# Data Binding

**Scott Violet**

Swing Architect
Sun Microsystems
http://www.sun.com

TS-1594

## Goal

Understand data binding, and how it will make your life easier

## Agenda

**What Is Data Binding?**

Beans Binding

Crazy Faces Revisited

Database Driven Application

Summary

## Disclaimer

- This is a preview of a prototype
- The details will almost certainly change

## Data Binding

Makes Your Life Easier!

- Simplifies keeping two objects in sync
  - No longer need to know about TableModel, TreeModel, Document...
- Examples
  - ResultSet to JTable
  - DataSet to JTable
  - Property of a POJO to a JTextField
  - List<POJO> to JList

## Data Binding (Cont.)

- Typically used to connect an application model to a UI component
- May be used to connect any two objects together
  - Often handy to connect two UI objects together
- Ability to transform values
  - String to Color, Date to String
- May include validation
- Useful for any application!

## DEMO

Crazy Faces

## JCaricature

- Configured via standard bean property methods
  `setEyeStyle/getEyeStyle, setHairStyle/getHairStyle...`
- Changing a property notifies registered PropertyChangeListeners

## JCaricature

```
public void setEyeStyle(int style) {
    int oldStyle = eyeStyle;
    eyeStyle = style;
    firePropertyChange("eyeStyle", oldStyle, style);
    repaint();
}

public int getEyeStyle() {
    return eyeStyle;
}
```

## Crazy Faces

### Without Data Binding

- To track changes to JSlider, Controller installs ChangeListener on JSlider
  - Changes propagated to JCaricature
- To track changes to JCaricature, Controller installs PropertyChangeListener on JCaricature
  - Changes propagated back to JSlider

## Controller

### Listening for Changes

```
eyeSlider.addChangeListener(new ChangeListener() {
  public void stateChanged(ChangeEvent e) {
    caricature.setEyeStyle(eyesSlider.getValue());
  }
});

caricature.addPropertyChangeListener(new
    PropertyChangeListener() {
  public void propertyChange(PropertyChangeEvent e) {
    if (e.getPropertyName() == “eyeStyle”) {
      eyeSlider.setValue(caricature.getEyeStyle());
    }
  }
});
```

## Crazy Faces Summary

- Controller listens for changes on UI Component, invokes method on JCaricature
- Controller listens for changes on JCaricature, invokes method on UI Component

## Crazy Faces Summary

- Controller listens for changes on UI Component, invokes method on JCaricature
- Controller listens for changes on JCaricature, invokes method on UI Component
- This code is painful, and nearly the same in all apps!

## Agenda

What Is Data Binding?

**Beans Binding**

Crazy Faces Revisited

Database Driven Application

Summary

## Beans Binding

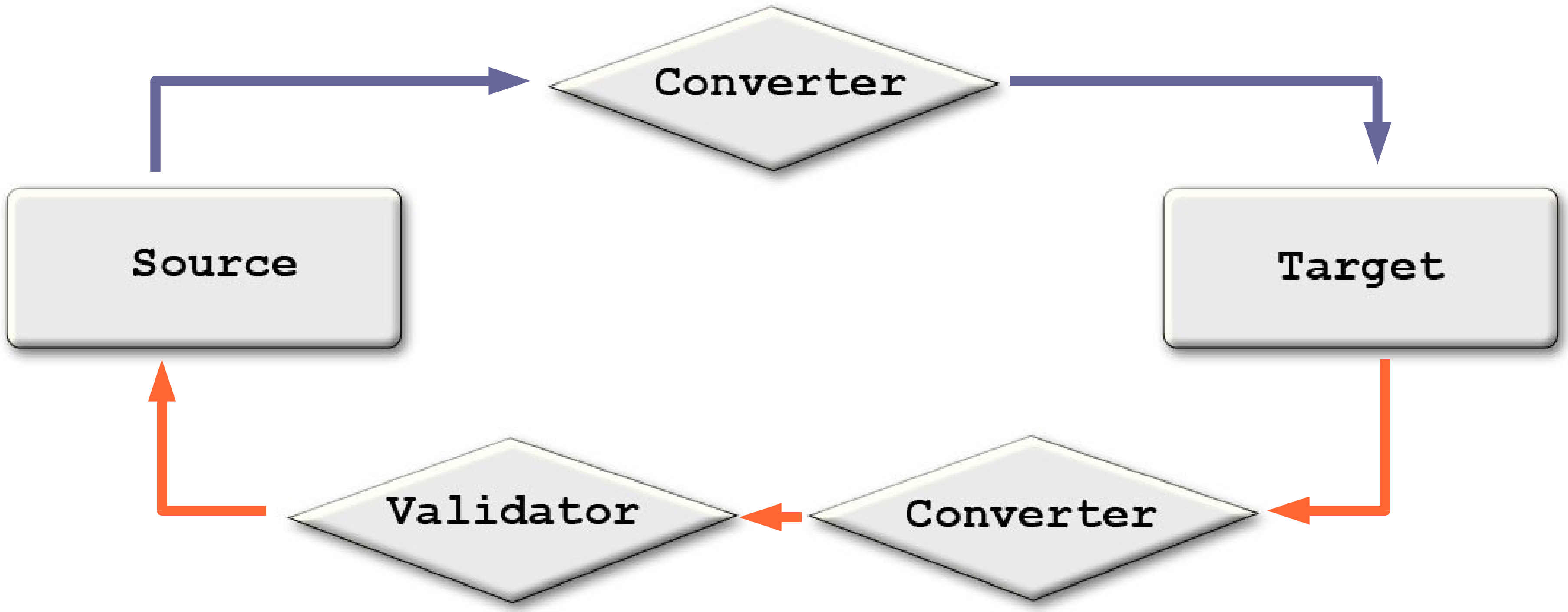
- New Java™ Specification Request (JSR)
  - Just passed inception ballot
- Targeted at Dolphin, but delivered standalone as well
- NetBeans™ software will support it
- Keeps two properties of two objects in sync
  - Source and target as Objects
  - PropertyChangeListener used to listen for changes

