Innere Klassen + Ereignisverarbeitung

Programmiermethodik 2

Zum Nachlesen:
Christian Ullenboom: Java ist auch eine Insel, Kapitel 8:
http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_08_001.htm
Carl Dea et al.: JavaFX 8: Introduction by Example, Apress, 2014
Oracle Docu: http://docs.oracle.com/javase/8/javase-clienttechnologies.htm

Wiederholung

- Grafische Benutzeroberflächen mit JavaFX
- Fenster mit Knopf
- Anordnen von Komponenten
- Properties
- Tabellen
- Worker Threads
- Scene Builder

Ausblick

Agenda

- Innere Klassen
  - Mitgliedsklasse
  - Anonyme Innere Klasse
  - Weitere Typen
- Ereignisverarbeitung in JavaFX
  - Ereignisse und Event-Handler
  - Ereignis-Ursachen
  - Event-Handler-Umsetzungen

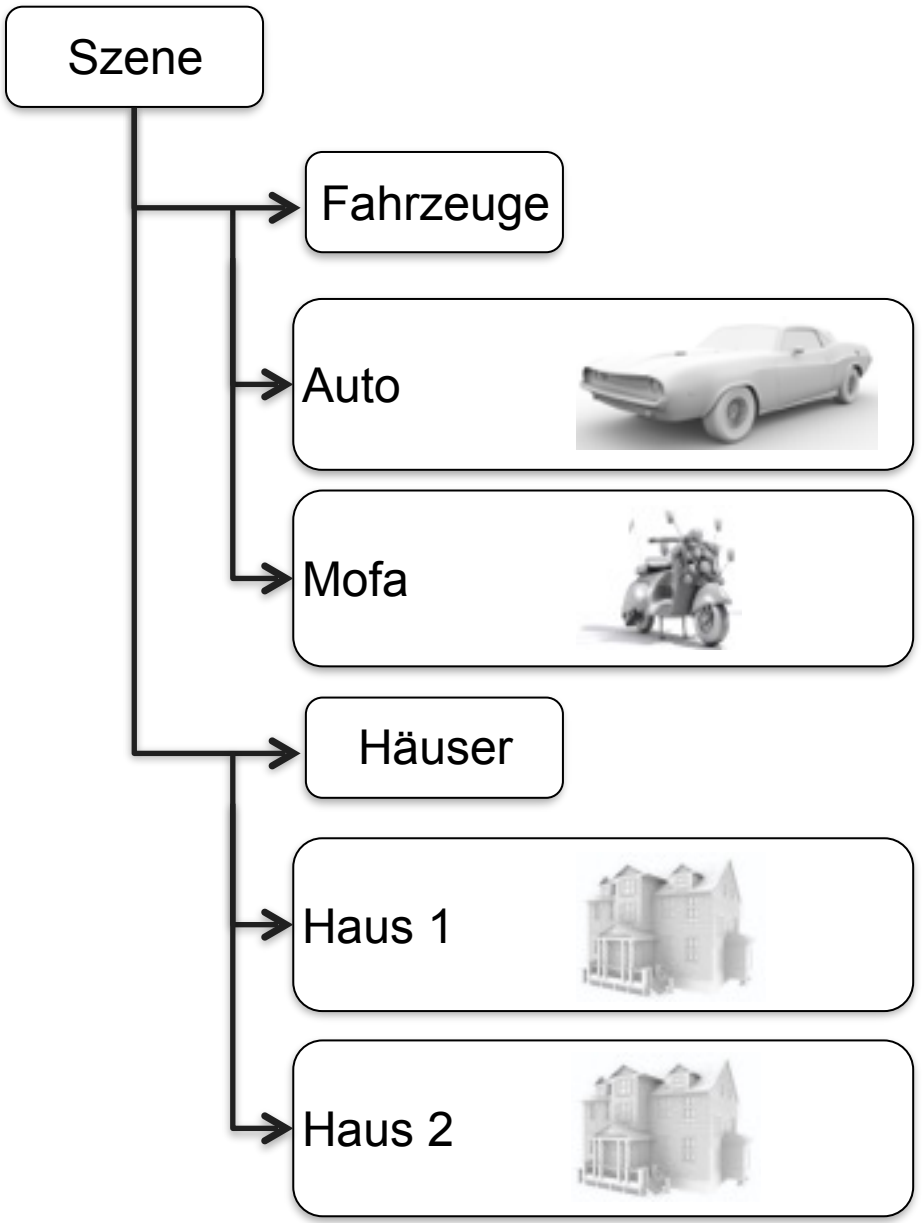
Zum Nachlesen:

