Universidad Autónoma de Zacatecas

Practica 5

Student:
José Francisco Hurtado Muro

Teacher:
Dr. Aldonso Becerra Sánchez

March 23, 2025

1 Introducción

"La facilidad que los arreglos tienen para permitir guardar más de un dato con una sola variable lo hace pertinentes para la resolución de muchos problemas dónde se requiere esta situación. El único detalle con esta cuestión es que es poco flexible el número de elementos que podemos manipular, ya que se requiere conocer a priori la cantidad de elementos a guardar."

Contents

2 Desarrollo

El sonido es una onda de presión que requiere un transductor de presión (un micrófono) para convertir las ondas de presión de aire (ondas sonoras) en señales eléctricas (señales analógicas). La conversión contraria se realiza mediante un altavoz (bocina), el cual convierte las señales eléctricas en ondas de presión de aire. Sele pide que grabe un frase auditiva usando el programa Wavesurfer. En la figura 1 se puede ver este programa.

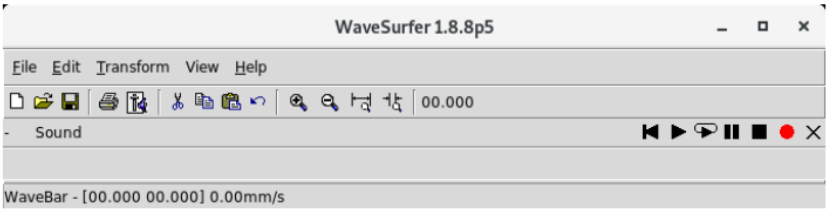

Figure 1: WaveSurfer

Oprimiendo el botón de botón rojo (circular) comienza la grabación de voz, sin embargo, antes de eso se requiere realizar lo siguiente:

- Oprimiendo el botón rojo (circular) comienza la grabación de voz, sin embargo, antes de eso se requiere realizar lo siguiente:
- Oprima el botón derecho sobre la zona que se encuentra debajo de la palabra "Sound". Aparecerá un menú como el de la Figura 2, ahí debe seleccionar agregar el panel de "waveform".
- Después puede oprimir el botón rojo para comenzar la grabación. Se le pide que, inmediatamente después de presionar el botón, diga su "nombre completo" con todo y apellidos, seguido de la frase indicada abajo.
- Para detener la grabación, oprima el botón negro cuadrado. Por ejemplo, el audio debe quedar grabado con:

  > "Aldonso Becerra Sánchez, quiero que sepan que mi mamá me mima y que el otro día vi que estaba la pájara pinta, sentada en su verde limón, con el pico cortaba la rama, con la rama cortaba la flor".

  Y el resultado será similar a la Figura 3.
- Posteriormente, guarde el archivo como un archivo `.wav`.

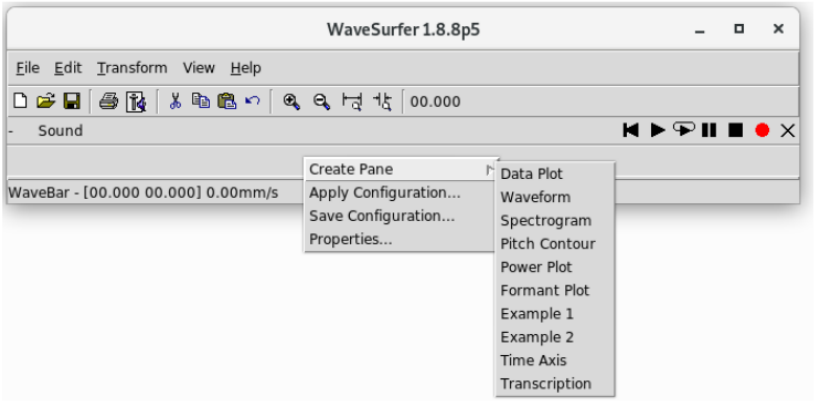

Figure 2: WaveSurfer - menu de paneles

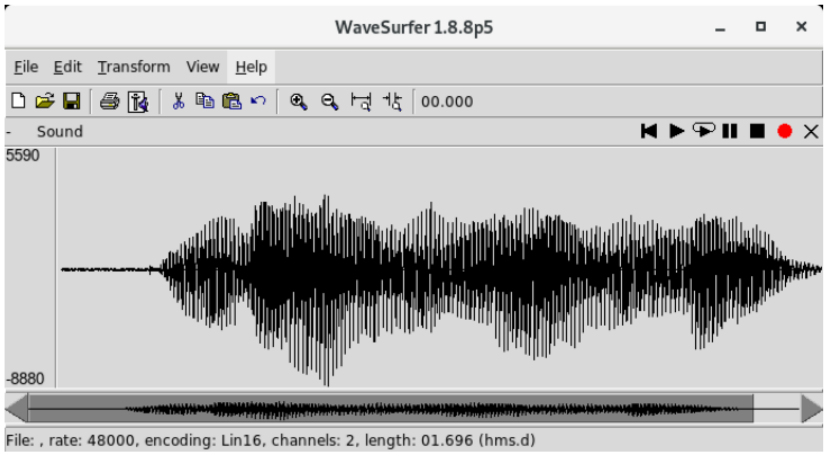

Figure 3: WaveSurfer - forma de onda de voz

Si se oprime botón derecho sobre la zona de la forma de onda aparece un menú, ahí oprima el botón del ratón sobre propiedades (figura 4). Ahí se pueden apreciar las características de la señal grabada; por ejemplo, la tasa de muestreo (el número de puntos que son obtenidos/discretizados (cada puntito equivale a un valor numérico que puede ser guardado en un arreglo) (ver figura 5) de la señal analógica continua [forma de onda], el número de canales [generalmente se llama sonido estereofónico o estéreo al sonido grabado y reproducido en dos canales. Los discos de vinilo, los CD audio, la mayoría de las estaciones de radio FM, casetes y la totalidad de canales de TV y televisión vía satélite transmiten señales de audio estéreo. El propósito de grabar en sonido estereofónico es el de recrear una experiencia más natural al escucharlo, y donde, al menos en parte, se reproducen las direcciones izquierda y derecha de las que proviene cada fuente de sonido grabada], y el número de bis usados para codificar/discretizar la señal numérica que es convertida por el micrófono).

