LasIESpeñetas

UNIÓN EUROPEA
Fondo Social Europeo
El FSE invierte en tu futuro

I.E.S. Las Espeñetas - Orihuela

1ºCFGS DAM
Entornos de Desarrollo

Practica Agenda En JAVA

Alumno: Adrián Tristán Cayuela
Alumno: Cristian Navarro Pertegal

Profesor: Jaime Riquelme Garcia

26 de febrero de 2023

Índice

1. Introducción

La práctica consta de programar una agenda en JAVA con interfaz gráfica entre dos personas para así usar GITHUB como controlador de versiones.

2. Objetivos

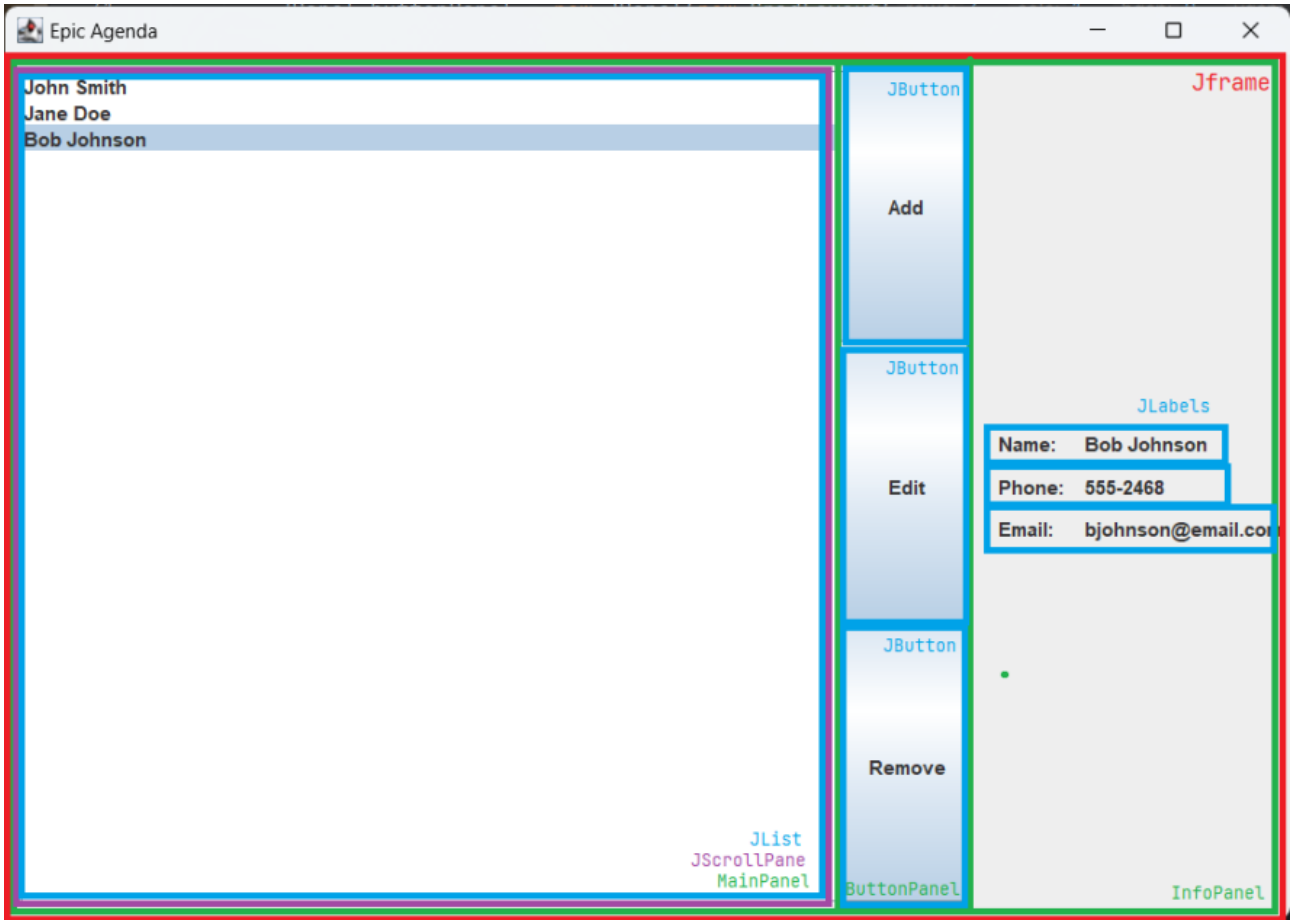
En este documento se va a explicar brevemente como hemos realizado las diferentes funciones de nuestra agenda.
Las funciones a explicar son las sigueintes:

- Creación de interfaz grafica de agenda.
- Función para crear contactos.
- Función para editar contactos
- Función para borrar contactos.
- Función para mostrar contactos mediante lista.
- Creación del InfoPanel.

