Manual de Usuario

Yefer Rodrigo Miguel Alvarado Tzul

Carné: 201731163

Universidad San Carlos de Guatemala

Centro Universitario de Occidente (CUNOC)

División de Ciencias de la Ingeniería

Organizaciones de Lenguajes y Compiladores 2

Ing. Moises Granados

Septiembre de 2023

Manual de Usuario

Requerimientos

Sistema Operativo (Windows o Linux)

Java 1.11.0 o compatibles.

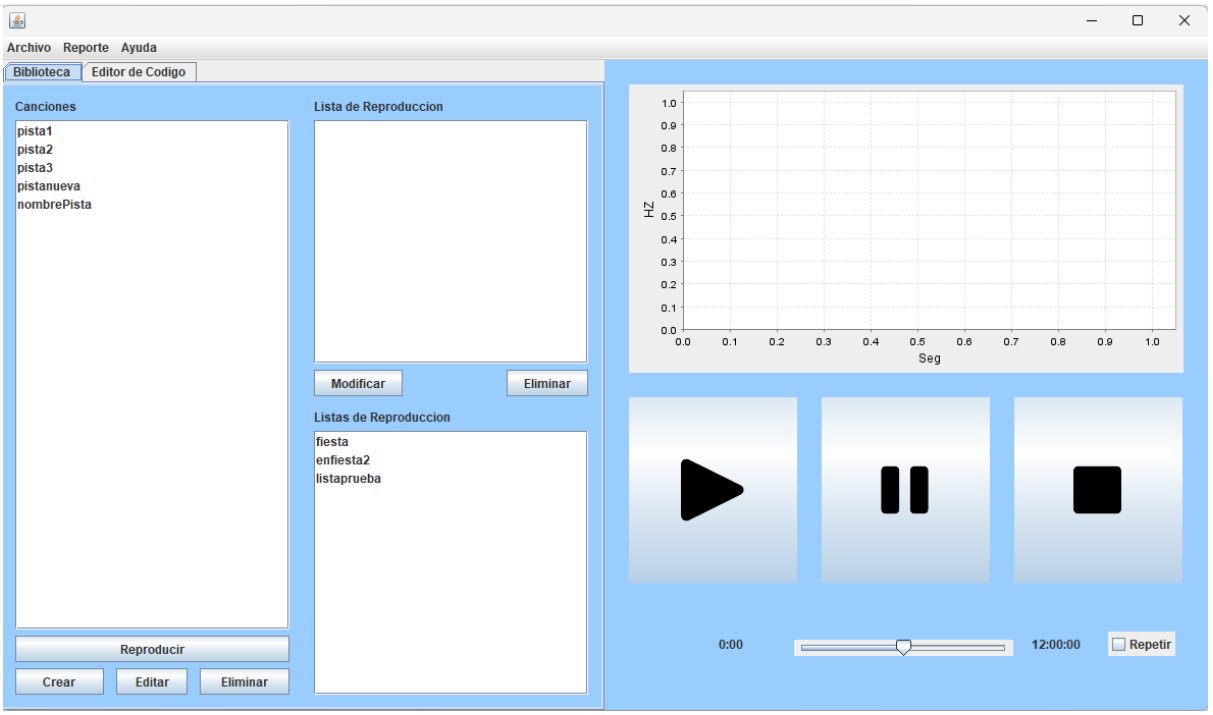
Jar ejecutable o proyecto completo.

Conocimientos de Programación en Kotlin

Descripción

El programa es un editor de código y reproductor de sonidos, el cual genera los sonidos a partir de un lenguaje de entrada.

