Beans Binding
JSR 295

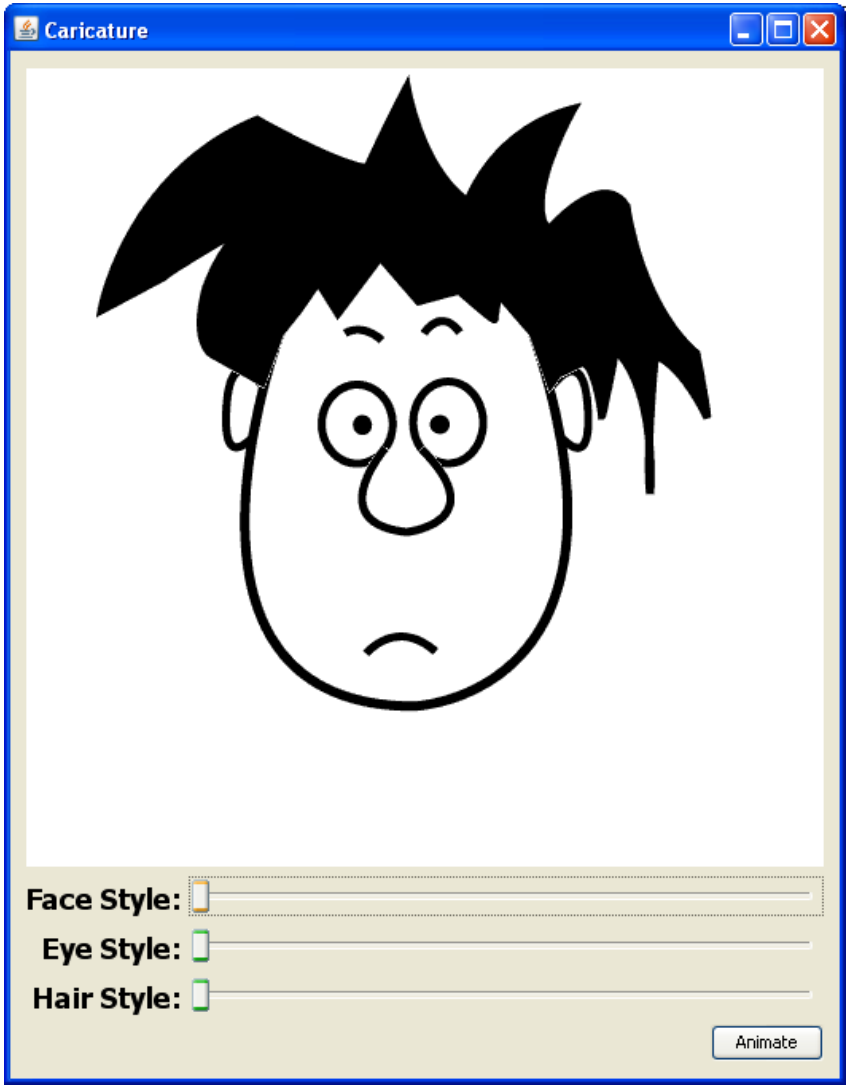
Shannon Hickey (Shannon.Hickey@Sun.COM)
Hans Muller (Hans.Muller@Sun.COM)

Caveat

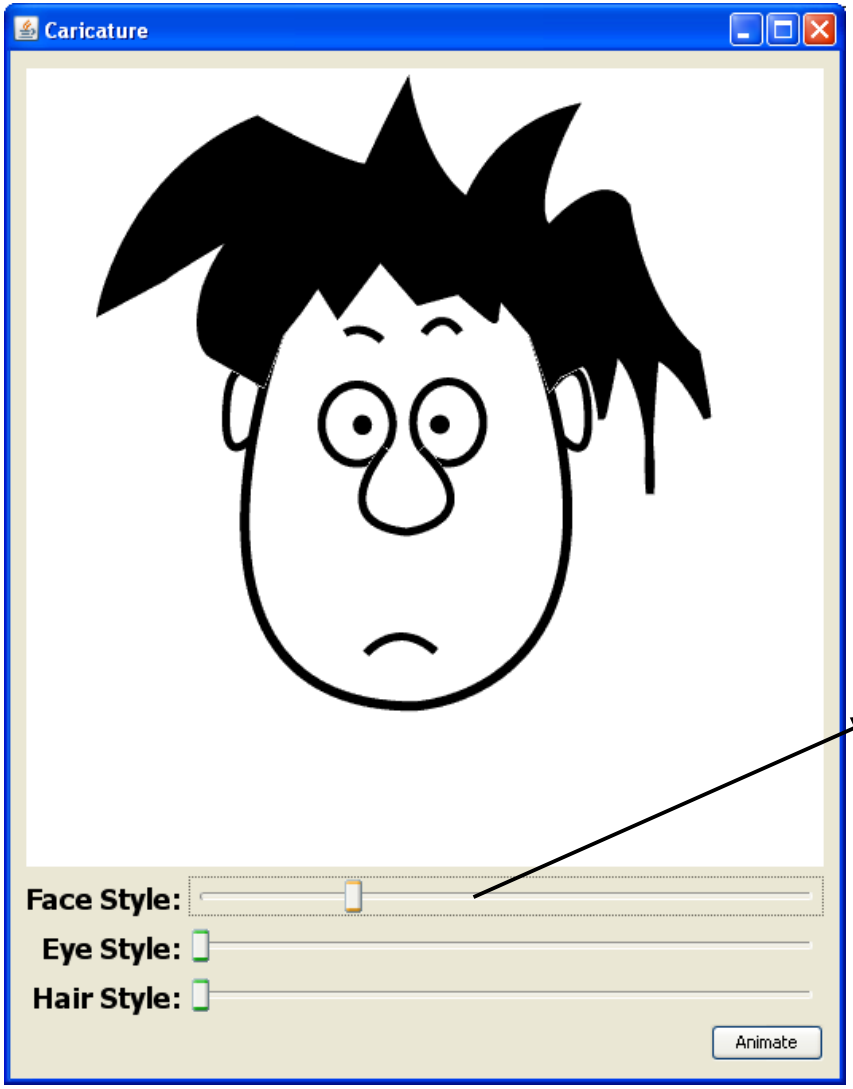
Beans Binding is not final!

