

UNIVERSIDAD DE CORDOBA

FACULTAD DE INGENIERIAS

PROGRAMA INGENIERIA DE SISTEMAS

CURSO: Programación I

TEMA: Archivos de texto en Java.

DESCRIPCION: A lo largo de este documento se realiza un estudio del concepto y características de los archivos de texto y la respectiva implementación de un ejemplo práctico en lenguaje *Java* mediante un proyecto de interfaz gráfica de usuario (GUI) construido con *NetBeans* usando el paquete *Swing*, describiendo además los paquetes, clases y métodos de estas disponibles en *Java* para el tratamiento de archivos texto. Para este efecto se diseña e implementa una primera clase abstracta especializada en las operaciones de creación y apertura de un archivo de texto en modo de escritura, de tal suerte que además permita almacenar en disco (serializar) cualquier tipo de objeto mediante un método abstracto. En ese mismo orden de ideas, se diseña e implementa una segunda clase para apertura y recorrido de un archivo de texto en modo de lectura, que además tiene un método abstracto para recuperar los objetos serializados en el archivo. El proyecto de ejemplo ilustra el uso de menús en *Java Swing* y cuadros de diálogos para la apertura y guardado de archivos, adicional al uso de filtros en estos para tipos de archivos.

OBJETIVO: Diseñar e implementar en el lenguaje *Java* las clases necesarias para realizar las operaciones de creación, apertura, escritura y lectura de archivos de texto, incluyendo la serialización y recuperación de cualquier tipo de objetos, desarrollando para ello una aplicación de ventana en *Java* con *NetBeans* que muestre la serialización de un tipo de objetos y su recuperación desde un archivo de texto.

PALABRAS CLAVES: Archivos de texto en *Java*, serialización de objetos, la clase *File*, *BufferedWriter* y *BufferedReader* de *Java*, controles *JTable* y *JMenuBar*.

1. Archivos de Texto

En cuanto a los archivos de texto, estos constituyen la forma más simple de guardar y leer datos hasta o desde un archivo físico en disco. Un archivo de texto, como su nombre lo indica, esta compuesto por un conjunto de líneas de caracteres, que eventualmente tiene una longitud variable, y que se almacenan en el archivo usando algún formato especial de codificación, entre los que se destacan el ASCII, UNICODE, UTF entre otros. Estos valores se almacenan sin ningún atributo de formato especial, tales como tamaño de letra, color y características similares; por esta razón, a los archivos de texto también se les llaman archivos planos. Por estas razones el contenido de un archivo de texto es legible, lo cual supone que podemos ver y comprender su contenido directamente, al menos desde cualquier editor de texto; incluso con independencia del sistema operativo, lenguaje de programación o software con el que se ha creado dicho archivo.

En términos de las operaciones de lectura y escritura en archivos de texto, estas se hacen línea a línea; es decir, que cada vez que se guarda se almacena una línea de caracteres completa; igualmente, cuando se lee, se lee una línea de caracteres completa; en ambos casos, las dos operaciones comprenden líneas de texto de longitud o cantidad de caracteres variables. Siempre que se guarda una línea o se lee, el puntero de archivo avanza a la siguiente línea; por lo cual, un archivo de texto en función de su modo de acceso es también un archivo secuencial. Así por ejemplo, si queremos acceder a la décima línea de un archivo, previamente debemos leer las nueve que le preceden. Para el caso de java, las operaciones de lectura y de escritura en archivos de texto son mutuamente excluyentes; es decir, un archivo de texto se abre en un solo modo; sea para leer o para escribir y no en ambos; no obstante se puede abrir en un modo, realizar las operaciones requeridas y luego cerrarse para abrirse en el otro modo.

2. Archivos de Texto en Java

En el lenguaje *Java*, para el tratamiento de archivos (de cualquier tipo) y de directorio se utilizan las clases del paquete *Java.IO*, que además contiene excepciones, enumeraciones y demás elementos especializados en operaciones de entrada y salida con archivos y directorios. Una de las clases más representativas de este paquete es la clase ***File***, la cual contiene métodos para gestión tanto de archivos como de directorios, entre los que se destacan los siguientes descritos enseguida:

- ◆ *File(String Nombre)*: Crea una nueva instancia del archivo (o directorio) cuya ruta y nombre se pasa por parámetro.
- ◆ *boolean canExecute()* : Comprueba si la aplicación puede ejecutar el archivo denotado por esta ruta abstracta.
- ◆ *boolean canRead()*: Indica si el archivo se puede leer.
- ◆ *boolean canWrite()*: Indica si el archivo se puede escribir.
- ◆ *String getName()*: Retorna el nombre del archivo sin incluir su directorio.
- ◆ *String getPath()*: Retorna la ruta relativa del archivo o directorio.
- ◆ *String getAbsolutePath()*: devuelve la ruta absoluta del archivo o directorio.
- ◆ *long length()*: Retorna el tamaño en bytes del archivo.
- ◆ *boolean isDirectory()*: Retorna true si la instancia de la clase es un directorio.
- ◆ *boolean isFile()*: Retoma true si la instancia de la clase es un archivo.
- ◆ *boolean exists()*: Retorna true si el archivo o directorio existe.
- ◆ *boolean delete()*: Borra el archivo o directorio, retornando true si fue posible borrarlo; en caso contrario retoma false.
- ◆ *boolean mkdir()*: Para el caso de directorios, este método crea un nuevo directorio y retoma true; si no puede crearlo retoma false.
- ◆ *boolean renameTo(File Nuevo)*: Renombra el archivo o el directorio con base en la instancia pasada por parámetro.
- ◆ *String[] list()*: Para el caso de un directorio, retorna un arreglo de cadenas que contiene los nombres de archivos y subdirectorios del directorio en cuestión.

En cuanto al acceso a un archivo de texto en modo de escritura, usamos una instancia de la clase **BufferedWriter**, la cual escribe texto como una secuencia de salida de caracteres, almacenando en un buffer (memoria intermedia) los caracteres a escribir, lo que hace más eficiente las operaciones de escritura de caracteres individuales y cadenas; así mismo, se puede especificar el tamaño del buffer, pero si se omite, entonces se usan un tamaño predeterminado que es lo suficientemente grande para la mayoría de los propósitos. La clase **BufferedWriter** tiene los siguientes métodos de interés:

- ◆ *void write()*: Método sobrecargado para guardar un dato en el archivo especialmente cadenas de caracteres y sin incluir un salto de línea automático.
- ◆ *void newLine()*: Este método añade un nuevo salto de línea en el archivo.

- ♦ *void flush()*: Método que vacía el buffer de archivo usado de modo que se garantiza que todos los cambios o escrituras hechos en el archivo a nivel de memoria sean efectivamente guardados en el archivo físico en disco.
- ♦ *void close()*: Método que cierra el archivo y todos los flujos de datos abiertos por este.

En cuanto al acceso de un archivo de texto en modo de lectura, usamos una instancia de la clase **BufferedReader**, la cual lee el texto de una secuencia de entrada de caracteres, almacenándolo en un buffer de caracteres; lo que proporciona una lectura eficiente de caracteres y líneas de texto. El tamaño del búfer puede especificarse, o puede usarse el tamaño predeterminado, que es apropiado para la mayoría de los casos. Los métodos principales de esta clase comprenden:

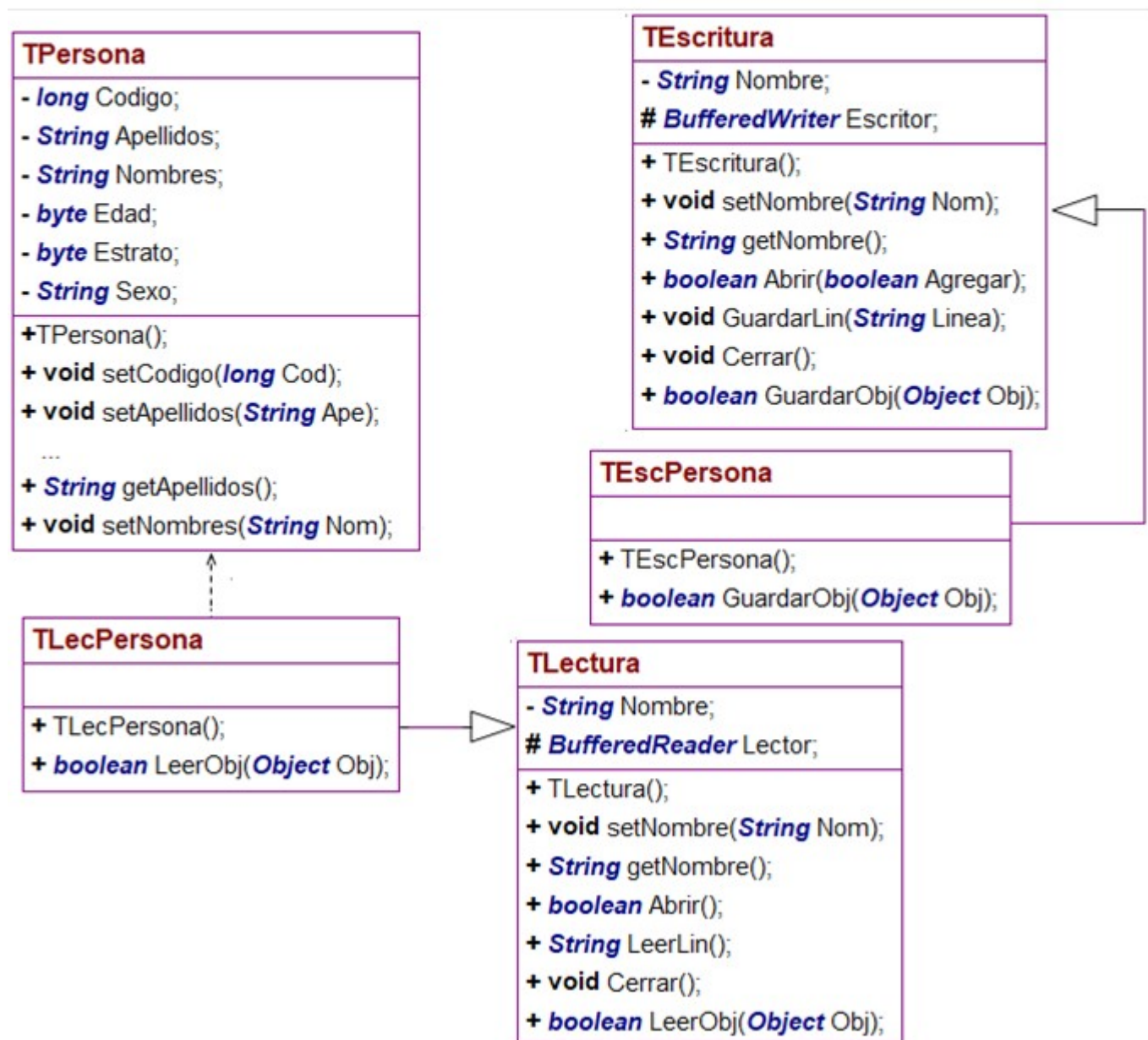
- ♦ *int read()*: Método sobrecargado para leer un carácter o un buffer (arreglo) de caracteres desde el archivo.
- ♦ *String readLine()*: Método que lee y retorna la línea de caracteres actual en el archivo; si no hay más líneas que leer (cuando se llega al final) retorna un **null**.
- ♦ *void mark(int Limite)*: Marca la posición actual en el flujo de datos.
- ♦ *boolean markSupported()*: Indica si el archivo soporta el método *mark()*.
- ♦ *void close()*: Método que cierra el archivo y todos los flujos de datos abiertos por este.
- ♦ *boolean ready()*: Indica si el archivo ya está listo para ser leído.
- ♦ *void reset()*: Restablece el archivo a posición marcada más reciente.
- ♦ *long skip(long n)*: Salta una cantidad de caracteres determinada por el parámetro *n*.

3. Presentación ejemplo serialización de objetos con archivos de texto

La serialización de objetos, es el proceso mediante el cual guardamos o leemos (recuperamos) el estado de los objetos (conjunto de valores actuales contenidos en sus atributos), empleando para ello archivos o cualquier otro mecanismo de persistencia de datos. La serialización también comprende el envío y/o recepción del estado de los objetos a través de una conexión de red. A efectos del ejemplo de serialización, consideraremos almacenar los datos básicos de una persona con la clase *TPersona*; se diseñará e implementará una clase para guardar datos en forma de líneas de caracteres, hasta un archivo de texto plano; la cual contendrá un método abstracto llamado *GuardarObj* para serializar un objeto de cualquier clase. A esta clase la llamaremos *TEscritura* y de ella heredará la clase *TEscPersona*, que se especializara en guardar o serializar un objeto de

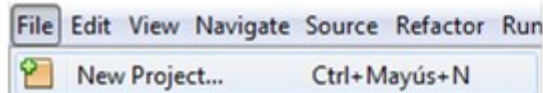
la clase *TPersona*, implementando para ello el método abstracto *GuardarObj* de su clase padre; de tal suerte que se guarde cada propiedad (atributo) del objeto *TPersona* en una línea independiente. De manera análoga tenemos la clase *TLectura*, la cual será una clase abstracta especializada en las operaciones de apertura, lectura, recorrido de un archivo de texto y cierre de este. Incluye además métodos para leer una línea cada vez y un método abstracto llamado *LeerObj* que permitirá recuperar un objeto de cualquier tipo desde el archivo. Para este efecto se diseña e implementa la clase *TLecPersona* que se especializa en leer o recuperar el estado de un objeto de la clase *TPersona*,

4. Diseño de clases para archivos de texto y serialización de objetos

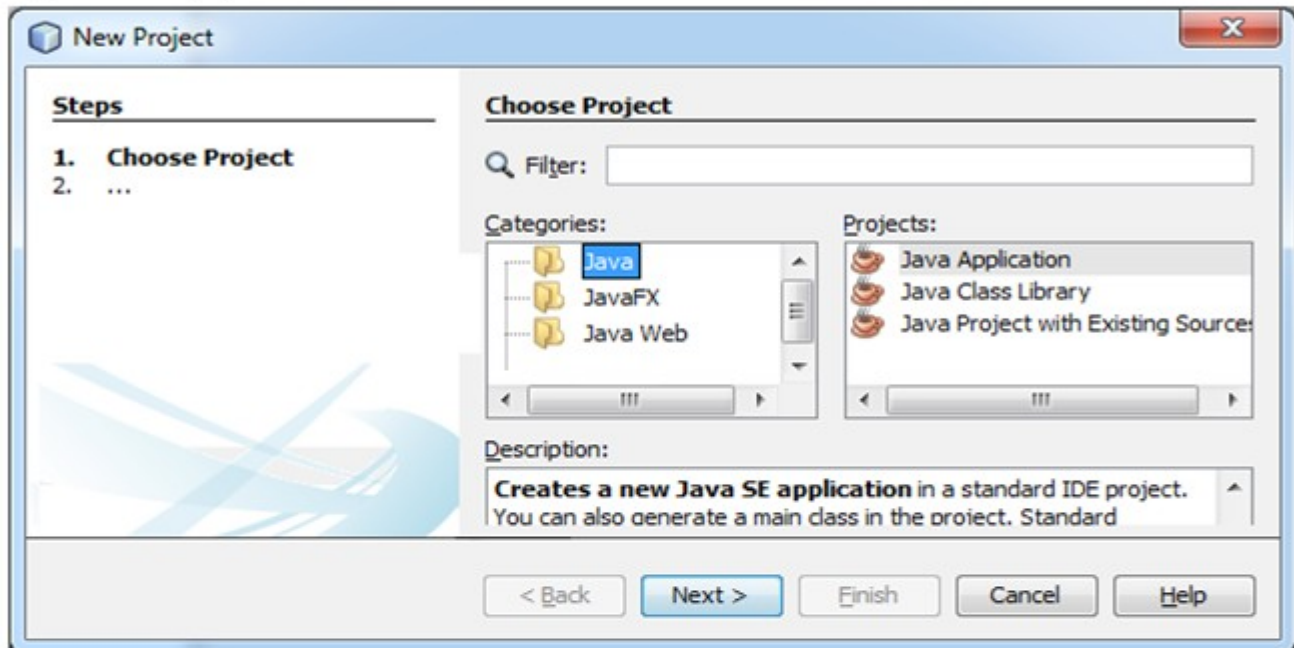


5. Creación del proyecto

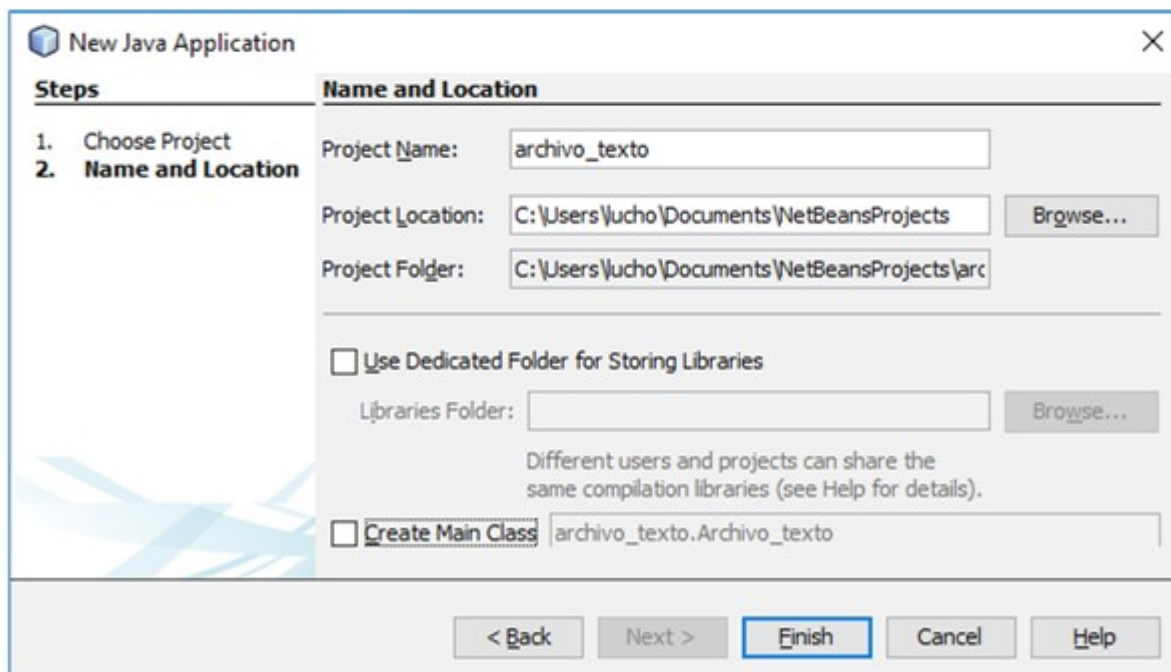
a. Entre *NetBeans* y cree un nuevo proyecto:



b. Seleccione *Java* en el panel **Categories**, en el panel **Projects** seleccione *Java Application* y pulse el botón **Next**

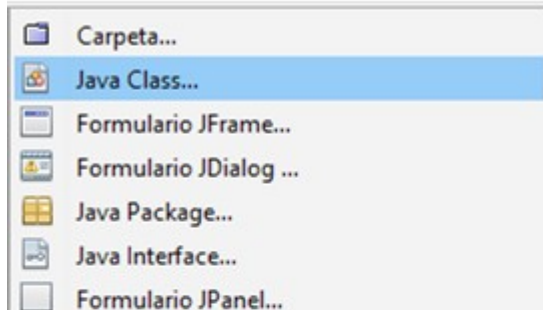


c. Ponga *archivo_texto* en la entrada **Project Name** y desmarque el **CheckBox** titulado **Create Main Class**.

