result + ((id == null) ? 0 : id.hashCode());
34         return result;
35     }
36
37     @Override
38     public boolean equals(Object obj) {
39         if (this == obj)
40             return true;
41         if (obj == null)
42             return false;
43         if (getClass() != obj.getClass())
44             return false;
45         AbstractEntity<I> other = (AbstractEntity<I>) obj;
46         if (id == null) {
47             if (other.id != null)
48                 return false;

```

Übung 3

---

```
49     } else if (!id.equals(other.id))
50         return false;
51     return true;
52 }
53
54 }
```

## Übung 3

### 1.6.2 Day.java

Folgende Enumeration spezifiziert die zur Verfügung stehenden Tage und auch einen Label, der in einen Produktivsystem über Keys in Form von String, Enumeration abgebildet werden sollte um Internationalisierung zu realisieren.

Dieser Datentyp kann in JPA mit Hibernate nativ gemapped werden ansonsten müsste man die Enumeration in einen String `enum.name()` serialisieren und wieder zu einer Enumeration de-serialisieren, was sich aber als nicht schwierig herausstellen sollte.

Listing 16: Day.java

```

1  package at.fh.ooe.swe4.fx.campina.jpa.constants;
2
3  import java.util.Objects;
4
5  /**
6   * Enumeration which specifies the available days.
7   * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
8   * @date Jun 5, 2015
9   */
10 public enum Day {
11     MONDAY("Montag"),
12     THUESDAY("Dienstag"),
13     WEDNESDAY("Mittwoch"),
14     THURSDAY("Donnerstag"),
15     FRIDAY("Freitag"),
16     SATARDAY("Samstag"),
17     SUNDAY("Sonntag");
18
19     public final String label;
20
21     private Day(String label) {
22         this.label = label;
23     }
24
25     /**
26      * Resolve {@link Day} by its held label.
27      *
28      * @param label
29      *         the label of the enumeration
30      * @return the found {@link Day} instance
31      * @throws NullPointerException
32      *         if the label is null
33      */
34     public static final Day fromLabel(final String label) {
35         Objects.requireNonNull(label);
36
37         for (Day day : Day.values()) {
38             if (day.label.equals(label)) {
39                 return day;
40             }
41         }
42         throw new IllegalArgumentException("No day with label '" + label + "' found");
43     }
44 }

```

## Übung 3

### 1.6.3 User.java

Folgende Klasse stellt den User auf der Datenbank dar.

Listing 17: User.java

```

1 package at.fh.ooe.swe4.fx.campina.jpa;
2
3 import java.util.HashSet;
4 import java.util.Set;
5
6 import at.fh.ooe.swe4.fx.campina.jpa.api.AbstractEntity;
7
8 /**
9  * The campina user.
10  *
11  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
12  * @date Jun 5, 2015
13  */
14 public class User extends AbstractEntity<Integer> {
15
16     private static final long serialVersionUID = 94057306870207307L;
17
18     private String      firstName;
19     private String      lastName;
20     private String      email;
21     private String      username;
22     private String      password;
23     private Boolean      adminFlag      = Boolean.FALSE;
24     private Boolean      blockedFlag    = Boolean.FALSE;
25     private Set<Order>   orders        = new HashSet<>();
26
27     /**
28      *
29      */
30     public User() {
31     }
32
33     /**
34      * @param id
35      * @param firstName
36      * @param lastName
37      * @param email
38      */
39     public User(Integer id, String firstName, String lastName, String email) {
40         super(id);
41         this.firstName = firstName;
42         this.lastName = lastName;
43         this.email = email;
44     }
45
46     public Integer getId() {
47         return _getId();
48     }
49
50     public String getFirstName() {
51         return firstName;
52     }
53
54     public void setFirstName(String firstName) {
55         this.firstName = firstName;
56     }
57
58     public String getLastName() {

```

# Übung 3

```

59     return lastName;
60 }
61
62 public void setLastName(String lastName) {
63     this.lastName = lastName;
64 }
65
66 public String getEmail() {
67     return email;
68 }
69
70 public void setEmail(String email) {
71     this.email = email;
72 }
73
74 public String getUsername() {
75     return username;
76 }
77
78 public void setUsername(String username) {
79     this.username = username;
80 }
81
82 public String getPassword() {
83     return password;
84 }
85
86 public void setPassword(String password) {
87     this.password = password;
88 }
89
90 public Boolean getAdminFlag() {
91     return adminFlag;
92 }
93
94 public void setAdminFlag(Boolean adminFlag) {
95     this.adminFlag = adminFlag;
96 }
97
98 public Boolean getBlockedFlag() {
99     return blockedFlag;
100 }
101
102 public void setBlockedFlag(Boolean blockedFlag) {
103     this.blockedFlag = blockedFlag;
104 }
105
106 public Set<Order> getOrders() {
107     return orders;
108 }
109
110 public void setOrders(Set<Order> orders) {
111     this.orders = orders;
112 }
113
114 }

```

## Übung 3

### 1.6.4 Menu.java

Folgende Klasse stellt den Menu auf der Datenbank dar.

Listing 18: Menu.java

```

1 package at.fh.ooe.swe4.fx.campina.jpa;
2
3 import java.util.SortedSet;
4 import java.util.TreeSet;
5
6 import at.fh.ooe.swe4.fx.campina.jpa.api.AbstractEntity;
7 import at.fh.ooe.swe4.fx.campina.jpa.constants.Day;
8
9 /**
10  * The menu.
11  *
12  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
13  * @date Jun 5, 2015
14  */
15 public class Menu extends AbstractEntity<Integer> implements Comparable<Menu> {
16
17     private static final long serialVersionUID = 7569033478037865818L;
18
19     private Day day;
20     private String label;
21     private SortedSet<MenuEntry> entires = new TreeSet<>();
22
23     /**
24      *
25      */
26     public Menu() {
27         super();
28     }
29
30     /**
31      *
32      * @param id
33      */
34     public Menu(Integer id) {
35         super(id);
36     }
37
38     /**
39      * @param id
40      * @param day
41      * @param label
42      */
43     public Menu(Integer id, Day day, String label) {
44         super(id);
45         this.day = day;
46         this.label = label;
47     }
48
49     @Override
50     public Integer getId() {
51         return _getId();
52     }
53
54     public Day getDay() {
55         return day;
56     }
57
58     public void setDay(Day day) {

```



## Übung 3

```

59     this.day = day;
60 }
61
62 public String getLabel() {
63     return label;
64 }
65
66 public void setLabel(String label) {
67     this.label = label;
68 }
69
70 public SortedSet<MenuEntry> getEntries() {
71     return entires;
72 }
73
74 public void setEntires(SortedSet<MenuEntry> entires) {
75     this.entires = entires;
76 }
77
78 @Override
79 public int compareTo(Menu o) {
80     if ((getId() == null) && (o.getId() == null)) {
81         return 0;
82     } else if (getId() == null) {
83         return -1;
84     } else if (o.getId() == null) {
85         return 1;
86     } else {
87         return getId().compareTo(o.getId());
88         // return day.compareTo(o.getDay());
89     }
90 }
91 }

```

## Übung 3

### 1.6.5 MenuEntry.java

Folgende Klasse stellt den Menu Eintrag auf der Datenbank dar.

Ein Menu (Fischtag) kann mehrere Gerichte haben (Forelle, Zander, ...)

Listing 19: MenuEntry.java

```

1 package at.fh.ooe.swe4.fx.campina.jpa;
2
3 import java.math.BigDecimal;
4 import java.util.HashSet;
5 import java.util.Set;
6
7 import at.fh.ooe.swe4.fx.campina.jpa.api.AbstractEntity;
8
9 /**
10  * The menu entry
11  *
12  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
13  * @date Jun 5, 2015
14  */
15 public class MenuEntry extends AbstractEntity<Integer> implements Comparable<MenuEntry> {
16
17     private static final long serialVersionUID = -1771596592654083392L;
18
19     private Integer ordinal;
20     private String label;
21     private BigDecimal price;
22     private Menu menu;
23     private Set<Order> orders = new HashSet<>();
24
25     /**
26      *
27      */
28     public MenuEntry() {
29         super();
30     }
31
32     /**
33      * @param id
34      */
35     public MenuEntry(Integer id) {
36         super(id);
37     }
38
39     /**
40      * @param id
41      * @param ordinal
42      * @param label
43      * @param price
44      * @param menu
45      */
46     public MenuEntry(Integer id, Integer ordinal, String label, BigDecimal price, Menu menu) {
47         this(id);
48         this.ordinal = ordinal;
49         this.label = label;
50         this.price = price;
51         this.menu = menu;
52     }
53
54     @Override
55     public Integer getId() {
56         return _getId();

```

# Übung 3

```

57     }
58
59     public Integer getOrdinal() {
60         return ordinal;
61     }
62
63     public void setOrdinal(Integer ordinal) {
64         this.ordinal = ordinal;
65     }
66
67     public String getLabel() {
68         return label;
69     }
70
71     public void setLabel(String label) {
72         this.label = label;
73     }
74
75     public BigDecimal getPrice() {
76         return price;
77     }
78
79     public void setPrice(BigDecimal price) {
80         this.price = price;
81     }
82
83     public Menu getMenu() {
84         return menu;
85     }
86
87     public void setMenu(Menu menu) {
88         this.menu = menu;
89     }
90
91     public Set<Order> getOrders() {
92         return orders;
93     }
94
95     public void setOrders(Set<Order> orders) {
96         this.orders = orders;
97     }
98
99     @Override
100    public int compareTo(MenuEntry o) {
101        if ((getId() == null) && (o.getId() == null)) {
102            return 0;
103        } else if (getId() == null) {
104            return -1;
105        } else if (o.getId() == null) {
106            return 1;
107        } else {
108            return getId().compareTo(o.getId());
109            // return day.compareTo(o.getDay());
110        }
111    }
112 }

```

## Übung 3

### 1.6.6 Order.java

Folgende Klasse stellt die Bestellung auf der Datenbank dar.  
Ein Menu (Fischtag) kann mehrere Gerichte haben (Forelle, Zander, ...)

Listing 20: Order.java

```

1 package at.fh.ooe.swe4.fx.campina.jpa;
2
3 import java.util.Calendar;
4
5 import at.fh.ooe.swe4.fx.campina.jpa.api.AbstractEntity;
6
7 /**
8  * The order on the database.
9  *
10 * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
11 * @date Jun 5, 2015
12 */
13 public class Order extends AbstractEntity<Integer> {
14
15     private static final long serialVersionUID = -4218683353334647328L;
16
17     private User user;
18     private MenuEntry menuEntry;
19     private Calendar orderDate;
20     private Calendar collectDate;
21
22     public Order(Integer id, User user, MenuEntry menuEntry, Calendar orderDate, Calendar
23 ↪ collectDate) {
24         super(id);
25         this.user = user;
26         this.menuEntry = menuEntry;
27         this.orderDate = orderDate;
28         this.collectDate = collectDate;
29     }
30
31     /**
32     */
33     public Order() {
34     }
35
36     /**
37     * @param id
38     */
39     public Order(Integer id) {
40         super(id);
41     }
42
43     @Override
44     public Integer getId() {
45         return _getId();
46     }
47
48     public User getUser() {
49         return user;
50     }
51
52     public void setUser(User user) {
53         this.user = user;
54     }
55

```

## Übung 3

```
56 public MenuEntry getMenuEntry() {
57     return menuEntry;
58 }
59
60 public void setMenuEntry(MenuEntry menuEntry) {
61     this.menuEntry = menuEntry;
62 }
63
64 public Calendar getOrderDate() {
65     return orderDate;
66 }
67
68 public void setOrderDate(Calendar orderDate) {
69     this.orderDate = orderDate;
70 }
71
72 public Calendar getCollectDate() {
73     return collectDate;
74 }
75
76 public void setCollectDate(Calendar collectDate) {
77     this.collectDate = collectDate;
78 }
79 }
```

## Übung 3

### 1.6.7 LoginEvent.java

Folgende Klasse stellt den Login Eintrag auf der Datenbank dar.  
Ermöglicht die Nachverfolgbarkeit der Logins der Benutzer.

Listing 21: LoginEvent.java

```

1  package at.fh.ooe.swe4.fx.campina.jpa;
2
3  import at.fh.ooe.swe4.fx.campina.jpa.api.AbstractEntity;
4
5  /**
6   * The login event which represents an valid login maybe an invalid too.
7   *
8   * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
9   * @date Jun 5, 2015
10  */
11 public class LoginEvent extends AbstractEntity<Integer> {
12
13     private static final long serialVersionUID = 1782498743658504195L;
14
15     // TODO: Add user references and calendar fields for login date
16
17     /**
18      *
19      */
20     public LoginEvent() {
21         this(null);
22     }
23
24     /**
25      * @param id
26      */
27     public LoginEvent(Integer id) {
28         super(id);
29     }
30
31     @Override
32     public Integer getId() {
33         return _getId();
34     }
35 }

```

## Übung 3

### 1.6.8 EntityCache.java

Folgende Klasse dient zu Simulation einer Datenbank damit Daten innerhalb der Laufzeit persistent gehalten werden können.

Listing 22: EntityCache.java