Demo

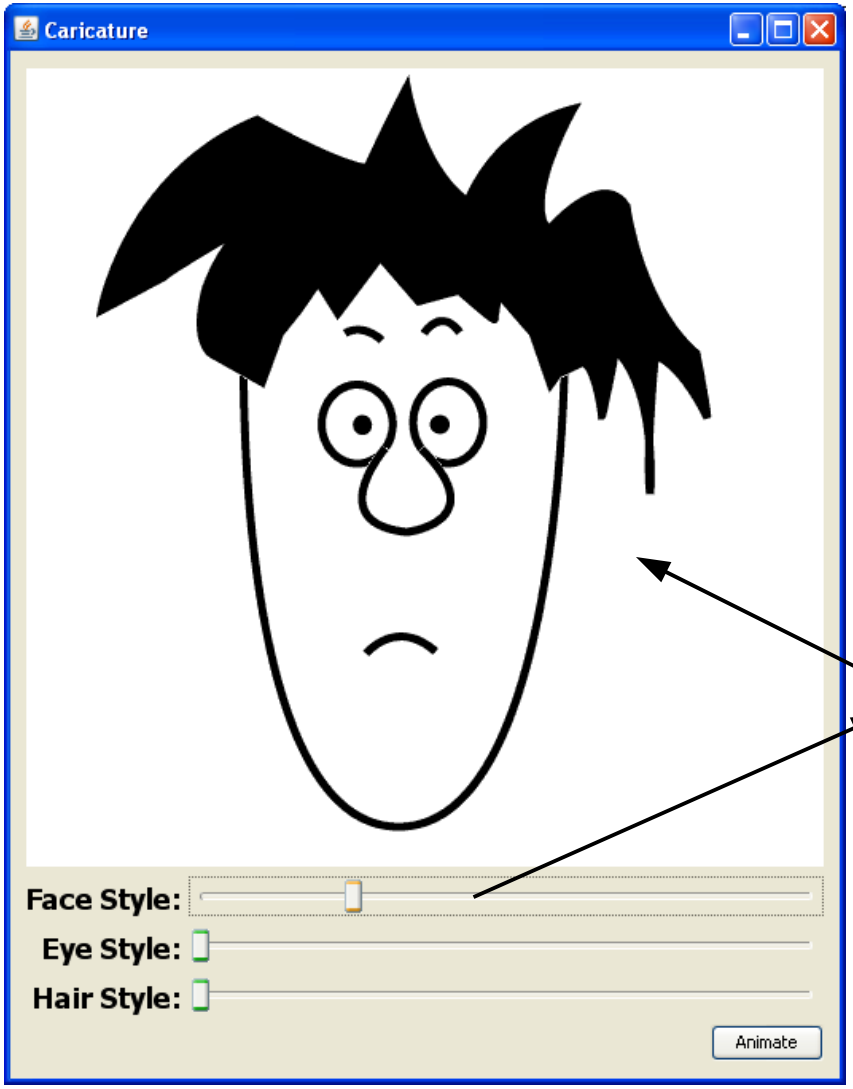
Demo Dissected



Demo Dissected

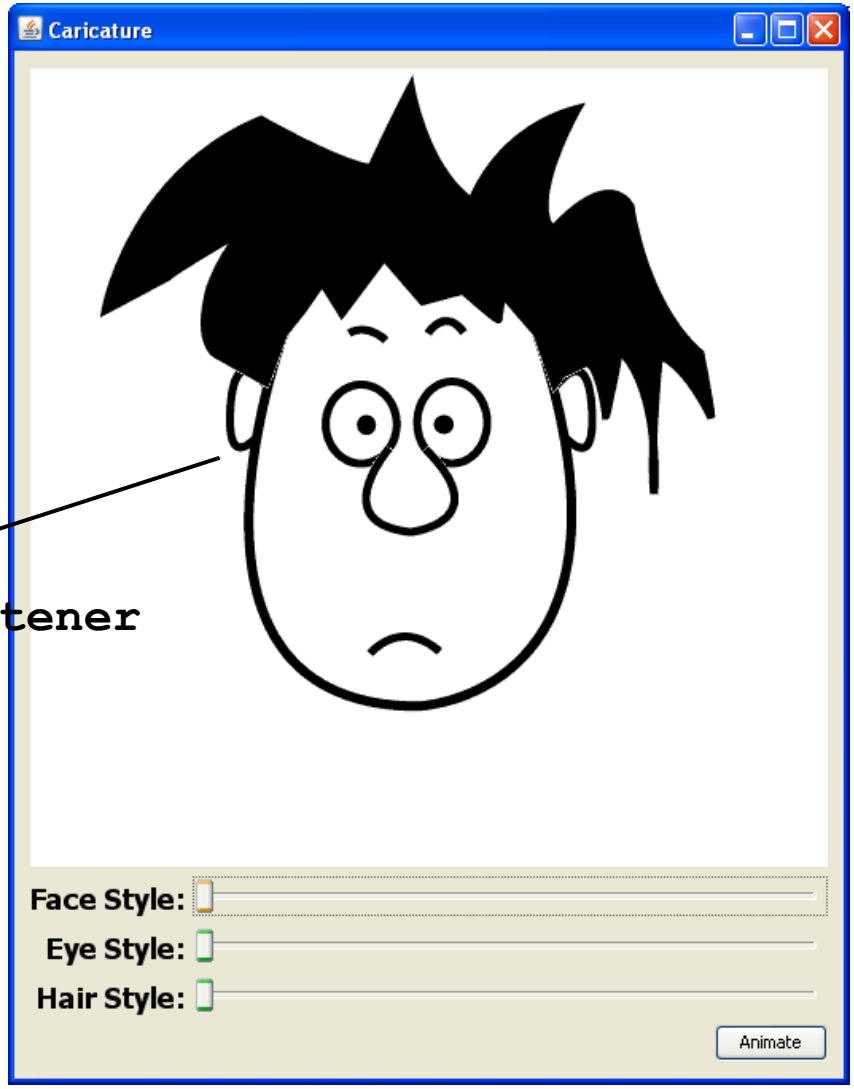