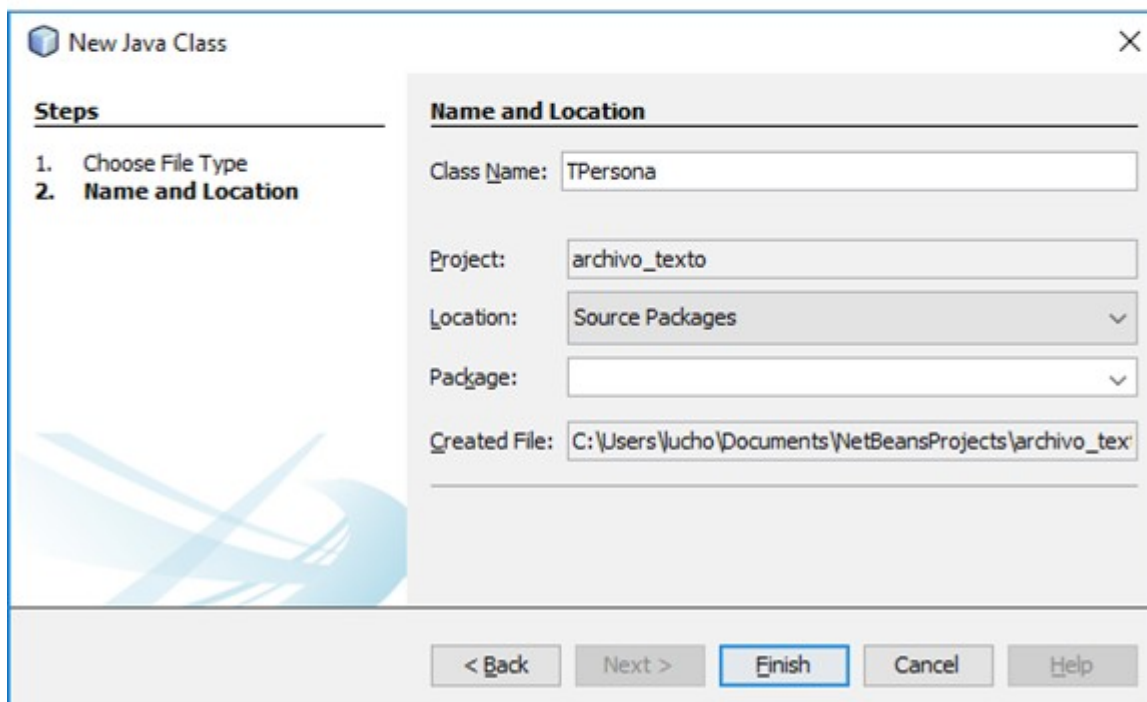


6. Creación de los archivos para implementación de las clases

a. Ahora cree un archivo nuevo para implementar la clase *TPersona* haciendo click derecho en el proyecto y siga las opciones **New + Java Class**.



b. En la ventana desplegada asigne *TPersona* por nombre y luego haga click en el botón **Finish** para crear el archivo java correspondiente como se aprecia en la siguiente imagen.



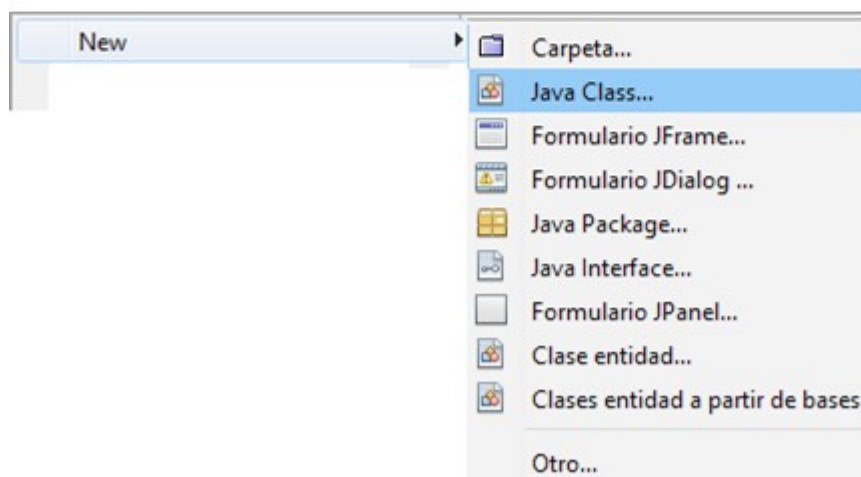
c. Para implementar asegúrese que el código quede como sigue en donde vemos que esta clase solo declara los atributos correspondiente a las características básicas de una persona y los objetos creados de esta clase son los que vamos a serializar:


```
1  /**
2   *
3   * @author lucho
4   */
5  public class TPersona {
6      private long Codigo;
7      private String Apellidos;
8      private String Nombres;
9      private byte Edad;
10     private byte Estrato;
11     private String Sexo;
12
13     public TPersona() {
14         Codigo=0;
15         Apellidos="";
16         Nombres="";
17         Edad=0;
18         Estrato=0;
19         Sexo="";
20     }
21
22     public void setCodigo(long Cod) {
23         Codigo = Cod;
24     }
25
26     public void setApellidos(String Ape) {
27         Apellidos = Ape;
28     }
29
30     public void setNombres(String Nom) {
31         Nombres = Nom;
32     }
33
34     public void setEdad(byte Eda) {
35         Edad = Eda;
36     }
37
38     public void setEstrato(byte Est) {
39         Estrato = Est;
40     }
41
42     public void setSexo(String Sex) {
43         Sexo = Sex;
44     }
```

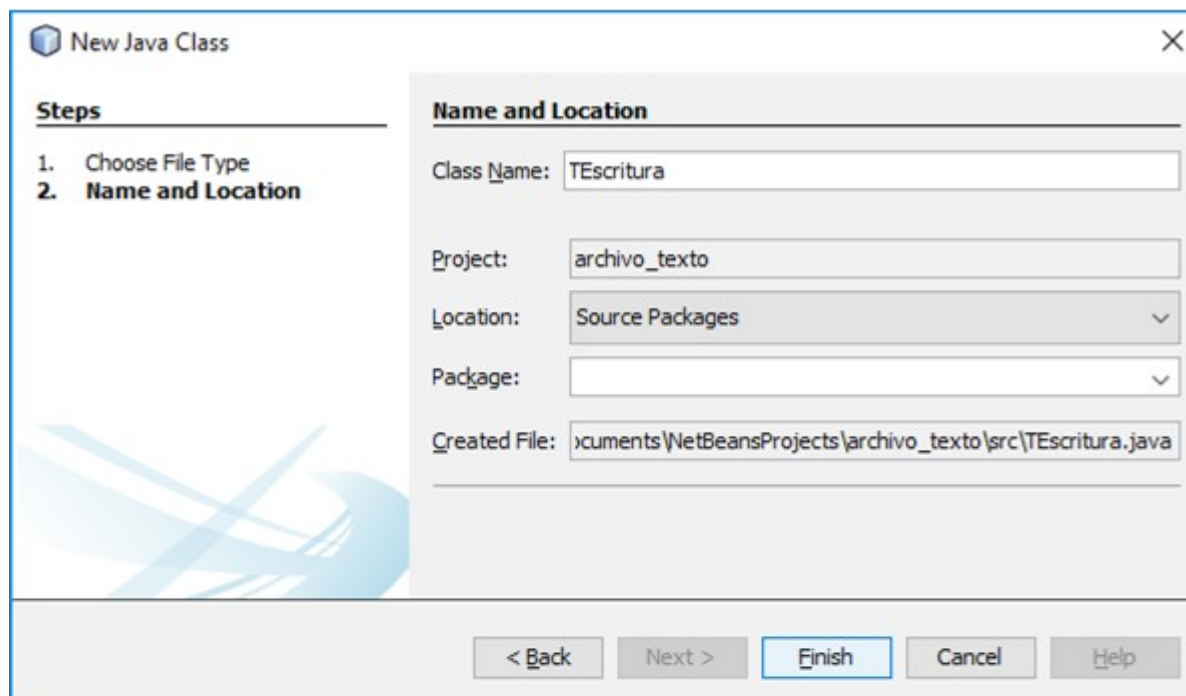


```
45
46 public long getCodigo() {
47     return Codigo;
48 }
49
50 public String getApellidos() {
51     return Apellidos;
52 }
53
54 public String getNombres() {
55     return Nombres;
56 }
57
58 public byte getEdad() {
59     return Edad;
60 }
61
62 public byte getEstrato() {
63     return Estrato;
64 }
65
66 public String getSexo() {
67     return Sexo;
68 }
69 }
```

a. Ahora cree un archivo nuevo para implementar la clase *TEscritura* haciendo click derecho en el proyecto y siga las opciones **New + Java Class**.