Ventana de Reproducción



Consta de 4 secciones

1. **Sección de canciones**
    a. En esta sección se mostrará la lista de canciones agregadas.
    b. Contiene 4 botones los cuales se encarga uno de reproducir una canción seleccionada, otro de crear una pista y otro de editar, por último eliminar, cada uno de estos se encarga de controlar pistas.
2. **Sección de Listas de Reproducción**
    a. Muestra el listado de listas de reproducción
    b. Contiene dos botones, uno se encarga de modificar y otro de eliminar
3. **Sección de de Lista de Reproducción**

a. Muestra el listado de listas de canciones de la lista seleccionada reproducción

b. Contiene dos botones, uno se encarga de modificar y otro de eliminar

4. **Sección de reproducción**

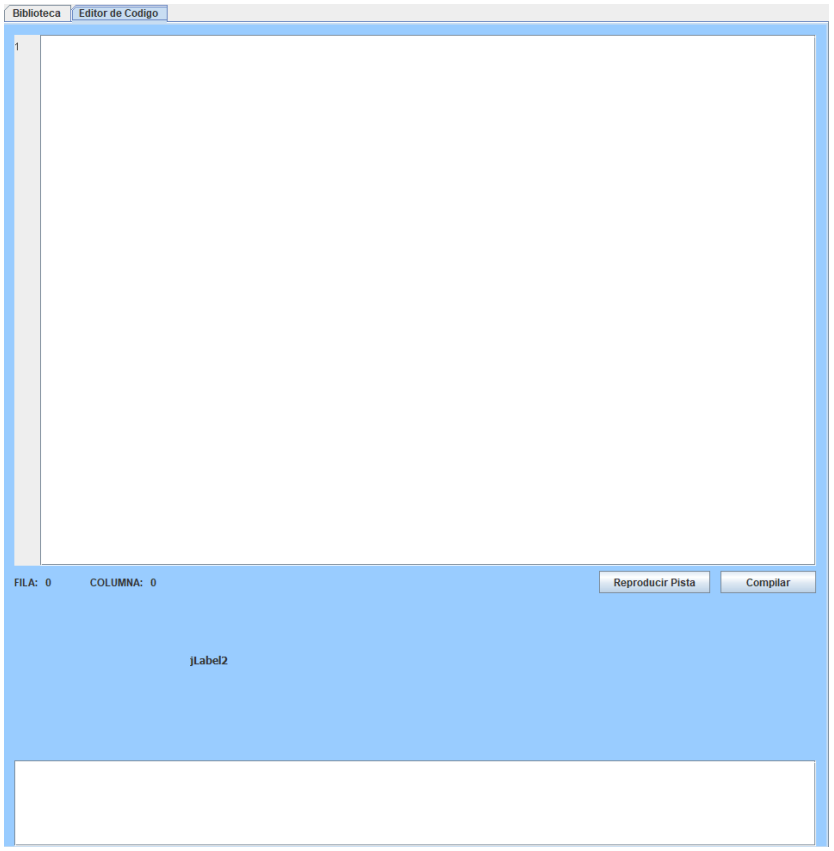
a. Muestra las opciones de reproducción

b. Contiene 3 botones, un play, pause, stop

c. Contiene un seleccionador de repetir pista y un slider de la canciones

d. Muestra una imagen de las frecuencias de la pista.

Ventana de Editor de Código



Contiene una sección de edición de código, una sección de reporte de errores, una sección de fila y columna.

Contiene dos botones el cual uno compila y otro se encarga de reproducir en caso de que se haya compilado correctamente.

Lenguaje

Comentarios de una línea

Estos deben empezar con dos signos “mayor que” y terminar con un salto de línea:

```
>>comentario de una sola línea
```

Comentarios de bloque

Estos deben empezar con un signo de “menor que” y un guión y termina con un guión y un signo de “mayor que”:

```
<- comentario de varias líneas
Segunda línea
Tercera línea
...
->
```

Pista

La sintaxis para la declaración de una pista es la siguiente, en donde debe de establecerse el nombre de la pista y también si se extiende de otra pista. Extender de otra pista es opcional para cada pista que se vaya a crear.

```
Pista FragmentoCoro {
     >>cuerpo de la pista
     ...

}
```