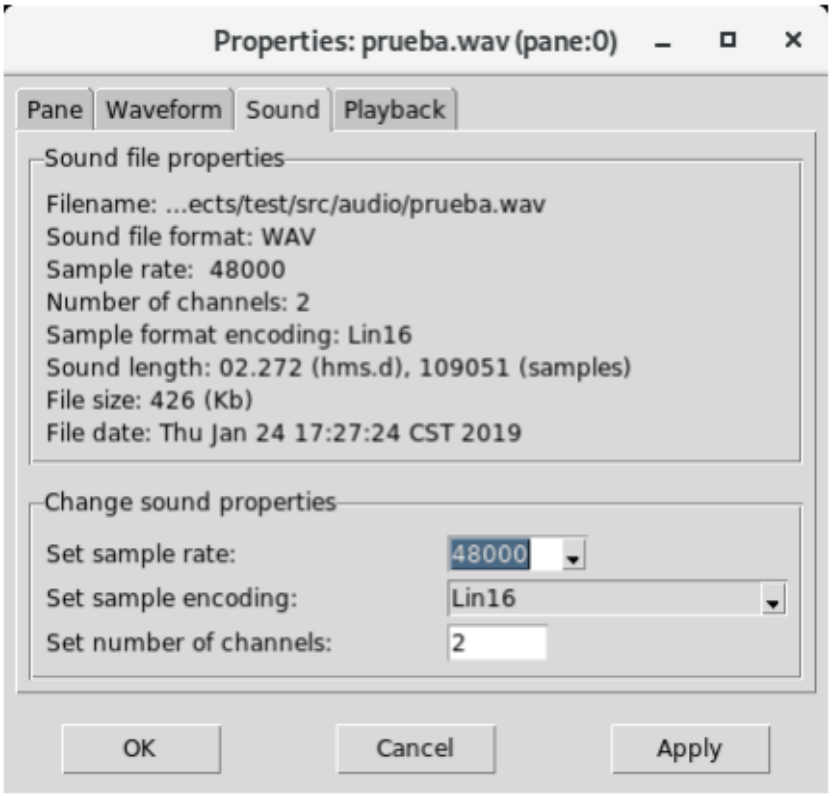

Figure 4: WaveSurfer - Configuración de audio

Figure 5: WaveSurfer - Discretización de voz

Tomando como base el programa que se proporciona en Java, el cual hace una lectura de un archivo de audio (wav), se le pide lo siguiente:

2.1 Acoplamiento de la clase AudioFileRecord - leerAudio

1. Acople el programa de la clase AudioFileRecord para que los datos del buffer2[] del método leerAudio ahora queden guardados en la clase ArregoNumerico. Por ejemplo, puede crear un método, en la clase Arreglo, que reciba el arreglo buffer[] (para que se pueda tener esta funcionalidad desde la super clase, no solo con valores numéricos) y lo guarde en el contenido del arreglo interno de la clase Arreglo (en la variable datos[], sobreescriba la variable interna del arreglo de los datos para que no se dupliquen los

almacenamientos de memoria). La idea es que todo procesamiento se haga sobre una variable de tipo ArregoNumerico

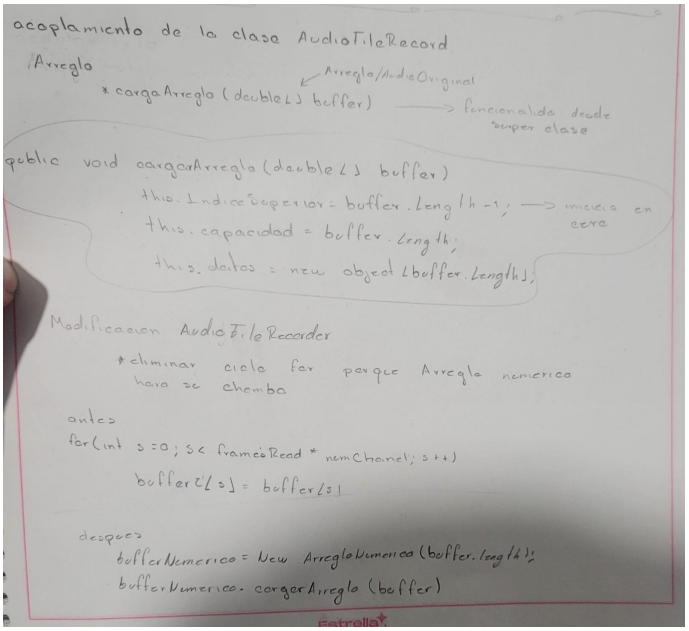

Figure 6: Análisis: Acoplamiento de la clase AudioFileRecord - leerAudio

Explicación:
Lo que se hizo fue en la clase AudioFileRecorder, modificar lo que es el método leerAudio **leerAudio:** se cambio el for que creaba la instancia en un arreglo double, por una instancia de tipo ArregloNumerico al cual solo se el pasa los datos gracias a un método se la súper clase llamado cargarArreglo quien se encargaba de crear un nuevo ArregloNumerico llamado buffer2

Figure 7: Prueba: Acoplamiento de la clase AudioFileRecord - leerAudio

2.2 Acoplamiento de la clase AudioFileRecord - escribirAudio

2. Acople el programa para que el método escribirAudio pueda guardar el buffer2[] como un datos[], para eso puede crear un método en la clase Arreglo (por ejemplo, public Object[] leerArreglo()) que proporcione una copia de los datos (no debe regresar el arreglo original, sino sólo una copia, esto garantiza la encapsulación) para después usar esa copia y almacenarla en buffer2[] como double, para que así pueda ser usado el método writeFrames. La idea es que la información guardada en la variable de tipo ArregoNumerico sea extraída para que wirteFrames pueda hacer su trabajo como arreglo con [].

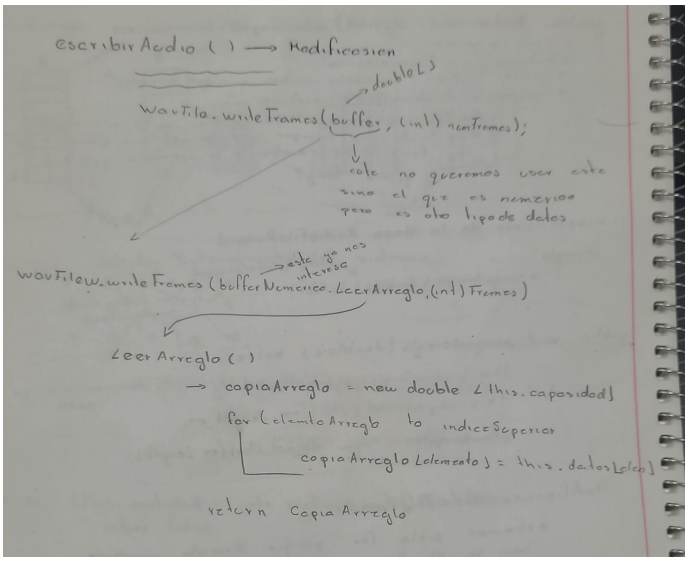

Figure 8: Análisis: Acoplamiento de la clase AudioFileRecord - escribirAudio

Explicación:
lo que se hizo fue modificar un elemento del método escribirAudio y ademas se hizo un metodo auxiliar que lo que hace es ayudarnos con el tipo de dato del arreglo ya que el arregloNumerico su datos[ ] es de tipo Number y lo que se hizo fue que este método ademas de castear los valores, tambien retornara una copia de ese arreglo pero ya con los valores casteados

