Design Patterns I

Observer, Listener & MVC

Design Patterns I - Gliederung

- Was sind Design Patterns?
  - Definition von Design Patterns
  - Entstehung
  - Nutzen & Verwendung
- MVC - Model, View, Controller
- Observer & Listener
- JavaFX ActionEvent
  - Was sind Events?
  - Event Dispatch Chain
  - Event Type Hierarchy
  - MouseListener
  - KeyListener
  - Codebeispiel zu Key- & Mouselistener

Was sind Design Patterns?

Was sind Design Patterns? - Definition

Entwurfsmuster (englisch design patterns) sind bewährte Lösungsschablonen für wiederkehrende Entwurfsprobleme [...]. Sie stellen damit eine wiederverwendbare Vorlage zur Problemlösung dar, die in einem bestimmten Zusammenhang einsetzbar ist.

(Quelle: Wikipedia, Entwurfsmuster)

Was sind Design Patterns? - Entstehung

- Sammlung von Entwurfsmustern von Architekt Christopher Alexander zwischen 1977 und 1979
- Verwendung in der Software für die Erstellung grafischer Benutzeroberflächen mit Smalltalk von Kent Beck und Ward Cunningham 1987
- Verbreitung durch Promotion Erich Gammas und schließlich Publikation des Buches *Design Patterns – Elements of Reusable Object-Oriented Software* zusammen mit Richard Helm, Ralph Johnson, John Vlissides (Gang of Four) 1994

Was sind Design Patterns? - Nutzen & Verwendung

- Vier Elemente eines Design Patterns:
  1. Pattern Name
  2. Problem
  3. Solution
  4. Consequences

- Drei Arten von Design Patterns:
  - Creational Patterns (Erzeugungsmuster)
  - Structural Patterns (Strukturmuster)
  - Behavioral Patterns (Verhaltensmuster)