Christian Ullenboom: Java ist auch eine Insel, Galileo Computing, 10. Auflage, 2011, Kapitel 8
http://openbook.galileocomputing.de/javainsel9/javainsel_08_001.htm, abgerufen am 24.04.2014

Innere Klassen

Beispiel

- Szenengraph (siehe JavaFX)
  - hierarchische Organisation von Objekten
  - Baumstruktur



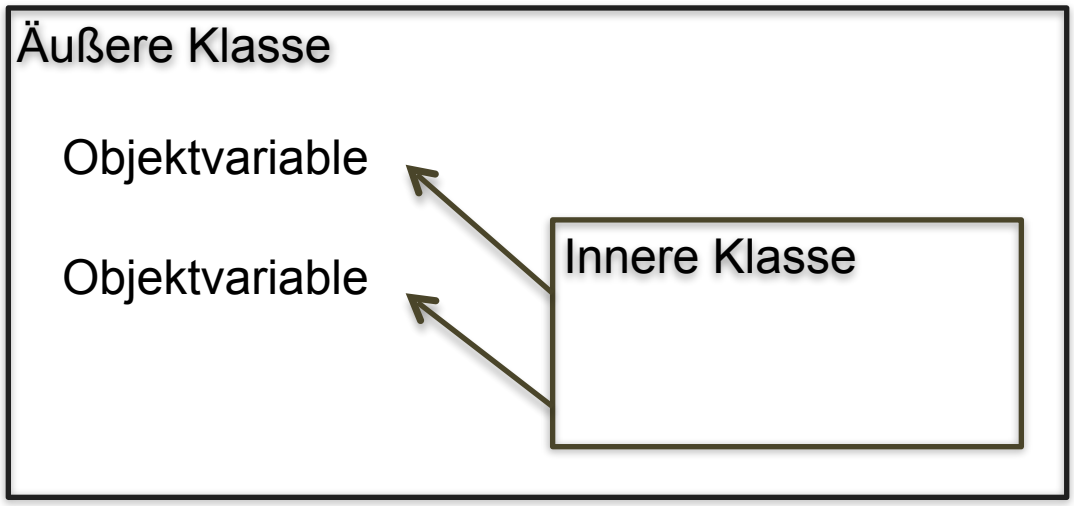
Einführung

- Suche im Szenengraph
  - interne Methode SzenenGraph.suche(...)?
  - zusätzliche Klasse ScenenGraphSuche?

```
public class SzenenGraph {

  private Knoten wurzelKnoten = null;

  public SzenenGraph(Knoten wurzelKnoten) {
    this.wurzelKnoten = wurzelKnoten;
  }

  public Knoten getWurzelKnoten( ) {
    return wurzelKnoten;
  }
}
```

Innere Klasse

- *engl. nested class*
- eigene Klasse
- innerhalb des Körpers definiert
  - Innerhalb einer anderen (äußeren) Klasse
  - Innerhalb eines Interfaces
- kann (fast) nur über äußere Klasse instanziiert werden



Typen

- in Java unterscheidet man vier Ausprägungen von inneren Klassen
  - Mitgliedsklasse
  - Statische Innere Klasse
  - Lokale Klasse
  - Anonyme Innere Klasse

Mitgliedsklasse

Mitgliedsklasse

```
public class Paket {
  private class StandortTracking {
    private List<String> standorte =
        new ArrayList<String>();
    public void standortHinzufuegen(
        String standort) {
      standorte.add(standort);
    }

    @Override
    public String toString() {
      return standorte.stream().reduce("",
          (standort1, standort2) -> standort1
          + " " + standort2);
    }
  }

 private StandortTracking standortTracking =
    new StandortTracking();

  public void paketWirdBearbeitet(
     LocalDateTime zeitpunkt, String standort) {
    standortTracking.standortHinzufuegen(
        standort);
  }

  @Override
  public String toString() {
    return "Paket:" + standortTracking;
  }
}
```

```
Paket paket = new Paket();
paket.paketWirdBearbeitet(
    LocalDateTime.of(2015, Month.JANUARY,
    1, 17, 17),"München");
paket.paketWirdBearbeitet(
    LocalDateTime.of(2015, Month.JANUARY,
    2, 12, 6), "Stuttgart");
paket.paketWirdBearbeitet(
    LocalDateTime.of(2015, Month.JANUARY,
    3, 8, 44), "Kassel");
paket.paketWirdBearbeitet(
    LocalDateTime.of(2015, Month.JANUARY,
    4, 19, 34), "Hamburg");
System.out.println(paket);
```