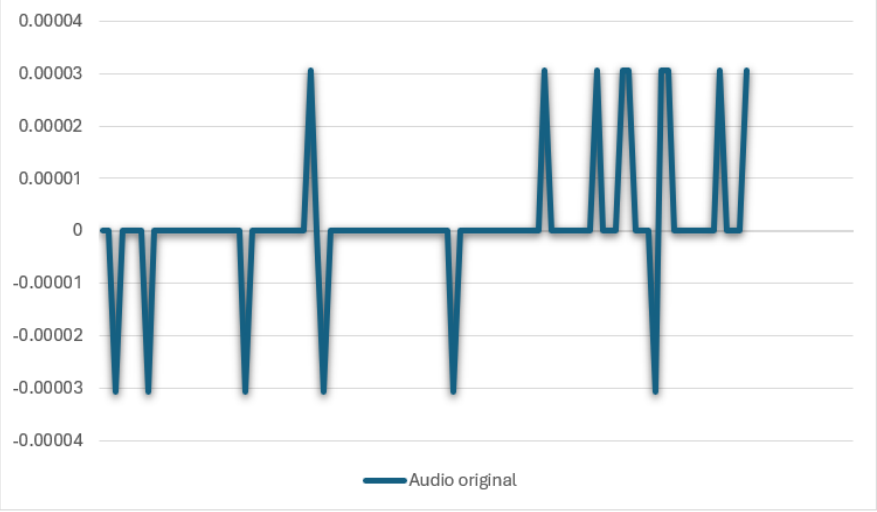

Figure 9: Audio original

2.3 public void subirVolumen(int intensidad)

Realizar un método en la clase AudioFileRecord que incremente el volumen del archivo de audio almacenado en la variable temporal buffer2 de tipo ArregoNumerico: public void subirVolumen(int intensidad). Intensidad es la cantidad de volumen a subir, la cual debe ser proporcionada por el usuario.

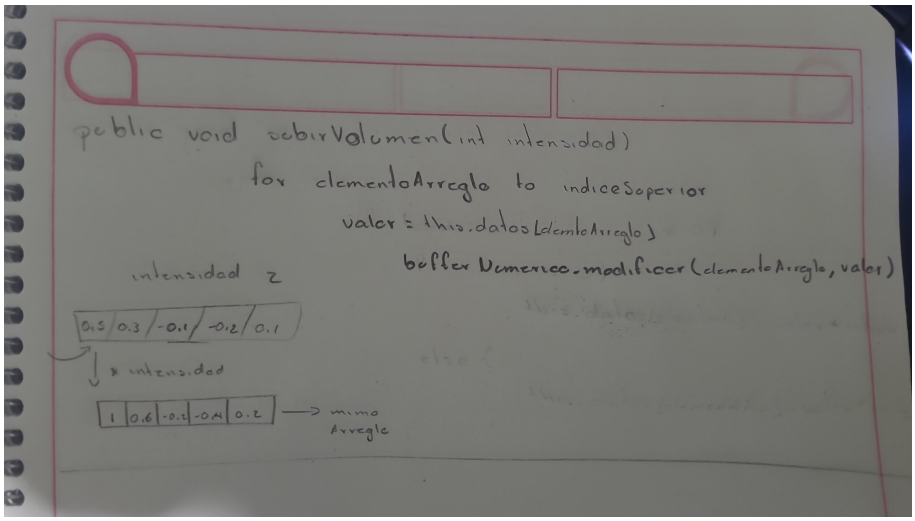

Figure 10: Análisis: SubirVolumen(int intensidad)

Explicación:
en este caso se propuso subir el volumen del audio original, donde a los valores originales del audio se multiplicaron por un valor diferente a 0 que el usuario quisiera, el cual iba aumentar los valores en el eje Y, tanto para positivo y negativos, en la figura 11 podemos ver una superposición de la honda subirVolumen y la del audio original que podemos apreciar mejor en la figura 9.

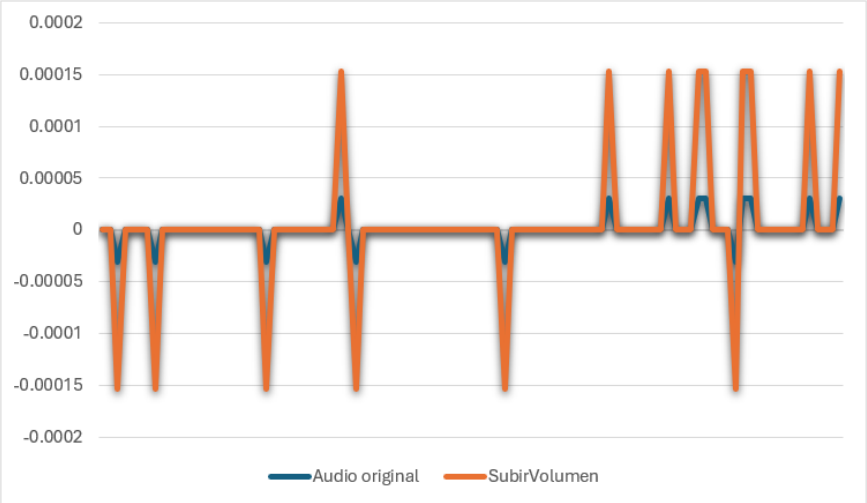

Figure 11: Comparación: Audio original y subirVolumen 5 niveles

2.4 public void bajarVolumen(int intensidad)

Se pide que haga un método en la clase AudioFileRecord que decremente el volumen del archivo de audio almacenado en la variable temporal buffer2 de tipo ArregoNumerico: public void bajarVolumen(int intensidad). Intensidad es la cantidad de volumen a bajar, la cual es proporcionada por el usuario.

