Universidad Autónoma del Estado de México

**Centro Universitario UAEM Atlacomulco**

**“Archivos de acceso aleatorio”**

**Presenta:**

**Segundo Antonio Elías Edgardo**

**Unidad de Aprendizaje:**

**Organización de archivos**

**Fecha de entrega:**

**29 de marzo de 2018**

# Objetivo

En esta sección se analizará la manera de diseñar rutinas que manipulen registros de productos o artículos en un archivo directo. A diferencia del archivo secuencial en el que se hace un recorrido secuencial para localizar la dirección del registro solicitado, en el archivo de acceso directo, se calcula la dirección física y se posiciona el apuntador del archivo directamente en el registro solicitado usando la función fseek.

En esta practica se tendrá como objetivo desarrollar un programa en Java que integre una UI con diversos campos para ingresar datos en un archivo, sin embargo, la organización del archivo tendrá que seguir las reglas descritas en el párrafo anterior.

# Resumen

En esta práctica se realizará una aplicación Java FX que contendrá todos los procedimientos de tratamiento de los archivos directos, se programaran funciones para las opciones de altas, bajas, edición, búsqueda y muestra completa del archivo en cuestión, además de eso se diseñara una GUI utilizando los componentes de la librería JFoenix para dar una mejor imagen al programa y apegar el diseño al estándar Material Design.

El programa en su totalidad fue desarrollado sobre el IDE Intellij Idea y la GUI se desarrolló en FXML con ayuda del Scene Builder de Gluon, para guardar los elementos en el archivo se utilizó una clase que representa el objeto a guardar en cuestión.

En este caso el programa corresponde a una aplicación para guardar datos de pedidos, incluyendo los siguientes datos en el objeto Product:

|  |  |
| --- | --- |
| Nombre | Tipo de dato |
| Id | String |
| Name | String |
| Model | String |
| Position | Integer |
| Date | LocalDate |
| Description | String |
| Original | Boolean |

Con estos datos se diseñaron los campos de la Interfaz y los tamaños en bytes para el archivo.

# Marco teórico

La clase Java RandomAccessFile se utiliza para acceder a un fichero de forma aleatoria.

Los constructores de la clase son:

RandomAccessFile(String path, String modo);

RandomAccessFile(File objetoFile, String modo);

Lanzan una excepción **FileNotFoundException**.

El argumento ***modo*** indica el modo de acceso en el que se abre el fichero.

Los valores permitidos para este parámetro son:

|  |  |
| --- | --- |
| modo | Significado |
| "r" | Abre el fichero en **modo solo lectura**.  El fichero debe existir.  Una operación de escritura en este fichero lanzará una excepción IOException. |
| "rw" | Abre el fichero en **modo lectura y escritura**. Si el fichero no existe se crea. |

**Ejemplo**: abrir un fichero aleatorio para lectura

Se abre el fichero clientes.dat para lectura usando el primer constructor.

RandomAccessFile fichero = new RandomAccessFile("/ficheros/clientes.dat", "r");

**Ejemplo**: abrir un fichero aleatorio para lectura/escritura

Se abre el fichero personas.dat para lectura/escritura usando el segundo constructor. Si el fichero no existe se crea.

File f = new File ("/ficheros/personas.dat");

RandomAccessFile fichero = new RandomAccessFile(f, "rw");

**ACCESO A LOS DATOS EN FICHEROS ALEATORIOS**

Para acceder de forma aleatoria a los datos contenidos en el fichero, la clase RandomAccessFile dispone de varios métodos. Entre ellos:

**long getFilePointer();**

Devuelve la posición actual del puntero del fichero. Indica la posición (en bytes) donde se va a leer o escribir.

**long length();**

Devuelve la longitud del fichero en bytes.

**void seek(long pos);**

Coloca el puntero del fichero en una posición *pos* determinada. La posición se da como un desplazamiento en bytes desde el comienzo del fichero. La posición 0 indica el principio del fichero. La posición *length()* indica el final del fichero.

Además, dispone de métodos de lectura/escritura:

**public int read()**

Devuelve el byte leído en la posición marcada por el puntero. Devuelve -1 si alcanza el final del fichero. Se debe utilizar este método para leer los caracteres de un fichero de texto.

