JavaOne™ San Francisco

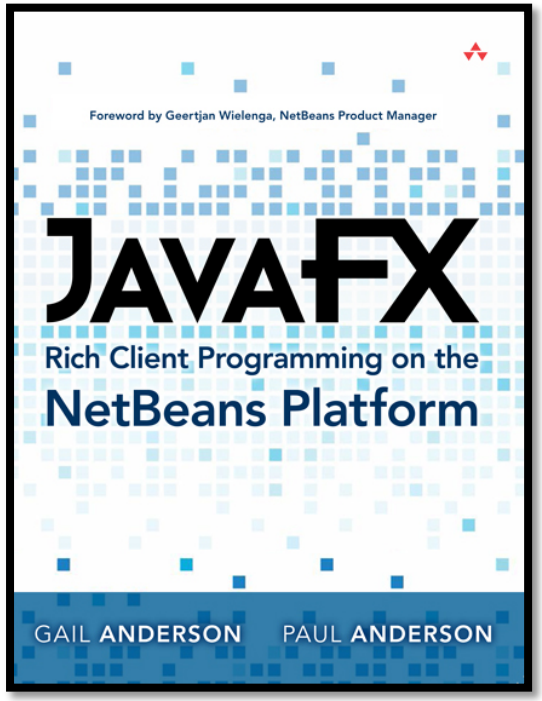
October 26, 2015

Are You Listening? JavaFX Binding Techniques for Rich Client UIs

Paul Anderson
Gail Anderson
Anderson Software Group, Inc.
asgteach.com

So Who Are We?

- Training Company
  - Java, JavaFX Courses
- JavaFX Authors
  - JavaFX Rich Client Programming on the NetBeans Platform
  - Essential JavaFX
- LiveLesson Videos
  - JavaFX Programming
  - Java Reflection



Agenda

- JavaFX BPMonitor Application
- JavaFX Properties with Model Data
- JavaFX Listeners and Binding
- JavaFX Observable Lists
- JavaFX Binding with Streams
- Trail of Listeners and Bindings
- JavaFX and Swing Integration
- Wrap Up, Q & A

