

UNIVERSIDAD DE CORDOBA

FACULTAD DE INGENIERIAS

PROGRAMA INGENIERIA DE SISTEMAS

CURSO: Programación I

TEMA: Archivos binarios en Java.

DESCRIPCION: En este documento se aborda el tema de los archivos binarios, su conceptualización, características e implementación en el lenguaje Java. Para este último efecto se hace un tratamiento de la clase que este lenguaje provee tanto para la lectura como para la escritura de datos en archivos binarios, los métodos que estas clases tienen para este propósito y las excepciones requeridas para los métodos de esta clase. En esta oportunidad la demostración práctica de los conceptos de archivos binarios se desarrollará con una aplicación de interfaz grafica de usuario (GUI) construido en NetBeans basada en los controles swing bajo el sistema operativo Windows, en la cual se ilustra el proceso de creación, apertura, lectura y escritura de archivos binarios. Para ello seguiremos la misma metodología orientada a objetos, diseñando e implementando una clase para escritura y otra para lectura de archivos binario, de modo que permitan guardar y leer arreglos de bytes desde un archivo binarios.

OBJETIVO: Diseñar e implementar en el lenguaje Java una clase para realizar las operaciones de creación, apertura, escritura y lectura de archivos binarios para guardar y recuperar bloques de bytes que permitan hacer operaciones como copia y encriptamiento de archivos, usando para ello una aplicación de interfaz gráfica bajo swing construida con el IDE Netbeans en Windows.

PALABRAS CLAVES: Archivos binarios en Java, clases *File*, *JFileChooser* y *RandomAccessFile*, diálogos para apertura y guardado de archivos en Java.

Luis Roberto Olascoaga Surmay

1. Archivos binarios

Los archivos binarios a diferencia de los archivos de textos son tratados como un flujo de bytes, lo que significa que las operaciones de lectura y de escritura se realizan en bloques o segmentos de bytes, donde la cantidad o número de bytes de cada bloque puede variar o ser constante durante cada operación. Cuando abrimos un archivo en forma binaria, no es obligado conocer el significado ni la estructura del archivo para poder hacer operaciones de lectura o de escritura, por lo cual los archivos binarios son ideales para realizar operaciones sobre archivos típicas de un sistema operativo, tales como copiar, partir, unir, encriptar archivos de cualquier tipo entre otras. Normalmente un archivo creado en forma binaria no es legible directamente, pues las secuencias de bytes que lo conforma no necesariamente se corresponde a los códigos y formatos de caracteres legibles o interpretables como sucede con un archivo de texto; lo que será comprobado en el ejemplo que se desarrollara posteriormente, en donde escribiremos cadenas de texto compuestos de caracteres normales y legibles hasta un archivo binario, sin embargo si se trata de abrir este archivo en un editor de texto no podrá encontrar la ocurrencia de las cadenas de texto que usted sabe que guardo en dicho archivo. Por esta razón el significado del contenido de un archivo binario depende de cómo (estructura internas), con qué fin y muchas veces del programa con que se han creado.

En consecuencia un archivo binario puede ser leído o escrito en bloques de bytes de tamaño variable, pudiendo tener cada uno de ellos un significado diferente. Así por ejemplo en los entornos Unix y GNU/Linux los programas o archivos ejecutables usan muy frecuentemente el formato ELF (*Executable and Linkable Format*) o formato ejecutable y enlazable por sus siglas en ingles. En este formato de archivos ejecutables la cabecera (primeros bytes) contiene información que permite interpretar el contenido de dichos archivos. El primer campo o propiedad de cabecera en el formato ELF es el llamado número mágico que identifica al ejecutable y comprende los cuatro primeros bytes del archivo ejecutable, de modo que el primer byte contiene el numero hexadecimal 7F (127 en decimal) y los otros tres bytes contienen los caracteres o letras 'E', 'L' y 'F'.

Para citar otro ejemplo al respecto consideremos el contenido de un archivo de sonido MP3, en el que existe una parte que contiene una descripción del archivo la cual comprende los primeros o los últimos 128 bytes del archivo (dependiendo de la versión del formato) y el resto de bytes suponemos contienen los datos propios de la canción. La estructura u

organización y el significado de estos 128 bytes de este tipo de archivos obedece a la definición de las llamadas etiquetas (Tags) ID3 de archivos de sonido MP3, la cual comprende siete campos o descriptores especificados en la siguiente tabla:

Campo	Tamaño (bytes)	Descripción
TagID	3	Los primeros tres bytes (del byte de posición 0 al 2) identifican a la etiqueta y siempre contienen el la version del TAG como por ejemplo ID3.
Title	30	Los siguientes treinta bytes (del byte 3 al 32) almacenan el titulo de la canción.
Artist	30	Los siguientes treinta bytes (del byte 33 al 62) contienen el nombre del artista.
Album	30	Los siguientes treinta bytes (del byte 63 al 92) almacenan el titulo del album.
Year	4	Los siguientes cuatro bytes (del byte 93 al 96) contiene el año de la canción o del album.
Comment	30	Los siguientes treinta bytes (del byte 97 al 126) almacenan un comentario o descripción de la canción.
Genere	1	El siguiente byte (el byte numero 127) contiene el genero de la canción.