## Beans Binding

- Will accommodate objects that don’t strictly follow beans pattern
  - Map treated as beans with dynamic properties
  - Will accommodate objects that don’t strictly follow beans pattern (Swing)
- Ability to specify different update strategies
  - Read once, read only from source, keep source and target in sync
- Ability to do validation as property changes
- Ability to transform value
  - String to Color, Date to String

# Beans Binding

## Data Flow



## Beans Binding Builds Upon...

- Beans
  - Standard way to track changes to a property
    - PropertyChangeListener
- Collection classes
  - Standard way to encapsulate common data types

## Beans and Collection Classes

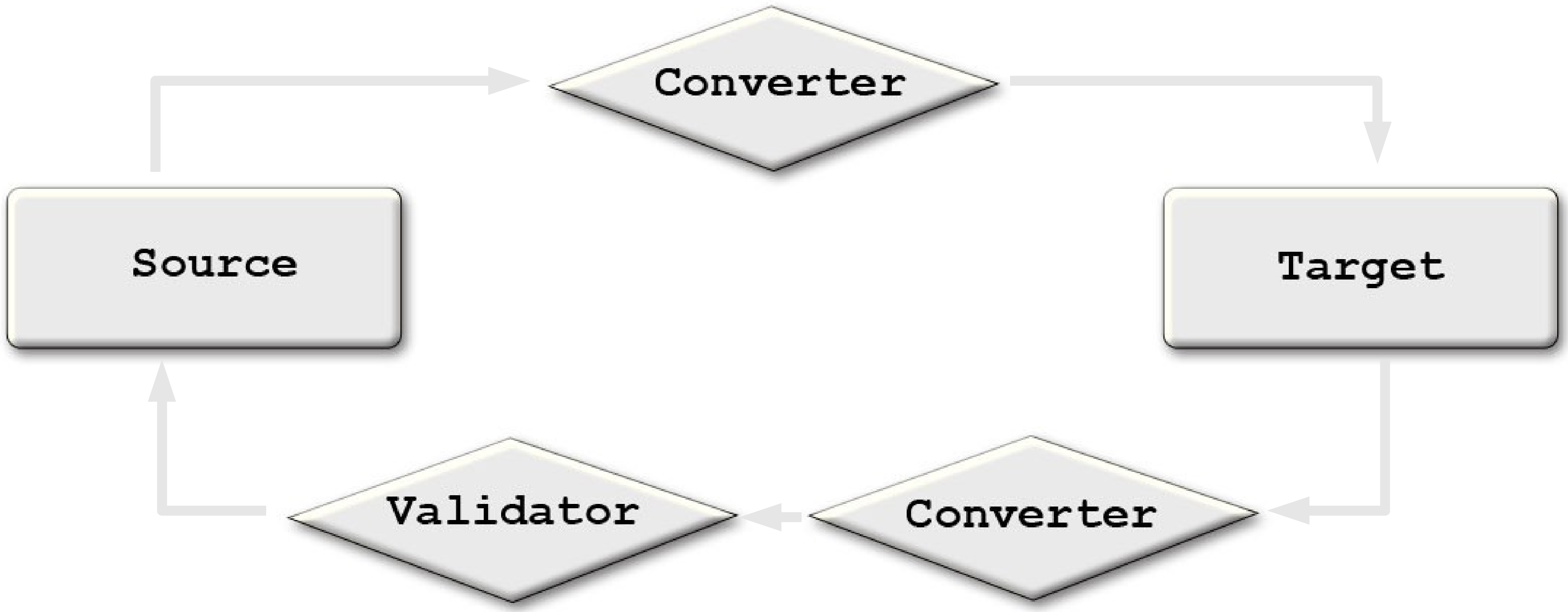
### Limitations

- Not all Swing components can be configured through properties
  - No JList.getElements
  - JList.getSelectedValues is not bound
  - ...
- No way to listen for changes to collection classes

## BindingDescription

- Describes a binding between a pair of properties
  - Source, target, source path, target path
- Converter
  - Ability to convert values from source or target
- Validator
  - Validates changes from the target
- Update strategy
  - How the two properties are kept in sync

