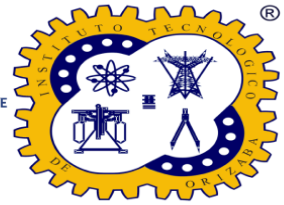




EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNM
TECNOLÓGICO NACIONAL DE
MÉXICO



**Tecnológico Nacional de México
Campus Orizaba**

Tópicos Avanzados de Programación

Ingeniería en Sistemas Computacionales

**Tema 2:
Componentes y librerías**

Integrantes:

**Muñoz Hernández Vania Lizeth – 21011009
Rodríguez Pulido Jesús Raymundo – 21011032
Romero Ovando Karyme Michelle – 21011037**

**Grupo:
4g2A**

Fecha de entrega: 27/Marzo /2023

1. Introducción

En la actualidad las necesidades de desarrollo de aplicaciones son cada vez mayores y necesarias por el uso primordial de las redes de Internet.

Para que los usuarios puedan interactuar con los dispositivos es necesario un medio para poder hacerlo. Aquí es donde entran las interfaces de usuario, la parte que va a interactuar con el usuario.

En esta materia se irán realizando códigos en donde se tenga una idea clara de los conceptos y componentes claves para que se puedan realizar interfaces de usuario que tengan una funcionalidad correcta, en este reporte se señalan los componentes vistos en clase y su importancia a la hora de poder crear una interfaz e ir indicando los pasos básicos a seguir en la construcción de una interfaz gráfica.

2. Competencia específica

Desarrolla soluciones de software para resolver problemas en diversos contextos utilizando programación concurrente, acceso a datos, que soporten interfaz gráfica de usuario y consideren dispositivos móviles.

3. Marco Teórico

La utilización de variables nos ayuda a organizar información cambiante, mientras que las funciones simplifican múltiples acciones. De manera similar, los componentes son estructuras que permiten organizar de manera cómoda y práctica diversos aspectos del desarrollo web. Estos componentes permiten organizar la estructura e información a través del código HTML, el aspecto visual mediante el código CSS, y la funcionalidad por medio del código Javascript. La reutilización de estos componentes en aplicaciones o componentes más grandes es posible gracias a su estructura modular.

En Java, los paquetes (packages) se utilizan para agrupar de manera lógica los componentes relacionados de una aplicación. Los paquetes pueden contener una variedad de elementos, como clases, interfaces, archivos de texto y más. Al utilizar paquetes, podemos organizar y modularizar nuestra aplicación de una manera efectiva, lo que ayuda a mantener una estructura clara y organizada. Los paquetes en Java nos permiten categorizar diferentes estructuras que componen nuestro software, lo que facilita el mantenimiento y la escalabilidad del proyecto en el futuro.

Pero los paquetes no son las únicas herramientas que tenemos para el desarrollo de software, de igual forma contamos con las librerías en Java y otros lenguajes de programación, permiten reutilizar código, lo que significa que es posible utilizar los métodos, clases y atributos que componen la librería, en lugar de tener que implementar esas funcionalidades nosotros mismos.

Para su utilización, no solo tenemos al alcance aquellas que han sido creadas por otros programadores y que ofrece por default Java o el lenguaje de programación que estemos utilizando, sino que además es posible crear nuestras propias librerías y hacer uso de ellas en nuestros propios proyectos. Básicamente, un paquete en Java puede ser considerado como una librería, pero una librería completa en Java puede estar compuesta por muchos paquetes más. Al importar un paquete, podemos acceder a las clases, métodos y atributos que lo conforman, lo que nos permite utilizar las funcionalidades proporcionadas por la librería en nuestro propio proyecto:

- Para importar librerías en Java se usa la palabra reservada **import** seguido de la "ruta" del paquete o clase que deseamos agregar al proyecto. Cabe resaltar que el **import** permite agregar a nuestro proyecto una o varias clases (paquete) según lo necesitemos.

```
1 package TestMetodos;
2
3 import EntradaSalida.Tools;
4 import Metodos.*;
```

En la imagen a partir de la línea 3, estamos diciendo que la clase contenida en el archivo Java, estará en el paquete "paquete/mipaquete", esa será la ruta de acceso a esa clase en el proyecto. En la siguiente línea, por medio del " * " hemos indicado a Java que queremos importar todas las clases pertenecientes a dicho paquete, puede ser una o más.

Ahora, para el desarrollo de nuestros programas se emplean componentes no visuales, de igual forma que pueden crearse componentes visuales, estos son diversos y limitados por la creatividad y necesidades de nuestra aplicación. Cabe mencionar, que pueden variar según la plataforma de desarrollo utilizada.

Los componentes visuales se refieren a objetos gráficos que aparecen en la interfaz de usuario de una aplicación, como etiquetas, campos de texto, botones personalizados, gráficos, tablas, barras de progreso, menús desplegables, controles de entrada de datos personalizados, ventanas emergentes y elementos de diseño personalizados.

En cuanto a los componentes no visuales definidos por el usuario, son elementos que no se ven directamente en la interfaz de usuario, pero que realizan una función detrás de escena, algunos ejemplos podrían ser clases y objetos personalizados para interactuar con bases de datos, servicios web personalizados, herramientas de procesamiento de datos personalizadas, y todo tipo de lógica de negocio personalizada.

Al permitir que los usuarios creen sus propios componentes, se puede mejorar la eficiencia y la personalización de una aplicación. Los programadores pueden diseñar componentes que cumplan con las necesidades específicas de su aplicación, en lugar de depender de los componentes predefinidos que se proporcionan con el lenguaje de programación o la plataforma de desarrollo. Además, los componentes definidos por el usuario pueden ser reutilizados en diferentes proyectos, lo que ahorra tiempo y esfuerzo en el proceso de desarrollo de software.

Teniendo en cuenta lo anterior, todo proyecto se compone de paquetes los que, a su vez, se componen de clases. Cuando no se especifica la declaración **package** se usa el paquete predeterminado (o global). Además, el paquete predeterminado no tiene nombre, lo que lo hace transparente.

Para crear un paquete, coloque un comando **package** en la parte superior del archivo fuente de java, o bien, trabajando en un proyecto con NetBeans, comprobaremos que en la ventana *Projects* los paquetes se representan con un icono específico y el menú contextual del proyecto nos ofrece la opción *New>Java Package*, que será el que usemos habitualmente para crear un nuevo paquete

Las clases declaradas dentro de ese archivo pertenecerán al paquete especificado. Puesto que un paquete define un espacio de nombre, los nombres de las clases que coloque dentro del archivo se convierten en parte de ese espacio del nombre del paquete.

Para utilizar los elementos de un paquete es necesario importar este en el módulo de código en curso, usando para ello la sentencia **import**:

```
import package.clase1;
```

Esta línea hace la importación del paquete “package”, siguiendo la ruta a la clase “clase1”, pero también se puede incluir la ruta completa del paquete, sustituyendo con un “ * “ posterior al texto:

```
import package.*;
```

Finalmente, la creación de librerías definidas por el usuario (así como su uso), tiene un proceso un poco más largo que trabajar en la creación de paquetes, sin embargo, su utilidad es mucha, ya que son una forma eficaz de crear y reutilizar código personalizado en múltiples proyectos, lo que puede mejorar la eficiencia del desarrollo y reducir los errores en el código.

Para crear una librería personalizada en Java, primero se deben escribir las clases y los paquetes que conformarán la librería. Luego, se pueden empaquetar en un archivo JAR (Java Archive), que es un archivo comprimido que contiene los archivos de clase de Java y otros recursos necesarios para la aplicación.

Una vez que se ha creado una librería personalizada, se puede utilizar en cualquier proyecto que utilice Java simplemente importando las clases y paquetes de la librería en el código del proyecto. Esto facilita la reutilización del código y reduce la cantidad de código redundante que se debe escribir en cada proyecto.

