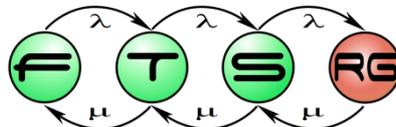


High Level GUI Programming

Eclipse Based Technologies

<http://inf.mit.bme.hu/edu/courses/eat>



SWT and JFace

- SWT
 - Native, low level widget kit
 - Controllable behaviour
 - Lot of coding required
- JFace
 - High level components (based on SWT)
 - More automatic
 - Structured data
 - Easier reuse
 - Sometimes does not work as needed

JFace Viewers

Displaying Structured Data

JFace Viewers

- Common handling for structured widgets
 - SWT Table, Tree és List
- Approach
 - Different widgets
 - Same processing methods
 - Data connection
 - Filtering, ordering
 - Editing

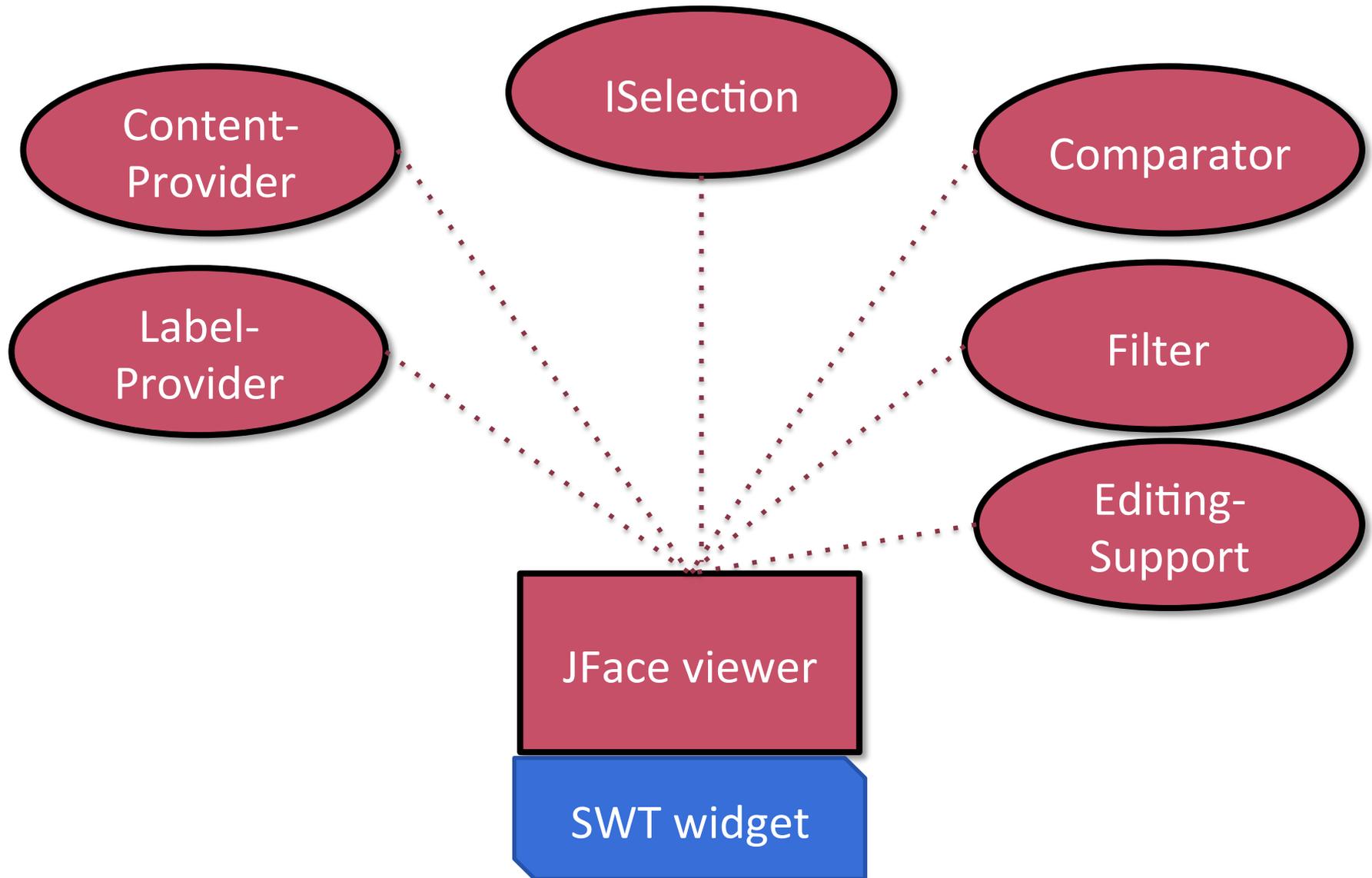
Important!

Eclipse Views and JFace Viewers are different!

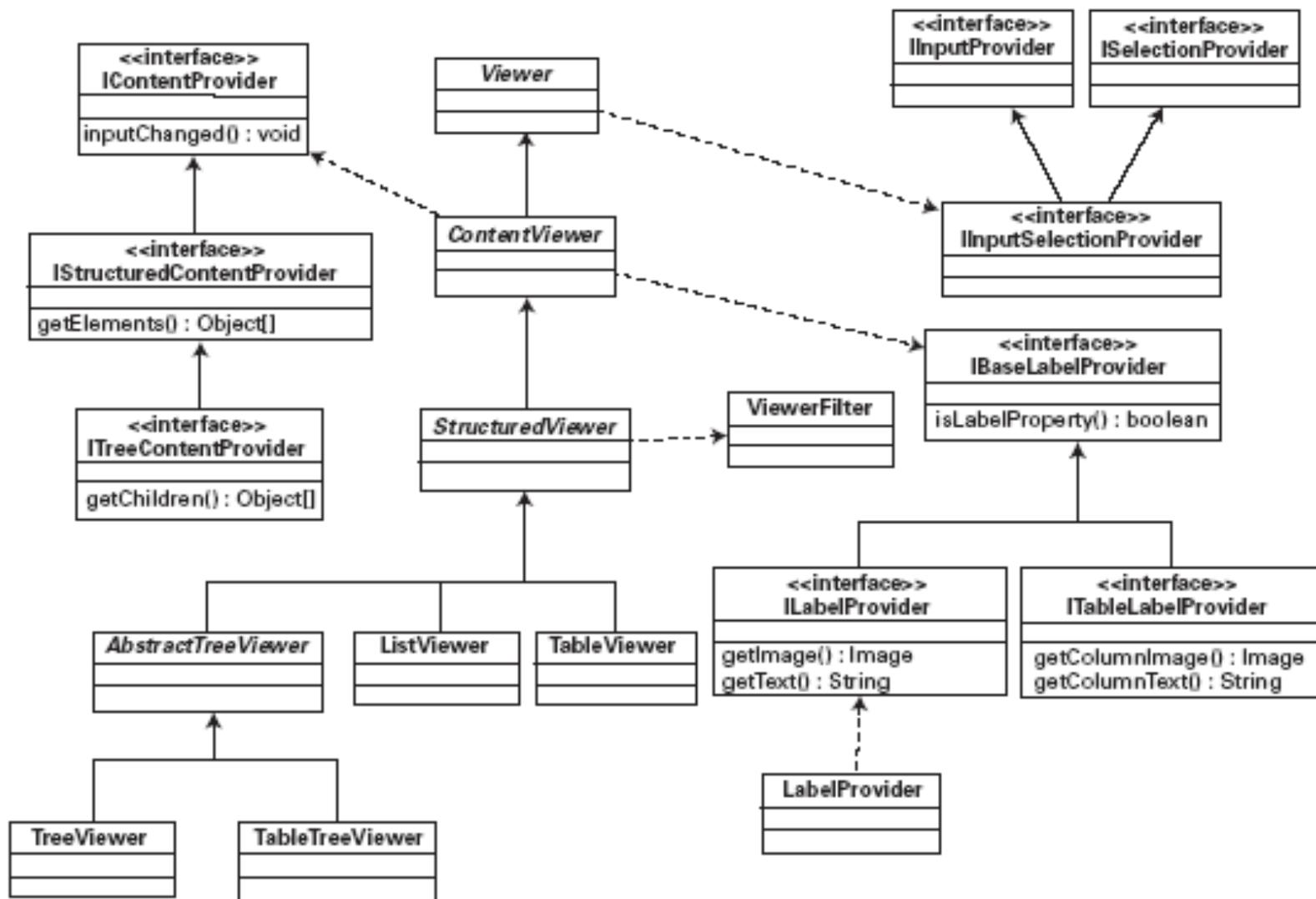
JFace Viewers

- MVC pattern
 - Model:
 - Any kind of model works
 - Binding to viewer: ContentProvider, LabelProvider
 - View:
 - Uses selected SWT widget
 - Filtering, ordering
 - Selection handling
 - Controller:
 - Event listeners
 - Editing support
 - ...

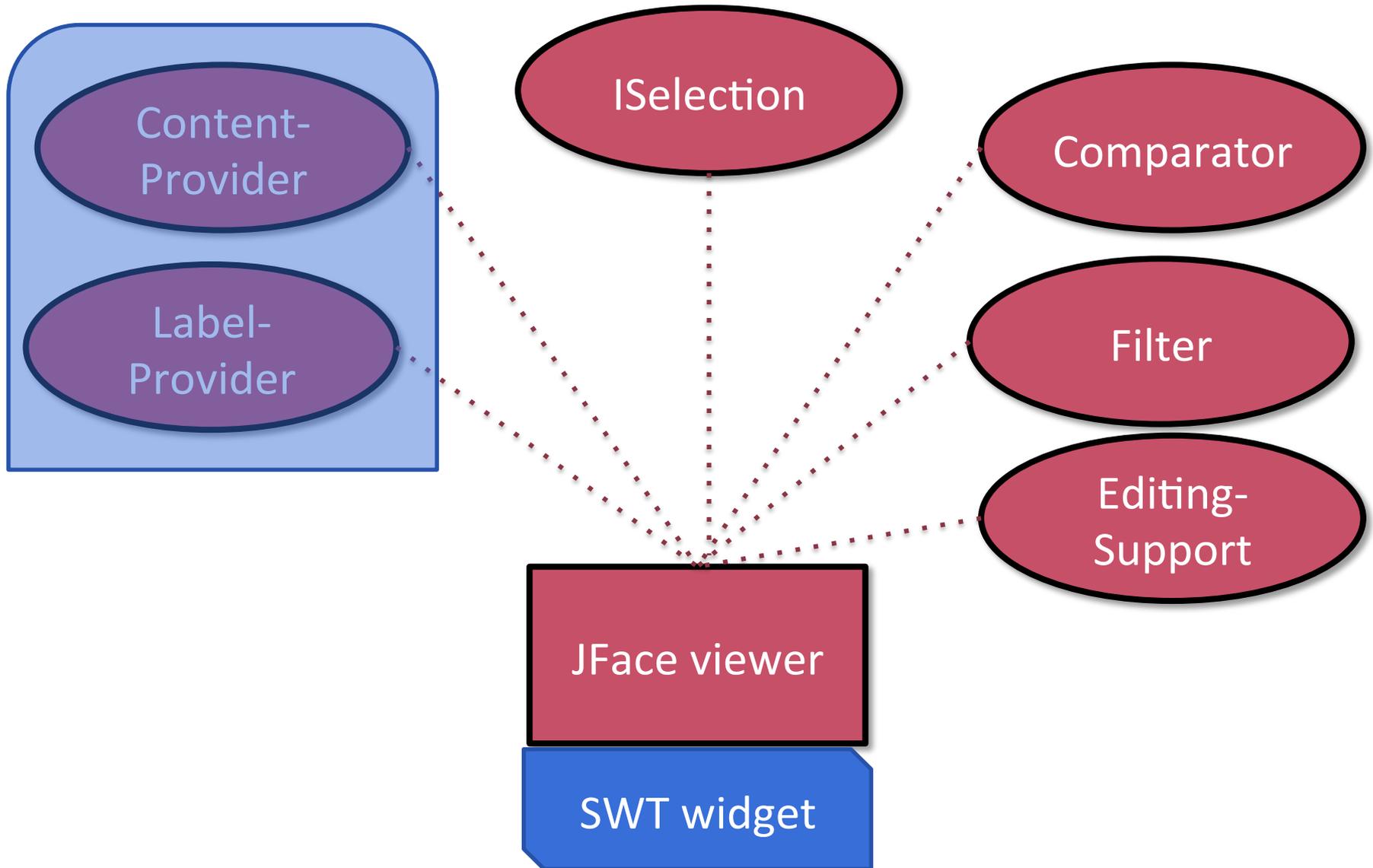
JFace Viewers - Keywords



JFace Viewers – Class Hierarchy



JFace Viewers - Keywords



Content Provider

- `#inputChanged(Viewer, Object, Object)`
 - Notification that the input is changed
 - input can be saved here
 - `getElements()` will be called later
 - Never call directly, use `viewer.setInput(Object)`
- `#getElements()`
 - Returns an array of items
 - Only called after `#inputChanged` was executed

Label Provider

- Assigns labels and icons
 - #getText()
 - #getImage()
 - #isLabelProperty()
 - Has the label changed because of a property change?
- Default implementation
 - For label it uses toString()
 - Does not return any icon

In runtime

	Viewer	Content provider	Label provider
1.	setInput		
2.		inputChanged	
3.		getElements	
4.			getLabel
5.			getImage

