Eclipse Layouts [1, 2]

by Shantha Ramachandran

Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada

Last revised: June 4, 2003

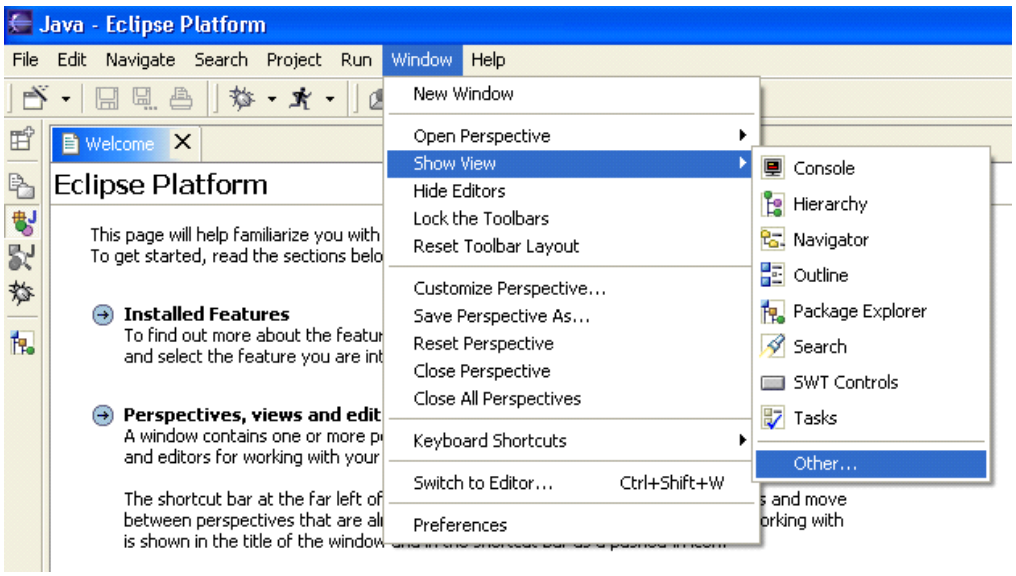
Overview:

In this section, we will be discussing the four different types of layout managers you can use for designing a GUI. These four types are FillLayout, RowLayout, GridLayout and FormLayout. A detailed specification on how to use the different layout managers can be found on the eclipse website, in the article entitled Understanding Layouts in SWT. This article can be found at the following address: www.eclipse.org/articles/Understanding Layouts/Understanding Layouts.htm. Before reading further or running any examples, it is advised that you read this article, as it is very informative and covers the basic function of all four layout managers.

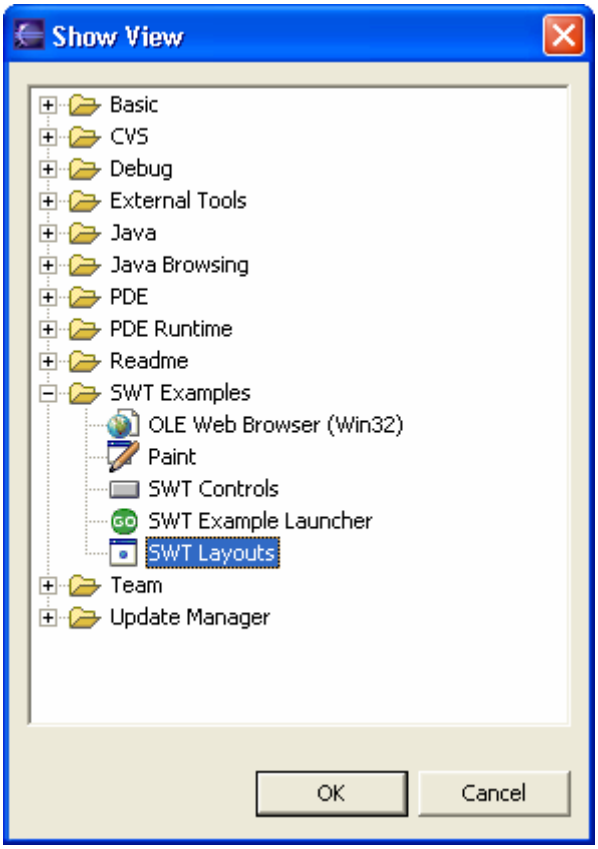
Layout Example:

Eclipse offers a plugin to preview how a shell would look using different layouts, called the Layout Example. To run this example, go to the Window menu in Eclipse, and select Show View, then select Other.