4. Material y Equipo

El material y equipo que se necesita para llevar a cabo la práctica son:

- ✓ Computadora
- ✓ Software y versión usados
- ✓ Materiales de apoyo para el desarrollo de la práctica

5. Desarrollo de la Práctica

Clase	Método	Descripción
Dimension		La clase dimensión contiene la altura y el ancho de un componente en un número entero, así como también en doble precisión.
ActionEvent		Representa una acción del usuario en la interfaz.

Clase	Método	Descripción
ChangeListener	ChangeListener	Recibe notificaciones cuando cambia el valor de una propiedad.
	stateChanged(ChangeEvent)	Se invoca cuando el destino del listener ha cambiado de estado.
ChangeEvent	ChangeEvent	Notifica a las partes interesadas que el estado ha cambiado en el origen del evento.
	void stateChanged(ChangeEvent)	Se le llama cuando el componente escuchado cambia de estado.

Clase	Método	Descripción
PlotOrientation		Se utiliza para indicar la orientación (horizontal o vertical) de un gráfico 2D.
ChartFactory		Una colección de métodos de utilidad para crear algunos gráficos estándar con JFreeChart.

Clase	Método	Descripción
DefaultCategoryDataset	DefaultCategoryDataset ()	Crea un nuevo conjunto de datos (vacío).
ChartPanel	ChartPanel()	El panel se registra en el gráfico para recibir notificaciones de cambios en cualquier componente del gráfico.

Clase	Método	Descripción
JRadioButton	JRadioButton()	Constructor de la clase ButtonGroup.

	Permite seleccionar solo una opción de un conjunto de opciones.
setBackground()	Establece el color de fondo de este componente.
setText()	Establece el texto del botón.
setBounds()	Mueve y cambia el tamaño de este componente.
addChangeListener()	Agrega un ChangeListener al botón.
add()	Agrega el menú emergente especificado al componente.

Clase	Método	Descripción
JCheckBox	JCheckBox()	Constructor de la clase JCheckBox.
	addItemListener()	Se conecta el componente con un objeto de la clase que maneja los sucesos originados en dicho componente.
	add()	Agrega el menú emergente especificado al componente.
	isSelected()	Devuelve true si el componente está seleccionado, en caso contrario devuelve false

Clase	Método	Descripción
DefaultListModel	DefaultListModel()	Constructor de la clase DefaultListModel.
		Permite realizar más acciones (como agregar, obtener, eliminar, ajustar),
	int size()	Devuelve el número de componentes de esta lista.
	void addElement (E element)	Agrega el componente especificado al final de esta lista.
	getSize()	Devuelve el número de componentes de esta lista.

e remove(int index)	Elimina el elemento en la posición especificada en esta lista.
void removeElementAt(int index)	Elimina el componente en el índice especificado.
void clear()	Elimina todos los elementos de esta lista.

Clase	Método	Descripción
JList	JList()	Constructor de la clase JList.
	void setText(String txt)	Define una línea de texto que mostrará el componente.
	void setSelectionMode(int modoSelección)	Establece el modo de selección de la lista.
	void setModel(ListModel <E> model)	Establece el modelo que representa el contenido de la lista.
	int getSelectedIndex()	Devuelve el índice seleccionado.
	void clearSelection()	Borra la selección; después de llamar a este método, isEmptySelection devolverá true.

Clase	Método	Descripción
JFrame	Frame()	Constructor de la clase JFrame.
	void setTitle(String titulo)	Establece el título de la ventana con el String especificado.
	void setSize(int x, int y)	Cambia el tamaño del componente para que tenga una anchura x y una altura y.
	void setLocationRelativeTo(Component c)	Establece la ubicación de la ventana en relación con el componente especificado.
	void setDefaultCloseOperation(opciones)	Usado para especificar una de las siguientes opciones del botón de cierre EXIT_ON_CLOSE.

	void setResizable (boolean resizable)	Para evitar que se cambie el tamaño de la ventana.
	void setVisible (boolean b)	Muestra u oculta la ventana según el valor del parámetro b.
ActionListener	ActionListener	Interface que debe ser implementada para gestionar eventos.
	void actionPerformed (actionEvent e)	se invoca cuando ocurre u evento.
Container	Container getContentPane()	Retorna un objeto ContentPane de la ventana.
	void setLayout (LayoutManager mgr)	Establece el layout de la ventana.
	Component add (Component comp)	Añade el componente especificado al final del contenedor.
Component	void addActionListener (this)	Añade un oyente de eventos al componente actual.
	void setBounds (int x, int y, int ancho, int alto)	Mueve y cambia el tamaño del componente.
JLabel	JLabel() void setText(String txt)	Constructor de la clase JLabel. Define una línea de texto que mostrará el componente.
TextField	TextField() String getText()	Constructor de la clase TextField. Retorna el texto contenido en el componente de texto.
JButton	JButton() void setText(String txt)	Constructor de la clase JButton. Define una línea de texto que mostrará el componente.
DefaultListModel	DefaultListModel() void addElement (Element e)	Constructor de la clase DefaultListModel. Añade el componente especificado al final de la lista.
	void removeElementAt (int índice)	Elimina el componente en el índice especificado.

	void removeAllElements()	Elimina todos los componentes de la lista y coloca su tamaño en cero.
	void clear()	Elimina todos los elementos de la lista.
JScrollPane	JScrollPane()	Descripcion de la clase JScrollPane.
	void setViewportView (component view)	Crea una ventana grafica si es necesario y luego establece su vista.
Event	Object getSource()	El objeto sobre el cual el evento inicialmente ha ocurrido.
JOptionPane	void showMessageDialog (Component componentePadre, Object mensaje)	Crea un cuadro de dialogo.

6. Resultados

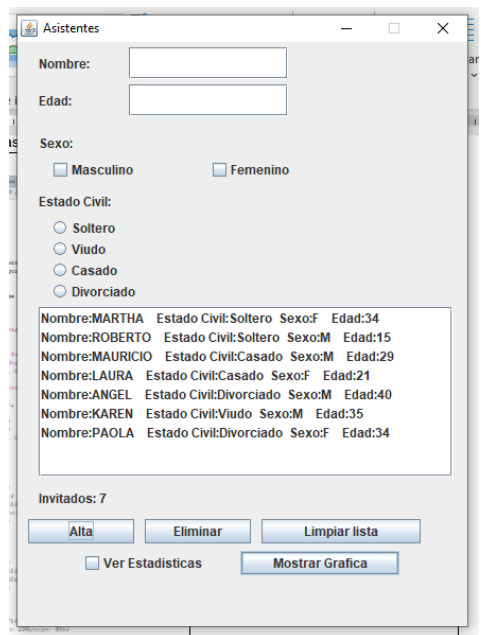
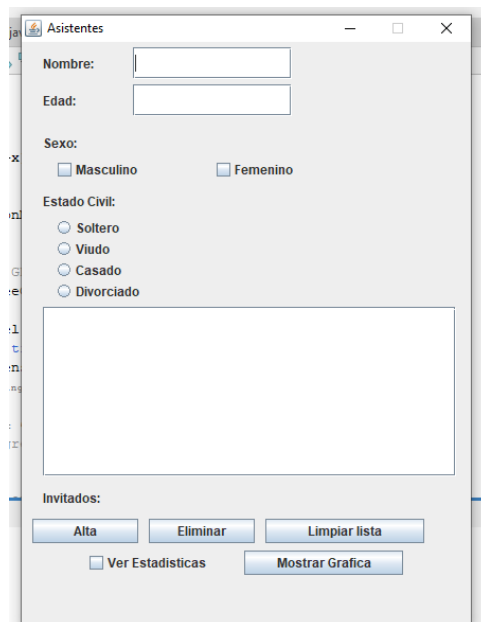
Presentar la clase que genera el GUI e imágenes que señalen los componentes usados e imagen con breves descripciones del funcionamiento de este.