## BindingDescription



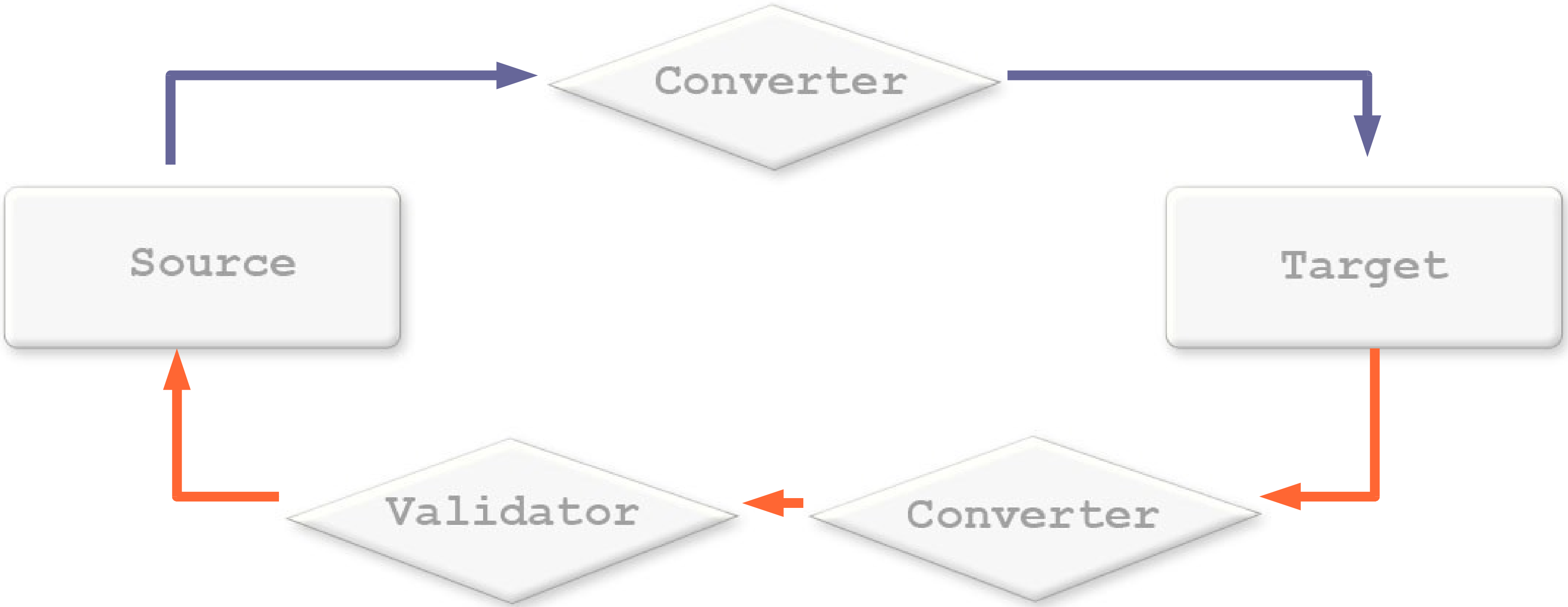
## ListBindingDescription

- Describes the binding between elements of a source and target list
- Typically used to define the binding between a source java.util.List and JTable, JList or JTree
- Subclass of BindingDescription that contains a List<BindingDescription>

## Binding

- Represents an active binding
- Created from a BindingDescription
- Maintains necessary listeners on source and target
- Maintains state of source and target values
  - Valid, Invalid, Newer,…
- Methods to update source and target values

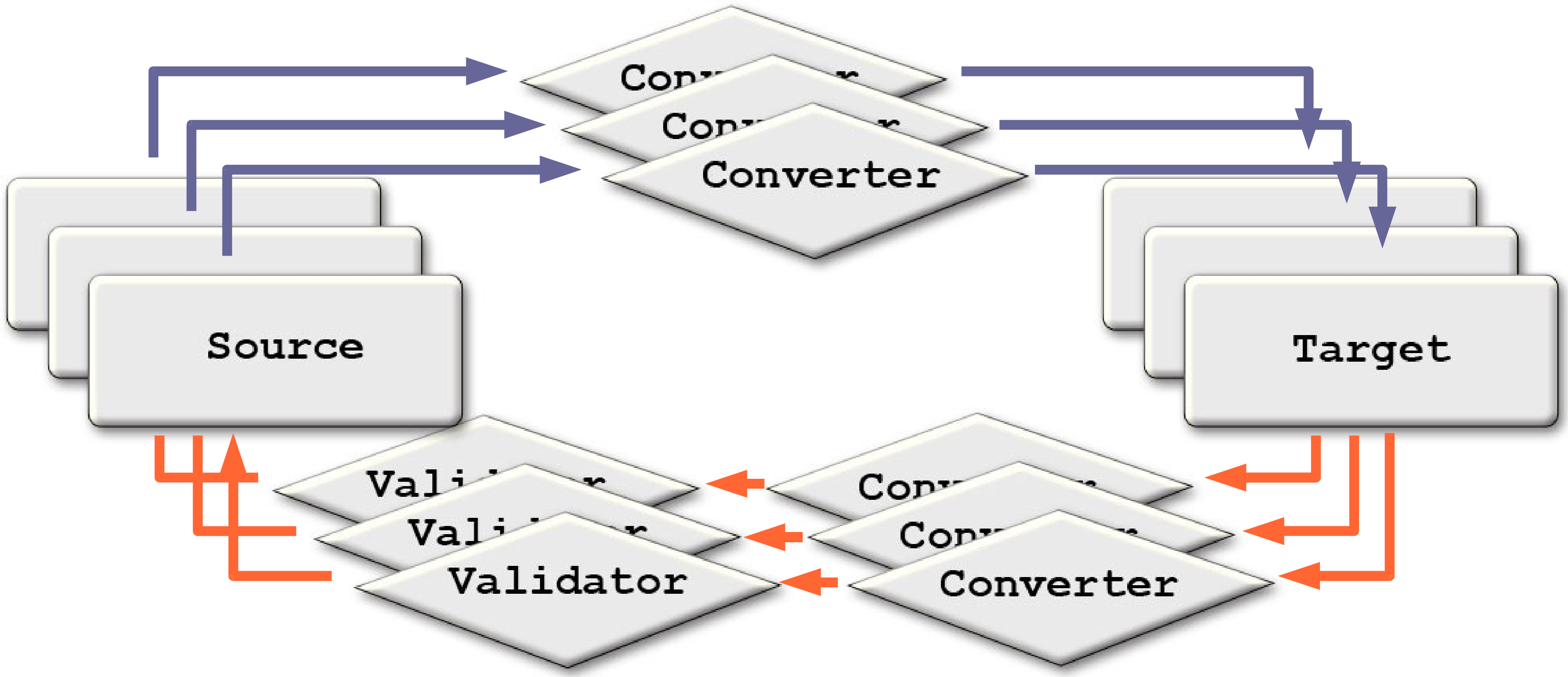
## Binding



## BindingContext

- Contains a Set<Binding> and Set<BindingDescription>
- Methods and listener to track state of all Bindings
  - Invalid, newer,...
- Single point to bind and unbind

# BindingContext



## Agenda

What Is Data Binding?

