



Reporte de proyecto final

Universidad Nacional Autónoma de México

Facultad de ingeniería

Laboratorios de computación salas A y B

Profesor: Dra. Rocío Alejandra Aldeco Pérez

Asignatura: Programación Orientada a Objetos – 1323

Grupo: 6

Integrante(s): Chong Hernández Samuel, López Campillo Francisco Daniel
Mendoza Hernández Carlos Emiliano, Roa Díaz Vanessa

No. de Lista o Brigada:

Semestre: 2023-1

Fecha de entrega: 13/01/2023

Observaciones:

CALIFICACIÓN: _____



Proyecto final

INSTRUCCIONES

- Creación de la clase principal: EjecutaWAV.
- Implementación de la clase para generar el archivo de audio: GeneraWAV.
- Inclusión de la clase de prueba: EscuchaWAV.

OBJETIVOS

Implementar un programa en el lenguaje de programación Java que genere sonido de forma estándar en cualquier reproductor haciendo uso de un archivo de audio en formato WAV y de las propiedades del audio definidas en un archivo de texto plano.

INTRODUCCIÓN

La programación orientada a objetos es un paradigma que en la actualidad es de los más usados por su versatilidad, pues el desarrollo de software en este entorno cuenta con beneficios notables como lo son el hecho de que el código sea organizado, reutilizable y cuenta con mayor facilidad para poder proporcionarle mantenimiento.

Una de las más grandes ventajas que diferencia por mucho al paradigma orientado a objetos respecto a otras formas de programar es que protege los datos e información empleada de acceso indeseado, es decir, evita la exposición de código, esto a través de una herramienta fundamental para este tipo de programación a la que se le denomina encapsulamiento.



PROGRAMACIÓN ORIENTADA A OBJETOS



Como se indica, la programación orientada a objetos es uno de los marcos más vigorosos bajo los que se desarrolla software además de ser el más cercano a como los seres humanos percibimos nuestra realidad (a través de objetos), por lo cual se ha elegido este paradigma para dirigir y poner en marcha la resolución al problema planteado, no sólo por sus virtuosas propiedades, sino que también y principalmente por la naturaleza del problema a resolver.

El proyecto que estamos tratando en esta ocasión plantea el problema central sobre la implementación de un programa en el lenguaje de programación Java que a partir de varios parámetros puede generar un archivo de audio con formato WAV de manera que sea posible reproducirlo en cualquier reproductor.

Particularmente el desarrollo del programa como solución al problema planteado, trata procesos de interpretación y desincronización de bits almacenados en un archivo para generar un archivo de audio en formato WAV con la información del audio entre la cual se encuentra el tamaño del archivo, el número de canales, la frecuencia de muestreo, entre otros.

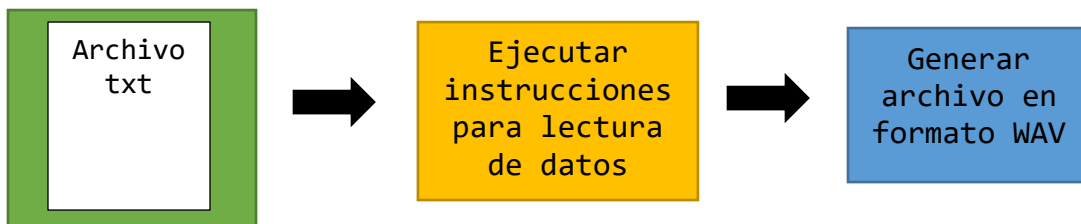
Para el caso de las pruebas de este programa se emplea un sonido senoidal monoaural, es decir, que sólo se define con el uso de un canal, el cual corresponde con la nota "La".

Es importante destacar que el desarrollo de este programa, además de permitir aplicar conceptos esenciales en la programación orientada a objetos como lo es la herencia y polimorfismo, propicia el empleo de archivos, pues se hace uso de archivos de entrada y también de salida, que en este caso es el archivo en formato WAV. El uso de este tipo de archivos puede tener desventajas respecto a otros, como lo es el tamaño, pues a comparación de los formatos MP3 y MP4, presenta mayor tamaño, sin embargo, la gran ventaja que posee el formato WAV es obtener una mejora en la calidad del sonido, por lo que, por efectos de los requerimientos en el problema planteado, es factible su manejo.



