Laboratorio N° 8

Escuela Profesional: Ingeniería de Sistemas. **Asignatura:** Programación orientada a objetos
Docente: Ing. Loncán Salazar, Pierre Paul

Sesión 8: Arreglo de Objetos

I. OBJETIVOS

Al término de esta experiencia, el estudiante será capaz de:

1. Registrar y recuperar datos desde arreglos de objetos

II. EQUIPOS Y MATERIALES

- Computador
- Guía de Laboratorio
- Material impreso con la información de la sesión de aprendizaje.

III. METODOLOGIA Y ACTIVIDADES

a) Teoría de Arreglo de Objetos
b) Teoría de manipulación de archivos de texto

IV. IMPORTANTE

Antes de iniciar con el desarrollo del Laboratorio, crearemos siempre, una carpeta, donde se guardará toda la información del presente laboratorio. Para ello realice lo siguiente:

- Ingrese al Explorador del Windows (puede hacerlo dando clic derecho sobre el Botón Inicio de la Barra de Tareas y seleccione la opción Explorar).
- La ventana del Explorador esta dividida en dos columnas, en la columna de la izquierda busque hacia abajo la unidad de almacenamiento (D:) y de un clic izquierdo sobre él. Luego dirija el mouse hacia la columna de la derecha y en un sector vacío, presione clic derecho, seleccione la opción Nuevo y luego la opción Carpeta.
- Aparecerá una carpeta amarilla con un texto: Nueva Carpeta sombreado en azul, digite sobre él, el nombre para su carpeta (este puede ser L8_POO_(Turno Apellido)), luego de digitar presione la tecla Enter. Listo, ya tiene su carpeta dentro de la cual guardará todo lo que trabaje a continuación.
- Cierre la ventana del Explorador del Windows.

V. PROCEDIMIENTO

a) Encender el computador.
b) Crear carpeta donde guardará el documento son su información.
c) Ingresar al software Microsoft Word y allí crear los cuadros de doble entrada y los diagramas de clases y objetos solicitados. Word
d) Ingresar al software NetBeans IDE y allí crear:
    - Cada uno de los Proyectos solicitados. Nómbrelos como Proyecto_Cuenta, Proyecto_Operacion, etc. Según el nombre que se ha asignado en este laboratorio.
e) Presentar avances al docente para la calificación correspondiente.
f) Guardar la carpeta de sus archivos a sus memorias y enviar por correo una copia del archivo al docente del curso.
g) Retirarse del laboratorio de forma ordenada.

Ejercicio 1: (Arreglo de Objetos - Archivo de Texto)

Clase Cliente

```java
package Modelo;

public class Cliente
{
    private String Codigo;
    private String Apellidos;
    private String Nombres;

    public void setDatos(String[] Registro)
    {
        Codigo = Registro[0];
        Apellidos = Registro[1];
        Nombres = Registro[2];
    }

    public String[] getDatos()
    {
        String[] Registro = {Codigo,Apellidos,Nombres};
        return Registro;
    }

    public String getApellidos() {
        return Apellidos;
    }

    public void setApellidos(String Apellidos) {
        this.Apellidos = Apellidos;
    }

    public String getCodigo() {
        return Codigo;
    }

    public void setCodigo(String Codigo) {
        this.Codigo = Codigo;
    }

    public String getNombres() {
        return Nombres;
    }

    public void setNombres(String Nombres) {
        this.Nombres = Nombres;
    }
}
```