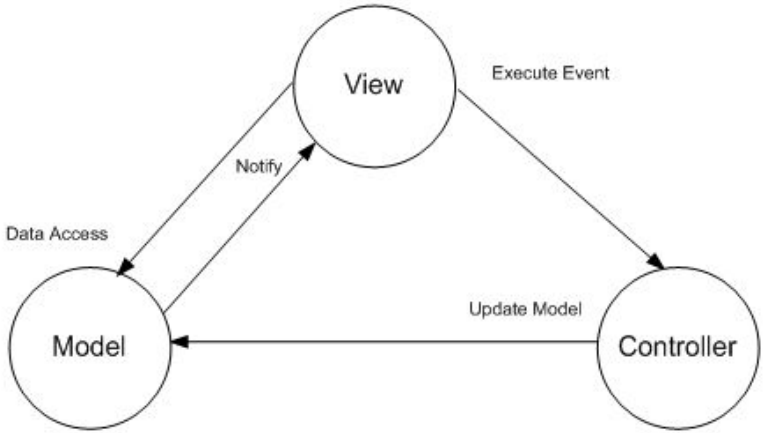
MVC - Model View Controller

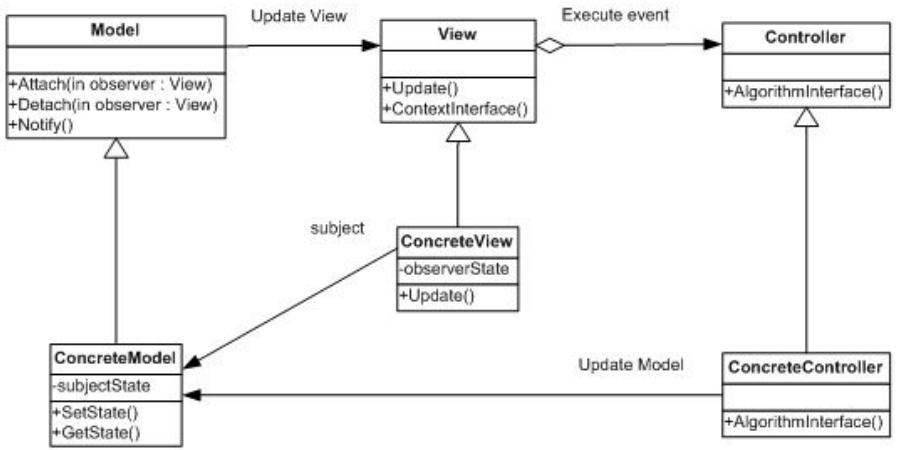
MVC - Definition

Model View Controller (**MVC**, englisch für *Modell-Präsentation-Steuerung*) ist ein Muster zur Trennung von Software in die drei Komponenten *Datenmodell* (englisch *model*), *Präsentation* (englisch *view*) und *Programmsteuerung* (englisch *controller*). Das Muster kann sowohl als Architekturmuster als auch als Entwurfsmuster eingesetzt werden.

(Quelle: Wikipedia, Model View Controller)

