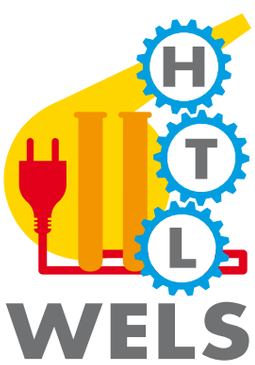
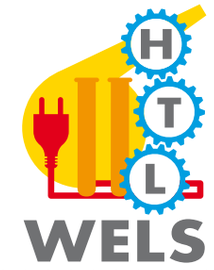
JAVAFX 8 – FORTGESCHRITTENE THEMEN

SEW

DI Thomas Helml



INHALT



- Properties & Bindings
- Concurrency in JavaFX

PROPERTIES & BINDINGS

PROPERTIES & BINDINGS

- Benutzeroberfläche
  - stellt Zustand von Datenobjekten dar
  - gibt Benutzer die Möglichkeit diesen Zustand zu ändern
- Bsp: Schieberegler für Breite eines Rechtecks
  - Wert auslesen
  - width des Models aktualisieren
  - Berechnung der Rechtecksfläche anstoßen
  - zeichnen

PROPERTIES

- Neu mit JavaFX: Properties

```
public class MyBean {
    private StringProperty sample = new SimpleStringProperty();
    public String getSample() {
        return sample.get();
    }
    public void setSample(String value) {
        sample.set(value);
    }
    public StringProperty sampleProperty() {
        return sample;
    }
}
```

PROPERTIES

- **Einfache Properties (abstrakte Klassen):**
  - BooleanProperty
  - DoubleProperty
  - FloatProperty
  - IntegerProperty
  - LongProperty
  - StringProperty

PROPERTIES

- Erzeugen von Properties über **konkrete** Klassen (Konstruktor mit maximalen Parametern):

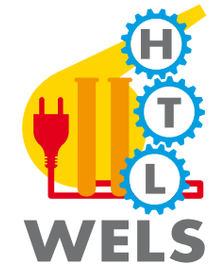
```
BooleanProperty booleanProperty = new
SimpleBooleanProperty(true, „b”, this);

DoubleProperty doubleProperty = new
SimpleDoubleProperty(1.5, „d”, this);

FloatProperty floatProperty = new
SimpleFloatProperty(1.5f, „f”, this);

IntegerProperty integerProperty = new
SimpleIntegerProperty(123, „i”, this);

LongProperty longProperty = new
SimpleLongProperty(12345678991, „l”, this);

StringProperty stringProperty = new
SimpleStringProperty("hallo", „s”, this);
```

PROPERTIES

- Object Properties
  - speichern beliebige Objekte

```
ObjectProperty<Image> objectProperty =
new SimpleObjectProperty<>();
```

BINDINGS



- Bindings „verknüpfen“ Properties mit Werten

```
label.textProperty().bind(myBean.sampleProperty());
```

- Binding lösen:

```
label.textProperty().unbind();
```

INITIALIZE

- Wo werden Bindings erstellt?
  - im Controller!
  - 2 Möglichkeiten:
    - `Interface Initializable`
    - `@FXML initialize`

INITIALIZE

- Variante 1:

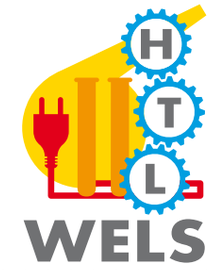
```
public class Controller implements Initializable {

  @Override
  public void initialize(URL location, ResourceBundle resources) {
     // Bindings here
  }
  …
}
```

INITIALIZE

- Variante 2:

```
public class Controller {

  @FXML
  public void initialize() {
     // Bindings here
  }
  …
}
```

ÜBUNG PROPERTIES



- Übung: **SimpleBindings**

CALCULATED BINDINGS

- Bindings realisieren eine komplexe Beziehungen zwischen Werten (Properties):
  - Unterscheide: High-Level- und Low-Level-API
    - High-Level:
      - Bindings-Klasse
      - Fluent-API
  - Low-Level-API

BINDINGS

- Bindings-Klasse

```
DoubleProperty number1 = new SimpleDoubleProperty(1);

DoubleProperty number2 = new SimpleDoubleProperty(2);

DoubleProperty number3 = new SimpleDoubleProperty(3);
```

```
NumberBinding calculated = Bindings.add(

    number1, Bindings.multiply(number2,number3));
```

FLUENT API

- Fluent-API

```
DoubleProperty number1 = new SimpleDoubleProperty(1);

DoubleProperty number2 = new SimpleDoubleProperty(2);

DoubleProperty number3 = new SimpleDoubleProperty(3);
```

```
NumberBinding calculated =

        number1.add(number2.multiply(number3));
```

LOW LEVEL API

- Low-Level-API

```
DoubleProperty number1 = new SimpleDoubleProperty(1);
DoubleProperty number2 = new SimpleDoubleProperty(2);
DoubleProperty number3 = new SimpleDoubleProperty(3);


NumberBinding calculated = new DoubleBinding() {
    {
      super.bind(number1, number2, number3);
    }
    @Override
    protected double computeValue() {
      return number1.get() + (number2.get() * number3.get());
    }
};
```

NUMERISCHE BINDINGS

- Berechnungen mit numerischen Bindings:
  - `.add / .substract`
  - `.multiply / .divide`
  - `.negate`
  - `.min / .max`

BIDIREKTIONALE BINDINGS

- 2 Properties können gegenseitig aneinander gebunden werden:

```
DoubleProperty number1 = new SimpleDoubleProperty(1);

DoubleProperty number2 = new SimpleDoubleProperty(2);

number2.bindBidirectional(number1);
```