[1] This work was funded by an IBM Eclipse Innovation Grant.



From the screen that appears, expand the SWT Examples folder, and choose SWT Layouts. Click OK.



The Layout Example will appear, and you can play around with it as you wish. Clicking on the Code button will generate sample code for the layouts that you have defined.

This next section will only cover how to use the layout managers very briefly. As stated above, for more information, you should see the article, Understanding Layouts in SWT. However, this section may be useful to help understand where and when to use the different layout managers, and how they can be useful to you.

FillLayout:

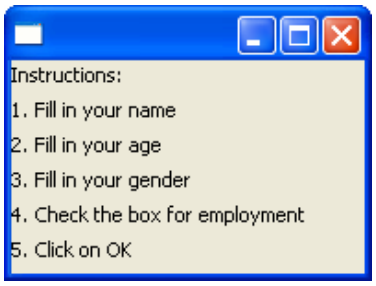
FillLayout is the most basic of all the layout managers. As you add widgets to the screen, FillLayout arranges them all horizontally in a row or vertically in a column. It spaces them all evenly so the entire composite is filled by the widgets. There is no layout data for FillLayout, so all you need to do is define whether the layout is horizontal or vertical and place the widgets on the screen.

This layout manager is useful when you have a list of labels or buttons that you want to arrange on the screen evenly. The good thing about FillLayout is that when you resize the screen, the spacing stays even. The same effect can be achieved with GridLayout, but with a lot more work.

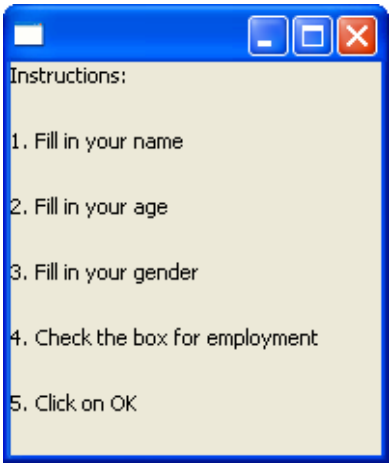
The following example shows how you can arrange labels on a screen using FillLayout. Note that you do not have to set any properties for the labels, as the layout manager takes care of their placement:

```
shell.setLayout(new FillLayout(SWT.VERTICAL));

Label label0 = new Label(shell, SWT.NONE);
label0.setText("Instructions:");
Label label1 = new Label(shell, SWT.NONE);
label1.setText("1. Fill in your name");
Label label2 = new Label(shell, SWT.NONE);
label2.setText("2. Fill in your age");
Label label3 = new Label(shell, SWT.NONE);
label3.setText("3. Fill in your gender");
Label label4 = new Label(shell, SWT.NONE);
label4.setText("4. Check the box for employment");
Label label5 = new Label(shell, SWT.NONE);
label5.setText("5. Click on OK");
```

This is the window that results:



Now if we resize the window, the labels are still spaced evenly:

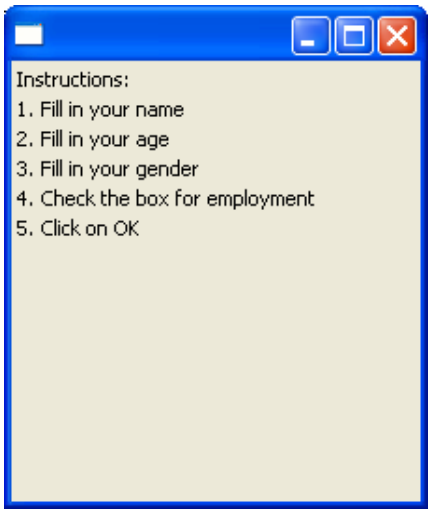


Next, we will look at RowLayout, and show how it differs from FillLayout. We will look at this same example again, plus look at a new one as well.