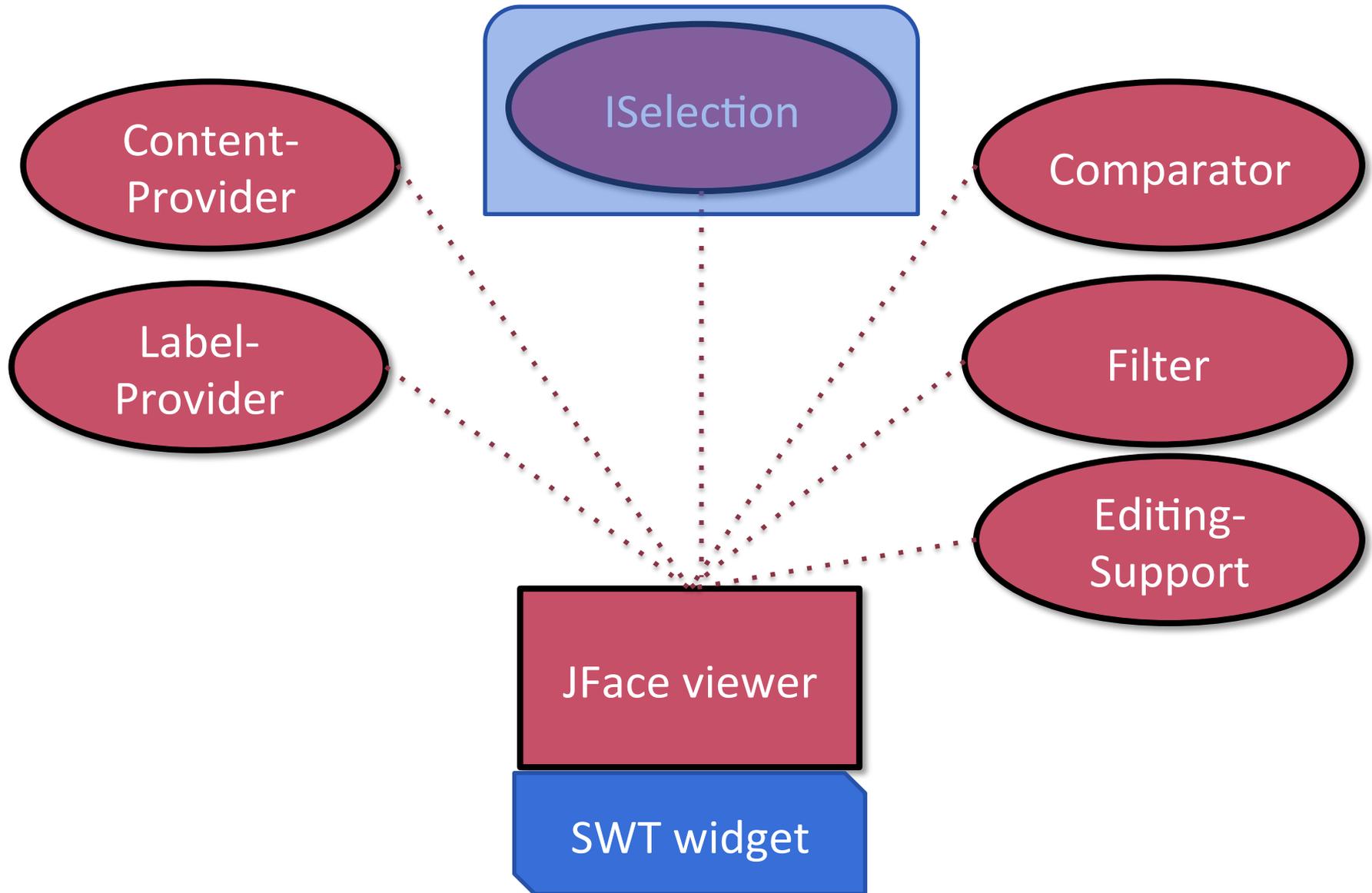
- Step 3

- May happen in several steps, e.g. lazy loading

- Step 4-5:

- Called for each item

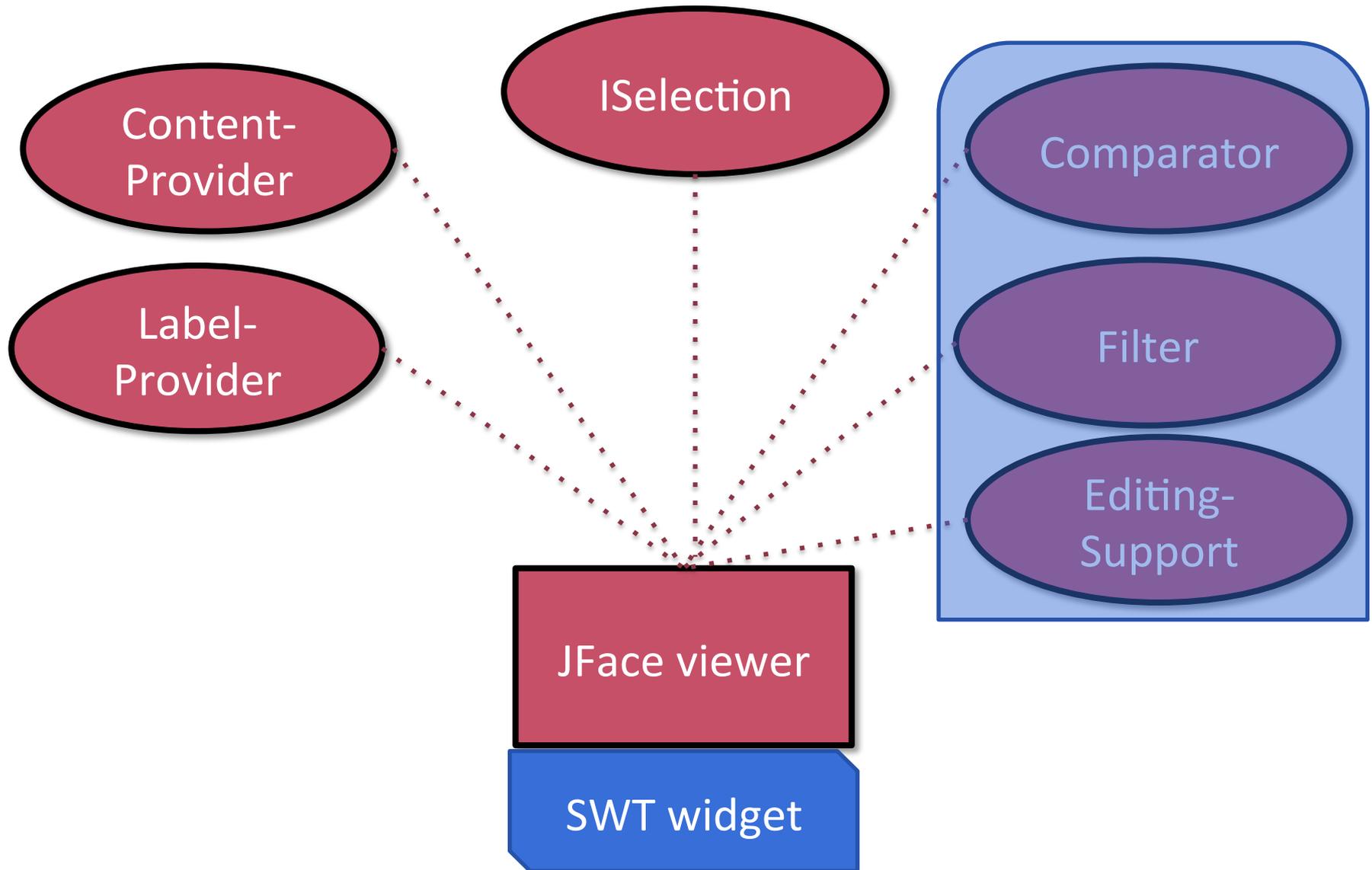
JFace Viewers - Keywords



Selection support

- `getSelection()`
 - `ISelection`
 - Generic selection marker
 - Cannot be used directly
 - `IStructuredSelection`
 - A selection of some items
 - Use an `Iterator` to traverse items
 - `ITreeSelection`
 - `IStructuredSelection` descendant
 - Describes hierarchy

JFace Viewers - Keywords



Ordering

- Define ordering
 - ViewerComparator interface
 - Compares any two instances
 - Must be fast!
 - Uses quick sort
 - Fast enough, but hard to debug

Filtering

■ Filter interface

- Decides for each item whether it should be visible
 - Must be fast
- Possible to set up multiple filters

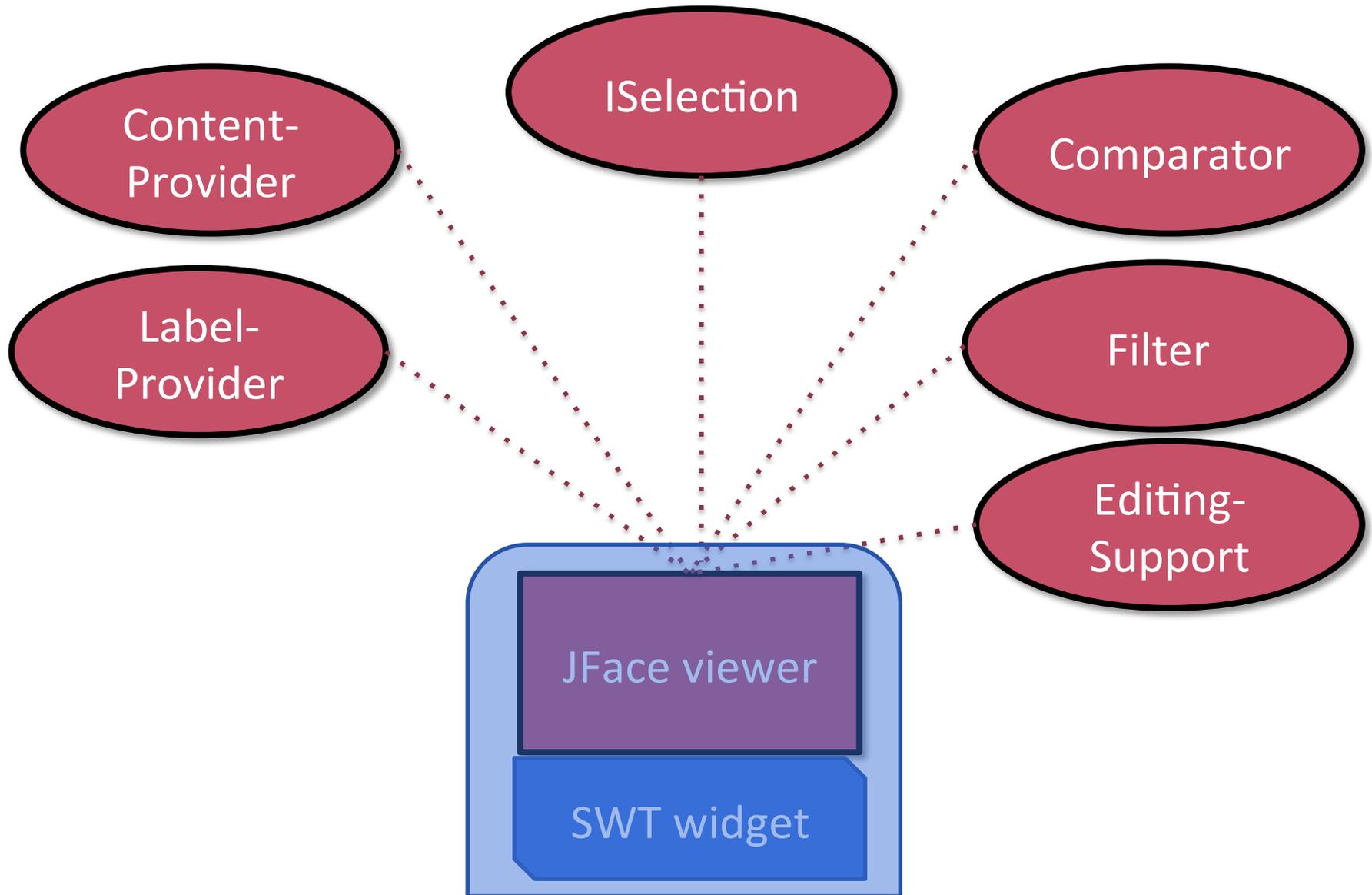
■ Where to filter content?

- Content provider:
 - Better performance
- As filter
 - More versatile
 - Can be turned on/off during runtime

Editing Support

- Viewer editing
 - Temporary editing widget on top of the viewer
- Editingsupport component
 - Not all Viewers support it
 - Read API 😊

JFace Viewers - Keywords



ListViewer

- Displays a list of items
- Uses IStructuredContentProvider
 - getElements()
 - Only returns the items of the list
- Every feature available
 - Sorter
 - Filter
 - Label Provider

TreeViewer

- Display items in a tree structure
- Uses ITreeContentProvider
 - Returns the elements to display
 - Additional methods for tree structure
 - `getChildren()`
 - Returns the children of an item
 - `hasChildren()`
 - Returns whether an item has children
 - If slow to calculate, always return true
 - `getParent()`
 - Return parent
 - Required for lazy trees!

TableView

- To manage tables
 - Each row is an item
- TableLayout – adjusting table columns
 - addColumnData()
- ITableLabelProvider
 - To display content in a cell (row/column pair)

Editing Tables

■ CellEditor

○ ICellModifier

- getValue()
 - Load default value from editor
- canModify()
 - Decide whether value is editable
- modify()
 - Writes back the new value

○ CellEditor

- Built-in: Checkbox, Combo box, pop-up dialog, text
- Custom celleditor possible

TableViewer problems

- Column identifier: index
- One cell editor per column
 - Row-dependant cell editor is problematic
 - E.g. drop-down list with row-specific content
- Many manual, error-prone code needed
- Since Eclipse 3.3 new API
 - TableViewerColumn

TableViewColumn

- Defines a column in the table
- Own LabelProvider
 - Use ColumnLabelProvider base class
- Custom editing support

EditingSupport

- Helper class for editing
- `canEdit(Object element)`
 - Can we edit the selected cell
- `getCellEditor(Object element)`
 - Returns the cell editor
- `setValue(Object element, Object value)`
 - Sets the modified value in the model
- `getValue(Object element)`
 - Returns the value from the model
 - The Cell editor needs to understand the return value!

EditingSupport example

```
protected abstract class AbstractEditingSupport
    extends
        EditingSupport {

    private TableCellEditor editor;

    public AbstractEditingSupport(TableViewer viewer)
    {
        super(viewer);
        editor = new TableCellEditor(viewer.getTable());
    }

    protected boolean canEdit(Object element) {
        return true;
    }
}
```

...

EditingSupport example

```
protected abstract class AbstractEditingSupport
    extends
        EditingSupport {

    private TableCellEditor editor;

    public AbstractEditingSupport(TableView<T> viewer)
    {
        super(viewer);
        editor = new TableCellEditor(viewer.getTable());
    }

    protected boolean canEdit(Object element) {
        return true;
    }
}
```

Textual cell editor
definition

...

EditingSupport example

```
protected abstract class AbstractEditingSupport
    extends
        EditingSupport {

    private TableCellEditor editor;

    public AbstractEditingSupport(TableViewer viewer)
    {
        super(viewer);
        editor = new TableCellEditor(viewer.getTable());
    }

    protected boolean canEdit(Object element) {
        return true;
    }
}
```

Setting
modifiability

...

EditingSupport example – 2.

...

```
protected CellEditor getCellEditor(Object element) {
    return editor;
}

protected void setValue(Object element, Object value)
{
    ((Person) element).email = value.toString();
    getViewer().update(element, null);
}

protected Object getValue(Object element) {
    return ((Person) element).email;
}
}
```

EditingSupport example – 2.

...

```
protected CellEditor getCellEditor(Object element) {  
    return editor;  
}
```

Return specific
CellEditor

```
protected void setValue(Object element, Object value)  
{  
    ((Person) element).email = value.toString();  
    getViewer().update(element, null);  
}
```

```
protected Object getValue(Object element) {  
    return ((Person) element).email;  
}
```

```
}
```

EditingSupport example – 2.

...

```
protected CellEditor getCellEditor(Object element) {  
    return editor;  
}  
  
protected void setValue(Object element, Object value)  
{  
    ((Person) element).email = value.toString();  
    getViewer().update(element, null);  
}  
  
protected Object getCellEditorValue(Object element) {  
    return ((Person) element).email;  
}  
}
```

Set correct values

EditingSupport example – 2.