3. Desarrollo

3.1. Creación de interfaz grafica de agenda.

A continuación se muestra el contenido de la interfaz gráfica de la ventana principal del programa.



3.2. Función para crear contacto

```
1         private void createAddButton() {
2         addButton = new JButton("Add");
3         addButton.addActionListener(e -> {
4             new AddContactDialog(Agenda.this, model).setVisible(true);
5         });
6     }
```

Creamos un metodo privado llamado ***createAddButton()***, el cual crea un boton de Agregar ***(addButton = new JButton(Add);*** y establecemos un ActionListener, que mostrara un nuevo dialogo llamado ***AddContactDialog.***

Agenda.this es una referencia al objeto Agenda actual, que se utiliza para establecer la posición del cuadro de diálogo en la pantalla y para actualizar la lista de contactos de la agenda cuando se agrega un nuevo contacto.

AddContactDialog es básicamente una ventana emergente que sirve para añadir contactos.

//Interfaz grafica//

Empezamos creando las etiquetas de la interfaz graficas Jlabels, TextField, JButton y DefaultListModel.

Tras esto creamos el constructor de la clase ***AddContactDialog***. Esta toma como parámetros JFrame padre y un DefaultListModel.
Dentro de este constructor vamos a establecer el título del cuadro de dialogo, diseño de tipo "***GridBagLayout***"
Y crearemos los campos de entrada de nombre, telefono, y correo electronico, tambien los botones para agregar y cancelar y los agregamos al panel de botones y ajustamos el tamaño de la ventana y lo ubicamos junto con el Jframe Padre**(setLocationRelativeTo(parent)**;

Tras esto, para nombre, teléfono y email vamos a añadirles un controlador de posición mediante su eje, basado en el Layout GridBag. Y a cada uno le vamos crear un TextField.

Para los botones, igual le añadimos un controlador de posición y un listener para que recoja los datos de cada campo (Email,nombre,telefono) y si estos están vacíos se crea un mensaje de error.

Por último, realizamos él envió de los datos introducidos mediante tres métodos públicos que devolverán el contenido de (Email, Teléfono y nombre) mediante cadenas de texto.

3.3. Función para editar contactos

```
1         private void createEditButton() {
2         editButton = new JButton("Edit");
3         editButton.addActionListener(e -> {
4             if (contactList.getSelectedValue() == null)
5                 return;
6
7             new EditContactDialog(Agenda.this, model, contactList.getSelectedValue())
                  .setVisible(true);
8         });
9     }
```

El botón de editar sigue la misma estructura que el botón de añadir, creamos un metodo privado llamado ***createEditButton*** , creamos el boton, y hacemos el listener, aquí la única diferencia con el de añadir es que antes de redirigir a la ventana emergente del dialog, mediante un if si no existe nada en seleccionado no sucede nada, si esta seleccionado crea la ventana emergente.

//Interfaz grafica//

En ***EditContactDialog*** las únicas diferencias son en createEditButton. Si todos los campos están completos se crea un nuevo Contacto con los valores ingresados y mediante ***int index = model.indexOf(contact);*** el Contacto que se está editando se elimina y se reemplaza por un nuevo contacto y tras eso se cierra la ventana mediante ***dispose().***

3.4. Función para borrar contactos

```
1         private void createContactsJList() {
2         model = new DefaultListModel<>();
3         model.addElement(new Contact("John Smith", "555-1234", "jsmith@email.com"));
4         model.addElement(new Contact("Jane Doe", "555-5678", "jdoe@email.com"));
5         model.addElement(new Contact("Bob Johnson", "555-2468", "bjohnson@email.com")
              );
6
7         contactList = new JList<>(model);
8         contactList.setCellRenderer(new ContactRenderer());
9     }
```

Para el botón de borrar igual que con los anteriores botones creamos un método privado llamado createRemoveButton, dentro creamos el botón y añadimos un listener que al pulsar el botón se asegure de se ha seleccionado algún contacto de la lista, se ha creado una ventana emergente de confirmación que viene ya integrada en java en la clase JOptionPane.

La cual nos preguntara si estamos seguros de eliminar el contacto. Si la opción es elegida es si, se elimina mediante model.removeElement(contactList.getSelectedValue());.