BPMonitor Application

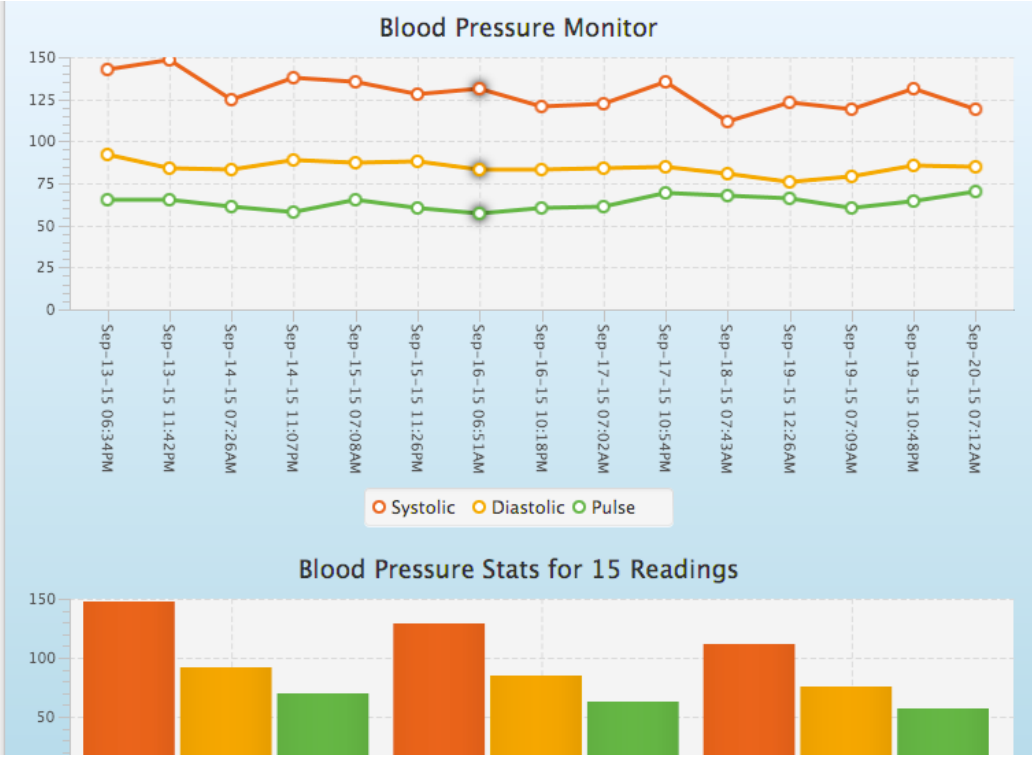
Systolic 131

Diastolic 83

Pulse 57

Add Update Delete

Date	Systolic	Diastolic	Pulse
Sep-13-15 06:34PM	143	92	65
Sep-13-15 11:42PM	148	84	65
Sep-14-15 07:26AM	125	83	61
Sep-14-15 11:07PM	138	89	58
Sep-15-15 07:08AM	135	87	65
Sep-15-15 11:26PM	128	88	60
Sep-16-15 06:51AM	131	83	57
Sep-16-15 10:18PM	121	83	60
Sep-17-15 07:02AM	122	84	61
Sep-17-15 10:54PM	135	85	69
Sep-18-15 07:43AM	112	81	68
Sep-19-15 12:26AM	123	76	66
Sep-19-15 07:09AM	119	79	60
Sep-19-15 10:48PM	131	86	64



BPMonitor Overview

- Observable Properties
  - Property Change Support
  - Listeners, Binding
- Observable Lists
  - BackingList, SortedList, FilteredList
  - Extractors
- Listeners
  - ChangeListener, InvalidationListener
  - ListChangeListener

Agenda

- JavaFX BPMonitor Application
- JavaFX Properties with Model Data
- JavaFX Listeners and Binding
- JavaFX Observable Lists
- JavaFX Binding with Streams
- Trail of Listeners and Bindings
- JavaFX and Swing Integration
- Wrap Up, Q & A

JavaFX Properties

- What are Properties?
  - Wraps field value, observable
  - Listeners notified when property updates
  - Used in binding expressions
- Supports
  - Read-write properties
  - Read-only properties
  - Immutable properties

Class Properties

- Abstract Classes

```
IntegerProperty     FloatProperty     DoubleProperty
BooleanProperty     LongProperty      StringProperty
ObjectProperty<T>   ListProperty<E>   SetProperty<E>
MapProperty<K,V>    ReadOnlyXXX
```

- Wrapper Classes

```
SimpleIntegerProperty     SimpleFloatProperty
SimpleDoubleProperty      SimpleBooleanProperty
SimpleLongProperty        SimpleStringProperty
SimpleObjectProperty<T>   SimpleListProperty<E>
SimpleMapProperty<K,V>    SimpleSetProperty<E>
```

Read-Write Property

- JavaFX Property

```
private final DoubleProperty data =
    new SimpleDoubleProperty(1.0);
```

- Naming Conventions

```
public final double getData()
   { return data.get(); }
public final void setData(double data)
   { this.data.set(data); }
public DoubleProperty dataProperty()
   { return data; }
```

Read-Only Property

- JavaFX Property

```
private final DoubleProperty data =
    new SimpleDoubleProperty(1.0);
```

- Naming Conventions

```
public final double getData()
   { return data.get(); }
private void setData(double data)
   { this.data.set(data); }
public ReadOnlyDoubleProperty dataProperty()
   { return data; }
```


Immutable Property

- Cannot Change
- JavaFX Field

```
private final double data = 1.0;
```

- Naming Conventions

```
public final double getData() {
   return data;
}
```


BPData Properties

```
public class BPData {
    public final static String SYSTOLIC_PROP_NAME = "systolic";
    public final static String DIASTOLIC_PROP_NAME = "diastolic";
    public final static String PULSE_PROP_NAME = "pulse";
    public final static String DATE_PROP_NAME = "date";

    private final ObjectProperty<LocalDateTime> date;
    private final IntegerProperty systolic;
    private final IntegerProperty diastolic;
    private final IntegerProperty pulse;

    private final static DateTimeFormatter formatter =
            DateTimeFormatter.ofPattern("MMM dd/hh:mma");

    public BPData() {
        date = new SimpleObjectProperty<>(this, DATE_PROP_NAME);
        systolic = new SimpleIntegerProperty(this, SYSTOLIC_PROP_NAME);
        diastolic = new SimpleIntegerProperty(this, DIASTOLIC_PROP_NAME);
        pulse = new SimpleIntegerProperty(this, PULSE_PROP_NAME);
    }
```


Agenda

- JavaFX BPMonitor Application
- JavaFX Properties with Model Data
- **JavaFX Listeners and Binding**
- JavaFX Observable Lists
- JavaFX Binding with Streams
- Trail of Listeners and Bindings
- JavaFX and Swing Integration
- Wrap Up, Q & A

JavaFX Listeners

- What is a Listener?
  - Listens for changes to a property
  - Can be added and removed
  - Listeners notified when property changes
- Invalidation Listener
  - When values become invalid
- Change Listener
  - When values update
  - Access to old and new values

Invalidation Listeners

- The `Observable` Interface

```
void addListener(InvalidationListener listener)
void removeListener(InvalidationListener listener)
```