- b. En la ventana desplegada asigne *TEscritura* por nombre y luego haga click en el botón **Finish** para crear el archivo java correspondiente como se aprecia en la siguiente imagen.



- c. Implemente esta clase tal como se indica seguidamente:

```
TEscritura.java x
Source History
1 import java.io.FileWriter;
2 import java.io.IOException;
3 import java.io.BufferedWriter;
4
5 /**
6  *
7  * @author lucho
8  */
9
10 public abstract class TEscritura{
11
12     private String Nombre;
13     protected BufferedWriter Escritor;
14
15     public TEscritura() {
16         Nombre="";
17         Escritor=null;
18     }
19 }
```

```

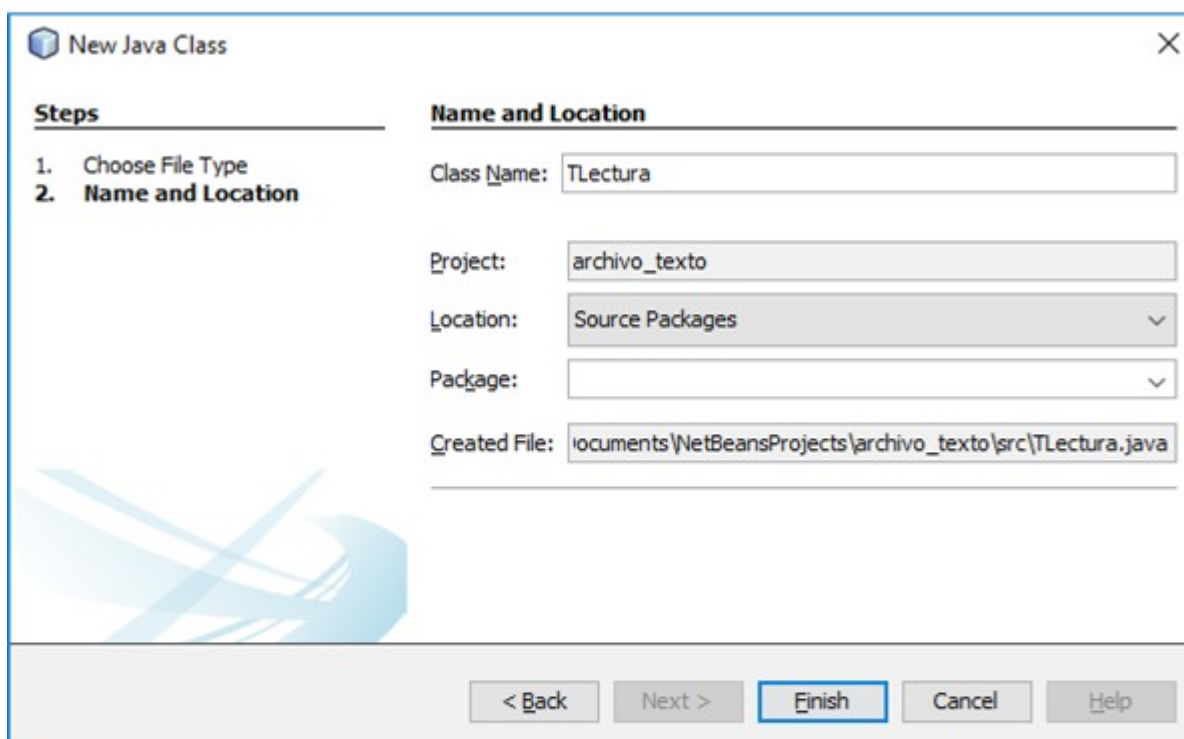
20 public void setNombre(String Nom) {
21     Nombre = Nom;
22 }
23
24 public String getNombre() {
25     return Nombre;
26 }
27
28 public boolean Abrir(boolean Agregar) {
29     Cerrar();
30     try{
31         Escritor=new BufferedWriter(new FileWriter(Nombre,Agregar));
32         return true;
33     }
34     catch(IOException Err) {
35         Err.printStackTrace();
36         return false;
37     }
38 }
39
40 public void GuardarLin(String Linea) {
41     try{
42         Escritor.write(Linea);
43         Escritor.newLine();
44     }
45     catch(IOException Err) {
46         Err.printStackTrace();
47     }
48 }
49
50 public void Cerrar() {
51     if(Escritor!=null) {
52         try{
53             Escritor.flush();
54             Escritor.close();
55             Escritor=null;
56         }
57         catch(IOException Err) {
58             Err.printStackTrace();
59         }
60     }
61 }
62
63 public abstract boolean GuardarObj(Object Obj);
64
65 }

```

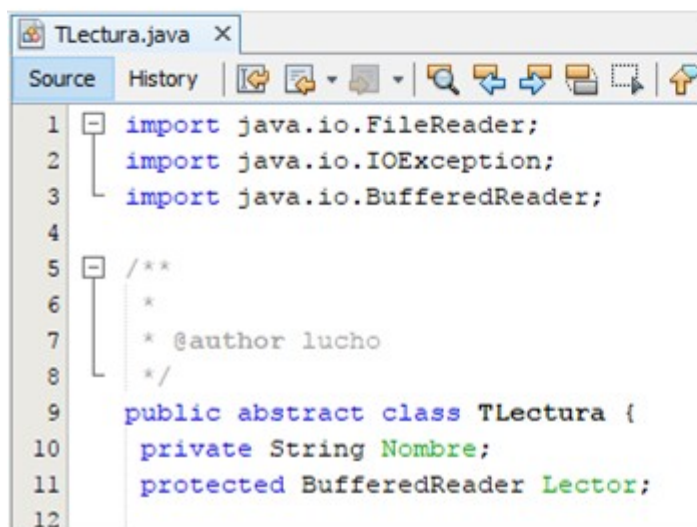
d. Ahora cree la clase *TLectura* haciendo click derecho en el proyecto y siga las opciones **New + Java Class**.



e. En la ventana desplegada asigne *TLectura* por nombre y luego haga click en el botón **Finish** para crear el archivo Java correspondiente como se aprecia en la siguiente imagen:



f. Implemente esta clase como se indica seguidamente:




```

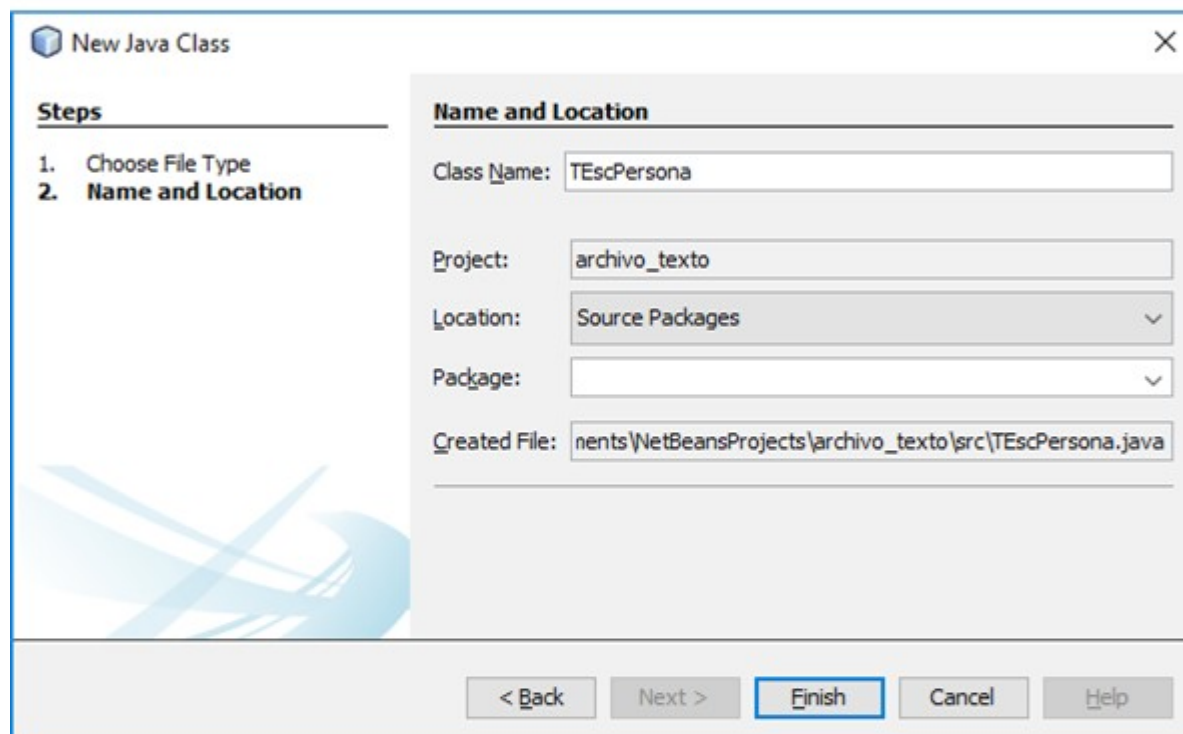
13 public Tlectura() {
14     Nombre="";
15     Lector=null;
16 }
17
18 public void setNombre(String Nom) {
19     Nombre = Nom;
20 }
21
22 public String getNombre() {
23     return Nombre;
24 }
25
26 public boolean Abrir() {
27     Cerrar();
28     try{
29         Lector=new BufferedReader(new FileReader(Nombre));
30         return true;
31     }
32     catch(IOException Err) {
33         Err.printStackTrace();
34         return false;
35     }
36 }
37
38 public String LeerLin() {
39     try{
40         return Lector.readLine();
41     }
42     catch(IOException Err) {
43         Err.printStackTrace();
44         return null;
45     }
46 }
47
48 public void Cerrar() {
49     if(Lector!=null) {
50         try{
51             Lector.close();
52             Lector=null;
53         }
54         catch(IOException Err) {
55             Err.printStackTrace();
56         }
57     }
58 }
59
60 public abstract boolean LeerObj(Object Obj);
61 }