Clase ArregloCliente

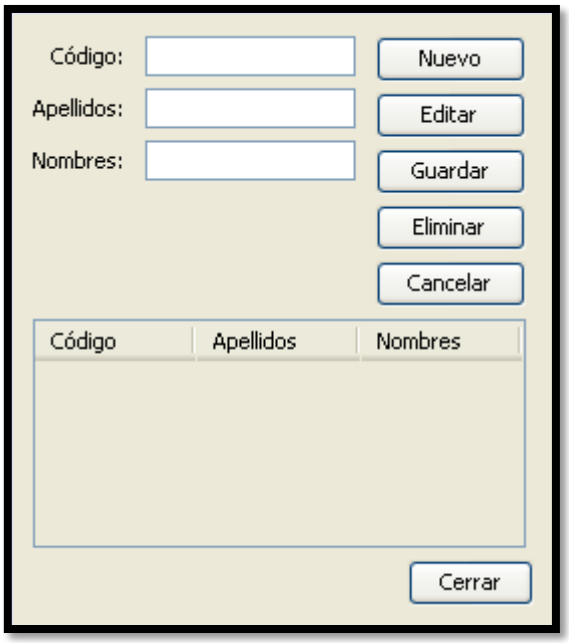
```
1   package Control;
2
3   import Modelo.Cliente;
4   import java.io.BufferedReader;
5   import java.io.FileReader;
6   import java.io.FileWriter;
7   import java.io.PrintWriter;
8   import java.util.StringTokenizer;
9   import javax.swing.table.DefaultTableModel;
10
11  public class ArregloCliente
12  {
13      //Variable tipo arreglo de objetos para instancias de la Clase "Cliente"
14      Cliente[] Arreglo = new Cliente[25];    //**** Codigo Cambiado ****//
15      //Variable que sirva como contador de Registros
16      int i=0;
17
18      //Agrega una Nueva Instancia de la Clase Cliente. El registro es colocado
19      //dentro de la instancia de la Clase Cliente
20      public void Registrar(String[] Registro)
21      {
22          //Evalua que aún exista espacio disponible en el arreglo
23          if(i<Arreglo.length)
24          {
25              //Crea la instancia de la clase Cliente
26              Cliente objCliente = new Cliente();    //**** Codigo agregado ****//
27              //Asigna los datos a la instancia
28              objCliente.setDatos(Registro);    //**** Codigo agregado ****//
29              //Agrega la instancia al arreglo
30              Arreglo[i] = objCliente;    //**** Codigo Cambiado ****//
31              //Incrementa el contador de registros
32              i++;
33          }
34      }
35
36      //Cambia o Actualiza los Datos de la Instancia de la Clase Cliente
37      //ubicada en la Posicion Indicada
38      public void Actualizar(int Pos, String[] Registro)
39      {
40          //Recupera la instacia de la posición indicada
41          Cliente objCliente = Arreglo[Pos];    //**** Codigo agregado ****//
42          //Asigna los nuevos datos a la instancia
43          objCliente.setDatos(Registro);    //**** Codigo Cambiado ****//
44      }
45
46      //Devuelve el Registro contenido dentro de la instancia de la Clase Cliente
47      //ubicada en la Posicion Indicada
48      public String[] getRegistro(int Pos)
49      {
50          //Recupera la instacia de la posición indicada
51          Cliente objCliente = Arreglo[Pos];    //**** Codigo agregado ****//
52          //Extrae y devuelve los datos contenidos en la instancia
53          return objCliente.getDatos();    //**** Codigo Cambiado ****//
54      }
55
```

```
56        //Elimina la instancia de la Clase Cliente ubicada en la Posicion Indicada
57        public void Eliminar_Registro(int Pos)
58        {
59            //Recorre todas las posiciones del arreglo que contengan datos
60            //pero iniciando desde la posición a eliminar
61            for(int x = Pos; x < i-1; x++)
62            {
63                //Copia los datos del arreglo de derecha a izquierda
64                Arreglo[x] = Arreglo[x+1];
65            }
66            //Elimina el último dato registrado
67            Arreglo[i] = null;
68            //Reduce el contador de registros
69            i--;
70        }
71
72        //Llena en la Tabla los Registros contenidos dentro de todas las instancias
73        //de la Clase Cliente guardadas en el arreglo
74        public void Llenar_Tabla(DefaultTableModel modTabla)
75        {
76            //Reinicia el contador de filas
77            modTabla.setRowCount(0);
78            //Recorre todas las posiciones del arreglo que contengan datos
79            for (int Pos = 0; Pos < i; Pos++)
80            {
81                //Recupera la instacia de la posición indicada
82                Cliente objCliente = Arreglo[Pos];    //**** Codigo agregado ****//
83                //Agrega al modelo el registro contenido en la instancia
84                modTabla.addRow(objCliente.getDatos());//**** Codigo Cambiado ****//
85            }
86        }
87
88        //Lee los Registros de todas las instancias de la Clase Cliente guardadas
89        //en el arreglo y los guarda dentro de un Archivo de Texto.
90        //Usa ";" como Token para separar los datos antes de escribir cada línea
91        public void GuardarTodo(String NomArchivo)
92        {
93            try
94            {
95                //Variable que abre el archivo en modo sobreescritura
96                FileWriter fw = new FileWriter(NomArchivo);
97                //Variable para escribir el el archivo
98                PrintWriter pw = new PrintWriter(fw);
99                //Evalua que exista la variable de escritura
100               if(pw != null)
101               {
102                   //Recorre todo el arreglo
103                   for(int Pos=0 ; Pos<i ; Pos++)
104                   {
105                       //Recupera la instacia de la posición indicada
106                       Cliente objCliente = Arreglo[Pos]; //****  agregado ****//
107                       //Extrae los datos de cada posición del Arreglo
108                       String Dato1 = objCliente.getCodigo();
109                       String Dato2 = objCliente.getApellidos();
110                       String Dato3 = objCliente.getNombres();
111                       //Construye la Linea usando los datos y el Token ";"
112                       String Linea = Dato1 + ";" + Dato2 + ";" + Dato3;
```

```
113                         //Imprine la linea en la variable de escritura
114                         pw.println(Linea);
115                     }
116                     //Cierra la variable de escritura y envia los datos al archivo
117                     pw.close();
118                 }
119             }
120             catch (Exception e)
121             {
122 
123             }
124         }
125 
126         public void GuardarUno(String NomArchivo, String[] Registro)
127         {
128             try
129             {
130                 //Variable que abre el archivo en modo añadidura
131                 FileWriter fw = new FileWriter(NomArchivo, true);
132                 //Variable para escribir el el archivo
133                 PrintWriter pw = new PrintWriter(fw);
134                 //Evalua que exista la variable de escritura
135                 if(pw != null)
136                 {
137                     //Extrae los datos del Registro
138                     String Dato1 = Registro[0];
139                     String Dato2 = Registro[1];
140                     String Dato3 = Registro[2];
141                     //Construye la Linea usando los datos y el Token ";"
142                     String Linea = Dato1 + ";" + Dato2 + ";" + Dato3;
143                     //Imprine la linea en la variable de escritura
144                     pw.println(Linea);
145                     //Cierra la variable de escritura y envia los datos al archivo
146                     pw.close();
147                 }
148             }
149             catch (Exception e)
150             {
151 
152             }
153         }
154 
155         //Lee todas las líneas del Archivo de Texto y las registra
156         //dentro del arreglo.
157         //Usa ";" como Token para separar los datos contenidos en cada línea
158         public void Abrir(String NomArchivo)
159         {
160             try
161             {
162                 //Variable que abre el archivo en modo lectura
163                 FileReader fr = new FileReader(NomArchivo);
164                 //Variable para leer el archivo
165                 BufferedReader br = new BufferedReader(fr);
166                 if(br != null)
167                 {
168                     String Linea = null;
```

```
169                     //Recorre todas las líneas del archivo y se detiene
170                     //cuando encuentre una línea nula
171                     while((Linea = br.readLine()) != null)
172                     {
173                         //Fragmenta la línea según el Token ";"
174                         StringTokenizer st = new StringTokenizer(Linea,";");
175                         //Extrae cada fragmento al pasar al siguiente Token
176                         String Dato1 = st.nextToken();
177                         String Dato2 = st.nextToken();
178                         String Dato3 = st.nextToken();
179                         //Crea un arreglo con los datos
180                         String[] Registro = {Dato1, Dato2, Dato3};
181                         //Agrega el Registro al Arreglo
182                         Registrar(Registro);
183                     }
184                     //Cierra la variable de lectura
185                     br.close();
186                 }
187             }
188             catch (Exception e)
189             {
190
191             }
192         }
193     }
```

