Leibniz Universität Hannover

Interactive Systems (ISY)

Auditorium Exercise 03

Human-Computer Interaction Group

Jan Feuchter
jan.feuchter@hci.uni-hannover.de

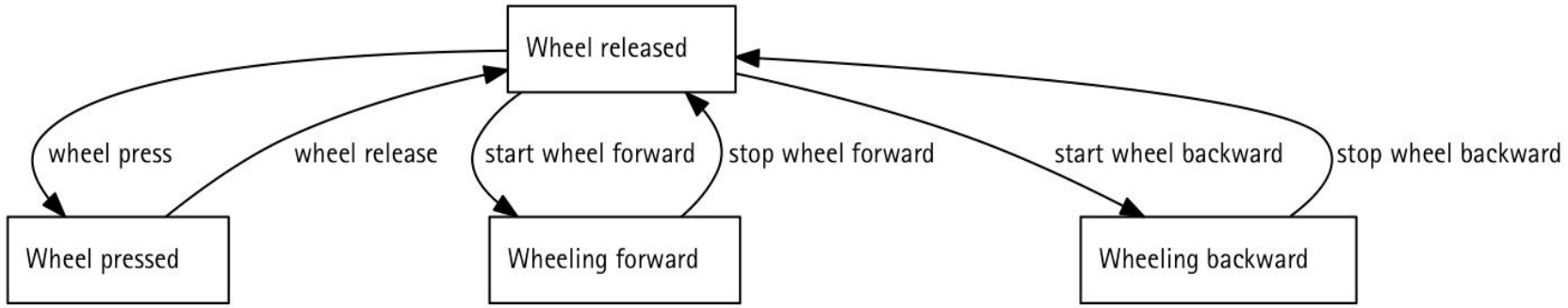
Lectures

Session	Date	Topic	Details
1	2.4.	Introduction	human performance, empirical research, modeling
2	9.4.	Interaction elements	input devices, interaction elements, states, layouts
3	16.4.	Event handling	events, bindings, reactive programming, scene graph
4	23.4.	Scene graphs	event delivery, coordinate systems, nodes, animation, concurrency
5	30.4.	Interaction techniques	alignment and pointing techniques
6	7.5.	Interaction techniques	
7	14.5.	Web-based user interfaces	document object model, client-server issues
	21.5.	Pfingstwoche	
8	28.5.	Web-based user interfaces	reactive Programming for the Web
9	4.6.	Experiments and data analysis	designing experiments, hypothesis testing
10	11.6.	Modeling interaction	descriptive and predictive models, keystroke-level model, regression
11	18.6.	Visualization	visual encodings, perceptual accuracy, treemaps, dynamic queries
12	25.6.	Human-Centered AI	introduction to human-centered AI, human control and automation, examples
13	2.7.	Deep learning in HCI	guidelines for human-AI interaction, neural networks
14	9.7.	Deep learning in HCI	convolutional and recurrent NNs, face recognition, gesture recognition

ASSIGNMENT 2

Aufgabe 1: Maus mit Scrollrad

- 6 Actions (exclude simultaneous button presses)
  - Start/stop wheel forward,
    start/stop wheel backward, wheel press/release
- 4 States (exclude simultaneous button presses)
  - Wheel released, wheeling forward,
    wheeling back, wheel pressed
- State transition diagram



Aufgabe 2: Instrumental Interaction (Beaudouin-Lafon)

- Properties of interaction instruments
  - Degree of indirection: spatial and temporal offsets between the logical part of the instrument and the domain object
  - Degree of integration: ratio between the number of DOFs of the logical part of the instrument and the number of DOFs of the input device
  - Degree of compatibility: similarity between the user's physical actions and the response of the domain object
- Instrumental interaction model helps to analyze interfaces
  - Menus and toolbars as meta-instruments
  - Dialog boxes to specify complex commands
  - Handles used for graphical editing

Beaudouin-Lafon. Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. CHI 2000.

Aufgabe 2: Instrumental Interaction (Beaudouin-Lafon)

a) Menu

high degree of indirection, shift of attention

low (menu bars) to medium (context menus) degree of integration

low degree of compatibility

b) Dialogbox

high degree of indirection (spatial and temporal)