DESARROLLO

La resolución al problema planteado en este proyecto, es decir, la implementación del programa que genera un archivo en formato WAV usando las propiedades del archivo de audio (proporcionadas en un archivo de texto plano) como parámetros, se divide en dos procesos principales: ejecutar las instrucciones para la lectura de datos que indican las características elementales para generar el archivo WAV y generación de este.



CLASE PRINCIPAL: EjecutaWAV

La clase principal en el programa es la clase denominada EjecutaWAV. En ella es donde se realiza la implementación de lo correspondiente al manejo de archivos (formato txt para el archivo de entrada y WAV para el archivo de salida) además de que en esta se hace uso de los métodos pertenecientes a la clase GeneraWAV, la cual se encarga, de manera general, de la generación del archivo WAV. Es importante señalar que el uso del archivo txt permite obtener los datos relevantes para generar el archivo de audio, los cuales corresponden con el nombre que recibirá el archivo WAV (cadena), la frecuencia de muestreo (valor entero en Hertz), el número de canales que para este caso por defecto será de 1 pues se trata de un sonido monoaural, el valor de armónico (valor entero expresado en Hertz) y la duración del sonido (valor entero en segundos).

Para desarrollar la clase principal, se siguió el siguiente proceso:



PROGRAMACIÓN ORIENTADA A OBJETOS



1. Se declaran y crean dos objetos que permiten efectuar la lectura de datos sobre el archivo txt, para lo que se emplean las clases `FileReader` y `BufferedReader`. Para la instancia correspondiente `FileReader`, se pasa por parámetro el nombre del archivo txt a leer, el cual se recibe en terminal a través del argumento de la clase principal `args [0]`. Para la instancia `BufferedReader` se pasa por referencia el objeto `FileReader` generado previamente.

```
FileReader fr = new FileReader(args[0]);  
BufferedReader br = new BufferedReader(fr);
```

2. Después pasamos a leer cada uno de los datos del archivo txt, además de realizar el manejo de excepciones que pueden ser generadas durante la lectura de este. Para cada uno de los datos se utiliza el método `readLine ()` accediendo a él a través del objeto `BufferedReader`.
 - Lectura del nombre: para leer el nombre a asignar al archivo de audio usando el método ya mencionado, es importante controlar los errores que se pueden originar en dado caso de que no se proporcione el nombre del archivo, es decir, que lo leído por el `BufferedReader` sea nulo (`""`). Para esto se emplea la excepción `NullPointerException` con la cual indicamos (si es el caso) que el nombre del archivo es nulo.

```
String name = br.readLine();  
if (name.equals("")) {  
    br.close();  
    throw new NullPointerException("Valor nulo para el nombre del archivo.");  
}
```



PROGRAMACIÓN ORIENTADA A OBJETOS



- Lectura de frecuencia de muestreo: en este caso las excepciones de relevancia a tratar para la lectura de frecuencia de muestreo son `NullPointerException`, que se usaría en un caso en el que ocurra algo similar con la lectura del nombre, es decir, que el valor de la frecuencia sea nulo e `IllegalArgumentException` para manejar el caso en el que la frecuencia que se tenga no cuente con el formato adecuado, es decir, que sus dígitos no sean valores numéricos (o sea cuando contiene caracteres). NOTA: en el caso de que el valor corresponda con el formato correcto, la frecuencia de muestreo debe convertirse a un valor entero.

```
String iFrecuenciaMuestreoString = br.readLine();
if (iFrecuenciaMuestreoString.equals("")) {
    br.close();
    throw new NullPointerException("Valor nulo para la frecuencia de muestreo.");
}
for (int i = 0; i < iFrecuenciaMuestreoString.length(); i++) {
    if (!Character.isDigit(iFrecuenciaMuestreoString.charAt(i))) {
        br.close();
        throw new IllegalArgumentException("Argumento no permitido para la frecuencia de muestreo.");
    }
}
int iFrecuenciaMuestreo = Integer.parseInt(iFrecuenciaMuestreoString);
```