...

```
protected CellEditor getCellEditor(Object element) {  
    return editor;  
}  
  
protected void setValue(Object element, Object value)  
{  
    ((Person) element).email = value.toString();  
    getViewer().setSelection(0, null);  
}  
  
protected Object getValue(Object element) {  
    return ((Person) element).email;  
}  
}
```

Return the
stored value

JFace Dialogs and Wizards

JFace dialogs

- **MessageDialog**
 - Displays messages
- **ErrorDialog**
 - Displays error messages
 - Uses IStatus to describe error
 - Severity
 - Message
 - Exception
- **InputDialog**
 - Simple text input
 - IInputValidator
 - Input validation!

JFace dialogs

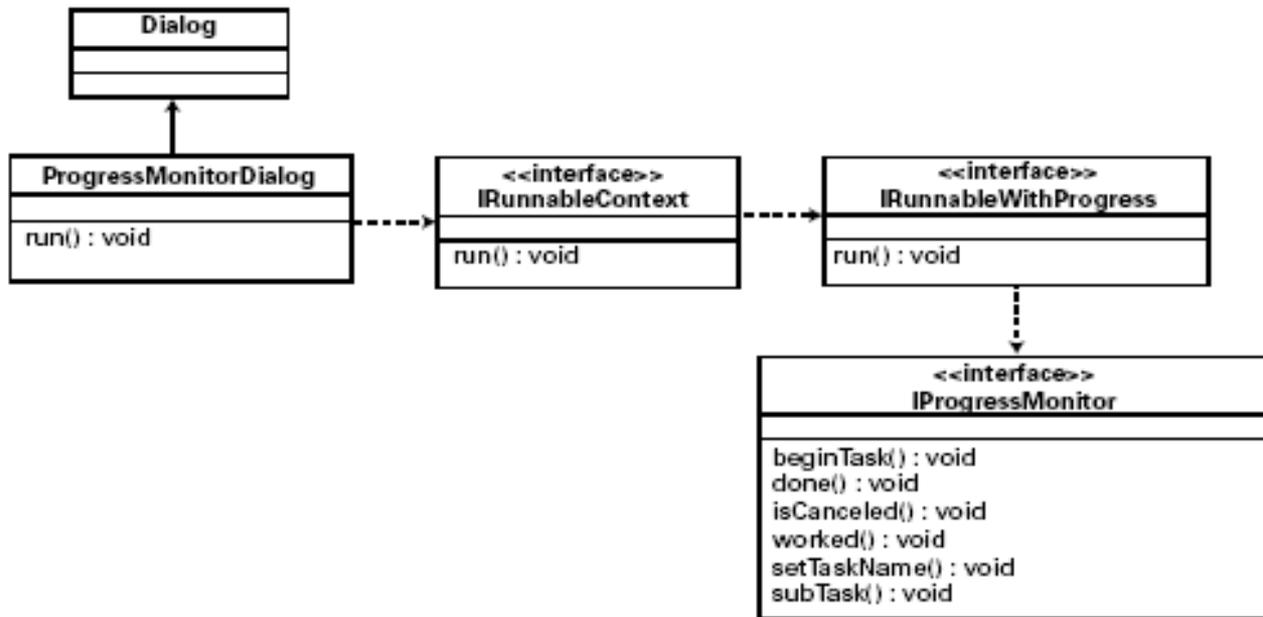
- **MessageDialog**
 - Displays messages
- **ErrorDialog**
 - Displays error messages
 - Uses IStatus to describe error
 - Severity
 - Message
 - Exception
- **InputDialog**
 - Simple text input
 - IInputValidator
 - Input validation!

JFace dialogs

- **MessageDialog**
 - Displays messages
- **ErrorDialog**
 - Displays error messages
 - Uses IStatus to describe error
 - Severity
 - Message
 - Exception
- **InputDialog**
 - Simple text input
 - IInputValidator
 - Input validation!

ProgressDialog

- Display background progress



Custom dialog

- Base class: `org.eclipse.jface.dialogs.Dialog`
- Callback methods
 - `createDialogArea(Composite parent)`
 - Dialog content
 - `createButtonBar(Composite parent)`
 - Button setup
- Header configuration
 - Redefine `configureShell` method

```
protected void configureShell(Shell
    shell) {
    super.configureShell(shell);
    shell.setText("My Dialog Title");
}
```

Dialog Configuration

- Keeping dialog values (save/restore):
DialogSettings class

- DialogSettings (String)
- put (String, Object)
- save (String)
- load (String)
- get (String)
- get* (String)

Wizards

- WizardContainer class
 - A set of wizards
 - E.g. Import wizards
- Wizard class
 - Stores a set of wizard pages
 - Manages the control flow
 - Most important methods
 - `canFinish()`
 - `performCancel()`, `performFinish()`
 - `createPageControls()`

Wizards

■ WizardPage

- A single wizard page
- Manages input (+ validation)
- Does not manage final setup
- Methods
 - `getName()`
 - `getNextPage()`, `getPreviousPage()`
 - `isPageComplete()`
 - `canFlipToNextPage()`

Wizards

■ Tasks

○ WizardPage

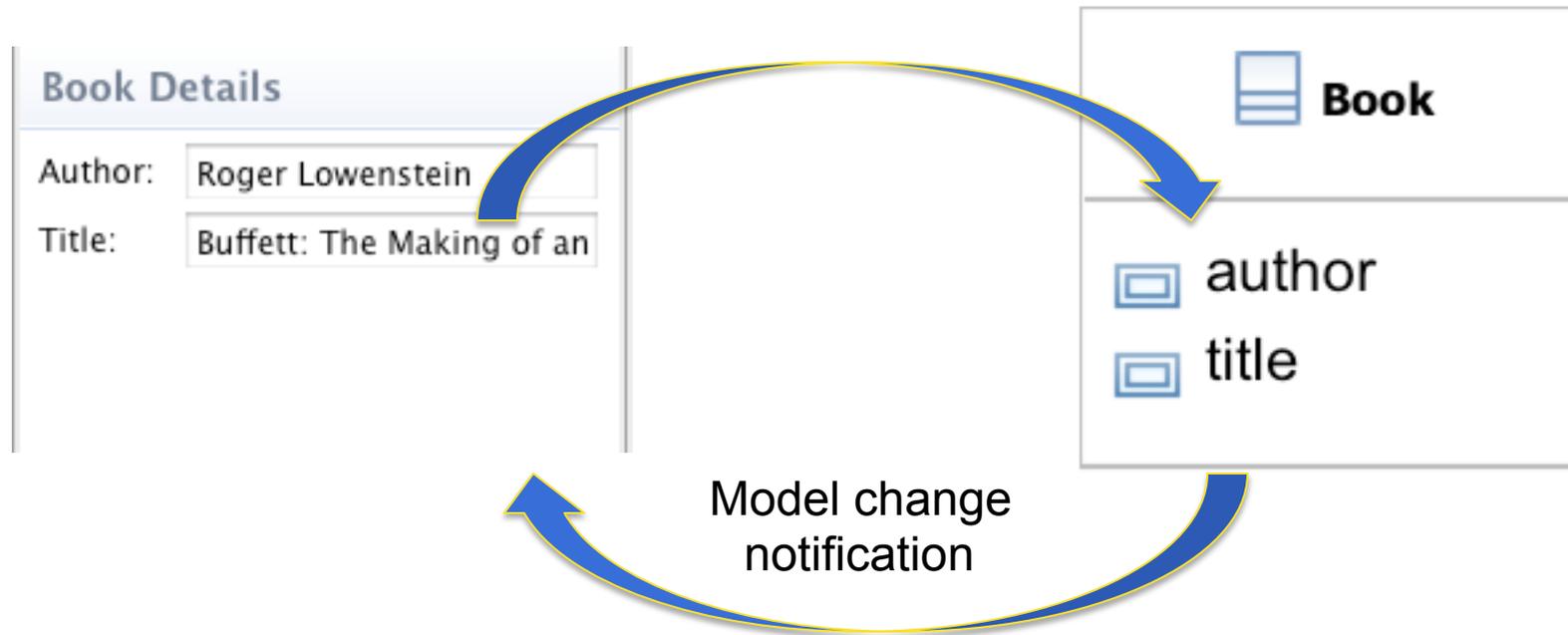
- Information presentation
- Data input and validation

○ Wizard

- Order of pages
- Read of data/context
- Business logic/control flow
 - Between pages: collect store the entered data
 - Optionally leave out pages
 - After finished: make the required changes in the environment

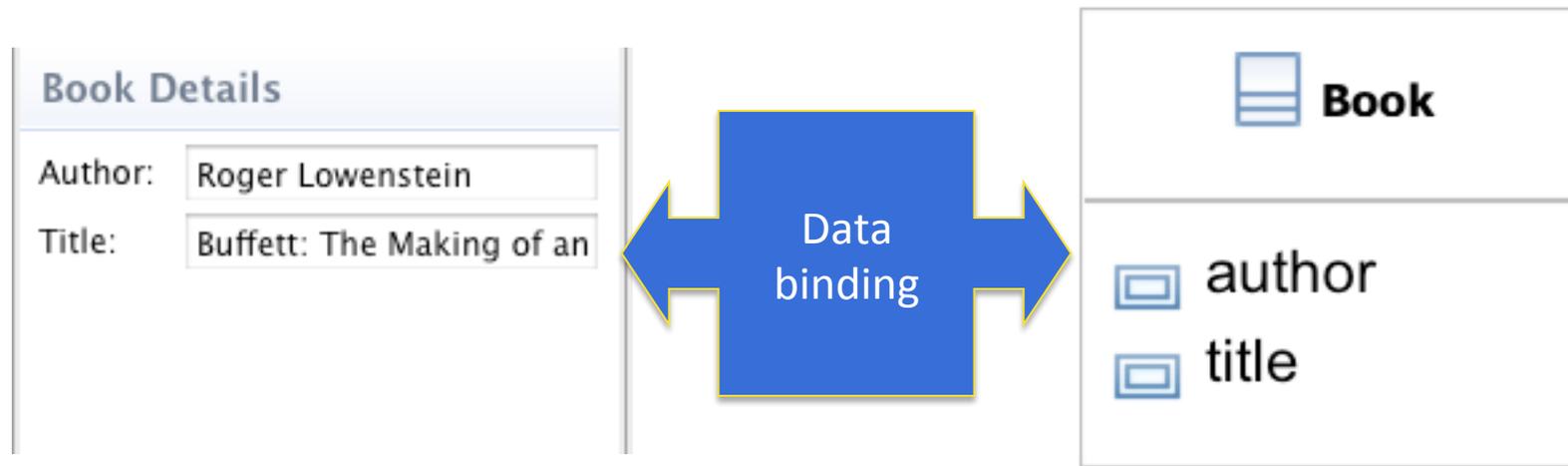
Data Binding

User Interface and Data Models



- Synchronization: complex, but mechanical
 - Easy to make mistakes
 - Is it automatable?

User Interface and Data Models



- Data Binding
 - Synchronization between data model and UI
- JFace Data Binding
 - Set up model and UI objects
 - Conversion and validation support

JFace Data Binding - Terminology

■ Observable

- Structured object, e.g., value, list, set...
- State changes can be observed

■ Binding

- Connects two *Observables*
- Uni- or bidirectional mapping

■ Data binding context

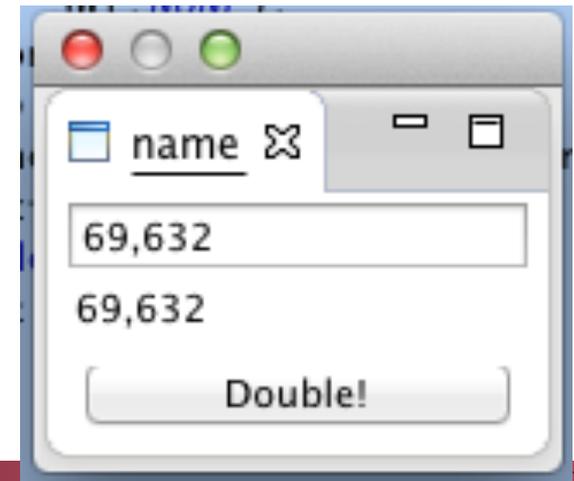
- Stores all bindings

■ Realm

- Serializes all *Observable* access
- Automatically created during Workbench startup

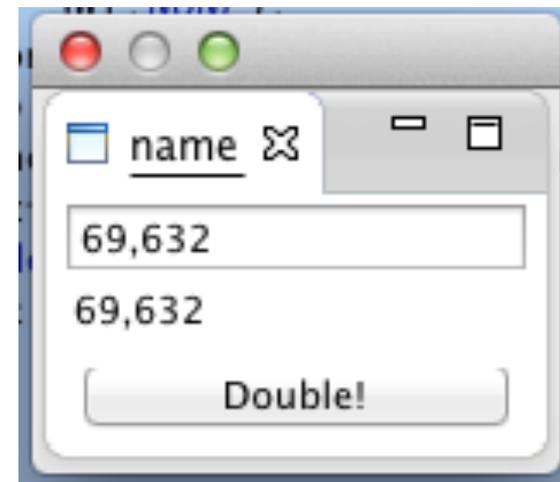
Data Binding Example - Form

```
text = new Text(parent, SWT.BORDER);  
label = new Label(parent, SWT.NONE);  
Button button = new Button(parent, SWT.PUSH);  
button.setText("Double!");  
button.addListener(new SelectionAdapter(){  
    public void widgetSelected(SelectionEvent e){  
        final int amount = model.getAmount();  
        model.setAmount(amount * 2);  
    }  
});
```



Data Binding Example - Bindings

```
DataBindingContext dbc = new DataBindingContext();  
IObservableValue modelObservable =  
    BeansObservables.observeValue(model, "amount");  
dbc.bindValue(  
    SWTObservables.observeText(text, SWT.Modify),  
    modelObservable,  
    null, null);  
  
dbc.bindValue(  
    SWTObservables.observeText(label),  
    modelObservable,  
    null, null);
```