low degree of integration

low degree of compatibility

c) “Handles” for graphical objects

low degree of indirection

high degree of integration

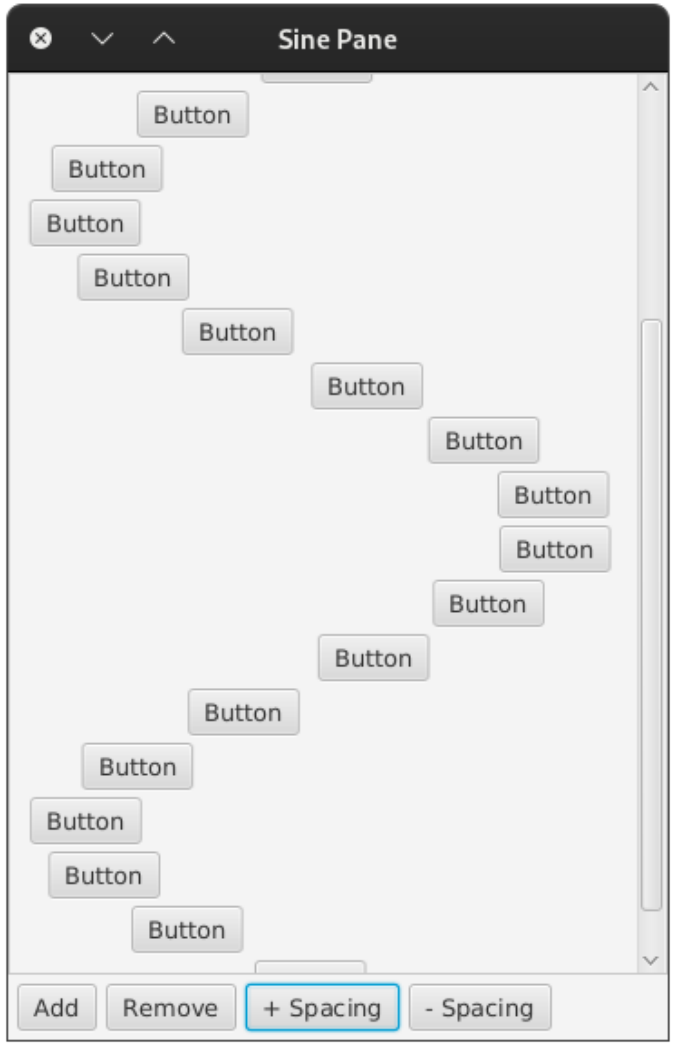
high degree of compatibility

Augabe 3: GUI Design für ältere Menschen

- a) Größe von Buttons
  - >11,43 mm² für einzelne Buttons
  - >16,51 mm² für mehrere Buttons
- b) Spacing
  - Zwischen 3,17 mm und 12,7 mm
  - Teilnehmer*innen präferierten 6,35 mm
  - Zu großer Abstand erhöht Suchzeit
- c) Beispiele für GUIs in allen Altersgruppen
  - Geldautomat, Radios, Fernseher, Fahrkartenautomaten, …

Aufgabe 4

- Probleme?
- Fragen?



Assignment 2

- Textual solutions need to be exported to PDF
  - No .docx, .odt, .md, .txt
  - Otherwise our tutors cannot be sure that they see everything as intended
  - And thus won’t grade your solutions
- Programming exercises need to be exported to a Zip archive
  - Otherwise our tutors cannot be sure that they run the project the way you intended
- This will be enforced starting with the 3rd assignment