MVC - Abbildung und UML Diagramm



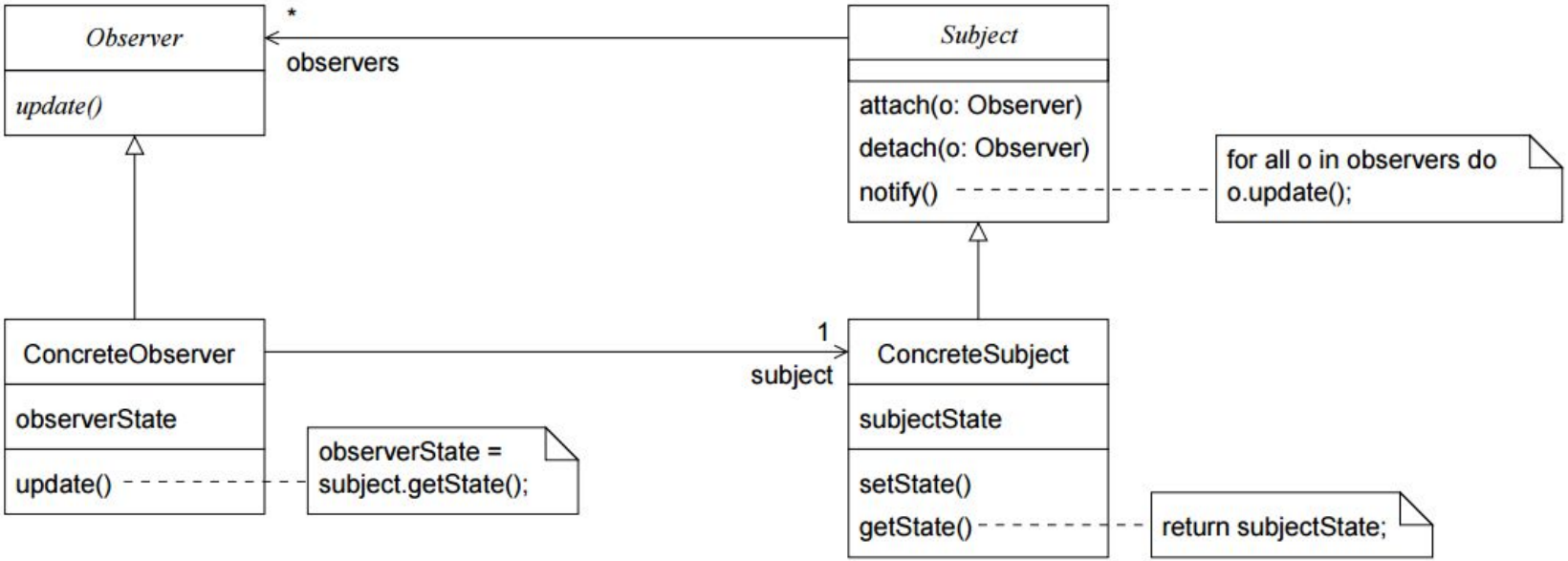


Observer & Listener

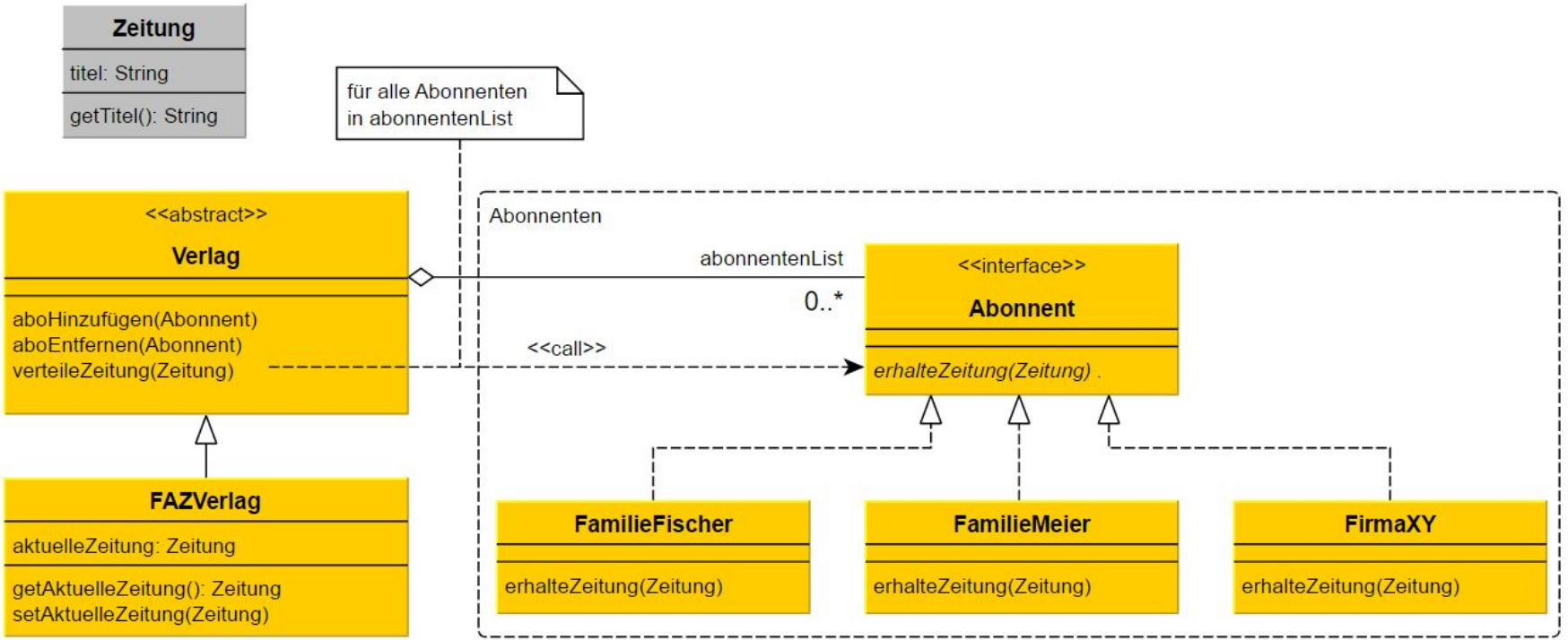
Observer & Listener - Idee

- Gehört zu den Verhaltensmustern
- Dient der Weitergabe von Änderungen an einem Objekt an von diesem Objekt abhängigen Strukturen
- Wird verwendet, wenn Änderungen an einem Objekt (z.B. Model) Änderungen an einem anderen Objekt nach sich zieht (z.B. View) oder wenn ein Objekt andere Objekte benachrichtigen soll, ohne diese im Detail zu kennen

Observer & Listener - Konzept UML Diagramm



Observer & Listener - Konkretes Beispiel UML



Observer & Listener - Java Codebeispiel

```
interface Observer<T> {
        public void update( Observable<T> o, T arg );
}
```