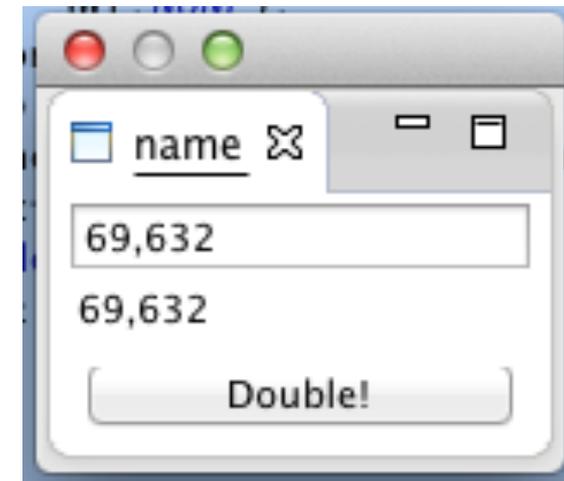


Data Binding Example - Bir

Observable for
the „amount”
property of
model

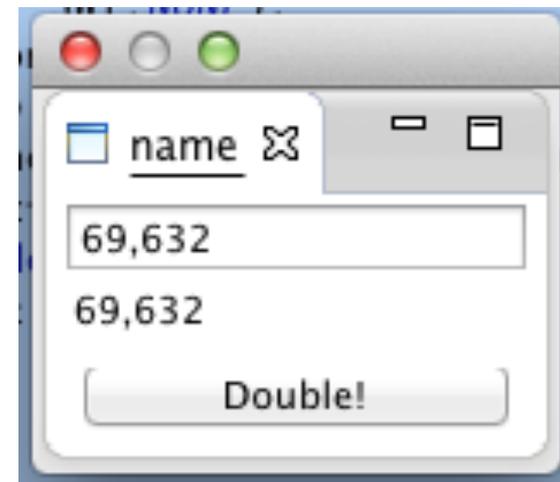
```
DataBindingContext dbc = new DataBindingContext();  
IObservableValue modelObservable =  
    BeansObservables.observeValue(model, "amount");  
dbc.bindValue(  
    SWTObservables.observeText(text, SWT.Modify),  
    modelObservable,  
    null, null);
```

```
dbc.bindValue(  
    SWTObservables.observeText(label),  
    modelObservable,  
    null, null);
```



Data Binding Example - Bindings

```
DataBindingContext dbc = new DataBindingContext();  
IObservableValue modelObservable =  
    BeansObservables.observeValue(model, "amount");  
dbc.bindValue(  
    SWTObservables.observeText(text, SWT.Modify),  
    modelObservable,  
    null, null);  
  
dbc.bindValue(  
    SWTObservables.observeText(label),  
    modelObservable,  
    null, null);
```

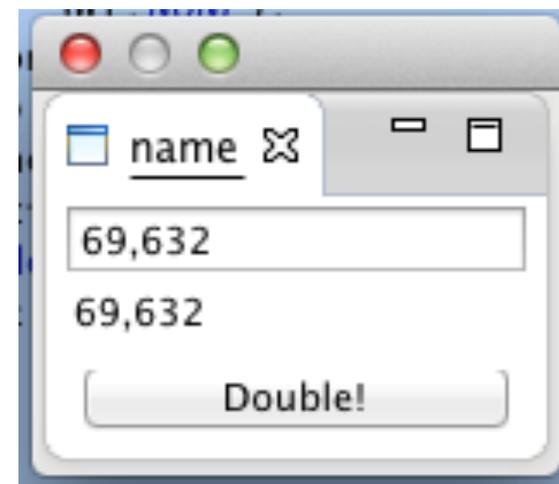


Data Binding Example - Bindings

```
DataBindingContext dbc = new DataBindingContext();  
IObservableValue modelObservable =  
    BeansObservables.observeValue(model,  
dbc.bindValue(  
    SWTObservables.observeText(text, SWT.Modify),  
    modelObservable,  
    null, null);
```

Binding text ...

```
dbc.bindValue(  
    SWTObservables.observeText(label),  
    modelObservable,  
    null, null);
```

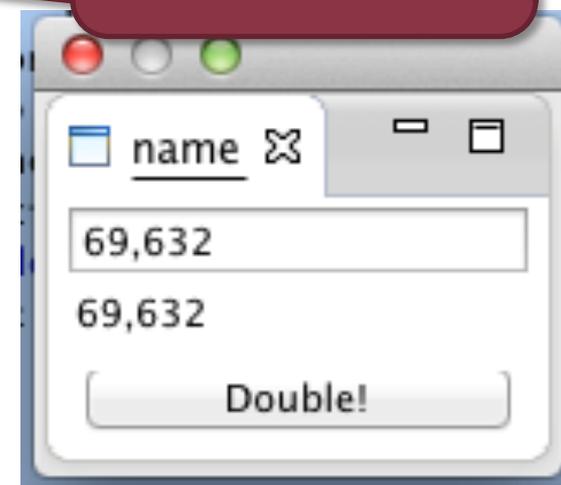


Data Binding Example - Bindings

```
DataBindingContext dbc = new DataBindingContext();  
IObservableValue modelObservable =  
    BeansObservables.observeValue(model,  
dbc.bindValue(  
    SWTObservables.observeText(text, SWT.  
    modelObservable,  
    null, null);  
  
dbc.bindValue(  
    SWTObservables.observeText(label),  
    modelObservable,  
    null, null);
```

Binding text ...

... to model
observable ...



Data Binding Example - Bindings

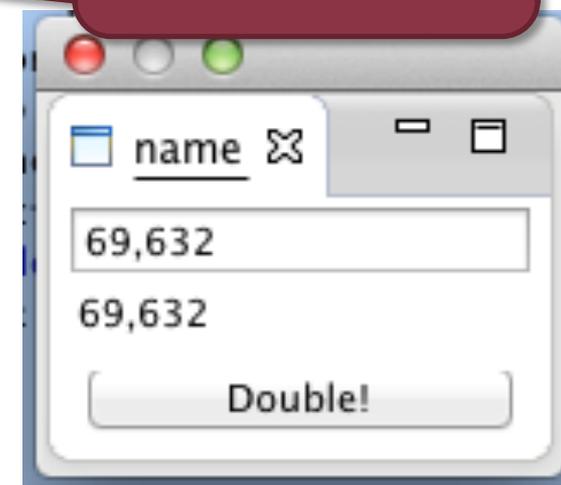
```
DataBindingContext dbc = new DataBindingContext();  
IObservableValue modelObservable =  
    BeansObservables.observeValue(model,  
dbc.bindValue(  
    SWTObservables.observeText(text, SWT.  
modelObservable,  
    null, null);
```

Binding text ...

... to model
observable ...

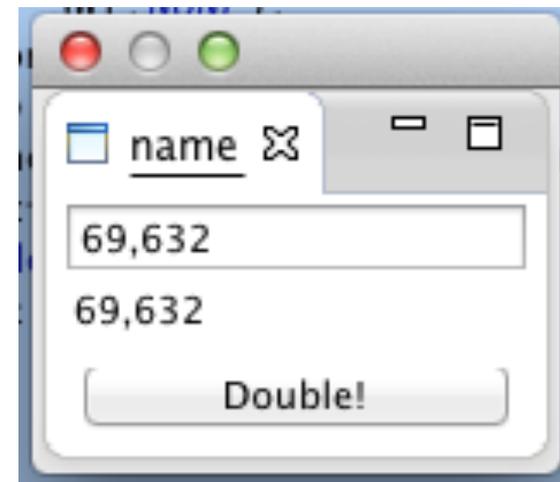
... using default
conversion

```
dbc.bindValue(  
    SWTObservables.observeText(label),  
modelObservable,  
    null, null);
```



Data Binding Example - Bindings

```
DataBindingContext dbc = new DataBindingContext();  
IObservableValue modelObservable =  
    BeansObservables.observeValue(model, "amount");  
dbc.bindValue(  
    SWTObservables.observeText(text, SWT.Modify),  
    modelObservable,  
    null, null);  
  
dbc.bindValue(  
    SWTObservables.observeText(label),  
    modelObservable,  
    null, null);
```



Data Binding Example - Bindings

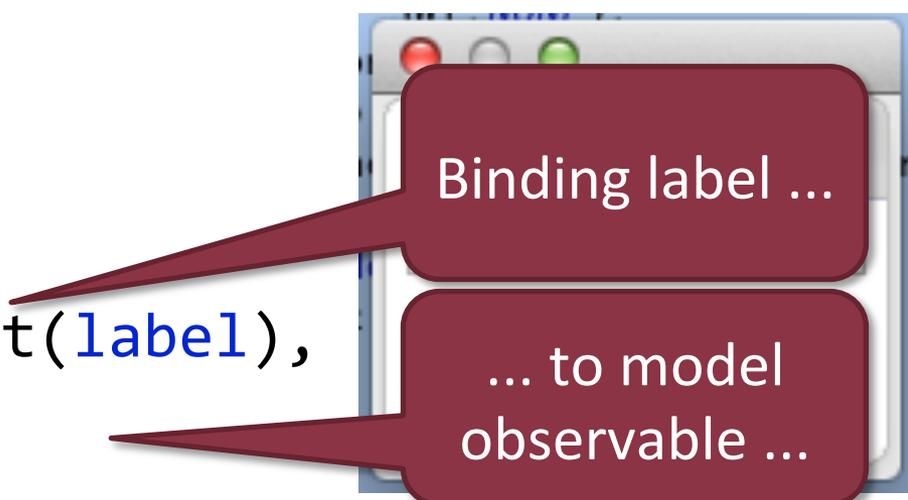
```
DataBindingContext dbc = new DataBindingContext();
IObservableValue modelObservable =
    BeansObservables.observeValue(model, "amount");
dbc.bindValue(
    SWTObservables.observeText(text, SWT.Modify),
    modelObservable,
    null, null);

dbc.bindValue(
    SWTObservables.observeText(label),
    modelObservable,
    null, null);
```



Data Binding Example - Bindings

```
DataBindingContext dbc = new DataBindingContext();  
IObservableValue modelObservable =  
    BeansObservables.observeValue(model, "amount");  
dbc.bindValue(  
    SWTObservables.observeText(text, SWT.Modify),  
    modelObservable,  
    null, null);  
  
dbc.bindValue(  
    SWTObservables.observeText(label),  
    modelObservable,  
    null, null);
```



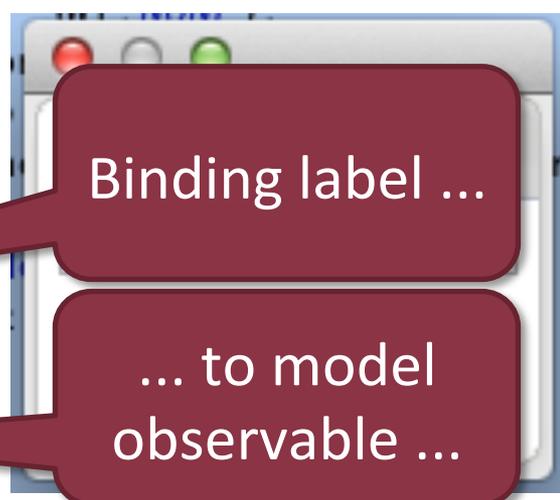
Binding label ...

... to model
observable ...

Data Binding Example - Bindings

```
DataBindingContext dbc = new DataBindingContext();  
IObservableValue modelObservable =  
    BeansObservables.observeValue(model, "amount");  
dbc.bindValue(  
    SWTObservables.observeText(text, SWT.Modify),  
    modelObservable,  
    null, null);
```

```
dbc.bindValue(  
    SWTObservables.observeText(label),  
    modelObservable,  
    null, null);
```



Binding label ...

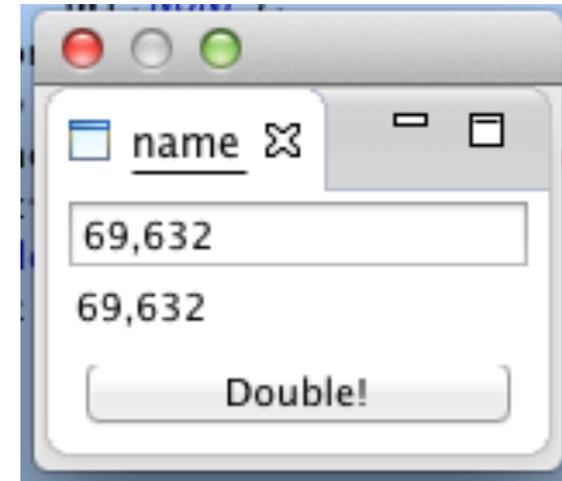
... to model
observable ...

... using default
conversion

Data Binding Example - Bindings

```
DataBindingContext dbc = new DataBindingContext();  
IObservableValue modelObservable =  
    BeansObservables.observeValue(model, "amount");  
dbc.bindValue(  
    SWTObservables.observeText(text, SWT.Modify),  
    modelObservable,  
    null, null);
```

```
dbc.bindValue(  
    SWTObservables.observeText(label),  
    modelObservable,  
    null, null);
```



Creating Observables – Factory Classes

Creating Observables – Factory Classes

- SWTObservables
 - SWT widget properties
 - Label, color, enabledness, etc.
- ViewersObservables
 - Sets up viewer contents
 - Multiple valued Observables
- BeansObservables
 - Connects to simple Java objects
 - **Notification support needed!**

Creating Observables – Factory Classes

- SWTObservables
 - SWT widget properties
 - Label, color, enabledness, etc.
- ViewersObservables
 - Sets up viewer contents
 - Multiple valued Observables
- BeansObservables
 - Connects to simple Java objects
 - **Notification support needed!**

Creating Observables – Factory Classes

- SWTObservables
 - SWT widget properties
 - Label, color, enabledness, etc.
- ViewersObservables
 - Sets up viewer contents
 - Multiple valued Observables
- BeansObservables
 - Connects to simple Java objects
 - **Notification support needed!**

Creating Observables – Factory Classes

- SWTObservables
 - SWT widget properties
 - Label, color, enabledness, etc.
- ViewersObservables
 - Sets up viewer contents
 - Multiple valued Observables
- BeansObservables
 - Connects to simple Java objects
 - **Notification support needed!**

Bean Observable - Notifications

- Use PropertyChangeSupport
 - Helps listener management
- Requires
 - Public listener registration/deregistration
 - Notifying the support class in setters
- Problems
 - Relies on conventions
 - In case of errors, no feedback is available

Bean Model

```
public class Model {  
  
    private int amount = 0;  
    public void setAmount(int newAmount) {  
  
        this.amount = newAmount;  
  
    }  
    public int getAmount() { return amount; }  
}
```

Bean Model

```
public class Model {
    private PropertyChangeSupport change = new PropertyChangeSupport(this);
    public void addPropertyChangeListener(String name,
PropertyChangeListener listener) {
        change.addPropertyChangeListener(name, listener);
    }
    public void removePropertyChangeListener(String name,
PropertyChangeListener listener) {
        change.removePropertyChangeListener(name, listener);
    }
    private int amount = 0;
    public void setAmount(int newAmount) {

        this.amount = newAmount;

    }
    public int getAmount() { return amount; }
}
```

Bean Model

```
public class Model {
    private PropertyChangeSupport change = new PropertyChangeSupport(this);
    public void addPropertyChangeListener(String name,
PropertyChangeListener listener) {
        change.addPropertyChangeListener(name, listener);
    }
    public void removePropertyChangeListener(String name,
PropertyChangeListener listener) {
        change.removePropertyChangeListener(name, listener);
    }
    private int amount = 0;
    public void setAmount(int newAmount) {

        this.amount = newAmount;

    }
    public int getAmount() { return amount; }
}
```