```

1 package at.fh.ooe.swe4.fx.campina.jpa;
2
3 import java.io.Serializable;
4 import java.math.BigDecimal;
5 import java.util.Calendar;
6 import java.util.HashSet;
7 import java.util.Iterator;
8 import java.util.Objects;
9 import java.util.Set;
10
11 import at.fh.ooe.swe4.fx.campina.jpa.constants.Day;
12 import at.fh.ooe.swe4.fx.campina.view.admin.login.model.LoginModel;
13
14 public class EntityCache {
15
16     public static Set<User>      userCache      = new HashSet<>(100);
17     public static Set<Menu>      menuCache      = new HashSet<>(100);
18     public static Set<MenuEntry> menuEntryCache = new HashSet<>(100);
19     public static Set<Order>     orderCache     = new HashSet<>(100);
20
21     static {
22         final User u1 = new User(1, "Thomas", "Herzog", "t.herzog@bla.bla");
23         final User u2 = new User(2, "Hugo", "Fichtner", "h.fichtner@bla.bla");
24         final User u3 = new User(3, "Christian", "Beikov", "c.beickov@bla.bla");
25         final User u4 = new User(4, "Rainer", "Rudolf", "r.rudolf@bla.bla");
26         final User u5 = new User(5, "Bernd", "Maier", "b.maier@bla.bla");
27
28         final Menu m1 = new Menu(1, Day.MONDAY, "Fisch Tag");
29         final Menu m2 = new Menu(1, Day.WEDNESDAY, "Steak Tag");
30         final Menu m3 = new Menu(1, Day.THURSDAY, "Nudel Tag");
31
32         final MenuEntry me1 = new MenuEntry(1, 1, "Forelle", BigDecimal.ONE, m1);
33         final MenuEntry me2 = new MenuEntry(2, 2, "Zander", BigDecimal.ONE, m1);
34         m1.getEntries()
35             .add(me1);
36         m1.getEntries()
37             .add(me2);
38
39         final Order o1 = new Order(1, u1, me1, Calendar.getInstance(), Calendar.getInstance());
40         me1.getOrders()
41             .add(o1);
42         u1.getOrders()
43             .add(o1);
44         final Order o2 = new Order(2, u2, me2, Calendar.getInstance(), Calendar.getInstance());
45         me1.getOrders()
46             .add(o1);
47         u2.getOrders()
48             .add(o1);
49
50         userCache.add(u1);
51         userCache.add(u2);
52         userCache.add(u3);
53         userCache.add(u4);
54         userCache.add(u5);
55
56         menuEntryCache.add(me1);

```

# Übung 3

```

57     menuEntryCache.add(me2);
58
59     menuCache.add(m1);
60     menuCache.add(m2);
61     menuCache.add(m3);
62
63     orderCache.add(o1);
64     orderCache.add(o2);
65 }
66
67 private EntityCache() {
68 }
69
70 public static final User isValidLogin(final LoginModel model) {
71     Objects.requireNonNull(model);
72
73     final Iterator<User> it = userCache.iterator();
74     boolean valid = Boolean.FALSE;
75     User user = null;
76     while ((it.hasNext()) && (!valid)) {
77         user = it.next();
78         valid = ((model.getUsername().equals(user.getUsername())) &&
79             ↪ (model.getPassword().equals(user.getPassword())));
80     }
81     return valid ? user : null;
82 }
83
84 public static final Menu byMenuId(final Serializable id) {
85     Objects.requireNonNull(id);
86
87     for (Menu menu : menuCache) {
88         if (id.equals(menu.getId())) {
89             return menu;
90         }
91     }
92     throw new IllegalArgumentException("id does not correspond to an menu");
93 }
94
95 public static final void deleteForMenuId(final Serializable id) {
96     Objects.requireNonNull(id);
97
98     final Iterator<MenuEntry> it = menuEntryCache.iterator();
99     while (it.hasNext()) {
100         final MenuEntry entry = it.next();
101         if (entry.getId()
102             .equals(id)) {
103             it.remove();
104         }
105     }
106 }
107 }

```



## Übung 3

### 1.7 Source-Code(Implementation Login View)

Folgend ist der Source der Login View angeführt.

#### 1.7.1 LoginModel.java

Folgend ist das Login Model angeführt welches für das Login Formular verwendet wird.

Listing 23: LoginModel.java

```

1 package at.fh.ooe.swe4.fx.campina.view.admin.login.model;
2
3 import java.util.Objects;
4
5 import at.fh.ooe.swe4.fx.campina.jpa.LoginEvent;
6 import at.fh.ooe.swe4.fx.campina.view.annotation.FormField;
7 import at.fh.ooe.swe4.fx.campina.view.api.AbstractViewModel;
8 import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;
9
10 /**
11  * The model backing the {@link LoginEvent} entity.
12  *
13  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
14  * @date Jun 5, 2015
15  */
16 public class LoginModel extends AbstractViewModel<Integer, LoginEvent> {
17
18     private Integer      counter          = 0;
19     private String       username         = "hello";
20     private String       password         = "hello";
21
22     public static final Integer MAX_COUNTER          = Integer.valueOf(5);
23     public static final String LOGIN_GREETING_MESSAGE_PATTERN = "Sie haben %d Versuche um sich
24     ↪ einzuloggen";
25     public static final String LOGIN_FAILED_MESSAGE_PATTERN = "Login ungültig !!! %d Versuche
26     ↪ verbleiben";
27     public static final String LOGIN_BLOCKED_MESSAGE      = "Zu viele Fehlversuche !!! Zugang
28     ↪ gesperrt";
29
30     @Override
31     public void reset() {
32         prepare(new LoginEvent());
33     }
34
35     @Override
36     public void prepare(LoginEvent loginEvent) {
37         Objects.requireNonNull(loginEvent);
38
39         setEntity(loginEvent);
40         setId(loginEvent.getId());
41     }
42
43     public void increaseCounter() {
44         counter++;
45     }
46
47     @FormField(
48         id = "login-username",
49         label = "Benutzernamen",
50         ordinal = 1,
51         required = true,
52         requiredMessage = "Bitte Benutzernamen angeben",
53         type = FormFieldType.INPUT_TEXT)

```

## Übung 3

```
51 public String getUsername() {
52     return username;
53 }
54
55 public void setUsername(String username) {
56     this.username = username;
57 }
58
59 @FormField(
60     id = "login-password",
61     label = "Passwort",
62     ordinal = 1,
63     required = true,
64     requiredMessage = "Bitte Passwort angeben",
65     type = FormFieldType.INPUT_TEXT)
66 public String getPassword() {
67     return password;
68 }
69
70 public void setPassword(String password) {
71     this.password = password;
72 }
73
74 public Integer getCounter() {
75     return counter;
76 }
77
78 }
```

## Übung 3

### 1.7.2 LoginEventControl.java

Folgend ist die Klasse angeführt, welche die Aktionen behandelt, wie Speichern, Löschen, usw.

Listing 24: LoginEventControl.java

```

1 package at.fh.ooe.swe4.fx.campina.view.admin.login.control;
2
3 import javafx.event.ActionEvent;
4 import javafx.scene.Node;
5 import javafx.scene.control.Button;
6 import javafx.scene.text.Text;
7 import javafx.scene.text.TextFlow;
8
9 import org.apache.commons.lang.StringUtils;
10
11 import at.fh.ooe.swe4.fx.campina.jpa.EntityCache;
12 import at.fh.ooe.swe4.fx.campina.jpa.User;
13 import at.fh.ooe.swe4.fx.campina.view.admin.login.model.LoginModel;
14 import at.fh.ooe.swe4.fx.campina.view.admin.login.part.LoginTabViewHandler;
15 import at.fh.ooe.swe4.fx.campina.view.api.FormContext;
16
17 /**
18  * This class contains all of the action logic:
19  *
20  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
21  * @date Jun 5, 2015
22  */
23 public class LoginEventControl {
24
25     private User loggedUser = null;
26
27     /**
28      *
29      */
30     public LoginEventControl() {
31     }
32
33     /**
34      * Handles the login event
35      *
36      * @param event the
37      *           {@link ActionEvent}
38      */
39     public void handleLogin(final ActionEvent event) {
40         final FormContext<LoginModel> ctx = (FormContext<LoginModel>) ((Node)
41             ↪ event.getSource()).getUserData();
42         populateFormMessage(null, ctx);
43         ctx.formHandler.validateForm(ctx);
44
45         if (ctx.valid) {
46             ctx.formHandler.fillModel(ctx);
47
48             // TODO: Search for username and password on db
49             loggedUser = EntityCache.isValidLogin(ctx.model);
50             // increase counter
51             if (loggedUser == null) {
52                 ctx.model.increaseCounter();
53                 if (ctx.model.getCounter() >= LoginModel.MAX_COUNTER) {
54                     populateFormMessage(LoginModel.LOGIN_BLOCKED_MESSAGE, ctx);
55                     ((Button) ctx.getNode(LoginTabViewHandler.LOGIN_BUTTON_ID)).setDisable(Boolean.TRUE);
56                 } else {
57                     populateFormMessage(String.format(LoginModel.LOGIN_FAILED_MESSAGE_PATTERN,
58                         ↪ (LoginModel.MAX_COUNTER - ctx.model.getCounter())), ctx);
59                 }
60             }
61         }
62     }
63 }

```

## Übung 3

```

57     }
58   }
59 }
60 // for invalid
61 else {
62     populateFormMessage("Formular ungültig !!! Bitte Eingaben prüfen", ctx);
63 }
64 }
65
66 /**
67  * Populates a message to the message box. <br>
68  * If message is null the actual set message will be cleared
69  *
70  * @param message
71  *           the message to populate
72  * @param ctx
73  *           the form context
74  */
75 private void populateFormMessage(final String message, final FormContext<LoginModel> ctx) {
76     final TextFlow flow = ((TextFlow) ctx.getNode(LoginTabViewHandler.FORM_MESSAGE));
77     flow.getChildren()
78         .clear();
79     flow.setPrefHeight(0);
80     if (!StringUtils.isEmpty(message)) {
81         flow.getChildren()
82             .add(new Text(message));
83         flow.setPrefHeight(30);
84     }
85 }
86 }

```

## Übung 3

### 1.7.3 LoginTabViewHandler.java

Folgend ist der View Handler der LoginTab angeführt. Sie stellt einen View Teil der Scene dar und implementiert `ViewHandler<T>` Interface.

Listing 25: LoginTabViewHandler.java

```

1  package at.fh.ooe.swe4.fx.campina.view.admin.login.part;
2
3  import java.util.Objects;
4
5  import javafx.scene.Node;
6  import javafx.scene.Scene;
7  import javafx.scene.control.Button;
8  import javafx.scene.control.Tab;
9  import javafx.scene.layout.GridPane;
10 import javafx.scene.text.Text;
11 import javafx.scene.text.TextFlow;
12 import at.fh.ooe.swe4.fx.campina.view.admin.login.control.LoginEventControl;
13 import at.fh.ooe.swe4.fx.campina.view.admin.login.model.LoginModel;
14 import at.fh.ooe.swe4.fx.campina.view.api.FormContext;
15 import at.fh.ooe.swe4.fx.campina.view.api.ViewHandler;
16 import at.fh.ooe.swe4.fx.campina.view.form.FormHandler;
17
18 /**
19  * {@link ViewHandler} implementation for the login tab.
20  *
21  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
22  * @date Jun 5, 2015
23  */
24 public class LoginTabViewHandler implements ViewHandler<Tab> {
25
26     /**
27      * The form builder for the user form
28      */
29     private final FormHandler<LoginModel> formHandler;
30
31     /**
32      * The user form controller
33      */
34     private final LoginEventControl loginControl;
35
36     /**
37      * The form context for the user form
38      */
39     final FormContext<LoginModel> ctx;
40
41     // #####
42     // Ids of nodes and form backed model class
43     // #####
44     public static final String LOGIN_FORM_ID = "login-form";
45     public static final String FORM_MESSAGE = "login-form-message";
46     public static final String LOGIN_BUTTON_ID = "login-button-login";
47
48     /**
49      * @param scene
50      */
51     public LoginTabViewHandler(final Scene scene) {
52         Objects.requireNonNull(scene);
53
54         final LoginModel model = new LoginModel();
55         model.reset();
56         this.formHandler = new FormHandler<LoginModel>().init();
57         this.loginControl = new LoginEventControl();
58         this.ctx = new FormContext<LoginModel>(LOGIN_FORM_ID, formHandler, model, scene);

```

# Übung 3