**public final String readLine()**

Devuelve la cadena de caracteres que se lee, desde la posición marcada por el puntero, hasta el siguiente salto de línea que se encuentre.

**public xxx readXxx()**

Hay un método read para cada tipo de dato básico: **readChar**, **readInt**, **readDouble**, **readBoolean, etc**.

**public void write(int b)**

Escribe en el fichero el byte indicado por parámetro. Se debe utilizar este método para escribir caracteres en un fichero de texto.

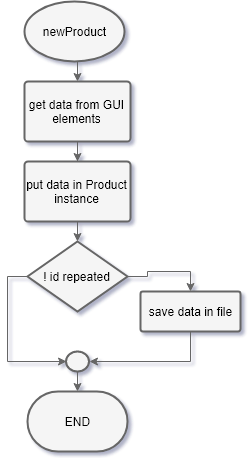
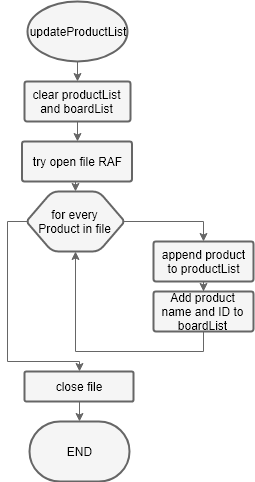
**public final void writeBytes(String s)**

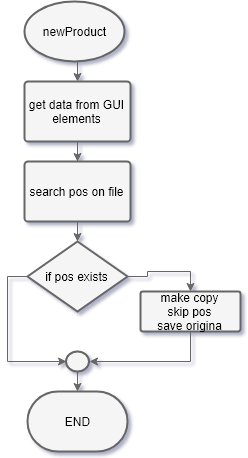
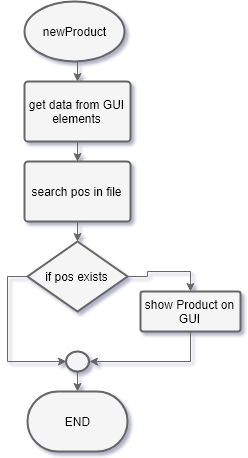
Escribe en el fichero la cadena de caracteres indicada por parámetro.

**public final void writeXxx(*argumento*)**

También existe un método write para cada tipo de dato básico: **writeChar**, **writeInt**, **writeDouble**, **writeBoolean, etc**.