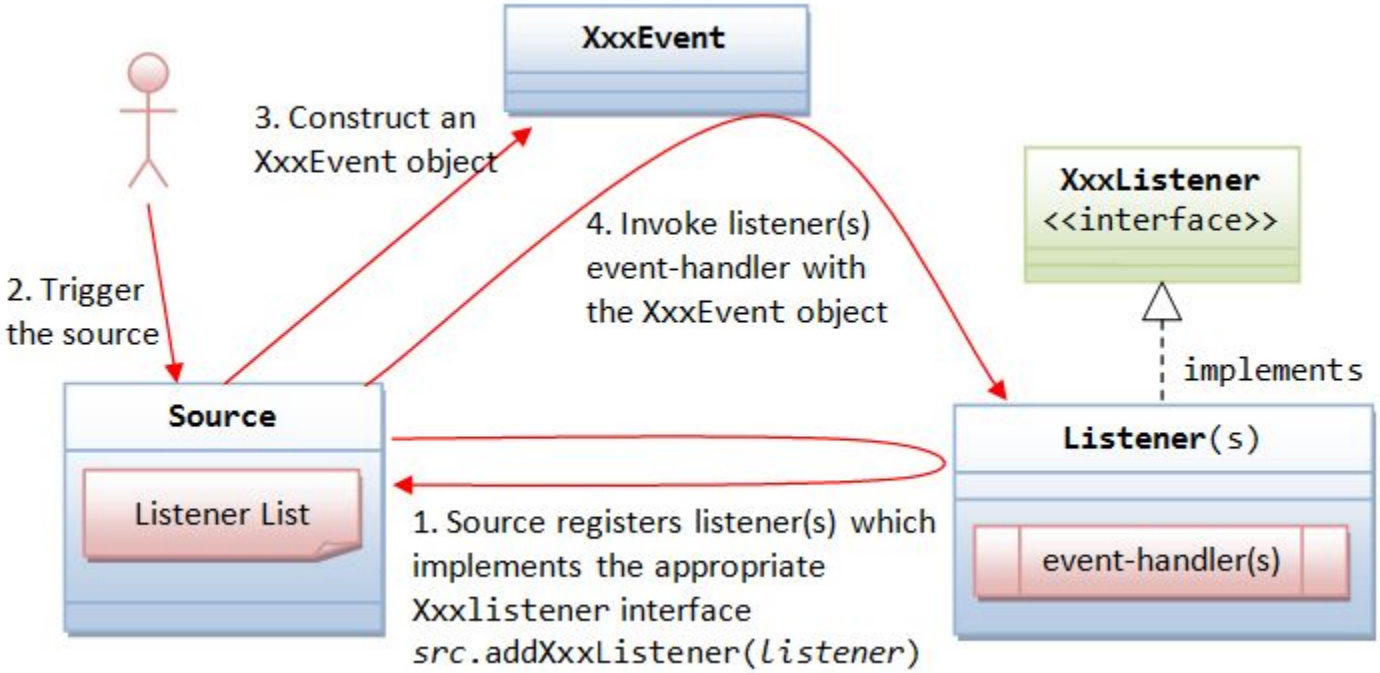
```
public class Observable<T> {
   private final List<Observer<T>> observers = new ArrayList<Observer<T>>();

   public void addObserver( Observer<T> observer ) {
      if ( ! observers.contains( observer ) )
         observers.add( observer );
   }

   public void deleteObserver( Observer<?> observer ) {
      observers.remove( observer );
   }

   public void notifyObservers( T arg ) {
      for ( Observer<T> observer : observers )
         observer.update( this, arg );
   }
}
```

```
class JokeListener implements Observer {
   final private String name;

   JokeListener( String name ) {
      this.name = name;
   }

   @Override public void update( Observable o, Object arg ) {
      System.out.println( name + " lacht über: \"" + arg + "\"" );
   }
}
```

```
class JokeTeller extends Observable {
   private static final List<String> jokes = Arrays.asList(
         "Eine Null kann ein bestehendes Problem verzehnfachen.",
         "Unsere Luft hat einen Vorteil: Man sieht, was man einatmet." );

   public void tellJoke() {
      setChanged();
      notifyObservers();
   }
}
```