- The `InvalidationListener` Interface

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


Invalidation Listeners

- Anonymous Class

```
myBean.wordProperty().addListener(
                    new InvalidationListener() {
   @Override
   public void invalidated(Observable observable) {
      // word is invalid...
   }
});
```

- Lambda Expression

```
myBean.wordProperty().addListener(observable -> {
    // word is invalid...
});
```


InvalidationListener

```
// Configure InvalidationListener for spinners, combobox, datepicker,
// and textfields
InvalidationListener listener = observable -> {
    if (!deleteBPButton.isDisabled()) {
        updateBPButton.setDisable(false);
    }
};

hourSpinner.valueProperty().addListener(listener);
minuteSpinner.valueProperty().addListener(listener);
ampmControl.valueProperty().addListener(listener);
systolicField.textProperty().addListener(listener);
diastolicField.textProperty().addListener(listener);
pulseField.textProperty().addListener(listener);
datePicker.valueProperty().addListener(listener);

// Add InvalidationListener to the startDate. When startDate
// changes, re-invoke filteredList's predicate
// (which depends on startDate)
startDate.valueProperty().addListener(observable -> {
        filteredList.setPredicate(data -> inRange(data.getDate()));
});
```


Change Listeners

- The `ObservableValue<T>` Interface

```
void addListener(
        ChangeListener<? super T> listener)
void removeListener(
        ChangeListener<? super T> listener)
T getValue()
```

- The `ChangeListener<T>` Interface

```
@FunctionalInterface
public interface ChangeListener<T> {
    void changed(ObservableValue<? extends T>
        observable, T oldValue, T newValue)
}
```


Change Listeners

- Anonymous Class

```
myBean.wordProperty().addListener(
                    new ChangeListener<String>() {
    @Override public
    void changed(ObservableValue<? Extends String>
      observable, String oldValue, String newValue) {
        // word has changed...
    }
});
```

- Lambda Expression

```
myBean.wordProperty().addListener(
    (observable, oldValue, newValue) -> {
        // word has changed...
});
```


ChangeListener

```
// Define a Changelistener for the table selectedIndexProperty,
// which changes when the user selects a different row.
// Although the selectedIndexProperty is ReadOnly, we can
// change it using the SelectionModel select() method (see the
// Line Chart nodes MOUSE_CLICKED event handler).
// This lets users manually select rows in the table in
// addition to the Line Chart selection UI defined.
// If the selectedIndexProperty is -1, the selection has been
// cleared by method clearFields() and we return
bpTable.getSelectionModel().selectedIndexProperty().addListener(
        (observable, oldValue, newValue) -> {
            System.out.println("row is selected = " +
                bpTable.getSelectionModel().getSelectedIndex() +
                    " old row = " + oldValue);
            // Remove drop shadow from previously selected Line Chart
            // points if necessary
            if (oldValue.intValue() >= 0 &&
                    oldValue.intValue() < sortedsysData.size()) {
                sortedsysData.get(oldValue.intValue())
                        .getNode().setEffect(null);
                sorteddiasData.get(oldValue.intValue())
                        .getNode().setEffect(null);
                sortedpulseData.get(oldValue.intValue())
                        .getNode().setEffect(null);
            }
```


JavaFX Binding

- What is Binding?
  - Calculates a value from sources
  - Sources are dependencies
  - Observes its dependencies for changes
  - Updates value automatically
- Why Use Binding?
  - Avoids writing listeners
  - More concise, less error prone
  - Keeps UI controls in sync with their model data

Binding Strategies

- Unidirectional
  - Updates property when dependent property changes
- Bidirectional
  - Property updates in either direction
- Fluent API and Factory Methods
  - Binds properties from libraries of binding expressions
- Custom Binding
  - Specifies property dependencies and compute values

Unidirectional Binding

- The `Property<T>` Interface

```
void bind(ObservableValue<? extends T> observable)
void unbind()
boolean isBound()
```

- Formats

```
targetProperty.bind(dependentProperty)
targetProperty.unbind()
```

- Binding
  - Dependency in one direction
  - Cannot explicitly update bound properties

Unidirectional Bind

```
    // FXML controls to manipulate Date range of the displayed data
    @FXML
    private Button firstButton;
    @FXML
    private Button prevButton;
    @FXML
    private Button nextButton;
    @FXML
    private Button lastButton;
    @FXML
    private Label toLabel;

    @Override
    public void initialize(URL url, ResourceBundle rb) {

        nextButton.disableProperty().bind(lastButton.disableProperty());
        prevButton.disableProperty().bind(firstButton.disableProperty());

        // Sets the charts series data to the sorted lists
        // so we don't have to worry about keeping the list sorted!
        systolicSeries.setData(sortedsysData);
        diastolicSeries.setData(sorteddiasData);
        pulseSeries.setData(sortedpulseData);
```