# Desarrollo





|  |
| --- |
| Controller.java |
| package sample;  import com.jfoenix.controls.\*; import de.jensd.fx.glyphs.materialdesignicons.MaterialDesignIconView; import javafx.fxml.FXML; import javafx.scene.control.Label;  import java.io.File; import java.io.IOException; import java.io.RandomAccessFile; import java.time.LocalDate; import java.util.\*;  public class Controller {  @FXML MaterialDesignIconView closeIcon;  @FXML JFXListView<JFXButton> boardList = new JFXListView<>();   @FXML JFXTextField nameField;  @FXML JFXTextField modelField;  @FXML JFXTextField positionField;  @FXML JFXTextArea descriptionArea;  @FXML JFXTextField idField;  @FXML JFXToggleButton originalToggle;  @FXML JFXDatePicker orderDate;  @FXML Label warningDialog;    @FXML JFXButton saveButton;  @FXML JFXButton updateButton;  @FXML JFXButton readButton;  @FXML JFXButton editButton;  @FXML JFXButton eraseButton;      private Integer position;   List<Product> productList= new ArrayList<>();    public void closeApp()  {  System.*exit*(0);  }    public String fillString(String x,Integer size)  {  while(x.length()<size)  x+="|";  return x;  }   public String chop(String x)  {  String chopped="";  int i=0;  while(x.charAt(i)!='|')  chopped+=x.charAt(i++);  return chopped;  }   public void updateProductList()  {  try{  productList.clear();  boardList.getItems().clear();  Product aux1= new Product();  RandomAccessFile file= new RandomAccessFile("products.txt","r");  for(Integer i=0;i<file.length()/aux1.getProductSize();i++)  {  Product aux= new Product();  //Set pos  aux.setPosition(i);   //Set ID  byte[] buffer= new byte[aux.getIdSize()];  file.read(buffer);  aux.setId(Integer.*parseInt*( chop(new String(buffer))) );   //Set name  buffer= new byte[aux.getNameSize()];  file.read(buffer);  aux.setName( chop(new String(buffer)) );   //Set model  buffer= new byte[aux.getModelSize()];  file.read(buffer);  aux.setModel( chop(new String(buffer)) );   //Set original bool value  buffer= new byte[aux.getOriginalSize()];  file.read(buffer);  if(new String(buffer)=="1"){  aux.setOriginal(true);  } else aux.setOriginal(false);   //Set description field  buffer = new byte[aux.getDescriptionSize()];  file.read(buffer);  aux.setDescription( chop(new String(buffer)) );   //Set date  buffer= new byte[aux.getDateSize()];  file.read(buffer);  LocalDate ld= LocalDate.*parse*(new String(buffer));  aux.setDate(ld);   productList.add(aux);  boardList.getItems().  add(new JFXButton(i.toString()+"-"+aux.getName() +"--ID: "+  aux.getId().toString()));   }  file.close();  } catch(IOException e){e.printStackTrace();}  }   public void newProduct()  {  String strOriginal="0";  String name= nameField.getText();  String model= modelField.getText();  String description=descriptionArea.getText();  String id=idField.getText();  Boolean original= originalToggle.isSelected();  LocalDate date= orderDate.getValue();  Boolean noIdProblem=true;  for(Product x: productList)  {  if(x.id==Integer.*parseInt*(id)) noIdProblem=false;  }  if(noIdProblem) {  warningDialog.setVisible(false);  name = fillString(name, 30);  model = fillString(model, 30);  description = fillString(description, 100);  id = fillString(id, 10);  String strDate = date.toString();  if (strDate.length() == 0) strDate = LocalDate.*now*().toString();  if (original) strOriginal = "1";   try {  RandomAccessFile file = new RandomAccessFile("products.txt", "rw");  long fileLength = file.length();  file.seek(fileLength);   System.*out*.println(fileLength);   file.writeBytes(id);  file.writeBytes(name);  file.writeBytes(model);  file.writeBytes(strOriginal);  file.writeBytes(description);  file.writeBytes(strDate);   file.close();  } catch (IOException e) {  e.printStackTrace();  }  }  else  {  warningDialog.setVisible(true);  }  }   public void updateProduct()  {   String strOriginal="0";  String name= nameField.getText();  String model= modelField.getText();  String description=descriptionArea.getText();  String id=idField.getText();  Boolean original= originalToggle.isSelected();  LocalDate date= orderDate.getValue();  Boolean noIdProblem=true;  Product aux= new Product();    name = fillString(name, 30);  model = fillString(model, 30);  description = fillString(description, 100);  id = fillString(id, 10);  String strDate = date.toString();  if (strDate.length() == 0) strDate = LocalDate.*now*().toString();  if (original) strOriginal = "1";   position=Integer.*parseInt*(positionField.getText());  Integer actualId=productList.get(position).getId();  System.*out*.println(actualId);  for(Product x: productList)  {  if(x.id==Integer.*parseInt*(chop(id))) noIdProblem=false;  if(x.id==actualId)noIdProblem=true;  System.