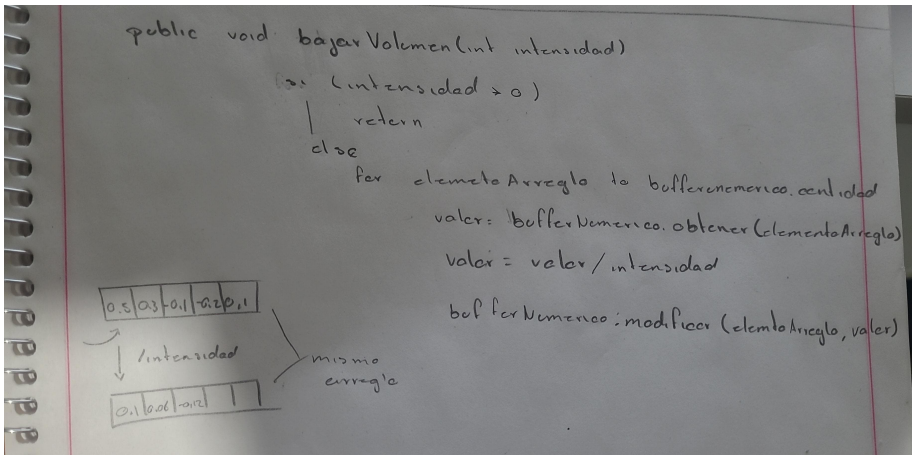

Figure 12: Análisis: public void bajarVolumen(int intensidad)

Explicación:
en este caso se propuso subir el volumen del audio original, donde a los valores originales del audio se multiplicaron por un valor diferente a cero que el usuario quisiera, el cual iba aumentar los valores en el eje Y, tanto para positivo y negativos, en la figura 11 podemos ver una superposición de la honda subirVolumen y la del audio original que podemos apreciar mejor en la figura 9.

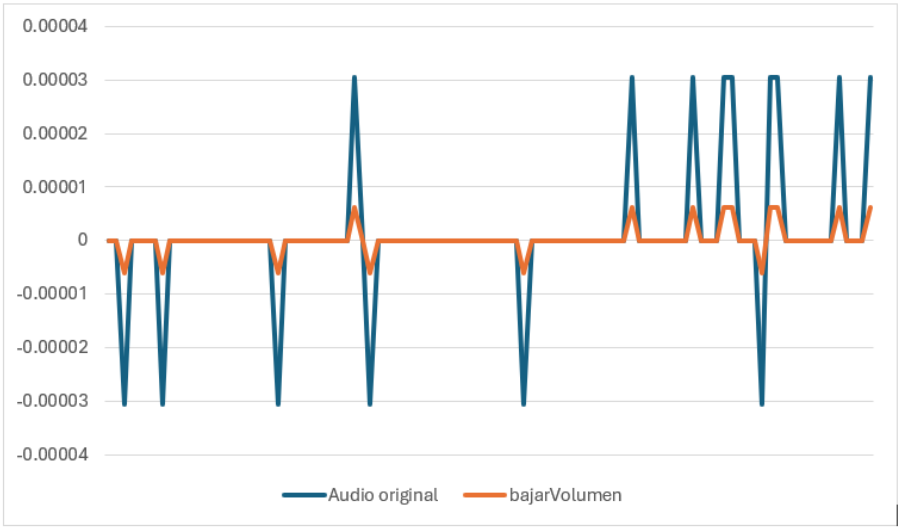

Figure 13: Comparación: Audio original y bajarVolumen 5 niveles

2.5 public acelerarPista(int velosidad)

Haga un método para acelerar la pista de audio (sonido tipo Alvin y las ardillas). Para hacer esto existen varias maneras, la que se pide que use es reducir el número de muestras por unidad de tiempo (por ejemplo a la mitad, las muestras como en la figura 5). Por ejemplo, si originalmente eran 10000 muestras (tramas*número de canales), el nuevo archivo de audio quedará de 5000. El proceso para conseguir esto es agarrar dos muestras consecutivas y promediarlas (esto es acelerar el sonido al doble), creando un nuevo arreglo ahora con la mitad de muestras. Recuerde que todo este proceso es sobre una variable de tipo ArregoNumerico. El incremento en la velocidad debe ser un parámetro, No siempre debe ser el doble, este valor debe ser configurable en el método.