```

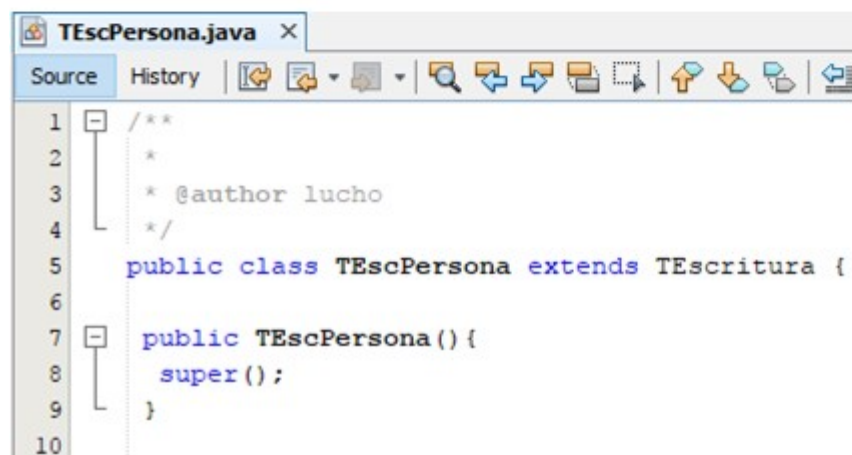
g. Ahora cree la clase *TEscPersona* haciendo click derecho en el proyecto y siga las opciones **New + Java Class**.



h. En la ventana desplegada asigne *TEscPersona* por nombre y luego haga click en el botón **Finish** para crear el archivo java correspondiente como se aprecia en la siguiente imagen:



i. Implemente esta clase como se indica seguidamente:

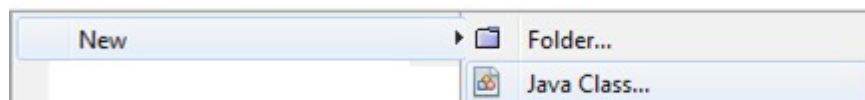


```

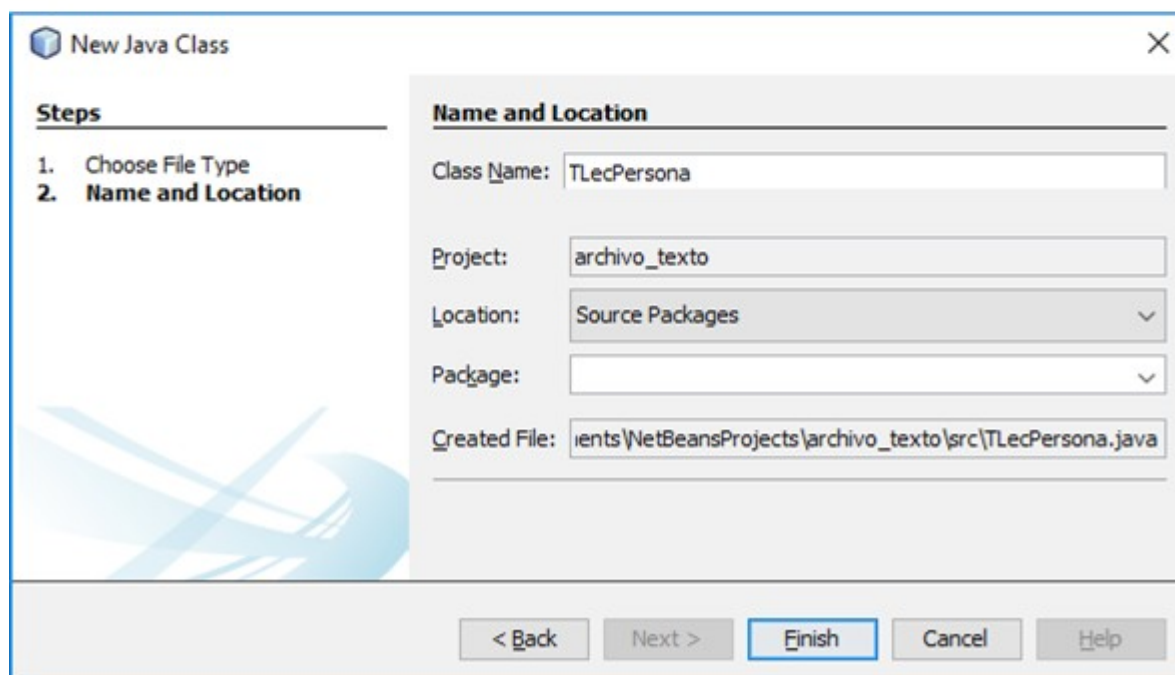
11  @Override
12  public boolean GuardarObj(Object Obj){
13      TPersona Per;
14      if(Obj instanceof TPersona){
15          Per=(TPersona)Obj;
16          GuardarLin(Per.getCodigo() + "");
17          GuardarLin(Per.getApellidos());
18          GuardarLin(Per.getNombres());
19          GuardarLin(Per.getEdad() + "");
20          GuardarLin(Per.getEstrato() + "");
21          GuardarLin(Per.getSexo() + "");
22          return true;
23      }
24      else{
25          return false;
26      }
27  }
28  }

```

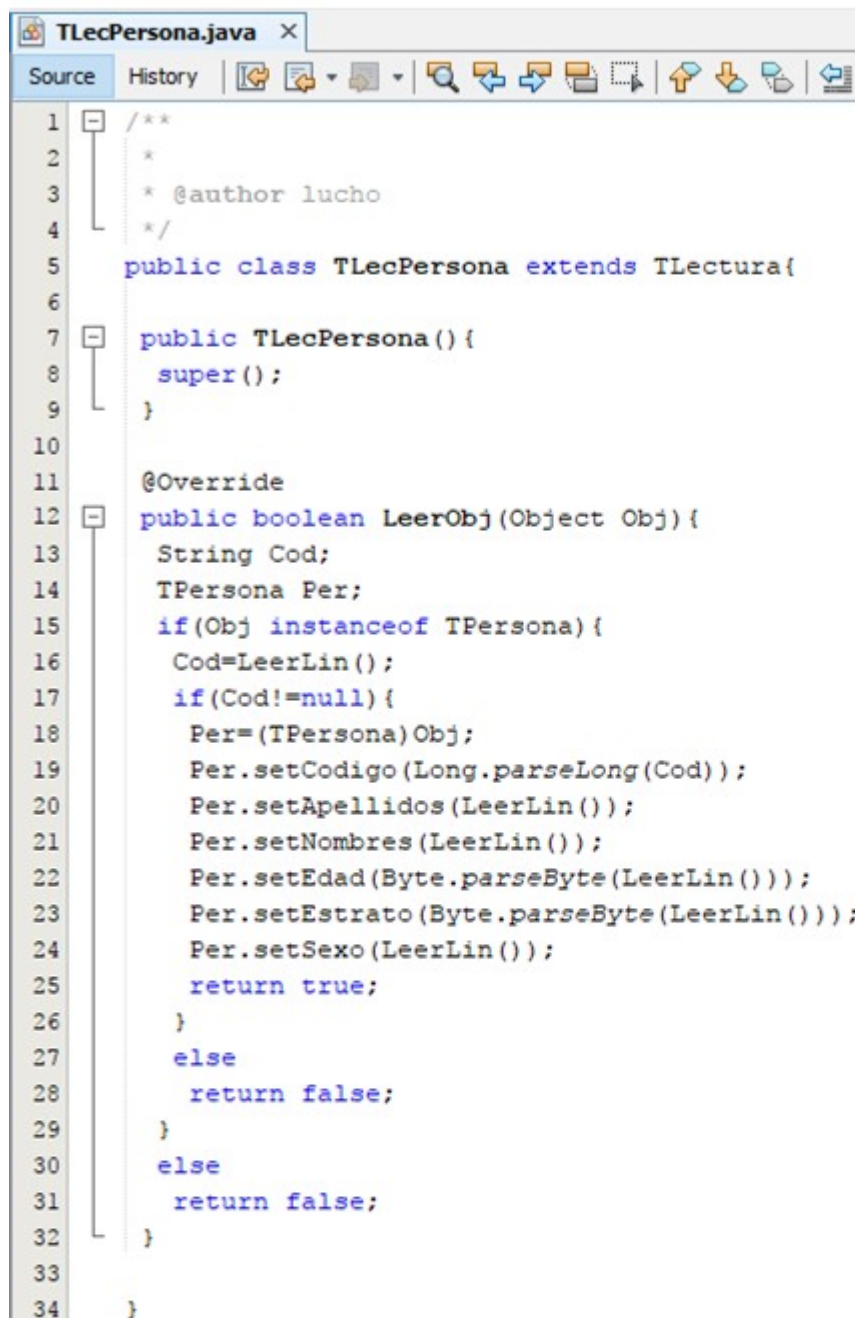
j. Ahora cree la clase *TLecPersona* haciendo click derecho en el proyecto y siga las opciones **New + Java Class**.



k. En la ventana desplegada asigne *TLecPersona* por nombre y luego haga click en el botón **Finish** para crear el archivo Java correspondiente como se aprecia en la siguiente imagen:



I. Implemente esta clase como se indica seguidamente:



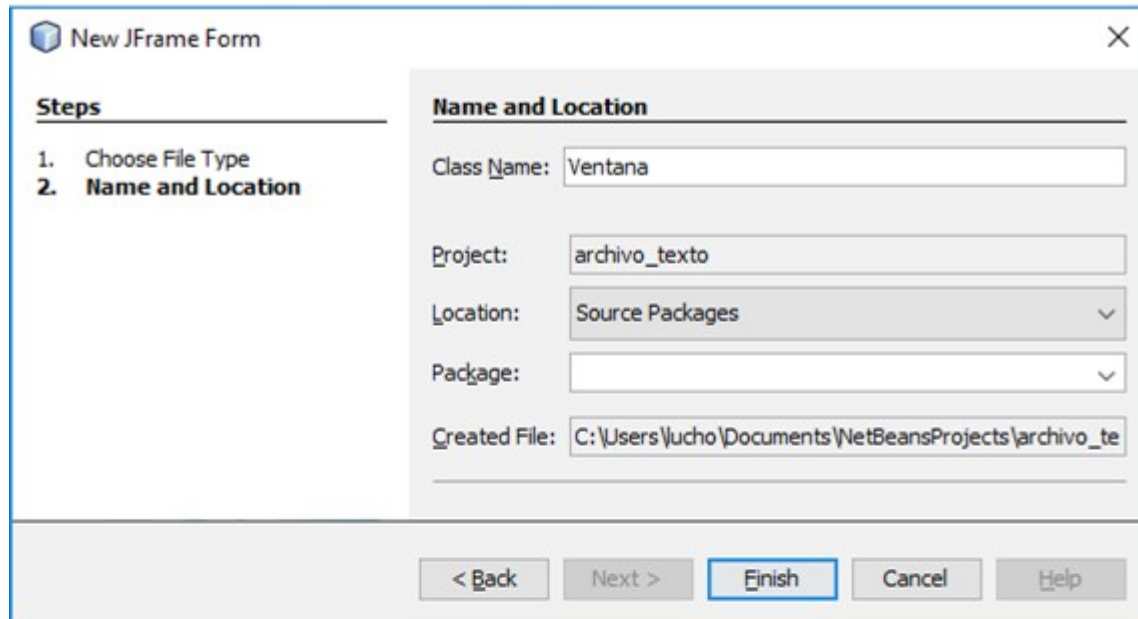
```
1  /**
2   *
3   * @author lucho
4   */
5  public class TlecPersona extends Tlectura{
6
7      public TlecPersona() {
8          super();
9      }
10
11     @Override
12     public boolean LeerObj(Object Obj){
13         String Cod;
14         TPersona Per;
15         if(Obj instanceof TPersona){
16             Cod=LeerLin();
17             if(Cod!=null){
18                 Per=(TPersona)Obj;
19                 Per.setCodigo(Long.parseLong(Cod));
20                 Per.setApellidos(LeerLin());
21                 Per.setNombres(LeerLin());
22                 Per.setEdad(Byte.parseByte(LeerLin()));
23                 Per.setEstrato(Byte.parseByte(LeerLin()));
24                 Per.setSexo(LeerLin());
25                 return true;
26             }
27             else
28                 return false;
29         }
30         else
31             return false;
32     }
33
34 }
```

7. Diseño de la ventana principal.

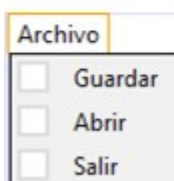
a. Haga click derecho en el proyecto y siga las opciones **New + JFrame Form**.



b. En **Class Name** ponga *Ventana* y haga click en el botón **Finish**:



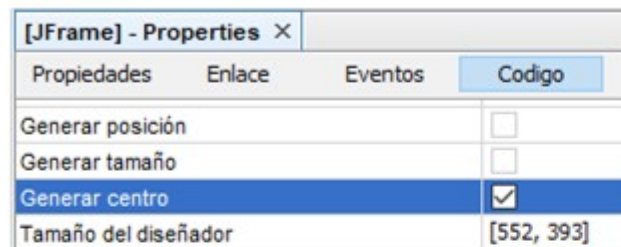
c. En la vista de diseño de la ventana, arrastre una barra de menú (componente **MenuBar**) desde el nodo **Swing menús** de la paleta de componentes. Ahora seleccione y elimine de este la opción **Edit** y renombre la opción **File** por **Archivo**. Luego arrastre sobre el componente **Archivo** un componente **MenuItem** (**Elemento de menú**); cámbiele el texto por **Guardar**. Arrastre otro **MenuItem** (**Elemento de menú**), déjelo caer debajo del anterior y póngale por texto **Abrir**. Repita este mismo proceso y ponga otro **MenuItem** (**Elemento de menú**) debajo de la opción **Abrir**, por título ponga **Salir**, de modo que el menú tenga el siguiente aspecto:



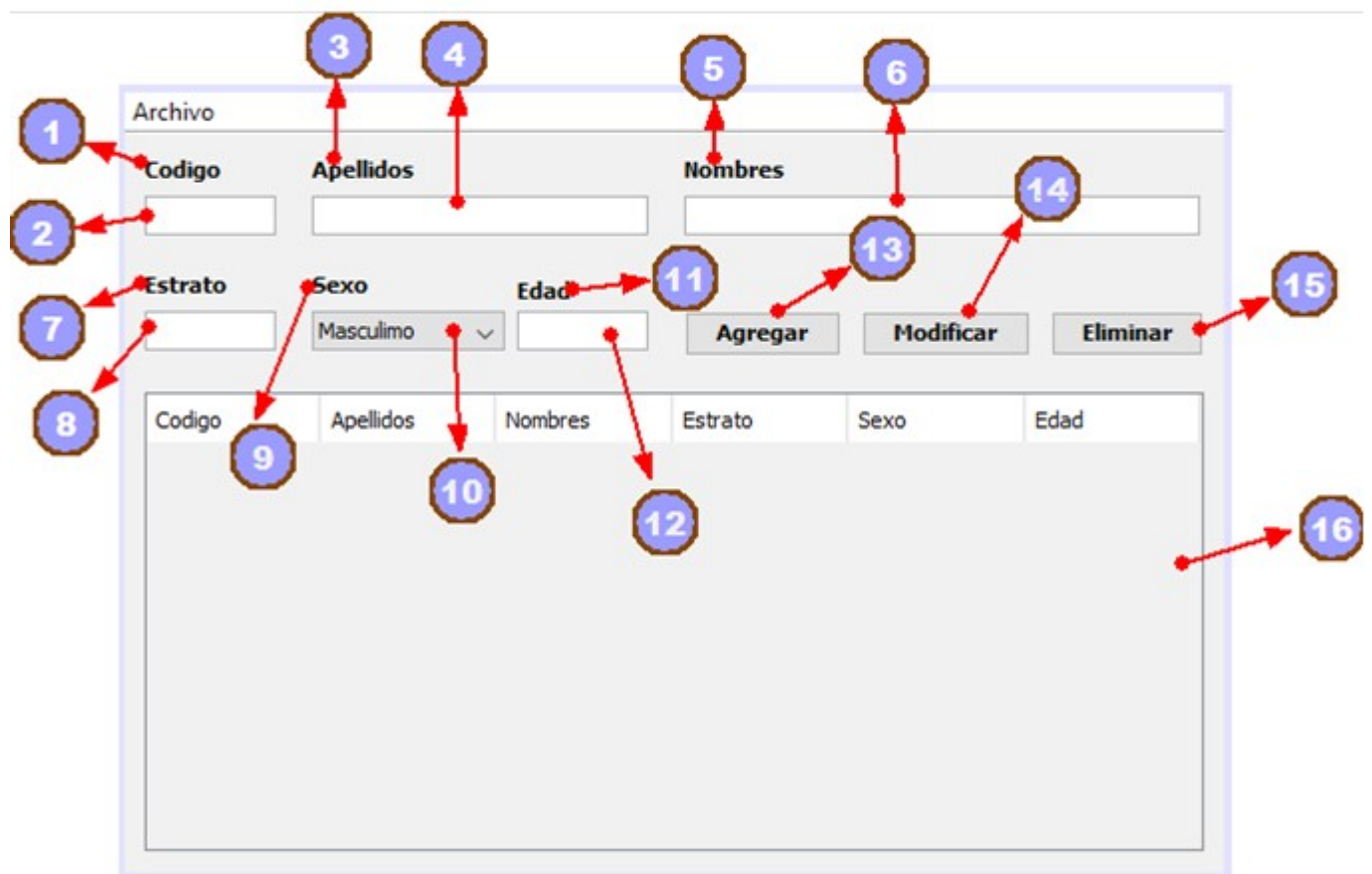
Ahora seleccione el **JFrame** y en el inspector de propiedades póngale en la propiedad **title** el valor de **Ejemplo archivo de texto**, tal como se aprecia en la imagen de abajo:

[JFrame] - Properties	
Propiedades	Codigo
Propiedades	
defaultCloseOperation	EXIT_ON_CLOSE
title	Ejemplo archivo de texto
Otras Propiedades	

Ahora vaya a la ficha **Code** del inspector de propiedades y marque la casilla de verificación **Generar centro**, para que la ventana en ejecución aparezca centrada.