Listener
Management

Bean Model

```
public class Model {
    private PropertyChangeSupport change = new PropertyChangeSupport(this);
    public void addPropertyChangeListener(String name,
PropertyChangeListener listener) {
        change.addPropertyChangeListener(name, listener);
    }
    public void removePropertyChangeListener(String name,
PropertyChangeListener listener) {
        change.removePropertyChangeListener(name, listener);
    }
    private int amount = 0;
    public void setAmount(int newAmount) {

        this.amount = newAmount;

    }
    public int getAmount() { return amount; }
}
```

Bean Model

```
public class Model {
    private PropertyChangeSupport change = new PropertyChangeSupport(this);
    public void addPropertyChangeListener(String name,
PropertyChangeListener listener) {
        change.addPropertyChangeListener(name, listener);
    }
    public void removePropertyChangeListener(String name,
PropertyChangeListener listener) {
        change.removePropertyChangeListener(name, listener);
    }
    private int amount = 0;
    public void setAmount(int newAmount) {
        int oldAmount = this.amount;
        this.amount = newAmount;
        change.firePropertyChange("amount", oldAmount, newAmount);
    }
    public int getAmount() { return amount; }
}
```

Bean Model

```
public class Model {
    private PropertyChangeSupport change = new PropertyChangeSupport(this);
    public void addPropertyChangeListener(String name,
PropertyChangeListener listener) {
        change.addPropertyChangeListener(name, listener);
    }
    public void removePropertyChangeListener(String name,
PropertyChangeListener listener) {
        change.removePropertyChangeListener(name, listener);
    }
    private int amount = 0;
    public void setAmount(int newAmount) {
        int oldAmount = this.amount;
        this.amount = newAmount;
        change.firePropertyChange("amount", oldAmount, newAmount);
    }
    public int getAmount() { return amount; }
}
```

Notifying listeners
in setter

Validation

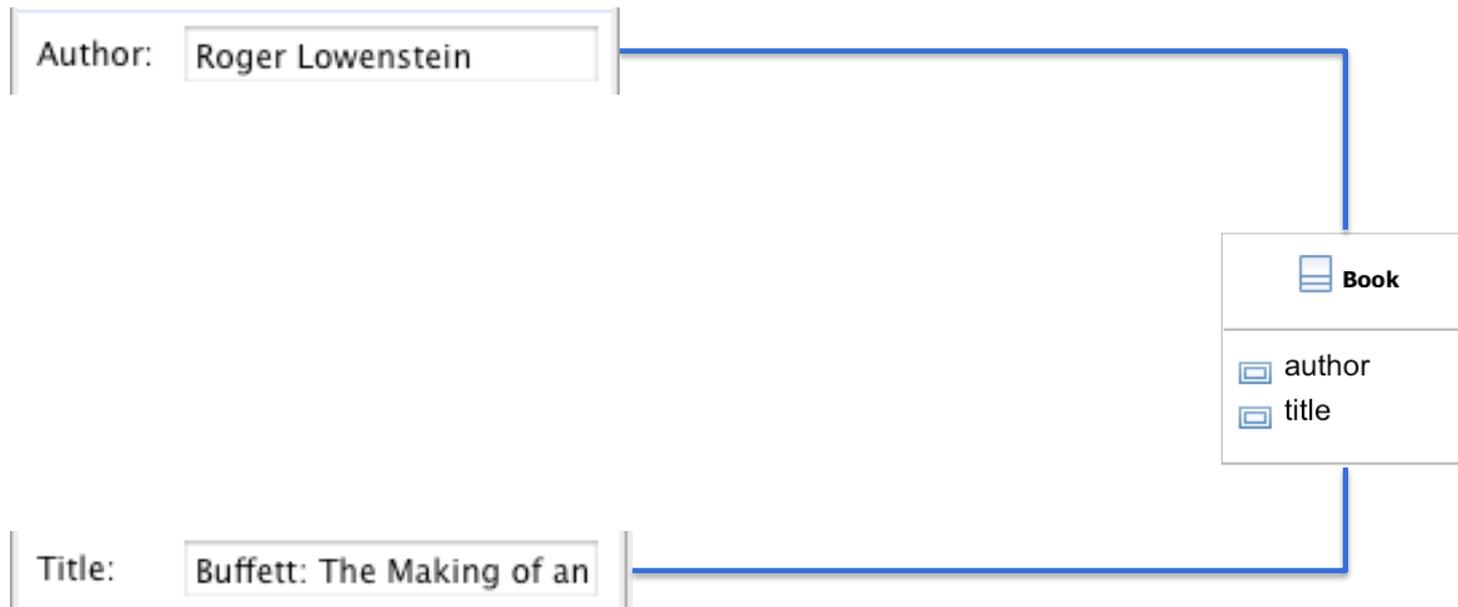
- Text-number conversion in example
 - Null parameters: default conversion and validation
- Custom validation
 - Define an UpdateValueStrategy

```
dbc.bindValue(SWTObservables.observeText(text, SWT.Modify),
    modelObservable,
    // UI to model
    new UpdateValueStrategy().
        setAfterConverValidator(anIntValidator),
    //model to UI
    new UpdateValueStrategy().
        setConverter(anIntToStringConverter);
```

UpdateValueStrategy

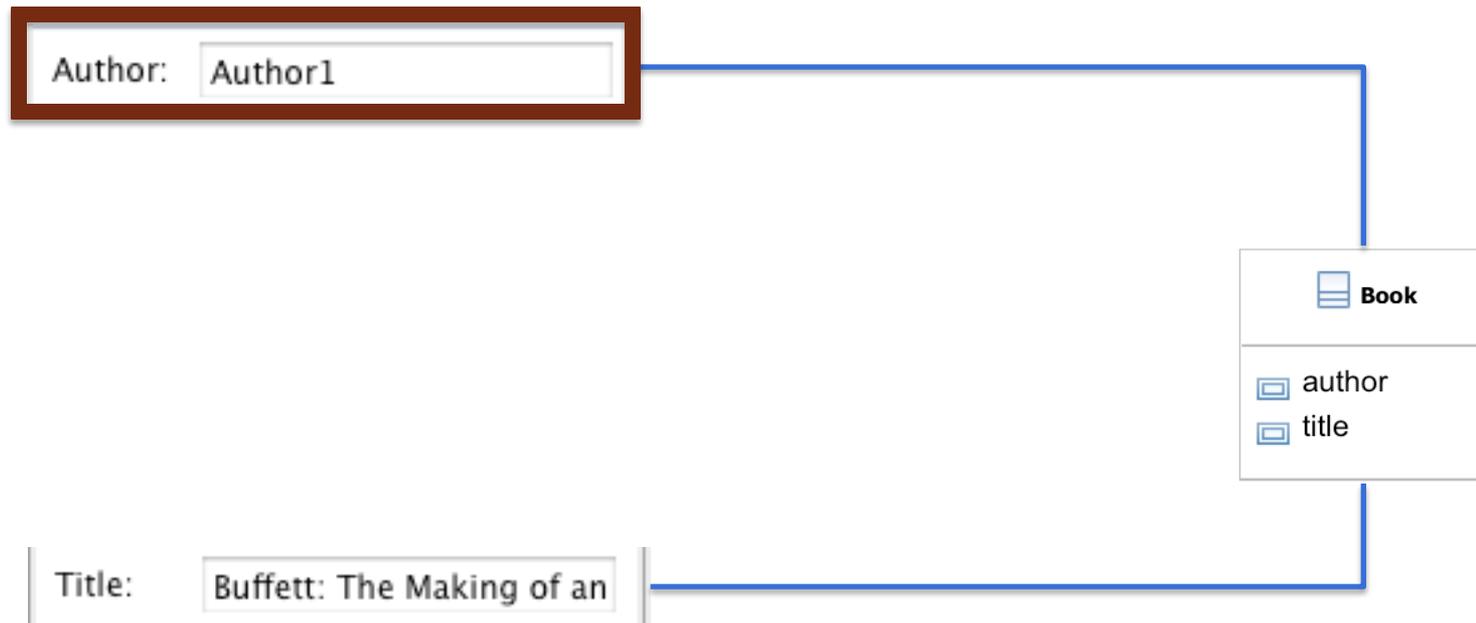
- Common object to describe validation and update
- Phases
 1. Validate after get
 - Validation after reading the value
 2. Conversion
 - Between source and target domains
 3. Validate after conversion
 - Validation after the conversion finishes
 4. Validate before set
 - Validation before writing back the value
 5. Value set
 - Writing back the value to the target object

Data binding – In Action



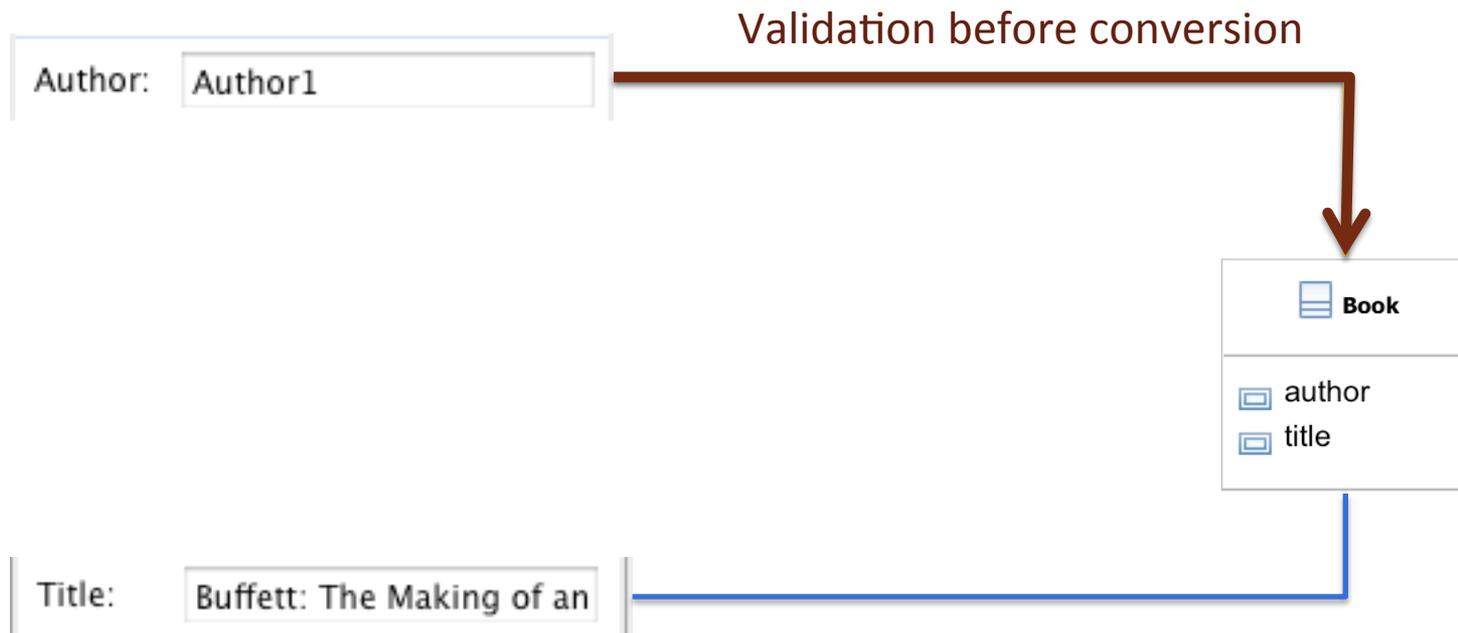
Both text field is bound to the same Book object

