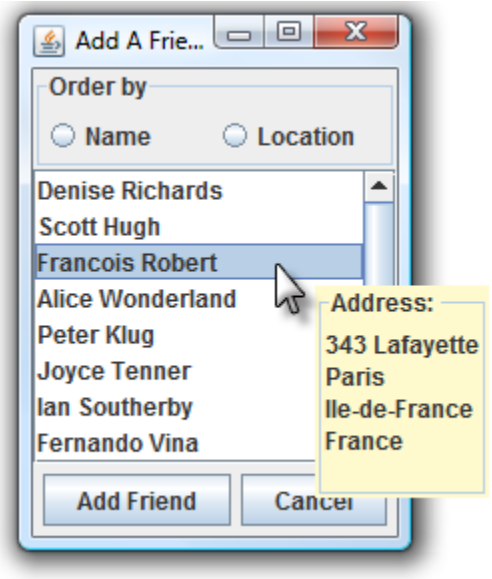
Tutorial: Implementing a Graphical User Interface

In this tutorial we have marked questions we think are harder than others with a **[‡]**. A link to the solution is provided at the end of the tutorial, but you are strongly encouraged to work it out yourself, before you look at the solutions. You will learn a lot more that way. If you have any questions, ask one of the instructors or TAs or post a question to the google group!

Adding a Friend (Foomix)

We have introduced the Foomix framework in the last two exercise question sets. In this exercise, we ask you to work on a small part of the graphical user interface. Presented below is the GUI for selecting a friend from a list of users and adding him/her as a friend.



The GUI lists all possible users, allows you to sort the users by name and location and provides a popup with the address when left-clicking on a user in the list. The following questions will guide you step by step through the implementation of the GUI.

View

1. Determine the containment hierarchy for the GUI. (Note: for simplicity reasons, we used a `JFrame` as top-level container for this part; if we implemented the whole Foomix Network with a GUI we would most likely implement this part as a `JDialog` instead.)

2. Implement the top-level container with two buttons, one for adding a friend and one for cancelling. Use the `java.awt.BorderLayout` as the layout for the top-level container and place the buttons at the bottom of the GUI (`BorderLayout.SOUTH`). Furthermore, add a `main` method that creates an instance of the GUI and sets it visible. As it is difficult to write automated tests for graphical user interfaces, GUIs are often tested by running and visually checking the GUI. Running the main method after each step will allow you to incrementally check your implementation, which will allow you to detect errors early on. Do not wait until the end, as by then the code will be more complex and it will be more difficult to detect the errors.
For now, do not worry about implementing any event listeners, we will get to this soon.

3. Create a scroll pane (`javax.swing.JScrollPane`) and add it to the center of the GUI. A scroll pane will automatically provide scroll bars if necessary without you having to worry about it. Now, add a `javax.swing.JList` to the scroll pane. To see whether your code works as intended, create an array of `String`s,

```
String[] names= {"Denise Richards", "Scott Hugh", "Francois Robert", "Alice
                 Wonderland", "Peter Klug", "Joyce Tenner", "Ian Southerby",
                 "Fernando Vina", "Andreas Lanz"};
```

and set it as the data of the list. Test your code by running the main method and checking whether the list is displayed properly.

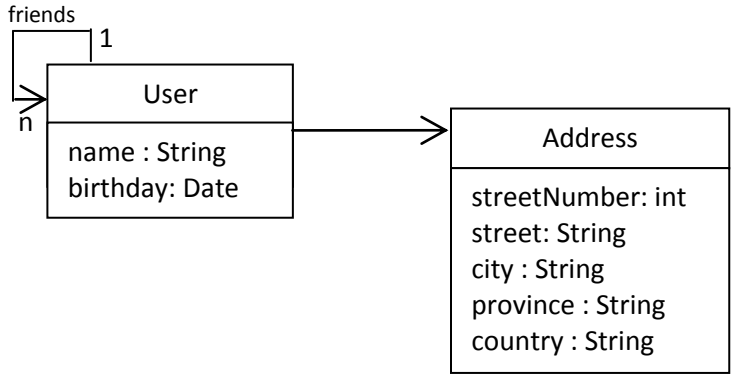
4. For the last part of the view, you will have to implement two radio buttons that will allow the user to choose the sort order. For now you don't have to write any listeners yet, we will add these soon. Place the buttons into a panel and use a `java.awt.GridLayout` for aligning them next to each other. Use a `javax.swing.TitleBorder` to add a border and a title to the panel as shown in the picture. Furthermore, add the two buttons to a `javax.swing.ButtonGroup`. The button group ensures that there is always at most one button selected. Test your code by running the main method again.

Model

Now that you have implemented the view part of the GUI, we will work on the model. As you might remember, the Swing framework uses a modified version of the MVC called a separable model architecture. In the Models section of the Graphical User Interface reading you saw that a `JList` has a model object of type `ListModel`. By default, the `JList` creates an object of type `DefaultListModel` that implements the `ListModel` interface and adds the data the user provides to the model. In our case the `JList` added the `String` array to an object of type `DefaultListModel`.

`JList` does not sort the data, it only displays the data in the order provided by its underlying model. In our case it will just present the names in the order we added them to the String array. Now, to sort the items in the list, you could sort the `String` array, remove the old one from the `JList` and add the sorted array again. However, a better way is to provide a customized model for the list that allows it to sort the data. Over the following steps, you will implement your own model that provides sorting support.

5. The model in our list represents users of Foomix. Therefore, we first need to have an implementation of type `User` as well as type `Address` as specified in the following UML class diagram and in the question set on Implementing an OO Design. You can reuse the implementation of these two types from the question set, you will just need to add street and streetNumber.