Por otra parte el acceso a un archivo binario puede hacerse de forma aleatoria, lo que significa que tanto en las operaciones de entrada (lectura de datos) como en las de salida (escritura de datos) relacionadas con archivos binarios se permiten hacer saltos hacia adelante o hacia atrás en cualquier momento. Estos desplazamientos se miden en bytes o bloques de bytes y no necesariamente tiene que ser del mismo tamaño siempre. Los saltos o desplazamientos pueden hacerse de forma relativa a la posición del byte actual en el archivo (puntero de lectura o de escritura del archivo), o también se realizan de forma absoluta y para este ultimo caso puede referirse a moverse en el archivo sea desde el principio de este o desde el final. En este punto es necesario aclarar que los mecanismos de desplazamiento dentro del archivo depende mucho de la implementación (API) subyacente del lenguaje de programación empleado.

Luis Roberto Olascoaga Surmay

2. Archivos binarios en Java

En lo que respecta al lenguaje java para el tratamiento de archivos binarios se emplea la clase **RandomAccessFile** del paquete *java.io*, clase que es la responsable de la creación y/o apertura de archivos binarios tanto en modo de escritura como en modo de lectura e incluso habitualmente en ambos modos. De esta manera uno de los constructores de esta clase tiene definido dos parámetros ambos de tipo *String*: el primero de ellos es el nombre del archivo como una cadena de caracteres, el cual eventualmente contiene la unidad y directorio del archivo. El segundo corresponde al modo de modo de apertura del archivo, el cual puede contener la letra *r* para lectura, la *w* para escritura y más habitualmente usamos la combinación *rw* para habilitar ambos tipos de operación. Los métodos más importantes de la clase **RandomAccessFile** para tratar con archivos binarios son los siguientes, teniendo en cuenta que todos ellos deben tratar con la excepción del tipo *IOException*:

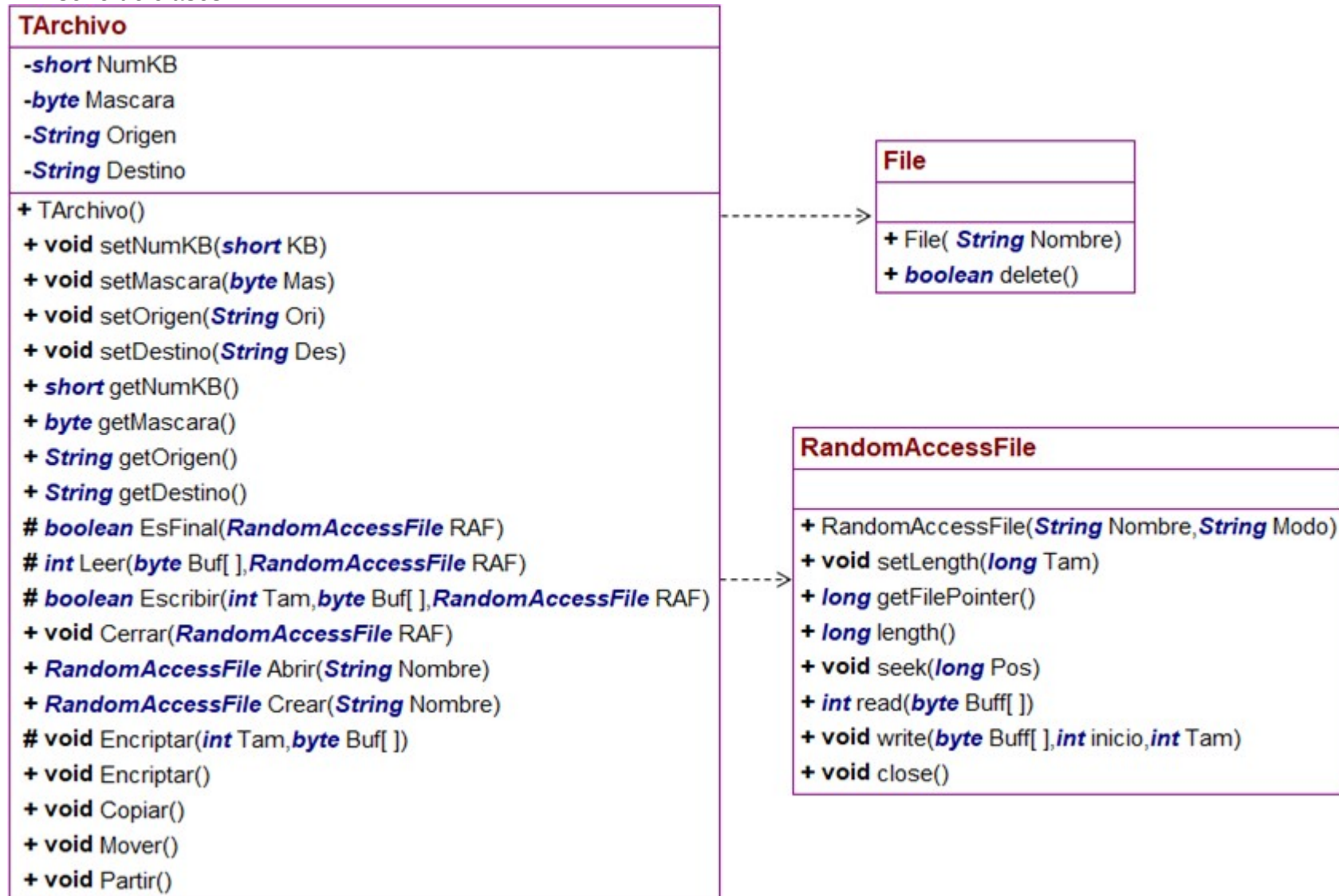
- ♦ *length*: Retorna un tipo *long* con el número de bytes del tamaño del archivo.
- ♦ *getFilePointer*: Retorna un tipo *long*, que es la posición del puntero de lectura o escritura; es decir, la posición del byte activo en el archivo.
- ♦ *seek*: Este método mueve el puntero de lectura o escritura a una posición o byte dado dentro del archivo.
- ♦ *getFD*: Devuelve el objeto descriptor asociado al flujo del archivo.
- ♦ *setLength*: Cambia el tamaño del archivo asignándole un nuevo tamaño dado en bytes.
- ♦ *read*: Método sobrecargado para leer un buffer (arreglo) de bytes o también una parte dada de un arreglo de bytes y en ambos casos retorna el número de bytes leídos.
- ♦ *readByte*: Método que lee y retorna un entero del tipo *byte*.
- ♦ *readBoolean*: Método que lee y retorna un valor de tipo *boolean*.
- ♦ *readChar*: Método que lee y retorna un valor de tipo *char*.
- ♦ *readFloat*: Método que lee y retorna un número real del tipo *float*.
- ♦ *readDouble*: Método que lee y retorna un número real del tipo *double*.
- ♦ *readShort*: Método que lee y retorna un valor entero del tipo *short*.
- ♦ *readInt*: Método que lee y retorna un valor entero del tipo *int*.
- ♦ *readLong*: Método que lee y retorna un valor entero del tipo *long*.
- ♦ *readLine*: Lee la siguiente línea de texto del archivo retornando un *String*.