Beans Binding

**Crazy Faces Revisited**

Database Driven Application

Summary

## Controller

### Without Data Binding

```
eyeSlider.addChangeListener(new ChangeListener() {
  public void stateChanged(ChangeEvent e) {
    caricature.setEyeStyle(eyesSlider.getValue());
  }
});

caricature.addPropertyChangeListener(new
PropertyChangeListener() {
  public void propertyChange(PropertyChangeEvent e) {
    if (e.getPropertyName() == "eyeStyle") {
      eyeSlider.setValue(caricature.getEyeStyle());
    }
  }
});
```

## Controller

### With Data Binding

```
BindingContext context = new BindingContext();
BindingDescription bd = new BindingDescription(
    caricature, "eyeStyle", // Source, Source Path
    eyesSlider, "value");   // Target, Target Path
context.addDescription(bd);
context.bind();
```

## Scale JSlider

- JSlider operates on integer coordinates (BoundedRangeModel)
- JCaricature's scale property is floating
- Need to use a converter to connect them

## Scale JSlider

```
class ScaleConverter extends BindingConverter {
  public Object convertToTarget(BindingDescription d,
                 Object value) {
    return (int) ((Float) value * 100f);
  }
  public Object convertToSource(BindingDescription d,
                 Object value) {
    return (float) ((Integer) value) / 100.0f;
  }
}

BindingDescription bd = new BindingDescription(
  caricature, "scale", scaleSlider, "value");
bd.setConverter(new ScaleConverter());
```

# Crazy Faces

## Without Data Binding (83 Lines)

```
public class NoBindingCaricatureController extends CaricatureController {
    NoBindingCaricatureController() {
        eyesSlider.addChangeListener(new EyesChangeHandler());
        faceSlider.addChangeListener(new FaceChangeHandler());
        mouthSlider.addChangeListener(new MouthChangeHandler());
        hairSlider.addChangeListener(new HairChangeHandler());
        noseSlider.addChangeListener(new NoseChangeHandler());
        scaleSlider.addChangeListener(new ScaleChangeHandler());
        rotationSlider.addChangeListener(new RotationChangeHandler());
        caricature.addPropertyChangeListener(new CaricaturePropertyChangeHandler());
    }


    private class CaricaturePropertyChangeHandler implements PropertyChangeListener {
        public void propertyChange(PropertyChangeEvent e) {
            String name = e.getPropertyName();
            if (name == "eyeStyle") {
                eyesSlider.setValue(caricature.getEyeStyle());
            } else if (name == "faceStyle") {
                faceSlider.setValue(caricature.getFaceStyle());
            } else if (name == "mouthStyle") {
                mouthSlider.setValue(caricature.getMouthStyle());
            } else if (name == "hairStyle") {
                hairSlider.setValue(caricature.getHairStyle());
            } else if (name == "noseStyle") {
                noseSlider.setValue(caricature.getNoseStyle());
            } else if (name == "scale") {
                int sliderValue = (int)((caricature.getScale() - 1f) * 100f) + 50;
                scaleSlider.setValue(sliderValue);
            } else if (name == "rotation") {
                rotationSlider.setValue(caricature.getRotation());
            }
        }
    }
}
```

```
private class EyesChangeHandler implements ChangeListener {
    public void stateChanged(ChangeEvent e) {
        caricature.setEyeStyle(eyesSlider.getValue());
    }
}

private class FaceChangeHandler implements ChangeListener {
    public void stateChanged(ChangeEvent e) {
        caricature.setFaceStyle(faceSlider.getValue());
    }
}

private class MouthChangeHandler implements ChangeListener {
    public void stateChanged(ChangeEvent e) {
        caricature.setMouthStyle(mouthSlider.getValue());
    }
}

private class HairChangeHandler implements ChangeListener {
    public void stateChanged(ChangeEvent e) {
        caricature.setHairStyle(hairSlider.getValue());
    }
}

private class NoseChangeHandler implements ChangeListener {
    public void stateChanged(ChangeEvent e) {
        caricature.setNoseStyle(noseSlider.getValue());
    }
}

private class ScaleChangeHandler implements ChangeListener {
    public void stateChanged(ChangeEvent e) {
        float scale = (float)(scaleSlider.getValue() - 50) /
             100.0f + 1.0f;
        caricature.setScale(scale);
    }
}

private class RotationChangeHandler implements ChangeListener
    public void stateChanged(ChangeEvent e) {
        caricature.setRotation(rotationSlider.getValue());
    }
}
                                                                    }
```

## Crazy Faces

### With Data Binding (28 Lines)

```
public BindingCaricatureController() {
    BindingContext context = new BindingContext();
    context.addDescription(new BindingDescription(caricature, "eyeStyle", eyesSlider, "value"));
    context.addDescription(new BindingDescription(caricature, "faceStyle", faceSlider, "value"));
    context.addDescription(new BindingDescription(caricature, "mouthStyle", mouthSlider, "value"));
    context.addDescription(new BindingDescription(caricature, "hairStyle", hairSlider, "value"));
    context.addDescription(new BindingDescription(caricature, "noseStyle", noseSlider, "value"));
    context.addDescription(new BindingDescription(caricature, "rotation", rotationSlider, "value"));
    BindingDescription scaleDescription = new BindingDescription(
            caricature, "scale", scaleSlider, "value");
    scaleDescription.setConverter(new ScaleConverter());
    context.addDescription(scaleDescription);
    context.bind();
}


private static class ScaleConverter extends BindingConverter {
    public Object convertToTarget(BindingDescription description,
            Object value) {
        return (int)((Float)value * 100f);
    }

    public Object convertToSource(BindingDescription description,
            Object value) {
        return (float)((Integer)value) / 100.0f;
    }
}
```

## Agenda

What Is Data Binding?