*out*.println(x.name);  System.*out*.println(x.id);  }   if(noIdProblem) {  warningDialog.setVisible(false);  try {  RandomAccessFile file = new RandomAccessFile("products.txt", "rw");  file.seek(position \* aux.getProductSize());  file.writeBytes(id);  file.writeBytes(name);  file.writeBytes(model);  file.writeBytes(strOriginal);  file.writeBytes(description);  file.writeBytes(strDate);  file.close();  updateProductList();  } catch (IOException e) {  e.printStackTrace();  }  } else warningDialog.setVisible(true);   }   public void eraseProduct()  {  Integer pos= Integer.*parseInt*(positionField.getText());  try  {  RandomAccessFile file= new RandomAccessFile("products.txt","rw");  RandomAccessFile backup= new RandomAccessFile("back.up","rw");  Product aux= new Product();  for(Integer i=0;i<file.length();i++) {  if (file.getFilePointer() / aux.getProductSize() == pos) {  file.seek(file.getFilePointer() + aux.getProductSize());  i = i + aux.getProductSize()-1;  } else  backup.write(file.read());  }  file.close();  backup.close();   File rawFile= new File("products.txt");  rawFile.delete();   file= new RandomAccessFile("products.txt","rw");  backup= new RandomAccessFile("back.up","rw");   for(Integer i=0;i<backup.length();i++)  file.write(backup.read());  file.close();  backup.close();   rawFile= new File("back.up");  rawFile.delete();   } catch (IOException e){e.printStackTrace();}  }   public void showProduct()  {  position=Integer.*parseInt*(positionField.getText());  try  {  Product aux= new Product();  RandomAccessFile file= new  RandomAccessFile("products.txt","r");   Long pos= Long.*parseLong*(positionField.getText());  file.seek(pos\*aux.getProductSize());   //Set ID in field  byte[] buffer= new byte[aux.getIdSize()];  file.read(buffer);  aux.setId(Integer.*parseInt*( chop(new String(buffer))) );   //Set name in field  buffer= new byte[aux.getNameSize()];  file.read(buffer);  aux.setName( chop(new String(buffer)) );   //Set model in field  buffer= new byte[aux.getModelSize()];  file.read(buffer);  aux.setModel( chop(new String(buffer)) );   //Set original bool value in toggle  buffer= new byte[aux.getOriginalSize()];  file.read(buffer);  String x=new String(buffer);  if(x.compareTo("1")==0){  aux.setOriginal(true);  } else aux.setOriginal(false);   //Set description field  buffer = new byte[aux.getDescriptionSize()];  file.read(buffer);  aux.setDescription( chop(new String(buffer)) );   //Set date in datePicker  buffer= new byte[aux.getDateSize()];  file.read(buffer);  LocalDate ld= LocalDate.*parse*(new String(buffer));  aux.setDate(ld);   idField.setText(aux.getId().toString());  orderDate.setValue(aux.getDate());  nameField.setText(aux.getName());  modelField.setText(aux.getModel());  descriptionArea.setText(aux.getDescription());  if(aux.getOriginal()==true)  {  originalToggle.setSelected(true);  } else originalToggle.setSelected(false);   file.close();  } catch (IOException e){e.printStackTrace();}  }   public void initialize()  {  updateProductList();  }    public class Product {  Integer id;  Integer position;  LocalDate date;  String name;  String description;  String model;  Boolean original;   public Product(Integer id, Integer position, LocalDate date, String name,  String description, String model, Boolean original) {  this.id = id;  this.position = position;  this.date = date;  this.name = name;  this.description = description;  this.model = model;  this.original = original;  }   public Product(){}   public Integer getProductSize(){return 181;}  public Integer getIdSize() {return 10;}  public Integer getNameSize() { return 30;}  public Integer getModelSize(){return 30;}  public Integer getOriginalSize(){return 1;}  public Integer getDescriptionSize(){return 100;}  public Integer getDateSize(){return LocalDate.*now*().toString().length();}   public Integer getId() {  return id;  }   public void setId(Integer id) {  this.id = id;  }   public Integer getPosition() {  return position;  }   public void setPosition(Integer position) {  this.position = position;  }   public LocalDate getDate() {  return date;  }   public void setDate(LocalDate date) {  this.date = date;  }   public String getName() {  return name;  }   public void setName(String name) {  this.name = name;  }   public String getDescription() {  return description;  }   public void setDescription(String description) {  this.description = description;  }   public String getModel() {  return model;  }   public void setModel(String model) {  this.model = model;  }   public Boolean getOriginal() {  return original;  }   public void setOriginal(Boolean original) {  this.original = original;  }  }  } |