3.5. Función para mostrar contactos mediante lista

```
1         private void createContactsJList() {
2         model = new DefaultListModel<>();
3         model.addElement(new Contact("John Smith", "555-1234", "jsmith@email.com"));
4         model.addElement(new Contact("Jane Doe", "555-5678", "jdoe@email.com"));
5         model.addElement(new Contact("Bob Johnson", "555-2468", "bjohnson@email.com")
              );
6
7         contactList = new JList<>(model);
8         contactList.setCellRenderer(new ContactRenderer());
9     }
```

Para crear la lista de contactos, definimos un método llamado: ***createContactsJList()*** primero se crea un modelo de lista por defecto ***("DefaultListModel<>()")*** el cual contendrá los elementos de la lista de contactos.

Luego hemos añadido tres contactos a la lista para que aparezcan por defecto mediante el metodo ***addElement()***, y hemos creado una instancia **contactList = new JList<>();** y le pasamos el modelo como instancia (model),

Por último, se establece un renderizador de celdas para la lista de contactos ***contactList.setCellRenderer(new ContactRenderer());***

```
1 public class ContactRenderer extends JLabel implements ListCellRenderer<Contact> {
2     public ContactRenderer() {
3         setOpaque(true);
4     }
5
6     @Override
7     public Component getListCellRendererComponent(JList<? extends Contact> list,
          Contact value, int index, boolean isSelected, boolean cellHasFocus) {
8         setText(value.getName());
9         if (isSelected) {
10            setBackground(list.getSelectionBackground());
11            setForeground(list.getSelectionForeground());
12        } else {
13            setBackground(list.getBackground());
14            setForeground(list.getForeground());
15        }
16        return this;
17    }
```

La clase ***ContactRenderer*** extends JLabel, lo que significa que cada celda de la lista se representará como una etiqueta con un texto correspondiente al nombre del contacto.

El método ***getListCellRendererComponent()*** se debe usar cuando se implementa la interfaz ***ListCellRenderer***. Este método se llama para cada celda en la lista de contactos y se utiliza para establecer un aspecto en la celda.

Dentro de este método establecemos él se establece el texto de la etiqueta en el nombre del contacto mediante **setText(value.getName()**. Entonces creamos un if para verificar si la celda seleccionada "***isSelected***", si esta seleccionada se establece el fondo y los colores de la selección. en caso de que no simplemente se seleccionan los colores de la lista predeterminada.

Finalmente devolvemos la etiqueta this como componente de la celda personalizada

3.6. Creación del InfoPanel

```
1  private void createInfoPanel() {
2          infoPanel = new JPanel(new GridBagLayout());
3          GridBagConstraints c = new GridBagConstraints();
4          c.anchor = GridBagConstraints.NORTHWEST;
5          c.insets = new Insets(5, 5, 5, 5);
6
7          nameLabel = new JLabel("Name: ");
8          c.gridx = 0;
9          c.gridy = 0;
10         infoPanel.add(nameLabel, c);
11         nameValueLabel = new JLabel();
12         c.gridx = 1;
13         c.gridy = 0;
14         infoPanel.add(nameValueLabel, c);
15
16         phoneLabel = new JLabel("Phone: ");
17         c.gridx = 0;
18         c.gridy = 1;
19         infoPanel.add(phoneLabel, c);
20         phoneValueLabel = new JLabel();
21         c.gridx = 1;
22         c.gridy = 1;
23         infoPanel.add(phoneValueLabel, c);
24
25         emailLabel = new JLabel("Email: ");
26         c.gridx = 0;
27         c.gridy = 2;
28         infoPanel.add(emailLabel, c);
29         emailValueLabel = new JLabel();
30         c.gridx = 1;
31         c.gridy = 2;
32         infoPanel.add(emailValueLabel, c);
33
34         contactList.addListSelectionListener(e -> {
35             Contact selectedContact = contactList.getSelectedValue();
36             if (selectedContact != null) {
37                 nameValueLabel.setText(selectedContact.getName());
38                 phoneValueLabel.setText(selectedContact.getPhoneNumber());
39                 emailValueLabel.setText(selectedContact.getEmail());
40             } else {
41                 nameValueLabel.setText("");
42                 phoneValueLabel.setText("");
43                 emailValueLabel.setText("");
44             }
45         });
46     }
```

Para la creación del infoPanel, lo que hemos buscado es que fuera dinámico según el contacto que seleccionaras, el panel lo creamos usando GridBagLayout.

Hemos creado un objeto llamado GridBagConstraints, y mediante c.anchor hemos especificado la posición en la que va a estar anclada este objeto dentro de la celda en este caso NORTHWEST.
Mediante c.insets vamos a establecer que cada margen tenga 5px en cada dirección.

Tras esto creamos los label para cada campo y le damos una posición mediante un eje x e y, cada etiqueta va a tener el valor del contacto seleccionado. Para ello hemos creado un listener el cual cada vez que seleccionemos un contacto, nos devuelva la información de cada etiqueta y cada una de estas se actualiza con esa información.