Bidirectional Binding

- The `Property<T>` Interface

```
void bindBidirectional(Property<T> property)
void unbindBidirectional(Property<T> property)
```

- Formats

```
property1.bindBidirectional(property2)
property1.unbindBidirectional(property2)
```

- Binding
  - Dependencies in both directions
  - May update bound properties

BindBidirectional

- Label Control and Bean Model
  - Bind `text` property of Label to Bean `word` property
- Example

```
label.textProperty().bindBidirectional(
    myBean.wordProperty())
```

- Behavior
  - If `text` property updates, `word` property changes
  - If `word` property updates, `text` property changes

Fluent API Binding

- Class **`BooleanExpression`**

```
and                 or                  not
isEqualTo           isNotEqualTo        asString
asObject            getValue            booleanExpression
```

- Interface **`NumberExpression`**

```
add                 subtract            multiply
divide              negate              isEqualTo
isNotEqualTo        lessThan            greaterThan
lessThanOrEqualTo   greaterThanOrEqualTo
asString
```


Fluent API Binding

- Class **StringExpression**

```
concat                  isNull                  isNotNull
isEmpty                 isNotEmpty              length
isEqualTo               isNotEqualTo            getValue
isEqualToIgnoreCase     isNotEqualToIgnoreCase
lessThan                lessThanOrEqualTo
greaterThan             greaterThanOrEqualTo
getValueSafe            stringExpression
```

- Class **ObjectExpression**

```
isNull                  isNotNull               getValue
isEqualTo               isNotEqualTo            asString
objectExpression
```


Fluent Binding

```
// FXML buttons to Add, Update, Delete BP readings
@FXML
private Button addBPButton;
@FXML
private Button updateBPButton;
@FXML
private Button deleteBPButton;

@Override
public void initialize(URL url, ResourceBundle rb) {
    // Sets the charts series data to the sorted lists
    // so we don't have to worry about keeping the list sorted!
    systolicSeries.setData(sortedsysData);
    diastolicSeries.setData(sorteddiasData);
    pulseSeries.setData(sortedpulseData);

    // Set up bindings with enabling/disabling the UI controls
    addBPButton.disableProperty().bind(
            systolicField.textProperty().isEmpty().or
                (diastolicField.textProperty().isEmpty()).or
                    (pulseField.textProperty().isEmpty()).or
                        (okToAdd.not()));
```


Factory Method Binding

- Interfaces `Binding<T>`, `NumberBinding`
  - Abstract Classes

`IntegerBinding`	`FloatBinding`	`DoubleBinding`
`BooleanBinding`	`LongBinding`	`StringBinding`
`ObjectBinding<T>`	`ListBinding<E>`	`SetBinding<E>`
`MapBinding<K,V>`		

- The `Bindings` Class
  - Creates bindings with factory methods
  - Overloaded methods handle primitive types
  - At least one argument must be observable

Static `Bindings` Methods

- Arithmetic and Numeric

```
add  subtract  multiply  divide
max  min  negate
```

- Relational and Logical

```
equal  notEqual  lessThan  lessThanOrEqual
equalIgnoreCase  notEqualIgnoreCase
greaterThan  greaterThanOrEqual and or not when
```

- Strings

```
concat  convert  format  length
```


Static `Bindings` Methods

- Bindings

  `createBooleanBinding` `createFloatBinding`
  `createDoubleBinding` `createIntegerBinding`
  `createLongBinding` `createStringBinding`
  `createObjectBinding` `bindContent` `unbindContent`
  `bindBidirectional` `bindContentBidirectional`
  `unbindBidirectional` `unbindContentBidirectional`

- Observable Lists

  `isEmpty` `isNotEmpty` `size` `valueAt`

- Observable Values

  `isNull` `isNotNull`

Factory Binding

```
// Bind the statChart's titleProperty to the size of the
// BP Readings list, so that the min,average, max data
// are based on the number of readings.
// Note that Bindings.size here takes an ObservableList,
// which is itself not a Property, but its size is observable
// with the Bindings.size() method.
// (We could also have used displayList with Bindings.size.)
statChart.titleProperty().bind(Bindings.concat(
        "Blood Pressure Stats for ",
            Bindings.size((systolicSeries.getData())).asString(),
                " Readings"));

// Configure range setting control buttons
startDate.disableProperty().bind(
        Bindings.equal(Bindings.size(sortedBackingList), 0));

// Initialize startDate so we can define binding for the Label
startDate.setValue(localDateTime.toLocalDate());
toLabel.textProperty().bind(Bindings.createStringBinding(() ->
        String.format("To:     %s",
            startDate.getValue().plusDays(daysWindow.get())
                .format(rangeFormatter)),
        startDate.valueProperty()));
```