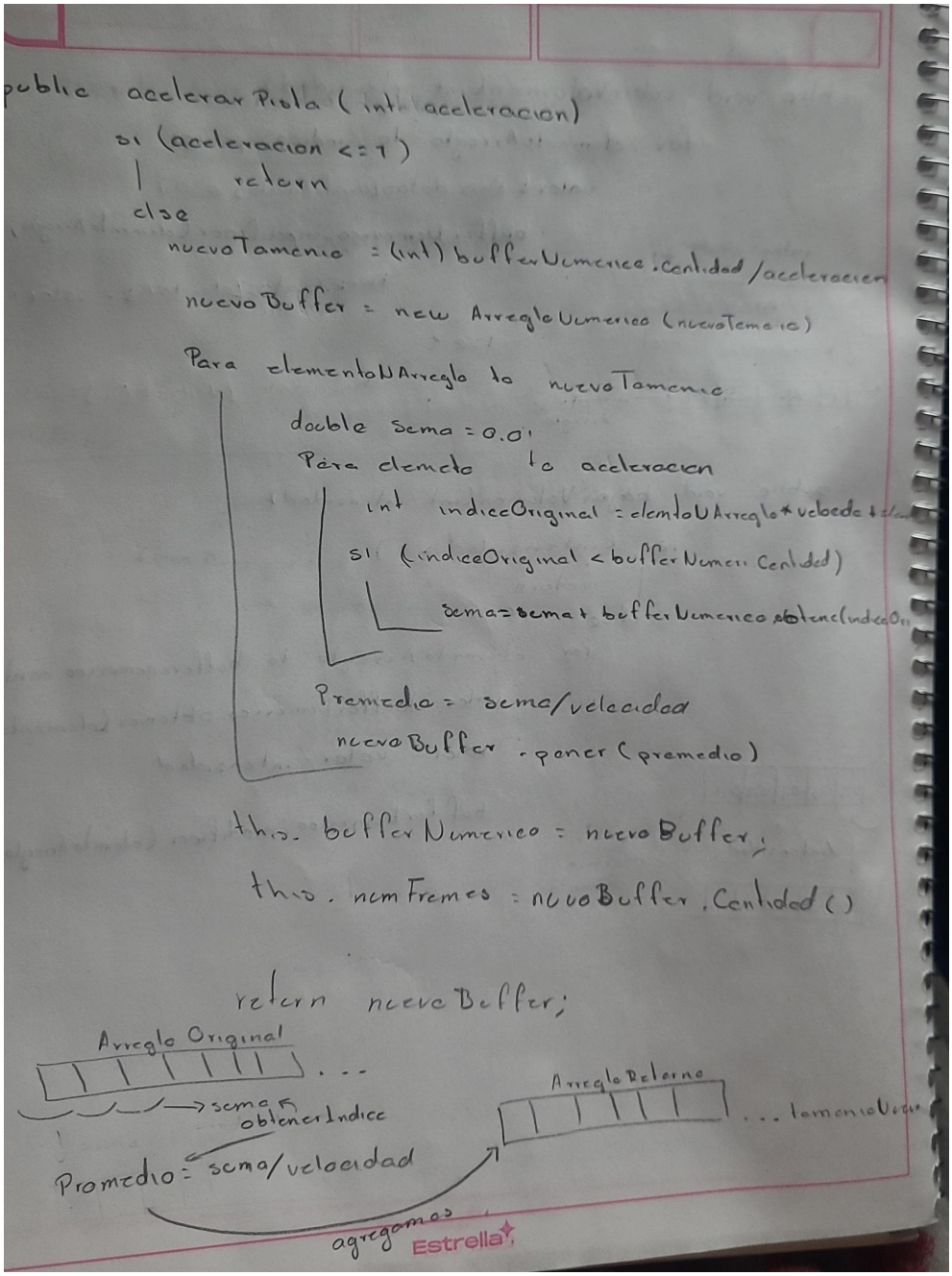

Figure 14: Análisis: public acelerarPista(int velosidad)

Explicación:
en este caso lo que se realizo es que se hizo un ArregloNumerico auxiliar el cual va a tener la mitad de elementos que bufferNumerico, sobre el cual por medio de iteración vamos analizando los datos, y sacando un promedio en el rango de la aceleración dada por el usuario, donde por medio de un indice buscaremos un elemento en el arreglo original del cual sacaremos un promedio y lo añadiremos en el nuevoBufferl

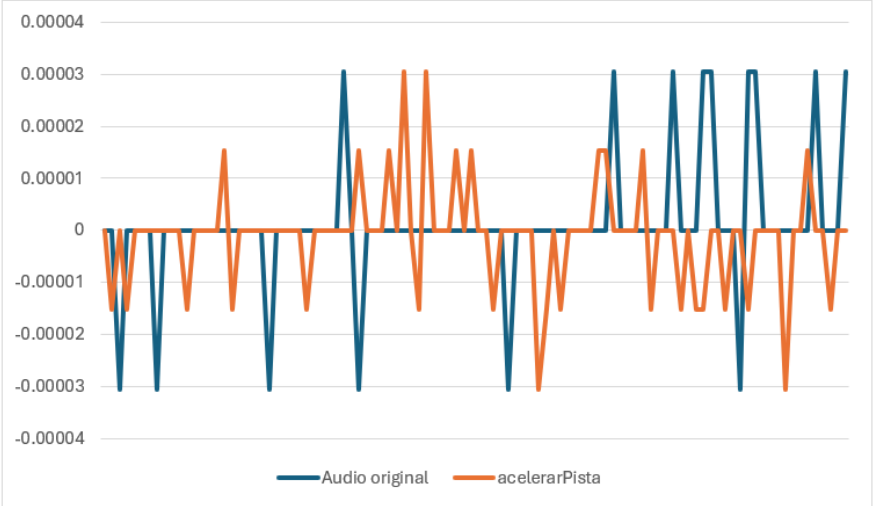

Figure 15: Comparación: Audio original y acelerarPista x2 aceleración

2.6 public retrasarPista(int desAceleracion)

Haga un método para retrasar la pista de audio (sonido fantasmal). Para hacer esto existen varias maneras, la que se pide que use es aumentar el número de muestras por unidad de tiempo (ver figura 5). Por ejemplo, si originalmente eran 10000 muestras (tramas*número de canales), el nuevo archivo de audio quedará de 20000 (en este caso fue al doble). El proceso para conseguir esto es insertar la muestra "n" en un nuevo arreglo de tipo ArregoNumerico, y después de cada muestra "n" (o antes) se debe insertar una muestra inventada que será el relleno, la cual se obtiene como un promedio de la muestra actual "n" más la muestra "n+1" (o n-1, dependiendo). El decremento en la velocidad debe ser un parámetro, el cual debe ser configurado por el usuario.

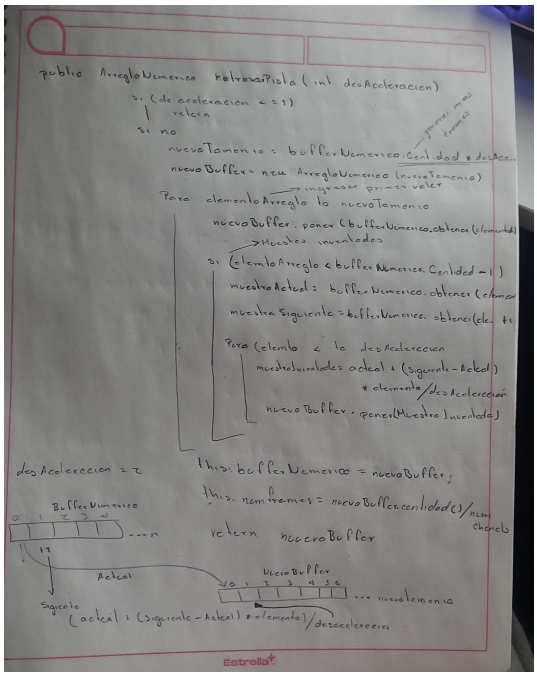

Figure 16: Análisis: public retrasarPista(int desAceleracion)

Explicación:
en este caso fue algo parecido al caso de la figura 15, pero con la diferencia que en este lo que se hacia era doblar la cantidad de elementos que tiene bufferNumerico, y aparte este saca un elemento de este y lo añade directamente al nuevobuffer, parea después por medio de operaciones creamos una serie de elementos aleatorios con el conocimiento del actual y el siguiente elemento, para después agregarlos al bufferNuevo