- ♦ *readUTF*: Lee una cadena de caracteres desde el archivo y la retorna como un *String* en formato UTF.
- ♦ *write*: Método sobrecargado para escribir un buffer (arreglo) de bytes o también una parte dada de un arreglo de bytes.
- ♦ *writeByte*: Método que escribe en el archivo un entero del tipo byte.
- ♦ *writeBoolean*: Método que escribe en el archivo un valor de tipo boolean.
- ♦ *writeChar*: Método que escribe en el archivo un valor de tipo char.
- ♦ *writeFloat*: Método que escribe en el archivo un número real del tipo float.
- ♦ *writeDouble*: Método que escribe en el archivo un número real del tipo double.
- ♦ *writeShort*: Método que escribe en el archivo un valor entero del tipo short.
- ♦ *writeInt*: Método que escribe en el archivo un valor entero del tipo int.
- ♦ *writeLong*: Método que escribe en el archivo un valor entero del tipo long.
- ♦ *writeBytes*: Escribe una cadena (*String*) en el archivo como una secuencia de bytes.
- ♦ *writeUTF*: Escribe en el archivo una cadena de caracteres (*String*) en formato UTF.
- ♦ *Close*: Método que cierra el archivo y todos los flujos de datos abiertos por este.

3. Presentación ejemplo archivos binarios

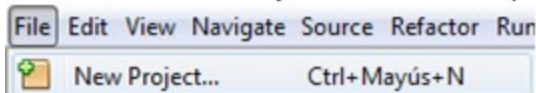
En este ejemplo de archivos binarios en Java, realizaremos las operaciones de copiar, encriptar, mover y particionar un archivo dado, que será seleccionado por el usuario usando un dialogo de apertura de archivo. Para todas las operaciones igualmente el usuario seleccionara desde un dialogo de archivos el nombre del archivo destino o archivo resultante de las operaciones en cuestión. La operación de encriptamiento la haremos byte a byte, de modo que para tal efecto el usuario deberá ingresar una máscara de encriptamiento (valor del tipo byte de -128 a +127) la cual se combinara con cada byte del archivo usando el operador de bits xor (^), de tal suerte que dicha operación sea reversible y podamos obtener el archivo original a partir del archivo resultante. Para el caso de la división de un archivo en partes, el usuario podrá indicar el tamaño de cada parte en kilo bytes (1024 bytes para el caso) y el programa deberá generar automáticamente cada archivo con el tamaño indicando y considerando que los nombre de cada parte se distinguen enumerándolos automáticamente en forma consecutiva.