Clase

Clase GUIFiesta:

```
1 package asistentes;
2 import java.awt.Container;
3 import java.awt.Dimension;
4 import javax.swing.*;
5 import java.awt.event.*;
6 import javax.swing.event.*;
7 import org.jfree.chart.*;
8 import org.jfree.chart.plot.PlotOrientation;
9 import org.jfree.data.category.CategoryDataset;
10
11 //CLASE GUIFIESTA
12 public class GUIFiesta extends JFrame implements ActionListener,
13     ChangeListener, ItemListener {
14     //VARIABLES DE LA CLASE
15
16     private final ListaPersonas lista;
17     private Container contenedor;
18
19     private JRadioButton Masculino, Femenino, Soltero,
20         Casado, Viudo, Divorciado;
21     private ButtonGroup BotonesSexo, BotonesEC;
22     private JCheckBox Ver;
23     private JLabel EstadoCivil, Nombre, Edad, Registro, Sexo;
24     private JTextField Nom, Ed;
25     private JButton Añadir, Eliminar, EliminarTodo, ImprimirGrafica;
26     private JList listaNombres;
27     private DefaultListModel modelo;
28     private JScrollPane scrollLista;
29     private String EdoCivil, Genero, Name;
30     private byte Age;
31     private boolean Des;
32
33     public GUIFiesta() {
34         lista=new ListaPersonas();
35         inicio();
36         setTitle("Asistentes");
37         setSize(width: 450, height: 750);
38         setLocationRelativeTo(e: null);
39         setDefaultCloseOperation(operation: JFrame.EXIT_ON_CLOSE);
40         setResizable(resizable: false);
41     }
42
43     private void inicio(){
44
45         contenedor=getContentPane();
46         contenedor.setLayout(null);
47         BotonesSexo=new ButtonGroup();
48         BotonesEC=new ButtonGroup();
49
50         //Parte del nombre
51         Nombre=new JLabel();
52         Nombre.setText(text: "Nombre:");
53         Nombre.setBounds(x: 5,y: 5,width: 100,height: 30);
54         Nom=new JTextField();
55         Nom.setBounds(x: 105,y: 5,width: 150,height: 30);
56
57         //Parte de la edad
58         Edad=new JLabel();
59         Edad.setText(text: "Edad:");
60         Edad.setBounds(x: 5,y: 40,width: 100,height: 30);
61         Ed=new JTextField();
62         Ed.setBounds(x: 105,y: 40,width: 150,height: 30);
63     }
```

Ejecución



```

66 Sexo=new JLabel();
67 Sexo.setText(text: "Sexo:");
68 Sexo.setBounds(x: 5,y: 80,width: 50,height: 30);
69
70 Masculino=new JRadioButton(text: "Masculino");
71 Masculino.setBounds(x: 65,y: 80,width: 100,height: 30);
72 Masculino.addChangeListener(l: this);
73 add(comp: Masculino);
74 BotonesSexo.add(b: Masculino);
75
76 Femenino=new JRadioButton(text: "Femenino");
77 Femenino.setBounds(x: 65,y: 115,width: 100,height: 30);
78 Femenino.addChangeListener(l: this);
79 add(comp: Femenino);
80 BotonesSexo.add(b: Femenino);
81
82 //Parte de estado civil
83 EstadoCivil=new JLabel();
84 EstadoCivil.setText(text: "Estado Civil");
85 EstadoCivil.setBounds(x: 215,y: 80,width: 100,height: 30);
86
87 Soltero=new JRadioButton(text: "Soltero");
88 Soltero.setBounds(x: 320,y: 80,width: 100,height: 30);
89 Soltero.addChangeListener(l: this);
90 add(comp: Soltero);
91 BotonesEC.add(b: Soltero);
92
93 Viudo=new JRadioButton(text: "Viudo");
94 Viudo.setBounds(x: 320,y: 115,width: 100,height: 30);
95 Viudo.addChangeListener(l: this);
96 add(comp: Viudo);
97 BotonesEC.add(b: Viudo);
98
99 Casado=new JRadioButton(text: "Casado");
100 Casado.setBounds(x: 320,y: 150,width: 100,height: 30);
101 Casado.addChangeListener(l: this);
102 add(comp: Casado);
103 BotonesEC.add(b: Casado);
104
105 Divorciado=new JRadioButton(text: "Divorciado");
106 Divorciado.setBounds(x: 320,y: 185,width: 100,height: 30);
107 Divorciado.addChangeListener(l: this);
108 add(comp: Divorciado);
109 BotonesEC.add(b: Divorciado);
110
111 //Registro
112 Registro=new JLabel();
113 Registro.setText(text: "Invitados: ");
114 Registro.setBounds(x: 5,y: 385,width: 135,height: 20);
115
116 jPanell=new JPanel();
117 jPanell.setBounds(x: 5,y: 410,width: 420,height: 200);
118
119 Añadir=new JButton();
120 Añadir.setText(text: "Alta");
121 Añadir.setBounds(x: 10,y: 650,width: 100,height: 23);
122 Añadir.addActionListener(l: this);
123
124 Eliminar=new JButton();
125 Eliminar.setText(text: "Eliminar");
126 Eliminar.setBounds(x: 280,y: 650,width: 150,height: 23);
127 Eliminar.addActionListener(l: this);
128
129 EliminarTodo=new JButton();
130 EliminarTodo.setText(text: "Limpiar lista");
131 EliminarTodo.setBounds(x: 280,y: 680,width: 150,height: 23);
132 EliminarTodo.addActionListener(l: this);
133
134 ImprimirGrafica=new JButton();
135 ImprimirGrafica.setText(text: "Mostrar Grafica");
136 ImprimirGrafica.setBounds(x: 220,y: 680,width: 150,height: 20);
137 ImprimirGrafica.addActionListener(l: this);
138
139 Ver=new JCheckBox();
140 Ver.setText(text: "Ver Estadísticas");
141 Ver.setBounds(x: 60,y: 680,width: 150,height: 20);
142 add(comp: Ver);
143 Ver.addItemListener(l: this);
144
145 listaNombres=new JList();
146 listaNombres.setSelectionMode(
147     selectionMode: ListSelectionMode.SINGLE_SELECTION);
148 modelo=new DefaultListModel();
149
150 scrollLista=new JScrollPane();
151 scrollLista.setBounds(x: 10,y: 230,width: 420,height: 150);
152 scrollLista.setViewportView(view: listaNombres);

```

Asistentes

Nombre:

Edad:

Sexo:

☐ Masculino ☐ Femenino

Estado Civil:

☐ Soltero ☐ Viudo ☐ Casado ☐ Divorciado

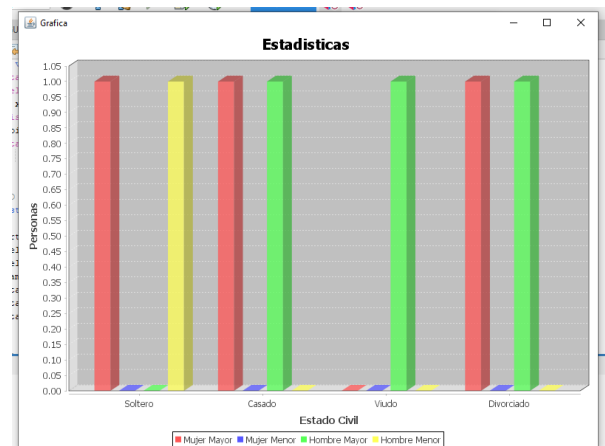
Nombre:LAURA Estado Civil:Casado Sexo:F Edad:21
 Nombre:ANGEL Estado Civil:Divorciado Sexo:M Edad:40
 Nombre:KAREN Estado Civil:Viudo Sexo:M Edad:35
 Nombre:PAOLA Estado Civil:Divorciado Sexo:F Edad:34

Personas Mayores: 6 Personas Menores: 1
 Solteros: 2 Casados: 2 Viudos: 1 Divorciados: 2
 Hombres: 4 Porcentaje: 57.14% Mujeres: 3 Porcentaje: 42.86%