Data binding – In Action

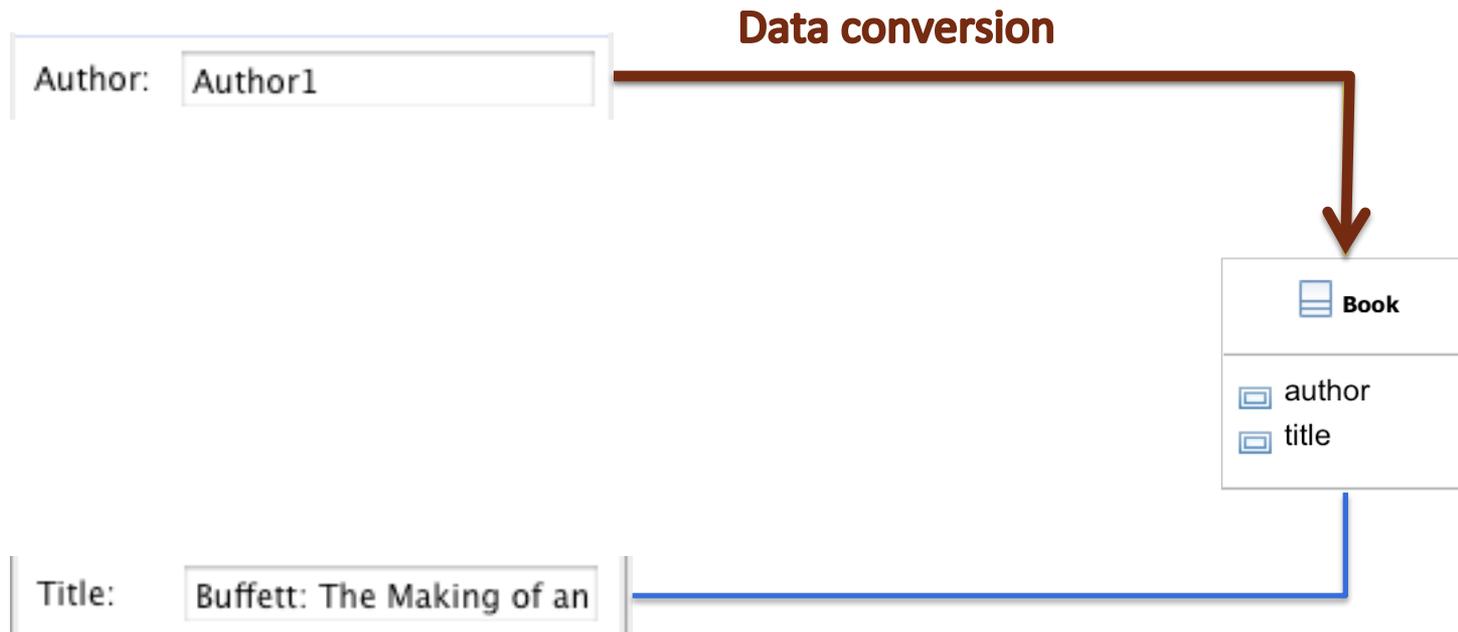


Author changed

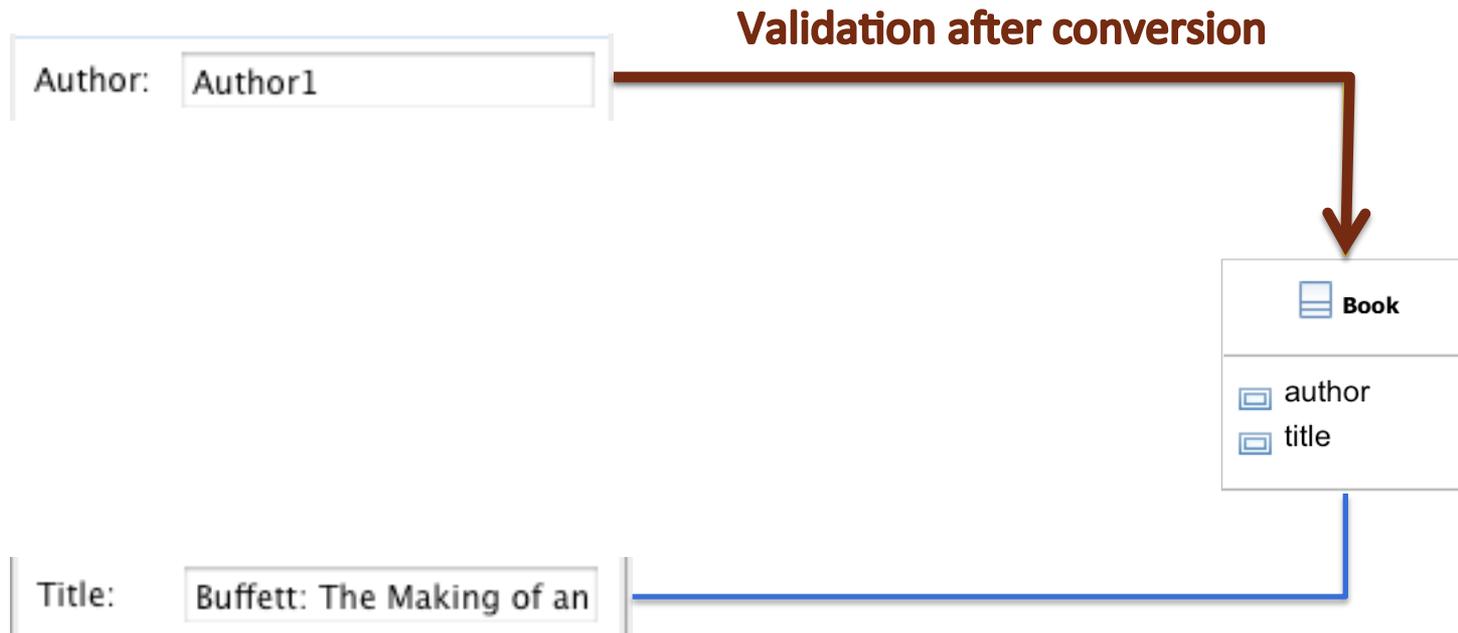
Data binding – In Action



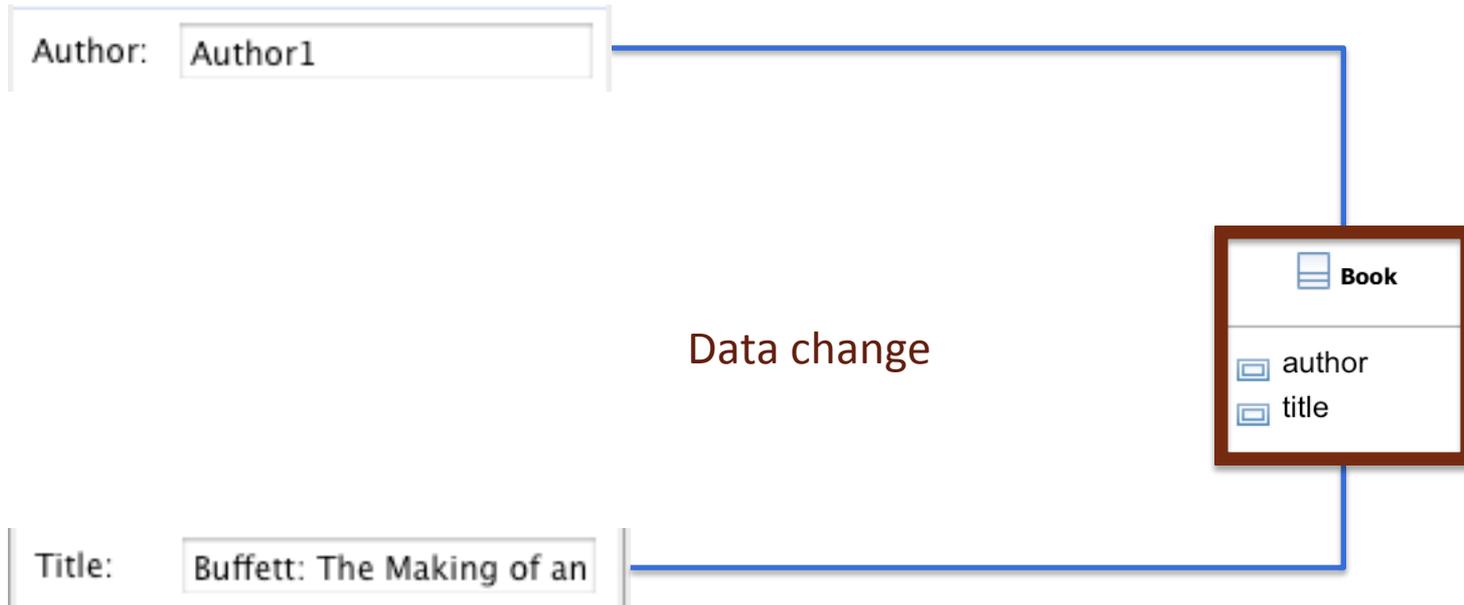
Data binding – In Action



Data binding – In Action

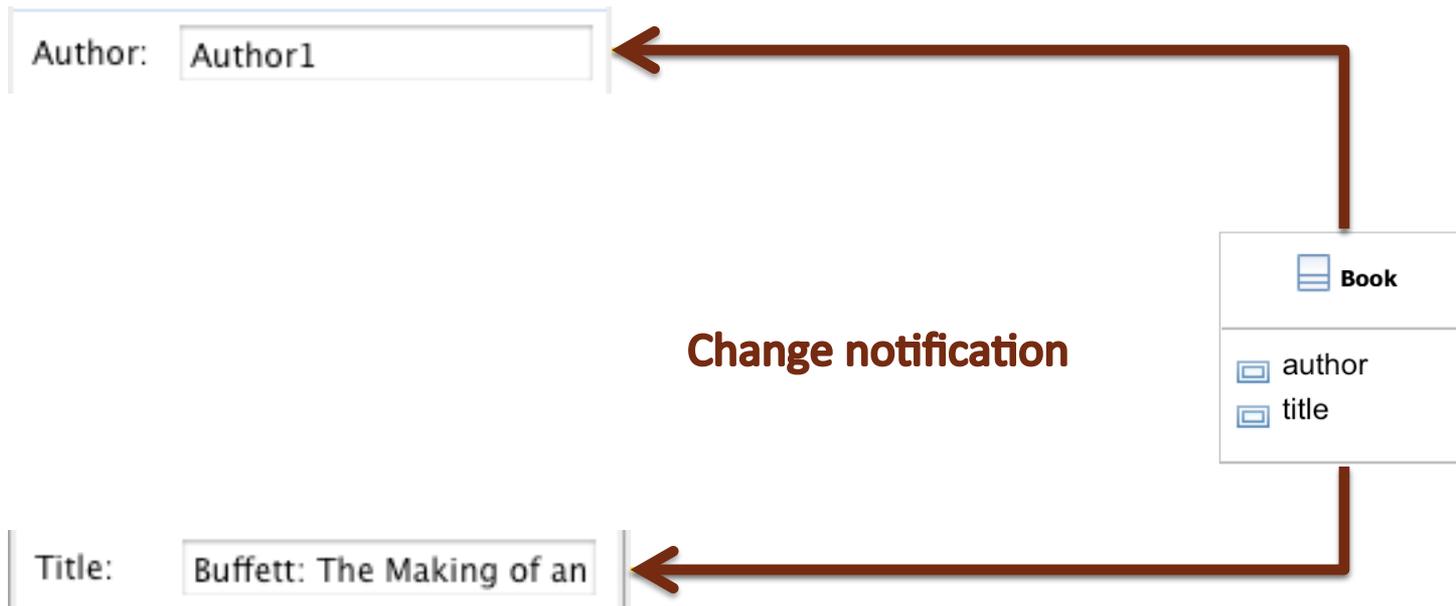


Data binding – In Action

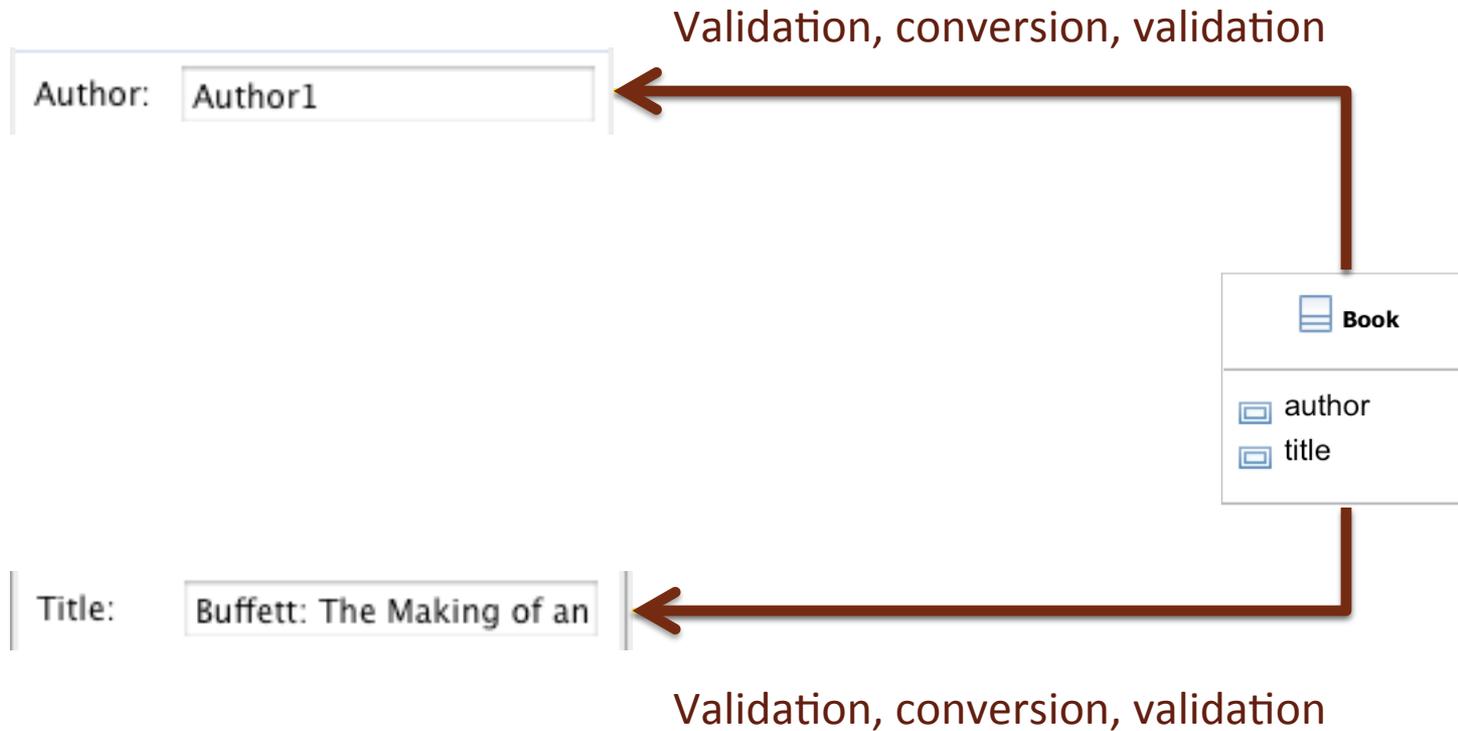


Author changed

Data binding – In Action



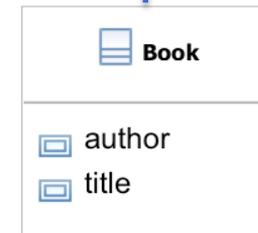
Data binding – In Action



Data binding – In Action

Presentation of change

Author:



Presentation of change

Title:

Writing Validators

- Use interface *IValidator*
- Return value:
 - `ValidationStatus.ok()`
 - `ValidationStatus.info(String message)`
 - `ValidationStatus.warning(String message)`
 - `ValidationStatus.error(String message)`

Writing Converters

- Use interface IConverter
 - Define source and target classes
 - Manages the real conversion
 - Built-in converters available
 - NumberToStringConverter
 - StringToNumberConverter

Presenting Validation Errors

- Error information
 - Available as `IObservable`
 - Boundable to the user interface
- Observing all binding errors:
 - `new AggregateValidationStatus(dbc.getBindings(), AggregateValidationStatus.MAX_SEVERITY), null, null);`
- Observable of a selected binding `b`
 - `b.getValidationStatus()`

Computed Observable Values

- Attribute changes are managed
 - Can be used to calculate computed values
 - ComputedValue, ComputedList change accordingly
- Example for computed value – full name

```
final IObservableValue firstName =
    SWTObservables.observeText(firstNameField, SWT.Modify);

final IObservableValue lastName =
    SWTObservables.observeText(lastNameField, SWT.Modify);

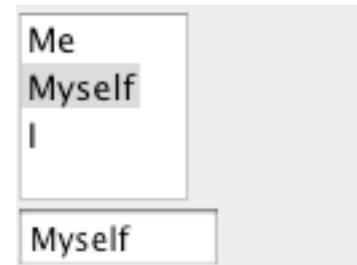
IObservableValue formattedName = new ComputedValue() {
    protected Object calculate() {
        return lastName.getValue() + firstName.getValue();
    }
};
```

Master-Detail Binding

- Task:
 - Observing (and binding) the current selection
 - If selection changes
 - Old bindings are to be disposed
 - New bindings are to be created
- Support: Master-Detail Binding
 - Define observable for selection (Master Observable)
 - Observe selection (observeDetailValue)

Example: Master-Detail Binding

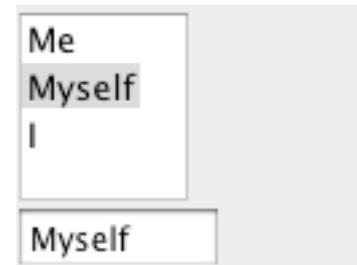
```
// 1. Observe changes in selection.
IObservableValue selection =
    ViewersObservables.observeSingleSelection(viewer);
// 2. Observe the name property of the current selection.
IObservableValue detailObservable =
    BeansObservables.observeDetailValue(selection, "name",
    String.class);
// 3. Bind the Text widget to the name detail (selection's
    name).
new DataBindingContext().bindValue(
    SWTObservables.observeText(name, SWT.None),
    detailObservable,
    new UpdateValueStrategy(false,
        UpdateValueStrategy.POLICY_NEVER),
    null);
```