- Lectura de número de canales: para leer el número de canales aplican las mismas excepciones que para la frecuencia: `NullPointerException`, en el caso de que el valor sea nulo e `IllegalArgumentException` en dado caso de que el valor recibido no sea un valor numérico entero positivo, es decir, que los dígitos del valor no correspondan con valores enteros.
NOTA: Si el valor si tiene el formato solicitado, entonces es necesario realizar la conversión de cadena a valor entero para poder operarlo adecuadamente.



```
String numCanalesString = br.readLine();
if (numCanalesString.equals("")) {
    br.close();
    throw new NullPointerException("Valor nulo para el numero de canales.");
}
for (int i = 0; i < numCanalesString.length(); i++) {
    if (!Character.isDigit(numCanalesString.charAt(i))) {
        br.close();
        throw new IllegalArgumentException("Argumento no permitido para el numero de canales.");
    }
}
short numCanales = Short.parseShort(numCanalesString);
```

- Lectura del armónico: la lectura del armónico igualmente necesita del manejo de excepciones `NullPointerException` e `IllegalArgumentException`. La primera es para controlar la ejecución si el valor del armónico es nulo y la segunda es para verificar el formato del valor recibido, es decir, se debe confirmar que el valor dado contenga dígitos de valor numérico. **NOTA:** Al igual que para la frecuencia de muestreo y el número de canales, debe efectuarse la conversión de cadena a entero si no hay excepciones que "atrapar".

```
String armonicoString = br.readLine();
if (armonicoString.equals("")) {
    br.close();
    throw new NullPointerException("Valor nulo para la frecuencia del armonico.");
}
for (int i = 0; i < armonicoString.length(); i++) {
    if (!Character.isDigit(armonicoString.charAt(i))) {
        br.close();
        throw new IllegalArgumentException("Argumento no permitido para la frecuencia del armonico.");
    }
}
int armonico = Integer.parseInt(armonicoString);
```



- Lectura del tiempo: la lectura de la duración del sonido también requiere del manejo de excepciones `NullPointerException` e `IllegalArgumentException`. La primera es para controlar la ejecución si el valor del tiempo es nulo y la segunda es para controlar el formato del valor recibido, es decir, se debe confirmar que el número proporcionado contenga dígitos numéricos; si esto no ocurre entonces se lanza la excepción. **NOTA:** Al igual que para la frecuencia de muestreo, el armónico y el número de canales, se debe realizar la conversión de cadena a entero si no hay excepciones a tratar.

```
String iTiempoString = br.readLine();
if (iTiempoString == null) {
    br.close();
    throw new NullPointerException("Valor nulo para la duracion del sonido.");
}
for (int i = 0; i < iTiempoString.length(); i++) {
    if (!Character.isDigit(iTiempoString.charAt(i))) {
        br.close();
        throw new IllegalArgumentException("Argumento no permitido para la duracion del sonido.");
    }
}
int iTiempo = Integer.parseInt(iTiempoString);
```

3. Para poder generar el archivo WAV, en la clase principal, primero se crea una instancia de la clase `GeneraWAV` pasando por parámetro el número de canales del audio al constructor de la clase (véase apartado CLASE AUXILIAR: `GeneraWAV`). Esta se crea con el fin de emplear el método `Escribe ()`, para poder crear y escribir sobre el archivo en formato WAV.

```
GeneraWAV generador = new GeneraWAV(numCanales);
```

4. Finalmente, se llama al método `Escribe ()` usando la instancia de la clase `GeneraWAV` anteriormente creada, para poder realizar la escritura sobre el archivo. El funcionamiento de este método se trata a detalle en el siguiente apartado.



```
generador.Escribe(name, iTiempo, iFrecuenciaMuestreo, armonico);
```

CLASE AUXILIAR: GeneraWAV

Otra de las clases desarrolladas en este proyecto es la que se encarga de la generación del archivo en formato WAV la cual es empleada en la clase principal. De esta sección del programa, podemos destacar el procedimiento que se sigue para realzar la implementación.