4. Diseño de clases

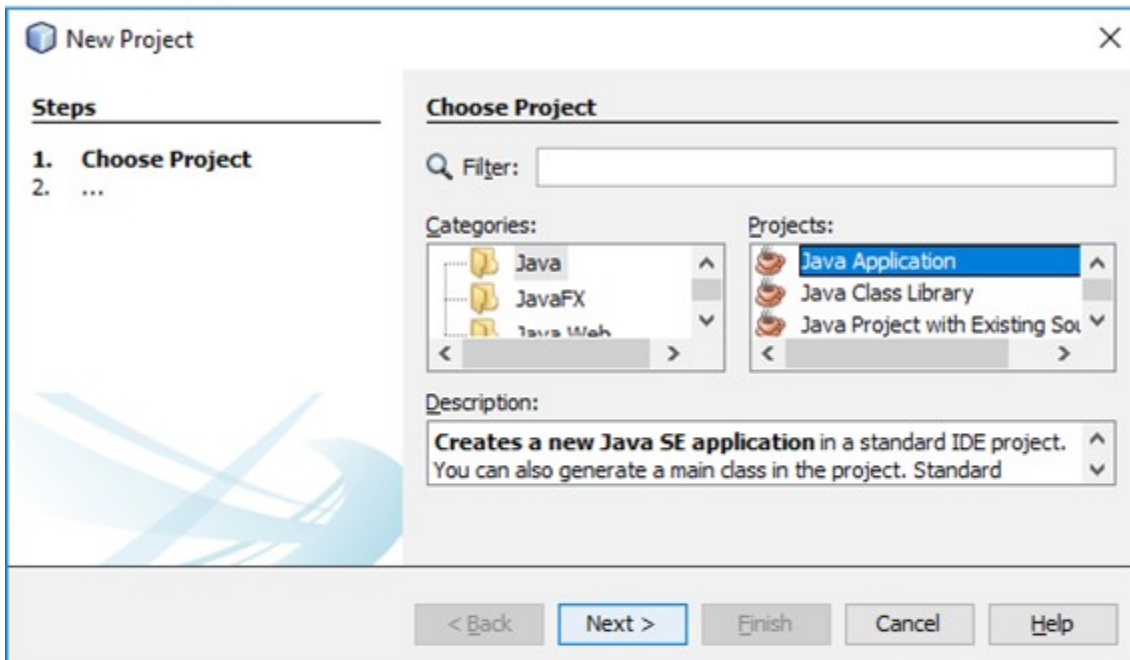


5. Creación del proyecto

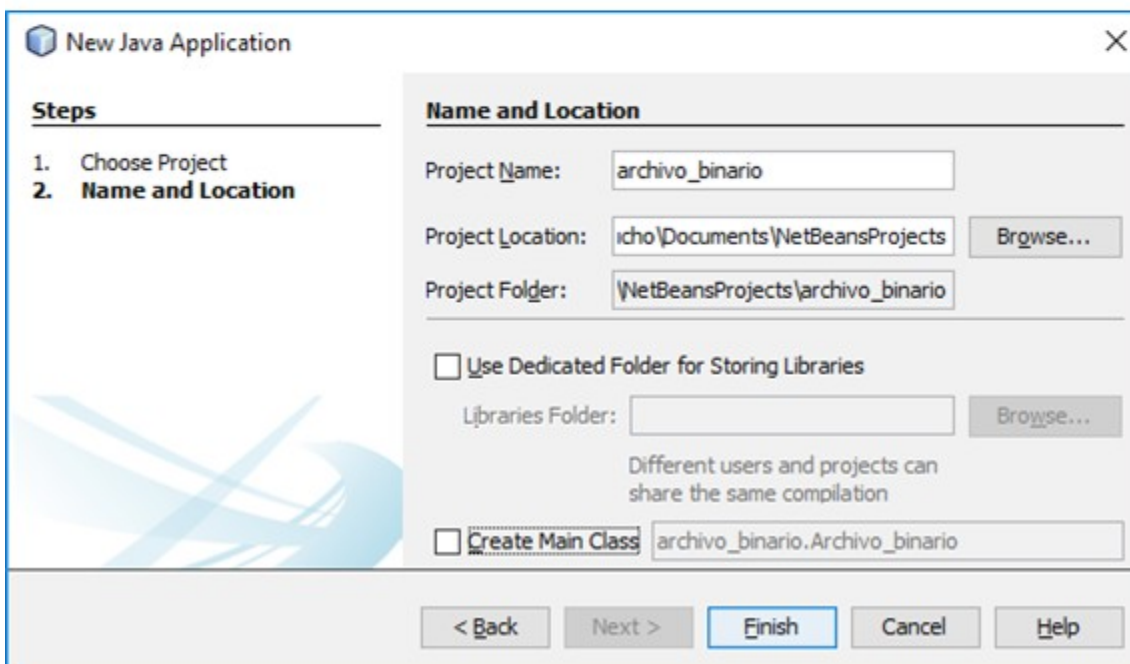
a. Entre NetBeans y cree un nuevo proyecto:



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

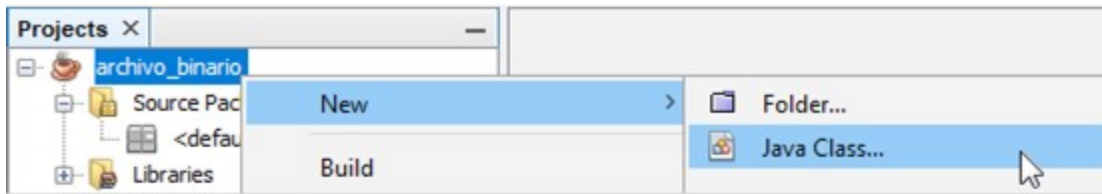


c. Ponga **archivo_binario** en **Project Name** y desmarque **Create Main Class**.

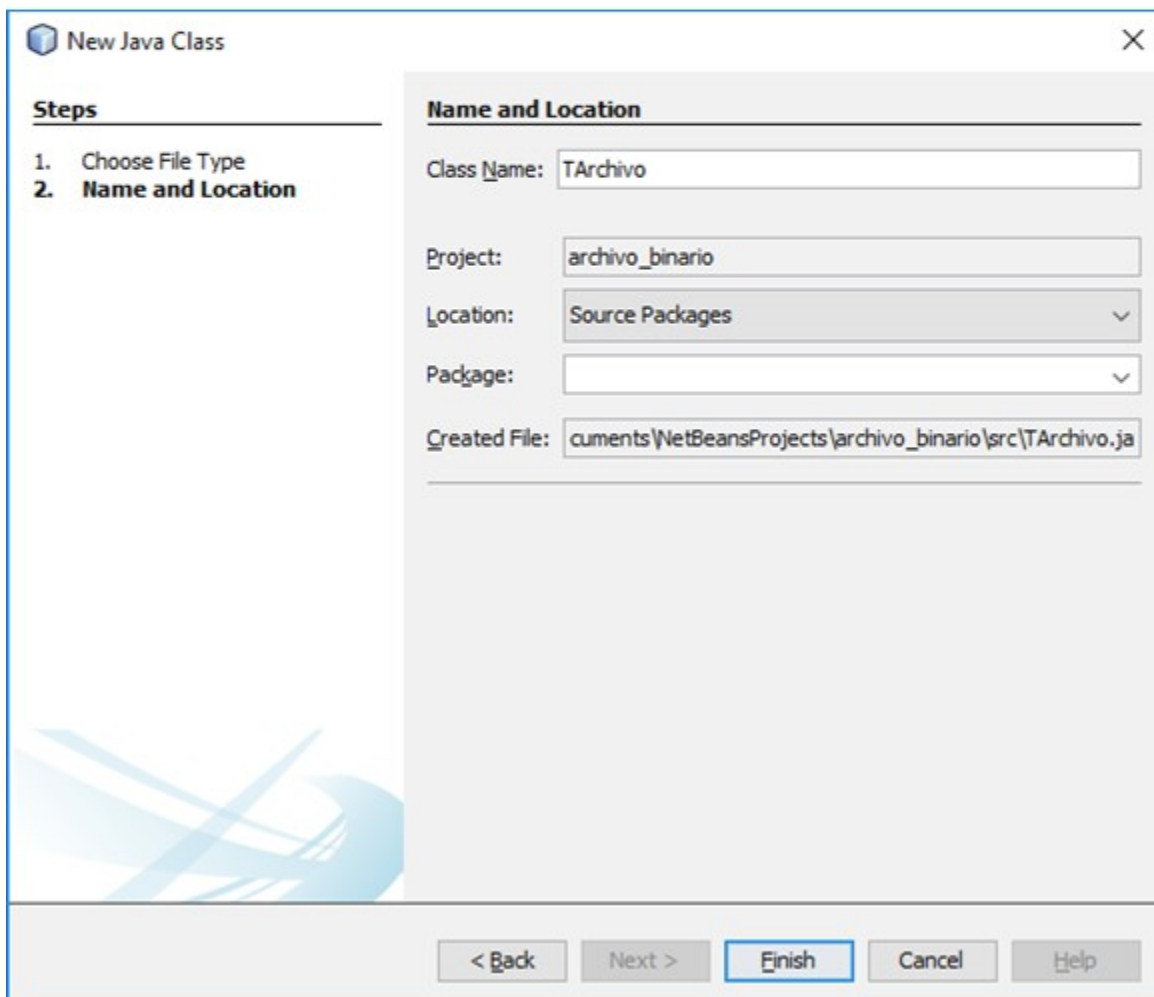


6. Creación archivo clase *TArchivo*

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

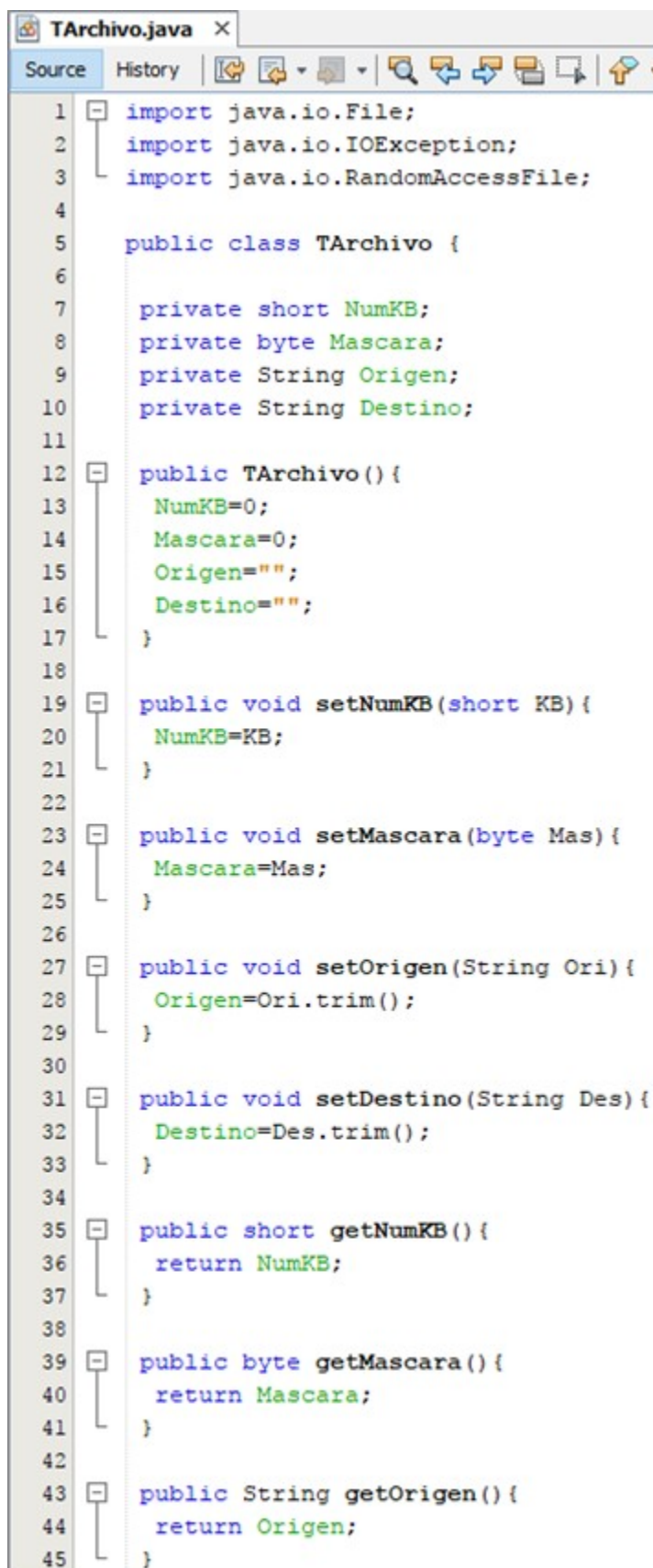


b. En la ventana desplegada asigne ***TArchivo*** por nombre y luego haga click en el botón ***Finish*** para crear el archivo java correspondiente como se aprecia en la siguiente imagen.



c. Para implementar esta clase asegúrese que el código quede como sigue:

Luis Roberto Olascoaga Surmay



```
1  import java.io.File;
2  import java.io.IOException;
3  import java.io.RandomAccessFile;
4
5  public class TArchivo {
6
7      private short NumKB;
8      private byte Mascara;
9      private String Origen;
10     private String Destino;
11
12     public TArchivo() {
13         NumKB=0;
14         Mascara=0;
15         Origen="";
16         Destino="";
17     }
18
19     public void setNumKB(short KB) {
20         NumKB=KB;
21     }
22
23     public void setMascara(byte Mas) {
24         Mascara=Mas;
25     }
26
27     public void setOrigen(String Ori) {
28         Origen=Ori.trim();
29     }
30