RowLayout:

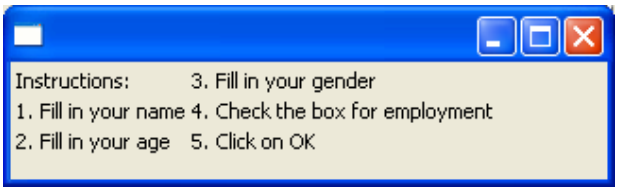
RowLayout arranges widgets in rows, as indicated by its name. We can declare a RowLayout to be horizontal or vertical, making the widgets arranged in either horizontal rows or vertical columns. Unlike FillLayout, RowLayout also has some other fields that can be set, such as wrap and pack as well as margins and spacing.

If we set the pack field to true, then all widgets will remain their natural size, and they will be placed as far to the left (or the top) as possible. This is a key feature that differentiates RowLayout from FillLayout. For example, if we set the layout for the previous example to a RowLayout with its pack field set to true, initially the window looks the same. However, when we resize it, the widgets remain the same size and are aligned as far to the top as possible:



Another feature of RowLayout that differentiates it from FillLayout is the wrap field. When the wrap field is set to true, if a row is too long for the window, the layout will wrap it around to form two rows. Take for example our previous FillLayout example. If we resize the window and make it too small for all the widgets to fit, they are simply overlapped:

However, if we use the same code but a RowLayout instead, the layout manager will compensate for the lack of space and create two columns instead of one:

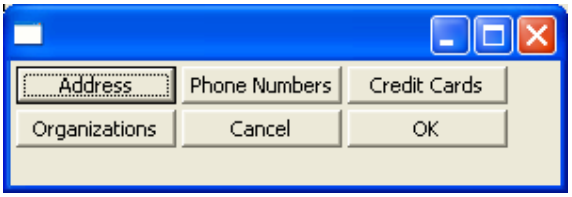


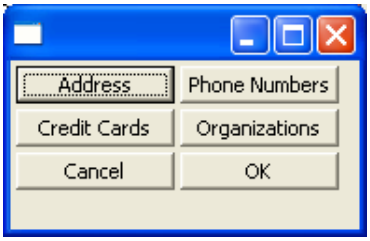
The following example shows how you can use the pack field to evenly space out your widgets. In this example, pack is set to false, so all the buttons on the screen will be the same size. The wrap field is set to true, so depending on the size of the window, there can be more than one row:

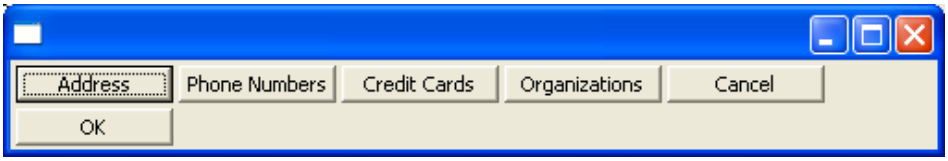
```
RowLayout rowLayout = new RowLayout();
rowLayout.pack = false;
shell.setLayout(rowLayout);

Button b1 = new Button(shell, SWT.PUSH);
```

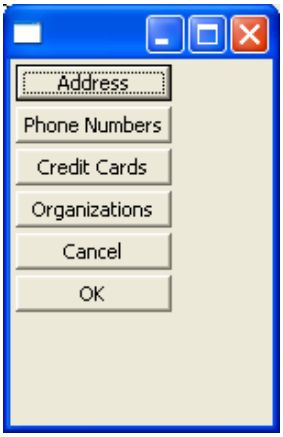
```
b1.setText("Address");
Button b2 = new Button(shell, SWT.PUSH);
b2.setText("Phone Numbers");
Button b3 = new Button(shell, SWT.PUSH);
b3.setText("Credit Cards");
Button b4 = new Button(shell, SWT.PUSH);
b4.setText("Organizations");
Button b5 = new Button(shell, SWT.PUSH);
b5.setText("Cancel");
Button b6 = new Button(shell, SWT.PUSH);
b6.setText("OK");
```

Here are some ideas of what the screen will look like, depending on the size of the window:









Notice how the layout manager attempts to put the buttons in horizontal rows. In the last screen shot, the window is only big enough for one button, so the layout manager has created six rows.