# Resultados

Capturas de ejecución

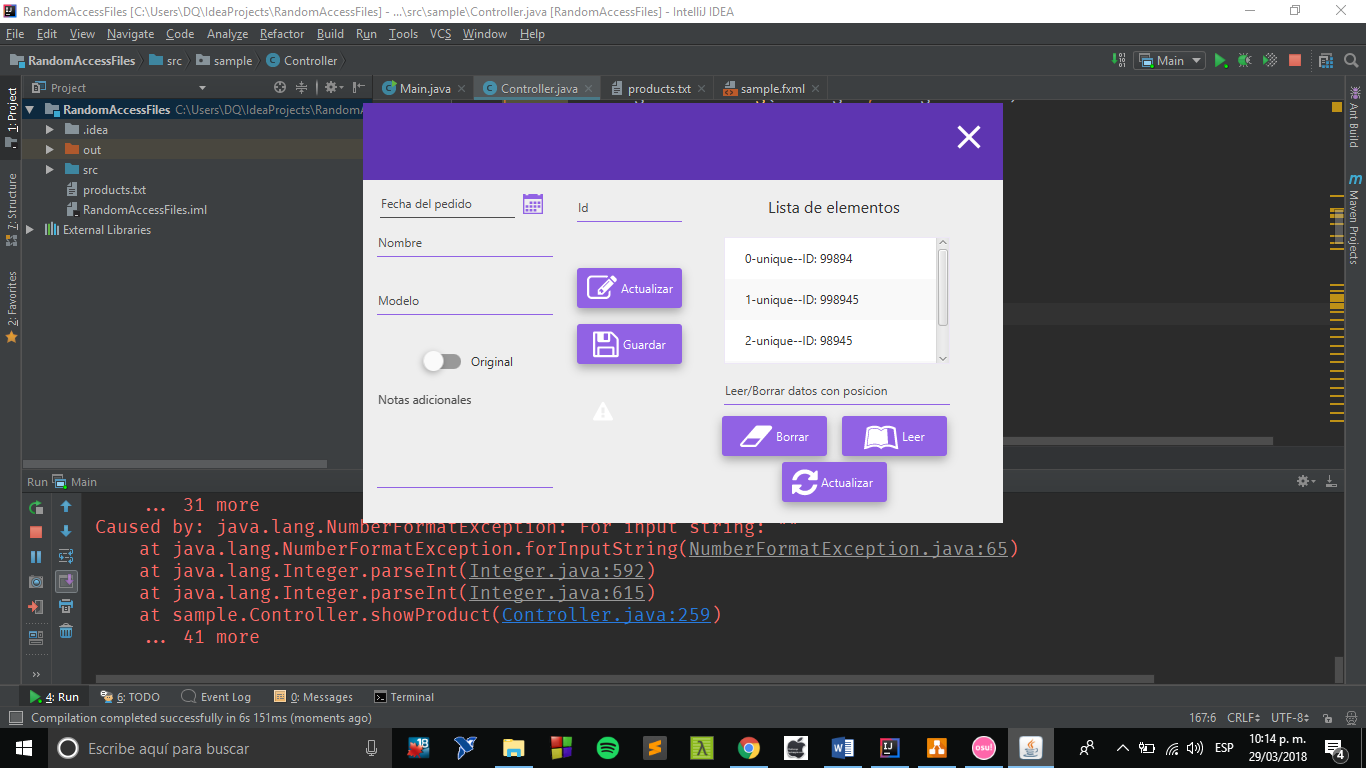


Ilustración 1 GUI del programa, se puede ver como el diseño material design es familiar e intuitivo