Extender una Pista

Cada pista tendrá la posibilidad de poder extender atributos, métodos y funciones de otra pista indicando el nombre de la pista de la cual se desea obtener toda su estructura. Al momento de extender una pista, se deberá buscar la pista padre en el archivo binario de pistas cargadas y compiladas para poder extender la pista hija (función similar al “import” tradicional).

```
Pista Coro extiende Ritmo, FramgentoCoro {
     >>cuerpo de la pista
     ...

}
```

Indentación de sentencias y definición de bloques

Toda sentencia termina siempre con un salto de línea y el cuerpo de toda la pista deberá tener una jerarquía de indentación con tabulación. Por lo que dentro de la pista el cuerpo de esta debe de tener un nivel más a la derecha de indentación. En las estructuras de control esta jerarquía debe de mantenerse, de lo contrario se tomará como un error sintáctico.

Ejemplo

```
Pista Coro extiende Ritmo, FramgentoCoro {
      >>Sentencia 1;
      >>Sentencia 2;
      >>Ciclo 1 {
            >>Sentencia 1;
            >>Sentencia 2;
            >>Ciclo 1_1 {
                  >>Sentencia 1;
                  >>Sentencia 2;
                  ...
            }
            >>Sentencia 3;
            ...
      }
      >>Sentencia n;

}
```

Tipos de datos

Los tipos que el lenguaje deberá soportar son los siguientes

Nombre	Descripción	Ejemplos	Observaciones
entero	Este tipo de dato acepta solamente números enteros	1, 50, 100, 255125	No tiene límite de tamaño
doble	Es un entero con decimal	1.2, 50.23, 0.34	Se manejara como regla general el manejo de 6 digitos despues del punto decimal
boolean	Admite valores de verdadero o falso, y variantes de los mismos	**verdadero**, **falso**, **true**, **false**. 1=verdadero, 0=falso	si se asigna los enteros 1 o 0 debe aceptar como verdadero o falso según el ejemplo
caracter	Solo admite un carácter por lo que no será posible ingresar cadenas enteras. Viene encerrado en comilla simple	'a', 'b', '#t', ')', '''	En el caso de querer escribir comilla simple se escribirá primero # (caracter de escape) y
	entre a-z, A-Z, -, _, : incluyendo símbolos y caracteres especiales #t, #n		después la comilla simple '#'', además si se quiere escribir # se pondrá 2 veces "##", los caracteres especiales se escriben siempre '#t' y '#n'
cadena	Este es un conjunto de caracteres que pueden ir separados por espacios en blanco encerrados en comilla doble, los posibles caracteres a ingresar serán los mismos que los del tipo caracter solo que agregando a estos el espacio	"cadena1", "est##$%o es una ca&/(de=na #n#n!!!"	En el caso de querer escribir comilla doble se escribirá primero # y después la comilla doble "ho#"la mundo #"", además si se quiere escribir # se pondrá 2 veces"##", por ejemplo "ho#la mundo ##", los caracteres especiales se escriben "#t" y "#n".

Operadores relacionales

Son los símbolos utilizados en expresiones booleanas, su finalidad es comparar expresiones entre sí dando como resultado booleanos. En el lenguaje serán soportados los siguientes:

Nombre	Símbolo	Ejemplos	Descripción
Igualación	==	1==1, "hola"=="hola", 25.2555==80.051	Compara ambos valores y verifica si son iguales
Diferenciación	!=	1!=2, martes!=vari1	Compara ambos lados y verifica si son distintos
Mayor que	>	(5+5.5)>8.98	Compara ambos lados y verifica si el izquierdo es mayor que el derecho izquierdo.
Menor que	<	(5/(5+5))<(8*8)	Compara ambos lados y verifica si el derecho es mayor que el izquierdo
Mayor o igual que	>=	5-6>=4+6	Compara ambos lados y verifica si el izquierdo es mayor o igual que el derecho
Menor o igual que	<=	55+66<=44	Compara ambos lados y verifica si el derecho es mayor o igual que el izquierdo
Es nulo	!!	!!var1	Verifica si la variable fue declarada pero no le fue asignado algún valor inicial o una asignación posterior