Example: Master-Detail Binding

```
// 1. Observe changes in selection.
IObservableValue selection =
    ViewersObservables.observeSingleSelection(viewer);
// 2. Observe the name property of the current selection.
IObservableValue detailObservable =
    BeansObservables.observeDetailValue(
        String.class);
// 3. Bind the Text widget to the name
    name).
new DataBindingContext().bindValue(
    SWTObservables.observeText(name, SWT.None),
    detailObservable,
    new UpdateValueStrategy(false,
        UpdateValueStrategy.POLICY_NEVER),
    null);
```

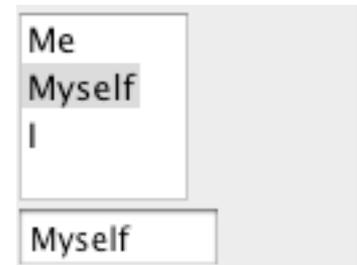
Create observer for
single selection
(Viewer)



Example: Master-Detail Binding

```
// 1. Observe changes in selection
IObservableValue selection =
    ViewersObservables.observeSelection(viewer);
// 2. Observe the name of the current selection.
IObservableValue detailObservable =
    BeansObservables.observeDetailValue(selection, "name",
    String.class);
// 3. Bind the Text widget to the name detail (selection's
    name).
new DataBindingContext().bindValue(
    SWTObservables.observeText(name, SWT.None),
    detailObservable,
    new UpdateValueStrategy(false,
        UpdateValueStrategy.POLICY_NEVER),
    null);
```

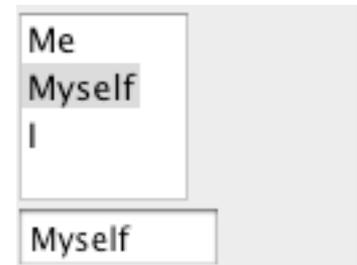
Creating Detail
Observable



Example: Master-Detail Binding

```
// 1. Observe changes in selection.
IObservableValue selection =
    ViewersObservables.observeSingleSelection(viewer);
// 2. Observe the name property of the current selection.
IObservableValue detailObservable =
    BeansObservables.observeDetailValue(selection, "name",
    String.class);
// 3. Bind the Text widget to the name property of the selection's
    name).
new DataBindingContext().bindValue(
    SWTObservables.observeText(name, SWT.None),
    detailObservable,
    new UpdateValueStrategy(false,
        UpdateValueStrategy.POLICY_NEVER),
    null);
```

Binding



Data Binding - Summary

- High level data synchronization
 - Different binding available (e.g. EMF)
- Documentation available on Eclipse Wiki:
http://wiki.eclipse.org/index.php/JFace_Data_Binding

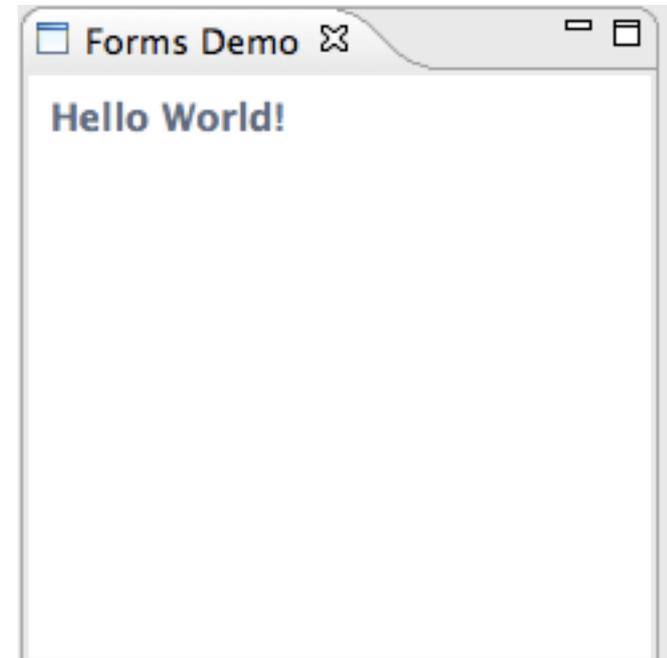
Forms API

Forms API

- Form definition API
 - Uniform appearance
 - Based on SWT and JFace
- New widgets
 - Sections
 - Layouts
 - Scroll support
 - ...
- Useful in
 - Dialogs, views, editors

Forms Hello, world!

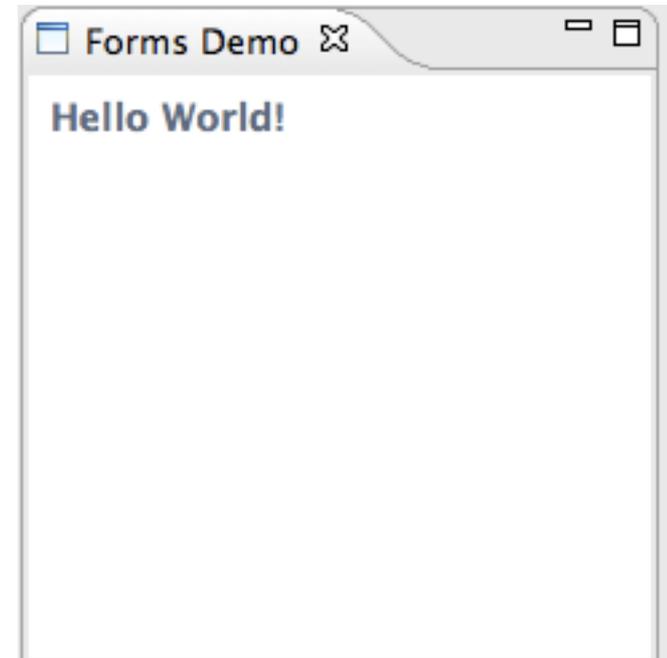
```
FormToolkit toolkit;  
Form form;  
  
public void createPartControl(Composite parent) {  
    toolkit = new FormToolkit(parent.getDisplay());  
  
    form= toolkit.createForm(parent);  
    form.setText("Hello World!");  
}  
  
@Override  
public void setFocus() {  
    form.setFocus();  
}  
  
@Override  
public void dispose(){  
    toolkit.dispose();  
    super.dispose();  
}
```



Forms Hello, world!

```
FormToolkit toolkit;  
Form form;  
  
public void createPartControl(Composite parent) {  
    toolkit = new FormToolkit(parent.getDisplay());  
  
    form= toolkit.createForm(parent);  
    form.setText("Hello World");  
}  
  
@Override  
public void setFocus() {  
    form.setFocus();  
}  
  
@Override  
public void dispose(){  
    toolkit.dispose();  
    super.dispose();  
}
```

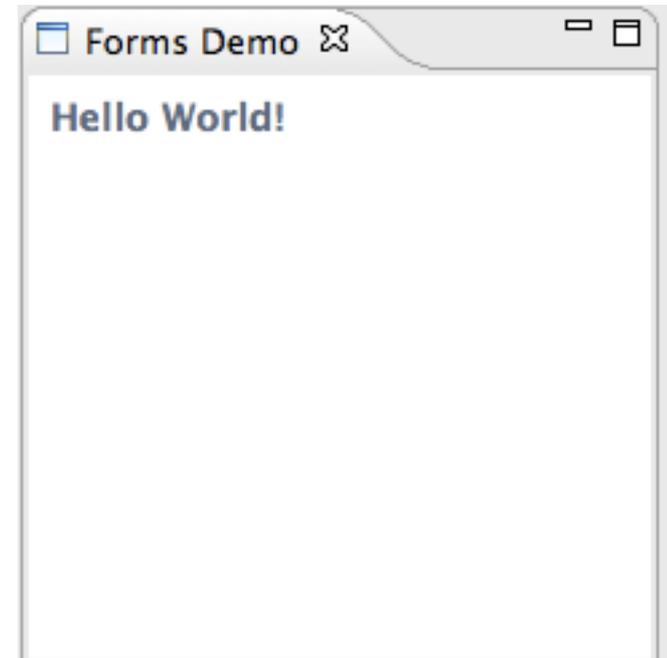
Initialize
factory



Forms Hello, world!

```
FormToolkit toolkit;  
Form form;  
  
public void createPartControl(Composite parent) {  
    toolkit = new FormToolkit(parent.getDisplay());  
  
    form= toolkit.createForm(parent);  
    form.setText("Hello World!");  
}  
  
@Override  
public void setFocus()  
    form.setFocus();  
}  
  
@Override  
public void dispose(){  
    toolkit.dispose();  
    super.dispose();  
}
```

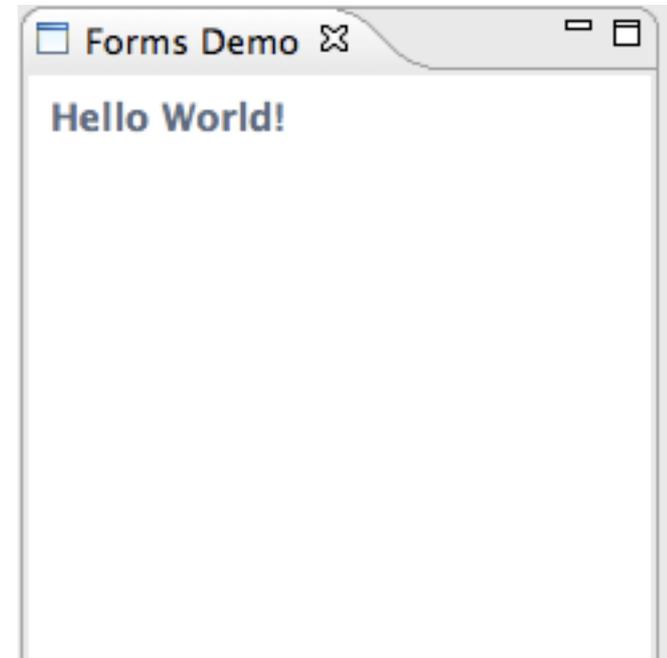
Initialize form



Forms Hello, world!

```
FormToolkit toolkit;  
Form form;  
  
public void createPartControl(Composite parent) {  
    toolkit = new FormToolkit(parent.getDisplay());  
  
    form= toolkit.createForm(parent);  
    form.setText("Hello World!");  
}  
  
@Override  
public void setFocus() {  
    form.setFocus();  
}  
  
@Override  
public void dispose() {  
    toolkit.dispose();  
    super.dispose();  
}
```

Dispose toolkit
(important!)



Structure of Forms

■ Head composite

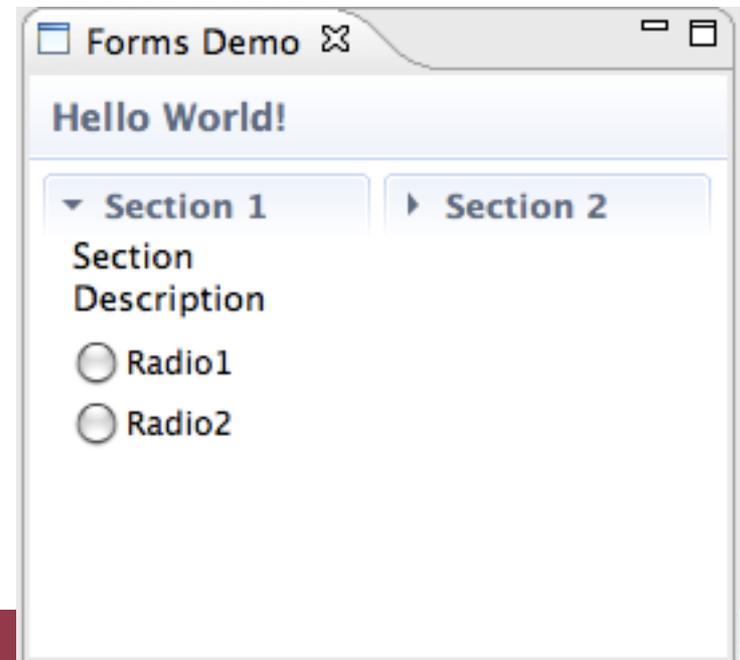
- Title area
- Warnings/errors usually displayed here
- Predefined decoration available
 - `toolkit.decorateFormHeading(form);`

■ Body composite

- Real body
- No preset layout
- Place your widgets here

Sections

- Composite
 - Heading
 - Textual description below title
 - Separator between header and contents
- Can be collapsed (via style bits)
 - EXPANDED/COLLAPSED



SectionExample

```
public void createPartControl(Composite parent) {  
    ...  
    Section section1 =  
        toolkit.createSection(form.getBody(),  
            Section.DESCRPTION|Section.TITLE_BAR|  
            Section.EXPANDED);  
    section1.setText("Section 1");  
    section1.setDescription("Section Description");  
    Composite sectionClient1 =  
        toolkit.createComposite(section1);  
    sectionClient1.setLayout(new GridLayout());  
    section1.setClient(sectionClient1);  
    toolkit.createButton(sectionClient1, "Radio1",  
        SWT.RADIO);  
    toolkit.createButton(sectionClient1, "Radio2",  
        SWT.RADIO);  
    ...  
}
```

SectionExample

```
public void createPartControl(Composite parent) {  
    ...  
    Section section1 =  
        toolkit.createSection(form.getBody(),  
            Section.DESCRPTION|Section.TITLE|Section.BAR|  
            Section.EXPANDED);  
    section1.setText("Section 1");  
    section1.setDescription("Section 1");  
    Composite sectionClient1 =  
        toolkit.createComposite(section1);  
    sectionClient1.setLayout(new GridLayout());  
    section1.setClient(sectionClient1);  
    toolkit.createButton(sectionClient1, "Radio1",  
        SWT.RADIO);  
    toolkit.createButton(sectionClient1, "Radio2",  
        SWT.RADIO);  
    ...  
}
```

Create section

SectionExample

```
public void createPartControl(Composite parent) {  
    ...  
    Section section1 =  
        toolkit.createSection(parent.getBody(),  
            Section.DESCRIBED, Section.TITLE_BAR|  
            Section.EXPANDED);  
    section1.setText("Section Title");  
    section1.setDescription("Section Description");  
    Composite sectionClient1 =  
        toolkit.createComposite(section1);  
    sectionClient1.setLayout(new GridLayout());  
    section1.setClient(sectionClient1);  
    toolkit.createButton(sectionClient1, "Radio1",  
        SWT.RADIO);  
    toolkit.createButton(sectionClient1, "Radio2",  
        SWT.RADIO);  
    ...  
}
```

New
composite for
contents!