Beans Binding

Crazy Faces Revisited

**Database Driven Application**

Summary

## Database Access Technologies

- Hibernate
  - Persists classes; just works with beans binding
- Mustang platform’s DataSet
  - Returns classes; just works with beans binding
- Enterprise JavaBeans™ (EJB™) 3 specification
  - Persists classes; just works with beans binding
- ResultSet/RowSet
  - Does not return classes
  - Requires a mapping layer to work with beans binding

# DEMO

Address Book

## Address Book

- Uses JDBC™ software (ResultSet) to connect to database
- Displays ResultSet in a JTable
- Details for selected element shown in JTextFields

## Binding to JTable

- Specify the List<T>, each T corresponds to a row
- Specify how the value for each column is obtained
- ListBindingDescription used to bind the java.util.List to the JTable
- BindingDescription used to specify how the value for a particular column is obtained

# Binding to JTable

## In an Ideal World

```
// Bindings the row elements of the table to the contents
// of the RowSet
bind(rowSet, table, "elements");
// Specifies the first column should be called 'First
// Name' with a value coming from firstName of the RowSet
bind("firstName", table, "First Name.value");
// Specifies the second column should be called
// 'Last Name' with a value coming from lastName
// of the RowSet
bind("lastName", table, "Last Name.value");
```

## Binding to JTable

### The Truth

- JTable doesn’t know about ResultSet
  - ResultSet is not a List<T>
- Have to create a List<Map<Object,Object>> around ResultSet
  - Treating each row as a Map enables beans binding to obtain appropriate value
- Mustang’s DataSet provides a type-safe view
  - Will just work with beans binding

## Binding to JTable

### Working Code

```
ResultSetHelper helper = new ResultSetHelper(resultSet);
List<Map<Object,Object>> asList = helper.getContentsAsList();
// Sets the contents of the table (rows) to the contents
// of the List
ListBindingDescription tableBD =
    new ListBindingDescription(asList, table, “elements”);
// Specifies how the values for the first two columns
// are extracted
tableBD.add(“firstName”, “0.value”);
tableBD.add(“lastName”, “1.value”);
```

## What Is JTable's “Elements”?

- JTable does not have an “elements” property
- JTable has a TableModel

## PropertyDelegate

### Enables JTable to Have an “elements” Property

- Enables an Object to have properties specific to binding
  - Will be used to add properties to Swing classes
- Registered with Class and property name
- Developer using binding can then bind to additional properties
  - JList.setElements()
  - JTable.setElements()

# Tracking Changes to Collections

- Currently no way to track changes to List or Map
  - Needed for dynamic displays
- Will add the ability to track changes to a List and Map
  - May be done as part of maintenance JSR for Dolphin
- Provide factory methods for creating observable variants wrapping your own
  - Just like Collections.unmodifiableList()

## ObservableListListener

```
public void listElementsAdded(
  ObservableList source, int index, int length);

public void listElementsRemoved(
  ObservableList source, int index, List oldElements);

public void listElementReplaced(
  ObservableList source, int index, Object oldElement);

public void listElementPropertyChanged(
  ObservableList source, int index);
```

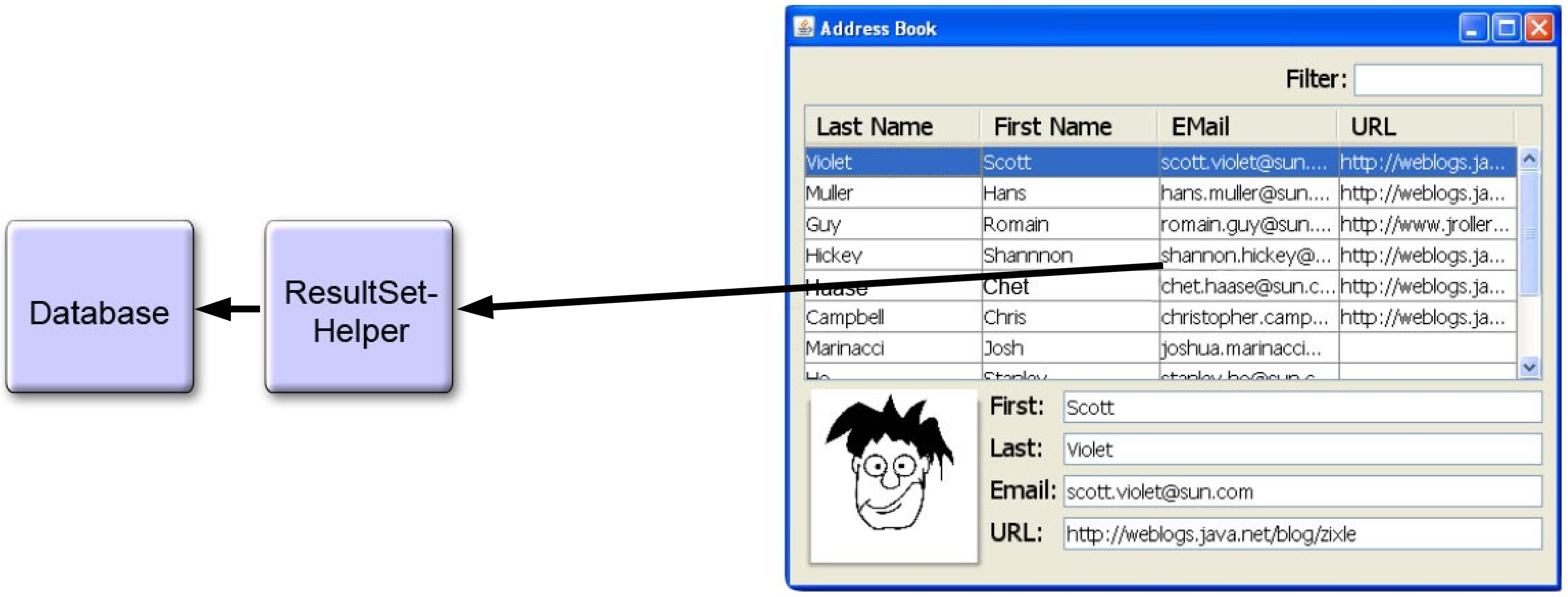
## ObservableMapListener

```
void mapKeyValueChanged(ObservableMap source,
    Object key, Object lastValue, Object newValue);

void mapKeyAdded(ObservableMap source,
    Object key, Object value);

void mapKeyRemoved(ObservableMap source,
    Object key, Object value);
```