Demo Dissected



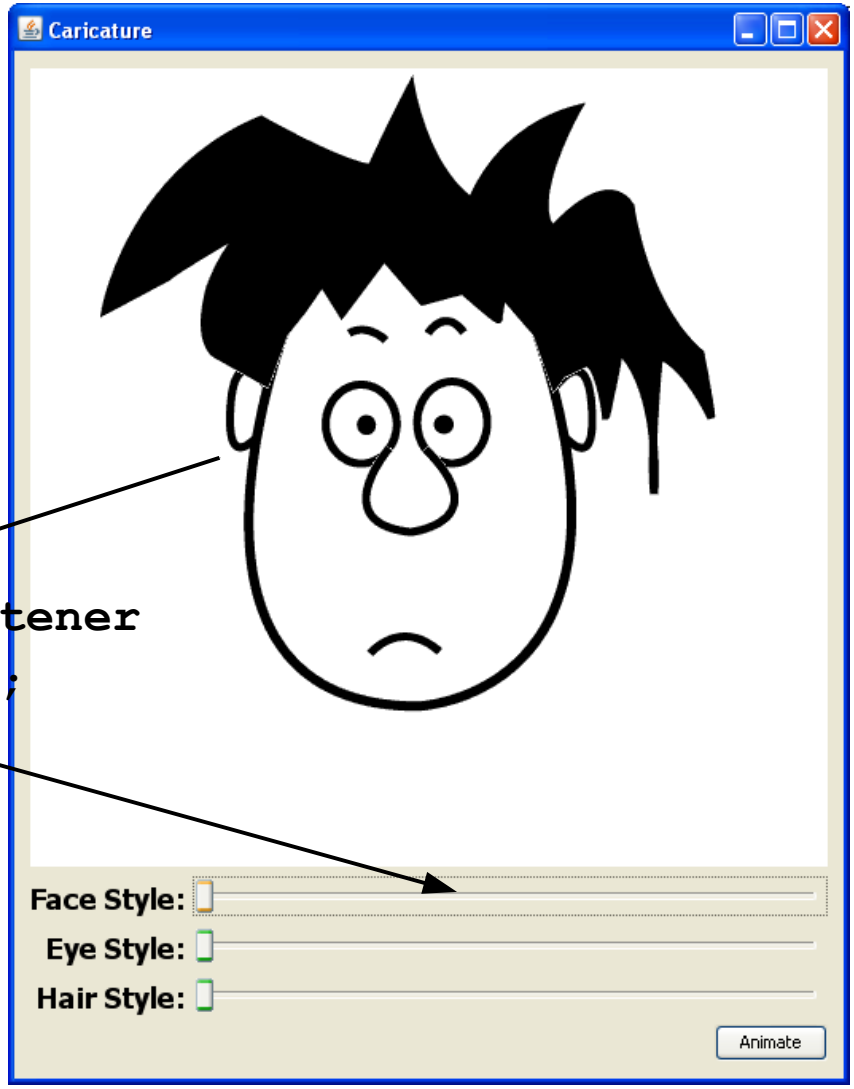

```
ChangeListener
setFaceStyle(x);
```