```

57     }
58
59     // #####
60     // ScenePart methods
61     // #####
62     @Override
63     public String getId() {
64         return "login-tab";
65     }
66
67     @Override
68     public Tab createNode() {
69         // Form message text
70         final TextFlow flow = new TextFlow();
71         flow.setId(FORM_MESSAGE);
72         flow.getChildren()
73             .add(new Text(String.format(LoginModel.LOGIN_GREETING_MESSAGE_PATTERN,
74                 ↪ LoginModel.MAX_COUNTER)));
75         flow.setStyle("-fx-font-size: 20pt");
76         ctx.putNode(FORM_MESSAGE, flow);
77
78         // Form
79         final Node form = formHandler.generateFormGrid(ctx);
80
81         // Buttons
82         final Button loginButton = new Button();
83         loginButton.setId(LOGIN_BUTTON_ID);
84         loginButton.setText("Login");
85         loginButton.setUserData(ctx);
86         loginButton.setOnAction(loginControl::handleLogin);
87         ctx.putNode(LOGIN_BUTTON_ID, loginButton);
88
89         final GridPane pane = new GridPane();
90         pane.setId("user-tab-content");
91
92         // left part of tab
93         final GridPane mainGrid = new GridPane();
94         mainGrid.setId("login-form-container");
95         mainGrid.add(flow, 0, 0);
96         mainGrid.add(form, 0, 1);
97         mainGrid.add(loginButton, 0, 2);
98         mainGrid.setPrefHeight(500);
99         pane.add(mainGrid, 0, 0);
100
101         final Tab tab = new Tab(getId());
102         tab.setText("Login");
103         tab.setClosable(false);
104         tab.setContent(pane);
105
106         return tab;
107     }
108
109     @Override
110     public void initHandler() {
111         ctx.formHandler.fillForm(ctx);
112     }
113 }

```

## Übung 3

### 1.8 Source-Code(Implementation User View)

Folgend ist der Source der User View angeführt.

#### 1.8.1 UserModel.java

Folgend ist das User Model angeführt welches für das User Formular verwendet wird.

Listing 26: UserModel.java

```

1 package at.fh.ooe.swe4.fx.campina.view.admin.user.model;
2
3 import java.util.Objects;
4
5 import at.fh.ooe.swe4.fx.campina.jpa.User;
6 import at.fh.ooe.swe4.fx.campina.view.annotation.FormField;
7 import at.fh.ooe.swe4.fx.campina.view.api.AbstractViewModel;
8 import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;
9
10 /**
11  * The view model which backs the {@link User} entity.
12  *
13  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
14  * @date Jun 5, 2015
15  */
16 public class UserModel extends AbstractViewModel<Integer, User> {
17
18     /**
19      *
20      */
21     public UserModel() {
22         super();
23     }
24
25     /**
26      * @param user
27      */
28     public UserModel(User user) {
29         super(user);
30     }
31
32     @Override
33     public void prepare(User user) {
34         Objects.requireNonNull(user);
35
36         setId(user.getId());
37         setEntity(user);
38     }
39
40     @Override
41     public void reset() {
42         prepare(new User());
43     }
44
45     public String getSelectionName() {
46         if (getId() == null) {
47             return "Neuer Benutzer";
48         } else {
49             return new StringBuilder().append(getLastName())
50                 .append(", ")
51                 .append(getFirstName())
52                 .append(" (")
53                 .append(getEmail())

```

# Übung 3

```

54         .append(" ")
55         .toString();
56     }
57 }
58
59 @FormField(
60     id = "user-first-name",
61     label = "Vorname",
62     required = true,
63     ordinal = 1,
64     requiredMessage = "Bitte Vornamen angeben",
65     type = FormFieldType.INPUT_TEXT)
66 public String getFirstName() {
67     return getEntity().getFirstName();
68 }
69
70 public void setFirstName(String firstName) {
71     getEntity().setFirstName(firstName);
72     ;
73 }
74
75 @FormField(
76     id = "user-last-name",
77     label = "Nachname",
78     required = true,
79     ordinal = 2,
80     requiredMessage = "Bitte Nachnamen angeben",
81     type = FormFieldType.INPUT_TEXT)
82 public String getLastName() {
83     return getEntity().getLastName();
84 }
85
86 public void setLastName(String lastName) {
87     getEntity().setLastName(lastName);
88     ;
89 }
90
91 @FormField(
92     id = "user-email",
93     label = "E-Mail",
94     ordinal = 3,
95     type = FormFieldType.INPUT_TEXT)
96 public String getEmail() {
97     return getEntity().getEmail();
98 }
99
100 public void setEmail(String email) {
101     getEntity().setEmail(email);
102 }
103
104 @FormField(
105     id = "user-username",
106     label = "Benutzername",
107     required = true,
108     ordinal = 4,
109     requiredMessage = "Bitte Benutzernamen angeben",
110     type = FormFieldType.INPUT_TEXT)
111 public String getUsername() {
112     return getEntity().getUsername();
113 }
114
115 public void setUsername(String username) {
116     getEntity().setUsername(username);

```



## Übung 3

```
117     }
118
119     @FormField(
120         id = "user-password",
121         label = "Password",
122         required = true,
123         ordinal = 5,
124         requiredMessage = "Bitte Password angeben",
125         type = FormFieldType.INPUT_TEXT)
126     public String getPassword() {
127         return getEntity().getPassword();
128     }
129
130     public void setPassword(String password) {
131         getEntity().setPassword(password);
132     }
133
134 }
```

## Übung 3

### 1.8.2 UserEventControl.java

Folgend ist die Klasse angeführt, welche die Aktionen behandelt, wie Speichern, Löschen, usw.

Listing 27: UserEventControl.java

```

1  package at.fh.ooe.swe4.fx.campina.view.admin.user.control;
2
3  import java.util.Objects;
4
5  import javafx.collections.ObservableList;
6  import javafx.event.ActionEvent;
7  import javafx.scene.Node;
8  import javafx.scene.control.Button;
9  import javafx.scene.control.ChoiceBox;
10 import javafx.scene.text.Text;
11 import javafx.scene.text.TextFlow;
12
13 import org.apache.commons.lang.StringUtils;
14
15 import at.fh.ooe.swe4.fx.campina.jpa.EntityCache;
16 import at.fh.ooe.swe4.fx.campina.jpa.User;
17 import at.fh.ooe.swe4.fx.campina.view.admin.user.model.UserModel;
18 import at.fh.ooe.swe4.fx.campina.view.admin.user.part.UserTabviewHandler;
19 import at.fh.ooe.swe4.fx.campina.view.api.FormContext;
20
21 /**
22  * The control bean for the user tab.
23  *
24  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
25  * @date Jun 3, 2015
26  */
27 public class UserEventControl {
28
29     /**
30      * Creates test data since we have no back-end yet
31      */
32     public UserEventControl() {
33     }
34
35     // #####
36     // Button controls
37     // #####
38     /**
39      * Handles the new action of the form.
40      *
41      * @param event
42      *         the {@link ActionEvent}
43      */
44     public void handleNewAction(final ActionEvent event) {
45         final FormContext<UserModel> ctx = (FormContext<UserModel>) ((Node)
46             ↪ event.getSource()).getUserData();
47         // clear former set message
48         populateFormMessage(null, ctx);
49         // reset form
50         ctx.formHandler.resetForm(ctx);
51         // create new user model with new user entity
52         ctx.model.reset();
53         // hide buttons
54         setButtonVisibility(ctx, Boolean.FALSE);
55         // reload users
56         handleUserReload(ctx);
57     }

```

## Übung 3

```

58  /**
59   * Handles the save action of the form.
60   *
61   * @param event
62   *         the {@link ActionEvent}
63   */
64  public void handleSaveAction(final ActionEvent event) {
65      final FormContext<UserModel> ctx = (FormContext<UserModel>) ((Node)
66          ↪ event.getSource()).getUserData();
67      // clear former set messages
68      populateFormMessage(null, ctx);
69      // validate form
70      ctx.formHandler.validateForm(ctx);
71      // is valid
72      if (ctx.valid) {
73          // fill model with form data
74          ctx.formHandler.fillModel(ctx);
75
76          // TODO: Persist entity here
77          final User user = ctx.model.getEntity();
78
79          // if not already managed increase id by size + 1
80          if (!EntityCache.userCache.contains(user)) {
81              user.setId(EntityCache.userCache.size() + 1);
82          }
83          // save model in backed list for testing
84          EntityCache.userCache.add(user);
85          // init model with new saved user
86          ctx.model.prepare(user);
87          // enable buttons
88          setButtonVisibility(ctx, Boolean.TRUE);
89          // reload data from db (now backing list)
90          handleUserReload(ctx);
91      } else {
92          populateFormMessage("Formular ungültig !!! Bitte Eingaben prüfen", ctx);
93      }
94      event.consume();
95  }
96
97  /**
98   * Handles the delete action of the form.
99   *
100   * @param event
101   *        the {@link ActionEvent}
102   */
103  public void handleDeleteAction(final ActionEvent event) {
104      final FormContext<UserModel> ctx = (FormContext<UserModel>) ((Node)
105          ↪ event.getSource()).getUserData();
106      // clear former set message
107      populateFormMessage(null, ctx);
108      // reset the form
109      ctx.formHandler.resetForm(ctx);
110
111      final UserModel model = ((ChoiceBox<UserModel>)
112          ↪ ctx.getNode(UserTabviewHandler.USER_SELECTION_KEY)).getSelectionModel()
113          .getSelectedItem();
114
115      // TODO: Delete entity from db here
116
117      // existing user gets deleted
118      if (model.getId() != null) {
119          EntityCache.userCache.remove(model.getEntity());
120      }
121  }

```

# Übung 3

```

118     }
119
120     // reset model
121     ctx.model = new UserModel();
122     // disable buttons
123     setButtonVisibility(ctx, Boolean.FALSE);
124     // reload users
125     handleUserReload(ctx);
126 }
127
128 /**
129  * Handles the block action of the form.
130  *
131  * @param event
132  *      the {@link ActionEvent}
133  */
134 public void handleBlockAction(final ActionEvent event) {
135     final FormContext<UserModel> ctx = (FormContext<UserModel>) ((Node)
136         ↪ event.getSource()).getUserData();
137     // clear old set message
138     populateFormMessage(null, ctx);
139     // selected user model
140     final UserModel model = ((ChoiceBox<UserModel>)
141         ↪ ctx.getNode(UserTabviewHandler.USER_SELECTION_KEY)).getSelectionModel()
142         .getSelectedItem();
143
144     final Button blockButton = (Button) ctx.getNode(UserTabviewHandler.BLOCK_BUTTON_ID);
145     final User user = model.getEntity();
146
147     // invert user blocked state
148     user.setBlockedFlag(!model.getEntity()
149         .getBlockedFlag());
150
151     ctx.model.prepare(user);
152     ctx.formHandler.fillForm(ctx);
153
154     // TODO: Update blocked flag on db
155
156     // got blocked
157     if (model.getEntity()
158         .getBlockedFlag()) {
159         blockButton.setText("Freigeben");
160     }
161     // got freed
162     else {
163         blockButton.setText("Blockieren");
164     }
165     // user reload
166     handleUserReload(ctx);
167 }
168
169 // #####
170 // Selection controls
171 // #####
172 public void handleUserSelection(final FormContext<UserModel> ctx, final UserModel user) {
173     // clear former set message
174     populateFormMessage(null, ctx);
175     // Selection present
176     if (user.getId() != null) {
177         ctx.model.prepare(user.getEntity());
178         ctx.formHandler.fillForm(ctx);
179         setButtonVisibility(ctx, Boolean.TRUE);
180     }
181 }

```

# Übung 3

```

179 // No selection present
180 else {
181     ctx.model.reset();
182     ctx.formHandler.fillForm(ctx);
183     setButtonVisibility(ctx, Boolean.FALSE);
184 }
185 }
186
187 // #####
188 // Load controls
189 // #####
190 /**
191  * Handles the load of the user for the selection
192  *
193  * @param userList
194  *      the {@link ObservableList} to add users to
195  */
196 public void handleUserReload(final FormContext<UserModel> ctx) {
197     Objects.requireNonNull(ctx);
198
199     final ObservableList<UserModel> userList = (ObservableList<UserModel>)
200         ↪ ctx.getObservable(UserTabviewHandler.USER_SELECTION_KEY);
201     userList.clear();
202     userList.add(new UserModel());
203     for (User user : EntityCache.userCache) {
204         userList.add(new UserModel(user));
205     }
206
207     userList.set(userList.indexOf(ctx.model), ctx.model);
208
209     ((ChoiceBox<UserModel>)
210         ↪ ctx.getNode(UserTabviewHandler.USER_SELECTION_KEY)).getSelectionModel()
211         .select(ctx.model);
212 }
213
214 /**
215  * Sets the button visibility of these buttons which required persistent
216  * user.
217  *
218  * @param ctx
219  *      the form context
220  * @param visible
221  *      the new visible flag
222  */
223 private void setButtonVisibility(final FormContext<UserModel> ctx, final boolean visible) {
224     Objects.requireNonNull(ctx);
225
226     ctx.getNode(UserTabviewHandler.DELETE_BUTTON_ID)
227         .setVisible(visible);
228     ctx.getNode(UserTabviewHandler.BLOCK_BUTTON_ID)
229         .setVisible(visible);
230 }
231
232 /**
233  * Populates a message to the message box. <br>
234  * If message is null the actual set message will be cleared
235  *
236  * @param message
237  *      the message to populate
238  * @param ctx
239  *      the form context
240  */
241 private void populateFormMessage(final String message, final FormContext<UserModel> ctx) {

```

## Übung 3