## ResultSetHelper

- Creates a List<Map<Object,Object>> from a ResultSet
- Returned List is an ObservableList
  - Methods for mutating List notify all ObservableListListeners
  - Enables JTable to update as rows added or deleted
- Each Map is an ObservableMap
  - Changes to map notify both ObservableListListeners and ObservableMapListeners
- Provides method to get all modified entries
  - Makes it easy to persist any changes back to db

# JTable Binding



Address Book

Filter:

| Last Name | First Name | EMail | URL |
|---|---|---|---|
| Violet | Scott | scott.violet@sun.... | http://weblogs.ja... |
| Muller | Hans | hans.muller@sun.... | http://weblogs.ja... |
| Guy | Romain | romain.guy@sun.... | http://www.jroller... |
| Hickey | Shannnon | shannon.hickey@... | http://weblogs.ja... |
| Haase | Chet | chet.haase@sun.c... | http://weblogs.ja... |
| Campbell | Chris | christopher.camp... | http://weblogs.ja... |
| Marinacci | Josh | joshua.marinacci... | |
| Ho | Stanley | stanley.ho@sun.c | |

First: Scott
Last: Violet
Email: scott.violet@sun.com
URL: http://weblogs.java.net/blog/zixle

## Master/Detail Views

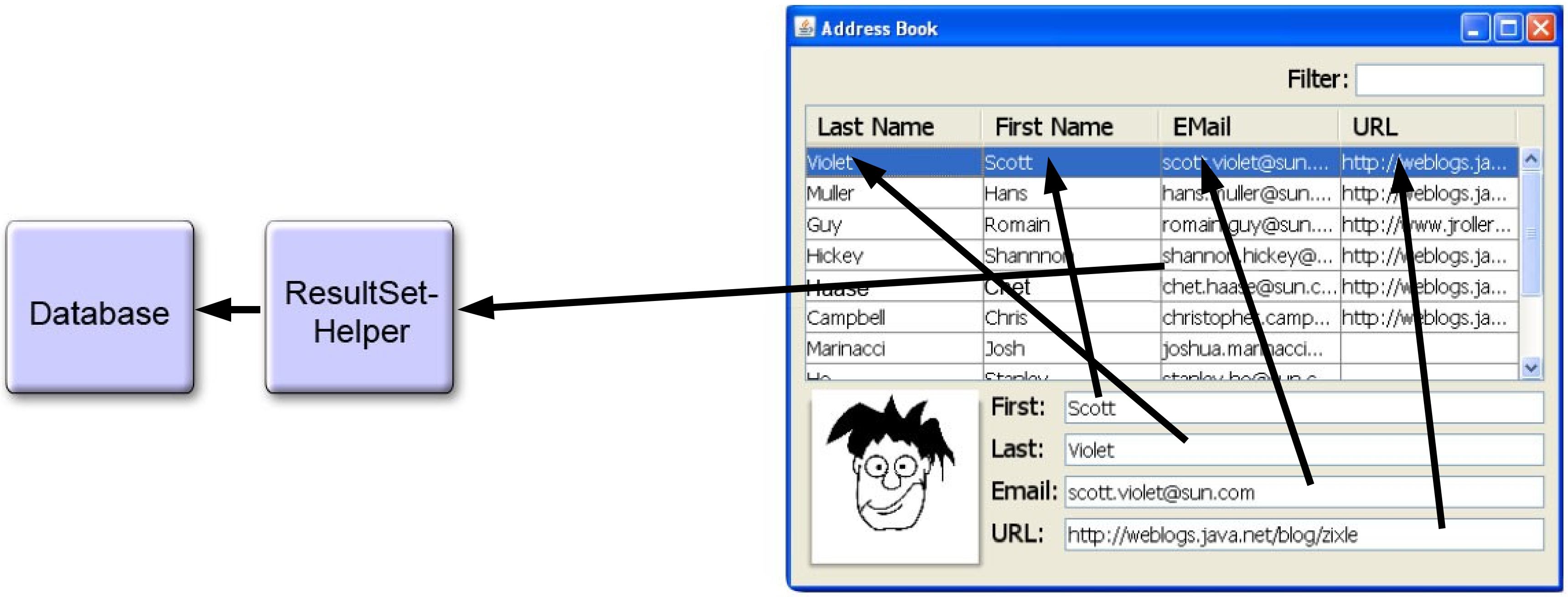
### Binding to the Selected Element

- As JTable’s selection changes, the “selectedElement” property changes
    - “selectedElement” is provided by JTable’s PropertyDelegate
- Detail components are bound to a property of “selectedElement”
    - “selectedElement” corresponds to a Map<Object,Object>

## Detail Bindings

```
BindingDescription bd = new BindingDescription(
    table,                                // Source
    “selectedElement.firstName”,          // Source Path
    firstNameTF,                          // Target
    “text”);                              // Target Path
```