GridLayout:

GridLayout offers much more flexibility that RowLayout or FillLayout does, but this flexibility comes at a cost. GridLayout has many fields that need manipulating, as well as GridData, which is almost always necessary to add to widgets in order to fully get the benefits of the GridLayout manager.

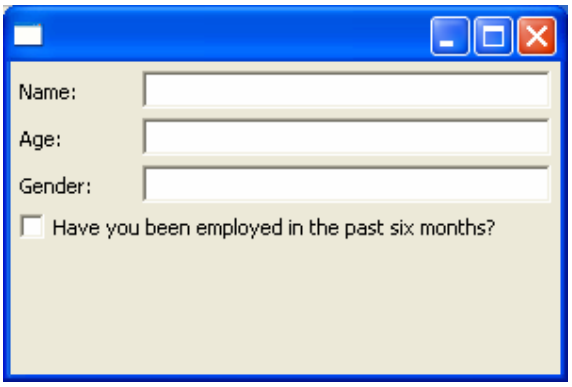
The most important fields to get a handle on with respect to GridLayout are the numColumn field and the makeColumnsEqualWidth field. This allows you to determine the number of columns of widgets you would like on your window, and whether they should all be the same size, or take the size of the widgets. You should always make the number of columns equal to the maximum number that you would desire in a particular row. Once you determine the number of columns, you can make any widget span as many columns as you wish. This allows the placement of the widgets to be very flexible.

As opposed to RowLayout, where RowData objects are available but not usually necessary, GridData objects are quite important in a GridLayout. The most important aspect of the GridData objects are the fields which allow you to control the behavior of widgets when the screen is resized. For example, let's look at the following piece of code, which demonstrates lining up labels and text boxes, a very common arrangement:

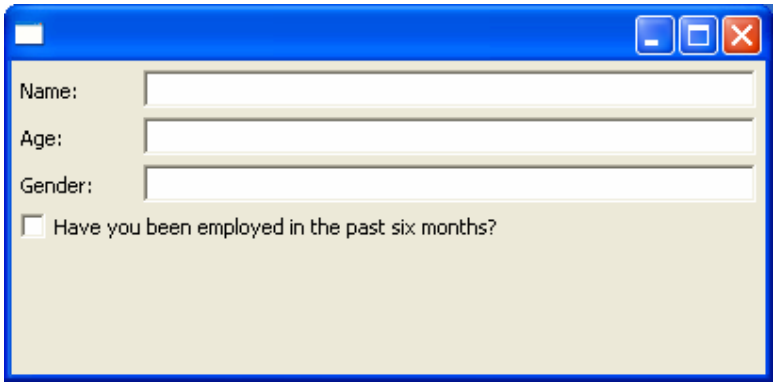
```
GridLayout gridLayout = new GridLayout();
gridLayout.numColumns = 2;
shell.setLayout(gridLayout);

Label label1 = new Label(shell, SWT.NONE);
label1.setText("Name:");
Text text1 = new Text(shell, SWT.BORDER);
Label label2 = new Label(shell, SWT.NONE);
label2.setText("Age:");
Text text2 = new Text(shell, SWT.BORDER);
Label label3 = new Label(shell, SWT.NONE);
label3.setText("Gender:");
Text text3 = new Text(shell, SWT.BORDER);
Button button = new Button(shell, SWT.CHECK);
button.setText("Have you been employed in the past six months?");

GridData data = new GridData();
data.widthHint = 60;
label1.setLayoutData(data);
```

```
data = new GridData();
data.widthHint = 60;
label2.setLayoutData(data);
data = new GridData();
data.widthHint = 60;
label3.setLayoutData(data);

GridData data2 = new GridData(GridData.FILL_HORIZONTAL);
text1.setLayoutData(data2);
data2 = new GridData(GridData.FILL_HORIZONTAL);
text2.setLayoutData(data2);
data2 = new GridData(GridData.FILL_HORIZONTAL);
text3.setLayoutData(data2);

GridData data3 = new GridData();
data3.horizontalSpan = 2;
button.setLayoutData(data3);
```