```
240     final TextFlow flow = ((TextFlow) ctx.getNode(UserTabviewHandler.FORM_MESSAGE));
241     flow.getChildren()
242         .clear();
243     flow.setPrefHeight(0);
244     if (!StringUtils.isEmpty(message)) {
245         flow.getChildren()
246             .add(new Text(message));
247         flow.setPrefHeight(30);
248     }
249 }
250 }
```

## Übung 3

### 1.8.3 UserTabViewHandler.java

Folgend ist der ViewHandddler der UserTab angeführt. Sie stellt einen View Teil der Scene dar und implementiert ViewHandler<T> Interface.

Listing 28: UserTabViewHandler.java

```

1 package at.fh.ooe.swe4.fx.campina.view.admin.user.part;
2
3 import java.util.Arrays;
4 import java.util.Objects;
5
6 import javafx.beans.value.ChangeListener;
7 import javafx.beans.value.ObservableValue;
8 import javafx.collections.FXCollections;
9 import javafx.collections.ObservableList;
10 import javafx.scene.Scene;
11 import javafx.scene.control.Button;
12 import javafx.scene.control.ChoiceBox;
13 import javafx.scene.control.Tab;
14 import javafx.scene.layout.GridPane;
15 import javafx.scene.text.TextFlow;
16 import javafx.util.StringConverter;
17 import at.fh.ooe.swe4.fx.campina.view.admin.user.control.UserEventControl;
18 import at.fh.ooe.swe4.fx.campina.view.admin.user.model.UserModel;
19 import at.fh.ooe.swe4.fx.campina.view.api.FormContext;
20 import at.fh.ooe.swe4.fx.campina.view.api.ViewHandler;
21 import at.fh.ooe.swe4.fx.campina.view.form.FormHandler;
22
23 /**
24  * This class builds the user tab.
25  *
26  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
27  * @date Jun 3, 2015
28  */
29 public class UserTabviewHandler implements ViewHandler<Tab> {
30
31     /**
32      * The form builder for the user form
33      */
34     private final FormHandler<UserModel> formHandler;
35
36     /**
37      * The user form controller
38      */
39     private final UserEventControl userControl;
40
41     /**
42      * The form context for the user form
43      */
44     final FormContext<UserModel> ctx;
45
46     // #####
47     // Ids of nodes and form backed model class
48     // #####
49     public static final String USER_SELECTION_KEY = "user-selection";
50     public static final String SAVE_BUTTON_ID = "user-save-button";
51     public static final String NEW_BUTTON_ID = "user-new-button";
52     public static final String DELETE_BUTTON_ID = "user-delete-button";
53     public static final String BLOCK_BUTTON_ID = "user-block-button";
54     public static final String FORM_MESSAGE = "user-form-message";
55
56     /**
57      * Inits this user tab manager for the given scene
58      */

```

# Übung 3

```

57  * @param scene
58  *         the backing scene
59  */
60  public UserTabviewHandler(final Scene scene) {
61      Objects.requireNonNull(scene);
62
63      this.formHandler = new FormHandler<UserModel>();
64      this.formHandler.init();
65      this.userControl = new UserEventControl();
66      this.ctx = new FormContext<UserModel>("tab-user", formHandler, new UserModel(), scene);
67  }
68
69  // #####
70  // ScenePart methods
71  // #####
72  @Override
73  public String getId() {
74      return "user-admin";
75  }
76
77  @Override
78  public void initHandler() {
79      userControl.handleUserReload(ctx);
80      ((ChoiceBox<UserModel>) ctx.getNode(USER_SELECTION_KEY)).getSelectionModel()
81          .select(0);
82  }
83
84  @Override
85  public Tab createNode() {
86      final GridPane pane = new GridPane();
87      pane.setId(getId() + "-content");
88
89      // form
90      final GridPane formGrid = formHandler.generateFormGrid(ctx);
91
92      // left part of tab
93      final GridPane mainGrid = new GridPane();
94      mainGrid.setId("user-form");
95      mainGrid.add(createFormMessage(ctx), 0, 0);
96      mainGrid.add(createUserChoice(ctx), 0, 1);
97      mainGrid.add(formGrid, 0, 2);
98      mainGrid.add(createButtonGroup(ctx), 0, 3);
99      mainGrid.setPrefHeight(500);
100     mainGrid.setHgap(10);
101     mainGrid.setVgap(10);
102     pane.add(mainGrid, 0, 0);
103
104     final Tab tab = new Tab(getId());
105     tab.setText("Benutzerverwaltung");
106     tab.setClosable(false);
107     tab.setContent(pane);
108     return tab;
109 }
110
111 // #####
112 // Node creation methods
113 // #####
114 /**
115  * Creates the form message box where the form messages are placed
116  *
117  * @param ctx
118  *         the form context
119  * @return the {@link TextFlow} instance for the messages

```



# Übung 3

```

120  */
121  private TextFlow createFormMessage(final FormContext<UserModel> ctx) {
122      final TextFlow flow = new TextFlow();
123      flow.setId(getId() + "-user-form-message");
124      flow.setStyle("-fx-font-size: 20pt");
125      ctx.putNode(FORM_MESSAGE, flow);
126      return flow;
127  }
128
129  /**
130   * Creates the button group for the user tab actions.
131   *
132   * @param ctx
133   *      TODO
134   *
135   * @return a {@link GridPane} instance holding the button
136   */
137  private GridPane createButtonGroup(FormContext<UserModel> ctx) {
138      final GridPane gridPane = new GridPane();
139      gridPane.setId(getId() + "-button-grid");
140      gridPane.setHgap(10);
141      gridPane.setVgap(10);
142
143      // TODO: Register events here
144      final Button newButton = new Button();
145      newButton.setText("Zurücksetzen");
146      newButton.setId(NEW_BUTTON_ID);
147      newButton.setUserData(ctx);
148      newButton.setOnAction(userControl::handleNewAction);
149
150      final Button saveButton = new Button();
151      saveButton.setId(SAVE_BUTTON_ID);
152      saveButton.setText("Speichern");
153      saveButton.setUserData(ctx);
154      saveButton.setOnAction(userControl::handleSaveAction);
155
156      final Button deleteButton = new Button();
157      deleteButton.setId(DELETE_BUTTON_ID);
158      deleteButton.setText("Löschen");
159      deleteButton.setUserData(ctx);
160      deleteButton.setOnAction(userControl::handleDeleteAction);
161      deleteButton.setVisible(Boolean.FALSE);
162
163      final Button blockButton = new Button();
164      blockButton.setId(BLOCK_BUTTON_ID);
165      blockButton.setText("Sperren");
166      blockButton.setUserData(ctx);
167      blockButton.setOnAction(userControl::handleBlockAction);
168      blockButton.setVisible(Boolean.FALSE);
169
170      gridPane.add(newButton, 0, 0);
171      gridPane.add(saveButton, 1, 0);
172      gridPane.add(deleteButton, 2, 0);
173      gridPane.add(blockButton, 3, 0);
174
175      // register in context
176      ctx.putNode(NEW_BUTTON_ID, newButton);
177      ctx.putNode(SAVE_BUTTON_ID, saveButton);
178      ctx.putNode(DELETE_BUTTON_ID, deleteButton);
179      ctx.putNode(BLOCK_BUTTON_ID, blockButton);
180
181      return gridPane;
182  }

```

## Übung 3

```

183
184 /**
185  * Creates the user choice node for the user selection.
186  *
187  * @param ctx the
188  *         form context
189  * @return the {@link ChoiceBox} instance
190  */
191 private ChoiceBox<UserModel> createUserChoice(final FormContext<UserModel> ctx) {
192     Objects.requireNonNull(ctx);
193
194     final ObservableList<UserModel> users = FXCollections.observableArrayList(Arrays.asList(new
195         ↪ UserModel()));
196
197     final ChoiceBox<UserModel> userChoice = new ChoiceBox<>(users);
198     userChoice.setUserData(ctx);
199     userChoice.getSelectionModel()
200         .selectedItemProperty()
201         .addListener(new ChangeListener<UserModel>() {
202             @Override
203             public void changed(ObservableValue<? extends UserModel> observable, UserModel
204                 ↪ oldValue, UserModel newValue) {
205                 if (observable.getValue() == null) {
206                     return;
207                 }
208                 userControl.handleUserSelection(ctx, observable.getValue());
209             }
210         });
211     userChoice.setConverter(new StringConverter<UserModel>() {
212         @Override
213         public String toString(UserModel object) {
214             return object.getSelectionName();
215         }
216
217         @Override
218         public UserModel fromString(String string) {
219             throw new UnsupportedOperationException("Not supported conversion from string to object");
220         }
221     });
222     userChoice.setPrefWidth(400);
223     ctx.putObservable(USER_SELECTION_KEY, users);
224     ctx.putNode(USER_SELECTION_KEY, userChoice);
225     return userChoice;
226 }
227 }

```

## Übung 3

### 1.9 Source-Code(Implementation Menu View)

Folgend ist der Source der Menu View angeführt.

#### 1.9.1 MenuModel.java

Folgend ist das Menu Model angeführt welches für das Menu Formular verwendet wird.

Listing 29: MenuModel.java

```

1 package at.fh.ooe.swe4.fx.campina.view.admin.menu.model;
2
3 import java.util.Arrays;
4 import java.util.List;
5 import java.util.Objects;
6 import java.util.SortedSet;
7
8 import javafx.collections.FXCollections;
9 import javafx.collections.ObservableList;
10 import javafx.util.StringConverter;
11 import at.fh.ooe.swe4.fx.campina.jpa.Menu;
12 import at.fh.ooe.swe4.fx.campina.jpa.MenuEntry;
13 import at.fh.ooe.swe4.fx.campina.jpa.constants.Day;
14 import at.fh.ooe.swe4.fx.campina.view.annotation.FormField;
15 import at.fh.ooe.swe4.fx.campina.view.annotation.SelectFormField;
16 import at.fh.ooe.swe4.fx.campina.view.api.AbstractViewModel;
17 import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;
18
19 /**
20  * The view model which backs the {@link Menu} entity.
21  *
22  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
23  * @date Jun 5, 2015
24  */
25 public class MenuModel extends AbstractViewModel<Integer, Menu> {
26
27     private static final ObservableList<Day> days = FXCollections.observableArrayList();
28
29     static {
30         days.add((Day) null);
31         days.addAll(Arrays.asList(Day.values()));
32     }
33
34     /**
35      * The converter for the {@link Day} type.
36      *
37      * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
38      * @date Jun 5, 2015
39      */
40     public static class DayConverter extends StringConverter<Day> {
41
42         private static final String PLEASE_CHOOSE = "Bitte wählen";
43
44         public DayConverter() {
45             super();
46         }
47
48         @Override
49         public String toString(Day object) {
50             return (object == null) ? PLEASE_CHOOSE : object.label;
51         }
52
53         @Override

```

## Übung 3

```

54     public Day fromString(String string) {
55         return (PLEASE_CHOOSE.equals(string)) ? null : Day.fromLabel(string);
56     }
57
58 }
59
60 /**
61  * The converter fro the {@link MenuModel} type.
62  *
63  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
64  * @date Jun 5, 2015
65  */
66 public static class MenuModelConverter extends StringConverter<MenuModel> {
67
68     @Override
69     public String toString(MenuModel object) {
70         Objects.requireNonNull(object);
71         return (object.getId() == null) ? "Neues Menu" : object.getLabel();
72     }
73
74     @Override
75     public MenuModel fromString(String string) {
76         throw new UnsupportedOperationException("Form text not supported by this converter");
77     }
78
79 }
80
81 /**
82  *
83  */
84 public MenuModel() {
85     super();
86 }
87
88 /**
89  * @param entity
90  */
91 public MenuModel(Menu entity) {
92     super(entity);
93 }
94
95 @Override
96 public void reset() {
97     prepare(new Menu());
98 }
99
100 @Override
101 public void prepare(Menu entity) {
102     super.prepare(entity);
103     entity.setDay(days.get(days.indexOf(entity.getDay())));
104 }
105
106 @FormField(id = "menu-label",
107     label = "Label",
108     ordinal = 1,
109     required = true,
110     requiredMessage = "Bitte Menu Label angeben",
111     type = FormFieldType.INPUT_TEXT)
112 public String getLabel() {
113     return getEntity().getLabel();
114 }
115
116 public void setLabel(String label) {

```

## Übung 3

```

117     getEntity().setLabel(label);
118 }
119
120 @FormField(id = "menu-day",
121     label = "Tag",
122     ordinal = 6,
123     required = true,
124     requiredMessage = "Bitte Tag wählen",
125     type = FormFieldType.SELECT,
126     valueClass = Day.class)
127 public Day getDay() {
128     return getEntity().getDay();
129 }
130
131 public void setDay(Day day) {
132     getEntity().setDay(day);
133 }
134
135 public void setLabel(Day day) {
136     System.out.println(day);
137 }
138
139 @SelectFormField(target = "day",
140     converter = DayConverter.class)
141 public List<Day> getDays() {
142     return days;
143 }
144
145 public SortedSet<MenuEntry> getMenuEntries() {
146     return getEntity().getEntries();
147 }
148
149 }

```

## Übung 3

### 1.9.2 MenuEventControl.java

Folgend ist die Klasse angeführt, welche die Actions behandelt, wie Speichern, Löschen, usw.

Listing 30: MenuEventControl.java