Lecture Recap

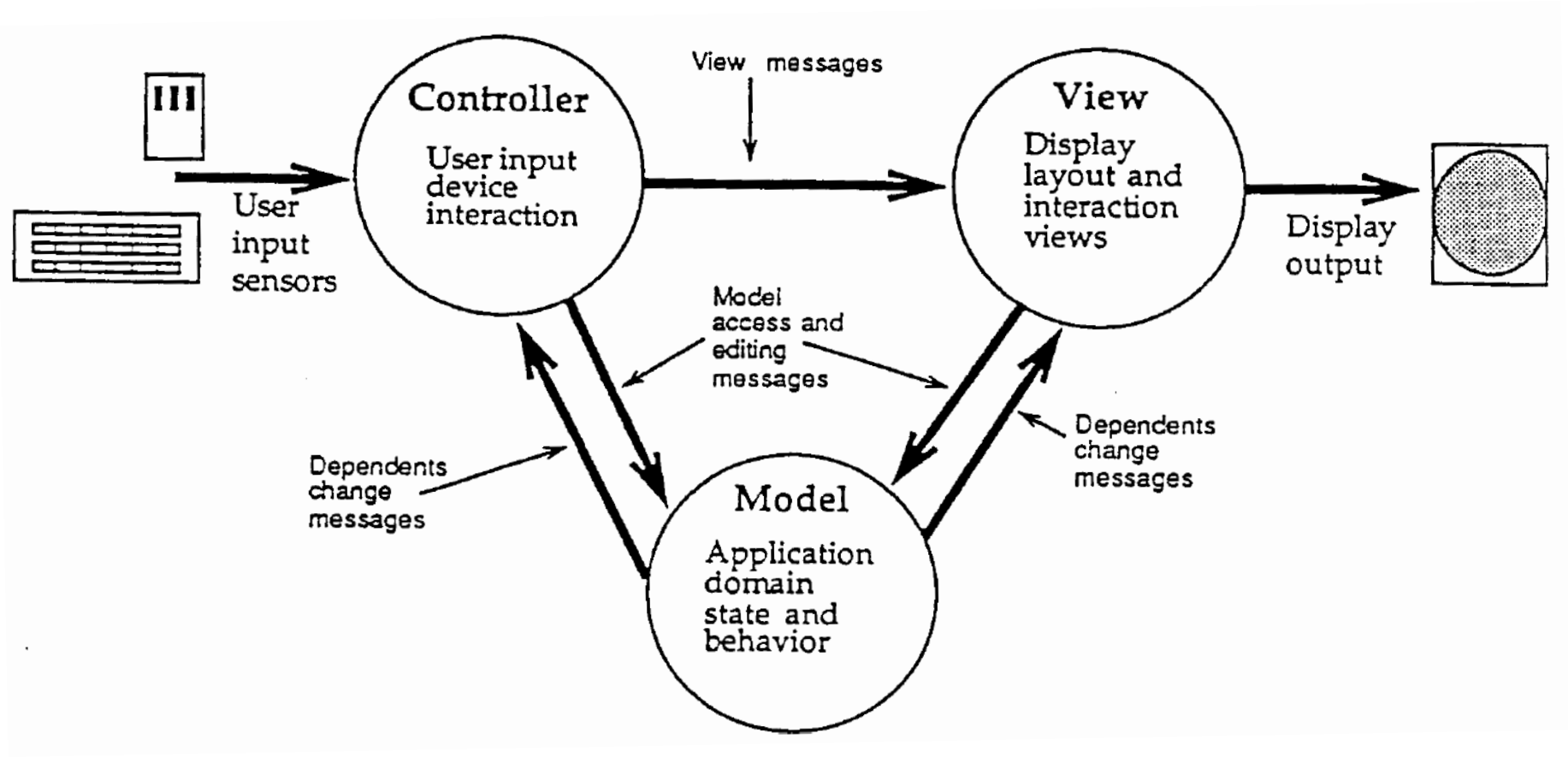
EVENT STREAMS & REACTIVE PROGRAMMING

Wo spielen “Events” eine Rolle?

- A: Bei der Strukturbeschreibung von Benutzungsschnittstellen.

- B: Bei der Verhaltensbeschreibung von Benutzungsschnittstellen.

Model-View-Controller (MVC) Pattern



Krasner, Pope. A cookbook for using the model–view controller user interface paradigm in Smalltalk-80. The Journal of Obj. Tech., Aug–Sep 1988.

Unterschied zwischen

javafx.collections ObservableList<E>

und

java.util.List<E>

interface ObservableList<E> extends List<E>, Observable

Welche Aussage ist korrekt?

Beim ListView **Focus Model** ist die Mehrfachauswahl möglich.

Beim ListView **Selection Model** ist die Mehrfachauswahl möglich.

Warum verwalten JavaFX Widgets ihre Eigenschaften als “Properties”?

Leibniz Universität Hannover

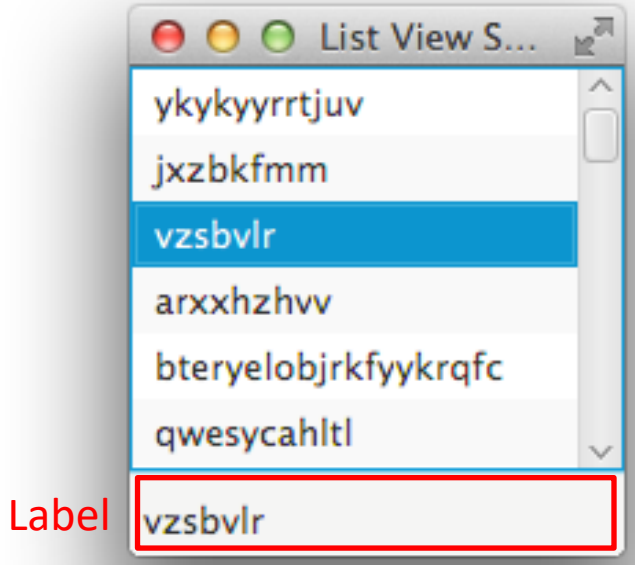
Example: Label Widget has a String Property (abbreviated)

```
class Label {
    ...
    private StringProperty text;
    public final StringProperty textProperty() {
        if (text == null) {
            text = new SimpleStringProperty(this,    "text",    "");
        }
        return text;
    }
    public final void setText(String value) {
textProperty().setValue(value); }
    public final String getText() { return text == null ? "" :
text.getValue(); }
    ...
}
```