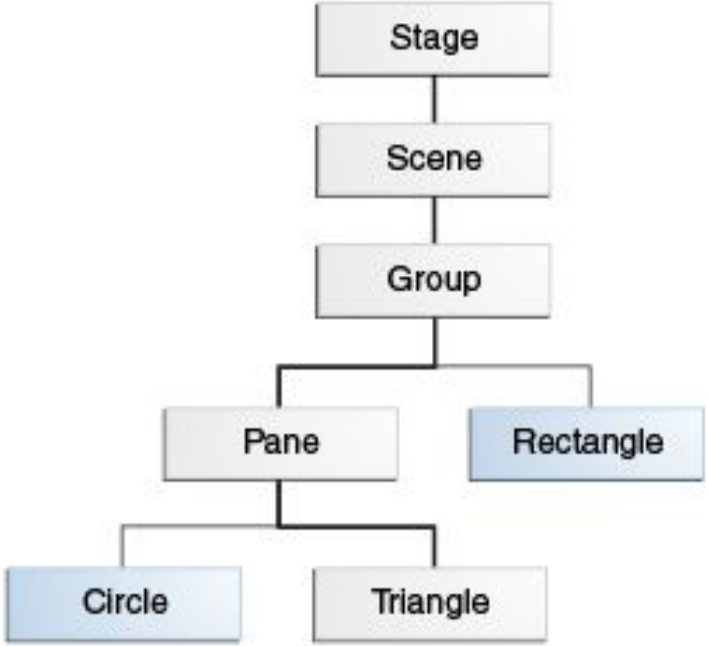
JavaFX ActionEvent

JavaFX ActionEvent - Was sind Events?