Formulario



Clase JFrame_Principal

```
1   package Vista;
2
3   import Control.ArregloCliente;
4   import javax.swing.JOptionPane;
5   import javax.swing.table.DefaultTableModel;
6
7   public class JFrame_Principal extends javax.swing.JFrame
8   {
9       //Instancia de la Clase ArregloCliente
10      ArregloCliente objArrCliente = new ArregloCliente();
11      //Variable que almacena el Nombre del Archivo
12      String NombreArchivo = "Clientes.txt";
13      //Variable que controla el modelo de la Tabla
14      DefaultTableModel modTabla;
15      //Variable que almacena la operación ejecutada
16      String Operacion = "";
17      //Variable que almacena el número de registro seleccionado
18      int Pos = -1;
19
20      public JFrame_Principal()
21      {
22          initComponents();
23          modTabla = (DefaultTableModel) tbl_Datos.getModel();
            Estado_Botones(false);
            Estado_Controles(false);
26      }
27
28      //Habilita o Deshabilita los Botones
29      public void Estado_Botones(boolean Estado)
30      {
31          //Grupo 01
32          btn_Guardar.setEnabled(Estado);
33          btn_Cancelar.setEnabled(Estado);
34          //Grupo 02
35          btn_Nuevo.setEnabled(!Estado);
36          btn_Editar.setEnabled(!Estado);
37          btn_Eliminar.setEnabled(!Estado);
38          btn_Cerrar.setEnabled(!Estado);
39      }
40
41      //Habilita o Dshabilita los Controles
42      public void Estado_Controles(boolean Estado)
43      {
44          txt_Codigo.setEnabled(Estado);
45          txt_Apellidos.setEnabled(Estado);
46          txt_Nombres.setEnabled(Estado);
47          tbl_Datos.setEnabled(!Estado);
48      }
49
50      //Limpia los Controles
51      public void Limpiar_Controles()
52      {
53          txt_Codigo.setText("");
54          txt_Apellidos.setText("");
```

```
55          txt_Nombres.setText("");
56      }
57
58      @SuppressWarnings("unchecked")
59      Generated Code
227
228     private void formWindowOpened(java.awt.event.WindowEvent evt) {
229         objArrCliente.Abrir(NombreArchivo);
230         objArrCliente.Llenar_Tabla(modTabla);
231     }
232
233     private void formWindowClosing(java.awt.event.WindowEvent evt) {
234         objArrCliente.GuardarTodo(NombreArchivo);
235     }
236
237     private void btn_NuevoActionPerformed(java.awt.event.ActionEvent evt) {
238         //Habilita botones y controles. Limpia los controles
239         Estado_Botones(true);
240         Estado_Controles(true);
241         Limpiar_Controles();
242         //Establece el tipo de operación invocada
243         Operacion = "Nuevo";
244     }
245
246     //Activa el modo edición recuperando los datos registrados
247     private void btn_EditarActionPerformed(java.awt.event.ActionEvent evt) {
248         //Evalúa si hay un registro seleccionado
249         if(tbl_Datos.getSelectedRow() != -1)
250         {
251             //Habilita botones y controles. Limpia los controles
252             Estado_Botones(true);
253             Estado_Controles(true);
254             Limpiar_Controles();
255             //Establece el tipo de operación invocada
256             Operacion = "Editar";
257             //Captura la posición del registro seleccionado
258             Pos = tbl_Datos.getSelectedRow();
259             //Recupera desde la instancia "ObjArrCliente" el registro en la
260             //posición indicada
261             String[] Registro = objArrCliente.getRegistro(Pos);
262             //Carga los datos del registro en los controles
263             txt_Codigo.setText(Registro[0]);
264             txt_Apellidos.setText(Registro[1]);
265             txt_Nombres.setText(Registro[2]);
266         }
267         else
268         {
269             JOptionPane.showMessageDialog(this, "Debe seleccionar un Registro");
270         }
271     }
272
273     //Guarda los datos ingresados en los controles en la instancia de la
274     //clase "objArrCliente"
275     private void btn_GuardarActionPerformed(java.awt.event.ActionEvent evt) {
276         //Extrae los datos convirtiendolos en mayúscula
277         String Codigo = txt_Codigo.getText().toUpperCase();
278         String Apellidos = txt_Apellidos.getText().toUpperCase();
```

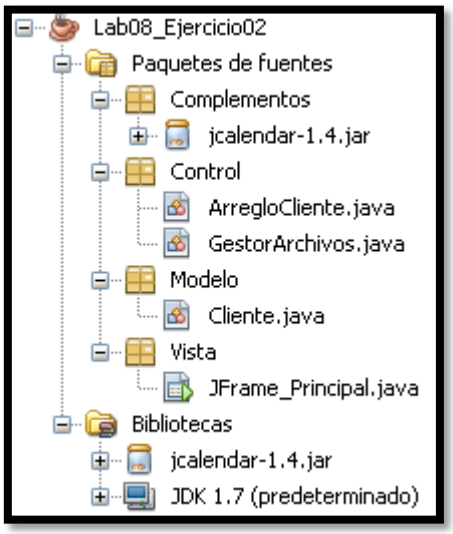
```
279             String Nombres = txt_Nombres.getText().toUpperCase();
280             //Crea un registro con los datos extraidos
281             String[] Registro = {Codigo,Apellidos,Nombres};
282             //Según la operación ejecutará Registro o Actualización
283             if(Operacion.equals("Nuevo"))
284             {
285                 objArrCliente.Registrar(Registro);
286             }
287             else if(Operacion.equals("Editar"))
288             {
289                 objArrCliente.Actualizar(Pos, Registro);
290             }
291             //Vuelve a llenar el contenido de la tabla
292             objArrCliente.Llenar_Tabla(modTabla);
293             //Deshabilita los botones y controles. Limpia los controles
294             Estado_Botones(false);
295             Estado_Controles(false);
296             Limpiar_Controles();
297         }
298
299         //Elimina el registro seleccionado en la Tabla.
300         //Antes de eliminar evalúa si hay un registro seleccionado
301         private void btn_EliminarActionPerformed(java.awt.event.ActionEvent evt) {
302             Pos = tbl_Datos.getSelectedRow();
303             if(Pos != -1) //El valor -1 indica que no hay registro seleccionado
304             {
305                 //Elimina el registro y vuelve a llenar la tabla
306                 objArrCliente.Eliminar_Registro(Pos);
307                 objArrCliente.Llenar_Tabla(modTabla);
308             }
309             else
310             {
311                 JOptionPane.showMessageDialog(this, "Debe seleccionar un Registro");
312             }
313         }
314
315         //Cancela la Operaciones "Nuevo" o "Editar" según la que se haya invocado
316         private void btn_CancelarActionPerformed(java.awt.event.ActionEvent evt) {
317             Estado_Botones(false);
318             Estado_Controles(false);
319             Limpiar_Controles();
320             Operacion = "";
321         }
322
323         //Cierra la aplicación
324         private void btn_CerrarActionPerformed(java.awt.event.ActionEvent evt) {
325             this.formWindowClosing(null);
326             System.exit(0);
327         }
328
```