Demo Dissected



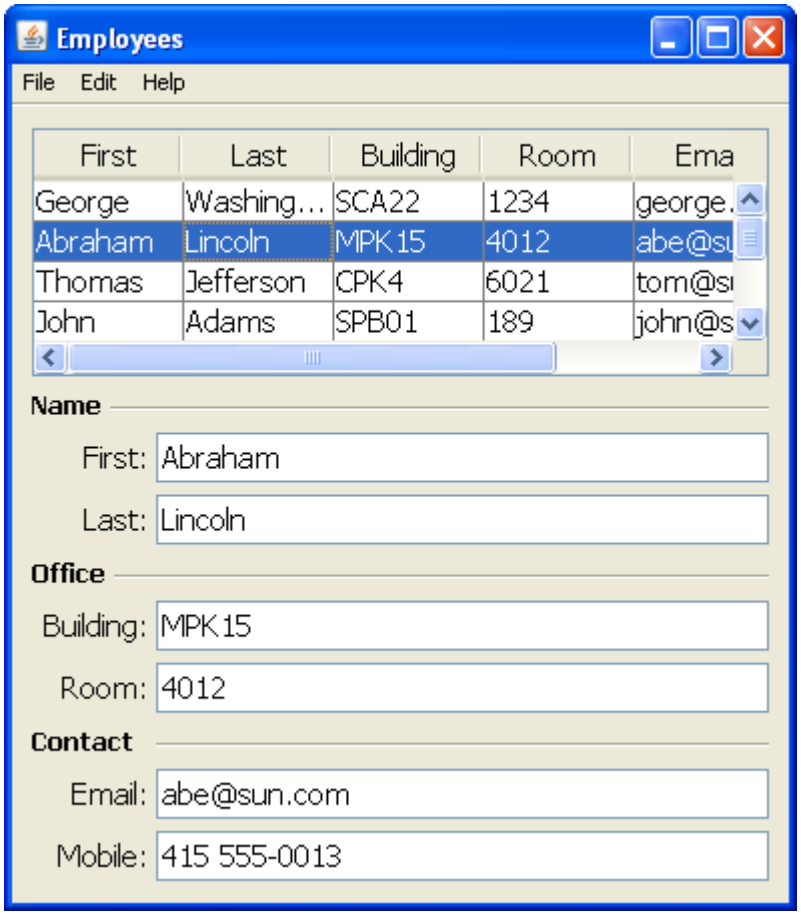
`PropertyChangeListener`

Demo Dissected

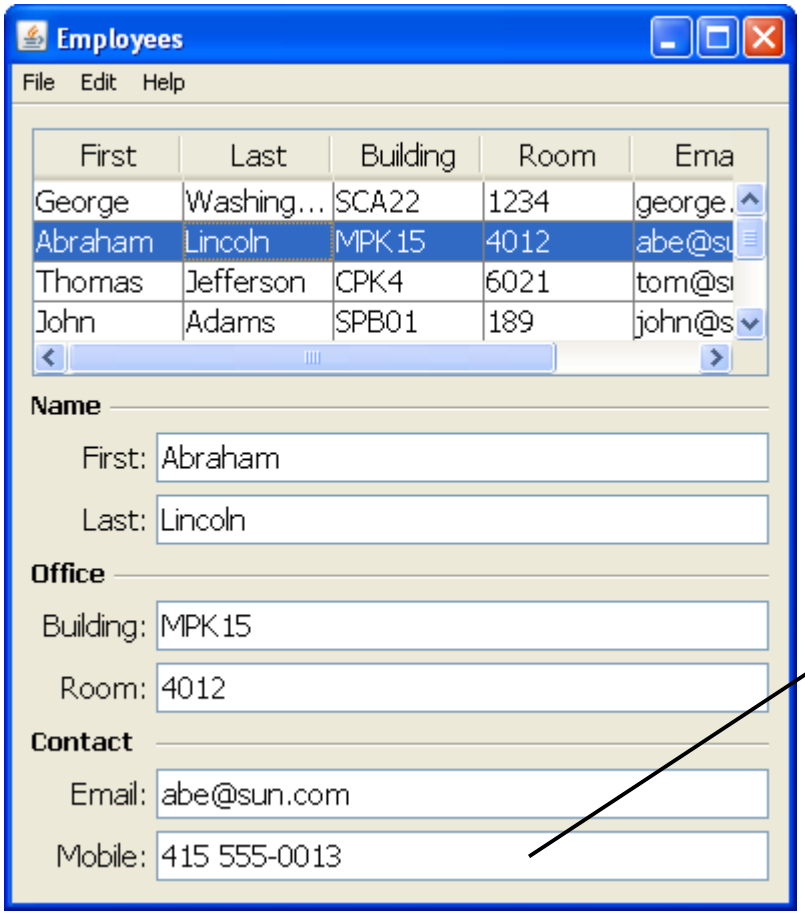



```
PropertyChangeListener
slider.setValue();
```