Custom Binding

- What is Custom Binding?
  - Lower level technique
  - Direct extension method
  - Specify dependencies and compute values
  - Callback methods
- Why Use Custom Binding?
  - `Bindings` class or Fluent API doesn't apply
  - More flexible, better performance

Custom Binding Format

- Implementation

```
XXXBinding myBinding = new XXXBinding() {
    {  super.bind(dependencies); }
    @Override
    protected bindingType computeValue() { ... }
};
```

- Binding Classes

```
IntegerBinding      FloatBinding      DoubleBinding
BooleanBinding      LongBinding       StringBinding
ObjectBinding<T>    ListBinding<E>    SetBinding<E>
MapBinding<K,V>
```


Custom Binding

```
        // Disable firstButton if sortedBackingList is empty,
        // or if the date in the first element of the (non-empty) filteredList
        // is the same as the first element in the sortedBackingList.
        // This custom binding is dependend on both filteredList
        // and sortedBackingList
//        firstButton.disableProperty().bind(Bindings.createBooleanBinding(() ->
//                sortedBackingList.isEmpty() ||
//                    (!filteredList.isEmpty() &&
//                        inRange(sortedBackingList.get(0).getDate())),
//                            filteredList, sortedBackingList));

        //Custom binder version
        firstButton.disableProperty().bind(new BooleanBinding() {
           { super.bind(filteredList, sortedBackingList); }

           @Override
           protected boolean computeValue() {
               return (sortedBackingList.isEmpty() ||
                   (!filteredList.isEmpty() &&
                       inRange(filteredList.get(0).getDate())));
           }
        });
```


Agenda

- JavaFX BPMonitor Application
- JavaFX Properties with Model Data
- JavaFX Listeners and Binding
- JavaFX Observable Lists
- JavaFX Binding with Streams
- Trail of Listeners and Bindings
- JavaFX and Swing Integration
- Wrap Up, Q & A

JavaFX Observable Lists

- Interface `ObservableList<E>`
  - Holds data
  - Notifies listeners of changes
- Class `SortedList<E>`
  - Read-only collection
  - Wraps observable list, sorts content with comparator
- Class `FilteredList<E>`
  - Read-only collection
  - Wraps observable list, filters content with predicate

Observable Lists

```
// ObservableList holds all BPData objects
private final ObservableList<BPData> backingList =
        FXCollections.observableArrayList();

// Define sortedBackingList based on source backingList and
// sort criteria the date property.  Note that sortedBackingList
// wraps backingList and lets you access the data sorted
private final SortedList<BPData> sortedBackingList =
        new SortedList<>(backingList, (data1, data2) ->
                data1.getDate().compareTo(data2.getDate()));

// Define filteredList to filter sortedBackingList for TableView control.
// Initially the predicate is false and the list is empty.
// When the startDate date picker control is set, the predicate will
// be updated and the items selected
private final FilteredList<BPData> filteredList =
        new FilteredList<>(sortedBackingList, p -> false);

// Configure the Line Chart Series observable lists
private final ObservableList<Data<String, Number>> sysData =
        FXCollections.observableArrayList();
private final ObservableList<Data<String, Number>> diasData =
        FXCollections.observableArrayList();
private final ObservableList<Data<String, Number>> pulseData =
        FXCollections.observableArrayList();
```


Line Chart Sorted Lists

```
// Make the chart lists sorted.
// First, use our custom comparator to convert
// LocalDateTime to and from a String
private final XDateComparator xDateComparator = new XDateComparator();
// Then define each sorted list to use this customized comparator
private final SortedList<Data<String, Number>> sortedsysData =
        new SortedList<>(sysData, (data1, data2) ->
            xDateComparator.compare(data1.getXValue(), data2.getXValue()));
private final SortedList<Data<String, Number>> sorteddiasData =
        new SortedList<>(diasData, (data1, data2) ->
            xDateComparator.compare(data1.getXValue(), data2.getXValue()));
private final SortedList<Data<String, Number>> sortedpulseData =
        new SortedList<>(pulseData, (data1, data2) ->
            xDateComparator.compare(data1.getXValue(), data2.getXValue()));

// Custom comparator that compares String representations of LocalDateTime
final class XDateComparator implements Comparator<String> {
    @Override
    public int compare(String t1, String t2) {
        LocalDateTime d1 = LocalDateTime.parse(t1, formatter);
        LocalDateTime d2 = LocalDateTime.parse(t2, formatter);
        return d1.compareTo(d2);
    }
}
```


Observable List Interactions

- `SortedList<BPData>`, `FilteredList<BPData>`
  - Keeps TableView sorted and filtered

```
bpTable.setItems(filteredList);
```