Ejercicio 2: (Arreglo de Objetos - Archivo de Texto)

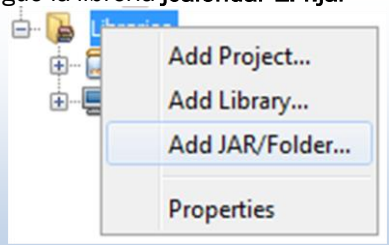
Diseñe un formulario que permita:

- Ingresar los Datos de un Cliente (Código, Apellidos, Nombres y Fecha de Nacimiento).
- Guardar y Editar todos los datos anteriores dentro de un arreglo de objetos
- Almacenar de forma permanente toda la información dentro de un archivo de texto

1. Agregue a la paleta de herramientas de NetBeans el control **JDateCooser.**
2. Cree un proyecto **Java Application** con el nombre **Sesion09_Ejercicio01**
3. Cree la siguiente estructura del proyecto.



- Agregue la librería **jcalendar-1.4.jar**

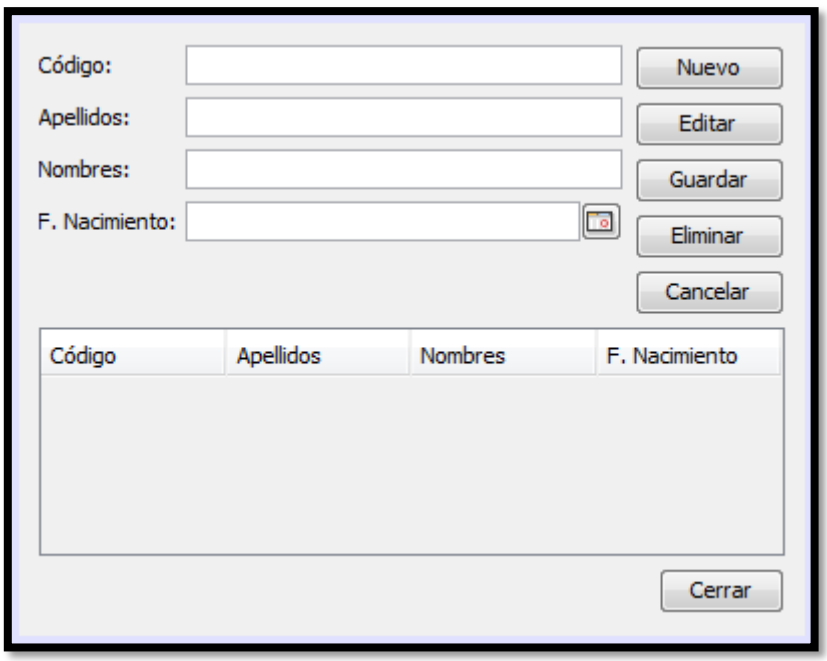


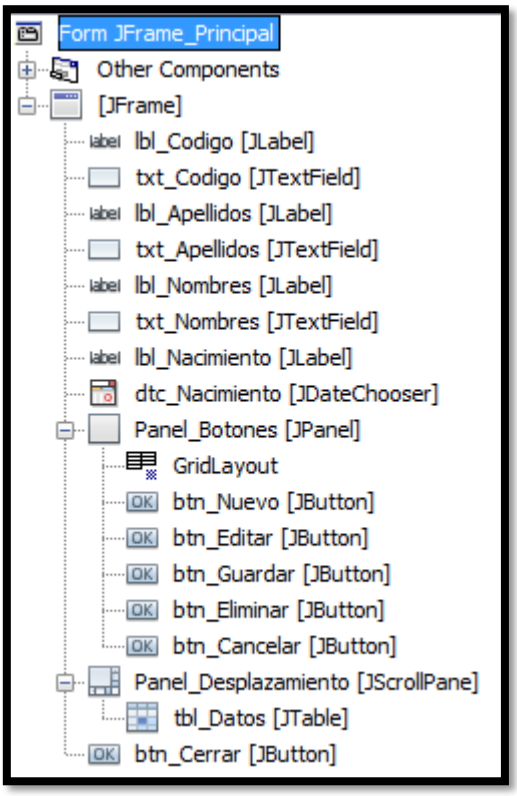
- Busque el archivo **jcalendar-1.4.jar** dentro de la carpeta **complementos** de tu proyecto

4. En el formulario **JFrame_Principal**
   a. Cambie las propiedades del **JFrame_Principal**

Control	Propiedad
JFrame	**DefaultCloseOperation**: DO_NOTHING **title**: Control de Clientes **resizable**: False

5. Cree el diseño del formulario **JFrame_Principal**, considerando la relación de controles aquí indicados





6. Cambie la propiedad de los controles del **JFrame_Pago**

Control	Propiedad
lbl_Codigo	**displayedMnemonic**: D **labelFor**: txt_Codigo **text**: Código
lbl_Apellidos	**displayedMnemonic**: A **labelFor**: txt_Apellidos **text**: Apellidos
lbl_Nombres	**displayedMnemonic**: M **labelFor**: txt_Nombres **text**: Nombres
lbl_Nacimiento	**displayedMnemonic**: F **labelFor**:dtc_Nacimiento **text**: F. Nacimiento
Txt_Codigo	**text**: “”
Txt_Apellidos	
Txt_Nombres	
Btn_Nuevo	**text:** Nuevo **mnemonic:** N
Btn_Editar	**text:** Editar **mnemonic:** E
Btn_Guardar	**text:** Guardar **mnemonic:** C
Btn_Eliminar	**text:** Eliminar **mnemonic:** G
Btn_Cancelar	**text:** Cancelar **mnemonic:** G
Btn_Cerrar	**text:** Cerrar **mnemonic**: R
Panel_Botones	**Layout:** GridLayout (Diseño de Rejilla) – (Columnas: 0 – Filas:1 –Separación Horizontal: 5)