name

Label

initial value



Leibniz Universität Hannover

Example: Simple String Property (abbreviated)

```
public class SimpleStringProperty extends ... {
    private String value; // initial/unbound value
    private ObservableValue<? extends String> observable; // bound to, if
    not null
    public SimpleStringProperty(Object bean, String name, String
    initialValue) {...}
    public void addListener(InvalidationListener listener) {...}
    public void addListener(ChangeListener<? super String> listener) {...}
    public String get() {...}
    public void set(String newValue) {...} // only if unbound
    public void bind(ObservableValue<? extends String> newObservable) {
       ...
       observable = newObservable;
       observable.addListener(this);
       markInvalid();
}  }
```

Reactive Programming

- ”Reaktive Programmierung” ist ein Paradigma, das den Datenfluss bzw. Ereignisfluss in den Vordergrund stellt.

- Richtig?

User Interfaces

- UI needs to react to and coordinate events
  - Mouse clicks, button presses, multitouch gestures, received data, etc.
- Control flow is driven by events (inverted control)
  - Difficult to predict or control events
  - State change propagation to dependencies is complex (with listeners)
- Event listeners, callbacks
  - No return value, side-effects
  - “Callback hell”
- Analysis of Adobe desktop applications
  - 33% of the code does event handling
  - 50% of reported bugs exist in event handling code

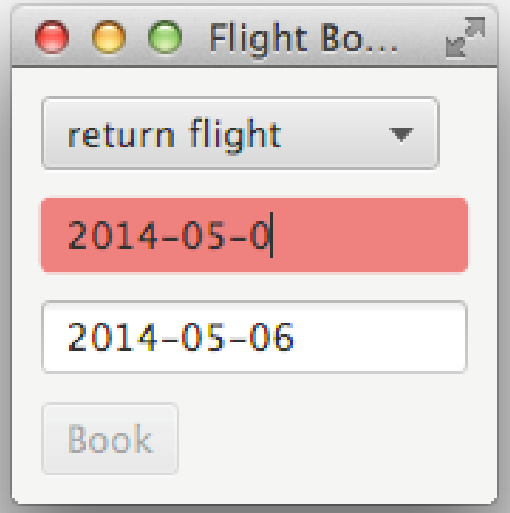
Bainomugisha et al. A survey on reactive programming. ACM Comput. Surv. 45, 4, Article 52 (August 2013).

Complex Widget Constraints with EventStreams

```
EventStream<Boolean> oneWayFlightStream = map(flightType.valueProperty(), "one-way flight"::equals);
link(oneWayFlightStream, returnDate.disableProperty());

EventStream<Boolean> startDateOkStream = map(startDate.textProperty(), Util::isDateString);
EventStream<Boolean> returnDateOkStream = map(returnDate.textProperty(), Util::isDateString);
link(startDateOkStream, Util::dateStyle, startDate.styleProperty());
link(returnDateOkStream, Util::dateStyle, returnDate.styleProperty());

link(oneWayFlightStream, startDateOkStream, returnDateOkStream,
      (o, s, r) -> o ? !s : !s || !r || isDateAfter(startDate.getText(),
      returnDate.getText()),
      book.disableProperty());
```



https://eugenkiss.github.io/7guis/tasks#flight

Imperative Programming (Pseudocode)

```
let c = 1
let b = 2

let a = c + b  // a == 3

c = 2

// a == ?
```

Reactive Programming (Pseudocode)

```
let c = 1
let b = 2

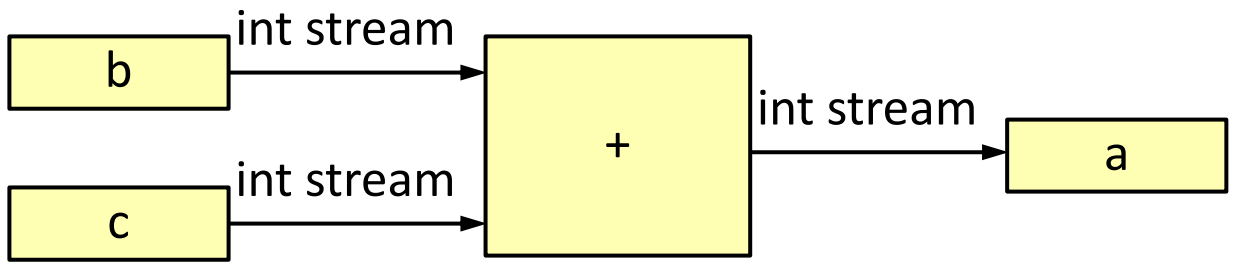
let a <- c + b  // a == 3

c = 2

// a == ?
```