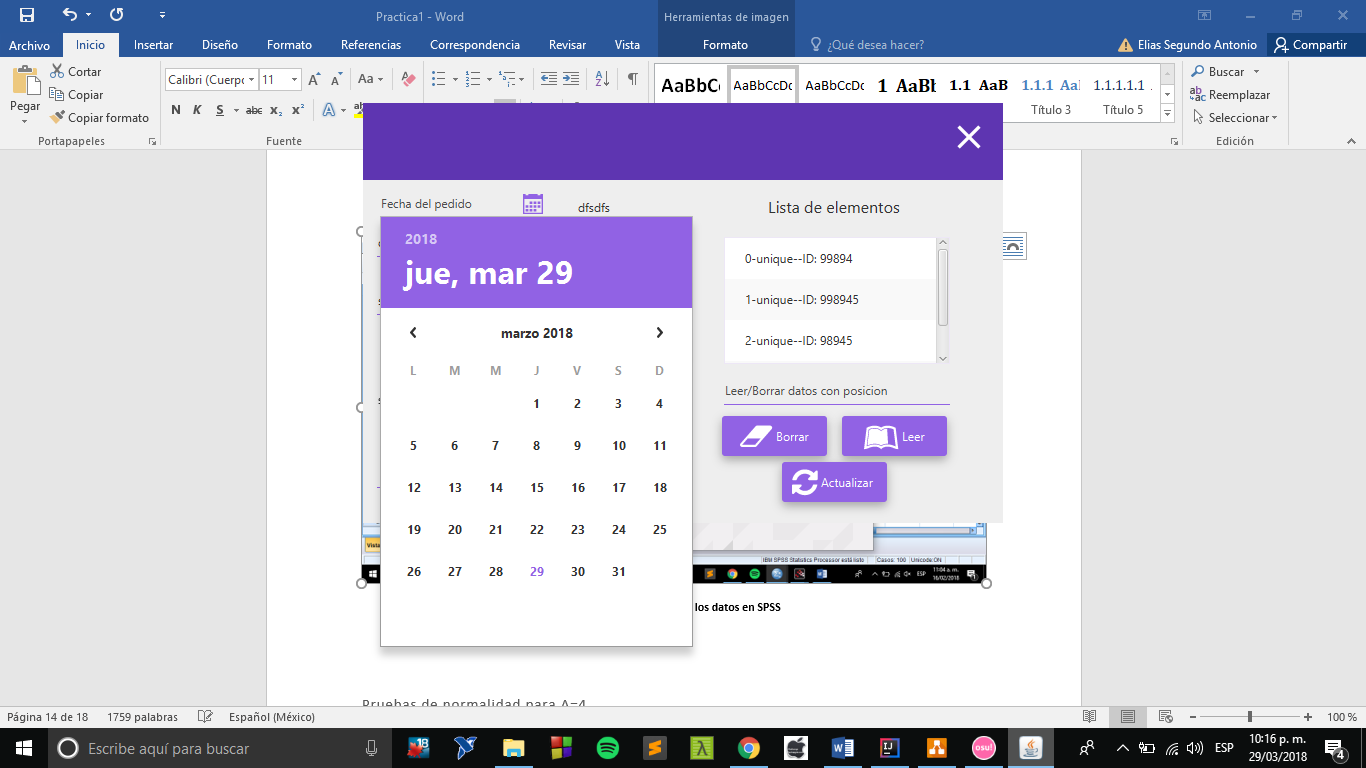


Ilustración 2 Inserción de los datos, se pueden incorporar diversos datos; i.e un LocalDate