7. En la clase **Cliente**
   a. Defina los atributos de la clase

```java
package Modelo;

public class Cliente
{
    private String Codigo;
    private String Apellidos;
    private String Nombres;
    private String Nacimiento;
```

   b. Implemente un constructor que inicialice todos los atributos de clase.

```java
    //Constructor de la Clase
    public Cliente(String[] Registro)
    {
        setRegistro(Registro);
    }
```

   c. Implemente los métodos para almacenar y extraer registros.

```java
    //Extrae los datos del Registro y los almacena en los atributos de la clase
    public void setRegistro(String[] Registro)
    {
        Codigo = Registro[0];
        Apellidos = Registro[1];
        Nombres = Registro[2];
        Nacimiento = Registro[3];
    }

    //Crea un Registro y lo devuelve
    public String[] getRegistro()
    {
        String[] Registro = {Codigo,Apellidos,Nombres,Nacimiento};
        return Registro;
    }
```

   d. Implemente métodos Getter y Setter para cada uno de los atributos de la clase.

```java
    //*** Metodos Getter y Setter ***//
    public String getApellidos() {
        return Apellidos;
    }

    public void setApellidos(String Apellidos) {
        this.Apellidos = Apellidos;
    }

    public String getCodigo() {
        return Codigo;
    }

    public void setCodigo(String Codigo) {
        this.Codigo = Codigo;
    }
```

```
49      public String getNombres() {
50          return Nombres;
51      }
52
53      public void setNombres(String Nombres) {
54          this.Nombres = Nombres;
55      }
56
57      public String getNacimiento() {
58          return Nacimiento;
59      }
60
61      public void setNacimiento(String Nacimiento) {
62          this.Nacimiento = Nacimiento;
63      }
64  }
```

8. En la clase **ArregloCliente**
   a. Defina el atributo de la clase

```
1   package Control;
2
3   import Modelo.Cliente;
4   import java.util.ArrayList;
5   import javax.swing.table.DefaultTableModel;
6
7   public class ArregloCliente
8   {
9       //Variable tipo ArrayList para instancias de la Clase "Cliente"
10      ArrayList<Cliente> Arreglo = new ArrayList<Cliente>();
11
```

   b. Implemente los métodos para el control del arreglo

```
12      //Agrega una nueva instancia de la Clase Cliente dentro del arreglo
13      public void Registrar(String[] Registro)
14      {
15          Arreglo.add(new Cliente(Registro));
16      }
17
18      //Coloca una nueva instancia de la Clase Cliente en la posición indicada
19      //reemplazando al antiguo cliente
20      public void Actualizar(int Pos, String[] Registro)
21      {
22          Arreglo.set(Pos, new Cliente(Registro));
23      }
24
25      //Devuelve el Registro contenido dentro de la instancia de la Clase Cliente
26      //ubicada en la Posicion Indicada
27      public String[] getElementoRegistro(int Pos)
28      {
29          return Arreglo.get(Pos).getRegistro();
30      }
31
```

```
32        //Elimina la instancia de la Clase Cliente ubicada en la Posicion Indicada
33        public void Eliminar_Elemento(int Pos)
34        {
35            Arreglo.remove(Pos);
36        }
37        //Llena en la Tabla los Registros contenidos dentro de todas las instancias
38        //de la Clase Cliente guardadas en el arreglo
39        public void Llenar_Tabla(DefaultTableModel modTabla)
40        {
41            //Reinicia el contador de filas
42            modTabla.setRowCount(0);
43            //Recorre todas las posiciones del arreglo que contengan datos
44            for (int Pos = 0; Pos < Arreglo.size(); Pos++)
45            {
46                //Agrega al modelo el registro de cada cliente
47                modTabla.addRow(Arreglo.get(Pos).getRegistro());
48            }
49        }
50        //Devolver la cantidad de Elementos registrados
51        public int getNumElementos()
52        {
53            return Arreglo.size();
54        }
55    }
```

9. En la clase **GestorArchivos**
   a. Importe las clases y librerías

```
1   package Control;
2
3   import java.io.BufferedReader;
4   import java.io.FileReader;
5   import java.io.FileWriter;
6   import java.io.PrintWriter;
7   import java.util.StringTokenizer;
```

   b. Implemente el método **GuardarTodo**

```
9   public class GestorArchivos
10  {
11      //Lee los Registros de todas las instancias de la Clase Cliente guardadas
12      //en el arreglo y los guarda dentro de un Archivo de Texto.
13      //Usa ";" como Token para separar los datos antes de escribir cada línea
14      public void GuardarTodo(ArregloCliente objArrCliente, String NomArchivo)
15      {
16          try
17          {
18              //Variable que abre el archivo en modo sobreescritura
19              FileWriter fw = new FileWriter(NomArchivo);
20              //Variable para escribir el el archivo
21              PrintWriter pw = new PrintWriter(fw);
22              //Evalua que exista la variable de escritura
23              if(pw != null)
24              {
```

```
25                  //Recorre todo el arreglo
26                  for(int Pos=0 ; Pos<objArrCliente.getNumElementos(); Pos++)
27                  {
28                     //Recupera la instacia de la posición indicada
29                     String[] Registro = objArrCliente.getElementoRegistro(Pos);
30                     //Extrae los datos de cada posición del Arreglo
31                     String Dato1 = Registro[0];
32                     String Dato2 = Registro[1];
33                     String Dato3 = Registro[2];
34                     String Dato4 = Registro[3];
35                     //Construye la Linea usando los datos y el Token ";"
36                     String Linea = Dato1 + ";" + Dato2 + ";" + Dato3 + ";" + Dato4;
37                     //Imprine la linea en la variable de escritura
38                     pw.println(Linea);
39                  }
40                  //Cierra la variable de escritura y envia los datos al archivo
41                  pw.close();
42               }
43            }
44            catch (Exception e)
45            {
46
47            }
48      }
```