Operadores Lógicos

Nombre	Símbolo	Evento	Ejemplo	Observaciones
AND	&&	Compara expresiones lógicas y si son ambas verdaderas entonces devuelve verdadero en otro caso retorna falso	(flag1) && (“hola” == ”hola”) RESULTADO ES FALSO	flag1 es falso
NAND	!&&	Compara expresiones lógicas y si son ambas verdaderas entonces devuelve falso en otro caso retorna verdadero	(flag2) && (“hola” == ”hola”) RESULTADO ES FALSO	flag1 es verdadero
OR	||	Compara expresiones lógicas y si al menos una es verdadera entonces devuelve verdadero en otro caso retorna falso	(55.5<4) || (!!bandera) RESULTADO ES VERDADERO	bandera es null
NOR	!||	Compara expresiones lógicas y si al menos una es verdadera entonces devuelve falso en otro caso retorna verdadero	(band1) !|| (!!bandera) RESULTADO ES VERDADERO	band1 es falso bandera no es nulo
XOR	&|	Compara expresiones lógicas si al menos una es verdadera, pero no ambas entonces devuelve verdadero en otro caso retorna falso	(44+6!=4) &| (1+2>60) RESULTADO ES VERDADERO	
NOT	!	Devuelve el valor inverso de una expresión lógica	!(!!var1) RESULTADO ES FALSO	var1 es nulo

Operadores Aritméticos

+, -, *, /, %, ^

Declaración de variables

La declaración puede hacerse desde cualquier parte del código ingresado, pudiendo declararlas desde ciclos, métodos, funciones o fuera de estos siempre dentro de una Pista. La forma normal en que se declaran las variables es la siguiente:

[Keep] var Tipo nombre[,nombre2,nombre3…. , nombre n] [Asignación]

Lo que está encerrado en corchetes es opcional puede no venir.

Keep solo se acepta a nivel de variables globales de la pista.

```
Pista Coro extiende Ritmo, FramgentoCoro {
      Keep var doble contador = 50.5;
      var cadena palabra2,palabra3 = "esto es un ejemplo :D #n"+"v";
      var boolean bandera2, bandera3 =true;
      Keep var boolean flag2,flag3=!!contador;
      var boolean flag4=flag2;
      Keep var caracter letra="A";

}
```

Asignación de variables

```
...
numero1 = 25
numero1 = true
contador = 50.5 * 20+115*b
```

Manejo de arreglos

```
...
var entero arreglo arr0 []={5,10,15+varEntera};
var entero arreglo arr10,arr20,arr30 [][]={{5,10,15},{20,25,30}};
var entero arreglo arr1 [][]={{5,10,15},{1,3,8},{20,25,30}};
var entero arreglo arr1 [][]={{5,10},{1,3,8}};
```

Asignación de valores en arreglos

```
...
arr0[2] = 25 + 35 + 'a'
arr1 [(a+(5+1)%5)] = arr0[2]+5*8+b
arr4 [5+arr1[5]] [5*arr1[4]+5*arr1[5]-4] = arr1[(a+5)%2]
```

Condicional Si - sino si - sino

```
si(condición) {
    >> Sentencia 1
    >> Sentencia 2
    >> Sentencia 3
    ...
    >> Sentencia n

}
```

Switch

```
switch(variable) {
    caso valor1:
       >> Sentencias caso 1
       salir
    caso valor 2:
       >> Sentencias caso 2
       salir
    caso valor 3:
       >> Sentencias caso 3
       salir
    ...
    default:
       >> Sentencias default
```

Para

```
para(asignación; condición; incremento|disminución o acción posterior) {
    >> Sentencia 1
    >> Sentencia 2
    >> Sentencia 3
    ...
    >> Sentencia n

}
```

Mientras

```
mientras(condición) {
    >> Sentencia 1
    >> Sentencia 2
    >> Sentencia 3
    ...
    >> Sentencia n

}
```