Invitados: 7

Alta Eliminar Limpiar lista

☒ Ver Estadísticas



```

155     contenedor.add(comp: Nom);
156     contenedor.add(comp: Edad);
157     contenedor.add(comp: Ed);
158     contenedor.add(comp: Sexo);
159     contenedor.add(comp: Masculino);
160     contenedor.add(comp: Femenino);
161     contenedor.add(comp: EstadoCivil);
162     contenedor.add(comp: Soltero);
163     contenedor.add(comp: Viudo);
164     contenedor.add(comp: Casado);
165     contenedor.add(comp: Divorciado);
166     contenedor.add(comp: Registro);
167     contenedor.add(comp: AÑadir);
168     contenedor.add(comp: Eliminar);
169     contenedor.add(comp: EliminarTodo);
170     contenedor.add(comp: ImprimirGrafica);
171     contenedor.add(comp: Ver);
172     contenedor.add(comp: scrollLista);
173 }
174
175 //METODO PARA ACCION DE BOTONES
176 @Override
177 public void actionPerformed(ActionEvent evento) {
178     if(evento.getSource() == AÑadir) {
179         Des=ValidaEntrada();
180         if(Des==true)
181             AÑadirPersona();
182     }
183     if(evento.getSource() == Eliminar)
184         EliminarPersona(indice: listaNombres.getSelectedIndex());
185     if(evento.getSource() == EliminarTodo)
186         EliminarTodo();
187     if(evento.getSource() == ImprimirGrafica) {
188         lista.Actualizar();
189         grafica3D(datos: ListaPersonas.creaDatosCategory());
190     }
191 }
192
193 //ETODO PARA JBUTTON
194 @Override
195 public void stateChanged(ChangeEvent e) {
196
197     if(Masculino.isSelected()) {
198         Genero=Masculino.getLabel();
199     }
200     if(Femenino.isSelected()) {
201         Genero=Femenino.getLabel();
202     }
203     if(Soltero.isSelected()) {
204         EdoCivil=Soltero.getLabel();
205     }
206     if(Viudo.isSelected()) {
207         EdoCivil=Viudo.getLabel();
208     }
209     if(Casado.isSelected()) {
210         EdoCivil=Casado.getLabel();
211     }
212     if(Divorciado.isSelected()) {
213         EdoCivil=Divorciado.getLabel();
214     }
215 }
216 @Override
217 public void itemStateChanged(ItemEvent evento) {
218     if(Ver.isSelected()) {
219         MostrarCosas();
220     }
221     AÑadir.setEnabled(b: false);
222     Eliminar.setEnabled(b: false);
223     EliminarTodo.setEnabled(b: false);
224 }
225 else {
226     OcultarCosas();
227     AÑadir.setEnabled(b: true);
228     Eliminar.setEnabled(b: true);
229     EliminarTodo.setEnabled(b: true);
230 }
231 }
232 }
233
234 //METODO PARA VALIDAR ENTRADAS
235 private boolean ValidaEntrada() {
236     Name=Nom.getText();
237     Age=Byte.parseByte(Ed.getText());
238     if(Name!=null && Name.matches(regex: "([a-z A-Z]|\\s)+")) {
239         if(Age>0 && Age<150) {
240             return true;
241         }
242     }

```

```

243     else{
244         JOptionPane.showMessageDialog(parentComponent:null,
245             message:"Ingrese una edad valida");
246         Ed.setText(c: "");
247         return false;
248     }
249 }
250
251     else{
252         JOptionPane.showMessageDialog(parentComponent:null,
253             message:"Ingrese un nombre valido");
254         Nom.setText(c: "");
255         return false;
256     }
257 }
258
259 //METODO PARA LIMPIAR ENTRADAS
260 private void Limpiar() {
261     Nom.setText(c: "");
262     Ed.setText(c: "");
263     BotonesSexo.clearSelection();
264     BotonesEC.clearSelection();
265 }
266
267 //METODO AÑADIR PERSONAS
268 private void AñadirPersona() {
269     Persona p=new Persona(nombrePersona: Name,
270         edadPersona:Age, Sexo: Genero.charAt(index: 0), EstadoCivil: EdoCivil);
271     lista.Añadir(p: p);
272
273     Genero.charAt(index: 0) + "    Edad:" + Ed.getText();
274     modelo.addElement(element: cad);
275
276     listaNombres.setModel(model: modelo);
277
278     int x= modelo.getSize();
279     Registro.setText("Invitados: " + x);
280     Limpiar();
281 }
282
283 //METODO PARA OCULTAR ESTADISTICAS
284 private void OcultarCosas() {
285     for(int i=1; i<=4; i++) {
286         modelo.remove(modelo.size()-1);
287     }
288 }
289
290 //METODO PARA MOSTRAR ESTADISTICAS
291 private void MostrarCosas() {
292     modelo.addElement(element: "\r\n");
293     modelo.addElement(element: lista.CalcularTotal());
294     modelo.addElement(element: lista.CalcularEstadoCivil());
295     modelo.addElement(element: lista.CalcularSexo());
296     listaNombres.setModel(model: modelo);
297 }
298
299 //METODO PARA ELIMINAR UNA PERSONA SELECCIONADA
300 private void EliminarPersona(int indice) {
301     if(indice>=0) {
302         modelo.removeElementAt(index: indice);
303         lista.Eliminar(pos: indice);
304
305         int x=modelo.getSize();
306         Registro.setText("Invitados: "+x);
307         Limpiar();
308         listaNombres.clearSelection();
309     }
310     else{
311         JOptionPane.showMessageDialog(parentComponent:null,
312             message:"Debe seleccionar un elemento");
313     }
314 }
315
316 //METODO PARA LIMPIAR TODA LA LISTA
317 private void EliminarTodo() {
318     lista.EliminarTodo();
319     modelo.clear();
320     int x=modelo.getSize();
321     Registro.setText("Invitados: "+x);
322     Limpiar();
323     listaNombres.setSelectionMode(
324         selectionMode: ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
325 }
326

```

```

328 //METODO CREAR NUEVO JFrame CON LA GRAFICA
329 public static void panelJframe(JFreeChart grafica){
330
331     ChartPanel panel= new ChartPanel(chart: grafica);
332     panel.setMouseWheelEnabled(flag: true);
333     panel.setPreferredSize(new Dimension(i: 500,ii: 300));
334     JFrame ventana = new JFrame(string: "Grafica"); // titulo del frame
335     ventana.setVisible(b: true);
336     ventana.setSize(width: 800, height: 600); // tamaño del JFrame
337     ventana.add(comp: panel); // se agrega al JFrame el panel(Grafico)
338 }
339
340 //METODO PARA LA GRAFICA DE BARRAS 3D
341 public static void grafica3D(CategoryDataset datos) {
342     JFreeChart grafico;
343     grafico=ChartFactory.createBarChart3D(
344         title: "Estadísticas ", // nombre del grafico
345         categoryAxisLabel: "Estado Civil", //nombre de las barras
346         valueAxisLabel: "Personas", // nombre de la numeracio
347         dataset:datos, // datos del grafico
348         orientation:PlotOrientation.VERTICAL, //orientacion
349         legend: true, // leyenda de barras individuales por columna
350         tooltips: true, //herramientas
351         url: false// url del grafico
352     );
353     panelJframe(grafica:grafico);
354 }
355 }

```

Pasos para realizar la interfaz y su funcionamiento

1. Esta clase GUIFiesta se encargará únicamente de organizar los elementos para la interfaz y las acciones de aquellos botones que nosotros designemos.
2. Se comienza importando las clases que utilizaremos (java.awt.Container, java.awt.Dimension, javax.swing.*, java.awt.event.*, org.jfree.chart.*, org.jfree.chart.plot.PlotOrientation, org.jfree.data.category.CategoryDataset).
3. Posteriormente al comenzar a crear la clase se debe realizar un extends de la clase JFrame y un implements para las clases ActionListener, ChangeListener e ItemListener, las ocuparemos para darle acciones a los Botones que se usaran en la interfaz.
4. Antes de iniciar con la interfaz declararemos las variables que utilizaremos para nuestra interfaz.

```

private final ListaPersonas lista;
private Container contenedor;

private JRadioButton Masculino, Femenino, Soltero,
    Casado, Viudo, Divorciado;
private ButtonGroup BotonesSexo, BotonesEC;
private JCheckBox Ver;
private JLabel EstadoCivil, Nombre, Edad, Registro, Sexo;
private JTextField Nom, Ed;
private JButton Añadir, Eliminar, EliminarTodo, ImprimirGrafica;
private JList listaNombres;
private DefaultListModel modelo;
private JScrollPane scrollLista;
private String EdoCivil, Genero, Name;
private byte Age;
private boolean Des;