- Ausgabe:

Paket: München Stuttgart Kassel Hamburg

Mitgliedsklasse (oder Elementklasse)

- wird auch Elementklasse genannt
- engl. *member class* oder *element class*
- Zugriff auch auf statische und nicht-statische Member der äußeren Klasse
    - auch auf private Member
- benötigt Instanz der äußeren Klasse
- darf selber keine statischen Member haben

Benennung von Inneren Klassen

- Unterscheidung zwischen Package-Name und Klassenname
- Klein- und Großschreibung

```
pm2.mitgliedsklasse
pm2.mitgliedsklasse.Outer
pm2.mitgliedsklasse.Outer.Inner
```

Übung: Verschlüsselung

- Gegeben ist folgende Klasse zur Verschlüsselung von Buchstaben (a-z)
- Idee: Addiere zum Index des zu verschlüsselnden Buchstaben den Index des letzten verschlüsselten Buchstabens

Kapseln Sie die Verschlüsselung in eine Mitgliedsklasse!

```
public class VerschluesselungMitGedaechtnis {
  /**
   * Merkt sich den letzten Buchstaben.
   */
  char letzterBuchstabe = 'a';

  public char kodiere(char buchstabe) {
    char kodierterBuchstabe = (char) (buchstabe + letzterBuchstabe - 'a');
    letzterBuchstabe = buchstabe;
    while (kodierterBuchstabe > 'z') {
      kodierterBuchstabe = (char) (kodierterBuchstabe - 26);
    }
    return kodierterBuchstabe;
  }

  public static void main(String[] args) {
    VerschluesselungMitGedaechtnis verschluesselung =
        new VerschluesselungMitGedaechtnis();
    System.out.println(verschluesselung.kodiere('b')); // = b
    System.out.println(verschluesselung.kodiere('b')); // = b+b = 1+1 = 2 = c
    System.out.println(verschluesselung.kodiere('d')); // d + b = 3 + 1 = 4 = e
  }
}
```

Anonyme Innere Klasse

Anonyme Innere Klasse

```
public class AnonymeInnereKlasse {
  public static void main(final String[] args) {
    // Anonyme innere Klasse für Punkte, die nur gerade Koordinaten haben.
    Point geradeZahlenPunkt = new Point() {
      @Override
      public void setLocation(int x, int y) {
        super.setLocation((x % 2 == 0) ? x : x + 1, (y % 2 == 0) ? y : y + 1);
      }

      private static final long serialVersionUID = -4796272828855630554L;
    };

    // Ein normaler Punkt
    Point normalerPunkt = new Point();
    // Beide Punkte mit den gleichen Werten initialisieren
    normalerPunkt.setLocation(23, 42);
    geradeZahlenPunkt.setLocation(23, 42);
    // Ausgabe
    System.out.println("Normaler Punkt: (" + normalerPunkt.x + ", "
        + normalerPunkt.y + ")");
    System.out.println("Gerade-Zahlen-Point: (" + geradeZahlenPunkt.x + ", "
        + geradeZahlenPunkt.y + ")");
  }
}
```

Ausgabe:

```
(23,42)
(24,42)
```

Anonyme Innere Klasse

- Abgeleitet von Klasse oder Interface
- engl. anonymous inner class
- Darf selber kein extends oder implements definieren
- Schnittstellen nur über äußere Klasse oder Interface
- Hat keinen eigenen Konstruktor (super() ist implizit)
    - aber: Exemplarinitialisierungsblöcke
    - Point p = new Point(){{ x = -1; y = -1; }};
- Anwendung
    - Threads
    - "Kleine" Erweiterung bestehender Klasse
- Zugriff nur auf final Member der äußeren Klasse