c. Implemente el método **GuardarUno**

```
50      public void GuardarUno(String NomArchivo, String[] Registro)
51      {
52          try
53          {
54              //Variable que abre el archivo en modo añadidura
55              FileWriter fw = new FileWriter(NomArchivo, true);
56              //Variable para escribir el el archivo
57              PrintWriter pw = new PrintWriter(fw);
58              //Evalua que exista la variable de escritura
59              if(pw != null)
60              {
61                  //Extrae los datos del Registro
62                  String Dato1 = Registro[0];
63                  String Dato2 = Registro[1];
64                  String Dato3 = Registro[2];
65                  String Dato4 = Registro[3];
66                  //Construye la Linea usando los datos y el Token ";"
67                  String Linea = Dato1 + ";" + Dato2 + ";" + Dato3 + ";" + Dato4;
68                  //Imprine la linea en la variable de escritura
69                  pw.println(Linea);
70                  //Cierra la variable de escritura y envia los datos al archivo
71                  pw.close();
72              }
73          }
74          catch (Exception e)
75          {
76          }
77      }
```

d. Implemente el método **Abrir**

```
79      //Lee todas las líneas del Archivo de Texto y las registra
80      //dentro del arreglo a traves de la instancia objArrCliente.
81      //Usa ";" como Token para separar los datos contenidos en cada línea
82      public void Abrir(ArregloCliente objArrCliente, String NomArchivo)
83      {
84          try
85          {
86              //Variable que abre el archivo en modo lectura
87              FileReader fr = new FileReader(NomArchivo);
88              //Variable para leer el archivo
89              BufferedReader br = new BufferedReader(fr);
90              if(br != null)
91              {
92                  String Linea = null;
93                  //Recorre todas las líneas del archivo y se detiene
94                  //cuando encuentre una línea nula
95                  while((Linea = br.readLine()) != null)
96                  {
97                      //Fragmenta la línea según el Token ";"
98                      StringTokenizer st = new StringTokenizer(Linea,";");
99                      //Extrae cada fragmento al pasar al siguiente Token
100                     String Dato1 = st.nextToken();
101                     String Dato2 = st.nextToken();
102                     String Dato3 = st.nextToken();
103                     String Dato4 = st.nextToken();
104                     //Crea un arreglo con los datos
105                     String[] Registro = {Dato1, Dato2, Dato3, Dato4};
106                     //Agrega el Registro al Arreglo
107                     objArrCliente.Registrar(Registro);
108                 }
109                 //Cierra la variable de lectura
110                 br.close();
111             }
112         }
113         catch (Exception e)
114         {
115             System.out.println(e.getMessage());
116         }
117     }
118 }
```

10. En la clase **JFrame_Principal**

a. Importe las clases y librerías

```
1   package Vista;
2
3   import Control.ArregloCliente;
4   import Control.GestorArchivos;
5   import java.text.ParseException;
6   import java.text.SimpleDateFormat;
7   import java.util.Date;
8   import javax.swing.JOptionPane;
9   import javax.swing.table.DefaultTableModel;
```

b. Declare los atributos de la clase

```
11  public class JFrame_Principal extends javax.swing.JFrame
12  {
13      //Instancia de la Clase ArregloCliente
14      ArregloCliente objArrCliente = new ArregloCliente();
15      //Variable que controla el modelo de la Tabla
16      DefaultTableModel modTabla;
17      //Variable que almacena la operación ejecutada
18      String Operacion = "";
19      //Variable que almacena el número de registro seleccionado
20      int Pos = -1;
21      //Variable para dar formato a los datos tipo fecha
22      SimpleDateFormat SDF = new SimpleDateFormat("dd/MM/yyyy");
23      //Instancia de la la clase Gestor de Archivos
24      GestorArchivos Gestor = new GestorArchivos();
25      //Variable que almacena el Nombre del Archivo
26      String NombreArchivo = "Clientes.txt";
```

c. Edite el constructor de la clase

```
23      public JFrame_Principal()
24      {
25          initComponents();
26          //Obtener y asignar el modelo de la Lista
27          modTabla = (DefaultTableModel) tbl_Datos.getModel();
28          //Deshabilita botones y controles.
29          Estado_Botones(false);  Estado_Controles(false);
30          //Centrar Formulario
31          setLocationRelativeTo(null);
32      }
```

d. Implemente un método llamado **Estado_Botones** que reciba un valor tipo **boolean** (Estado).

```
33      //Habilita o Deshabilita los Botones
34      private void Estado_Botones(boolean Estado)
35      {
36          //Grupo 01
37          btn_Guardar.setEnabled(Estado);
38          btn_Cancelar.setEnabled(Estado);
39          //Grupo 02
40          btn_Nuevo.setEnabled(!Estado);
41          btn_Editar.setEnabled(!Estado);
42          btn_Eliminar.setEnabled(!Estado);
43          btn_Cerrar.setEnabled(!Estado);
44      }
```

e. Implemente un método llamado **Estado_Controles** que reciba un valor tipo **boolean** (Estado).

```
46      //Habilita o Deshabilita los Controles
47      private void Estado_Controles(boolean Estado)
48      {
49          txt_Codigo.setEnabled(Estado);
50          txt_Apellidos.setEnabled(Estado);
51          txt_Nombres.setEnabled(Estado);
52          dtc_Nacimiento.setEnabled(Estado);
53          tbl_Datos.setEnabled(!Estado);
54      }
```

f. Implemente un método llamado **Limpiar_Controles** que no reciba ningún valor.

```
56      //Limpia los Controles
57      public void Limpiar_Controles()
58      {
59          txt_Codigo.setText("");
60          txt_Apellidos.setText("");
61          txt_Nombres.setText("");
62          dtc_Nacimiento.setDate(new Date());
63      }
```