```

5. Cada variable de tipo (JRadioButton, Container, JCheckBox, JLabel, JTextField, JButton, JList, JScrollPane) serán elementos que se usaran en la interfaz
6. Después se usara un constructor de la clase que iniciara la interfaz cuando la clase main cree un objeto de esta clase, este constructor instanciara la variable lista de la clase ListaPersona(), y este mismo constructor creara la ventana con las dimensiones que le asignemos y así mismo el nombre de dicha ventana.


```

34 public GUIFiesta() {
35     lista=new ListaPersonas();
36     inicio();
37     setTitle("Asistentes");
38     setSize(width: 450 ,height: 750);
39     setLocationRelativeTo(null);
40     setDefaultCloseOperation(operation: JFrame.EXIT_ON_CLOSE);
41     setResizable(resizable: false);
42 }

```

7. Ahora, como en el constructor de la clase GUIFiesta manda a llamar el método inicio(), se tiene que crear este método de tipo void, y lo que hará será asignar los nombres, dimensiones y ubicaciones de los elementos de la interfaz. (Por ejemplo primero instancia los grupos de botones que estarán presentes en la interfaz y luego comienza ubicando la etiqueta "Nombre" y su cuadro de texto en ciertas coordenadas de la ventana)

```

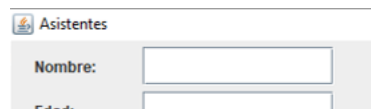
private void inicio() {

    contenedor=getContentPane();
    contenedor.setLayout(null);
    BotonesSexo=new ButtonGroup();
    BotonesEC=new ButtonGroup();

    //Parte del nombre
    Nombre=new JLabel();
    Nombre.setText("Nombre:");
    Nombre.setBounds(x: 5,y: 5,width: 100,height: 30);
    Nom=new JTextField();
    Nom.setBounds(x: 105,y: 5,width: 150,height: 30);
}

```

aquí vemos como el JLabel de Nombre se posiciona a la izquierda y su cuadro de texto a la derecha



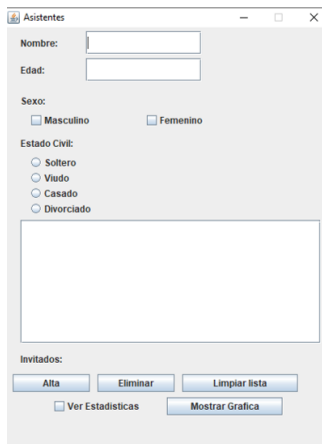
8. **NOTA:** No hay que olvidar añadir cada elemento al contenedor con el método ".add(elemento)". por ejemplo:

```

contenedor.add(comp: Nom);
contenedor.add(comp: Edad);
contenedor.add(comp: Ed);
contenedor.add(comp: Sexo);
contenedor.add(comp: Masculino);
contenedor.add(comp: Femenino);
contenedor.add(comp: EstadoCivil);
contenedor.add(comp: Soltero);
contenedor.add(comp: Viudo);
contenedor.add(comp: Casado);
contenedor.add(comp: Divorciado);

```

9. Se realizará el paso anterior para todos los elementos que deseemos agregar a la interfaz, de esta forma lograremos conseguir esta estructura.



10. Recordemos agregar los elementos de tipo JRadioButton a su ButtonGroup, para así evitar la múltiple selección de datos, y así mismo a cada JButton escribirle el método “.addActionListener(this)” para mas adelante darle acciones a los botones
11. Una vez añadidos todos los elementos en la ventana, seguiremos con los métodos que se usaran para describir las acciones que tendrá cada JButton, JCheckBox y JRadioButton.
12. Comenzamos con el método para las acciones del JButton el cual es el siguiente.
Debido a que tenemos 4 botones, serán 4 eventos posibles.

```
@Override
public void actionPerformed(ActionEvent evento){
    if(evento.getSource() == Añadir){
        Des=ValidaEntrada();
        if(Des==true)
            AñadirPersona();
    }
    if(evento.getSource() == Eliminar)
        EliminarPersona(indice: listaNombres.getSelectedIndex());
    if(evento.getSource() == EliminarTodo)
        EliminarTodo();
    if(evento.getSource() == ImprimirGrafica){
        lista.Actualizar();
        grafica3D(datos: ListaPersonas.creaDatosCategory());
    }
}
```

Evento 1 “Añadir una persona”: antes de añadir una persona se deberá verificar que los datos ingresados estén bien escritos(el nombre no puede llevar caracteres especiales o números, y la edad tiene que ser positiva), sabiendo esto se comienza con la primera condición donde si se aprieta el botón añadir se realizara lo siguiente, a la variable “Des” que es de tipo boolean sera el resultado del método ValidaEntrada(),y luego se vuelve a hacer otra decisión donde si “Des” es true, significa que los datos si son correcto, por lo tanto se usara otro método de esta clase que se llama AñadirPersona()).

NOTA: el método ValidaEntrada() es el siguiente.

```
236 private boolean ValidaEntrada(){
237     Name=Nom.getText();
238     Age=Byte.parseByte(s: Ed.getText());
239     if(Name!=null && Name.matches(regex: "([a-z A-Z]|\\s)+")){
240         if(Age>0 && Age<150){
241             return true;
242         }
243     }
244     else{
245         JOptionPane.showMessageDialog(parentComponent:null,
246             message:"Ingrese una edad valida");
247         Ed.setText(s: "");
248         return false;
249     }
250     else{
251         JOptionPane.showMessageDialog(parentComponent:null,
252             message:"Ingrese un nombre valido");
253         Nom.setText(s: "");
254         return false;
255     }
256 }
```

Este método recupera el Nombre y la edad en una variable diferente, y con ayuda de un if else compara las 2 variables y de esta forma retornara un true o un false dependiendo si están bien las entradas

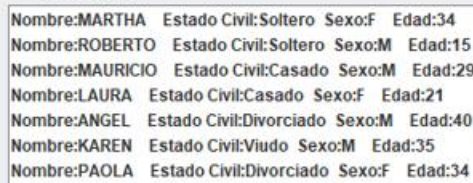
NOTA: el método AñadirPersona() es el siguiente:

```
//METODO AÑADIR PERSONAS
private void AñadirPersona() {
    Persona p=new Persona(nombrePersona: Name,
    edadPersona:Age,Sexo:Genero.charAt(index: 0),EstadoCivil:EdoCivil);
    lista.Añadir(p: p);

    Genero.charAt(index: 0) +"    Edad:"+Ed.getText();
    modelo.addElement(element:cad);

    listaNombres.setModel(model: modelo);

    int x= modelo.getSize();
    Registro.setText("Invitados: " + x);
    Limpiar();
}
```



Invitados: 7

Este método lo que realiza es una instanciación de un objeto de la clase Persona con los parametros que solicita su constructor(Nombre, Edad, Genero y Estado Civil), una vez creado el objeto se almacena en la variable “lista” que es un array creado en esta clase, y posteriormente se añade una cadena a la variable “modelo” con “.addElement(cad), y se actualiza la variable listaNombres, también se actualiza la cantidad de registros que se han hecho iniciando una variable local “x” que tendrá el valor del tamaño de la variable “modelo” y una vez actualizado todo en el registro se termina usando el método Limpiar() para vaciar los cuadros de textos y así quitar los datos seleccionados.