1. Primero se define una variable con modificador de acceso privado para almacenar el número de canales del audio, a la cual se le asignará el valor cuando se use el método constructor.

```
private short numCanales;
```

2. Definimos el método constructor de la clase, el cual recibirá como parámetro la cantidad de canales del audio, la cual a su vez asignará el valor al atributo correspondiente dentro la clase.

```
public GeneraWAV(short numCanales) {  
    this.numCanales = numCanales;  
} // Cierre del constructor
```

3. Se establece el método que permitirá efectuar la escritura sobre el archivo. Este debe ser de tipo void y recibe como parámetros el nombre a asignar al archivo, el tiempo o duración del sonido, la frecuencia de muestreo y el armónico.

```
public void Escribe(String name, int iTiempo, int iFrecuenciaMuestreo, int armonico)
```



PROGRAMACIÓN ORIENTADA A OBJETOS



- Después comenzamos por crear un objeto tipo File, usando el constructor de la clase y pasando como parámetro el nombre del archivo. Acto seguido, utilizamos el método de la clase `createNewFile()`, para crear el archivo de audio, el cual se almacenará sobre el directorio de trabajo que se maneje. En esta sección en específico del programa y en asuntos relacionados a la escritura, puede darse una excepción, pues se está trabajando con lectura y escritura de archivos, por lo cual es necesario atrapar el error. Para ellos se incluye un bloque try-catch para el control de una `IOException`.

```
File archivo = new File(name);  
archivo.createNewFile();
```

- Se crea un objeto tipo `DataOutputStream` para el manejo de la salida de datos (bytes) pasando como parámetro una instancia `FileOutputStream` en el constructor. El objeto `FileOutputStream` se crea pasando como parámetro al constructor el objeto File creado previamente y este se define para poder emplear el objeto `DataOutputStream` de forma adecuada para la escritura en el archivo.

```
DataOutputStream salida = new FileOutputStream(new FileOutputStream(name));
```

- Definimos las constantes en cantidad de bytes para cada tipo de variable: en el caso de variables enteras corresponden 4 bytes, para variables short corresponden 2 bytes y para el encabezado se contemplan 36 bytes.

```
final short bytesInt = 4;  
final short bytesShort = 2;  
final short restodebytes = 36;
```



7. Se declaran las cadenas que indican los encabezados en el archivo, los cuales deben ser "RIFF", "WAVEfmt" y "data". Igualmente se usan variables para el formato con valor de 16, el número de bytes calculando como el número de canales por el formato (16) entre 8, el número de bits que corresponde con el valor de formato en short, el número de muestras que se calcula como el tiempo por la frecuencia de muestreo, el tamaño del archivo, el cual se obtiene como el producto del número de muestras por los bytes sumado al resto de bytes, el PCM (con valor 1), etc.

```
final String Riff = "RIFF";
final String Wave = "WAVEfmt ";
final String Data = "data";

long Formato = 16;
short Bytes_m = (short) (numCanales * Formato / 8);
short Bits_m = (short) Formato;
int iNumMuestras = iTiempo * iFrecuenciaMuestreo;
long Tamano = iNumMuestras * Bytes_m + restodebytes;

short PCM = 1;
int F_muestreo = (int) (iFrecuenciaMuestreo * numCanales * Formato / 8);
int BytesArchivo = iNumMuestras * numCanales * Bytes_m;
```

8. Se introducen al archivo, las banderas para el encabezado, pues se debe escribir los indicadores de datos como los son el RIFF, el tamaño, WAVEfmt, formato, PCM, número de canales, la frecuencia, la frecuencia de muestreo, cantidad de bytes por muestra, cantidad de bits por muestra, data y finalmente, la cantidad de bytes que ocupan las muestras en el archivo en formato WAV.



```
salida.writeBytes(Riff);
salida.write(convertEndiannessInt((int) Tamano), 0, bytesInt);
salida.writeBytes(Wave);

// Format chunk
salida.write(convertEndiannessInt((int) Formato), 0, bytesInt);
salida.write(convertEndiannessShort(PCM), 0, bytesShort);
salida.write(convertEndiannessShort(numCanales), 0, bytesShort);
salida.write(convertEndiannessInt(iFrecuenciaMuestreo), 0, bytesInt);
salida.write(convertEndiannessInt(F_muestreo), 0, bytesInt);
salida.write(convertEndiannessShort(Bytes_m), 0, bytesShort);
salida.write(convertEndiannessShort(Bits_m), 0, bytesShort);

// Data chunk
salida.writeBytes(Data);
salida.write(convertEndiannessInt(BytesArchivo), 0, bytesInt);
```

NOTAS:

- Para la realización de este paso se hace uso de dos métodos de la clase `DataOutputStream`: `writeBytes ()` y `write ()`.
 - Al pasar como parámetro los datos a escribir a los métodos anteriores, en particular, los datos como lo son el tamaño, el formato, PCM, el número de canales, la frecuencia de muestreo, etc., deben ser convertidos a cadenas de bytes en Little Endian: para el caso de enteros se implementó el método `convertEndiannessInt ()` y para el caso de shorts se desarrolló el método `convertEndiannessShort ()`, ambos en la clase `GeneraWAV`.
9. Al finalizar, se escriben sobre el archivo en formato WAV, las muestras resultantes de ángulo y los valores tipo short en una estructura lineal con elementos con tipo de dato byte para efectuar la escritura.



```
// Generando señal senoidal
int amplitud = 32760;
int muestra;

for (int i = 0; i < iNumMuestras; i++) {
    muestra = (int) Math.floor(amplitud * Math.sin((armónico * Math.PI * bytesShort * i) / iFrecuenciaMuestreo));
    salida.write(convertEndiannessInt(muestra), off: 0, bytesInt);
}
```

- Para calcular el ángulo se usa la siguiente expresión:

$$\text{ángulo} = \frac{(\text{armónico} * \text{PI} * \text{bytesShort})}{\text{frecuencia de muestreo}}$$

- Cada calcular cada valor de muestra, se abre un ciclo que itere desde 0 hasta numMuestras-1 y se utiliza la siguiente expresión para el cálculo de estas:

$$\text{muestra} = \text{amplitud} * \sin(\text{ángulo} * i)$$

Además, dentro del mismo ciclo se realiza la escritura de cada una de las muestras en el archivo, para lo cual hacemos uso de la función write (), a la cual pasamos por parámetro los valores de muestra en bytes.

- **convertEndiannessInt ()**: Convierte un dato de tipo entero de Big Endian a Little Endian. Recibe como parámetro el valor de tipo int a convertir. Regresa una cadena de bytes invertidos a Little Endian.

```
private static byte[] convertEndiannessInt(long valor) {
    byte[] resultado;
    byte b0 = (byte) (valor & 0xFF);
    byte b1 = (byte) ((valor >> 8) & 0xFF);
    byte b2 = (byte) ((valor >> 16) & 0xFF);
    byte b3 = (byte) ((valor >> 24) & 0xFF);
    resultado = new byte[] { b0, b1, b2, b3 };
    return resultado;
} // Cierre del método
```



- **convertEndiannessShort ()**: Convierte un dato de tipo short de Big Endian a Little Endian. Recibe como parámetro el valor de tipo short a convertir. Regresa una cadena de bytes invertidos a Little Endian.

```
private static byte[] convertEndiannessShort(short valor) {  
    byte[] resultado;  
    byte b0 = (byte) (valor & 0xFF);  
    byte b1 = (byte) ((valor >> 8) & 0xFF);  
    resultado = new byte[] { b0, b1 };  
    return resultado;  
} // Cierre del método
```

CLASE PRUEBA: EscucharWAV

La clase de prueba EscuchaWAV fue proporcionada para poder probar la ejecución del programa. Esta fue facilitada por el docente con el propósito de poder escuchar el audio generado en formato WAV.

```
import java.io.File;  
import javax.sound.sampled.AudioSystem;  
import javax.sound.sampled.Clip;  
  
/**  
 * Esta clase sirve para escuchar un sonido de un archivo <i>.wav</i> de 10 segundos.  
 * El código fue proporcionado por el profesor. Para generar el archivo <i>.wav</i> se  
 * tiene que ejecutar primero la clase <i>EjecutaWAV</i>.  
 *  
 * @see EjecutaWAV  
 */  
public class EscucharWAV {  
    /**  
     * Para ejecutar se utiliza el siguiente comando en terminal (desde la raíz del proyecto):  
     * <ul>  
     * <li><i>java -cp bin EscucharWAV Prueba.wav</i></li>  
     * </ul>  
     *  
     * <p>  
     * El argumento <i>Prueba.wav</i> puede variar, dependiendo del nombre que se eligió para el  
     * archivo.  
     */  
}
```



```
* @param args Command-line arguments. <i>args[0]</i> debe corresponder al nombre del archivo
* <i>.wav</i>
* que se va a leer.
*/
public static void main(String[] args) {
    try {
        Clip sonido = AudioSystem.getClip();
        File a = new File(args[0]);
        sonido.open(AudioSystem.getAudioInputStream(a));
        sonido.start();
        System.out.println("Reproduciendo 10s de sonido...");
        Thread.sleep(10000);
        sonido.close();
    } catch (Exception e) {
        System.out.println(e);
    }
} // Cierre del método
} // Cierre de la clase
```

