JavaFX Datasources:

Getting Real-World Data into JavaFX Controls

Johan Vos - LodgON
Jonathan Giles - Oracle

JavaOne 2011

Disclaimer

Disclaimer #1:

This is **not** an Oracle talk – this is a talk by Johan and Jonathan on a project that we do in our own time. This may or may not have any relationship to future Oracle products.

Disclaimer #2:

THE FOLLOWING IS INTENDED TO OUTLINE OUR GENERAL PRODUCT DIRECTION. IT IS INTENDED FOR INFORMATION PURPOSES ONLY, AND MAY NOT BE INCORPORATED INTO ANY CONTRACT. IT IS NOT A COMMITMENT TO DELIVER ANY MATERIAL, CODE, OR FUNCTIONALITY, AND SHOULD NOT BE RELIED UPON IN MAKING PURCHASING DECISION. THE DEVELOPMENT, RELEASE, AND TIMING OF ANY FEATURES OR FUNCTIONALITY DESCRIBED FOR ORACLE'S PRODUCTS REMAINS AT THE SOLE DISCRETION OF ORACLE.

The Problem

Manager: "Bob, our users are complaining we aren't showing them enough information"

Bob: *internalises anguished scream, already knowing what the next sentence is going to be*

Manager: "We need you to add a table into our app that shows all the information that is missing"

Bob: *facepalm*

Problem Statement

- Getting data into a Table can often be difficult:
  - Bringing data into memory
  - Massaging data
  - Loading data into table efficiently (maybe on-demand)
  - Displaying the data can be tricky
  - Allowing editing of data (with proper bindings to the data source) can be complex
- Hint: We think we have some code in this talk that may help…

Outline

- Intro to JavaFX 2.0 TableView / ListView controls
- Retrieving/parsing/rendering data
- Mapping data with controls
- Static and dynamic data
- Cell factories
- ‘EnergyCo’ Example

JavaFX 2.0 Controls

- Primary focus of this talk: ListView and TableView

- Common API:
  - Both have a generic type
  - 'items' ObservableList: the raw data.
  - 'cell factory' support: the renderer of the raw data

- JavaFX Controls leverage generics, e.g.
  - ListView<T>
  - TableView<T>
  - TreeView<T>
- With <T> the type of the objects contained within the control.

Creating a ListView Instance

```
// Get the data that we want to show in the ListView
ObservableList<Person> people = …

// Create a ListView control
ListView<Person> peopleList = new ListView<Person>();

// Set the data in the items list (or just pass in to the constructor)
peopleList.setItems(people);
```

Creating a TableView Instance

```
// Get the data that we want to show in the TableView
ObservableList<Person> people = …

// Create a TableView control
ListView<Person> peopleList = new ListView<Person>();

// Set the data in the items list (or just pass in to the constructor)
peopleList.setItems(people);

// Create TableColumn instances for each column we want
TableColumn<Person, String> firstNameCol = new TableColumn<>(“First”);

// Tell the TableColumn how to extract the value from the row object
firstNameCol.setCellValueFactory(…);
```

What Is A Cell Value Factory?

- Each TableColumn needs a Cell Value Factory
  - Otherwise the column will be blank
- It tells the TableColumn how to extract a value for a cell from a single item from the TableView.items list
- It is a Callback<CellDataFeatures<S,T>, ObservableValue<T>>
  - S is the type of the TableView
  - T is the type of the TableColumn

Creating Cell Value Factories

```
Callback<CellDataFeatures<Person, String>, ObservableValue<String>> callback =
    new Callback<>() {
        public ObservableValue<String> call(CellDataFeatures<Person, String> p) {
            // Note: We are returning an ObservableValue, not the value itself!
            return p.getValue().lastNameProperty();
        }
    };

firstNameCol.setCellValueFactory(callback);
```