El resto del diseño de la ventana es el que se muestra en la siguiente imagen:



Asegúrese de arrastrar los componentes adecuados hasta la ventana según la imagen anterior, configurando los ilustrados con números según la siguiente tabla:

Nº	Componente	Nombre Variable	Otras propiedades	
1	JLabel		text	Código
			font	font Tahoma 16 Bold
2	JTextField	tf1	text	Borre el texto del control
3	JLabel		text	Apellidos
			font	font Tahoma 16 Bold
4	JTextField	tf2	text	Borre el texto del control
5	JLabel		text	Nombres
			font	font Tahoma 12 Bold
6	JTextField	tf3	text	Borre el texto del control
7	JLabel		text	Estrato
			font	font Tahoma 12 Bold
8	JTextField	tf4	text	Borre el texto del control
9	JLabel		text	Sexo
			font	font Tahoma 12 Bold
10	JComboBox	Cb1	model	Click al botón de tres puntos y escriba: Masculino Femenino
11	JLabel		text	Edad
			font	font Tahoma 12 Bold
12	JTextField	tf5	text	Borre el texto del control

Nº	Componente	Nombre Variable	Otras propiedades	
13	JButton	b1	text	Agregar
			font	font Tahoma 12 Bold ...
14	JButton	b2	text	Modificar
			font	font Tahoma 12 Bold ...
15	JButton	b3	text	Eliminar
			font	font Tahoma 12 Bold ...
16	JTable	tab	<div> <div>model</div> <div>...</div> </div>	Click al botón de tres puntos y defina las propiedades de la tabla como se indica en la siguiente imagen:

tab [JTable] - model

Establecer propiedad **tab's model** utilizando:

Personalizador del modelo de tabla

Modelo de tabla

Ajustes de la tabla

Valores predeterminados

Indicar los tipos del título y de la columna aquí:

Columna	Título	Tipo	Editable
1	Codigo	Long	<input type="checkbox"/>
2	Apellidos	String	<input type="checkbox"/>
3	Nombres	String	<input type="checkbox"/>
4	Estrato	Byte	<input type="checkbox"/>
5	Sexo	String	<input type="checkbox"/>
6	Edad	Byte	<input type="checkbox"/>

Insertar

Eliminar

Subir

Bajar

Filas: 0

+

-

Columnas: 6

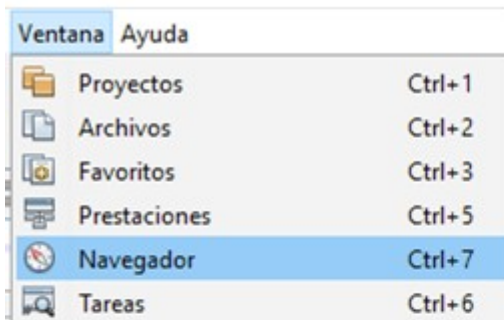
+

-

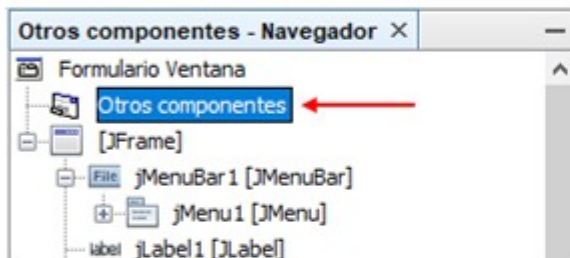
Aceptar

Restablecer Valores por Defecto

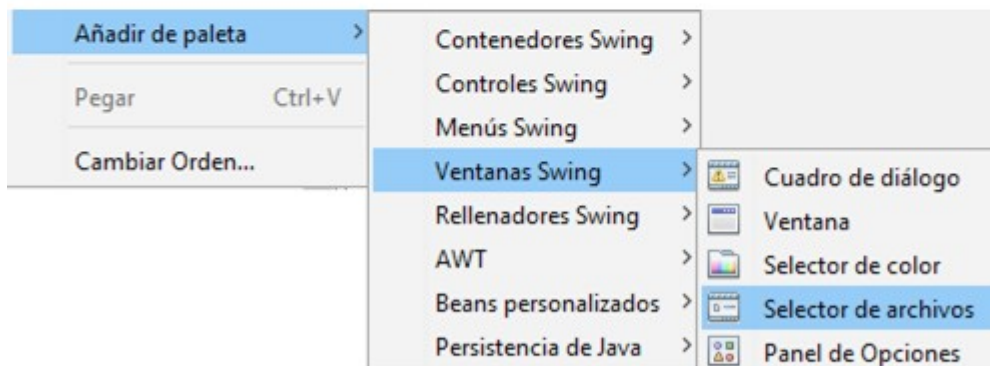
Active la ventana del **Navegador (Navigator)**; para tal caso, si no está visible, vaya a la opción de menú Ventana (**Windows**) **Navegador (Navigator)**; o pulse la combinación de teclas CTRL + 7:



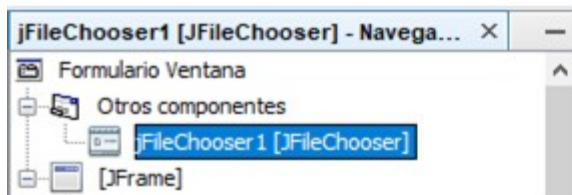
En esta ventana seleccione el nodo **Otros componentes (Others components)**



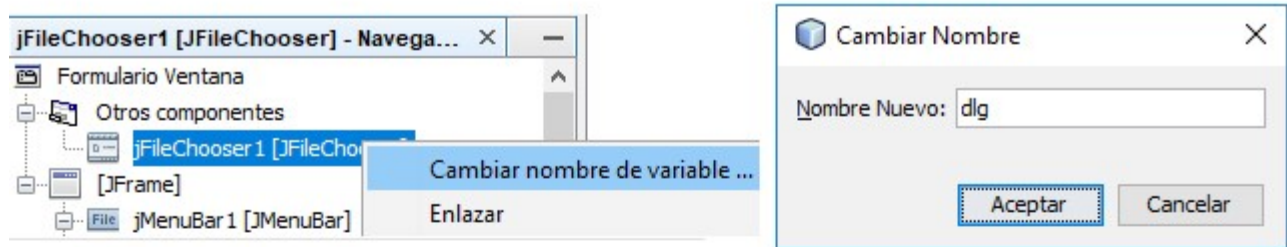
Haga click derecho sobre este nodo y tome las opciones **Añadir de paleta + Ventanas Swing + Selector de archivos (JFileChooser)**



Esto añade un componente **JFileChooser** que es un cuadro de dialogo para abrir y guardar archivos.



Haga click derecho sobre este control para cambiarle el nombre y llámelo como **dlg**



8. Implementación de métodos de la ventana

a. Vaya la vista de fuente de la ventana, ubíquese al principio del archivo y antes de la línea `public class Ventana extends javax.swing.JFrame {` haga la importación de las siguientes clases:

```
import javax.swing.JOptionPane;
import javax.swing.JFileChooser;
import javax.swing.table.DefaultTableModel;
```

b. Ahora diríjase debajo de la declaración y añada el atributo **Mod** y su respectiva inicialización, tal como se señaladas con la flecha roja:

```
public class Ventana extends javax.swing.JFrame {

    private DefaultTableModel Mod; ←

    public Ventana() {
        initComponents();
        Mod=(DefaultTableModel) tab.getModel(); ←
    }
}
```

c. Dentro del cuerpo de la clase de la ventana, añada los siguientes métodos:

```
private void Llenar(TPersona Per) {
    Per.setCodigo(Long.parseLong(tf1.getText()));
    Per.setApellidos(tf2.getText());
    Per.setNombres(tf3.getText());
    Per.setEstrato(Byte.parseByte(tf4.getText()));
    Per.setEdad(Byte.parseByte(tf5.getText()));
    Per.setSexo(cb1.getSelectedItem()+"");
}
```

```

private void Llenar(int Fila,TPersona Per){
    Per.setCodigo((long)Mod.getValueAt(Fila,0));
    Per.setApellidos(Mod.getValueAt(Fila,1)+"");
    Per.setNombres((String)Mod.getValueAt(Fila,2));
    Per.setEstrato((byte)Mod.getValueAt(Fila,3));
    Per.setSexo(Mod.getValueAt(Fila,4)+"");
    Per.setEdad((byte)Mod.getValueAt(Fila,5));
}

private void Mostrar(int Fila,TPersona Per){
    Mod.setValueAt(Per.getCodigo(),Fila,0);
    Mod.setValueAt(Per.getApellidos(),Fila,1);
    Mod.setValueAt(Per.getNombres(),Fila,2);
    Mod.setValueAt(Per.getEstrato(),Fila,3);
    Mod.setValueAt(Per.getSexo(),Fila,4);
    Mod.setValueAt(Per.getEdad(),Fila,5);
}

private void Agregar(TPersona Per){
    int fila;
    fila=Mod.getRowCount();
    Mod.setRowCount(fila+1);
    Mostrar(fila,Per);
}

private void Limpiar(){
    tf1.setText("");
    tf2.setText("");
    tf3.setText("");
    tf4.setText("");
    tf5.setText("");
    tf1.grabFocus();
}

private void VerMensaje(String Texto){
    JOptionPane.showMessageDialog(null,Texto);
}

```

```

private void Guardar(String Archivo){
    int i;
    TPersona Per;
    TEscPersona EP;
    EP=new TEscPersona();
    EP.setNombre(Archivo);
    if(EP.Abrir(false)){
        Per=new TPersona();
        for(i=0;i<Mod.getRowCount();i++){
            Llenar(i,Per);
            EP.GuardarObj(Per);
        }
        EP.Cerrar();
        VerMensaje("Archivo guardado");
    }
    else{
        VerMensaje("Error creando archivo");
    }
}

private void Cargar(String Archivo){
    TPersona Per;
    TLecPersona LP;
    LP=new TLecPersona();
    LP.setNombre(Archivo);
    if(LP.Abrir()){
        Per=new TPersona();
        while(LP.LeerObj(Per)){
            Agregar(Per);
        }
        LP.Cerrar();
    }
    else{
        VerMensaje("Error abriendo archivo");
    }
}