Bytecode-Dateinamen

- Statische Innere Klasse, Mitgliedsklasse, Lokale Klasse

  Outer.class

  Outer$Inner.class

- Anonyme Klasse

  Outer.class

  Outer$1.class

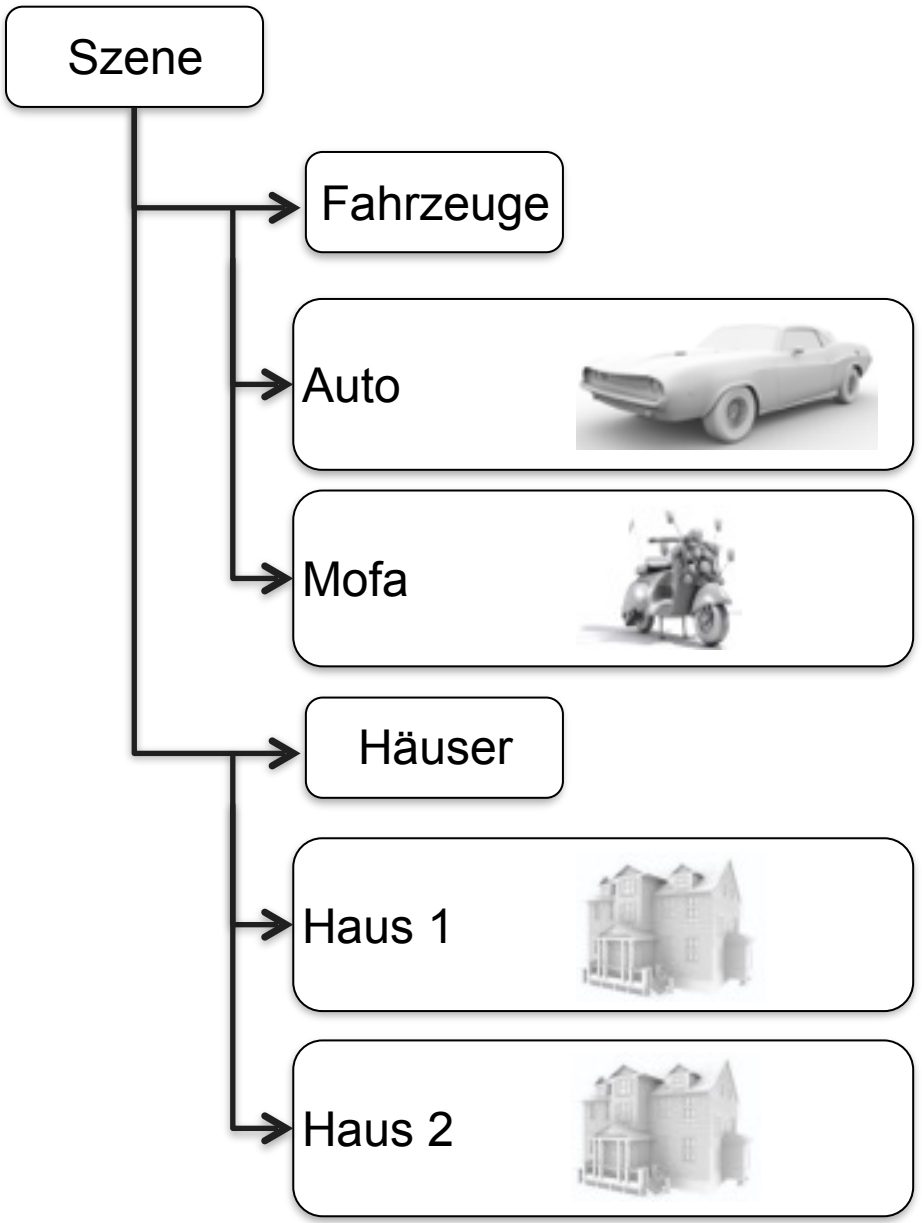
  Outer$2.class

  ...

Zurück zum Beispiel

- Suche als innere Klasse für den Szenengraphen

- Vorteile
  - Kapselung
  - Sichtbarkeit
  - Objektorientiertes Design



Weitere Typen

- Statische Innere Klasse
- Lokale (Innere) Klasse

Ereignisverarbeitung

Ereignisverarbeitung

- Feststellung
  - GUI-Komponenten müssen auf Benutzerinteraktionen reagieren können

- Beispiele
  - Mausklick
  - Tastatureingabe
  - Schließen eines Fensters, ...