31     public void setDestino(String Des) {
32         Destino=Des.trim();
33     }
34
35     public short getNumKB() {
36         return NumKB;
37     }
38
39     public byte getMascara() {
40         return Mascara;
41     }
42
43     public String getOrigen() {
44         return Origen;
45     }
```

Luis Roberto Olascoaga Surmay

```

46
47 public String getDestino(){
48     return Destino;
49 }
50
51 protected boolean EsFinal(RandomAccessFile RAF){
52     try{
53         return RAF.getFilePointer() >= RAF.length();
54     }
55     catch(IOException Err){
56         Err.printStackTrace();
57         return true;
58     }
59 }
60
61 protected int Leer(byte Buf[], RandomAccessFile RAF){
62     try{
63         return RAF.read(Buf);
64     }
65     catch(IOException Err){
66         Err.printStackTrace();
67         return -1;
68     }
69 }
70
71 protected boolean Escribir(int Tam, byte Buf[], RandomAccessFile RAF){
72     try{
73         RAF.write(Buf, 0, Tam);
74         return true;
75     }
76     catch(IOException Err){
77         Err.printStackTrace();
78         return false;
79     }
80 }
81
82 public void Cerrar(RandomAccessFile RAF){
83     try{
84         RAF.close();
85     }
86     catch(IOException Err){
87         Err.printStackTrace();
88     }
89 }
90

