**5.1. Purpose of the JFace viewer framework**

The JFace viewer framework allows you to display a domain model in a standard SWT widget like list, combo, tree or table without converting the domain model beforehand.

A viewer allows you to set a *content provider* which provides the data for the viewer. The content provider makes no assumption about the presentation of the data model.

You can also assign at least one *label provider* to a viewer. The label provider defines how the data from the model will be displayed in the viewer.

**5.2. Standard JFace viewer**

JFace provides several standard viewer implementations. These viewers are part of the org.eclipse.jface.viewers package. The following list contains the most important ones.

* ComboViewer
* ListViewer
* TreeViewer
* TableViewer

**5.3. Standard content and label provider**

The related interfaces for defining a content provider are described in the following table.

| *Table 1. Content providers* | | |
| --- | --- | --- |
| **Interface** | **Default implementation** | **Description** |
| IStructuredContentProvider | ArrayContentProvider | Used for the List-, Combo- and TableViewer. JFace provides a default implementation for Collections and Arrays with the ArrayContentProvider class. Because the ArrayContentProvider class does not store any data, it is possible to share an instance with several viewers. To get a shared instance use the ArrayContentProvider.getInstance() method. |
| ITreeContentProvider | Not available | Used for the TreeViewer class. Has additional methods to determine the children and the parents of the elements. |

Important standard label providers are listed in the following table.

| *Table 2. Label providers* | | |
| --- | --- | --- |
| **Required class** | **Standard label providers** | **Description** |
| ILabelProvider | LabelProvider | Used for lists and trees, can return an icon and a label per element. |
| CellLabelProvider | ColumnLabelProvider | Used for tables. Defines a label provider per column. |

**6. JFace ComboViewer**

The ComboViewer class simplifies the implementation of a SWT Combo (Drop-down box).

Assume the folloiwng data model.

**public** **class** **Person** **{**

**private** **String** firstName**;**

**private** **String** lastName**;**

**private** **boolean** married**;**

**public** Person**(String** firstName**,** **String** lastName**)** **{**

**this.**firstName **=** firstName**;**

**this.**lastName **=** lastName**;**

**}**

**public** **String** getFirstName**()** **{**

**return** firstName**;**

**}**

**public** **void** setFirstName**(String** firstName**)** **{**

**this.**firstName **=** firstName**;**

**}**

**public** **String** getLastName**()** **{**

**return** lastName**;**

**}**

**public** **void** setLastName**(String** lastName**)** **{**

**this.**lastName **=** lastName**;**

**}**

**public** **boolean** isMarried**()** **{**

**return** married**;**

**}**

**public** **void** setMarried**(boolean** married**)** **{**

**this.**married **=** married**;**

**}**

**}**

The following example snippet shows you how you could use this given data model in a ComboViewer. The listener is is notified whenever the selection of the viewer changes.

*// the following code is executed by the method which*

*// creates the user interface*

*// assumes parent is an SWT Composite*

**GridLayout** layout **=** **new** **GridLayout(2,** **false);**

parent**.**setLayout**(**layout**);**

**Label** label **=** **new** **Label(**parent**,** **SWT.**NONE**);**

label**.**setText**("Select a person:");**

**final** **ComboViewer** viewer **=** **new** **ComboViewer(**parent**,** **SWT.**READ\_ONLY**);**

*// the ArrayContentProvider object does not store any state,*

*// therefore, you can re-use instances*

viewer**.**setContentProvider**(ArrayContentProvider.**getInstance**());**

viewer**.**setLabelProvider**(new** **LabelProvider()** **{**

**@Override**

**public** **String** getText**(Object** element**)** **{**

**if** **(**element **instanceof** **Person)** **{**

**Person** person **=** **(Person)** element**;**

**return** person**.**getFirstName**();**

**}**

**return** **super.**getText**(**element**);**

**}**

**});**

**List<Person>** persons **=** **Arrays.**asList**(new** **Person("Lars",** **"Vogel"),**

**new** Person**("Tim",** **"Taler"),** **new** **Person("Jim",** **"Knopf"));**

*// set the input of the Viewer,*

*// this input is send to the content provider*

*// must be called after setContentProvider*

viewer**.**setInput**(**persons**);**

*//react to the selection change of the viewer*

*//note that the viewer returns the actual object*

viewer**.**addSelectionChangedListener**(new** **ISelectionChangedListener()** **{**

**@Override**

**public** **void** selectionChanged**(SelectionChangedEvent** event**)** **{**

**IStructuredSelection** selection **=** **(IStructuredSelection)** event

**.**getSelection**();**

**if** **(**selection**.**size**()** **>** **0){**

**System.**out**.**println**(((Person)** selection**.**getFirstElement**())**

**.**getLastName**());**

**}**

**}**

**});**

You can get and set selections using Java objects based on your domain model.

*// you can select an object directly via the domain object*

**Person** person **=** persons**[0];**

viewer**.**setSelection**(new** **StructuredSelection(**person**));**

*// retrieves the selection, returns the data model object*

**IStructuredSelection** selection **=** viewer**.**getStructuredSelection**();**

**Person** p **=** **(Person)** selection**.**getFirstElement**();**