- Dann können Sie auch gebundene Properties gesetzt werden:

```
number2.setValue(3);

number1.setValue(4);

System.out.println(„number2 hat Wert: „ + number2.getValue());
```

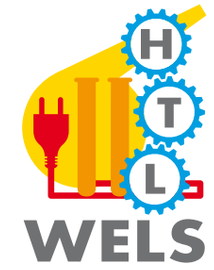
ÜBUNG BINDINGS

- Übung: **CalculatedBindings**

OBJECT BINDINGS

- Object Bindings
  - mit ObjectBindings können beliebige Objekte an Properties gebunden werden
  - Vorgehensweise:
    - eigene Klasse ableiten von `ObjectBinding<T>`
    - im `Konstruktor` ein passendes Property annehmen (muss nicht vom Typ T sein!)
    - Property als Attribut speichern
    - `T computeValue()` Methode implementieren, die ein Objekt retourniert, welches über das Property gebindet wurde

ÜBUNG BINDINGS



- Übung: **ImageViewer**

BOOLEAN BINDINGS

- Boolean Bindings: Logische Verknüpfung von BooleanProperty
  - .greaterThan / .greaterThanOrEqualTo
  - .isEqualTo / .isNotEqualTo
  - .lessThan / .lessThanOrEqualTo
  - .and / .or
  - isNotEmpty
  - …

BOOLEAN BINDINGS

- Bsp:
  - in einem Textfield überprüfen, ob Eingabe mit mindestens 3 Zeichen:

```
BooleanBinding textFieldEntered =
   textField.textProperty()
   .isNotEmpty()
   .and(textField.textProperty().length().greaterThan(3));
```

- Button soll deaktiviert werden, wenn im Textfield nichts steht

```
button.disableProperty().bind(textFieldEntered.not());
```

ÜBUNG BINDINGS

- Übung: **207_BooleanBindings**

SERIALISIERUNG VON PROPERTIES

- Properties können NICHT serialisiert werden
  - daher müssen sie in dem Fall mit transient gekennzeichnet werden
  - sie werden dann aber auch NICHT gespeichert!

```
public class MyBean implements Serializable {
    private transient StringProperty sample = new SimpleStringProperty();
    public String getSample() {
        return sample.get();
    }
    public void setSample(String value) {
        sample.set(value);
    }
    public StringProperty sampleProperty() {
        return sample;
    }
}
```

SERIALISIERUNG VON PROPERTIES

- Möchte man Datenobjekte mit Properties serialisieren, dann muss man muss man die Serialisierung selbst in die Hand nehmen (z.B. StringProperty):

```
public class xyz implements Externalizable {

    private SimpleStringProperty x = new SimpleStringProperty("");

    @Override
    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
        setX ((String) in.readObject());
    }

    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeObject(getX());
    }
```

KLASSEN MIT PROPERTIES SERIALISIEREN

```java
public class Packet implements Externalizable {
    private static final long serialVersionUID = -8256294089416034037L;
    private SimpleStringProperty varName = new SimpleStringProperty("");
    private SimpleStringProperty varValue = new SimpleStringProperty("");

    public Packet() {
        this("", "");
    }
    public Packet(String varName, String varValue) {
        setVarName(varName);
        setVarValue(varValue);
    }
    public String getVarName() {
        return varName.get();
    }
    public void setVarName(String var) {
        varName.set(var);
    }

    public String getVarValue() {
        return varValue.get();
    }
    public void setVarValue(String value) {
        varValue.set(value);
    }
    public SimpleStringProperty getVarNameProperty() {
        return varName;
    }
    public SimpleStringProperty getVarValueProperty() {
        return varValue;
    }

    @Override
    public String toString() {
        return getVarName() + ": " + getVarValue();
    }

    @Override
    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
        setVarName((String) in.readObject());
        setVarValue((String) in.readObject());
    }

    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeObject(getVarName());
        out.writeObject(getVarValue());
    }
}
```

LISTVIEW

- siehe JavaFX Dokumentation:
  - https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/list-view.htm#CEGGEDBF
  -

ERZEUGEN + BEFÜLLEN EINER LISTVIEW

```
// Definiere ListProperty -> es sollen Strings gespeichert werden
ListProperty<String> listProperty = new SimpleListProperty<>();

// Erstelle Collection mit Inhalt
List<String> iosList = new ArrayList<>();

iosList.add("iPhone 4");
iosList.add("iPhone 5");
iosList.add("iPhone 6");
iosList.add("iPhone SE");
iosList.add("iPhone 8");

// setzen der Werte durch umwandeln einer ArrayList in eine observableList
listProperty.set(FXCollections.observableArrayList(iosList));

// binde die ListView Items an die ListProperty
myListView.itemsProperty().bind(listProperty);

// ändere Elemente in der listProperty
listProperty.add("iPhone11");
listProperty.remove(0);
➤
```

LISTVIEW

Die Anweisung:

```
... myListView.getSelectionModel().getSelectedItem()
```

liefert das ausgewählte Item

Hingegen:

```
... myListView.getSelectionModel().selectedItemProperty()
```

liefert das Property des ausgewählten Elements für Bindings

➤

BINDINGS VON COLLECTIONS

- Beispiel 208_ListBindings
  - `ListView` verfügt über `itemsProperty` an welches ein `ListProperty` „gebindet“ werden kann
  - `ObserverableArrayList` ist eine spezielles `ListProperty`, welches „beobachtbar“ ist
  - `ObservableArrayList` kann über Hilfsklasse `FXCollections` aus einer Standard-Collection erzeugt werden

BINDINGS VON COLLECTIONS

- Beispiel ListBindings