g. Implemente un evento tipo **ActionPerformed** para el control **btn_Nuevo**.

```
243     private void btn_NuevoActionPerformed(java.awt.event.ActionEvent evt) {
244         //Habilita botones y controles. Limpia los controles
245         Estado_Botones(true);
246         Estado_Controles(true);
247         Limpiar_Controles();
248         //Establece el tipo de operación invocada
249         Operacion = "Nuevo";
250     }
```

h. Implemente un evento tipo **ActionPerformed** para el control **btn_Editar**.

```
252     //Activa el modo edición recuperando los datos registrados
253     private void btn_EditarActionPerformed(java.awt.event.ActionEvent evt) {
254         //Evalúa si hay un registro seleccionado
255         if(tbl_Datos.getSelectedRow() != -1)
256         {
257             //Habilita botones y controles. Limpia los controles
258             Estado_Botones(true);
259             Estado_Controles(true);
260             Limpiar_Controles();
261             //Establece el tipo de operación invocada
262             Operacion = "Editar";
263             //Captura la posición del registro seleccionado
264             Pos = tbl_Datos.getSelectedRow();
265             //Recupera desde la instancia "ObjArrCliente" el registro en la
266             //posición indicada
267             String[] Registro = objArrCliente.getElementoRegistro(Pos);
268             //Carga los datos del registro en los controles
269             txt_Codigo.setText(Registro[0]);
270             txt_Apellidos.setText(Registro[1]);
271             txt_Nombres.setText(Registro[2]);
272             try
273             {
274                 //Crea una fecha a partir de una cadena
275                 dtc_Nacimiento.setDate(SDF.parse(Registro[3]));
276             }
277             catch (ParseException ex)
278             {
279                 System.out.println(ex.getMessage());
280             }
281         }
282         else
283         {
284             JOptionPane.showMessageDialog(this, "Debe seleccionar un Registro");
285         }
286     }
```

i. Implemente un evento tipo **ActionPerformed** para el control **btn_Guardar**.

```
288         //Guarda los datos ingresados en los controles en la instancia de la
289         //clase "objArrCliente"
290         private void btn_GuardarActionPerformed(java.awt.event.ActionEvent evt) {
291             //Extrae los datos convirtiendolos en mayúscula
292             String Codigo = txt_Codigo.getText();
293             String Apellidos = txt_Apellidos.getText().toUpperCase();
294             String Nombres = txt_Nombres.getText().toUpperCase();
295             String Nacimiento = SDF.format(dtc_Nacimiento.getDate());
296             //Crea un registro con los datos extraidos
297             String[] Registro = {Codigo,Apellidos,Nombres,Nacimiento};
298             //Según la operación ejecutará Registro o Actualización
299             if(Operacion.equals("Nuevo"))
300             {
301                 objArrCliente.Registrar(Registro);
302             }
303             else if(Operacion.equals("Editar"))
304             {
305                 objArrCliente.Actualizar(Pos, Registro);
306             }
307             //Vuelve a llenar el contenido de la tabla
308             objArrCliente.Llenar_Tabla(modTabla);
309             //Deshabilita los botones y controles. Limpia los controles
310             Estado_Botones(false);
311             Estado_Controles(false);
312             Limpiar_Controles();
313         }
```

j. Implemente un evento tipo **ActionPerformed** para el control **btn_Eliminar**.

```
315         //Elimina el registro seleccionado en la Tabla.
316         //Antes de eliminar evalúa si hay un registro seleccionado
317         private void btn_EliminarActionPerformed(java.awt.event.ActionEvent evt) {
318             Pos = tbl_Datos.getSelectedRow();
319             if(Pos != -1) //El valor -1 indica que no hay registro seleccionado
320             {
321                 //Evalua si desea eliminar
322                 int Rpta = JOptionPane.showConfirmDialog(this, "Desea Eliminar?",
323                                         "Eliminación", JOptionPane.YES_NO_OPTION);
324                 if(Rpta == JOptionPane.YES_OPTION)
325                 {
326                     //Elimina el registro y vuelve a llenar la tabla
327                     objArrCliente.Eliminar_Elemento(Pos);
328                     objArrCliente.Llenar_Tabla(modTabla);
329                 }
330             }
331             else
332             {
333                 JOptionPane.showMessageDialog(this, "Debe seleccionar un Registro");
334             }
335         }
```

k. Implemente un evento tipo **ActionPerformed** para el control **btn_Cancelar**.

```
337         //Cancela la Operaciones "Nuevo" o "Editar" según la que se haya invocado
338         private void btn_CancelarActionPerformed(java.awt.event.ActionEvent evt) {
339             //Deshabilita botones y controles. Limpia los controles
340             Estado_Botones(false);
341             Estado_Controles(false);
342             Limpiar_Controles();
343             Operacion = "";
344         }
```

l. Implemente un evento tipo **ActionPerformed** para el control **btn_Cerrar.**

```
343         //Cierra la aplicación
344         private void btn_CerrarActionPerformed(java.awt.event.ActionEvent evt) {
345             //LLama al evento WindowClosing antes de cerrar
346             this.formWindowClosing(null);
347             System.exit(0);
348         }
```