NOTA: el método Limpiar() es el siguiente:

Este método realiza 2 acciones , la primera es limpiar los cuadros de textos con ayuda de un getText("") ya que cambiara todo el contenido del cuadro de texto por un espacio en blanco, y los botones se limpiaran con el método de .clearSelection(), ya que esta opción quita todas las selecciones en los JCheckBox y JRadioButton

Evento 2 “EliminarPersona”: este evento usara el método EliminarPersona(listaNombres.getSelectedIndex()), el “.getSelectedIndex()” básicamente la persona que hayamos seleccionado dentro de la lista de registros.

NOTA: El método EliminarPersona(int n) es el siguiente:

```
//METODO PARA ELIMINAR UNA PERSONA SELECCIONADA
private void EliminarPersona(int indice){
    if(indice>=0){
        modelo.removeElementAt(indice: indice);
        lista.Eliminar(p:=indice);

        int x=modelo.getSize();
        Registro.setText("Invitados: "+x);
        Limpiar();
        listaNombres.clearSelection();
    }
    else{
        JOptionPane.showMessageDialog(parentComponent:null,
        message:"Debe seleccionar un elemento");
    }
}
```

Este método recibe una variable de tipo entera que se llamara índice, realizara una condición para verificar que el elemento a eliminar si tenga una posición en el array, de esta forma con el método .removeElementAt(índice) removerá el elemento que se encuentra en el array en la posición seleccionada, posteriormente se limpiaran las entradas y se actualizara la cantidad de invitados que hay

Evento 3 “EliminarTodo” : este evento manda a llamar al método EliminarTodo()

NOTA: El Método EliminarTodo() es el siguiente:

```
//METODO PARA LIMPIAR TODA LA LISTA
private void EliminarTodo() {
    lista.EliminarTodo();
    modelo.clear();
    int x=modelo.getSize();
    Registro.setText("Invitados: "+x);
    Limpiar();
    listaNombres.setSelectionMode(
        selectionMode: ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
}
```

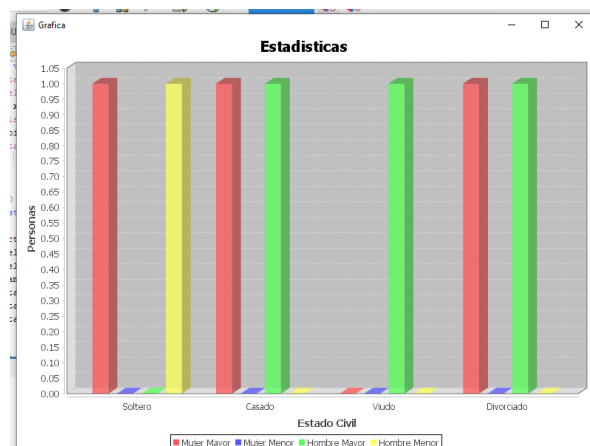
La variable “lista” manda a llamar al método EliminarTodo() de su clase (“ListaPersonas”), después la variable “modelo” se limpia con el método .clear() y de misma forma se actualizan las entradas y se limpian las casillas.

Evento 4 “ImprimirGrafica”: este evento manda a llamar al método Actualizar() que está en la clase “ListaPersonas” y después una vez actualizados los datos llama al método grafica3D(listaPersona.creaDatosCategory).

NOTA: El método grafica3D es el siguiente:

```
340 //METODO PARA LA GRAFICA DE BARRAS 3D
341 public static void grafica3D(CategoryDataset datos) {
342     JFreeChart grafico:
343     grafico=ChartFactory.createBarChart3D(
344         title: "Estadísticas ", // nombre del grafico
345         categoryAxisLabel: "Estado Civil", //nombre de las barras
346         valueAxisLabel: "Personas", // nombre de la numeracio
347         dataset: datos, // datos del grafico
348         orientation: PlotOrientation.VERTICAL, //orientacion
349         legend: true, // leyenda de barras individuales por columna
350         tooltips: true, //herramientas
351         url: false// url del grafico
352     );
353     panelJframe(grafica:grafico);
354 }
355 }
```

Este método recibe un elemento que se llamara datos y es de tipo CategoryDataset, y con ayuda de unas librerías importadas al proyecto realizara una ventana adicional donde se imprimirá una gráfica de título “Estadísticas” y con la distribución correcta se lograra observar así la ventana:



13. Ahora seguirá el método para darle valor a los JRadioButton, este método es el siguiente:

```
// Método para dar valor a los JRadioButton
@Override
public void stateChanged(ChangeEvent e) {

    if (Masculino.isSelected()) {
        Genero = Masculino.getLabel();
    }
    if (Femenino.isSelected()) {
        Genero = Femenino.getLabel();
    }
    if (Soltero.isSelected()) {
        EdoCivil = Soltero.getLabel();
    }
    if (Viudo.isSelected()) {
        EdoCivil = Viudo.getLabel();
    }
    if (Casado.isSelected()) {
        EdoCivil = Casado.getLabel();
    }
    if (Divorciado.isSelected()) {
        EdoCivil = Divorciado.getLabel();
    }
}
```

Al ser JRadioButton que se encuentran en un mismo grupo de botones solo se podrá seleccionar uno, por lo tanto, con una variable llamada "Genero" se le asignará dependiendo que JRadioButton seleccione con ayuda de un ".getLabel", y de misma forma se ocupara para otra variable llamada "EdoCivil", ya que en nuestra interfaz se ocuparon 2 grupos de botones, uno para género y otro para Estado Civil

Clase

Clase ListaPersonas:

```

1 package asistentes;
2
3 //Import de las bibliotecas que utilizaremos
4 import java.util.ArrayList;
5 import org.jfree.data.category.CategoryDataset;
6 import org.jfree.data.category.DefaultCategoryDataset;
7
8 //CLASE ListaPersonas
9 public class ListaPersonas {
10     private static int MMS, MHS, MMC, MHC, MMV, MHV, MMD, MHD, mMS,
11         mHS, mMC, mHC, mMV, mHV, mMD, mHD;
12     private int Mayor, Menor, Hombre, Mujer;
13     private ArrayList<Persona> Lista;
14     private String Cadena;
15
16     public ListaPersonas() {
17         Lista = new <Persona>ArrayList();
18     }
19
20     public void Añadir(Persona P) {
21         Lista.add(e: P);
22     }
23
24     public void Eliminar(int pos) {
25         Lista.remove(index: pos);
26     }
27
28     public boolean EliminarTodo() {
29         return Lista.removeAll(e: Lista);
30     }
31
32     public void Actualizar() {
33
34         MMS=0;MHS=0;MMC=0;MHC=0;MMV=0;MHV=0;MMD=0;MHD=0;mMS=0;
35         mHS=0;mMC=0;mHC=0;mMV=0;mHV=0;mMD=0;mHD=0;
36         for(Persona P:Lista) {
37             String cad=P.getEdad()+P.getEstadoCivil()+P.Sexo();
38             switch (cad) {
39                 case "MayorSolteroMasculino" : MHS++; break;
40                 case "MayorSolteroFemenino" : MMS++; break;
41                 case "MayorCasadoMasculino" : MHC++; break;
42                 case "MayorCasadoFemenino" : MMC++; break;
43                 case "MayorViudoMasculino" : MHV++; break;
44                 case "MayorViudoFemenino" : MMV++; break;
45                 case "MayorDivorciadoMasculino" : MHD++; break;
46                 case "MayorDivorciadoFemenino" : MMD++; break;
47                 case "MenorSolteroMasculino" : mHS++; break;
48                 case "MenorSolteroFemenino" : mMS++; break;
49                 case "MenorCasadoMasculino" : mHC++; break;
50                 case "MenorCasadoFemenino" : mMC++; break;
51                 case "MenorViudoMasculino" : mHV++; break;
52                 case "MenorViudoFemenino" : mMV++; break;
53                 case "MenorDivorciadoMasculino" : mHD++; break;
54                 case "MenorDivorciadoFemenino" : mMD++; break;
55                 default: {} break;
56             }
57         }
58
59         public String CalculoSexo() {
60             Actualizar();
61             Hombre = (MHS + MHC + MHV + MHD + mHS + mHC + mHV + mHD);
62             Mujer = (MMS + MMC + MMV + MMD + mMS + mMC + mMV + mMD);
63             float PM, PH, TA = Lista.size();