SectionExample

```
public void createPartControl(Composite parent) {  
    ...  
    Section section1 =  
        toolkit.createSection(form.getBody(),  
            Section.DESCRPTION|Section.TITLE_BAR|  
            Section.EXPANDED);  
    section1.setText("Section description");  
    section1.setDescription("Section description");  
    Composite sectionClient1 =  
        toolkit.createComposite(parent, section1);  
    sectionClient1.setLayout(new GridLayout());  
    section1.setClient(sectionClient1);  
    toolkit.createButton(sectionClient1, "Radio1",  
        SWT.RADIO);  
    toolkit.createButton(sectionClient1, "Radio2",  
        SWT.RADIO);  
    ...  
}
```

Section stores new
Composite

Instantiating Widgets

■ SWT widgets

- `FormToolkit.create«Widgetname»()`
- Style bits also supported

```
form.getBody().setLayout(new RowLayout());  
Button button = toolkit.createButton(form.getBody(), "My Checkbox",  
    SWT.CHECK);
```

■ JFace viewer

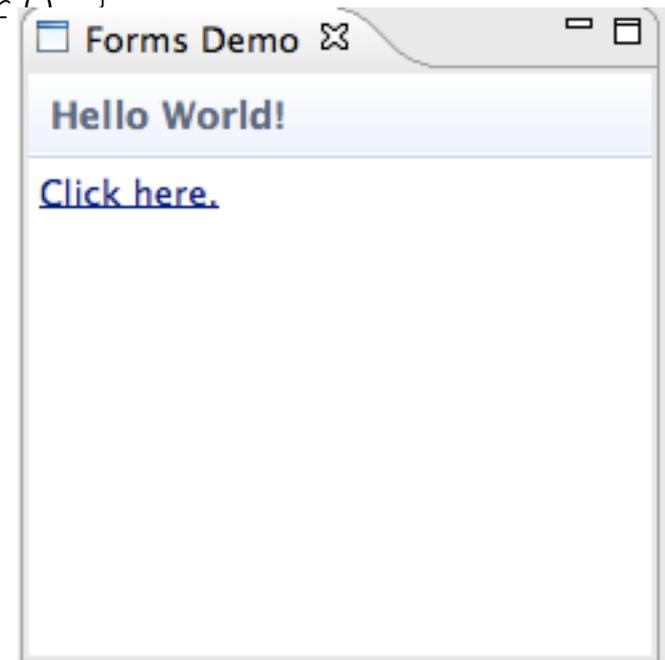
- Create corresponding SWT widget using Factory
- Create JFace wrapper around it

```
Table table = toolkit.createTable(form.getBody(), SWT.NONE);  
TableViewer viewer = new TableViewer(table);
```

Form elements - Hyperlink

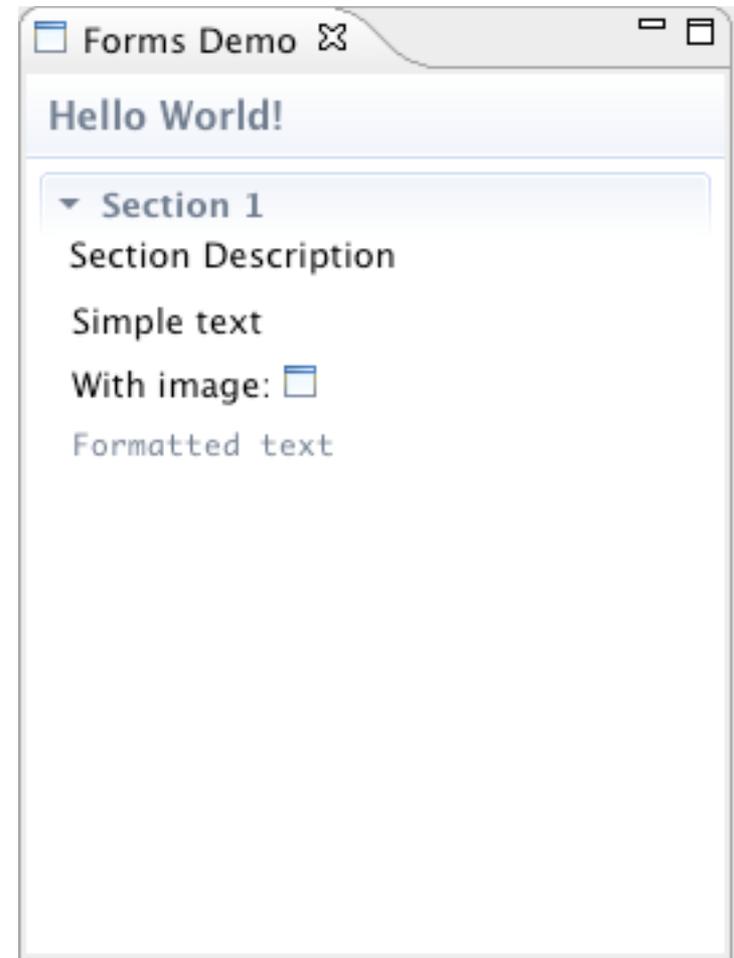
- Define links between for elements
 - href property: target of linke
 - Or use HyperlinkListener

```
Hyperlink link = toolkit.createHyperlink(form.getBody(),  
    "Click here.", SWT.WRAP);  
link.addHyperlinkListener(new HyperlinkAdapter() {  
    public void linkActivated(HyperlinkEvent e)  
        System.out.println("Link activated!");  
    }  
});
```



Form elements - FormText

- Handles text, icons and links
 - Difficult with separate widgets
- HTML-like rich text
 - HTML labels
 - Icons/fonts loaded manually
- Not editable



FormText example

```
StringBuffer buf = new StringBuffer();
buf.append("<form>");
buf.append("<p>Simple text</p>");
buf.append("<p>With image: <img href=\"image\" /></p>");
buf.append("<p><span color=\"sp\" font=\"sp\">Formatted
  text</span></p>");
buf.append("</form>");
```

```
FormText text = toolkit.createFormText(sectionClient1,
  true);
text.setImage("image",
  PlatformUI.getWorkbench().getSharedImages()
  .getImage(ISharedImages.IMG_DEF_VIEW));
text.setColor("sp",
  toolkit.getColors().getColor(IFormColors.TITLE));
text.setFont("sp", JFaceResources.getTextFont());
text.setText(buf.toString(), true, true);
```

FormText example

```
StringBuffer buf = new StringBuffer();
buf.append("<form>");
buf.append("<p>Simple text</p>");
buf.append("<p>With image: <img href=\"image\" /></p>");
buf.append("<p><span color=\"sp\" font=\"sp\">Formatted  
text</span></p>");
buf.append("</form>");
```

Assemble text

```
FormText text = toolkit.createFormText(sectionClient1,  
    true);  
text.setImage("image",  
    PlatformUI.getWorkbench().getSharedImages()  
    .getImage(ISharedImages.IMG_DEF_VIEW));  
text.setColor("sp",  
    toolkit.getColors().getColor(IFormColors.TITLE));  
text.setFont("sp", JFaceResources.getTextFont());  
text.setText(buf.toString(), true, true);
```

FormText example

```
StringBuffer buf = new StringBuffer();
buf.append("<form>");
buf.append("<p>Simple text</p>");
buf.append("<p>With image: <img href=\"image\" /></p>");
buf.append("<p><span color=\"sp\" font=\"sp\">Formatted
  text</span></p>");
buf.append("</form>");
```

Map SWT image to
name

```
FormText text = toolkit.createFormText(sectionClient1,
  true);
text.setImage("image",
  PlatformUI.getWorkbench().getSharedImages()
  .getImage(ISharedImages.IMG_DEF_VIEW));
text.setColor("sp",
  toolkit.getColors().getColor(IFormColors.TITLE));
text.setFont("sp", JFaceResources.getTextFont());
text.setText(buf.toString(), true, true);
```

FormText example

```
StringBuffer buf = new StringBuffer();
buf.append("<form>");
buf.append("<p>Simple text</p>");
buf.append("<p>With image: <img href=\"image\" /></p>");
buf.append("<p><span color=\"sp\" font=\"sp\">Formatted
  text</span></p>");
buf.append("</form>");
```

```
FormText text = toolkit.createFormText(Shell.getClient1,
  true);
text.setImage("image",
  PlatformUI.getWorkbench().getImages().getImage(ISharedImages.IMC_PREF_VIEW));
text.setColor("sp",
  toolkit.getColors().getColor(IFormColors.TITLE));
text.setFont("sp", JFaceResources.getTextFont());
text.setText(buf.toString(), true, true);
```

Map SWT Color to
name

FormText example

```
StringBuffer buf = new StringBuffer();
buf.append("<form>");
buf.append("<p>Simple text</p>");
buf.append("<p>With image: <img href=\"image\" /></p>");
buf.append("<p><span color=\"sp\" font=\"sp\">Formatted
  text</span></p>");
buf.append("</form>");
```

```
FormText text = toolkit.createFormText(sectionClient1,
  true);
text.setImage("image",
  PlatformUI.getWorkbench().getImages().
  .getImage(ISharedImages.I_SHARED_IMAGE))
text.setColor("sp",
  toolkit.getColors().getColor(IFormColors.TITLE));
text.setFont("sp", JFaceResources.getTextFont());
text.setText(buf.toString(), true, true);
```

Map SWT Font to
name

FormText example

```
StringBuffer buf = new StringBuffer();
buf.append("<form>");
buf.append("<p>Simple text</p>");
buf.append("<p>With image: <img href=\"image\" /></p>");
buf.append("<p><span color=\"sp\" font=\"sp\">Formatted
    text</span></p>");
buf.append("</form>");
```

```
FormText text = toolkit.createFormText(sectionClient1,
    true);
text.setImage("image",
    PlatformUI.getWorkbench().getSharedImages()
        .getImage(ISharedImages.IMC...));
text.setColor("sp",
    toolkit.getColors().getColor(...));
text.setFont("sp", JFaceResources.getTextFont());
text.setText(buf.toString(), true, true);
```

2th parameter: Rich
text

FormText example

```
StringBuffer buf = new StringBuffer();
buf.append("<form>");
buf.append("<p>Simple text</p>");
buf.append("<p>With image: <img href=\"image\" /></p>");
buf.append("<p><span color=\"sp\" font=\"sp\">Formatted
  text</span></p>");
buf.append("</form>");
```

```
FormText text = toolkit.createFormText(sectionClient1,
  true);
text.setImage("image",
  PlatformUI.getWorkbench().getSharedImages()
  .getImage(ISharedImages.IMG_DEF_VI
text.setColor("sp",
  toolkit.getColors().getColor(IFo
text.setFont("sp", JFaceResources.getFont());
text.setText(buf.toString(), true, true);
```

3rd parameter:
automatic links

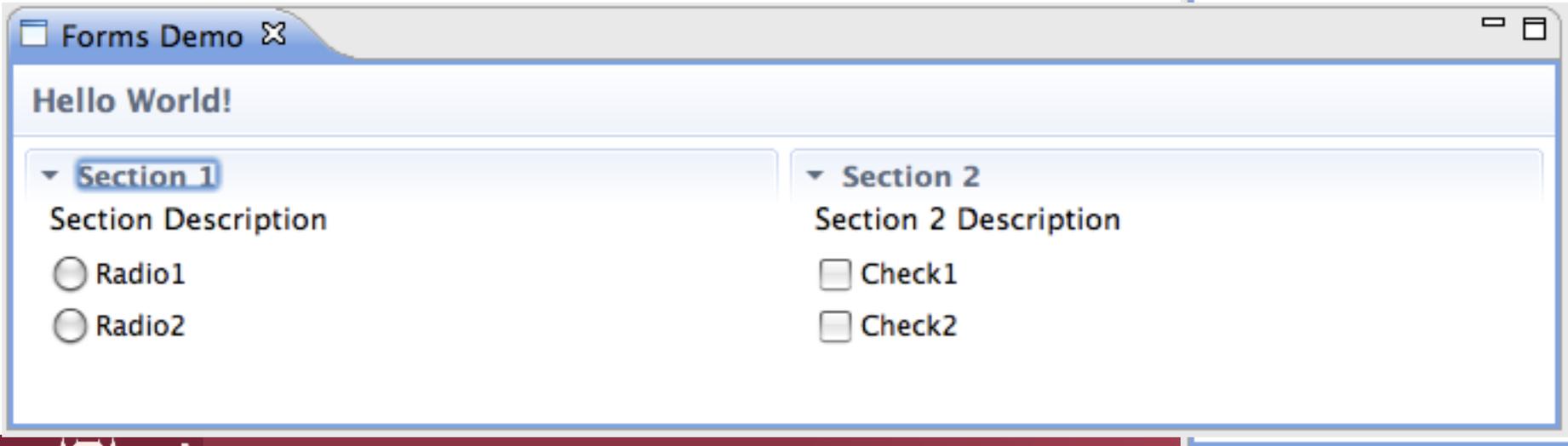
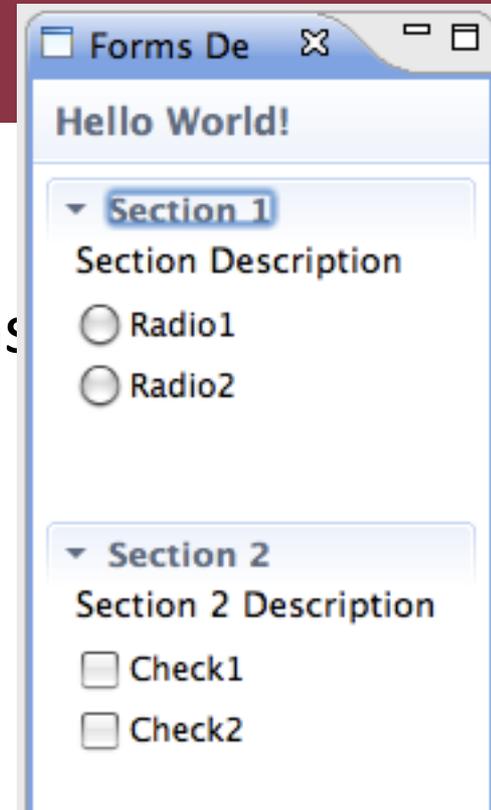
TableWrapLayout

- Similar to GridLayout
 - But: goal is to wrap labels
- TableWrapData layout data objects
 - Size hints

```
TableWrapLayout layout = new TableWrapLayout();
form.getBody().setLayout(layout);
layout.numColumns = 2;
Hyperlink link = toolkit.createHyperlink(form.getBody(), "This
    is an example of a form that is much longer and will need to
    wrap.", SWT.WRAP);
TableWrapData td = new TableWrapData();
td.colspan = 2;
link.setLayoutData(td);
```

Column layout

- Similar to RowLayout
 - But: Balances content between columns
 - But: Varied number of columns
- To define
 - Minimal number of columns (def.: 1)
 - Maximum number of columns (def.: 3)



Forms API – Additional Capabilities

- Form validation
 - Mark widgets with error markers
- Extended headings
 - Error messages
 - Toolbars
 - Dropdown menus
- Multipage editor
 - Look at the plug-in manifest editor in PDE

Forms API - Summary

- Complex form handling
 - Reuse of SWT/JFace widgets
 - New widgets/layouts
 - Uniform presentation

Additional GUI Support

Further GUI Projects

- XWT project
 - Experimental, XML-based form design
- Sapphire project
 - High-level definition of complex forms
 - Relies on annotated Java objects
- JavaFX support
 - SWT over JavaFX under construction
 - Launch workbench directly over JavaFX (Eclipse 4.x)
 - GEF4 support