Reactive Programming

- Reactive programming
  - *a* <- *b* + *c*; // pseudo code
  - *a* is dependent on *b* and *c*
  - Dependency graph
  - Recomputation (+) automatically happens as *b* or *c* change



Interface Observable

- Informs other objects when its state changes
- Registering for changes
  - void addListener(InvalidationListener listener);
  - void removeListener(InvalidationListener listener);
- Invalidation listeners

```
public interface InvalidationListener {
    void invalidated(Observable observable);
}
```

- Lazy evaluation
  - Content may be marked as invalid,
    but not immediately recomputed after a change
  - Recomputed the next time the content is requested

Leibniz Universität Hannover

JavaFX Script (obsolete), bind keyword

- Could be simpler, if Java had a “bind” keyword
- Variable semantics
  - Java: Variables only change when explicitly set, no notion of time-varying values/events/signals
  - Required: Variable can change on events, expressions are continuously evaluated, language support for setting up the structure of the dataflow
- Example: JavaFX Script (obsolete, but valuable for illustrating the idea)

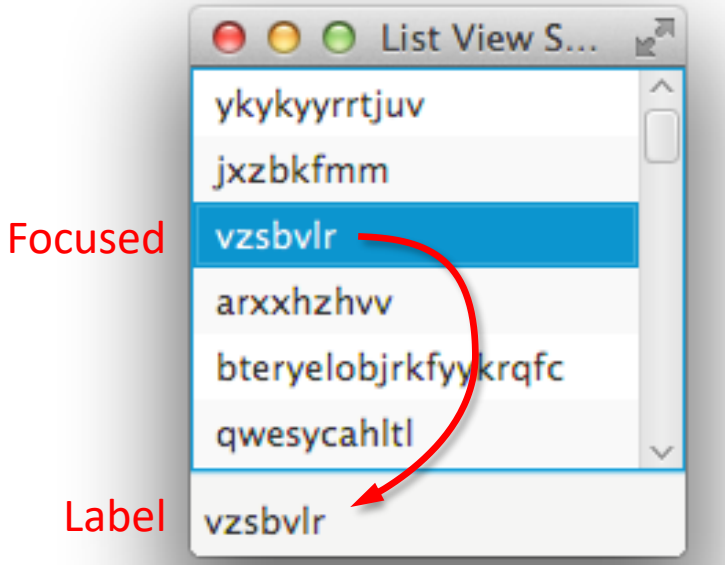
```
var priceInput: TextBox;
var quantityInput: TextBox;
var price: Double = bind new Double(priceInput.text);
var quantity: Double = bind new Double(quantityInput.text);
var total: Double = bind price * quantity;
var total: Label { text: bind "Total: ${total}"  };
```

bind: variable tracks changes of its source

Leibniz Universität Hannover

Tracking State Changes in JavaFX

- JavaFX Nodes allow for fine-granular state tracking
- State of a Node (e.g., a ListView) is a set of “properties”
  - Properties allow registering listeners
  - Each property change can thus be detected and reacted upon
- Example: Properties of ListView
  - Items, selection model, focus model, context menu, height, width, hover, etc.
  - Register listener with focus model to get notified when focus changes, then update label with the focused item



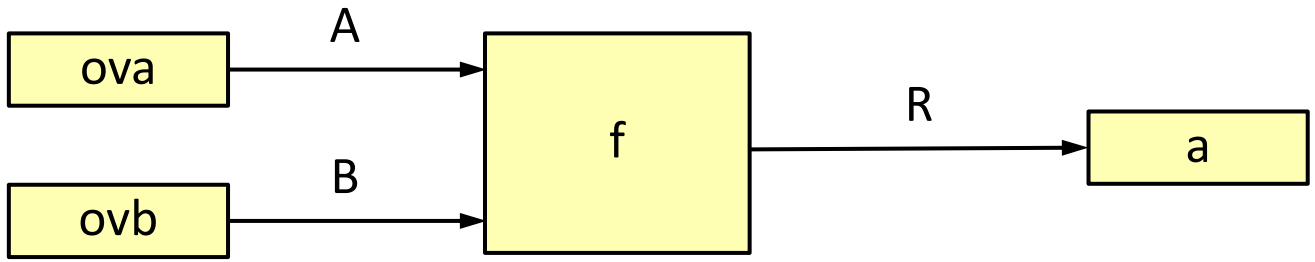
Practical Example

REACTIVE PROGRAMMING: JAVAFX EVENTSTREAMS

LET'S IMPLEMENT A SIMPLE CALCULATOR

Implementing the Pseudocode Example in ReactiveJavaFX

- ```
  static <A, B, R> EventStream<R> map(
      Function1<A, B, R> f, ObservableValue<A> ova, ObservableValue<B> ova);
  ```