RESULTADOS

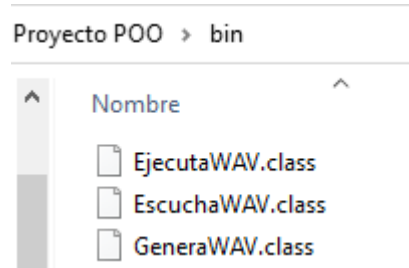
Con el fin de realizar pruebas al programa implementado, es necesario comenzar por efectuar la compilación de los archivos .java para obtener los archivos .class. Esto se efectuó empleando consola, indicando el comando javac seguido de los nombres de los archivos .java:

```
C:\> Símbolo del sistema
C:\Users\Elsa\Desktop\Proyecto P00\src>javac EjecutaWAV.java GeneraWAV.java EscucharWAV.java
C:\Users\Elsa\Desktop\Proyecto P00\src>
```

Con lo cual generamos los tres archivos: EjecutaWAV.class GeneraWAV.class y EscucharWAV.class:

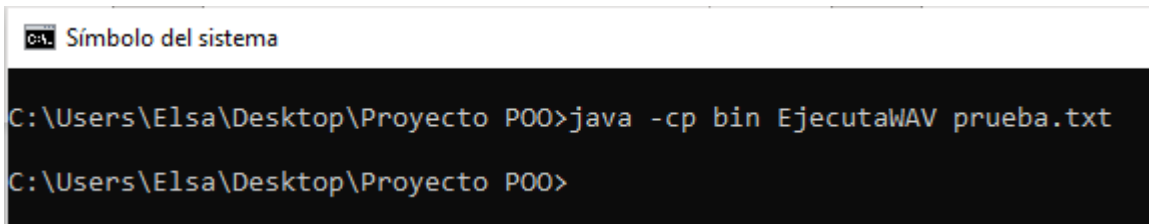
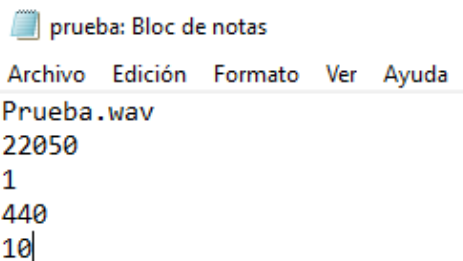


PROGRAMACIÓN ORIENTADA A OBJETOS



Después de esto, es importante ejecutar la clase EjecutaWAV pues por medio de esta clase es que se pueden obtener los valores de las características que se usarán para crear el archivo de audio en formato WAV y también es donde se realiza la llamada a la función para la escritura sobre el archivo. Al ejecutar la clase EjecutaWAV, es necesario pasar como parámetro el nombre del archivo de texto plano en donde se encuentran las características que mencionamos anteriormente, que en este caso es "prueba.txt". Para ejecutar se hace uso del comando:

```
java -cp bin EjecutaWAV prueba.txt
```





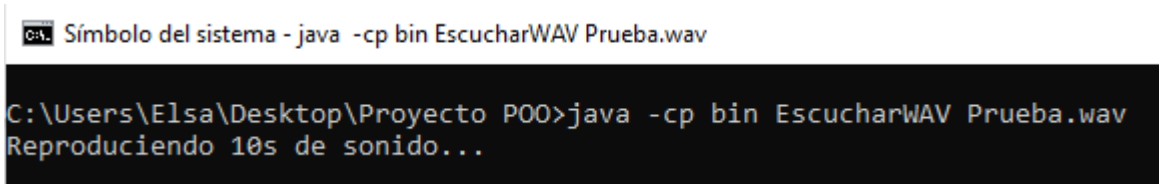
PROGRAMACIÓN ORIENTADA A OBJETOS



NOTA: Como se observa se indica el directorio donde se encuentra el archivo .class para la clase Ejecuta WAV: para los archivos .class (como es el caso) los encontramos en el directorio “bin” mientras que los archivos .java se localizan en “src”.