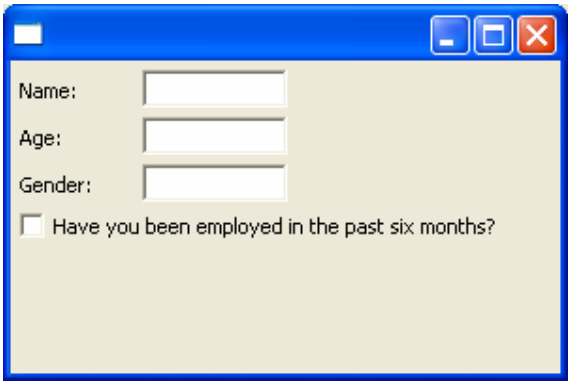
Note that a separate GridData object has to be declared for each widget. We have two columns in this layout, and the button at the bottom of the screen spans both columns. The window that results from this code is as follows:



In this example, the text widgets are set to fill horizontally. This means that not only do the text widgets initialize their size to fill up the rest of the screen width, but also when we resize the screen, they will grab the extra space:



Notice the difference if we take out the lines of code that give us this feature. Not only do the text widgets become the default size, but they will not resize when the screen is resized:



Once again, take a look at the Eclipse article Understanding Layouts in SWT for a more complex example of GridLayout. The many different fields regarding space usage for GridLayout are difficult to understand. However, once you get a handle on them, they can give you extreme flexibility for resizing each widget differently on a screen resize.

FormLayout:

The last layout we will look at is FormLayout. FormLayout is the newest layout manager, and it is debatable whether it is any better than GridLayout. Whatever the case, FormLayout offers an enormous amount of flexibility, like GridLayout, but in an extremely different fashion.

When using a GridLayout, it is necessary to plan ahead, as the order in which you add your widgets is the order in which they appear in the grid. When using a FormLayout, each widget is separate from all other widgets in terms of the overall layout.

In RowLayout, the RowData object is useful, but not altogether necessary. In GridLayout, the GridData object is fairly necessary but the layout manager can function

without it. In FormLayout, the FormData object is absolutely crucial. If you do not assign a FormData object to each widget, they will all be placed in the same default space, and all your widgets will simply sit on top of one another.

Each widget has its own FormData object, which is placed by using FormAttachments. The idea behind the FormAttachments, and essentially behind the FormLayout, is that we can tack the edges of the widgets to various places on the window. We can assign positions for the top, bottom, left and right edges of each widget. We can define these positions as being percentages, offsets from other widgets, or even alignments. To fully get a handle on how FormAttachments work, we will go over a fairly complex example.

Example – Putting it all Together:

The following example takes all the previous examples of the different types of layouts and puts them together using a FormLayout. Each of the previous examples are placed in a composite, and the three composites are placed on the shell using FormLayout. We will go over the example piece by piece.

First we will set the layout onto the shell, and create a composite which holds the FillLayout example, called fillComp. We want this composite to sit in the top left corner with it taking up 75% of the height of the screen, and 40% of the width. We need to create a FormData object for fillComp with the following FormAttachment objects:

```
shell.setLayout(new FormLayout());

//Fill Layout panel
Composite fillComp = new Composite(shell, SWT.BORDER);
fillComp.setLayout(new FillLayout(SWT.VERTICAL));
Label label0 = new Label(fillComp, SWT.NONE);
label0.setText("Instructions:");
Label label1 = new Label(fillComp, SWT.NONE);
label1.setText("1. Fill in your name");
Label label2 = new Label(fillComp, SWT.NONE);
label2.setText("2. Fill in your age");
Label label3 = new Label(fillComp, SWT.NONE);
label3.setText("3. Fill in your gender");
Label label4 = new Label(fillComp, SWT.NONE);
label4.setText("4. Check the box for employment");
Label label5 = new Label(fillComp, SWT.NONE);
label5.setText("5. Click on OK");

FormData formFill = new FormData();
formFill.top = new FormAttachment(0, 10);
formFill.left = new FormAttachment(0, 10);
formFill.bottom = new FormAttachment(75, 0);
formFill.right = new FormAttachment(40,0);
fillComp.setLayoutData(formFill);
```