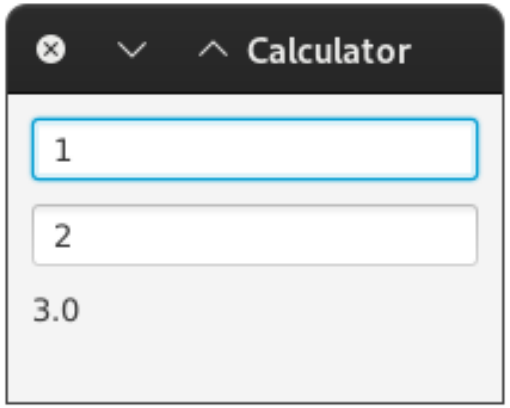



- ```
  EventStream<Integer> a = map((x, y) -> x + y, b, c);
  ```

Calculator Example

- Three approaches in JavaFX (as discussed here)
- Listeners
  - Treat properties as observable values and react to changes
- Bindings
  - Create a binding which is updated when one of the dependencies change
- EventStreams
  - Create a reactive mapping of multiple properties into one value

Calculator Example



- Three approaches in JavaFX (as discussed here)
- Template: `Calculator.zip` (in StudIP)

- Implement behaviour with:
  - Listeners
  - Bindings
  - EventStreams

- You may simply comment out the finished approaches

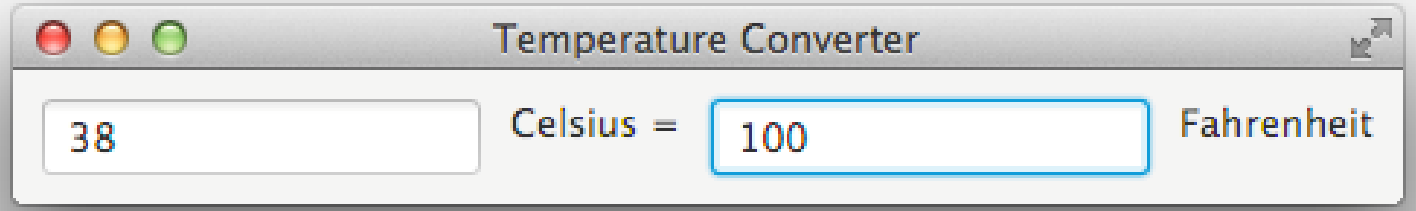
Lecture Recap

BIDIRECTIONAL DATAFLOW

Leibniz Universität Hannover

Bidirectional Bindings Example: Temperature Converter

```
TextField celsius = new TextField();
TextField fahrenheit = new TextField();
HBox root = new HBox(10, celsius,
                         new Label("Celsius ="),
                         fahrenheit,
                         new Label("Fahrenheit"));
```

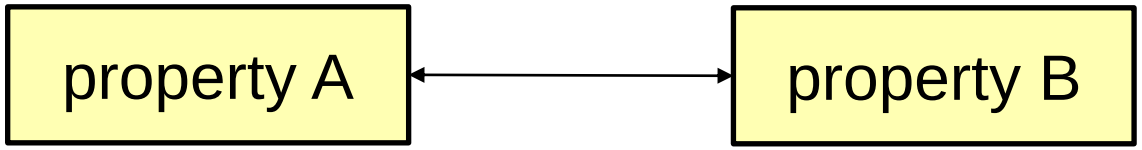


Author: E. Kiss

Issues with Bidirectional DataFlow

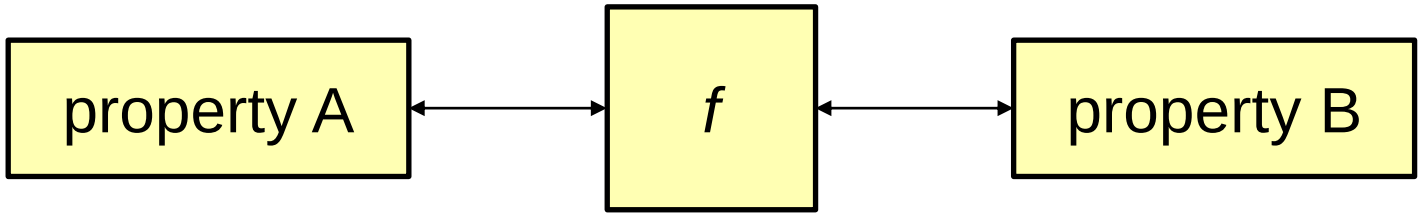
- In theory: endless cycle in reactive graph
- In praxis: avoided by not further propagating the backwards change
  - The observable value will be set to the value it already has, thus not causing a *changed* or *invalidation* event
- Explicit bidirectional APIs are often provided