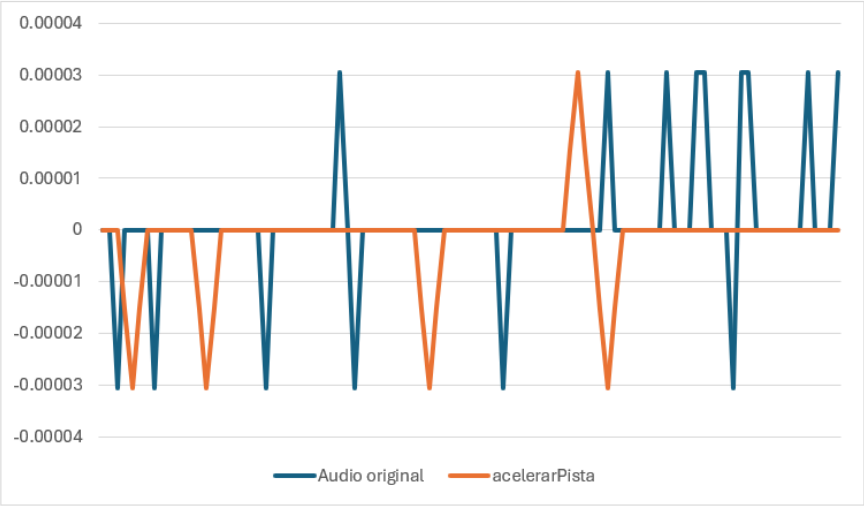

Figure 17: Comparación: Audio original y DescelerarPista x2 desAceleración

2.7 public eliminarSilencio()

Haga un método para que usted diseñe la manera de eliminar el silencio existente en un archivo de audio. Por ejemplo el mostrado en la figura 18:

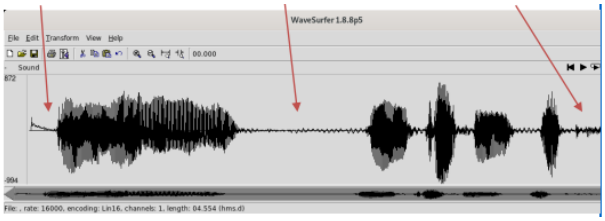

Figure 18: Ejemplo de silencios de voz

Fíjese en el silencio puede existir en cualquier posición del audio, normalmente es antes o después de palabras pronunciadas (flechas en rojo). Ese silencio debe ser removido.

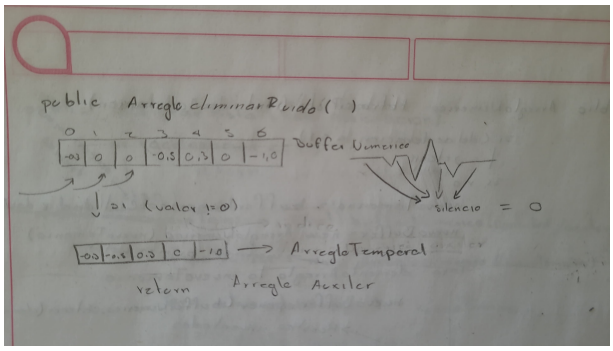

Figure 19: Análisis: public eliminarSilencio()

Explicación:
para este caso lo primero que se hizo fue hacer uso de de dos arreglos Auxiliares, uno que para contendrá todos los indices en los que se encuentra ceros que representan silencios, este paso fue con el motivo de tenerlos contados para después al tamaño bufferNumerico restarle el tamaño de este arreglo, pero en este arregloTemporal, le iremos agrando los valores diferentes a cero, para después regresar este como resultado.

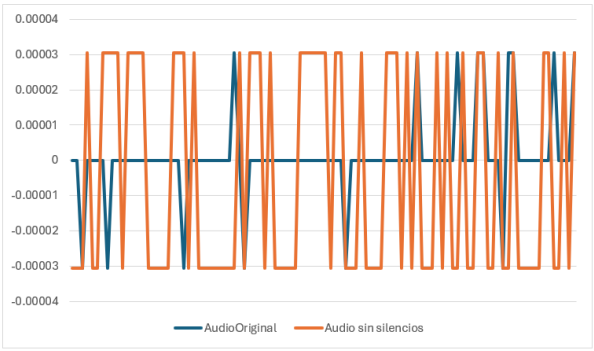

Figure 20: Comparación: Audio original y eliminación del silencio

2.8 public invertirEjeX()

Haga que el audio grabado sufra una inversión en el eje "x" (se volteé al revés en tiempo, es decir lo que estaba a la izquierda ahora estará a la derecha, y así sucesivamente). Tal como se muestra la figura 21 y 22

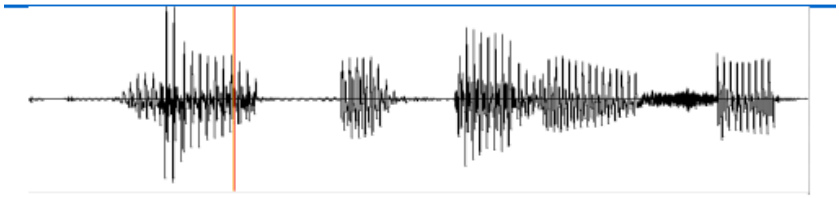

Figure 21: Audio de muestra de inversión original

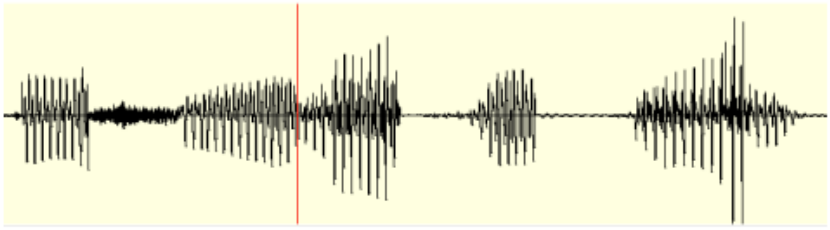

Figure 22: Audio invertido en X de figura 21

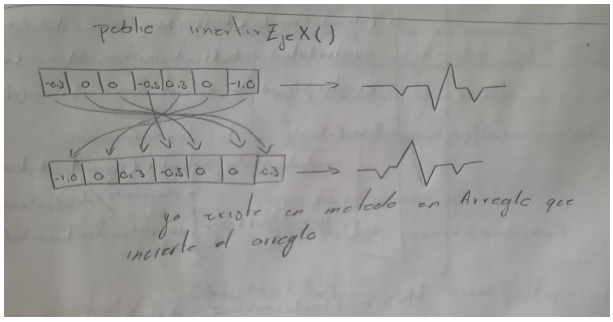

Figure 23: Análisis: public invertirEjeX()

Explicación:
lo que se planteo fue mandar todos los elementos del final a posiciones del inicio de un segundo arreglo Auxiliar, para al final regresarlos en ese orden al arreglo original, para esto se uso un método que ya existe en la clase Arreglo que invierte el orden **invertir()**

Figure 24: Comparación: Audio original y invertirEjeX()

2.9 public invertirEjeY()

Haga que el audio grabado sufra una inversión en el eje y (se volteé de cabeza en intensidad, es decir lo que estaba arriba ahora estará abajo, y así sucesivamente). Tal como se muestra la figura 25 y 26.