Notice how we are putting a 10 pixel offset from the top and the left of the screen. We could eliminate this by setting the margin fields of FormLayout.

Next we will create the composite for the RowLayout code, called rowComp. We want this composite to be placed below fillComp, and we would like it to stretch to the left, right and bottom, and fill out whatever space is left:

```
//Row Layout panel
Composite rowComp = new Composite(shell, SWT.NONE);
RowLayout rowLayout = new RowLayout();
rowLayout.pack = false;
rowComp.setLayout(rowLayout);

Button b1 = new Button(rowComp, SWT.PUSH);
b1.setText("Address");
Button b2 = new Button(rowComp, SWT.PUSH);
b2.setText("Phone Numbers");
Button b3 = new Button(rowComp, SWT.PUSH);
b3.setText("Credit Cards");
Button b4 = new Button(rowComp, SWT.PUSH);
b4.setText("Organizations");
Button b5 = new Button(rowComp, SWT.PUSH);
b5.setText("Cancel");
Button b6 = new Button(rowComp, SWT.PUSH);
b6.setText("OK");

FormData formRow = new FormData();
formRow.top = new FormAttachment(fillComp, 10);
formRow.left = new FormAttachment(0, 10);
formRow.bottom = new FormAttachment(100, -10);
formRow.right = new FormAttachment(100, -10);
rowComp.setLayoutData(formRow);
```

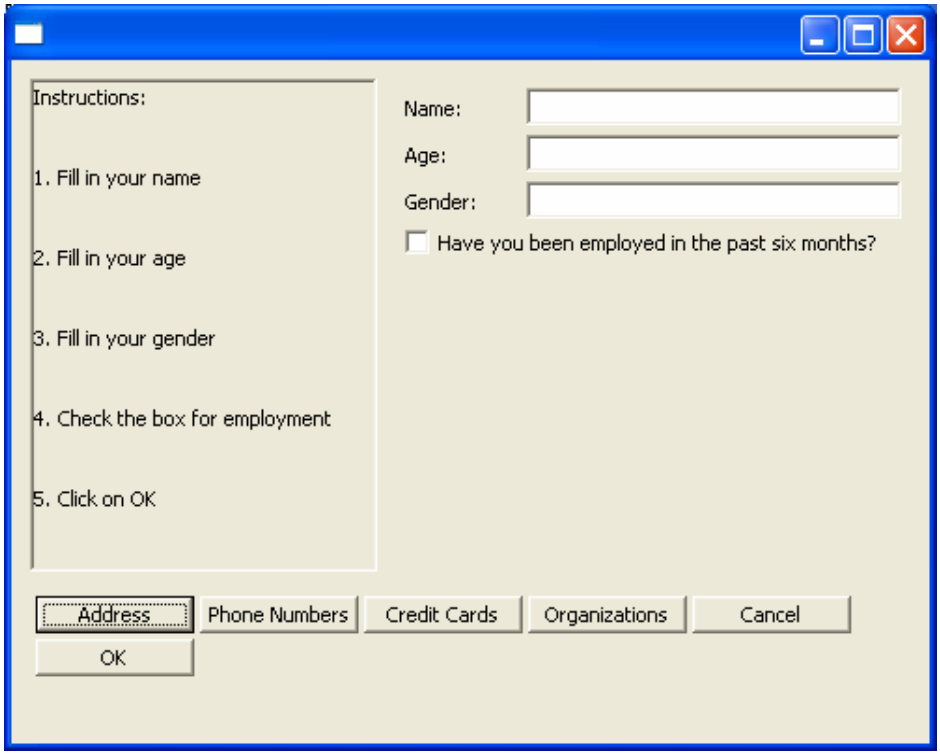
Finally, we will create the composite for the GridLayout portion, called gridComp. We want gridComp to be aligned with fillComp on the bottom, and to be attached to it on the left. We will extend to the edges on the top and the right:

```
//Grid Layout panel
Composite gridComp = new Composite(shell, SWT.NONE);
GridLayout gridLayout = new GridLayout();
gridLayout.numColumns = 2;
gridComp.setLayout(gridLayout);

Label label11 = new Label(gridComp, SWT.NONE);
label11.setText("Name:");
Text text1 = new Text(gridComp, SWT.BORDER);
Label label12 = new Label(gridComp, SWT.NONE);
label12.setText("Age:");
```

