Swing Containers and Components

Overview

In this lab, you will create a Swing user interface comprising a number of containers and components, such as tabbed panes, a list box, a combo box, and a slider.

Source folders

Student project: `StudentSwingContainersComponents`
Solution project: `SolutionSwingContainersComponents`

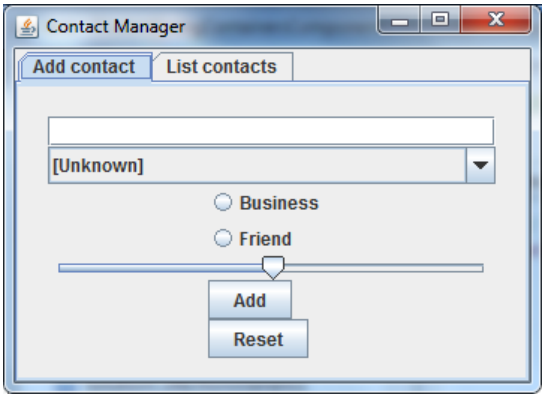
Roadmap

There are 5 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Familiarization with the solution application
2. Familiarization with the student code base
3. Creating UI components
4. Implementing event handlers
5. Additional suggestions

Exercise 1: Familiarization with the solution application

Open the *solution* project. The project contains a `Contact` class (which stores contact info), and a `ContactManager` class (this is the main class). Run the `ContactManager` class to display the following GUI:

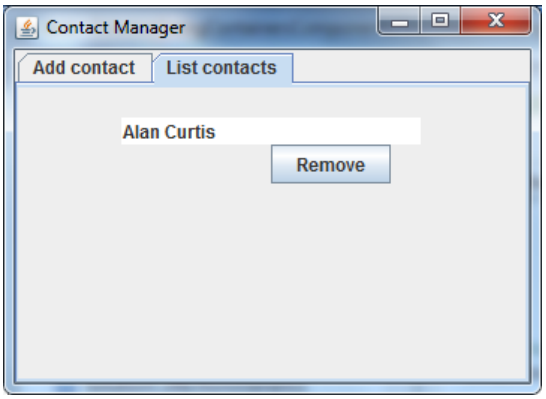


(The layout of this window is a bit rough-and-ready at the moment – we'll improve this layout in the next lab, when we take a closer look at layout managers.)

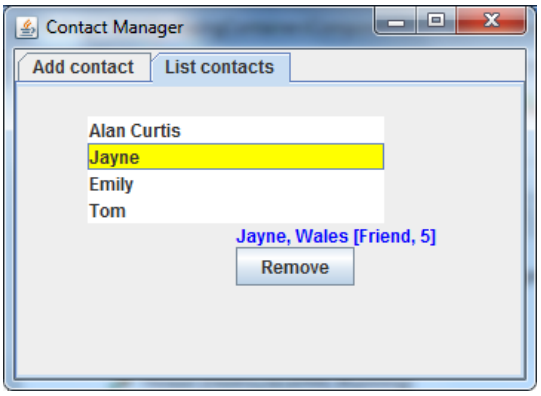
The window has two tabs, allowing you to add a contact and to display a list of all contacts respectively. In the *Add contact* tab, enter details for a new contact as follows:

- In the text box, enter the contact's name.
- In the combo box, select the country where the contact lives.
- Select either the *Business* radio button or the *Friend* radio button.
- Drag the slider to the left or right to indicate how familiar you are with the person.
- Click the *Add* button to add these details to the application.

The application creates a new `Contact` object based on the information you've entered, and adds it to a list box on the *List contacts* tab. If you click this tab, you should see the person's name appearing in the list box (again, don't worry too much about UI aesthetics just yet, we'll improve this in the next lab).



Go back to the *Add contact* tab and add a few more contacts (you can also click *Reset* to reset the fields at any stage). You can then view all the contacts in the *List contacts* tab. If you select one of the items in the list box, the full details for the contact are displayed in a label beneath the list box:



Likewise, if you click the *Remove* button, it removes the selected item (i.e. `Contact` object) from the list box.

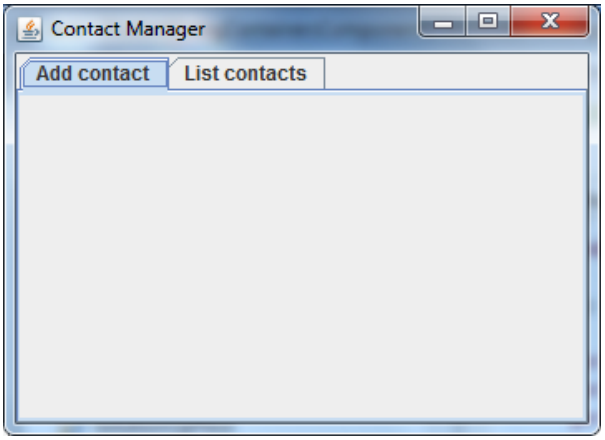
When you're happy with how everything works, close the window.