```

```

91 public RandomAccessFile Abrir(String Nombre){
92     RandomAccessFile RAF;
93     try{
94         RAF=new RandomAccessFile(Nombre,"rw");
95         RAF.seek(0);
96     }
97     catch(IOException Err){
98         RAF=null;
99         Err.printStackTrace();
100     }
101     return RAF;
102 }
103
104 public RandomAccessFile Crear(String Nombre){
105     RandomAccessFile RAF=Abrir(Nombre);
106     try{
107         RAF.setLength(0);
108     }
109     catch(IOException Err){
110         RAF=null;
111         Err.printStackTrace();
112     }
113     return RAF;
114 }
115
116 protected void Encriptar(int Tam,byte Buf[]){
117     int i;
118     for(i=0;i<Tam;i++){
119         Buf[i]=(byte) (Buf[i] ^ Mascara);
120     }
121 }
122
123 public void Encriptar(){
124     int Leidos;
125     byte Buff[];
126     RandomAccessFile RO,RD;
127     Buff=new byte[1024];
128     RO=Abrir(Origen);
129     RD=Crear(Destino);
130     do{
131         Leidos=Leer(Buff,RO);
132         if(Leidos>0){
133             Encriptar(Leidos,Buff);
134             if(!Escribir(Leidos,Buff,RD))
135                 break;
136         }

```

```

137     }while (Leidos>0);
138     Cerrar (RO);
139     Cerrar (RD);
140 }
141
142 public void Copiar() {
143     int Leidos;
144     byte Buff[];
145     RandomAccessFile RO,RD;
146     Buff=new byte[1024];
147     RO=Abrir (Origen);
148     RD=Crear (Destino);
149     do{
150         Leidos=Leer (Buff,RO);
151         if (Leidos>0) {
152             if (!Escribir (Leidos,Buff,RD))
153                 break;
154         }
155     }while (Leidos>0);
156     Cerrar (RO);
157     Cerrar (RD);
158 }
159
160 public void Mover() {
161     Copiar();
162     new File (Origen).delete();
163 }
164
165 public void Partir() {
166     byte Buff[];
167     int i,k,Leidos;
168     String Dest;
169     RandomAccessFile RO,RD;
170     k=1;
171     Leidos=1;
172     Buff=new byte[1024];
173     RO=Abrir (Origen);
174     Dest=Destino.replace(".", "%d.");
175     while (Leidos>0 && !EsFinal (RO)) {
176         RD=Crear (String.format (Dest,k));
177         for (i=1;i<=NumKB;i++) {
178             Leidos=Leer (Buff,RO);
179             if (Leidos>0) {
180                 if (!Escribir (Leidos,Buff,RD))
181                     break;
182             }