Ahora bien, ya que ejecutamos nuestra clase principal, podemos pasar a probar el programa a través de la clase EscucharWAV (clase de prueba). Ejecutamos la clase por medio del comando java y pasando como parámetro el nombre del archivo en formato WAV del audio de prueba, el cual es “Prueba.wav”:

```
java -cp bin EscucharWAV Prueba.wav
```



```
C:\Users\Elsa\Desktop\Proyecto P00>java -cp bin EscucharWAV Prueba.wav
Reproduciendo 10s de sonido...
```

Como se observa el programa imprime un mensaje indicando que el audio está reproduciéndose. La nota empleada para las pruebas fue “La”, la cual tiene como frecuencia de 440 Hz. El sonido reproducido debe tener una duración de 10 segundos, según lo esperado en el archivo de texto plano “prueba.txt” pues en este el último parámetro indica la duración del sonido en segundos.

CONCLUSIONES

A lo largo de la materialización del programa realizado, el cual permitió llevar a cabo la implementación de la solución al problema central sobre la generación de un sonido por un archivo en formato WAV, tuvimos la oportunidad de reconocer la relevancia del uso



PROGRAMACIÓN ORIENTADA A OBJETOS



de la programación a objetos, pues este paradigma permite modularizar, es decir, realizar la división de un programa en secciones que puedan ser compiladas por separado, pero al mismo tiempo tener comunicación y conexiones con otras. Por secciones nos referimos a las clases, pues la modularidad nos permitió usar tres distintas clases relacionadas entre sí, haciendo uso de sus métodos, para poder obtener el resultado deseado: el archivo WAV.

Otra de las propiedades importantes de este tipo de programación que pudimos poner en práctica, fue el manejo de excepciones mediante clases, pues al estar empleando archivos, es muy probable que existan excepciones de tipo `IOException` en el caso de que existan problemas con la entrada y salida de datos, `NullPointerException` para aquellos casos en los que los valores leídos del archivo de texto plano sean nulos o bien, `IllegalArgumentException` cuando los valores de los datos no correspondan con el formato esperado, como por ejemplo el caso de la lectura del valor de la frecuencia, la cual tenía que ser un valor entero, por lo cual debía corroborarse que sus dígitos tuviesen estrictamente valores numéricos.

El desarrollo de la clase principal `EjecutaWAV` y la clase `GeneraWAV`, nos posibilitó aplicar el manejo de archivos en el lenguaje de programación Java, para lo cual se hizo uso de dos clases para la lectura de datos en el archivo de texto plano, principalmente: `FileReader` y `BufferedReader` (el objeto de la segunda clase permite eficientar la lectura). En el caso de la escritura de datos en el archivo de salida en formato WAV, se hizo uso de las clases `DataOutputStream` y `FileOutputStream`. Ambas facilitan la escritura de datos en el archivo WAV en tipo de dato a nivel de bytes, pues recordamos que al final lo que escribimos sobre este son bytes en Little Endian, para lo que su cálculo se realizó con conversiones desde bytes en Big Endian.

Finalmente, consideramos que el desarrollo del proyecto tiene condiciones significativas



PROGRAMACIÓN ORIENTADA A OBJETOS

en la aplicación tanto de conocimientos básicos en el paradigma orientado a objetos, como lo son la creación de clases, el uso de la modularidad a favor de la resolución y el uso de métodos por medio de instancias de objeto, como conocimientos más avanzados como lo son la conversión entre tipos de datos a nivel de bytes, el manejo de archivos y el control de errores en tiempo de ejecución.

REFERENCIAS

- "¿Qué es la Programación Orientada a Objetos?" Profile Software Services. <https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/#:~:text=¿Por%20qué%20POO?,de%20esta%20manera%20programas%20eficientes>. (accedido el 09 de enero de 2023).
- "WAV Extensión de archivo -¿Qué es .wav y cómo abrir? - ReviverSoft". ReviverSoft | Software and Tips to Make Your PC Run Like New. <https://www.reviversoft.com/es/file-extensions/wav> (accedido el 10 de enero de 2023).
- "Big Endian vs Little Endian - Arquitectura de Computadoras seccion 1500". Google Sites: Sign-in. <https://sites.google.com/site/arquitectura1500/home/big-endian-vs-little-endian> (accedido el 09 de enero de 2023).