JavaFX allows this



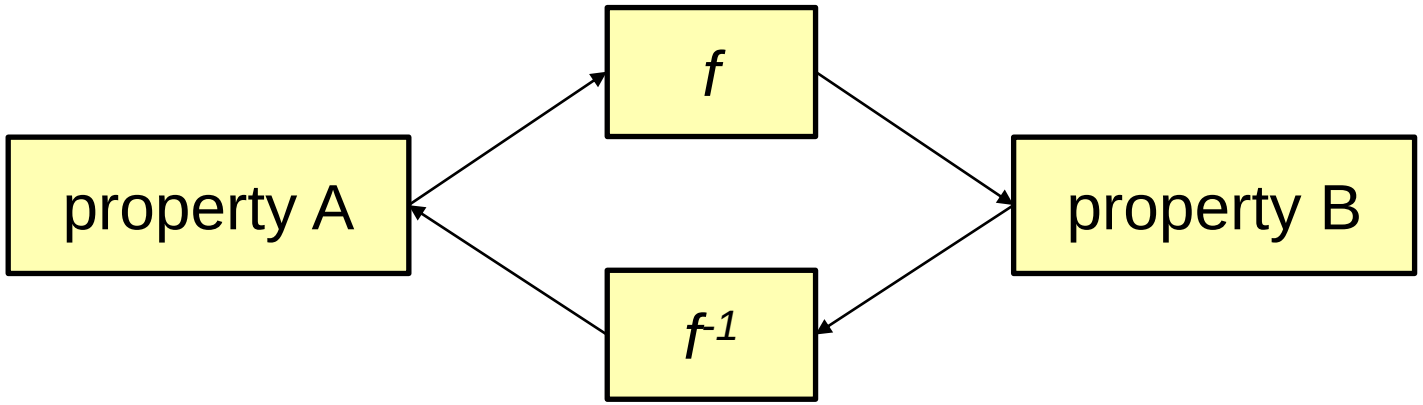
- Directly bind two properties to each other
  - `void Property.bindBidirectional(Property<T> other);`
- i.e. if A is set to some value, B will be set to the same value

JavaFX does not allow this



- This would also bind two properties to each other
- But we would also transform the value between them
- In JavaFX terms:
  - We'd, for example, create a *binding* from A
  - Apply a function onto that *binding*
  - Bind its result to B
  - And then do the same but in reverse

In Effect:



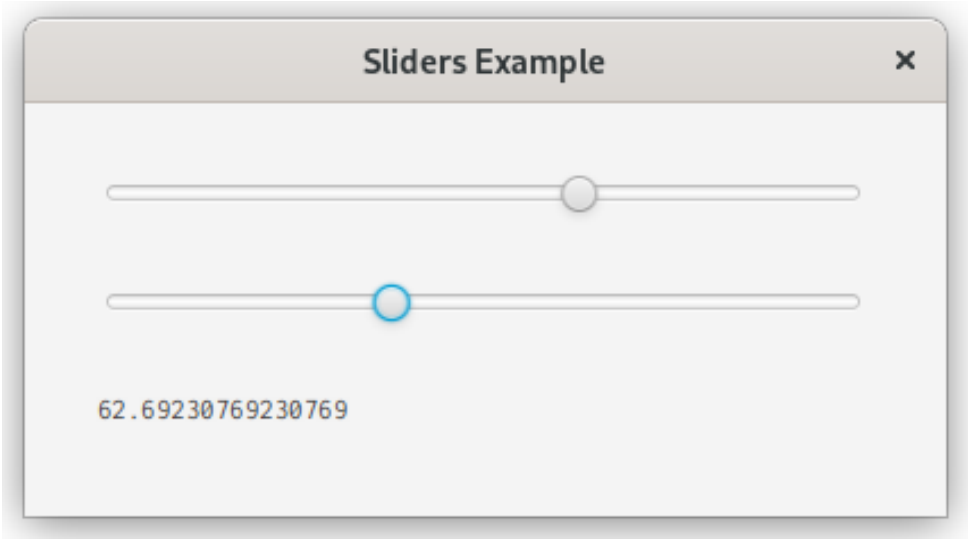
- Automatically calculating f^{-1} is not possible (for general Java functions)
- Thus, it would have to be provided explicitly
- For this example, we'll provide a `bindBidirectional` function:
  - ```
    BidirectionalBinding<A, B> bindBidirectional(
     Property<A> a, Property<B> b,
     Function1<A, B> aToB, Function1<B, A> bToA
    );
    ```
  - Where `aToB` is f and `bToA` is f^{-1}
    ```

LET'S IMPLEMENT A BIDIRECTIONAL EXAMPLE

Two Sliders

- Aim: Have two sliders
- One slider inverts the position of the other slider
- Both sliders should be movable
  - Bidirectional data flow

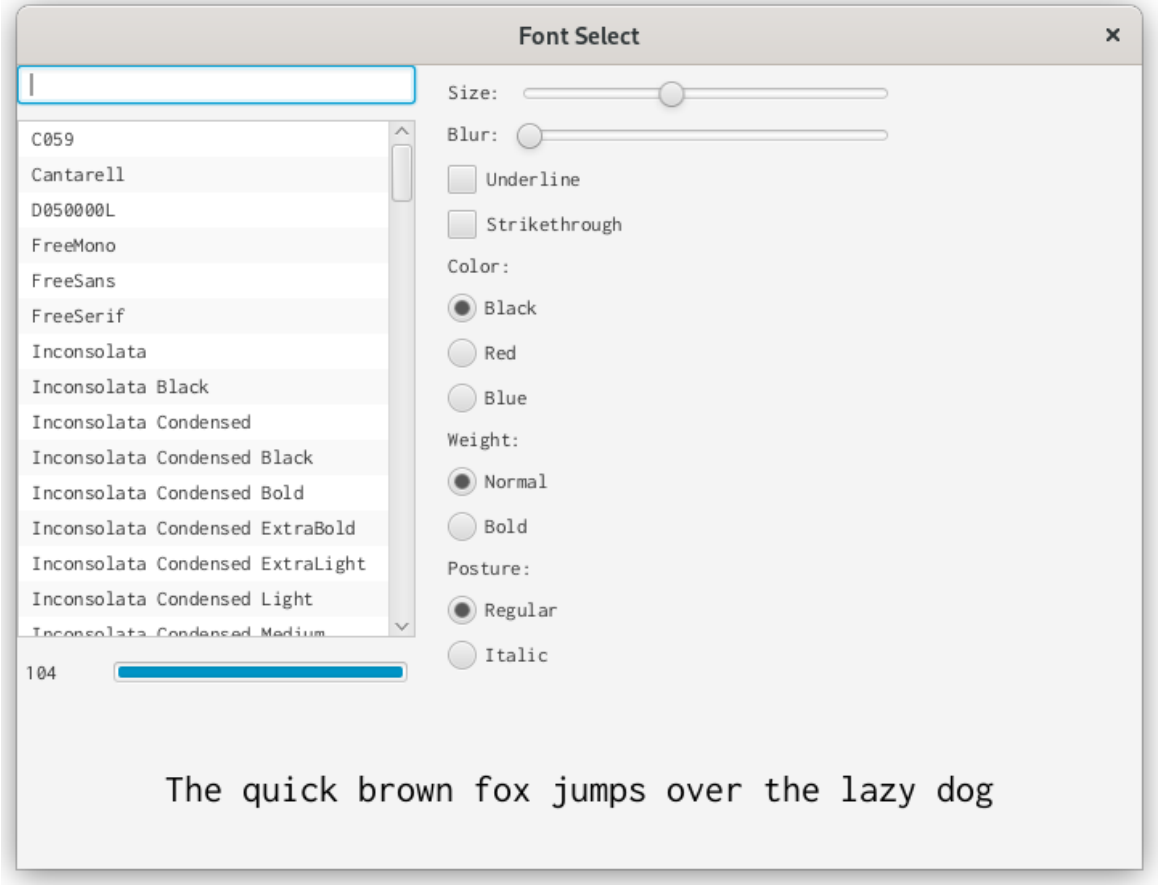
- Two approaches:
  - Listeners
  - bindBidirectional



ASSIGNMENT 3

Programming Task: FontSelect

- Implement a font select dialog
- The text below is to be displayed using the selected configuration
  - E.g. in the selected font, **bold**, *italic*, colored, etc.
- The TextField in the top-left corner is to act as a search bar
  - Filter the fonts in the list by name
- Use JavaFX *Listeners* and *Bindings* to achieve this



Assignment 03

- Deadline next Monday (22.04.24 23:59)
- Use PDFs for textual solutions!

- Export the programming exercise to a Zip File!
  - If the .zip file you send to our assignment system contains another .zip file you probably did everything right 😉