- Idee
  - jede Benutzerinteraktion löst ein Ereignis aus (engl. event)
  - GUI-Komponente ist Event-Quelle (engl. event source)

Ereignisverarbeitung

- Frage
  - Wie kann auf Ereignis reagiert werden?
  - Wie wird über ein Ereignis informiert?

Anwendungscode

- Implementierung eines vorgegebenen Interfaces

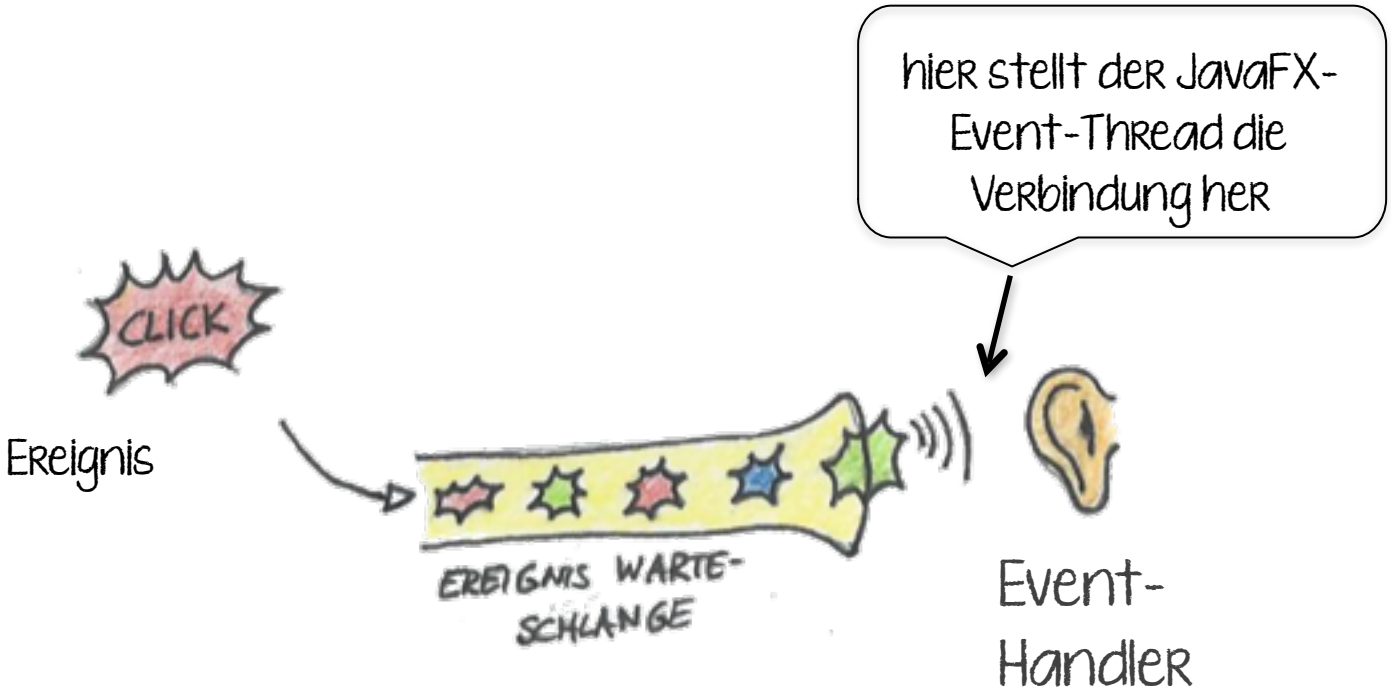
```
public class FensterMitKnopfEventHandler implements
    EventHandler<ActionEvent> {
  @Override
  public void handle(ActionEvent event) {
    System.out.println("Knopf geklickt.");
  }
}
```

- Registrierung bei der Event-Quelle

```
FensterMitKnopfEventHandler eventHandler =
        new FensterMitKnopfEventHandler();
ToggleButton knopf = new ToggleButton();
knopf.setText("Knopf!");
knopf.setOnAction(eventHandler);
```

Ablauf der Ereignisverarbeitung

- Anwender klickt auf den Button
- Ereignis vom Typ ActionEvent wird erzeugt
  - vom Event-Erzeuger (Betriebssystem)
- Ereignis wird in die JavaFX-Event-Queue gestellt