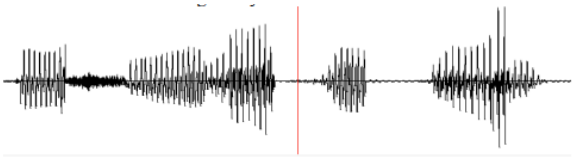

Figure 25: Audio de muestra de inversión original

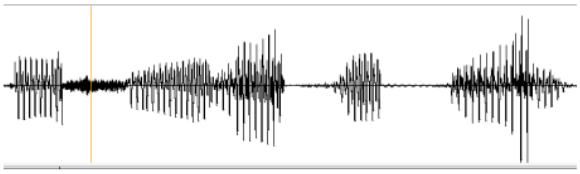

Figure 26: Audio invertido en Y de figura 25

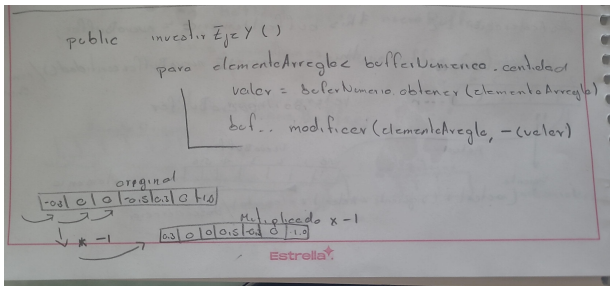

Figure 27: Análisis: public invertirEjeY()

Explicación:
en este caso lo que se hizo fue que a cada elemeto del Buffernumerico se multiplicara por -1, dando como resultado el cambio en el eje Y

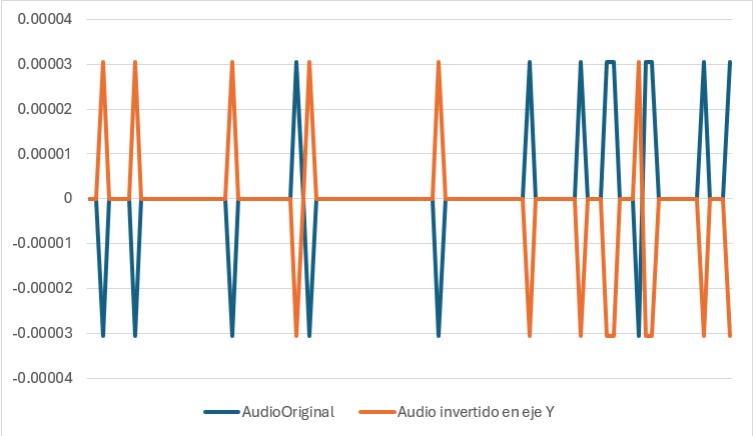

Figure 28: Comparación: Audio original y invertirEjeY()

2.10 Analizando vocales

Haga unas grabaciones de vocales (pronunciadas una vocal sola por cada archivo). Grabe varias veces una misma vocal, por ejemplo, unas 5 veces. Posteriormente obtenga la energía que producen cada una de ellas (la energía de una señal de audio se obtiene elevando al cuadrado cada una de las muestras individuales, posteriormente se suman todas). Verifique si a través de la energía puede clasificar vocales hasta cierto grado. Haga pruebas y documente los resultados

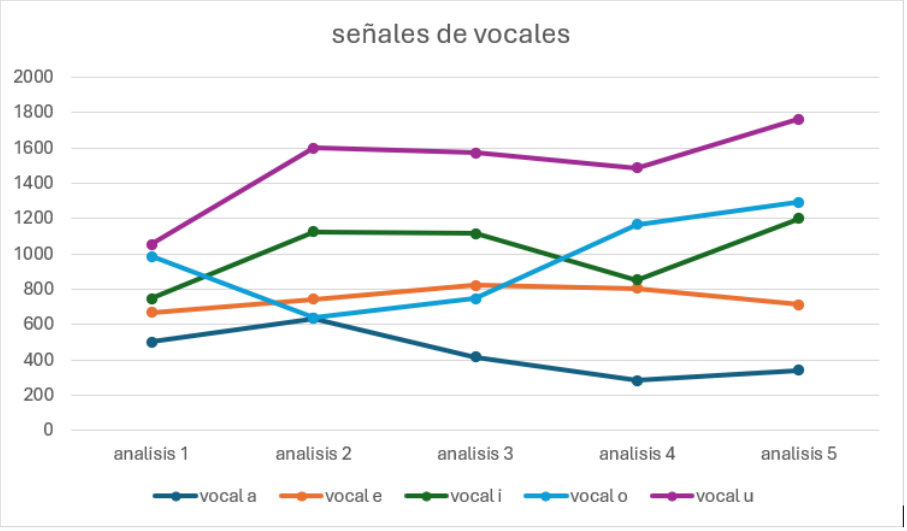

Figure 29: Análisis de vocales

Explicación:
analizando cada una de las vovales en 5 audios diferentes obtuvimos los siguientes resultados:

- **Análisis vocal a:** (Promedio: 434.157561)
  - Análisis 1: 501.785539
  - Análisis 2: 632.596885
  - Análisis 3: 416.784427
  - Análisis 4: 282.037083
  - Análisis 5: 340.582871
- **Análisis vocal e:** (Promedio: 749.648224)
  - Análisis 1: 666.806569
  - Análisis 2: 743.587402
  - Análisis 3: 822.415759
  - Análisis 4: 803.109833
  - Análisis 5: 712.323555

- **Análisis vocal i:** (Promedio: 1006.167594)
  - Análisis 1: 744.410162
  - Análisis 2: 1124.85897
  - Análisis 3: 1115.47975
  - Análisis 4: 851.42532
  - Análisis 5: 1198.66377
- **Análisis vocal o:** (Promedio: 965.683624)
  - Análisis 1: 983.567498
  - Análisis 2: 637.844319
  - Análisis 3: 747.439621
  - Análisis 4: 1165.17627
  - Análisis 5: 1292.39101
- **Análisis vocal u:** (Promedio: 1494.372736)
  - Análisis 1: 1053.74091
  - Análisis 2: 1597.11013
  - Análisis 3: 1572.65858
  - Análisis 4: 1485.32483
  - Análisis 5: 1763.02923