- `SortedList<Data<String,Number>>`
  - Keeps Line Chart series sorted

```
systolicSeries.setData(sortedsysData);
diastolicSeries.setData(sorteddiasData);
pulseSeries.setData(sortedpulseData);
```


List Change Listeners

- The `ListChangeListener<E>` Interface

```
@FunctionalInterface
public interface ListChangeListener<E> {
    void onChanged(ListChangeListener.
        Change<? extends E> change)
}
```

- Nested Class `Change<E>`
  - `wasAdded()` – added to list
  - `wasRemoved()` – removed from list
  - `wasUpdated()` – updated in list
  - `wasPermutated()` – resorted in list

ListChangeListener

```
// Add ListChangeListener to the filteredList to keep the Line Chart
// synchronized with the TableView data
filteredList.addListener(
        (ListChangeListener.Change<? extends BPData> change) -> {
            while (change.next()) {
                if (change.wasRemoved()) {
                    System.out.println("Remove from Filtered List (" +
                        change.getRemovedSize() + " items)");
                    change.getRemoved().forEach(data ->
                        removeBPDataFromChart(change.getFrom()));
                }
                if (change.wasAdded()) {
                    System.out.println("Add to Filtered List (" +
                        change.getAddedSize() +
                            " items) at index " + change.getFrom());
                    change.getAddedSubList().forEach(data ->
                            addBPDataToChart(data));
                }
            }
});
```


JavaFX Extractors

- What's an Extractor?
  - Named properties in `Observable` array
- Lambda Implementation

```
FXCollections.observableArrayList(data ->
    new Observable[] { data.dateProperty() });
```

- Using Extractors
  - Enables `wasUpdated()` in `ListChangeListener`
  - Makes SortedList re-sort, FilteredList refilter
  - Handles specific property updates but adds overhead

Agenda

- JavaFX BPMonitor Application
- JavaFX Properties with Model Data
- JavaFX Listeners and Binding
- JavaFX Observable Lists
- **JavaFX Binding with Streams**
- Trail of Listeners and Bindings
- JavaFX and Swing Integration
- Wrap Up, Q & A

Binding with Streams

```
// Use binding to find max, average, min values for each observable list.
// In each of these bindings, sourceList is the binding dependency
private void setUpChartBindings(
        ObservableList<Data<String, Number>> sourceList,
        ObservableList<Data<String, Number>> targetList) {

    // Use custom binding to compute maximum
    targetList.get(0).YValueProperty().bind(new IntegerBinding() {
        { super.bind(sourceList); }      // dependency

        @Override
        protected int computeValue() {
            return sourceList.stream()
                .mapToInt(d -> d.getYValue().intValue())
                .max()
                .orElse(1);
        }
    });

    // Use Factory Bindings method to compute average
    targetList.get(1).YValueProperty().bind(
            Bindings.createDoubleBinding(() -> sourceList.stream()
                .mapToInt(d -> d.getYValue().intValue())
                .average()
                .orElse(1.0),
            sourceList));           // dependency
```


Agenda

- JavaFX BPMonitor Application
- JavaFX Properties with Model Data
- JavaFX Listeners and Binding
- JavaFX Observable Lists
- JavaFX Binding with Streams
- **Trail of Listeners and Bindings**
- JavaFX and Swing Integration
- Wrap Up, Q & A

Trail of Listeners and Bindings

- Add BP Reading
  - New BP reading added to BackingList
  - SortedList sorts reading by date
  - BP reading included in FilteredList if predicate is true
  - Add triggers FilteredList ListChangeListener
    - Addition to sorted Line Chart
    - Bar Chart bindings trigger
    - Updates to Bar Chart data
    - Bar Chart title changes to new size of Line Chart

Trail of Listeners and Bindings

- Delete BP Reading
  - Remove selected BP reading from BackingList
  - SortedList and FilteredList update from removal
  - Removal triggers FilteredList ListChangeListener
    - Removal from sorted Line Chart
    - Bar Chart bindings trigger
    - Updates to Bar Chart data
    - Bar Chart title changes to new size of Line Chart

Trail of Listeners and Bindings

- Update BP Reading
  - Remove and add BP reading to BackingList
  - SortedList comparator sorts added reading
  - FilteredList predicate checks date range
    - Add item to FilteredList if date in range
  - Remove/add trigger FilteredList ListChangeListener
    - Possible add to sorted Line Chart
    - Bar Chart bindings trigger
    - Updates to Bar Chart data
    - Bar Chart title set to size of Line Chart