```

1 package at.fh.ooe.swe4.fx.campina.view.admin.menu.control;
2
3 import java.util.Objects;
4
5 import javafx.collections.ObservableList;
6 import javafx.event.ActionEvent;
7 import javafx.scene.Node;
8 import javafx.scene.control.ChoiceBox;
9 import javafx.scene.text.Text;
10 import javafx.scene.text.TextFlow;
11
12 import org.apache.commons.lang.StringUtils;
13
14 import at.fh.ooe.swe4.fx.campina.jpa.EntityCache;
15 import at.fh.ooe.swe4.fx.campina.jpa.Menu;
16 import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuModel;
17 import at.fh.ooe.swe4.fx.campina.view.admin.menu.part.MenuTabViewHandler;
18 import at.fh.ooe.swe4.fx.campina.view.admin.user.model.UserModel;
19 import at.fh.ooe.swe4.fx.campina.view.api.FormContext;
20
21 /**
22  * The event control for the {@link Menu} entity.
23  *
24  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
25  * @date Jun 5, 2015
26  */
27 public class MenuEventControl {
28
29     public MenuEventControl() {
30         // TODO Auto-generated constructor stub
31     }
32
33     /**
34      * Handles the new action of the form.
35      *
36      * @param event
37      *         the {@link ActionEvent}
38      */
39     public void newAction(final ActionEvent event) {
40         final FormContext<MenuModel> ctx = (FormContext<MenuModel>) ((Node)
41             ↪ event.getSource()).getUserData();
42         // clear former set message
43         populateFormMessage(null, ctx);
44         // reset form
45         ctx.formHandler.resetForm(ctx);
46         // create new user model with new user entity
47         ctx.model.reset();
48         // hide buttons
49         ctx.getNode(MenuTabViewHandler.MENU_DELETE_BUTTON_ID)
50             .setVisible(Boolean.FALSE);
51         // reload users
52         handleMenuReload(ctx);
53     }
54
55     /**
56      * Saves a menu
57      *
58      * @param event

```

# Übung 3

```

58      *           the {@link ActionEvent}
59      */
60  public void saveMenu(final ActionEvent event) {
61      final FormContext<MenuModel> ctx = (FormContext<MenuModel>) ((Node)
62          ↪ event.getSource()).getUserData();
63      ctx.getNode(MenuTabViewHandler.MENU_DELETE_BUTTON_ID)
64          .setVisible(Boolean.TRUE);
65      ctx.formHandler.validateForm(ctx);
66      ctx.formHandler.fillModel(ctx);
67
68      if (ctx.valid) {
69          final Menu menu;
70          if (ctx.model.getId() != null) {
71              menu = ctx.model.getEntity();
72              EntityCache.menuCache.add(menu);
73          } else {
74              menu = ctx.model.getEntity();
75              menu.setId(EntityCache.menuCache.size() + 1);
76              EntityCache.menuCache.add(menu);
77          }
78          ctx.model.prepare(menu);
79          handleMenuReload(ctx);
80      } else {
81          populateFormMessage("Formular ungültig !! Bitte Eingaben prüfen", ctx);
82          handleMenuReload(ctx);
83      }
84  }
85
86  /**
87   * Deletes an menu
88   *
89   * @param event
90   *           the {@link ActionEvent}
91   */
92  public void deleteMenu(final ActionEvent event) {
93      final FormContext<MenuModel> ctx = (FormContext<MenuModel>) ((Node)
94          ↪ event.getSource()).getUserData();
95
96      if (ctx.model.getId() != null) {
97          ctx.getNode(MenuTabViewHandler.MENU_DELETE_BUTTON_ID)
98              .setVisible(Boolean.TRUE);
99          final Menu menu = ctx.model.getEntity();
100          EntityCache.menuCache.remove(menu);
101          EntityCache.deleteForMenuId(ctx.model.getEntity()
102              .getId());
103          ctx.formHandler.resetForm(ctx);
104          ctx.model.reset();
105      }
106      handleMenuReload(ctx);
107  }
108
109  /**
110   * The handles the load of the menus. Resets the current ctx.model in the
111   * observed list and set this model as selected
112   *
113   * @param ctx
114   *           the form context
115   */
116  public void handleMenuReload(final FormContext<MenuModel> ctx) {
117      Objects.requireNonNull(ctx);
118
119      final ObservableList<MenuModel> list = (ObservableList<MenuModel>)
120          ↪ ctx.getObservable(MenuTabViewHandler.MENU_SELECTION_KEY);

```

## Übung 3

```

119     list.clear();
120     list.add(new MenuModel());
121     for (Menu menu : EntityCache.menuCache) {
122         final MenuModel model = new MenuModel();
123         model.prepare(menu);
124         list.add(model);
125     }
126
127     // need to replace observed instance
128     list.set(list.indexOf(ctx.model), ctx.model);
129
130     // need to select current context hold model
131     ((ChoiceBox<MenuModel>)
132         ↪ ctx.getNode(MenuTabViewHandler.MENU_SELECTION_KEY)).getSelectionModel()
133         .select(ctx.model);
134
135     /**
136      * Populates a message to the message box. <br>
137      * If message is null the actual set message will be cleared
138      *
139      * @param message
140      *         the message to populate
141      * @param ctx
142      *         the form context
143      */
144     private void populateFormMessage(final String message, final FormContext<MenuModel> ctx) {
145         final TextFlow flow = ((TextFlow) ctx.getNode(MenuTabViewHandler.MENU_FORM_MESSAGE));
146         flow.getChildren()
147             .clear();
148         flow.setPrefHeight(0);
149         if (!StringUtil.isEmpty(message)) {
150             flow.getChildren()
151                 .add(new Text(message));
152             flow.setPrefHeight(30);
153         }
154     }
155 }

```



## Übung 3

### 1.9.3 MenuEntryModel.java

Folgend ist das Menu-Eintrag Model angeführt welches für das Menu-Eintrag Formular verwendet wird.

Listing 31: MenuEntryModel.java

```

1 package at.fh.ooe.swe4.fx.campina.view.admin.menu.model;
2
3 import java.math.BigDecimal;
4 import java.util.Objects;
5
6 import javafx.collections.FXCollections;
7 import javafx.collections.ObservableList;
8 import javafx.util.StringConverter;
9 import at.fh.ooe.swe4.fx.campina.jpa.Menu;
10 import at.fh.ooe.swe4.fx.campina.jpa.MenuEntry;
11 import at.fh.ooe.swe4.fx.campina.view.annotation.FormField;
12 import at.fh.ooe.swe4.fx.campina.view.annotation.SelectFormField;
13 import at.fh.ooe.swe4.fx.campina.view.api.AbstractViewModel;
14 import at.fh.ooe.swe4.fx.campina.view.form.FormUtils.FormFieldType;
15
16 /**
17  * The view model which backs the {@link MenuEntry} entity
18  *
19  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
20  * @date Jun 5, 2015
21  */
22 public class MenuEntryModel extends AbstractViewModel<Integer, MenuEntry> {
23
24     public ObservableList<Menu> menus;
25
26     /**
27      * The converter for the {@link Menu} type
28      *
29      * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
30      * @date Jun 5, 2015
31      */
32     public static class MenuConverter extends StringConverter<Menu> {
33
34         @Override
35         public String toString(Menu object) {
36             return (object.getId() == null) ? "Bitte Wählen" : object.getLabel();
37         }
38
39         @Override
40         public Menu fromString(String string) {
41             throw new UnsupportedOperationException("From text not supported by this converter");
42         }
43     }
44
45     /**
46      * The converter for the {@link MenuEntryModel} type.
47      *
48      * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
49      * @date Jun 5, 2015
50      */
51     public static class MenuEntryModelConverter extends StringConverter<MenuEntryModel> {
52
53         @Override
54         public String toString(MenuEntryModel object) {
55             Objects.requireNonNull(object);
56             return (object.getId() == null) ? "Neuer Menu Eintrag" : object.getLabel();
57         }
58     }
59 }

```

# Übung 3

```

58     }
59
60     @Override
61     public MenuEntryModel fromString(String string) {
62         throw new UnsupportedOperationException("Form text not supported by this converter");
63     }
64
65 }
66
67 /**
68  *
69  */
70 public MenuEntryModel() {
71     super();
72 }
73
74 /**
75  * @param entity
76  */
77 public MenuEntryModel(MenuEntry entity) {
78     super(entity);
79 }
80
81 @Override
82 public void reset() {
83     final MenuEntry entry = new MenuEntry();
84     entry.setPrice(BigDecimal.ZERO);
85     entry.setMenu(new Menu());
86     entry.setOrdinal(0);
87     menus = FXCollections.observableArrayList();
88     menus.add(new Menu());
89     prepare(entry);
90
91 }
92
93 @FormField(id = "menu-entry-label",
94     label = "Beschreibung",
95     ordinal = 1,
96     required = true,
97     requiredMessage = "Bitte Beschreibung angeben",
98     type = FormFieldType.INPUT_TEXT)
99 public String getLabel() {
100     return getEntity().getLabel();
101 }
102
103 public void setLabel(String label) {
104     getEntity().setLabel(label);
105 }
106
107 @FormField(id = "menu-entry-price",
108     label = "Preis",
109     ordinal = 2,
110     required = true,
111     requiredMessage = "Bitte Preis angeben",
112     type = FormFieldType.DECIMAL)
113 public BigDecimal getPrice() {
114     return getEntity().getPrice();
115 }
116
117 public void setPrice(BigDecimal price) {
118     getEntity().setPrice(price);
119 }
120

```

# Übung 3

```

121  @FormField(id = "menu-entry-menu",
122           label = "Menu",
123           ordinal = 3,
124           required = true,
125           requiredMessage = "Bitte Menu wählen",
126           type = FormFieldType.SELECT,
127           valueClass = Menu.class)
128  public Menu getMenu() {
129      return getEntity().getMenu();
130  }
131
132  public void setMenu(Menu menu) {
133      getEntity().setMenu(menu);
134      if (menu != null) {
135          getEntity().setOrdinal(menu.getEntries()
136                                .size() + 1);
137      }
138  }
139
140  @SelectFormField(target = "menu",
141                  converter = MenuConverter.class)
142  public ObservableList<Menu> getMenus() {
143      return menus;
144  }
145  }

```

## Übung 3

### 1.9.4 MenuEntryEventControl.java

Folgend ist die Klasse angeführt, welche die Actions behandelt, wie Speichern, Löschen, usw.

Listing 32: MenuEntryEventControl.java

```

1 package at.fh.ooe.swe4.fx.campina.view.admin.menu.control;
2
3 import java.util.Objects;
4
5 import javafx.collections.ObservableList;
6 import javafx.event.ActionEvent;
7 import javafx.scene.Node;
8 import javafx.scene.control.ChoiceBox;
9 import javafx.scene.text.Text;
10 import javafx.scene.text.TextFlow;
11
12 import org.apache.commons.lang.StringUtils;
13
14 import at.fh.ooe.swe4.fx.campina.jpa.EntityCache;
15 import at.fh.ooe.swe4.fx.campina.jpa.Menu;
16 import at.fh.ooe.swe4.fx.campina.jpa.MenuEntry;
17 import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuEntryModel;
18 import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuModel;
19 import at.fh.ooe.swe4.fx.campina.view.admin.menu.part.MenuTabViewHandler;
20 import at.fh.ooe.swe4.fx.campina.view.api.FormContext;
21
22 /**
23  * The event handler for the {@link MenuEntry} entity.
24  *
25  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
26  * @date Jun 5, 2015
27  */
28 public class MenuEntryEventControl {
29
30     public MenuEntryEventControl() {
31         // TODO Auto-generated constructor stub
32     }
33
34     /**
35      * Handles the new action of the form.
36      *
37      * @param event
38      *         the {@link ActionEvent}
39      */
40     public void newAction(final ActionEvent event) {
41         final FormContext<MenuEntryModel> ctx = (FormContext<MenuEntryModel>) ((Node)
42             ↪ event.getSource()).getUserData();
43         // clear former set message
44         populateFormMessage(null, ctx);
45         // reset form
46         ctx.formHandler.resetForm(ctx);
47         // create new user model with new user entity
48         ctx.model.reset();
49         // hide buttons
50         ctx.getNode(MenuTabViewHandler.MENU_ENTRY_DELETE_BUTTON_ID)
51             .setVisible(Boolean.FALSE);
52         // reload users
53         handleMenuEntryReload(ctx);
54     }
55
56     /**
57      * Saves a {@link MenuEntry}
58      */

```

## Übung 3