Alternatively…

```
firstNameCol.setCellValueFactory(new PropertyValueFactory(“firstName”));

// PropertyValueFactory uses reflection to attempt to find either a
// firstNameProperty() method that returns the correct type, and if that
// fails, it will attempt to return getFirstName() / isFirstName().

// Of course, getFirstName() returns a String, not an
// ObservableValue<String>…

// So, we wrap using a ReadOnlyObjectWrapper
// You can do this too if your objects aren’t JavaFX classes
// The downside is that the UI won’t update dynamically
```

Visualization of data

- Controls can visualize
  - Different kinds of data (ListView<T>)
  - From different sources (file, database, network)
  - In different formats (XML,CSV,JSON)
  - With different dynamic capabilities (static versus dynamic)

Examples of datasources

In-memory arrays:

```
String[][] people = new String[][] {
        {"Jonathan", "Giles", "jonathan.giles@oracle.com"},
        {"Johan", "Vos", "johan@lodgon.com"}
    };
```

CSV Data:

- [show example of large file]

XML Resource:

- [show online XML resource]

Retrieving/parsing/rendering

- Retrieving, parsing and rendering data are three different tasks.
- These tasks are domain-agnostic
- JavaFX Controls provide a way to render data
- JavaFX DataSources project provides a way to retrieve and parse data by providing abstractions for a number of sources and formats

Retrieving and parsing

- Retrieving data:
  - File, local/remote database, network resources
- Parsing data:
  - XML, JSON, CSV, Java Objects

Retrieving Data

- Retrieving data:
  - File, local/remote database, network resources
- Javafx.datasource.io package contains a number of different (fysical) sources.
- DataSource classes do not know what parts of the data are needed
  - a JDBC based resource contains many columns, but only 2 of those should be rendered in a TableView
  - An XML resource contains structured information, but not all elements/attributes should be rendered

Mapping Data with Controls

- Javafx.datasource.control package contains wrappers that gather the application-specific relevant portions of the data and provide those to the JavaFX Controls
  - Column headers match the CSV headers
  - Column headers can be mapped with XML elements or attributes (Xpath)
  - Column headers can be mapped with database column names
  - ...

Static versus Dynamic

- Some data elements are static
- Some data elements are dynamic
  - Update requested by control (pull)
  - Update requested by datasource (push)

RedFX

- Library that allows remote synchronization of data
- If the value of a data element changes on one client, or in a backend component, the same element is updated on all other clients
- Using Binding, the visualized data is updated immediately

Examples

- Code + demo shown for
  - ArrayDataSource
  - CSVDataSource
  - JDBCDataSource
  - XmlFileDataSource
  - XmlRESTDataSource
  - JSONRESTDataSource
  - RedFXDataSource

Cell Factories

- Common API across ListView, TreeView, TableView
- Responsible for displaying a single row/cell in a control
  - ListView: One cell per row
  - TableView: One cell per row + one cell for each column

- Default cells call toString() on the cell.item object
- Custom cells can do anything…

Creating a Cell Factory

```
// todo
todo
```

Common Cell Factories

- Ranked in order of important (imho):
  - Text editing
  - Check box
  - Drop-down menu
  - Context-specific rendering (red for negative, green for positive)
  - Images
  - Progress bar
  - Expand-on-click
  - Everything else…

Common Cell Factories

- Ranked in order of importance:
  - Text editing
  - Check box
  - Drop-down menu
  - Context-specific rendering (red for negative, green for positive)
  - Images
  - Progress bar
  - Expand-on-click
  - Everything else…

EnergyCo Demo

JavaOne 2011

What's Next?

- Current implementation is read-only.
- Write support can be imagined simply as list write operations.

- Support for TreeView API
- Support for Charts API

- More data sources
- More cell factories
- More convenience

Where Can I Get This?

- This is a free, open source project built on top of JavaFX 2.0
- It is **not** an Oracle-sponsored project
- BSD-ish license (we are not lawyers) – contact us if you're concerned
- It can be downloaded right now:
  http://javafxdata.jonathangiles.net

Thanks

Johan Vos johan@lodgon.com
Jonathan Giles jonathan.giles@oracle.com