Agenda

- JavaFX BPMonitor Application
- JavaFX Properties with Model Data
- JavaFX Listeners and Binding
- JavaFX Observable Lists
- JavaFX Binding with Streams
- Trail of Listeners and Bindings
- **JavaFX and Swing Integration**
- Wrap Up, Q & A

NetBeans Platform

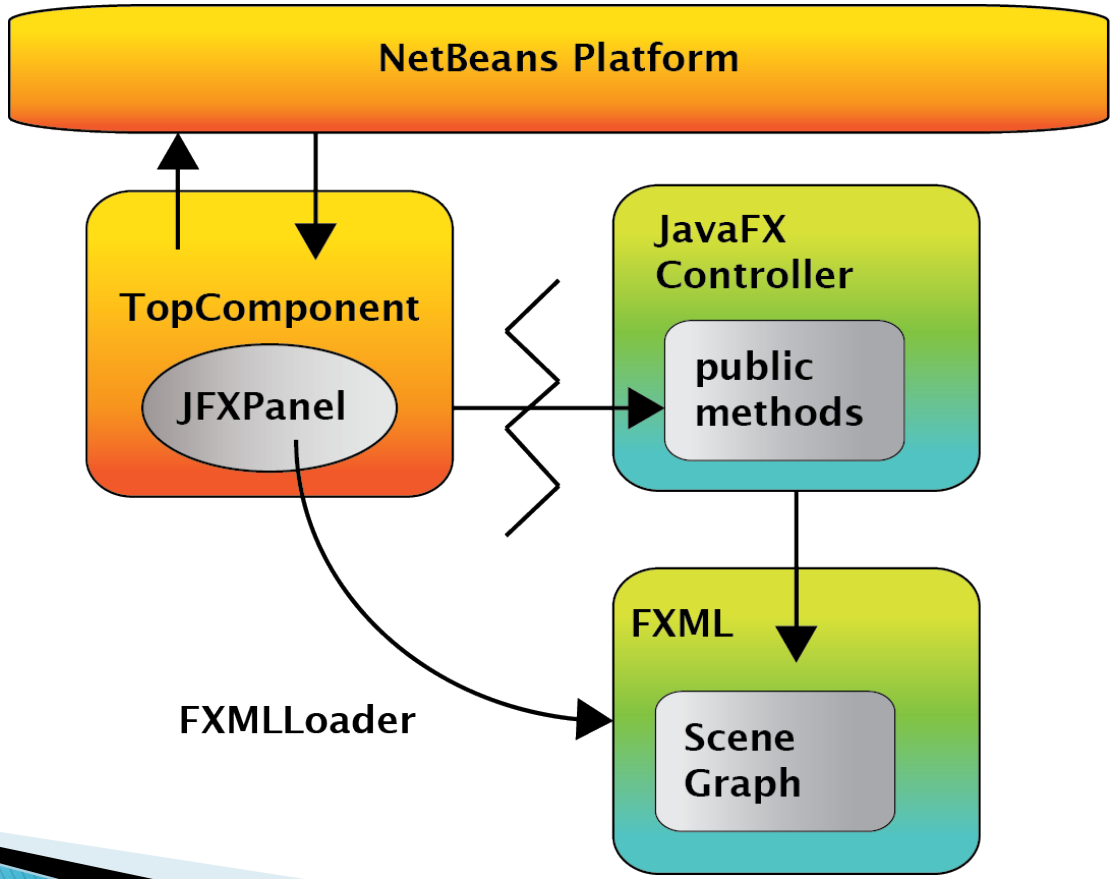
- Swing Based Framework
  - Windows and services
  - TopComponents
  - Detachable, resizable windows
  - Toolbar, Menu bar
  - Divided into modules
  - Lookup to find services
  - Lookup for selection and communication

JavaFX and Swing

- The Magic of **`JFXPanel`**
  - Package **`javafx.embed.swing`**
  - Embed JavaFX content into Swing apps
  - Creating **`JFXPanel`** instance starts JavaFX runtime
  - Input/focus events forwarded to scene
- Example

```
JFXPanel fxPanel = new JFXPanel();
add(fxPanel, BorderLayout.CENTER);
```

JavaFX in TopComponents



Thread Issues

- Swing
  - Create and update on Event Dispatch Thread (EDT)
  - Single-threaded UI
- JavaFX
  - Create and update on JavaFX Application Thread
  - Single-threaded UI
  - Separate thread from EDT

Update Swing from JavaFX

- Must access Swing Thread (EDT)
  - Wrap code in `Runnable` Object
  - Use lambdas to reduce boilerplate code

```
SwingUtilities.invokeLater(() -> mySwingMethod());
```