Also, if you haven't done so already, implement `Comparable` for type `User` so that `User`s can be sorted by the `Address` (in alphabetical order first by country, then by province, city, street and street number, e.g. a user from BC, Canada would be after a user from Alberta, Canada, but before a user from Bavaria, Germany). Therefore, delegate the comparison to `Address`.

Make sure to provide a `toString()` method for type `User` (override the `toString()` method of type `Object`) that returns a reasonable string representation of a user. This is necessary, as the `toString()` method will be called to determine how to display a user in the `JList` later on.

6. In our example we want to provide a model that is specific to our Foomix scenario, in which we need to sort a list of users by name or location/address. Therefore, create a class `SortedUserListModel` that extends `javax.swing.AbstractListModel`. The constructor of `SortedUserListModel` should take an object of type `List<User>` as parameter, representing the list of users it is modeling. You need to provide implementations for the methods `getElementAt` and `getSize`.

For the actual sorting, create a method `setSortOrder` that takes a parameter as input that lets you determine which way the list should be ordered (i.e. either by name or by location). Dependent on the sort

order, this method should sort the list of users. As the `User` class already provides the necessary `compareTo` method (the class `User` implements the `Comparable` interface) to sort by location, sorting by location can simply be done by using the method `sort` from the `Collections` class that takes a list as input. For sorting the list of users by name, you should use the sort method of class `Collections` that takes a list as input as well as an object of type `Comparator`. Therefore, you need to implement your own `Comparator` that sorts two users by name. To implement your own comparator, you create a class that implements the interface `Comparator` and provides an implementation for the method `compare` (more information can be found under http://java.sun.com/j2se/1.4.2/docs/api/java/util/Comparator.html or google for comparator examples). Note, you can delegate the actual comparison to the compare method of class `String`. **[‡]**

7. Now that you created your own model class, you also need to use it. Therefore, when you create the object of type `JList`, you need to create a model of type `SortedUserListModel` and pass it to `JList` as the model (either in the constructor or by explicitly setting it using the `setModel` method).
This example shows that the separation of model and view in the Swing framework makes it easy to change the model without having to adapt the view (i.e. the `JList`).
To create a list of users in our example, you can copy and use the following method:

```
private static List<User> createSampleUsers() {
        List<User> users= new ArrayList<User>();
        String[] names= {"Denise Richards", "Scott Hugh", "Francois Robert", "Alice Wonderland", "Peter
                         Klug","Joyce Tenner", "Ian Southerby", "Fernando Vina", "Andreas Lanz"};
        Date[] bdays= {new Date(1983, 4, 21), new Date(1989, 2, 1), new Date(1967, 9, 25), new Date(1981,
                       5, 13), new Date(1983, 3, 19), new Date(1989, 1, 7), new Date(1956, 8, 11), new
                       Date(1982, 10, 4), new Date(1975, 3, 4)};
        String[] streets= {"Broadway", "10th", "Lafayette", "14th", "Tannenweg", "Broadway", "Downing
                         Street", "Calle de Ramiro", "Ostender"};
        int[] numbers= {2310, 10, 343, 13, 45, 4639, 345, 38, 112};
        String[] cities= {"Vancouver", "Calgary", "Paris", "Edmonton", "Munich", "Vancouver", "London",
                         "Madrid", "Berlin"};
        String[] provinces= {"BC", "AB", "Ile-de-France", "AB", "Bavaria", "BC", "London", "Madrid",
                         "Berlin"};
        String[] countries= {"Canada", "Canada", "France", "Canada", "Germany", "Canada", "England",
                         "Spain", "Germany"};

        for (int i=0; i<names.length; i++) {
                users.add(new User(names[i], bdays[i], new Address(streets[i], numbers[i], cities[i],
                         provinces[i], countries[i])));
        }
        return users;
}
```

Now we have completed the view and the model and we need to provide the model change notification mechanism by adding event listeners.

8. Add an action listener to the Cancel button that closes the GUI by invoking the `dispose()` method of `JFrame`. Run your application and test it.

9. Add an action listener to the Add Friend button that prints out the name of the user to the console using `Sytem.out.println` and then closes the GUI if an item (a user in our case) in the `JList` object is selected and prints an error message to the console if Add Friend is pressed when no user is selected. Run your application and test the functionality you just added.

10. Add action listeners to the two radio buttons for the sort order. Depending on the selected sort order, set the sort order of the model of the `JList` object by invoking your previously implemented `setSortOrder` method. Run the application and test if the sorting works. If the displayed list does not update, you might need to debug your code, in particular your `setSortOrder` method. Make sure that the `setSortOrder` method contains a call

```
fireContentsChanged(ListDataEvent.CONTENTS_CHANGED, 0, ....);
```

with "..." replaced by the proper argument. **[‡]**

11. Finally, we would like to add a popup that displays the address of a user when selected. Therefore, add a mouse listener to the `JList` object that creates a popup on a left mouse button click and shows it. The popup should contain the address of the user just selected (as presented in the picture near the top of page 3). If possible, hide the popup again after 3 seconds. It might help to google for examples of class `javax.swing.Popup`. Don't forget to test it ☺. **[‡]**

SOLUTIONS:

2. A solution is provided under

svn+ssh://westham.ugrad.cs.ubc.ca/home/c/cs210/repositories/exerciseQuestions/gui

However, it is advisable to do the implementation yourself, as you will learn a lot more from doing the example than from reading the solution.