Ablauf der Ereignisverarbeitung

- JavaFX-Event-Thread ermittelt alle Objekte, die für den Button einen Event-Handler registriert haben
    - hier: FensterMitKnopfEvenHandler
- JavaFX-Event-Thread führt die entsprechende Methode aus
        - hier: handle()

Event-Handler

- allgemeine Form des EventHandlers

  `EventHandler<T>`

- Registrierung: addEventHandler(EventHandler<T>)
  - z.B. `addEventHandler(ActionEvent.ACTION, new ActionEventHandler())`

- verschiedene Ereignisse (Auszug): ActionEvent, InputEvent, DragEvent, MouseEvent, TouchEvent, WindowEven t,…

Event-Handler

- zusätzlich zur generischen Registrierungs-Methode

  `addEventHandler(EventHandler<T>)`

- Bequemlichkeitsmethoden (Auszug)
  - engl. convenience methods
  - z.B. `setOnAction(EventHandler<ActionEvent>);`

Event-Handler

- können auch wieder entfernt (deregistriert werden)

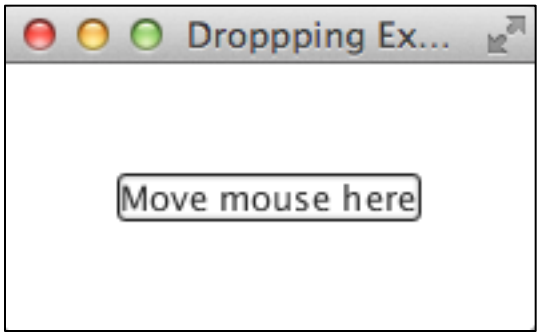
```
removeEventHandler(Event<T>, EventHandler<? super T>)
```

- auch hier: Bequemlichkeitsmethoden

```
z.B. setOnMouseDragged(null)
```

Übung: Ereignis-Verarbeitung

- Schreiben Sie Quellcode zur Darstellung eines JavaFX-Labels. Fährt man mit der Maus über das Label, soll die Mausposition auf der Konsole ausgegeben werden.
- Hinweise
  - Maus-Ereignis: MouseEvent.MOUSE_MOVED
  - Koordinaten: MouseEvent.getSceneX(), MouseEvent.getSceneY()



Ereignis-Ursachen

Ereignisverarbeitung - Identifikation

- bisher: zentralisierte Ereignisverarbeitung
- ein einziger Event-Handler verarbeitet alle Ereignisse
  - Implementiert das Interface EventHandler
  - Definiert eine Methode handle()
- handle() – Methode herausfinden, welche Komponente das Ereignis ausgelöst hat

Eine Klasse pro Komponentenereignis

- jede Komponente erhält für jedes mögliche Ereignis eine Event-Handler-Klasse
- Nachteil
    - sehr viele kleine Klassen
    - unübersichtlich

Quelle und ID

- Verwenden einer eindeutigen ID pro Komponente

```
FensterMitMehrerenKnoepfenEventHandler eventHandler =
        new FensterMitMehrerenKnoepfenEventHandler();
Button knopf1 = new Button("Knopf 1");
knopf1.setId(KNOPF_1_ID);
knopf1.setOnAction(eventHandler);
Button knopf2 = new Button("Knopf 2");
knopf2.setId(KNOPF_2_ID);
knopf2.setOnAction(eventHandler);
```

- Abfragen der Quelle und ID des Ereignisses im Event-Handler

```
@Override
public void handle(ActionEvent event) {
  Object ereignisVerursacher = event.getSource();
  if (ereignisVerursacher instanceof Control) {
    Button knopf = (Button) ereignisVerursacher;
    switch (knopf.getId()) {
      case FensterMitMehrerenKnoepfen.KNOPF_1_ID:
        System.out.println("Knopf 1 geklickt.");
        break;
      case FensterMitMehrerenKnoepfen.KNOPF_2_ID:
        System.out.println("Knopf 2 geklickt.");
        break;
    }
  }
}
```

Event-Handler-Umsetzungen

Innere Klassen

- Umsetzung des Event-Handlers mit einer anonymen inneren Klasse

```
Button knopf = new Button("Knopf!");
knopf.setOnAction(new EventHandler<ActionEvent>() {
 @Override
  public void handle(ActionEvent arg0) {
    System.out.println("Knopf gedrückt!");
  }
});
```