```


9. Implementación de eventos de la ventana

a. Seleccione el botón **Agregar** y haga doble click sobre él, asegurándose de implementar su evento de la siguiente manera:

```
private void b1ActionPerformed(java.awt.event.ActionEvent evt) {  
    TPersona Per;  
    Per=new TPersona();  
    Llenar(Per);  
    Agregar(Per);  
    Limpiar();  
}
```

b. Ahora seleccione el botón **Modificar** y haga doble click sobre este asegurándose de que el código para su evento sea el siguiente:

```
private void b2ActionPerformed(java.awt.event.ActionEvent evt) {  
    int fila;  
    TPersona Per;  
    fila=tab.getSelectedRow();  
    if(fila>=0){  
        Per=new TPersona();  
        Llenar(Per);  
        Mostrar(fila,Per);  
        Limpiar();  
    }  
    else{  
        VerMensaje("Seleccione una persona en la tabla");  
    }  
}
```

c. Seleccione el botón **Eliminar** y haga doble click sobre este asegurándose de que el código para su evento sea el siguiente:

```
private void b3ActionPerformed(java.awt.event.ActionEvent evt) {  
    int fila=tab.getSelectedRow();  
    if(fila>=0){  
        Mod.removeRow(fila);  
        Limpiar();  
    }  
    else{  
        VerMensaje("Seleccione una persona en la tabla");  
    }  
}
```

d. Para implementar las opciones de la barra de menú, haga click en el elemento **Archivo** (parte superior de la ventana) y haga doble click en la opción **Guardar**. El código del evento para esta opción debe ser el siguiente:

```
private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
    dlg.setDialogTitle("Guardar archivo de personas como");
    if(dlg.showSaveDialog(null)==JFileChooser.APPROVE_OPTION) {
        Guardar(dlg.getSelectedFile().getAbsolutePath());
    }
}
```

e. Siguiendo los mismos pasos anteriores, ahora haga doble click en la opción **Cargar** y asegúrese que su implementación sea la siguiente:

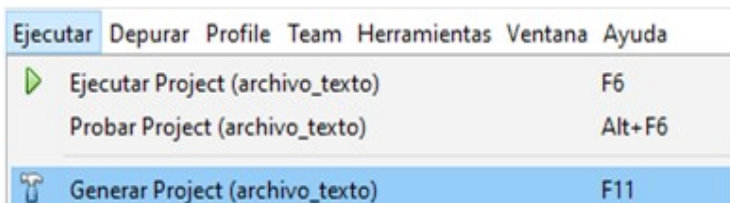
```
private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
    dlg.setDialogTitle("Abrir archivo de personas");
    if(dlg.showOpenDialog(null)==JFileChooser.APPROVE_OPTION) {
        Cargar(dlg.getSelectedFile().getAbsolutePath());
    }
}
```

f. Finalmente haga doble click en la opción **Salir** y asegúrese que su implementación sea la siguiente:

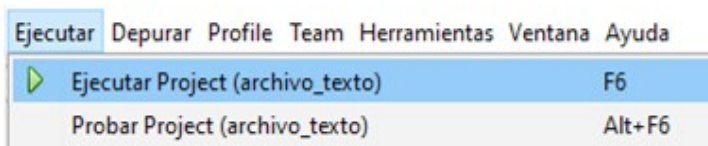
```
private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
    dispose();
}
```

10. Compilación y ejecución del programa.

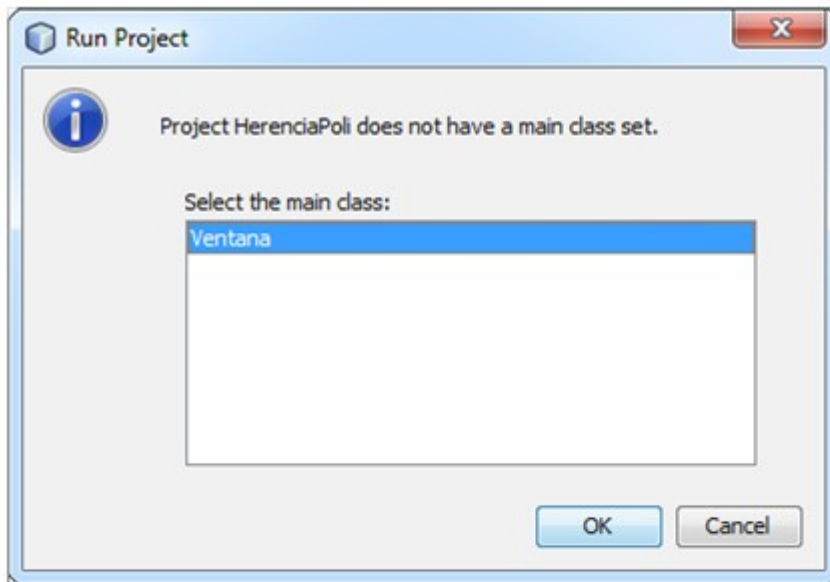
Compile con la opción **"Ejecutar"** + **"Generar Project"** del menú o pulse la tecla **F11**.



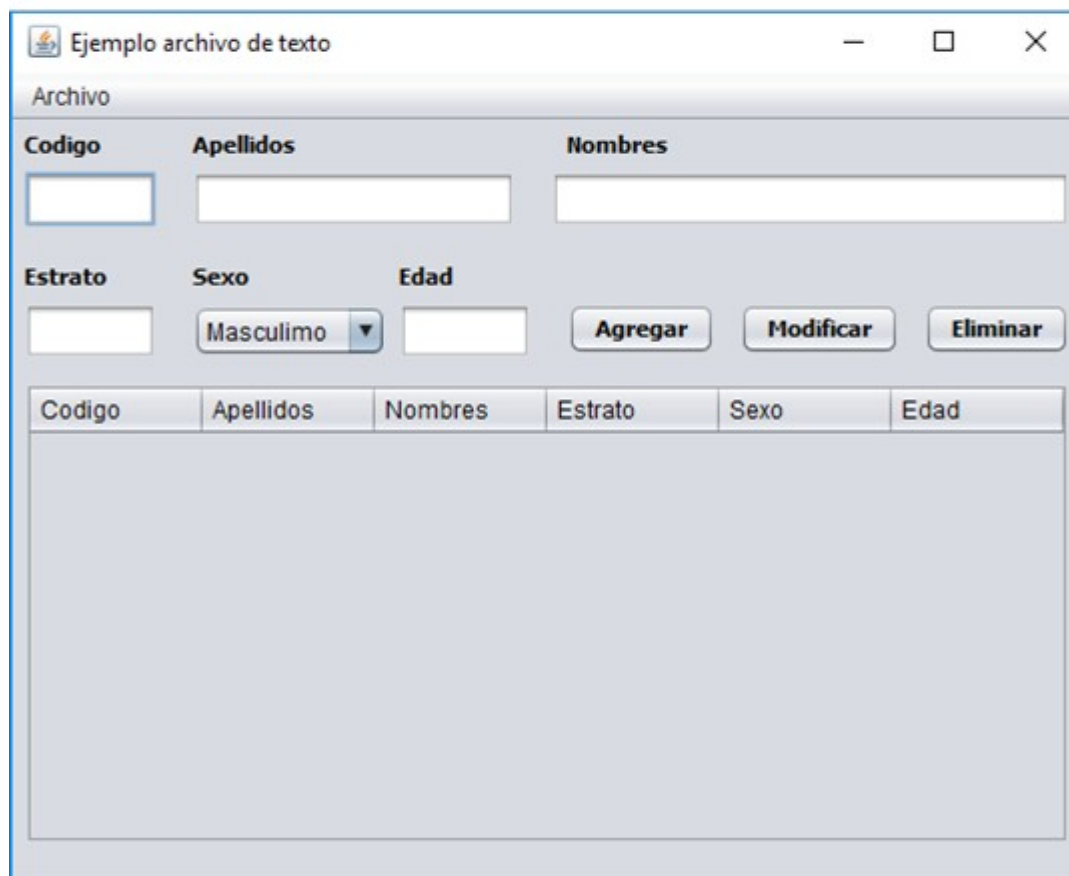
Corrija los errores comparando con el código fuente anterior y corra el programa con la opción **"Ejecutar"** + **"Ejecutar Project"** del menú principal, o pulsando la tecla **F6**:



Si le aparece la siguiente ventana, en ella haga click al botón “**OK**”.



La siguiente imagen ilustra una captura del programa en ejecución:



Codigo	Apellidos	Nombres	Estrato	Sexo	Edad

Actividad propuesta.

1. Explique qué significan las dos siguientes líneas.

```
if (Dlg.showSaveDialog(null) == JFileChooser.APPROVE_OPTION)
if (Dlg.showOpenDialog(null) == JFileChooser.APPROVE_OPTION)
```

2. Modifique el programa anterior para que no permita el ingreso de dos personas con el mismo código.
3. Modifique el ejemplo anterior insertando al principio una nueva columna en la tabla, de modo que tenga el título N° y muestre el número de fila o registro de cada persona.
4. A la aplicación anterior cuando el usuario seleccione una fila de la tabla, los datos de la persona correspondiente a esa fila, deben ser mostrados en los **JTextField** y el **JComboBox** automáticamente.