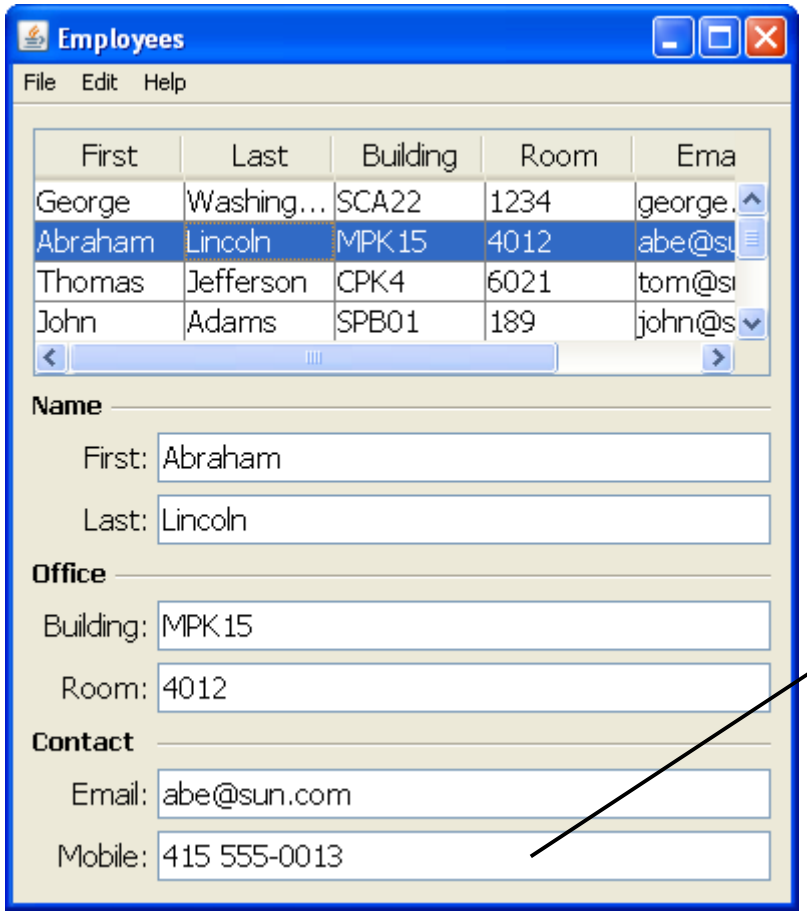
Typical Application



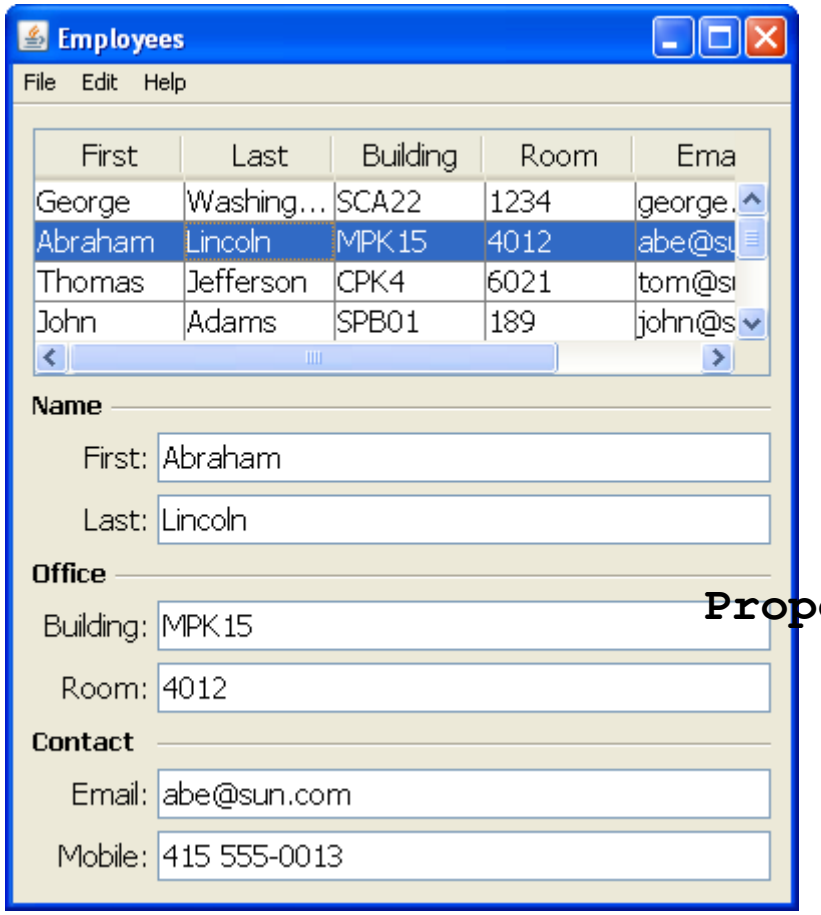
Typical Application



Typical Application



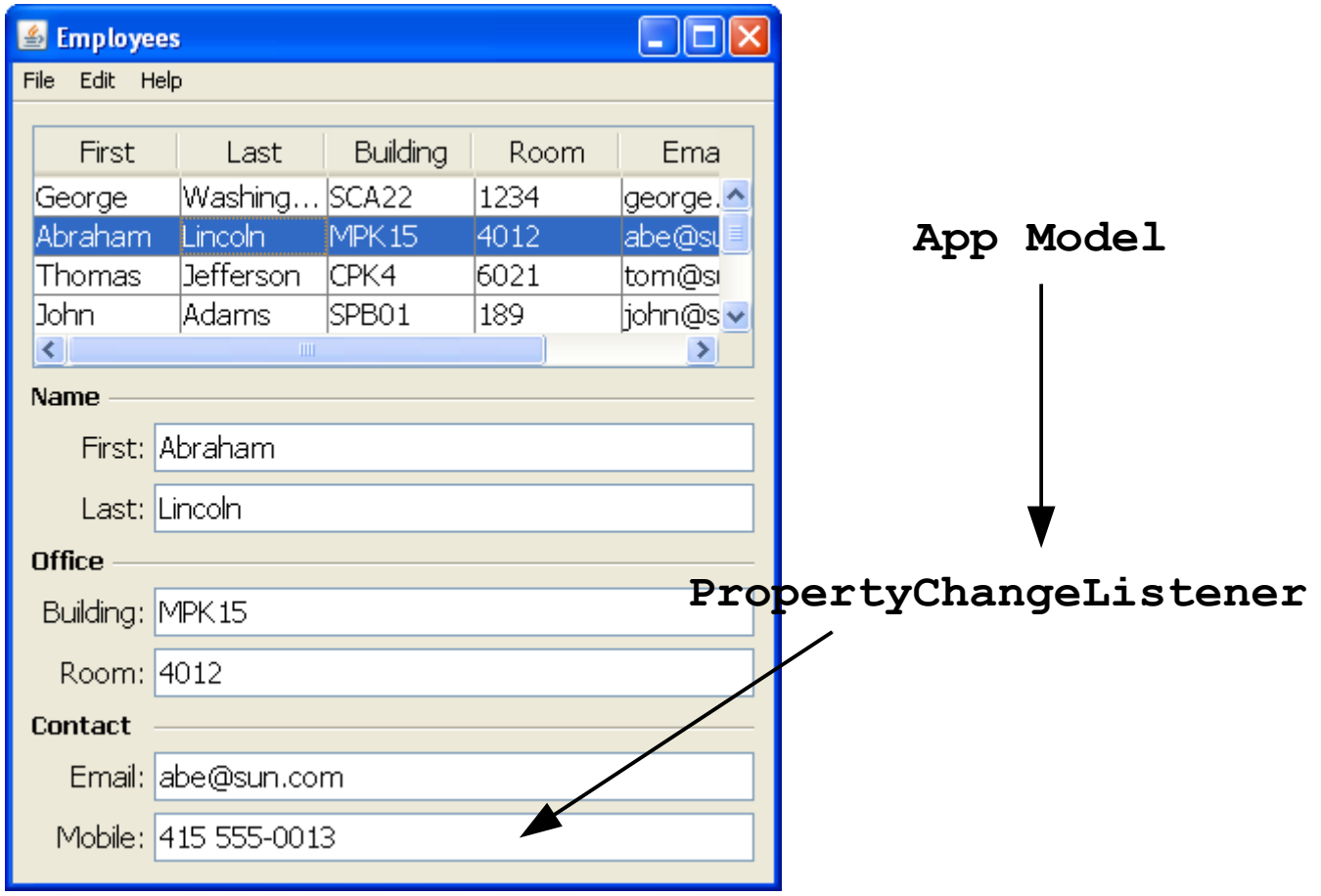
Typical Application



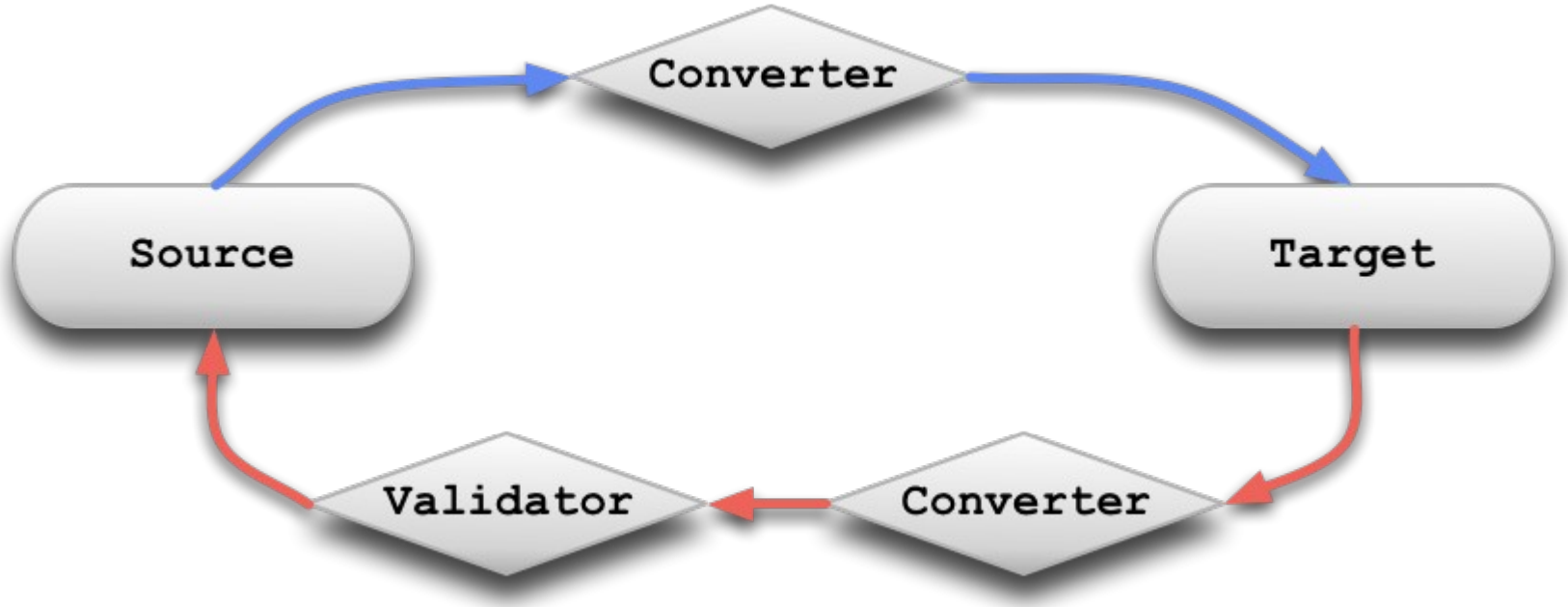
App Model

PropertyChangeListener

Typical Application



General Data Flow



Beans Binding

Keeps two properties of two objects in sync

Beans Binding

- Keeps two properties of two objects in sync
- Source properties are specified using Java Expression Language (EL) syntax:
  - ex: “${customer}” or “${employee.salary}”
- Does not require special object types, endpoints typed to Object

```
bind(Object source,
     String sourcePath,
     Object target,
     String targetPropertyName);
```

Beans Binding Builds Upon...