- Vorteil
  - Ereignisverarbeitung im Code an gleicher Stelle wie Quelle
- Nachteil
  - unübersichtlich bei komplexer Ereignisverarbeitung
  - selten sinnvoll bei mehreren Ereignis-Quellen zu verwenden

Lambda-Ausdrücke

- Umsetzung mit Lambda-Ausdruck
- Umzusetzen: Methode handle():

```
void handle(T event);
```

- Implementierung mit Lambda-Ausdruck:

```
(ereignis) -> { <etwas-machen> }
```

- oder kompakter:

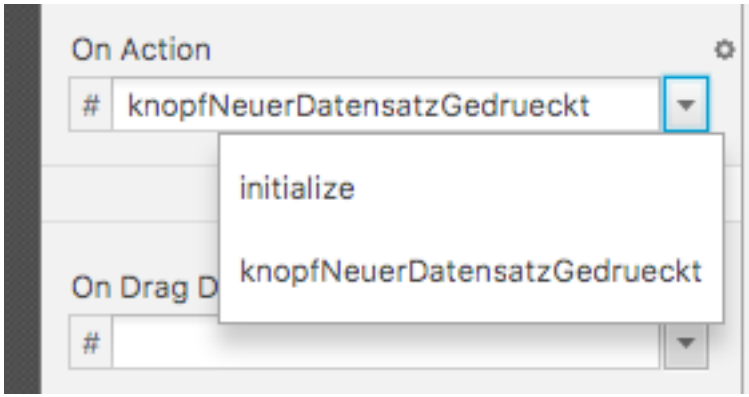
```
Button knopf = new Button("Knopf!");
knopf.setOnAction((ereignis) -> {
  System.out.println(ereignis);
});
```

Scene Builder

- Handler-Methode im Controller

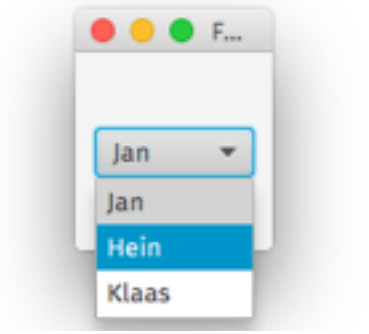
```
@FXML
protected void knopfNeuerDatensatzGedrueckt(ActionEvent event) {
  personen.add(new Person(textAlias.getText(),
    textVorname.getText(), textNachname.getText()));
}
```

- Handler im Scene Builder verknüpfen (Inspector, Code)



Übung: ComboBox

- Gegeben ist folgende Klasse, die ein JavaFX-Fenster mit einer Auswahlbox erstellt.
- Erweitern Sie die Klasse, sodass der aktuelle Eintrag auf der Konsole ausgegeben wird, wenn er sich verändert.
- Verwenden Sie dazu
  - einen Lambda-Ausdruck,
  - eine anonyme innere Klasse.




```
public class FensterMitComboBox extends Application {
  @Override
  public void start(Stage primaryStage) {
    ObservableList<String> eintraege =
        FXCollections.observableArrayList(
        "Jan", "Hein", "Klaas");
    ComboBox<String> auswahlBox = new
        ComboBox<String>(eintraege);
    auswahlBox.setValue(eintraege.get(0));

    // Mit Lambda-Ausdruck
    auswahlBox
        .setOnAction(ereignis ->
        System.out.println(auswahlBox.getValue()));

    // Mit anonymer innerer Klasse
    auswahlBox.setOnAction(
        new EventHandler<ActionEvent>() {
      @Override
      public void handle(ActionEvent arg0) {
        System.out.println(auswahlBox.getValue());
      }
    });

    primaryStage.setTitle("Fenster mit Knopf");
    StackPane root = new StackPane();
    root.getChildren().add(auswahlBox);
    primaryStage.setScene(new Scene(root, 100, 100));
    primaryStage.show();
  }

  public static void main(String[] args) {
    launch(args);
  }
}
```

Zusammenfassung

- Innere Klassen
  - Mitgliedsklasse
  - Anonyme Innere Klasse
  - Weitere Typen
- Ereignisverarbeitung in JavaFX
  - Ereignisse und Event-Handler
  - Ereignis-Ursachen
  - Event-Handler-Umsetzungen