```

Ejecución

Asistentes

Nombre:

Edad:

Sexo:

☐ Masculino ☐ Femenino

Estado Civil:

☐ Soltero

☐ Viudo

☐ Casado

☐ Divorciado

Nombre:LAURA Estado Civil:Casado Sexo:F Edad:21

Nombre:ANGEL Estado Civil:Divorciado Sexo:M Edad:40

Nombre:KAREN Estado Civil:Viudo Sexo:M Edad:35

Nombre:PAOLA Estado Civil:Divorciado Sexo:F Edad:34

Personas Mayores: 6 Personas Menores: 1

Solteros: 2 Casados: 2 Viudos: 1 Divorciados: 2

Hombres: 4 Porcentaje: 57.14% Mujeres: 3 Porcentaje: 42.86%

Invitados: 7

☒ Ver Estadísticas

```

64     PM = (Mujer / TA) * 100f;
65     PH = (Hombre / TA) * 100f;
66     Cadena = "Hombres: " + Hombre + " Porcentaje: " +
67         Math.round((PH * 100)) / 100f + "% ";
68     Cadena += "Mujeres: " + Mujer + " Porcentaje: " +
69         Math.round((PM * 100)) / 100f + "%";
70     return Cadena;
71 }
72
73 public String CalculoEstadoCivil() {
74     Actualizar();
75     Cadena = "Solteros: " + (MMS + MHS + mMS + mHS) + " ";
76     Cadena += "Casados: " + (MMC + MHC + mMC + mHC) + " ";
77     Cadena += "Viudos: " + (MMV + MHV + mMV + mHV) + " ";
78     Cadena += "Divorciados: " + (MMD + MHD + mMD + mHD);
79     return Cadena;
80 }
81
82 public String CalculaTotal() {
83     Actualizar();
84     Mayor = (MHS + MHC + MHV + MHD + mMS + mMC + mMV + mMD);
85     Menor = (mHS + mHC + mHV + mHD + mMS + mMC + mMV + mMD);
86     Cadena = "Personas Mayores: " + Mayor + "
87         + "Personas Menores: " + Menor + "\n";
88     return Cadena;
89 }
90
91 public static CategoryDataset creaDatosCategory() {
92
93     DefaultCategoryDataset datosEdad = new DefaultCategoryDataset();
94
95     datosEdad.setValue(value: MMS, rowKey: "Mujer Mayor", columnKey: "Soltero");
96     datosEdad.setValue(value: mMS, rowKey: "Mujer Menor", columnKey: "Soltero");
97     datosEdad.setValue(value: MHS, rowKey: "Hombre Mayor", columnKey: "Soltero");
98     datosEdad.setValue(value: mHS, rowKey: "Hombre Menor", columnKey: "Soltero");
99
100     datosEdad.setValue(value: MMC, rowKey: "Mujer Mayor", columnKey: "Casado");
101     datosEdad.setValue(value: mMC, rowKey: "Mujer Menor", columnKey: "Casado");
102     datosEdad.setValue(value: MHC, rowKey: "Hombre Mayor", columnKey: "Casado");
103     datosEdad.setValue(value: mHC, rowKey: "Hombre Menor", columnKey: "Casado");
104
105     datosEdad.setValue(value: MMV, rowKey: "Mujer Mayor", columnKey: "Viudo");
106     datosEdad.setValue(value: mMV, rowKey: "Mujer Menor", columnKey: "Viudo");
107     datosEdad.setValue(value: MHV, rowKey: "Hombre Mayor", columnKey: "Viudo");
108     datosEdad.setValue(value: mHV, rowKey: "Hombre Menor", columnKey: "Viudo");
109
110     datosEdad.setValue(value: MMD, rowKey: "Mujer Mayor", columnKey: "Divorciado");
111     datosEdad.setValue(value: mMD, rowKey: "Mujer Menor", columnKey: "Divorciado");
112     datosEdad.setValue(value: MHD, rowKey: "Hombre Mayor", columnKey: "Divorciado");
113     datosEdad.setValue(value: mHD, rowKey: "Hombre Menor", columnKey: "Divorciado");
114     return datosEdad;
115 }
116 }

```

Pasos para realizar la clase y su funcionamiento

Esta clase se dedica únicamente para crear el array de listaPersonas, y aquellos métodos que nos ayudaran a conocer la cantidad de personas.

1. Primero debemos crear las variables que se ocuparan para manejar el array.

```

// Variables para el array de personas
public class ListaPersonas {
    private static int MMS, MHS, MMC, MHC, MMV, MHV, MMD, MHD, mMS,
        mHS, mMC, mHC, mMV, mHV, mMD, mHD;
    private int Mayor, Menor, Hombre, Mujer;
    private ArrayList<Persona> Lista;
    private String Cadena;
}

```

Las variables importantes son "lista" que sera un ArrayList de la clase Persona, después para ayudarnos a manejar unos contadores creamos variables para cada posibilidad de persona, ya sea Hombre o Mujer, Mayor de edad o Menor de edad, si es Soltero, Casado, Viudo o Divorciado, por lo que en total habrá 16 variables, y añadimos una variable de nombre "cadena" que nos servirá al momento de crear cadenas para los métodos que ocuparemos

- Después se creará un constructor de clase el cual se iniciara cuando en la clase main se invoque el GUI que a su vez invocara un objeto de esta clase, por lo que el método es el siguiente:

```
public ListaPersonas() {  
    Lista = new <Persona>ArrayList();  
}
```

- Una vez teniendo el constructor, crearemos los métodos que nos permitan manipular el Array, (los métodos de Añadir, Eliminar, Limpiar).

```
public void Añadir(Persona P) {  
    Lista.add(e: P);  
}  
  
public void Eliminar(int pos) {  
    Lista.remove(index: pos);  
}  
  
public boolean EliminarTodo() {  
    return Lista.removeAll(e: Lista);  
}
```

- Ahora añadimos un método adicional que permita actualizar los contadores de las variables que al inicio creamos, este método realizara un ciclo for y hasta que todo el array haya pasado elemento por elemento, de esta manera con un switch case tendremos los contadores y así sabremos con exactitud la cantidad de personas por estado civil y por genero hay en nuestro array, el método es el siguiente:

```
public void Actualizar() {  
  
    MMS=0; MHS=0; MMC=0; MHC=0; MMV=0; MHV=0; MMD=0; MHD=0; mMS=0;  
    mHS=0; mMC=0; mHC=0; mMV=0; mHV=0; mMD=0; mHD=0;  
    for(Persona P:Lista) {  
        String cad=P.getEdad()+P.getEstadoCivil()+P.Sexo();  
        switch (cad) {  
            case "MayorSolteroMasculino" : MHS++; break;  
            case "MayorSolteroFemenino" : MMS++; break;  
            case "MayorCasadoMasculino" : MHC++; break;  
            case "MayorCasadoFemenino" : MMC++; break;  
            case "MayorViudoMasculino" : MHV++; break;  
            case "MayorViudoFemenino" : MMV++; break;  
            case "MayorDivorciadoMasculino": MHD++; break;  
            case "MayorDivorciadoFemenino": MMD++; break;  
            case "MenorSolteroMasculino": mHS++; break;  
            case "MenorSolteroFemenino": mMS++; break;  
            case "MenorCasadoMasculino": mHC++; break;  
            case "MenorCasadoFemenino": mMC++; break;  
            case "MenorViudoMasculino": mHV++; break;  
            case "MenorViudoFemenino": mMV++; break;  
            case "MenorDivorciadoMasculino": mHD++; break;  
            case "MenorDivorciadoFemenino": mMD++; break;  
            default: {} break;  
        }  
    }  
}
```