```

58  * @param event
59  *         the {@link ActionEvent}
60  */
61  public void saveMenuEntry(final ActionEvent event) {
62      final FormContext<MenuEntryModel> ctx = (FormContext<MenuEntryModel>) ((Node)
        ↳ event.getSource()).getUserData();
63      ctx.getNode(MenuTabViewHandler.MENU_ENTRY_DELETE_BUTTON_ID)
64          .setVisible(Boolean.TRUE);
65      ctx.formHandler.validateForm(ctx);
66      ctx.formHandler.fillModel(ctx);
67
68      if (ctx.valid) {
69          final MenuEntry entry;
70          if (ctx.model.getId() != null) {
71              entry = ctx.model.getEntity();
72              EntityCache.menuEntryCache.add(entry);
73              EntityCache.byMenuId(entry.getMenu()
74                  .getId())
75                  .getEntries()
76                  .add(entry);
77          } else {
78              entry = ctx.model.getEntity();
79              entry.setId(EntityCache.menuCache.size() + 1);
80              EntityCache.menuEntryCache.add(entry);
81          }
82          ctx.model.prepare(entry);
83          handleMenuEntryReload(ctx);
84      } else {
85          populateFormMessage("Formular ungültig !! Bitte Eingaben prüfen", ctx);
86          handleMenuEntryReload(ctx);
87      }
88  }
89
90  /**
91   * Deletes a {@link MenuEntry}
92   *
93   * @param event
94   *         the {@link ActionEvent}
95   */
96  public void deleteMenuEntry(final ActionEvent event) {
97      final FormContext<MenuEntryModel> ctx = (FormContext<MenuEntryModel>) ((Node)
        ↳ event.getSource()).getUserData();
98
99      if (ctx.model.getId() != null) {
100          ctx.getNode(MenuTabViewHandler.MENU_ENTRY_DELETE_BUTTON_ID)
101              .setVisible(Boolean.TRUE);
102          final MenuEntry entry = ctx.model.getEntity();
103          EntityCache.menuEntryCache.remove(entry);
104          EntityCache.byMenuId(ctx.model.getEntity()
105              .getMenu()
106              .getId())
107              .getEntries()
108              .remove(entry);
109          ctx.formHandler.resetForm(ctx);
110          ctx.model.reset();
111      }
112      handleMenuEntryReload(ctx);
113  }
114
115  /**
116   * Handles the {@link MenuEntry} reload
117   */
118  public void handleMenuEntryReload(final FormContext<MenuEntryModel> ctx) {

```

# Übung 3

```

119     Objects.requireNonNull(ctx);
120
121     final ObservableList<MenuEntryModel> list = (ObservableList<MenuEntryModel>)
122         ↪ ctx.getObservable(MenuTabViewHandler.MENU_ENTRY_SELECTION_KEY);
123     list.clear();
124     list.add(new MenuEntryModel());
125     for (MenuEntry entry : EntityCache.menuEntryCache) {
126         final MenuEntryModel model = new MenuEntryModel();
127         model.prepare(entry);
128         list.add(model);
129     }
130
131     // need to replace observed instance
132     list.set(list.indexOf(ctx.model), ctx.model);
133
134     // need to select current context hold model
135     ((ChoiceBox<MenuEntryModel>)
136         ↪ ctx.getNode(MenuTabViewHandler.MENU_ENTRY_SELECTION_KEY)).getSelectionModel()
137         .select(ctx.model);
138 }
139
140 /**
141  * Handles the {@link Menu} reload.
142  *
143  * @param ctx
144  *         the form context
145  */
146 public void handleMenuLoad(final FormContext<MenuEntryModel> ctx) {
147     Objects.requireNonNull(ctx);
148
149     ctx.model.getMenus()
150         .clear();
151     ctx.model.getMenus()
152         .add(new Menu());
153     for (Menu menu : EntityCache.menuCache) {
154         ctx.model.getMenus()
155             .add(menu);
156     }
157
158     ctx.model.getMenus()
159         .set(ctx.model.getMenus()
160             .indexOf(ctx.model.getMenu()), ctx.model.getMenu());
161 }
162
163 /**
164  * Populates a message to the message box. <br>
165  * If message is null the actual set message will be cleared
166  *
167  * @param message
168  *         the message to populate
169  * @param ctx
170  *         the form context
171  */
172 private void populateFormMessage(final String message, final FormContext<MenuEntryModel> ctx) {
173     final TextFlow flow = ((TextFlow) ctx.getNode(MenuTabViewHandler.MENU_ENTRY_FORM_MESSAGE));
174     flow.getChildren()
175         .clear();
176     flow.setPrefHeight(0);
177     if (!StringUtils.isEmpty(message)) {
178         flow.getChildren()
179             .add(new Text(message));
180         flow.setPrefHeight(30);
181     }
182 }

```

## Übung 3

---

```
180 |   }  
181 | }
```

## Übung 3

### 1.9.5 MenuTabViewHandler.java

Folgend ist der ViewHandddler der MenuTab angeführt. Sie stellt einen View Teil der Scene dar und implementiert ViewHandler<T> Interface.

Listing 33: MenuTabViewHandler.java

```

1  package at.fh.ooe.swe4.fx.campina.view.admin.menu.part;
2
3  import java.util.ArrayList;
4  import java.util.Arrays;
5  import java.util.List;
6  import java.util.Objects;
7
8  import javafx.beans.value.ChangeListener;
9  import javafx.beans.value.ObservableValue;
10 import javafx.collections.FXCollections;
11 import javafx.collections.ObservableList;
12 import javafx.scene.Scene;
13 import javafx.scene.control.Button;
14 import javafx.scene.control.ChoiceBox;
15 import javafx.scene.control.RadioButton;
16 import javafx.scene.control.Tab;
17 import javafx.scene.control.Toggle;
18 import javafx.scene.control.ToggleGroup;
19 import javafx.scene.layout.GridPane;
20 import javafx.scene.text.TextFlow;
21 import javafx.util.StringConverter;
22 import at.fh.ooe.swe4.fx.campina.jpa.Menu;
23 import at.fh.ooe.swe4.fx.campina.view.admin.menu.control.MenuEntryEventControl;
24 import at.fh.ooe.swe4.fx.campina.view.admin.menu.control.MenuEventControl;
25 import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuEntryModel;
26 import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuEntryModel.MenuEntryModelConverter;
27 import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuModel;
28 import at.fh.ooe.swe4.fx.campina.view.admin.menu.model.MenuModel.MenuModelConverter;
29 import at.fh.ooe.swe4.fx.campina.view.api.FormContext;
30 import at.fh.ooe.swe4.fx.campina.view.api.ViewHandler;
31 import at.fh.ooe.swe4.fx.campina.view.form.FormHandler;
32
33 /**
34  * The view handler for the menu and menu entry view.
35  *
36  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
37  * @date Jun 5, 2015
38  */
39 public class MenuTabViewHandler implements ViewHandler<Tab> {
40
41     /**
42      * The form handler for the menu entry form
43      */
44     private final FormHandler<MenuEntryModel> menuEntryFormHandler;
45     /**
46      * The menu entry controller
47      */
48     private final MenuEntryEventControl menuEntryControl;
49     /**
50      * The form context for the menu entry form
51      */
52     final FormContext<MenuEntryModel> menuEntryCtx;
53
54     /**
55      * The form handler for the menu form
56      */

```



# Übung 3

```

57 private final FormHandler<MenuModel>    menuFormHandler;
58 /**
59  * The menu controller
60  */
61 private final MenuEventControl          menuControl;
62 /**
63  * The form context for the menu entry form
64  */
65 final FormContext<MenuModel>            menuCtx;
66
67 // #####
68 // Ids of nodes and form backed model class
69 // #####
70 public static final String                MENU_ENTRY_FORM_MESSAGE    = "tab-menu-entry-form-message";
71 public static final String                MENU_FORM_MESSAGE          = "tab-menu-form-message";
72 public static final String                EDIT_OPTION_ID              = "tab-menu-edit-option";
73 public static final String                MENU_SELECTION_KEY          = "tab-menu-menu-selection-data";
74 public static final String                MENU_ENTRY_SELECTION_KEY    =
75     ↳ "tab-menu-menu-entry-selection-data";
76 public static final String                MENU_SAVE_BUTTON_ID         = "tab-menu-menu-form-save-button";
77 public static final String                MENU_DELETE_BUTTON_ID       =
78     ↳ "tab-menu-menu-form-delete-button";
79 public static final String                MENU_ENTRY_SAVE_BUTTON_ID   =
80     ↳ "tab-menu-menu-entry-form-save-button";
81 public static final String                MENU_ENTRY_DELETE_BUTTON_ID =
82     ↳ "tab-menu-menu-entry-form-delete-button";
83 public static final String                MENU_NEW_BUTTON_ID          = "tab-menu-menu-form-new-button";
84 public static final String                MENU_ENTRY_NEW_BUTTON_ID    =
85     ↳ "tab-menu-menu-entry-form-new-button";
86
87 /**
88  * Enumeration which specifies the edit type. Provides an label too.
89  *
90  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
91  * @date Jun 5, 2015
92  */
93 public static enum EditMode {
94     MENU("Menus"),
95     MENU_ENTRY("Menu Einträge");
96
97     public final String label;
98
99     /**
100    * @param label
101    */
102     private EditMode(final String label) {
103         this.label = label;
104     }
105 }
106
107 /**
108  * The converter for the {@link EditMode} type.
109  *
110  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
111  * @date Jun 5, 2015
112  */
113 public static class EditModeConverter extends StringConverter<EditMode> {
114
115     @Override
116     public String toString(EditMode object) {
117         return object.label;
118     }
119 }

```

# Übung 3

```

115     @Override
116     public EditMode fromString(String string) {
117         throw new UnsupportedOperationException("From string not supported by this converter");
118     }
119
120 }
121
122 /**
123  * The change listener for the {@link Toggle} type.
124  *
125  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
126  * @date Jun 5, 2015
127  */
128 public static class ChangeModeListener implements ChangeListener<Toggle> {
129
130     private final GridPane parent;
131     private final GridPane menuGroup;
132     private final GridPane menuEntryGroup;
133     private final MenuEventControl menuControl;
134     private final MenuEntryEventControl menuEntryControl;
135
136     /**
137      * @param parent
138      * @param menuGroup
139      * @param menuEntryGroup
140      * @param menuControl
141      * @param menuEntryControl
142      */
143     public ChangeModeListener(GridPane parent, GridPane menuGroup, GridPane menuEntryGroup,
144         ↪ MenuEventControl menuControl,
145         MenuEntryEventControl menuEntryControl) {
146         super();
147         Objects.requireNonNull(parent);
148         Objects.requireNonNull(menuGroup);
149         Objects.requireNonNull(menuEntryGroup);
150         Objects.requireNonNull(menuControl);
151         Objects.requireNonNull(menuEntryControl);
152
153         this.parent = parent;
154         this.menuGroup = menuGroup;
155         this.menuEntryGroup = menuEntryGroup;
156         this.menuControl = menuControl;
157         this.menuEntryControl = menuEntryControl;
158     }
159
160     @Override
161     public void changed(ObservableValue<? extends Toggle> observable, Toggle oldValue, Toggle
162         ↪ newValue) {
163         if (observable.getValue() == null) {
164             return;
165         }
166
167         parent.getChildren()
168             .remove(menuEntryGroup);
169         parent.getChildren()
170             .remove(menuGroup);
171
172         switch ((EditMode) observable.getValue()
173             .getUserData()) {
174             case MENU:
175                 parent.add(menuGroup, 0, 1);
176                 final FormContext<MenuModel> menuCtx = (FormContext<MenuModel>) menuGroup.getUserData();
177                 menuCtx.model.reset();

```

# Übung 3

```

176     menuControl.handleMenuReload(menuCtx);
177     ((ChoiceBox<MenuModel>) menuCtx.getNode(MENU_SELECTION_KEY)).getSelectionModel()
178         .select(menuCtx.model);
179     menuCtx.getNode(MENU_DELETE_BUTTON_ID)
180         .setVisible(Boolean.FALSE);
181     menuCtx.formHandler.fillForm(menuCtx);
182     break;
183 case MENU_ENTRY:
184     parent.add(menuEntryGroup, 0, 1);
185     final FormContext<MenuEntryModel> menuEntryCtx = (FormContext<MenuEntryModel>)
186         ↳ menuEntryGroup.getUserData();
187     menuEntryCtx.model.reset();
188     menuEntryControl.handleMenuEntryReload(menuEntryCtx);
189     ((ChoiceBox<MenuEntryModel>)
190         ↳ menuEntryCtx.getNode(MENU_ENTRY_SELECTION_KEY)).getSelectionModel()
191         .select(menuEntryCtx.model);
192     menuEntryCtx.getNode(MENU_ENTRY_DELETE_BUTTON_ID)
193         .setVisible(Boolean.FALSE);
194     menuEntryControl.handleMenuLoad(menuEntryCtx);
195     menuEntryCtx.formHandler.fillForm(menuEntryCtx);
196     break;
197 default:
198     throw new IllegalArgumentException("'EditMode#" + ((EditMode) observable.getValue()
199         ↳ .getUserData()).name() + "' node supported");
200 }
201 }
202 }
203 /**
204  * The change listener for the {@link MenuModel} type.
205  *
206  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
207  * @date Jun 5, 2015
208  */
209 public static class MenuChangeListener implements ChangeListener<MenuModel> {
210
211     private final FormContext<MenuModel> ctx;
212     private final MenuEventControl control;
213
214     /**
215      * @param ctx
216      * @param control
217      */
218     public MenuChangeListener(FormContext<MenuModel> ctx, MenuEventControl control) {
219         super();
220         Objects.requireNonNull(ctx);
221         Objects.requireNonNull(control);
222
223         this.ctx = ctx;
224         this.control = control;
225     }
226
227     @Override
228     public void changed(ObservableValue<? extends MenuModel> observable, MenuModel oldValue,
229         ↳ MenuModel newValue) {
230         if (observable.getValue() == null) {
231             return;
232         }
233
234         ctx.model = observable.getValue();
235         ctx.model.prepare(observable.getValue()

```

## Übung 3