# Detail Bindings



Filter:

| Last Name | First Name | EMail | URL |
|---|---|---|---|
| Violet | Scott | scott.violet@sun.... | http://weblogs.ja... |
| Muller | Hans | hans.muller@sun.... | http://weblogs.ja... |
| Guy | Romain | romain.guy@sun.... | http://www.jroller... |
| Hickey | Shannon | shannon.hickey@... | http://weblogs.ja... |
| Haase | Chet | chet.haase@sun.c... | http://weblogs.ja... |
| Campbell | Chris | christopher.camp... | http://weblogs.ja... |
| Marinacci | Josh | joshua.marinacci... | |
| Ho | Stanley | stanley.ho@sun.c | |

| | |
|---|---|
| First: | Scott |
| Last: | Violet |
| Email: | scott.violet@sun.com |
| URL: | http://weblogs.java.net/blog/zixle |

## ListController

- Previous example bound detail components to JTable's "selectedElement" property
  - To change selection, application code talks to JTable
  - To obtain selection, application code talks to JTable
- ListController
  - Has an elements property of type List
  - Has a selection property of type List
  - Has a selected element property
- Using ListController allows you to change view, without effecting application code

## Binding to JTable
### Working Code (Repeated)

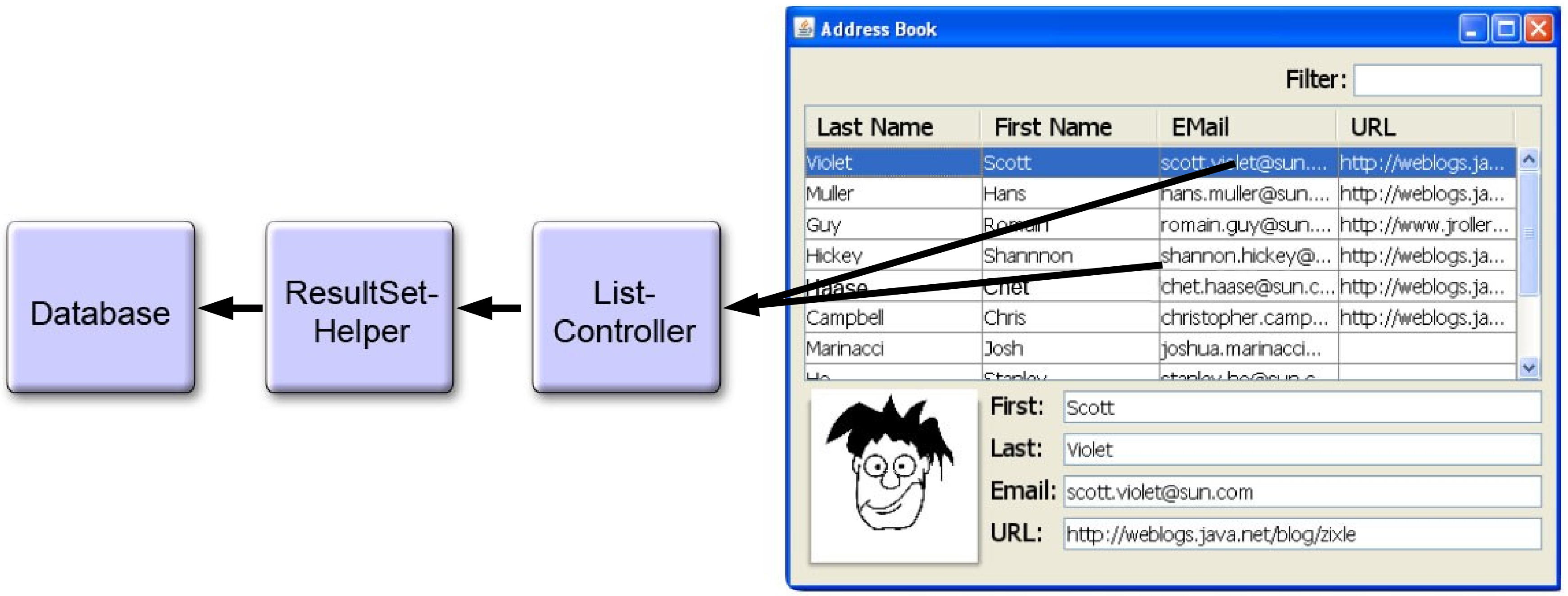
```
ResultSetHelper helper = new ResultSetHelper(resultSet);
List<Map<Object,Object>> asList = helper.getContentsAsList();
// Sets the contents of the table (rows) to the contents
// of the List
ListBindingDescription tableBD =
    new ListBindingDescription(asList, table, “elements”);
// Specifies how the values for the first two columns
// are extracted
tableBD.add(“firstName”, “0.value”);
tableBD.add(“lastName”, “1.value”);
```

## Binding to JTable

### With ListController

```
ResultSetHelper helper = new ResultSetHelper(resultSet);
List<Map<Object,Object>> asList = helper.getContentsAsList();
ListController<Map<Object,Object>> controller =
  new ListController<Map<Object,Object>>(asList);
// Sets the contents of the table (rows) to the contents
// of the List
ListBindingDescription tableBD =
    new ListBindingDescription(
        controller, “elements”, table, “elements”);
// Specifies how the values for the first two columns
// are extracted
tableBD.add(“firstName”, “0.value”);
tableBD.add(“lastName”, “1.value”);
```

# Binding JTable to ListController



| Last Name | First Name | EMail | URL |
|---|---|---|---|
| Violet | Scott | scott.violet@sun.... | http://weblogs.ja... |
| Muller | Hans | hans.muller@sun.... | http://weblogs.ja... |
| Guy | Romain | romain.guy@sun.... | http://www.jroller... |
| Hickey | Shannon | shannon.hickey@... | http://weblogs.ja... |
| Haase | Chet | chet.haase@sun.c... | http://weblogs.ja... |
| Campbell | Chris | christopher.camp... | http://weblogs.ja... |
| Marinacci | Josh | joshua.marinacci... | |