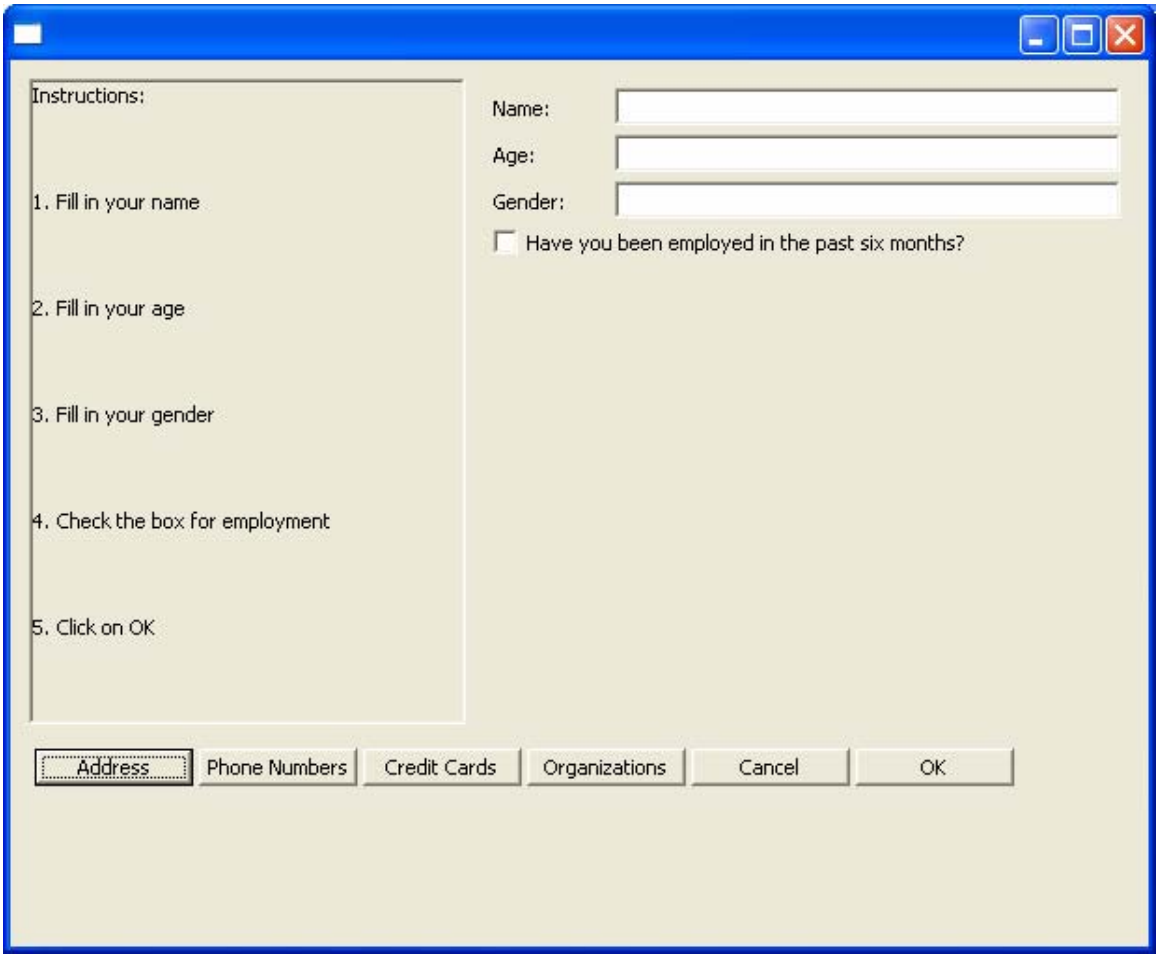
```
Text text2 = new Text(gridComp, SWT.BORDER);
Label label13 = new Label(gridComp, SWT.NONE);
label13.setText("Gender:");
Text text3 = new Text(gridComp, SWT.BORDER);
Button button = new Button(gridComp, SWT.CHECK);
button.setText("Have you been employed in the past six months?");

GridData data = new GridData();
data.widthHint = 60;
label11.setLayoutData(data);
data = new GridData();
data.widthHint = 60;
label12.setLayoutData(data);
data = new GridData();
data.widthHint = 60;
label13.setLayoutData(data);

GridData data2 = new GridData(GridData.FILL_HORIZONTAL);
text1.setLayoutData(data2);
data2 = new GridData(GridData.FILL_HORIZONTAL);
text2.setLayoutData(data2);
data2 = new GridData(GridData.FILL_HORIZONTAL);
text3.setLayoutData(data2);

GridData data3 = new GridData();
data3.horizontalSpan = 2;
button.setLayoutData(data3);

FormData formGrid = new FormData();
formGrid.top = new FormAttachment(0,10);
formGrid.left = new FormAttachment(fillComp,10);
formGrid.right = new FormAttachment(100,-10);
formGrid.bottom = new FormAttachment(fillComp,10, SWT.BOTTOM);
gridComp.setLayoutData(formGrid);
```