```

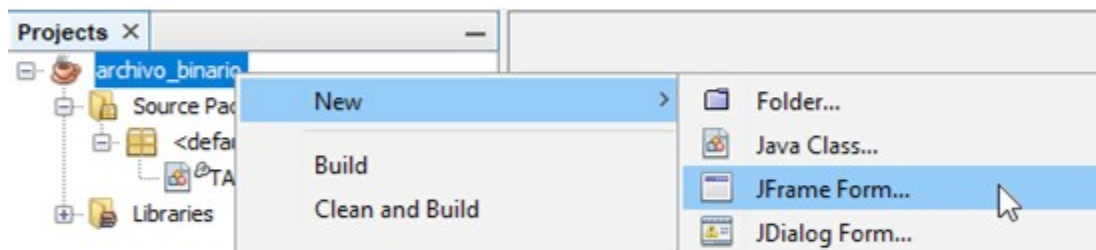
```

183     }
184     k++;
185     Cerrar (RD) ;
186 }
187 Cerrar (RO) ;
188 }
189
190 }
191

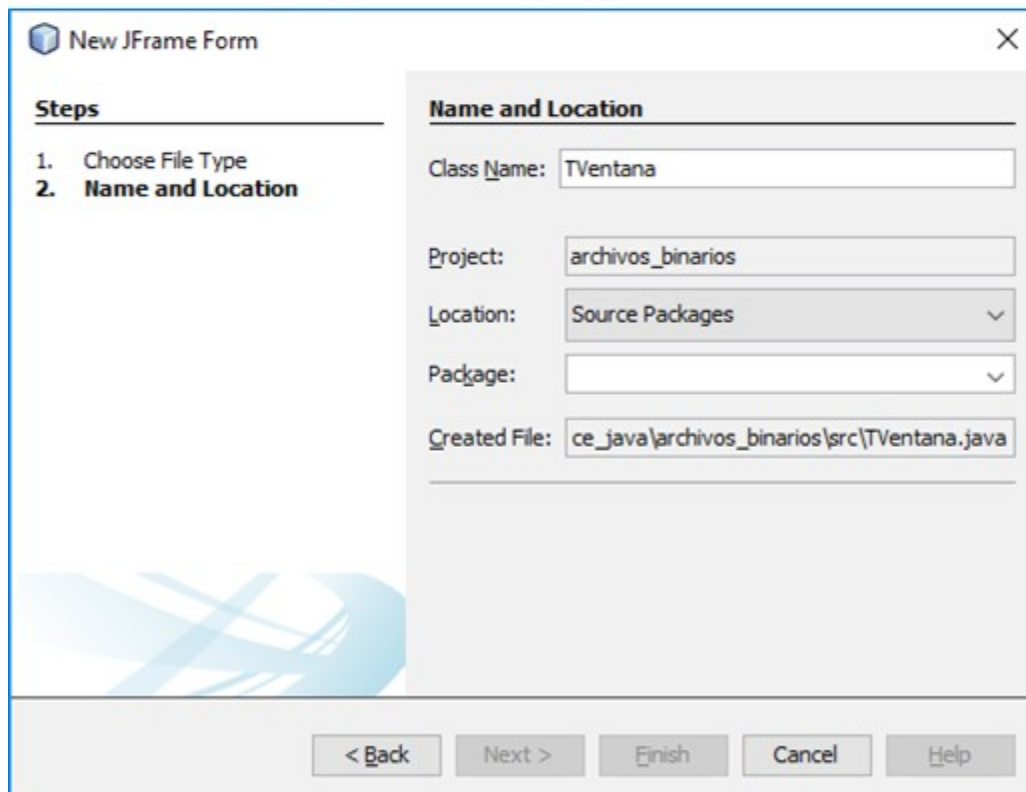
```

7. Diseño de la ventana principal.

a. Haga click derecho en el proyecto **archivo_binario** y tome las opciones **New + Java Class**.



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



c. Diseñe la ventana como sigue, en la cual los cuatro **JTextField** tienen por nombre de variable en su orden (de arriba hacia abajo) **tf1**, **tf2**, **tf3** y **tf4** respectivamente. Al **JComboBox** póngale **cb1** por nombre y en su propiedad **model** añada las opciones: **Encriptar**, **Copiar**, **Mover** y **Partir** respectivamente. Vaya al nodo Swing Windows de la paleta de componentes, arrastre y deje caer (POR FUERA DE LA VENTANA) un componente **JFileChooser**, al cual debe hacerle click derecho y cambiarle el nombre de variable por **Dlg**. Finalmente seleccione el **JForm** y póngale por título **Ejemplo archivos binarios**; además asegúrese de que la ventana aparezca centrada en tiempo de ejecución. De este modo la ventana debe tener una apariencia similar a la presentada en la siguiente imagen:

8. Implementación de métodos de la ventana

a. Vaya la vista de fuente de la ventana y al principio del archivo y haga la importación de la clase indicada:

```
import javax.swing.JOptionPane;
```

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

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    Dlg.setDialogTitle("Abrir archivo origen");
    if (Dlg.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
        tf1.setText(Dlg.getSelectedFile().getAbsolutePath());
    }
}
```

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

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {  
    Dlg.setDialogTitle("Guardar archivo destino como");  
    if(Dlg.showSaveDialog(null)==JFileChooser.APPROVE_OPTION) {  
        tf2.setText(Dlg.getSelectedFile().getAbsolutePath());  
    }  
}
```

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

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {  
    TArchivo AB=new TArchivo();  
    AB.setOrigen(tf1.getText());  
    AB.setDestino(tf2.getText());  
    switch(cb1.getSelectedIndex()) {  
        case 0:AB.setMascara(Byte.parseByte(tf3.getText()));  
                AB.Encriptar();  
                break;  
        case 1:AB.Copiar();  
                break;  
        case 2:AB.Mover();  
                break;  
        case 3:AB.setNumKB(Short.parseShort(tf4.getText()));  
                AB.Partir();  
                break;  
    }  
    JOptionPane.showMessageDialog(null,cb1.getSelectedItem() + " finalizada");  
}
```

Actividad propuesta.

Como complemento del anterior ejercicio desarrolle las siguientes actividades.

1. Modifique el ejemplo presentado de modo que todas las operaciones sean realizadas con un hilo.
2. Modifique el ejemplo presentado de modo que incluya la operación para pegar los archivos divididos.
3. Diseñe e implemente una clase que tenga una propiedad para el nombre de un archivo y que además tenga un método que determine si el archivo indicado es un ejecutable bajo el formato ELF.
4. Diseñe e implemente las clases necesarias para desarrollar el mismo ejemplo de archivos de texto presentado en clase, pero esta vez usando archivos binarios.

Luis Roberto Olascoaga Surmay

Luis Roberto Olascoaga Surmay