**COMP1161**

**LAB 6**

## PREAMBLE

This lab aims to introduce the student to the basics of creating a graphical user interface (GUI) using Java.

## THE PROBLEM

After the initial deployment of Vaccinator Prime(see Lab 5 description), it has become evident that a large number of users may need to interact with the person registration process on the front lines. As a result it has been decided that a GUI is needed for that part of the system. Your immediate tasks are

1. to complete a prototype that demonstrates basic functionality of displaying a list of persons, adding a person and sorting the list as required, and
2. then describe the effort required to implement a fully functional CRUD (Create/Read/Update/Delete) application.

Your entry point to the project comes at a time when a junior developer has completed a basic mockup of the appearance of the system, which has been approved by top management.

Specific objectives of the lab are:

- Integrating a GUI with an application
- Adding components to a GUI
- Writing listeners for GUI components

## NON-FUNCTIONAL ASSESSMENT CRITERIA

| **Criterion** | **Mark(s)** |
|---|---|
| Ability to describe the containment architecture used for the lab. | 1 |
| Ability to demonstrate, with examples, an implementation of the ActionListener interface (including descriptions of relevant method arguments) | 1 |
| An ability to describe how exception handling can be applied in a GUI application | 1 |
| Self Documenting Code | 1 |
| A description of effort required to implement features in a fully featured applications…One from a set of questions which may include the changes required to implement<br>i. Persistent storage (The system can will changes made before the computer is turned off)<br>ii. Secured access to allow a person to log in<br>iii. Screens that can allow previously entered information to be edited<br>iv. Screens that can allow the information to be viewed but not edited<br>v. The ability to remove data. | 1 |

*Please note that a total of **10 marks** are allocated to Lab 6.* ***ALL 10 MARKS WILL BE ASSIGNED IN THE LAB SESSION. THERE IS NO AUTOGRADER COMPONENT FOR THIS LAB***

## LAB EXERCISES (5 MARKS FOR DEMONSTRATED FUNCTIONALITY)

Java code for four classes are provided. The classes are:

- **BasePerson**:Abstract person class – Exact replica of class used as a starter code for Lab 5
- **Person**: Defines a person – Exact replica of class used as a starter code for Lab 5
- **PersonListing**: Starter code for screen that should list persons
- **PersonEntry** : Starter code for a screen that should allow entry into the list of persons.

Please download these classes and load them into a project for this lab. Spend a few moments to get acquainted with the code.

PersonListing.java exposes a public static main method that presents the screen shown in Figure 1. In addition, PersonListing.java also includes several attributes and methods that are used to implement basic functionality, some of which are presented in Table 1.

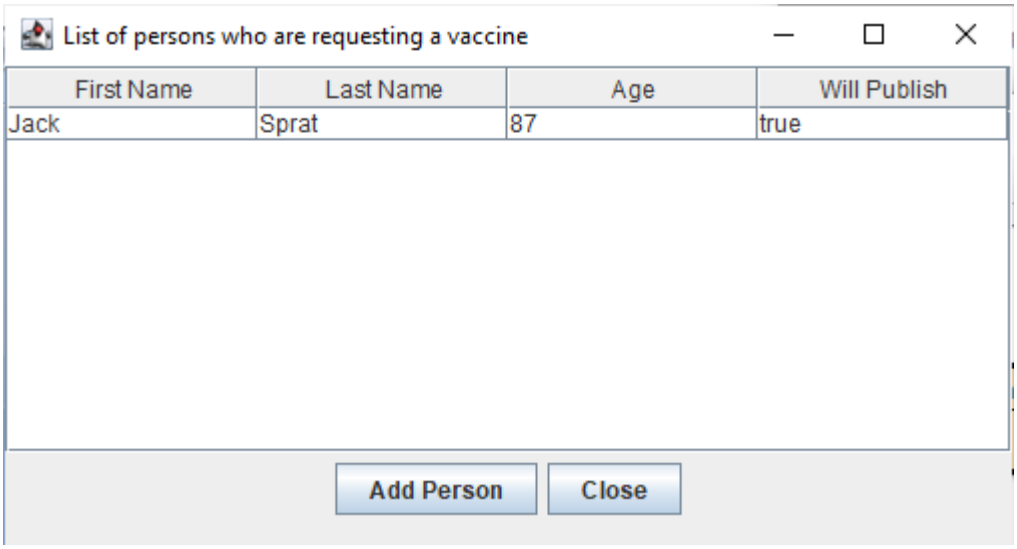


*Figure 1 Person Listing Screen*

| **PersonListing (Notable attributes and methods)** | |
|---|---|
| -plist:ArrayList<Person> | *ArrayList of Persons* |
| -thisForm: PersonListing | *Reference current object* |
| -cmdAddPerson :JButton | *Button to add a person* |
| -table: JTable | *Table to show list* |
| -loadPersons(string): ArrayList<Person> | *Load person method – same one given with Lab5* |
| -showTable( ArrayList<Person>):void | *Shows person list in table* |
| -addToTable(Person):void | *Adds one entry to the table* |
| +addPerson(Person):void | *Adds a person to the form* |

*Table 1. Notable attributes and methods that implement functionality in starting code*

The starting code include the class PersonEntry that displays the information in Figure 2, and includes no functionality.