```

236         ctx.getNode(MENU_DELETE_BUTTON_ID)
237             .setVisible(Boolean.TRUE);
238     } else {
239         ctx.getNode(MENU_DELETE_BUTTON_ID)
240             .setVisible(Boolean.FALSE);
241     }
242     ctx.formHandler.fillForm(ctx);
243 }
244 }
245
246 /**
247  * The change listener for the {@link MenuEntryModel} type
248  *
249  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
250  * @date Jun 5, 2015
251  */
252 public static class MenuEntryChangeListener implements ChangeListener<MenuEntryModel> {
253
254     private final FormContext<MenuEntryModel> ctx;
255     private final MenuEntryEventControl control;
256
257     /**
258      * @param ctx
259      * @param control
260      */
261     public MenuEntryChangeListener(FormContext<MenuEntryModel> ctx, MenuEntryEventControl control)
262         ↪ {
263         super();
264         Objects.requireNonNull(ctx);
265         Objects.requireNonNull(control);
266
267         this.ctx = ctx;
268         this.control = control;
269     }
270
271     @Override
272     public void changed(ObservableValue<? extends MenuEntryModel> observable, MenuEntryModel
273         ↪ oldValue, MenuEntryModel newValue) {
274         if (observable.getValue() == null) {
275             return;
276         }
277
278         ctx.model = observable.getValue();
279         ctx.model.prepare(observable.getValue()
280             ↪ .getEntity());
281         control.handleMenuLoad(ctx);
282         if (ctx.model.getId() == null) {
283             ctx.getNode(MENU_ENTRY_DELETE_BUTTON_ID)
284                 .setVisible(Boolean.FALSE);
285         } else {
286             ctx.getNode(MENU_ENTRY_DELETE_BUTTON_ID)
287                 .setVisible(Boolean.TRUE);
288         }
289         ctx.formHandler.fillForm(ctx);
290     }
291 }
292
293 /**
294  * @param scene
295  */
296 public MenuTabViewHandler(final Scene scene) {
297     Objects.requireNonNull(scene);
298 }

```

## Übung 3

```

297     this.menuEntryControl = new MenuEntryEventControl();
298     this.menuEntryFormHandler = new FormHandler<MenuEntryModel>();
299     this.menuEntryFormHandler.init();
300     this.menuEntryCtx = new FormContext<MenuEntryModel>("menu-entry-form", menuEntryFormHandler,
        ↪     new MenuEntryModel(), scene);
301
302     this.menuControl = new MenuEventControl();
303     this.menuFormHandler = new FormHandler<MenuModel>();
304     this.menuFormHandler.init();
305     this.menuCtx = new FormContext<MenuModel>("menu-form", menuFormHandler, new MenuModel(),
        ↪     scene);
306 }
307
308 @Override
309 public String getId() {
310     return "tab-menu";
311 }
312
313 @Override
314 public Tab createNode() {
315     // edit mode toggle
316     final ToggleGroup group = new ToggleGroup();
317     final List<RadioButton> buttons = createEditModeToggle(group);
318     final GridPane editOptionsGrid = new GridPane();
319     editOptionsGrid.setId(getId() + "-edit-options-grid");
320     editOptionsGrid.setHgap(10);
321     editOptionsGrid.setVgap(10);
322     for (int i = 0; i < buttons.size(); i++) {
323         editOptionsGrid.add(buttons.get(i), i, 0);
324     }
325     group.getToggles()
326         .get(0)
327         .setSelected(Boolean.TRUE);
328
329     // Form message text
330     final TextFlow menuEntryFlow = new TextFlow();
331     menuEntryFlow.setId(MENU_ENTRY_FORM_MESSAGE);
332     menuEntryFlow.setStyle("-fx-font-size: 20pt");
333     menuEntryCtx.putNode(MENU_ENTRY_FORM_MESSAGE, menuEntryFlow);
334
335     final TextFlow menuFlow = new TextFlow();
336     menuEntryFlow.setId(MENU_ENTRY_FORM_MESSAGE);
337     menuEntryFlow.setStyle("-fx-font-size: 20pt");
338     menuCtx.putNode(MENU_FORM_MESSAGE, menuFlow);
339
340     // menu form grid
341     final GridPane menuGrid = new GridPane();
342     menuGrid.setId(getId() + "-menu-grid");
343     menuGrid.add(menuFlow, 0, 0);
344     menuGrid.add(createMenuChoiceBox(menuCtx), 0, 1);
345     menuGrid.add(menuCtx.formHandler.generateFormGrid(menuCtx), 0, 2);
346     menuGrid.add(createMenuButtonGroup(menuCtx), 0, 3);
347     menuGrid.setUserData(menuCtx);
348     menuGrid.setHgap(10);
349     menuGrid.setVgap(10);
350
351     final GridPane menuEntryGrid = new GridPane();
352     menuEntryGrid.setId(getId() + "-menu-entry-grid");
353     menuEntryGrid.add(menuEntryFlow, 0, 0);
354     menuEntryGrid.add(createMenuEntryChoiceBox(menuEntryCtx), 0, 1);
355     menuEntryGrid.add(menuEntryCtx.formHandler.generateFormGrid(menuEntryCtx), 0, 2);
356     menuEntryGrid.add(createMenuEntryButtonGroup(menuEntryCtx), 0, 3);
357     menuEntryGrid.setUserData(menuEntryCtx);

```

# Übung 3

```

358 menuEntryGrid.setHgap(10);
359 menuEntryGrid.setVgap(10);
360
361 // content grid
362 final GridPane gridPane = new GridPane();
363 gridPane.setId(getId() + "-content");
364 gridPane.add(editOptionsGrid, 0, 0);
365 gridPane.add(menuGrid, 0, 1);
366 gridPane.setHgap(10);
367 gridPane.setVgap(10);
368
369 // TODO: Set change listener on edit mode choice;
370 group.selectedToggleProperty()
371     .addListener(new ChangeModeListener(gridPane, menuGrid, menuEntryGrid, menuControl,
372         ↪ menuEntryControl));
373
374 final Tab menuTab = new Tab();
375 menuTab.setId("tab-menu");
376 menuTab.setText("Menus");
377 menuTab.setClosable(Boolean.FALSE);
378
379 menuTab.setContent(gridPane);
380
381 return menuTab;
382 }
383
384 /**
385  * Creates the edit toggle for the edit mode.
386  *
387  * @param group
388  *     the toggle group which holds the toggles.
389  * @return the list of toggles
390  */
391 private List<RadioButton> createEditModeToggle(final ToggleGroup group) {
392     Objects.requireNonNull(group);
393
394     final List<RadioButton> buttons = new ArrayList<>();
395
396     for (EditMode editMode : EditMode.values()) {
397         final RadioButton button = new RadioButton();
398         button.setId(getId() + "-edit-mode-" + editMode.name()
399             ↪ .toLowerCase());
400         button.setText(editMode.label);
401         button.setToggleGroup(group);
402         button.setUserData(editMode);
403         buttons.add(button);
404     }
405
406     return buttons;
407 }
408
409 /**
410  * Creates a menu choice box.
411  *
412  * @param ctx
413  *     the form context
414  * @return the choice box
415  */
416 private ChoiceBox<MenuModel> createMenuChoiceBox(final FormContext<MenuModel> ctx) {
417     Objects.requireNonNull(ctx);
418
419     final ObservableList<MenuModel> menus = FXCollections.observableArrayList(Arrays.asList(new
420         ↪ MenuModel()));

```

# Übung 3

```

419     final ChoiceBox<MenuModel> menuChoice = new ChoiceBox<>(menus);
420     menuChoice.setUserData(ctx);
421     menuChoice.getSelectionModel()
422         .selectedItemProperty()
423         .addListener(new MenuChangeListener(ctx, menuControl));
424     menuChoice.setConverter(new MenuModelConverter());
425     menuChoice.setPrefWidth(400);
426
427
428     ctx.putObservable(MENU_SELECTION_KEY, menus);
429     ctx.putNode(MENU_SELECTION_KEY, menuChoice);
430
431     return menuChoice;
432 }
433
434 /**
435  * Creates a menu entry choice box
436  *
437  * @param ctx the
438  *           form context
439  * @return the choice box
440  */
441 private ChoiceBox<MenuEntryModel> createMenuEntryChoiceBox(final FormContext<MenuEntryModel>
↳ ctx) {
442     Objects.requireNonNull(ctx);
443
444     final ObservableList<MenuEntryModel> entries =
↳ FXCollections.observableArrayList(Arrays.asList(new MenuEntryModel()));
445
446     final ChoiceBox<MenuEntryModel> menuEntryChoice = new ChoiceBox<>(entries);
447     menuEntryChoice.setUserData(ctx);
448     menuEntryChoice.getSelectionModel()
449         .selectedItemProperty()
450         .addListener(new MenuEntryChangeListener(ctx, menuEntryControl));
451     menuEntryChoice.setConverter(new MenuEntryModelConverter());
452     menuEntryChoice.setPrefWidth(400);
453
454     ctx.putObservable(MENU_ENTRY_SELECTION_KEY, entries);
455     ctx.putNode(MENU_ENTRY_SELECTION_KEY, menuEntryChoice);
456
457     return menuEntryChoice;
458 }
459
460 /**
461  * Creates the menu button group
462  *
463  * @param ctx
464  *           the form context
465  * @return the creates button group
466  */
467 private GridPane createMenuButtonGroup(final FormContext<MenuModel> ctx) {
468     Objects.requireNonNull(ctx);
469
470     final GridPane gridPane = new GridPane();
471     gridPane.setId(getId() + "-button-grid");
472     gridPane.setHgap(10);
473     gridPane.setVgap(10);
474
475     final Button newButton = new Button();
476     newButton.setText("Neues Menu");
477     newButton.setId(MENU_NEW_BUTTON_ID);
478     newButton.setUserData(ctx);
479     newButton.setOnAction(menuControl::newAction);

```

## Übung 3

```

480
481     final Button saveButton = new Button();
482     saveButton.setText("Speichern");
483     saveButton.setId(MENU_SAVE_BUTTON_ID);
484     saveButton.setUserData(ctx);
485     saveButton.setOnAction(menuControl::saveMenu);
486
487     final Button deleteButton = new Button();
488     deleteButton.setId(MENU_DELETE_BUTTON_ID);
489     deleteButton.setText("Löschen");
490     deleteButton.setUserData(ctx);
491     deleteButton.setOnAction(menuControl::deleteMenu);
492
493     gridPane.add(newButton, 0, 0);
494     gridPane.add(saveButton, 1, 0);
495     gridPane.add(deleteButton, 2, 0);
496
497     // register in context
498     ctx.putNode(MENU_NEW_BUTTON_ID, newButton);
499     ctx.putNode(MENU_SAVE_BUTTON_ID, saveButton);
500     ctx.putNode(MENU_DELETE_BUTTON_ID, deleteButton);
501
502     return gridPane;
503 }
504
505 /**
506  * Creates the menu entry button group
507  *
508  * @param ctx
509  *         the from context
510  * @return the created button group
511  */
512 private GridPane createMenuEntryButtonGroup(final FormContext<MenuEntryModel> ctx) {
513     Objects.requireNonNull(ctx);
514
515     final GridPane gridPane = new GridPane();
516     gridPane.setId(getId() + "-button-grid");
517     gridPane.setHgap(10);
518     gridPane.setVgap(10);
519
520     final Button newButton = new Button();
521     newButton.setText("Neuer Menu Eintrag");
522     newButton.setId(MENU_ENTRY_NEW_BUTTON_ID);
523     newButton.setUserData(ctx);
524     newButton.setOnAction(menuEntryControl::newAction);
525
526     final Button saveButton = new Button();
527     saveButton.setText("Speichern");
528     saveButton.setId(MENU_ENTRY_SAVE_BUTTON_ID);
529     saveButton.setUserData(ctx);
530     saveButton.setOnAction(menuEntryControl::saveMenuEntry);
531
532     final Button deleteButton = new Button();
533     deleteButton.setId(MENU_ENTRY_DELETE_BUTTON_ID);
534     deleteButton.setText("Löschen");
535     deleteButton.setUserData(ctx);
536     deleteButton.setOnAction(menuEntryControl::deleteMenuEntry);
537
538     gridPane.add(newButton, 0, 0);
539     gridPane.add(saveButton, 1, 0);
540     gridPane.add(deleteButton, 2, 0);
541
542     // register in context

```



## Übung 3