Update JavaFX from Swing

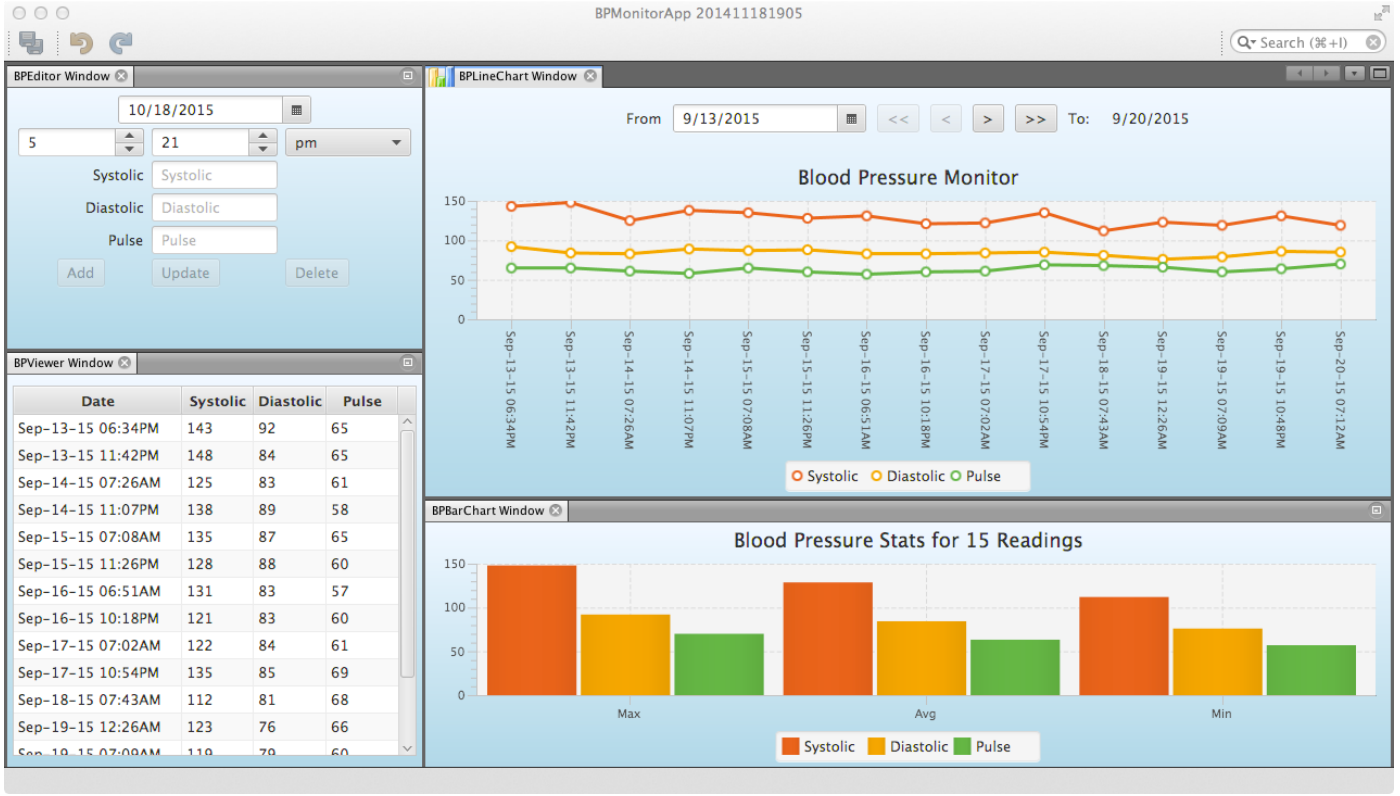
- Must access JavaFX Application Thread
  - Wrap code in `Runnable` Object
  - Use lambdas to reduce boilerplate code

```
Platform.runLater(() -> myJavaFXMethod());
```


Communication Strategies

- Self-contained JavaFX scene
  - All events occur within JavaFX scene
- One-way communication
  - TopComponent initiates change in JavaFX scene
  - Use `Platform.runLater()`
- One-way with synchronization
  - Use `CountDownLatch` to wait
- Two-way communication
  - Typically use `PropertyChangeEvent` to communicate from JavaFX to listening TopComponent code

BPMonitor NetBeans Platform



Summary

- JavaFX
  - Properties, Listeners, Binding
  - Observable Lists, Sorted Lists, Filtered Lists
  - Binding and Listeners with Streams
- JavaFX Swing Integration
  - Using `JFXPanel`
  - NetBeans Platform Framework
  - Thread issues
  - Communication Strategies

Wrap Up

- Thanks for Coming!
  - paul@asgteach.com @paul_asgteach
  - gail@asgteach.com @gail_asgteach
- Source Code
  - asgteach.com
    - JavaOne2015 Tutorial Examples
      - Click to Download
  - Book Give Away
  - Q & A