*Figure 2. Result of executing starting code for PersonEntry.*

## GRADEABLE ACTIVITIES

1. **Section1.- Populating the list of persons on the person listing form, and put functionality in the "Add Person" button (1 mark)**

    a. *In PersonListing:* The starting code places only one item in the list of persons. Discussions with the original developer have yielded that the method "showTable" in PersonListing is responsible for populating the table. The addToTable method which accepts one person and adds information regarding that person to the table should NOT be modified. Complete showTable so that an entry is placed in the table for each row in the file person.dat.

    b. *In PersonEntry:* Modify PersonEntry so that it is able to store an instance of PersonListing.

    c. *In PersonEntry:* Modify the Constructor of PersonEntry so that it accepts an instance of PersonListing, and then sets the local instance variable to the value accepted by the constructor.

    d. *In PersonListing:* Add a listener to the "Add Person" button so that when it is clicked, an instance of PersonEntry is displayed. When invoking the instance of PersonEntry, a reference to the current instance of PersonListing should be passed as an argument to the constructor. *//Hint… a listener is already included in the starting code that adds functionality to the Close button.*

2. **Section 2. Implementing functionality in the PersonEntry. (1.5 marks)**

    a. Modify the PersonEntry object so that the interface looks like that presented in Figure 3


*Figure 3.Expected Appearance of Person Entry Screen*

*Hints:*

*i. You may need to add a JLabel and a JCheckBox*

*ii. The layout from the previous developer for the panel displays data on two rows.*

*b.* Add functionality to the Close button of the PersonEntry form so that it is no longer visible when it is clicked. *Hint: one way to do it is to*

*i. Store a reference to a PersonEntry as an instance variable*

*ii. Set the instance of PersonEntry to **this** in the constructor*

iii. *In the appropriate listener, invoke setVisible on the instance of PersonEntry with an argument of false.*

c. Implement functionality in the Save button of PersonEntry so that when it is clicked, the data is first validated to ensure both first and last names have been entered, and that the age is an integer. If data is valid, the addPerson method of PersonListing is called, and the person entry form disappears. Note that the addPerson method effectively adds a person to the arraylist holding person list data, and updates the data in the list. *Hints/Questions*

i. *Have we already met a method to **split** a string into parts?*

ii. *What functionality can tell if a value can be converted to an Integer?*

iii. *Remember the getText() method of a jTextField returns the value of the text it currently displays.*

**3. Section 3 – Complete functionality on listing screen (1.5 marks)**

a. The starting code includes the definition of a JButton named cmdSortAge, but it is not shown on the PersonListing screen. Modify the PersonListingScreen to allow it to be shown

b. Add a listener which implements the functionality to sort the list in ascending order of age. *Hints:*

i. *You should already know how to sort data*

ii. *Presenting information in a specific order in the table may involve removing all the data and then re-adding it*

iii. *Data in the table is removed with the command **model.setRowCount(0);**- model is an object name that was declared with the starting code. Ie. model is an object of the class DefaultTableModel.*

c. Add a button and an associated listener to sort the list in order of first name .

**4. Change the colors of the form to implement a custom colour scheme. (1 mark)**

Figure 4 shows an example of a custom colour scheme. You should come up with your own and then set the colours on the Person Listing screen appropriately.

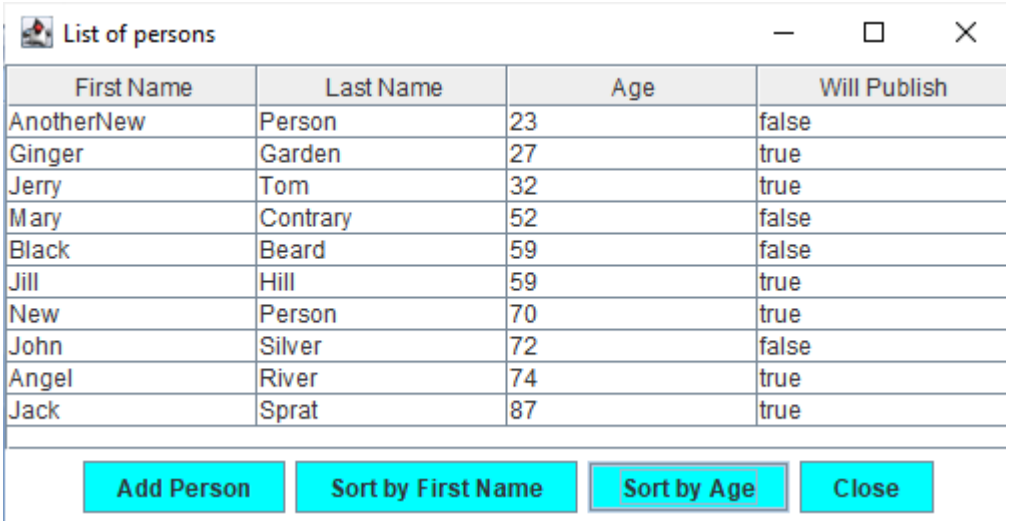


*Figure 4. An example of a custom color scheme after two more persons have been entered, sorted in order of age.*

**5. Compress the .java files into a .zip and submit to OurVLE.**