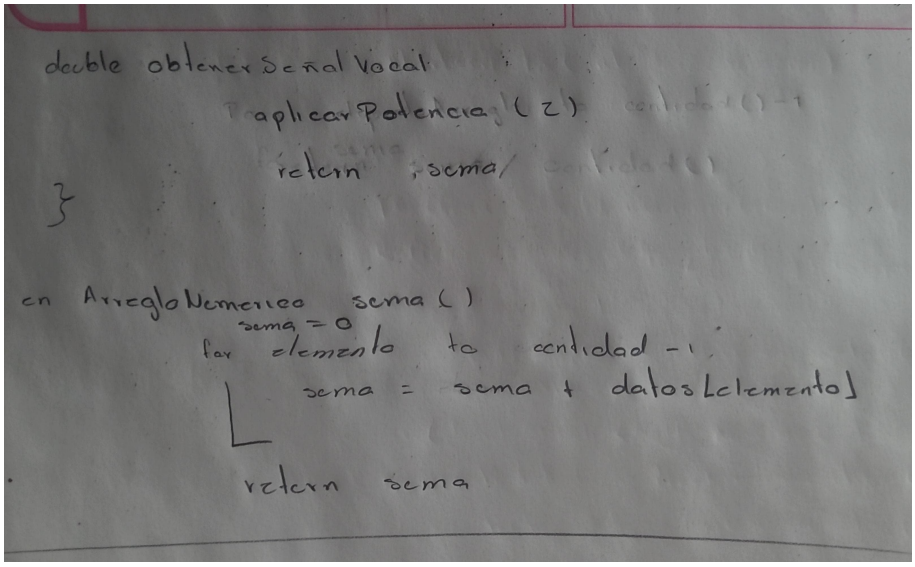

Figure 30: Análisis: public obtenerSeñal()

3 Código Agregado - UML

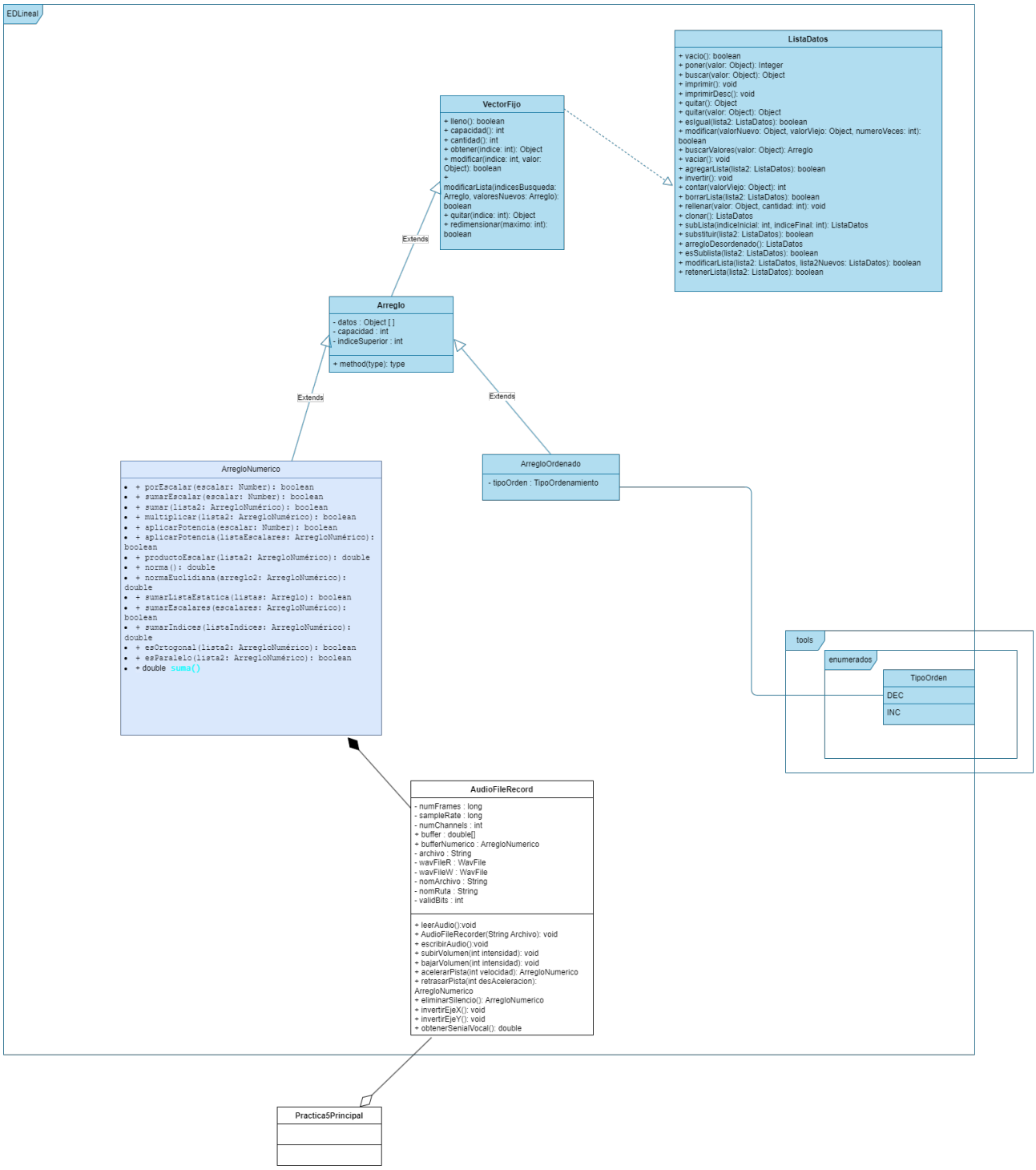

Figure 31: UML

4 Pre-evaluación del Alumno

Práctica NO:

José Francisco Hurtado Muro

Criterio	Evaluación
Cumple con la funcionalidad solicitada	**Sí**
Dispone de código auto-documentado	**Sí**
Dispone de código documentado a nivel de clase y método	**Sí**
Dispone de indentación correcta	**Sí**
Cumple la POO	**Sí**
Dispone de una forma fácil de utilizar el programa para el usuario	**Sí**
Dispone de un reporte con formato IDC	**Sí**
La información del reporte está libre de errores de ortografía	**Sí**
Se entregó en tiempo y forma la práctica	**No**
Incluye el código agregado en formato UML	**Sí**
Incluye las capturas de pantalla del programa funcionando	**Sí**
La práctica está totalmente realizada (especifique el porcentaje completado)	**95%**

Table 1: Evaluación de la práctica

5 Conclusión

se vio una manera muy practica del uso de los ArreglosNumericos, la verdad me dejo impresionado que se pudiera manejar de esta menara los audios