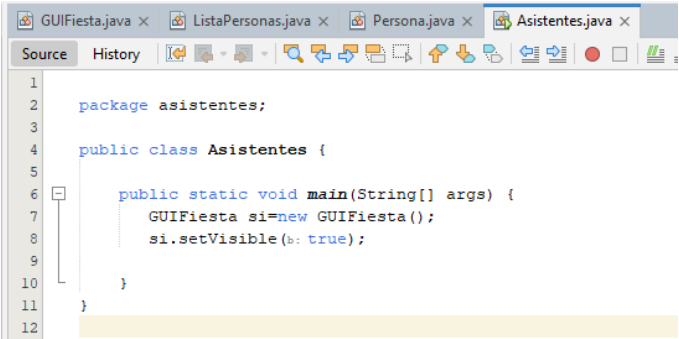

5. Otro método importante que añadimos es el método para crear los datos de la grafica, este método es de tipo CategoryDataset, donde con ayuda del ".setValue" añadiremos la cantidad de personas que son de la misma categoría(sabiendo que hay 16 categorías). El método es el siguiente:

```
public static CategoryDataset creaDatosCategory() {  
  
    DefaultCategoryDataset datosEdad = new DefaultCategoryDataset();  
  
    datosEdad.setValue(value: MMS, rowKey: "Mujer Mayor", columnKey: "Soltero");  
    datosEdad.setValue(value: mMS, rowKey: "Mujer Menor", columnKey: "Soltero");  
    datosEdad.setValue(value: MHS, rowKey: "Hombre Mayor", columnKey: "Soltero");  
    datosEdad.setValue(value: mHS, rowKey: "Hombre Menor", columnKey: "Soltero");  
  
    datosEdad.setValue(value: MMC, rowKey: "Mujer Mayor", columnKey: "Casado");  
    datosEdad.setValue(value: mMC, rowKey: "Mujer Menor", columnKey: "Casado");  
    datosEdad.setValue(value: MHC, rowKey: "Hombre Mayor", columnKey: "Casado");  
    datosEdad.setValue(value: mHC, rowKey: "Hombre Menor", columnKey: "Casado");  
  
    datosEdad.setValue(value: MMV, rowKey: "Mujer Mayor", columnKey: "Viudo");  
    datosEdad.setValue(value: mMV, rowKey: "Mujer Menor", columnKey: "Viudo");  
    datosEdad.setValue(value: MHV, rowKey: "Hombre Mayor", columnKey: "Viudo");  
    datosEdad.setValue(value: mHV, rowKey: "Hombre Menor", columnKey: "Viudo");  
  
    datosEdad.setValue(value: MMD, rowKey: "Mujer Mayor", columnKey: "Divorciado");  
    datosEdad.setValue(value: mMD, rowKey: "Mujer Menor", columnKey: "Divorciado");  
    datosEdad.setValue(value: MHD, rowKey: "Hombre Mayor", columnKey: "Divorciado");  
    datosEdad.setValue(value: mHD, rowKey: "Hombre Menor", columnKey: "Divorciado");  
    return datosEdad;  
}
```

Como podemos ver en el método de creaDatosCategory primero se instancia un objeto llamado datosEdad de tipo DefaultCategoryDataset, y después al objeto se le llamara al método ".setValue" y se añadirá la cantidad de personas que hay, y como vemos hay 16 categorías que se añadirán, por lo que habrá 16 entradas con diferentes datos.

NOTA: para ingresar el valor tuvimos que añadirle un "static" a las variables de las categorías

Clase	Ejecución
<div data-bbox="256 237 527 268"> Clase Persona:</div> <div data-bbox="154 283 880 1509"><pre>1 package asistentes; 2 3 public class Persona { 4 private String nombrePersona; 5 private byte edadPersona; 6 private char Sexo; 7 private String EstadoCivil; 8 //Constructores 9 public Persona() { 10 } 11 12 public Persona(String nombrePersona, byte edadPersona, 13 char Sexo, String EstadoCivil) { 14 this.nombrePersona = nombrePersona; 15 this.edadPersona = edadPersona; 16 this.Sexo = Sexo; 17 this.EstadoCivil = EstadoCivil; 18 } 19 20 public String getNombrePersona() { 21 return nombrePersona; 22 } 23 24 public void setNombrePersona(String nombrePersona) { 25 this.nombrePersona = nombrePersona; 26 } 27 28 public byte getEdadPersona() { 29 return edadPersona; 30 } 31 32 public void setEdadPersona(byte edadPersona) { 33 this.edadPersona = edadPersona; 34 } 35 36 public char getSexo() { 37 return Sexo; 38 } 39 40 public void setSexo(char Sexo) { 41 this.Sexo = Sexo; 42 } 43 44 public String getEstadoCivil() { 45 return EstadoCivil; 46 } 47 48 public void setEstadoCivil(String EstadoCivil) { 49 this.EstadoCivil = EstadoCivil; 50 } 51 52 public String getEdad() { 53 if(this.edadPersona>=18) 54 return "Mayor"; 55 else 56 return "Menor"; 57 } 58 59 public String Sexo() { 60 if(this.Sexo=='M') 61 return "Masculino"; 62 else 63 return "Femenino"; 64 } 65 }</pre></div>	
Funcionamiento de la clase	
<p>Esta clase está diseñada únicamente para los parametros de una persona, ya que la clase es la de “Persona” cuando se crea un objeto nuevo se instancia con los parametros que tiene el constructor de esta clase, de esta forma podemos almacenar los datos del objeto en el ArrayList de “listaPersonas”, por lo que únicamente esta clase contienen los métodos Getter, Setter y los constructores</p>	

Clase	Ejecución
<p> Clase Asistentes:</p>  <pre> 1 2 package asistentes; 3 4 public class Asistentes { 5 6 public static void main(String[] args) { 7 GUIFiesta si=new GUIFiesta(); 8 si.setVisible(true); 9 } 10 11 } 12 </pre>	
Funcionamiento	
<p>Esta clase se considera la clase main, ya que aquí es cuando se inicializa toda la interfaz, se crea un objeto de la clase GUIFiesta el cual generará toda la interfaz gracias al constructor que se ubica en dicha clase, y con el método setVisible(true) permitirá ver la interfaz.</p>	

7. Conclusiones

La utilización de interfaces de usuario es de gran importancia pues a partir de ella existe más facilidad en la relación de sistema-usuario. Java es una plataforma que brinda grandes posibilidades para el desarrollo de aplicaciones y el trabajo de interfaces gráficas.

Con esta practica se entendió cuales son los componentes necesarios para poder crear una interfaz desde cero a partir de la codificación.

8. Bibliografía

Bibliografía

García Rincón, J. (6 de Septiembre de 2022). *jairogarciarincon*. Obtenido de [www.jairogarciarincon.com](https://www.jairogarciarincon.com/clase/interfaces-de-usuario-con-java-swing/eventos-y-componente-jbutton): <https://www.jairogarciarincon.com/clase/interfaces-de-usuario-con-java-swing/eventos-y-componente-jbutton>

POL. (18 de Febrero de 2019). *PEREZ987*. Obtenido de [perez987.es](https://perez987.es/acciones-al-pulsar-un-boton-en-java/): <https://perez987.es/acciones-al-pulsar-un-boton-en-java/>

Blogspot. (Marzo de 2012). *Tópicos Avanzados de Programación - ITCA*. Obtenido de <http://progitca.blogspot.com/2012/03/35-creacion-y-uso-de-paqueteslibrerias.html>

Charte, F. (2019). *Paquetes en Java*. Obtenido de <https://www.campusmvp.es/recursos/post/paquetes-en-java-que-son-para-que-se-utilizan-y-como-se-usan.aspx>

González, J. D. (2022). *Importar librerías estándar de Java y librerías propias*. Obtenido de <https://www.programarya.com/Cursos/Java/Librerias>

González, J. D. (2022). *Uso de los paquetes en Java*. Obtenido de <https://www.programarya.com/Cursos/Java/Paquetes>

Yañez, C. (2018). *Desarrollo de interfaces: cómo crear componentes visuales*. Obtenido de <https://www.ceac.es/blog/desarrollo-de-interfaces-como-crear-componentes-visuales>