San Diego Java Users Group

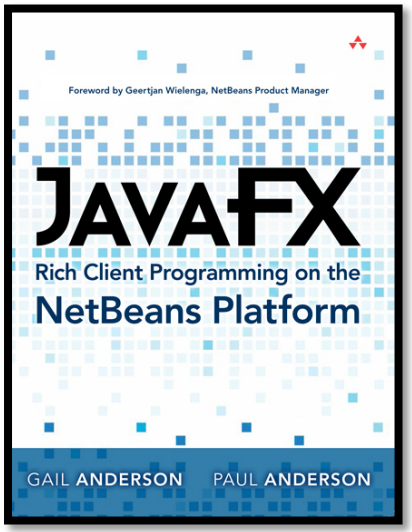
April 16, 2019

JavaFX Now and Beyond
Desktop to Mobile

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
- LiveLesson Videos
  - JavaFX Programming
  - Java Reflection



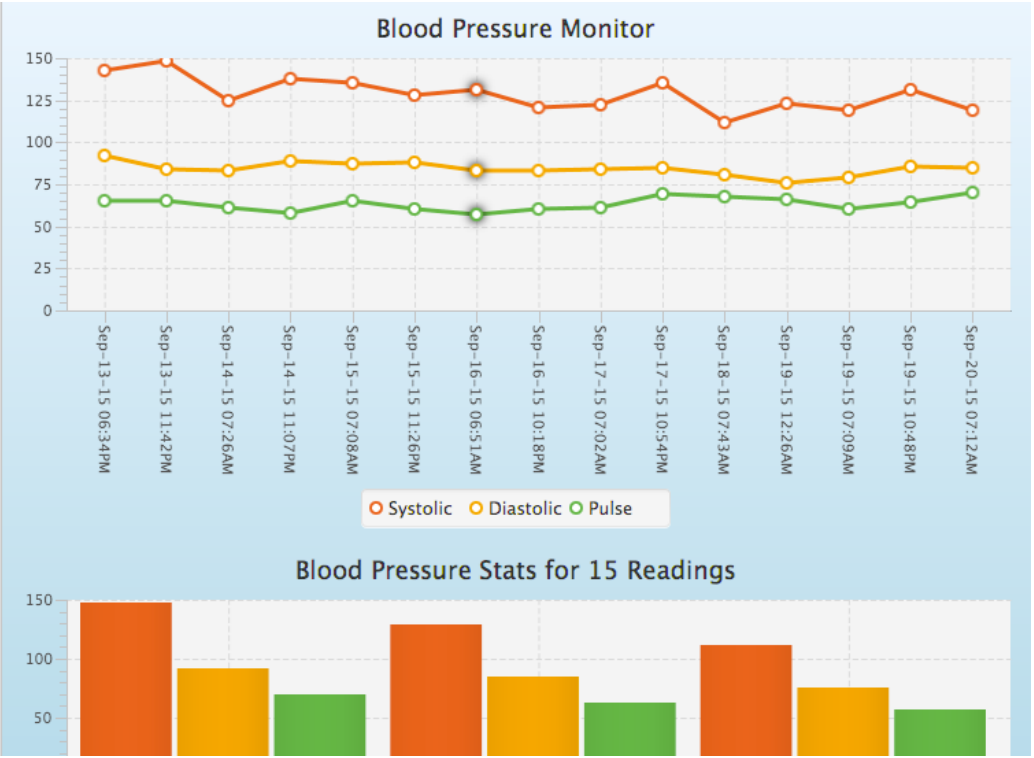
Agenda

- JavaFX BPMonitor Application
- JavaFX Properties with Model Data
- JavaFX Listeners
- JavaFX Binding Strategies
- JavaFX Observable Lists
- JavaFX Binding with Streams

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



JavaFX Properties

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

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);
    }
```

JavaFX Listeners

- Why Uses Listeners?
  - 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

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()));
});
```

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);
            }
```

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, compute values

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);
```


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

- What is Fluent API Binding?
  - Library of instance methods
  - Can be chained with Builder Pattern
- Binding Categories
  - `BooleanExpression`
  - `NumberExpression`
  - `StringExpression`
  - `ObjectExpression`

Fluent API 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

- What is Factory Method Binding?
  - Library of static methods
  - Invoked with `Bindings` class
  - At least one argument must be `Observable`
- Binding Categories
  - Arithmetic, Numeric
  - Relational and Logical
  - Strings, Creationals
  - Observable Lists and Values

Factory Method 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

```
        // 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())));
           }
        });
```


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 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();
```


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
```