- Beans
  - Standard way to track changes to a property
    - PropertyChangeListener
  - Adds ability to accommodate objects that don't strictly follow beans spec
- Collection classes
  - Standard way to encapsulate common data types
  - Observable variants of List and Map will be created

Beans Binding

- Will accommodate objects that don't strictly follow beans pattern
  - Map treated as a bean with dynamic properties
- 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

Demo: Pre-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());
    }
  }
});
```

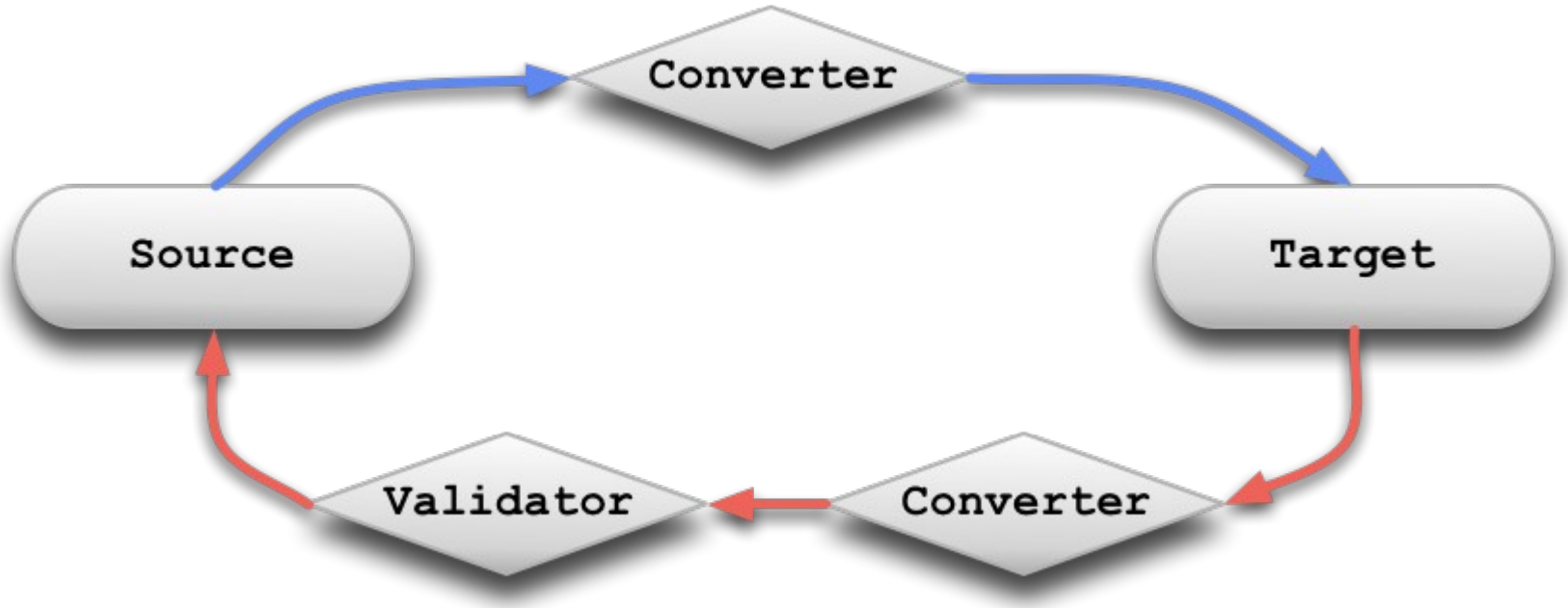
Demo: Binding

```
Binding binding = new Binding(
      caricature, "${eyeStyle}", // source
      eyeSlider, "value");     // target
binding.bind();
```

javax.beans.binding.Binding

- Describes and maintains binding between two objects
  - Source, target, source path, target property
- 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

Binding



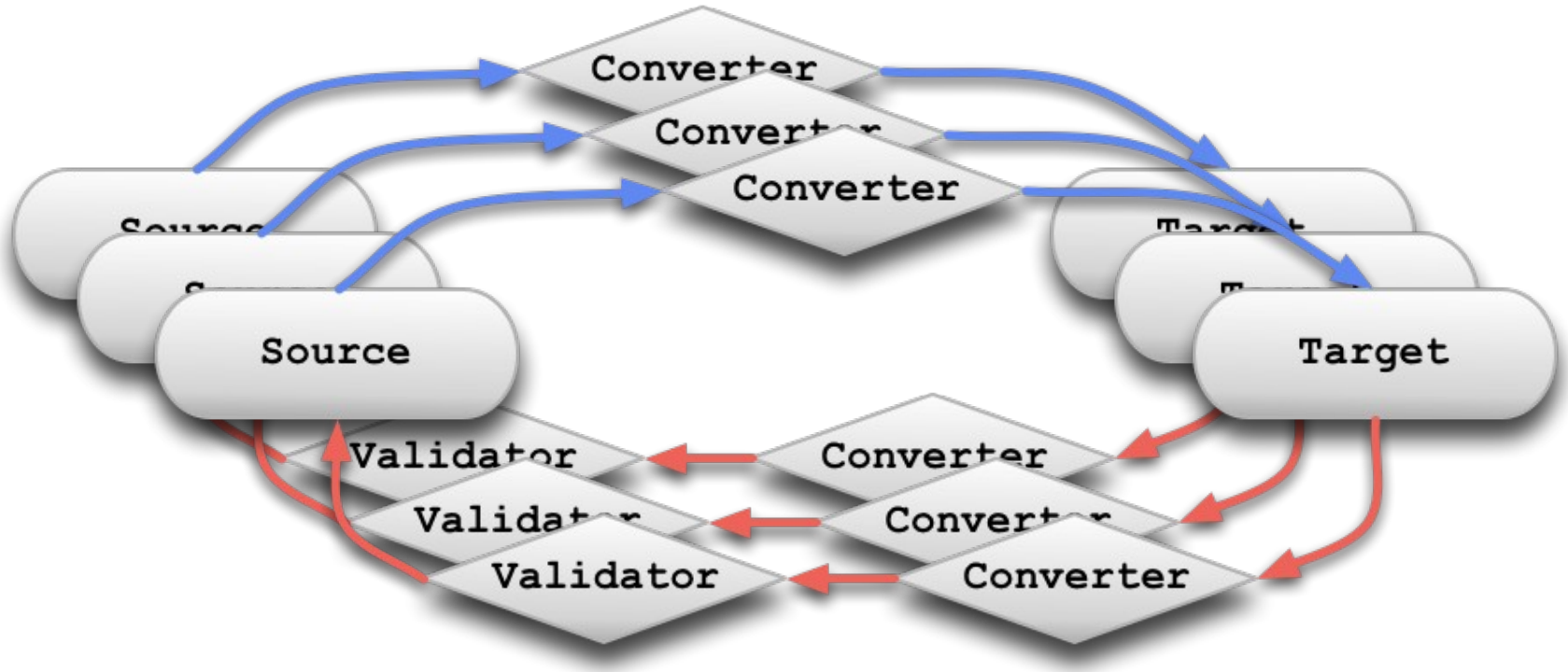
Binding: Example