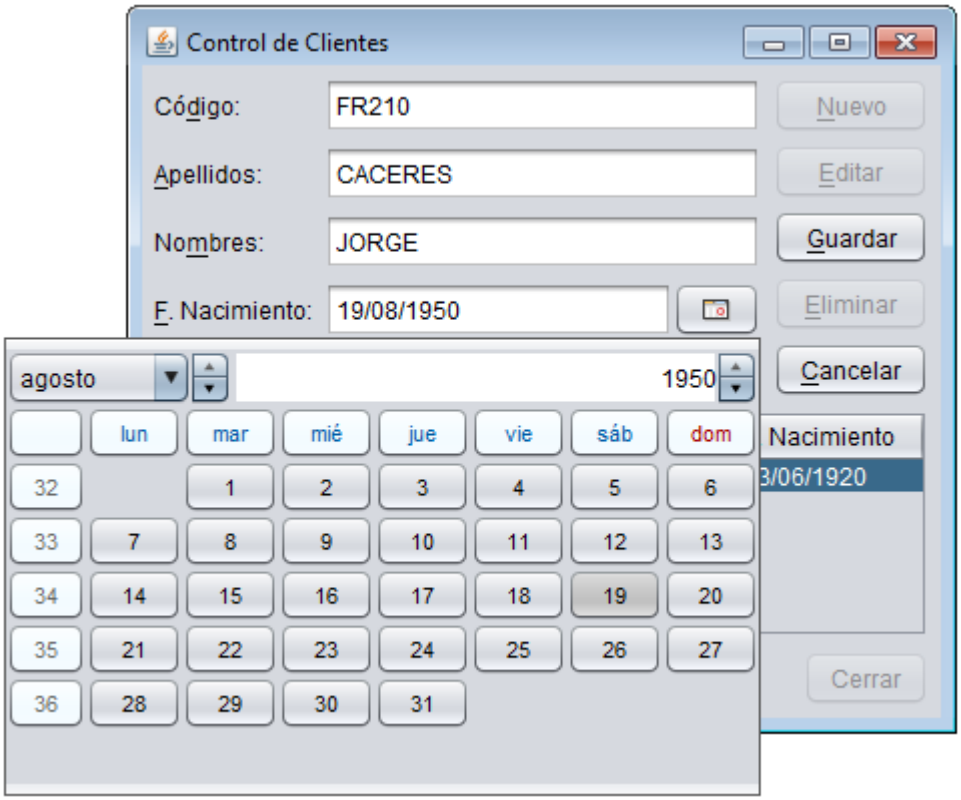
m. Implemente un evento tipo **WindowOpened** para el **JFrame_Principal.**

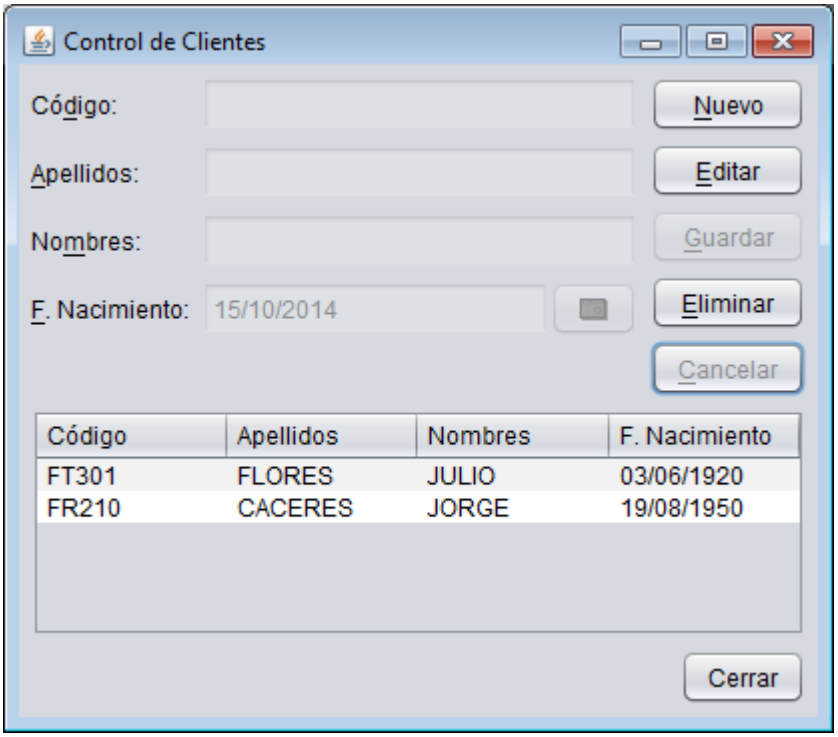
```
350         private void formWindowOpened(java.awt.event.WindowEvent evt) {
351             //Abre el archivo y almacena los datos en el arreglo
352             Gestor.Abrir(objArrCliente, NombreArchivo);
353             //Presenta los datos en la tabla
354             objArrCliente.Llenar_Tabla(modTabla);
355         }
```

n. Implemente un evento tipo **WindowClosing** para el **JFrame_Principal.**

```
357         private void formWindowClosing(java.awt.event.WindowEvent evt) {
358             //Abre el Archivo y guarda todos los datos dentro de ese archivo
359             Gestor.GuardarTodo(objArrCliente, NombreArchivo);
360         }
```

11. Ejecute la aplicación.





Ejercicio Propuesto:

Cree una aplicación que permita registrar (dentro de un arreglo de objetos) la venta de Pasajes en una agencia de transporte. Los datos a registrar son:

- Nombre del Pasajero
- Fecha de Venta
- Fecha de Viaje
- Destino
- Número de Asiento
- Precio del Pasaje

Operaciones:

- Registrar hasta un límite de 50 pasajes
- Mostrar todos los boletos vendidos de 3 formas:
  - Mostrar todos los boletos vendidos
  - Mostrar los boletos que tengan el mismo destino
  - Mostrar los boletos en donde el nombre del pasajero coincida con el nombre indicado
- Abrir y Guardar desde un Archivo de Texto llamado **Boletos.txt**

Opcional:

- No es posible vender 2 veces el mismo número de asiento

RUBRICA:

Inicio 0-10	Proceso 11-13	Logro previsto 14-17	Logro satisfactorio 18-20
Desarrollo correctamente del laboratorio hasta un 50 %	Desarrollo correctamente del laboratorio hasta un 60 %	Desarrollo correctamente del laboratorio hasta un 80 %	Desarrollo correctamente del laboratorio hasta un 100%

Bibliografía:

- THOMAS WU C. Introducción a la programación orientada a objetos con Java. 1ª Edición. España. McGraw-Hill Interamericana de España. 2008. 15-22pp. ISBN: 978-0-07-352339-2
- LEOBARDO LOPEZ. Román. Metodología de la programación orientada a objetos. 1ª Edición. México. Alafomega grupo editor de México. 2006. 241-253pp ISBN: 970-15-1173-5
- HERBERT SHILDT. JAVA 2 v5.0. España. Ediciones Anaya multimedia.2005. 79-99pp ISBN: 84-415-1865-3