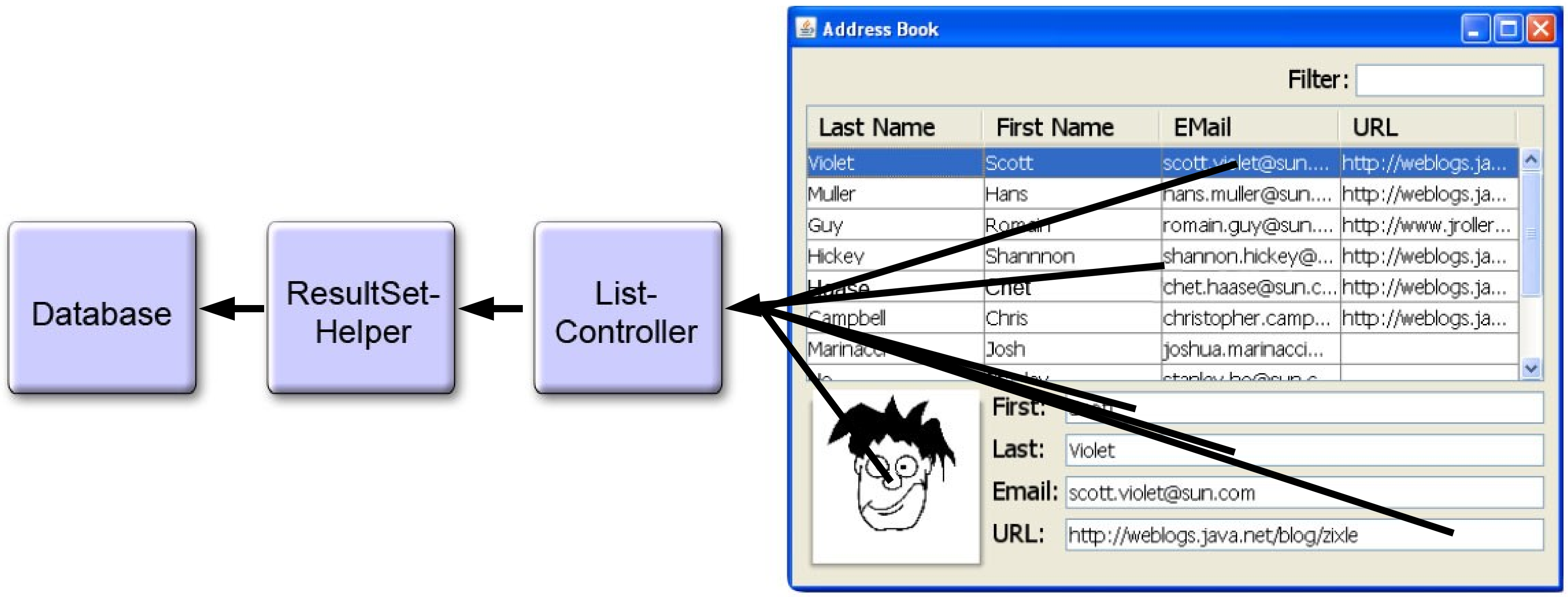
First: Scott
Last: Violet
Email: scott.violet@sun.com
URL: http://weblogs.java.net/blog/zixle

## Detail Bindings

### With ListController

```
BindingDescription bd = new BindingDescription(
  controller, “selectedElement.firstName”, // Source
  firstNameTF, “text”);                    // Target
```

# Detail Bindings with ListController



Filter:

| Last Name | First Name | EMail | URL |
|---|---|---|---|
| Violet | Scott | scott.violet@sun.... | http://weblogs.ja... |
| Muller | Hans | hans.muller@sun.... | http://weblogs.ja... |
| Guy | Romain | romain.guy@sun.... | http://www.jroller... |
| Hickey | Shannnon | shannon.hickey@... | http://weblogs.ja... |
| Haase | Chet | chet.haase@sun.c... | http://weblogs.ja... |
| Campbell | Chris | christopher.camp... | http://weblogs.ja... |
| Marinacci | Josh | joshua.marinacci... | |
| Ho | Stanley | stanley.ho@sun.c | |

First: Scott
Last: Violet
Email: scott.violet@sun.com
URL: http://weblogs.java.net/blog/zixle

## Application Code

### Interacting with ListController

```
// To obtain selection
List<Map<Object,Object>> selection =
    controller.getSelectedElements();



// To change selection
controller.setSelectedElements(null);
```

## DEMO

Address Book with JList

## Address Book Summary

- ResultSetHelper used to create a List<Map<Object,Object>> from a ResultSet
- ListController maintains selection and elements
- JTable's "elements" bound to ListController
  - ListBindingDescription specified for "elements"
  - Child BindingDescription for each column
- Detail components bound to ListController's selection
- Binding to Swing components with Beans Binding will just work with Mustang's DataSet, EJB 3 specification Persistence, and Hibernate

## Agenda

What Is Data Binding?

Beans Binding

Crazy Faces Revisited

Database Driven Application

**Summary**

## Summary

- Beans Binding will make binding your application model to Swing components trivial
    - Or binding between any two Objects
- Beans Binding is in its infancy
    - API covered here is a prototype, it will change

## For More Information

- Beans Binding (JSR 295): http://jcp.org/en/jsr/detail?id=295
- Related Sessions
    - TS-4635: Best Practices: Data Access Strategies (Thursday, 11:00AM)
    - TS-1074: Desktop Patterns and Data Binding (Thursday, 1:30PM)
    - TS-3399: A Simple Framework for Desktop Applications (Thursday, 4:00PM)

# Q&A

# Data Binding

**Scott Violet**

Swing Architect
Sun Microsystems
http://www.sun.com

TS-1594