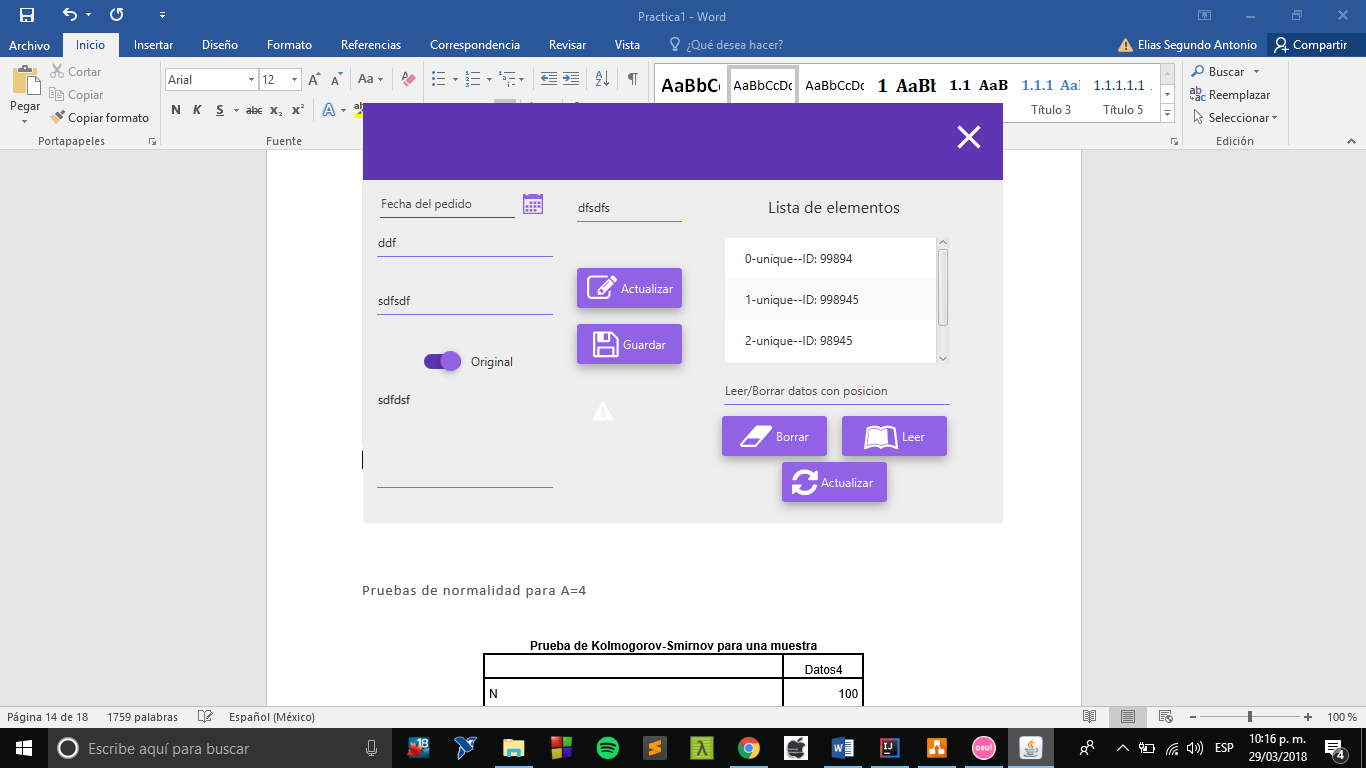


Ilustración 3 La tabla muestra los registros automáticamente llenando los campos con los datos

# Cuestionario

a) ¿Qué son los archivos directos?

Archivos en los que es posible saltar ciertas posiciones de un archivo para acceder directamente al valor deseado, esto debido a que existen funciones que pueden desplazar el apuntador de lectura cierta cantidad de bits a partir de un punto de referencia.

b) ¿Cuál es su principal característica de los archivos directos?

El apuntador de lectura o escritura del archivo de puede mover a voluntad.

c) ¿Qué es lo principal (código, palabra reservada, procedimiento) de una rutina de

altas?

En el caso de java es la función writeBytes de la clase RandomAccessFile, es la que permite escribir la nueva información en el archivo.

d) ¿Qué es lo principal (código, palabra reservada, procedimiento) de una rutina de

consultas?

La función seek y read de la clase RandomAccessFile, la que permite mover el apuntador de lectura para consultar la posición deseada.

e) ¿Qué es lo principal (código, palabra reservada, procedimiento) de una rutina de

listado?

La función read de la clase RandomAccessFile que en conjunto con un ciclo permite leer y mostrar todos los registros del archivo.

f) ¿Qué es lo principal (código, palabra reservada, procedimiento) de una rutina de

modificaciones?

La función seek que permite mover el apuntador al registro a modificar y después sobrescribir con writeBytes.

g) ¿Qué es lo principal (código, palabra reservada, procedimiento) de una rutina de

bajas?

La búsqueda con apuntador para modificar el archivo etiquetando el registro a dar de baja.

# Debate y conclusión

Los archivos directos presentan una ventaja muy clara frente a los archivos secuenciales, la arbitrariedad con la que se puede acceder a los registros que lo componen, sin embargo, esta ventaja presenta algunas limitantes dado que el archivo debe de tratarse en forma binaria siendo imposible recuperar los datos directamente del archivo sin tener un proceso de unpacking de intermedio.

Adicionalmente se debe de tener un control muy estricto de los tamaños en bytes del archivo a modificar dado que la función sizeof de C++ brilla por su ausencia en Java, además de que al ser los tipos de datos una clase en lugar de un tipo nativo se debe de extraer específicamente los datos en bytes que representan a la variable y tener cuidado de no obtener el tamaño o los datos del objeto completo.

# Bibliografía

|  |  |
| --- | --- |
| [1] | M. Lomis, Estructuras de datos y organizacion de archivos, Prentice Hall, 1991. |
| [2] | B. Garcia, Programacion en manejo de objetos, Mexico: Macrobit, 2006. |

# 