Note that in this example, all four sides of each FormData have been defined, this is not always necessary. If one or more edges are not defined, the widget will take its default size and determine the placement of its other edges.

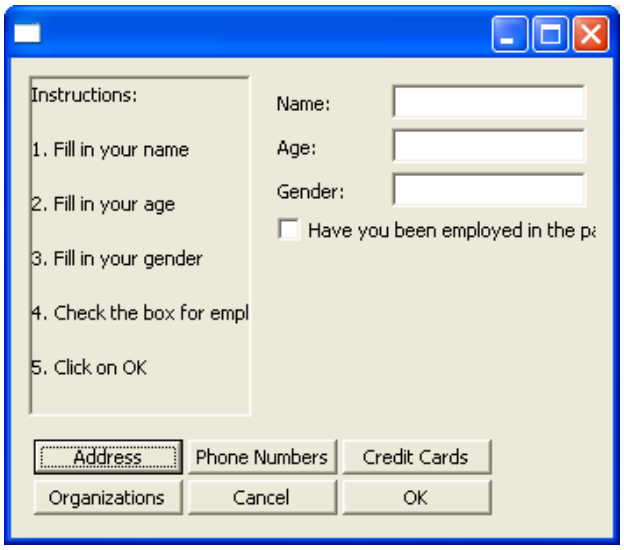
The window that results:



The important thing to note with this screen is the effect it has upon resize. Notice that since we have layouts within layouts, the resize effects are applied twice. Let's see what happens when we make the screen larger:



Since the rowComp composite was attached to the left and the right of the screen, when the window gets larger, the composite fills out the space. The RowLayout within rowComp now has enough room to place all the buttons in one row. Similarly, the gridComp was attached to the left side of the window, so the text widgets will grab the extra space provided by the GridLayout. The fillComp composite was attached at 40% of the screen, and as the screen gets larger, the composite will get larger too, so that it always sits at 40% on its left side. The same kind of effects will take place when the screen is made smaller:



There are many more features to all the layouts described in this chapter. Although it may take some time to become familiar with enough of the layout managers to be able to use them efficiently and effectively, it will soon become clear that it is much easier to design an SWT interface using layouts than without.