```
// Create an unbound binding
Binding binding = new Binding(
     source, “sourcePath”,
     target, “targetPropertyName”);
// Bind it
binding.bind();
// Force the target property to update
// from the source
binding.setTargetValueFromSourceValue();
// Remove the binding
binding.unbind();
```

BindingContext

- Manages a Set of Bindings
- Methods and listener to track state of all Bindings
  - Invalid, newer, ...
- Single point to bind and unbind the Set of Bindings

BindingContext



BindingContext: Example

```
// Create an unbound binding
BindingContext context = new BindingContext();
// Add some bindings
context.addBinding(source, "sourcePath",
                   target, "targetPropertyName");
context.addBinding(source, "sourcePath",
                   target, "targetPropertyName");
// Bind all bindings in the context
context.bind();
```

BindingConverter

- Converts value from source to target, and back
- Default implementations will be supplied for common conversions
  - Integer -> String
  - String -> Dates
  - ...

BindingConverter: Example

```
BindingConver colorConverter =
  new BindingConverter() {
    public void Object sourceToTarget(
                                  Object value){
      if (((Boolean)value).booleanValue())
        return Color.GREEN;
      return Color.RED;
    }
};
Binding binding = new Binding(
    customer, "${active}",
    textField, "background");
binding.setConverter(colorConverter);
```

JTable

- Driven by a TableModel
  - int getRowCount();
  - Object getValueAt(int row, int column);
- Using Binding:
  - Specify List<T>, each T corresponds to a row
    - Changes to List are tracked if List is an ObservableList
  - Specify how the value for each column is obtained

JTable: Binding

In an ideal world

```
// Binds the list (ObservableList<?>) to table's
// “elements” property
bind(list, table, "elements");
// Specifies the first column should use the 'firstName'
// property.
bind("${firstName}", 0);
// Specifies the second column should use the 'lastName'
// property.
bind("${lastName}", 1);
```

JTable: Binding to elements

Working Code

```
// Binds the list (ObservableList<?>) to table's
// "elements" property
Binding binding = new Binding(
    list, null,              // source
    table, "elements");      // target
```

JTable: Binding to elements

Working Code

```
// Binds the list (ObservableList<?>) to table's
// "elements" property
Binding binding = new Binding(
    list, null,              // source
    table, "elements");      // target
// Bind the "firstName" property of each element to the
// 1st column
binding.addBinding(
    "${firstName}",          // source property EL
    "value",                 // target property
    TableColumnParameter, 0); // 1st column
```

JTable: Binding to elements

```
// Binds the list (ObservableList<?>) to table's
// "elements" property
Binding binding = new Binding(
    list, null,              // source
    table, "elements");      // target
// Bind the "firstName" property of each element to the
// 1st column
binding.addBinding(
    "${firstName}",          // source property
    "value",                 // target property
    TableColumnParameter, 0); // 1st column
// "lastName" property to the 2nd column
binding.addBinding(
    "${lastName}",           // source property
    "value",                 // target property
    TableColumnParameter, 1); // 2nd column
```

What is JTable “elements”?

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

PropertyDelegate

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

JTable: Binding from selectedElement

Binding source corresponds to the selected table row

```
Binding binding = new Binding(
  bugTable,                                  // JTable source
  "${selectedElement.firstName}",            // source EL
  nameTextField,                             // target
  "text",                                    // target property
  // The 'text' property should change as you type.
  // Default: only change on focus in/out
  TextChangeStrategyParameter,
  TextChangeStrategy.CHANGE_ON_TYPE);
```

JTable: Binding from selectedElement

Using EL to combine selectedElement properties

```
Binding binding = new Binding(
  bugTable,       // JTable source
  // Source is "firstName, lastName"
  "${selectedElement.lastName}, ${selectedElement.firstName}",
  nameTextField, // target
  text");         // target property

// Changing nameTextField target doesn't change source
// Changes to the source do update nameTextField
binding.setUpdateStrategy(
    UpdateStrategy.READ_FROM_SOURCE);
```

JTable: Binding from selectedElements

Using EL to compute elements/selectedElements properties

```
Binding binding = new Binding(
  bugTable,   // JTable source
  // Source expression. The listSize EL function
  // is defined by Beans Binding
  "${bb:listSize(selectedElements)} of" +
  "${bb:listSize(elements)} are selected"
  summaryLabel, // Jlabel target
  "text");      // Jlabel target property
```

Demo

Status

- Early prototype done
- Working closely with Matisse team for support in Matisse
- Shooting for public project at end of month
  - http://beansbinding.dev.java.net

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

Beans Binding

Shannon Hickey (Shannon.Hickey@Sun.COM)
Hans Muller (Hans.Muller@Sun.COM)