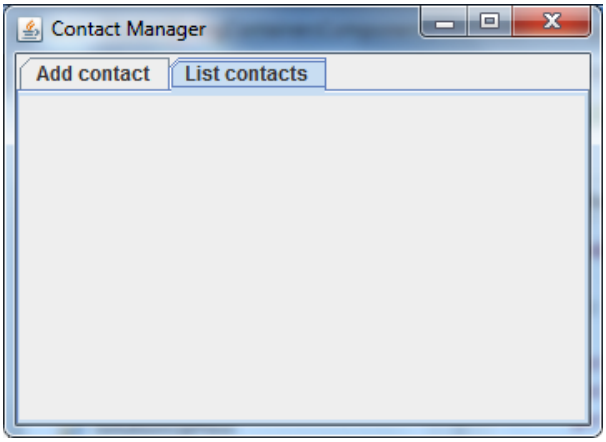
Exercise 2: Familiarization with the student code base

Switch to the *student* project and open the `Contact` class. This class is already complete. Note the following points:

- The constructor makes it easy to create a new `Contact` object with full contact details.
- The `toString()` method just returns the contact's name. This is significant in our application, because list boxes call `toString()` implicitly to determine how to display each object.
- The `getFullDetails()` method returns full details for the contact. We use this method in our application to display full details for a selected contact in a label.

Now take a quick look at the `ContactManager` class. It's partially implemented at the moment. If you run the class, it displays empty tabs as follows:





In Eclipse, take a closer look at the `ContactManager` class and observe that it implements two interfaces:

- `ActionListener`
  This interface represents button clicks, and requires an `actionPerformed()` method. We've already implemented this method near the bottom of the class. The method determines which button was clicked and invokes an appropriate method to process the event. You'll add code to each of these processing methods in the next exercise.
- `ListSelectionListener`
  This interface represents list selection changes (e.g. selecting an item in a list box), and requires a `valueChanged()` method. We've implemented a skeleton version of this method at the end of the class; you'll add full implementation details later on.

Note the following additional points in the `ContactManager` class:

- We've already declared instance variables for each of the components in the application, to save you a bit of time. All of these instance variables are `null` initially; you'll add code shortly to create the component objects and add them to the appropriate tab panel in the application.

- The `display()` method displays a `JFrame` window. The `JFrame` window contains a `JTabbedPane` comprising two tabs:

    - The 1st tab is entitled *Add contact*, and is populated with a panel created by the `createAddContactPanel()` method.

    - The 2nd tab is entitled *List contacts*, and is populated with a panel created by the `createListContactsPanel()` method.

You'll implement `createAddContactPanel()` and `createListContactsPanel()` in the next exercise.

Exercise 3: Creating UI components

Locate the `createAddContactPanel()` method and complete it as indicated by the TODO comments in the code. Here are some additional hints:

- `nameTextField` should be 20 characters in size.
- `countryComboBox` should contain an array of strings to initialize the combo box contents, e.g. `new String[]{"[Unknown]", "Wales", "Scotland", … }`
- `familiaritySlider` should specify the range `1, 5` for allowable slider values.

Now locate the `createListContactsPanel()` method and complete it as indicated by the TODO comments in the code. Here are some additional hints:

- `contactsList` is a `JList<Contact>` component. In keeping with many other components in Swing, `JList` uses a "model" class to store the data for the list box. If you want to allow the user to be able to add and remove items from the model, you have to create the `JList` as follows:

```
contactsList =
   new JList<>(new DefaultListModel<Contact>());
```

- `contactsList` generates "value changed" events when you select an item. To listen for these events, you use the following code:

```
contactsList.addListSelectionListener(this);
```

Run the application as it stands. Verify that the *Add contact* tab and the *List contacts* tabs display the GUI components as expected (note: the list box will not display yet, because it's empty).

Once you're happy, the next step is to add some event-handling functionality to the application...

Exercise 4: Implementing event handlers

Locate the `doAdd()` method in the `ContactManager` class. Add code to create a `Contact` object, based on the info entered by the user. Here are some hints:

- To get the text value entered in `nameTextField`, call the `getText()` method.
- To get the selected item in the `countriesComboBox()`, call the `getSelectedItem()` method. This method returns `Object`, so you'll need to cast it into a `String` to represent the contact's country.
- To determine which radio button is selected, call the `isSelected()` method on one of the radio buttons.
- To get the value of `familiaritySlider`, call the `getValue()` method.

Once you've created the `Contact` object, you can add it to the list box's model object. Here's the code you need:

```
// Get the "model" associated with the list box,
// and then add the new contact into the list box's model.
DefaultListModel<Contact> listModel =
                     (DefaultListModel<Contact>)contactsList.getModel();
listModel.addElement(contact);
```

Next, implement the `doReset()` method so that it resets each of the fields in the *Add contact* tab (e.g. clear the text in the text box, set the combo box's selected item to `0`, etc.).

Next, implement the `doRemove()` method, so that it removes the selected item from the `contactsList` list box. Here are some hints:

- Get the index of the selected item in the list box, via `getSelectedIndex()`.
- If there is an item selected (i.e. the selected index is not `-1`), remove that item from the list box's model (use the `remove()` method on the list box's model).

Finally, implement the `valueChanged()` method, so that it gets the selected item from the list box and displays its full details in the `selectedContactLabel`. Here are some hints:

- Get the selected item in the list box, via `getSelectedValue()`.
- If there is an item selected (i.e. the selected item is not `null`), display the full details for the contact in the `selectedContactLabel`. Otherwise, just display an empty string.

Run the application. Verify that you can add contacts, list contacts, view details for a contact, and remove a contact. Also verify that you can reset the fields on the *Add contact* tab.

Exercise 5 (If time permits): Additional suggestions

- Put the list box into a scroll pane, so that the list box is scrollable.
- Experiment with menus and toolbars in the window.