```
543     ctx.putNode(MENU_ENTRY_NEW_BUTTON_ID, newButton);
544     ctx.putNode(MENU_ENTRY_SAVE_BUTTON_ID, saveButton);
545     ctx.putNode(MENU_ENTRY_DELETE_BUTTON_ID, deleteButton);
546
547     return gridPane;
548 }
549
550 @Override
551 public void initHandler() {
552     menuControl.handleMenuReload(menuCtx);
553     menuCtx.formHandler.fillForm(menuCtx);
554     ((ChoiceBox<MenuModel>) menuCtx.getNode(MENU_SELECTION_KEY)).getSelectionModel()
555         .select(menuCtx.model);
556     menuCtx.getNode(MENU_DELETE_BUTTON_ID)
557         .setVisible(Boolean.FALSE);
558 }
559
560 }
```

## Übung 3

### 1.10 Source-Code(Implementation Order View)

Folgend ist der Source der Order View angeführt.

#### 1.10.1 OrderModel.java

Folgende Klasse repräsentiert das View Model für Order Entitäten.

Listing 34: OrderModel.java

```

1  package at.fh.ooe.swe4.fx.campina.view.admin.order.model;
2
3  import org.apache.commons.lang.time.DateFormatUtils;
4
5  import at.fh.ooe.swe4.fx.campina.jpa.Order;
6  import at.fh.ooe.swe4.fx.campina.jpa.User;
7  import at.fh.ooe.swe4.fx.campina.view.api.AbstractViewModel;
8
9  /**
10   * The view model backing the {@link Order} entity.
11   *
12   * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
13   * @date Jun 5, 2015
14   */
15  public class OrderModel extends AbstractViewModel<Integer, Order> {
16
17      /**
18       *
19       */
20      public OrderModel() {
21          super();
22      }
23
24      /**
25       * @param entity
26       */
27      public OrderModel(Order entity) {
28          super(entity);
29      }
30
31      @Override
32      public void reset() {
33          prepare(new Order());
34      }
35
36      // Resolved by TableColumn
37      public String getOrderDate() {
38          return DateFormatUtils.ISO_DATETIME_FORMAT.format(getEntity().getOrderDate());
39      }
40
41      public String getUsername() {
42          final User user = getEntity().getUser();
43          return new StringBuilder(user.getLastName()).append(", ")
44              .append(user.getFirstName())
45              .toString();
46      }
47
48      public String getMenuLabel() {
49          return getEntity().getMenuEntry()
50              .getMenu()
51              .getLabel();
52      }
53

```

Übung 3

---

```
54 | public String getMenuEntryLabel() {  
55 |     return getEntity().getMenuEntry()  
56 |         .getLabel();  
57 | }  
58 | }
```

## Übung 3

## 1.10.2 OrderEventHandler.java

Folgend ist die Klasse angeführt, welche die Aktionen behandelt, wie Speichern, Löschen, usw.

Listing 35: OrderEventHandler.java

```
1 package at.fh.ooe.swe4.fx.campina.view.admin.order.control;
2
3 import java.util.Objects;
4
5 import javafx.collections.ObservableList;
6 import at.fh.ooe.swe4.fx.campina.jpa.EntityCache;
7 import at.fh.ooe.swe4.fx.campina.jpa.Order;
8 import at.fh.ooe.swe4.fx.campina.view.admin.order.model.OrderModel;
9
10 public class OrderEventHandler {
11
12     public OrderEventHandler() {
13         // TODO Auto-generated constructor stub
14     }
15
16     public void reloadOrders(ObservableList<OrderModel> list) {
17         Objects.requireNonNull(list);
18
19         list.clear();
20         for (Order order : EntityCache.orderCache) {
21             list.add(new OrderModel(order));
22         }
23     }
24 }
```

## Übung 3

### 1.11 Source-Code(Implementation Main Scene)

Folgend ist der Source der Main Scene angeführt.

#### 1.11.1 MainSceneViewHandler.java

Diese Klasse baut sich die Scene auf und initialisiert sie. In Zukunft könnte diese Klasse auch das Wechseln zwischen View Teilen verwalten (Tab Visibility, usw.).

Dies könnte über Menu-Einträge erfolgen, die hier erstellt und verwaltet werden.

Listing 36: MainSceneViewHandler.java

```

1 package at.fh.ooe.swe4.fx.campina.view.scene;
2
3 import java.util.Arrays;
4 import java.util.Collections;
5 import java.util.List;
6 import java.util.Objects;
7 import java.util.stream.Collectors;
8
9 import javafx.scene.Scene;
10 import javafx.scene.control.Tab;
11 import javafx.scene.control.TabPane;
12 import javafx.scene.layout.Pane;
13 import javafx.scene.layout.VBox;
14 import at.fh.ooe.swe4.fx.campina.component.builder.impl.MenuBarBuilder;
15 import at.fh.ooe.swe4.fx.campina.component.builder.impl.MenuBuilder;
16 import at.fh.ooe.swe4.fx.campina.component.builder.impl.MenuItemBuilder;
17 import at.fh.ooe.swe4.fx.campina.view.admin.login.part.LoginTabViewHandler;
18 import at.fh.ooe.swe4.fx.campina.view.admin.menu.part.MenuTabViewHandler;
19 import at.fh.ooe.swe4.fx.campina.view.admin.order.part.OrderTabViewHandler;
20 import at.fh.ooe.swe4.fx.campina.view.admin.user.part.UserTabviewHandler;
21 import at.fh.ooe.swe4.fx.campina.view.api.EventHandlerFactory;
22 import at.fh.ooe.swe4.fx.campina.view.api.ViewHandler;
23
24 /**
25  * The view ahdnler for the main scene.
26  *
27  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
28  * @date Jun 4, 2015
29  */
30 public class MainSceneViewHandler implements ViewHandler<Scene> {
31
32     private static final long serialVersionUID = -7630460091326559133L;
33
34     private UserTabviewHandler userTab;
35     private LoginTabViewHandler loginTab;
36     private MenuTabViewHandler menuTab;
37     private OrderTabViewHandler orderTab;
38
39     /**
40      * Enumeration which specifies the menus placed in the menu bar.
41      *
42      * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
43      * @date May 29, 2015
44      */
45     public static enum MenuDefinition {
46         FILE(1, "menu-file", "Dateien");
47
48         public final int ordinal;
49         public final String id;
50         public final String label;

```

# Übung 3

```

51     public static final EventHandlerFactory eventHandlerFactory = null;
52
53     /**
54      * @param ordinal
55      * @param id
56      * @param label
57      */
58     private MenuDefinition(int ordinal, String id, String label) {
59         this.ordinal = ordinal;
60         this.id = id;
61         this.label = label;
62     }
63 }
64
65 /**
66  * This enumeration specifies the menu items for the specified menus.
67  *
68  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
69  * @date May 29, 2015
70  */
71 public static enum MenuItemDefinition {
72     /**-- User menu items --*/
73     CLOSE(3, "item-close", "Schließen", MenuDefinition.FILE);
74
75     public final String id;
76     public final String label;
77     public final Integer ordinal;
78     public final MenuDefinition menu;
79     public static final EventHandlerFactory<String> eventHandlerFactory = new
80         ↳ MainSceneEventHandlerFactory();
81
82     /**
83      * @param ordinal
84      * @param id
85      * @param label
86      * @param menu
87      */
88     private MenuItemDefinition(int ordinal, String id, String label, MenuDefinition menu) {
89         this.id = id;
90         this.label = label;
91         this.ordinal = ordinal;
92         this.menu = menu;
93     }
94
95     /**
96      * Filters the provided enumerations for the given menu.
97      *
98      * @param menu
99      *         the menu to get its specified items
100     * @return the menu related items, an empty list otherwise
101     */
102     public static final List<MenuItemDefinition> filterForMenu(final MenuDefinition menu) {
103         final List<MenuItemDefinition> items;
104         if (menu != null) {
105             items = Arrays.asList(MenuItemDefinition.values())
106                 .stream()
107                 .filter((el) -> menu.equals(el.menu))
108                 .collect(Collectors.toList());
109             Collections.sort(items, (o1, o2) -> o1.ordinal.compareTo(o2.ordinal));
110         } else {
111             items = Collections.EMPTY_LIST;
112         }
113         return items;

```

# Übung 3

```

113     }
114 }
115
116 /**
117  * Creates a main scene definition instance.<br>
118  * The main components ids are set here.
119  */
120 public MainSceneViewHandler() {
121 }
122
123 @Override
124 public Scene createNode() {
125     final Pane rootPane = new VBox();
126     final Pane menuBarBox = new VBox();
127     final TabPane tabPane = new TabPane();
128
129     rootPane.setId("root-pane");
130     menuBarBox.setId("menu-bar-box");
131     tabPane.setId("tab-pane");
132
133     final Scene scene = new Scene(rootPane);
134     this.userTab = new UserTabviewHandler(scene);
135     this.loginTab = new LoginTabViewHandler(scene);
136     this.menuTab = new MenuTabViewHandler(scene);
137     this.orderTab = new OrderTabViewHandler(scene);
138
139     prepareMenuBox(menuBarBox);
140     prepareTabs(tabPane);
141
142     rootPane.getChildren()
143         .add(menuBarBox);
144     rootPane.getChildren()
145         .add(tabPane);
146
147     return scene;
148 }
149
150 @Override
151 public void initHandler() {
152     // this.loginTab.init();
153     this.userTab.initHandler();
154     this.menuTab.initHandler();
155     this.orderTab.initHandler();
156 }
157
158 /**
159  * Prepares the menu box by adding all menus and its items.
160  *
161  * @param menuBox
162  *         the menu box to provide
163  */
164 public void prepareMenuBox(final Pane menuBox) {
165     Objects.requireNonNull(menuBox, "Cannot prepare null menu box");
166
167     MenuBarBuilder mbb = new MenuBarBuilder();
168     MenuBuilder mb = new MenuBuilder();
169     MenuItemBuilder mib = new MenuItemBuilder();
170
171     mbb.start();
172     for (MenuDefinition menu : MenuDefinition.values()) {
173         mb.start();
174         if (MenuDefinition.eventHandlerFactory != null) {
175             mb.registerEventHandlers(MenuItemDefinition.eventHandlerFactory.registerEventHandler(menu.id));

```

# Übung 3

```

176     }
177     for (MenuItemDefinition item : MenuItemDefinition.filterForMenu(menu)) {
178         mib.start();
179         if (MenuItemDefinition.eventHandlerFactory != null) {
180             mib.registerEventHandlers(MenuItemDefinition.eventHandlerFactory.registerEventHandler(item.id));
181         }
182         mb.addItem(mib.build(item.id, item.label));
183         mib.end();
184     }
185     mbb.addMenu(mb.build(menu.id, menu.label));
186     mb.end();
187 }
188 menuBox.getChildren()
189     .add(mbb.build());
190 mbb.end();
191 }
192
193 public void prepareTabs(final TabPane tabPane) {
194     Objects.requireNonNull(tabPane, "Cannot prepare null content box");
195
196     final Tab login = loginTab.createNode();
197     final Tab user = userTab.createNode();
198     final Tab menu = menuTab.createNode();
199     final Tab order = orderTab.createNode();
200     // user.setDisable(Boolean.TRUE);
201     // menu.setDisable(Boolean.TRUE);
202     // order.setDisable(Boolean.TRUE);
203
204     tabPane.getTabs()
205         .add(login);
206     tabPane.getTabs()
207         .add(user);
208     tabPane.getTabs()
209         .add(menu);
210     tabPane.getTabs()
211         .add(order);
212 }
213
214 @Override
215 public String getId() {
216     return "main-scene";
217 }
218 }

```



## Übung 3

### 1.11.2 MainSceneEventHandlerFactory.java

Folgend ist der Source der EventHandlerFactory für die Main Scene angeführt, die die Events für die Window Menus erstellt.

Listing 37: MainSceneEventHandlerFactory.java

```

1  package at.fh.ooe.swe4.fx.campina.view.scene;
2
3  import java.util.ArrayList;
4  import java.util.HashMap;
5  import java.util.List;
6  import java.util.Map;
7  import java.util.Objects;
8
9  import javafx.event.ActionEvent;
10 import javafx.event.EventHandler;
11 import javafx.event.EventType;
12 import at.fh.ooe.swe4.fx.campina.view.api.EventHandlerFactory;
13 import at.fh.ooe.swe4.fx.campina.view.scene.MainSceneViewHandler.MenuItemDefinition;
14
15 /**
16  * The event handler factory for the main scene.
17  *
18  * @author Thomas Herzog <thomas.herzog@students.fh-hagenberg.at>
19  * @date Jun 5, 2015
20  */
21 public class MainSceneEventHandlerFactory implements EventHandlerFactory<String> {
22
23     private static final long serialVersionUID = 6992457824162264209L;
24
25     /**
26      *
27      */
28     public MainSceneEventHandlerFactory() {
29     }
30
31     @SuppressWarnings("rawtypes")
32     @Override
33     public Map<EventType, List<EventHandler>> registerEventHandler(final String id) {
34         Objects.requireNonNull(id, "Cannot register event for null id");
35
36         final Map<EventType, List<EventHandler>> eventHandlers = new HashMap<>();
37         eventHandlers.put(ActionEvent.ACTION, new ArrayList<EventHandler>());
38
39         // new user event
40         if (id.equals(MenuItemDefinition.CLOSE.id)) {
41             eventHandlers.get(ActionEvent.ACTION)
42                 .add(new EventHandler<ActionEvent>() {
43                     @Override
44                     public void handle(ActionEvent event) {
45                         System.exit(0);
46                     }
47                 });
48         }
49
50         return eventHandlers;
51     }
52 }

```