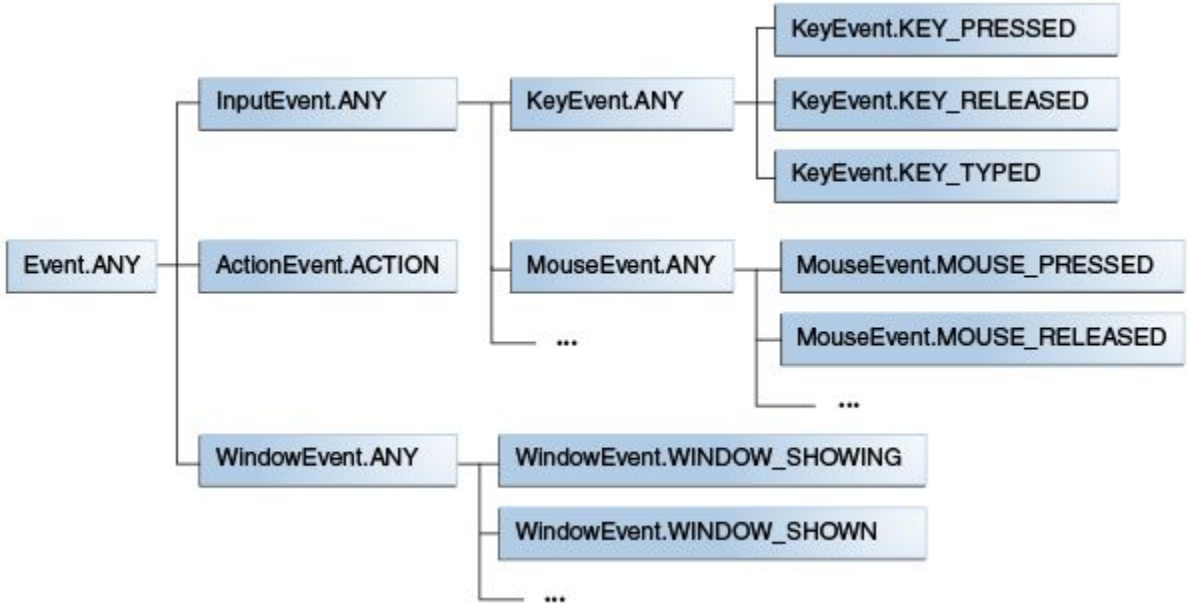


JavaFX ActionEvent - Event Dispatch Chain

- Interner Mechanismus bestimmt, welcher *Node* Ziel eines Ereignisobjekts ist
- Event-Bubbling: Behandlung der Events von unten nach oben (und Event wird möglicherweise bei Nichtbearbeitung vernichtet)



JavaFX ActionEvent - Event Type Hierarchy



JavaFX ActionEvent - MouseListener

- Interface, das folgende 5 abstrakten Methoden besitzt: mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased()
- Objekte der Klasse MouseEvent zeigen Ereignisse, die durch Mausklicks oder Mausbewegung ausgelöst werden
- werden immer erzeugt, wenn Maustaste gedrückt, geklickt oder losgelassen wird

- *MouseListener*: reagiert auf die Bedienung der Maustaste (obige Methoden) *MouseMotionListener*: erfasst die Mausbewegung (mouseDragged(), mouseMoved())

JavaFX ActionEvent - KeyListener

- Besitzt folgende Methoden, in denen jeweils ein KeyEvent als Parameter übergeben wird:
  - KeyPressed(KeyEvent e)
  - KeyReleased(KeyEvent e)
  - KeyTyped(KeyEvent e)

- weitere Methoden:
  - addKeyListener(KeyListener l)
  - removeKeyListener(KeyListener l)

JavaFX ActionEvent - Key- & MouseListener Code

```
private void handleKey(KeyEvent keyEvent) {
   int dist = 100;

   switch (keyEvent.getCode()) {
      case UP:     this.setTranslateY(this.getTranslateY() + dist); break;
      case DOWN:   this.setTranslateY(this.getTranslateY() - dist); break;
      case LEFT:   this.setTranslateX(this.getTranslateX() + dist); break;
      case RIGHT:  this.setTranslateX(this.getTranslateX() - dist); break;

      case BACK_SPACE:
         this.setTranslateX(0);
         this.setTranslateY(0);
         break;

      case EQUALS: this.setScale(this.getScaleX() * 1.25); break;
      case MINUS:  this.setScale(this.getScaleX() * 0.8); break;
      case DIGIT1: this.setScale(1); break;

      case Z:
         showInspector();
         break;

      case DELETE:
         removeSelected();
         break;
      default:
         break;
   }
}
```

```
private void setupListeners (final CalendarEntrySelectionListener selectionListener,
      final CalendarActionListener actionListener) {

   // notify selection listener on changes (if registered)
   if (selectionListener != null) {
      selected.addListener((observable, oldValue, newValue) -> //
      selectionListener.calendarEntrySelectionChanged(entry, newValue));
   }

   // update selection status when the user clicks on the entry label
   addEventHandler(MouseEvent.MOUSE_PRESSED, event -> {
      event.consume();

      if (!selected.get()) {
         selected.set(true);
      }
   });

   // register action listener for double clicks on the entry
   if (actionListener != null) {
      addEventHandler(MouseEvent.MOUSE_CLICKED, event -> {
         event.consume();

         if (event.getClickCount() > 1) {
            actionListener.onCalendarEntryAction(entry);
         }
      });
   }
}
```

Literatur und weiterführende Links

- *Design Patterns – Elements of Reusable Object-Oriented Software*
  Gang of Four: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- https://www.codeproject.com/Articles/42830/Model-View-Controller-Model-View-Presenter-and-Mod
- http://openbook.rheinwerk-verlag.de/javainsel/javainsel_10_002.html
- https://docs.oracle.com/javafx/2/events/processing.htm
- *...*

Vielen Dank für eure Aufmerksamkeit!