Hacer - Mientras

```
hacer {
    >> Sentencia 1
    >> Sentencia 2
    >> Sentencia 3
    ...
    >> Sentencia n
}mientras(condición)
```

Funciones y procedimientos

```
[keep] [Tipo] nombre([tipo parametro1, tipo parametro2,...tipo parametro n]) {
    >> Sentencia 1
    >> Sentencia 2
    ...
    >> Sentencia n
    [retorna valor]

}
```

Funciones y procedimientos nativos del lenguaje

Reproducir

Esta función agrega a un determinado canal de la pista, la nota y su respectiva octava definida con valor de tipo entero la cual solo puede estar entre 0 y 8 (incluyentes), y la cantidad determinada de milisegundos que será reproducida la nota.

```
Reproducir(Do, (2*1)+1, 50*arr0[3]+62*6*b, 3);
entero tiempo = Reproducir(Re, 6, 15000, 1);
```

Esperar

Esta función tiene como finalidad pausar el tiempo entre reproducciones continuas y entre sentencias de código. Los parámetros son la cantidad de milisegundos definido como entero y también el canal en el cual se genera la espera. En el canal no se reproducirá nada en este tiempo pero forma parte del tiempo total del canal.

```
Esperar(2000, 8);
```

Ordenar

El método ordenar recibe como parámetro un arreglo junto con un parámetro para indicar de qué manera se quiere ordenar, si el ordenamiento tuvo éxito retorna un valor de 1, si la La forma de ordenar no es aceptada o no se realizó el ordenamiento retorna 0.

Ordenar (<arreglo a ordenar>, <forma de ordenar>)

Sumarizar

El método sumarizar extrae la suma de todos los elementos del arreglo, recibe como parámetro un arreglo. El tipo de dato devuelto es cadena.

```
var entero arreglo arr0 [3]={5,10,15};
var caracter arreglo arr1 [4]={"a","b","F","5"};
var doble arreglo arr3 [2]={1.5,2.6};
var cadena cad;
cad=Sumarizar(arr0); >> valor de cad: "30"
cad=Sumarizar(arr1); >> valor de cad: "abF5"
cad=Sumarizar(arr3); >> valor de cad: "4.1"
cad=Sumarizar({"hola"," ","mundo"}); >> valor de cad: "hola mundo"
```

Longitud

El método devuelve el tamaño del arreglo o de la cadena introducida, el tipo de dato devuelto es un entero.

Longitud (<arreglo a medir>/<cadena a medir>)

```
var entero arreglo arr0 [3]={5,10,15};
var caracter arreglo arr1 [4]={"a","b","F","5"};
var entero num1;
num1=Longitud(arr0); >> valor de num1: 3
num1=Longitud(arr1); >> valor de num1: 4
num1=Longitud("mundo"); >> valor de num1: 5
```

Mensaje

El método despliega el mensaje en un área de log dentro del servidor.

Mensaje (<texto>)

```
var entero num1 = 22;
var cadena cad1="Compiladores 2";

Mensaje(cad1);  >>Mostrará: Compiladores 2
Mensaje(“Proyecto "+1);  >>Mostrará: Proyecto 1
Mensaje(cad1+" "+num1);  >>Mostrará: Compiladores 2 22
```

Método Principal de Pista

Éste método es el que el intérprete buscará al momento de reproducir la canción, a partir de este método el usuario puede llamar a otros métodos o funciones en donde define las sentencias necesarias para poder crear una Pista con sus respectivas notas anidadas en los distintos canales que en la pista se definan.

```
Principal() {
    >> Sentencia 1
    >> Sentencia 2
    .....

}
```

Lenguaje de definición de listas de reproducción

Las listas de reproducción serán definidas a partir de un lenguaje específico para ello, por lo tanto, el módulo servidor deberá contar con un editor y un analizador para este lenguaje. Una lista de reproducción se definirá de la siguiente manera:

```
{
    lista:
    {
       nombre: "<nombre de la lista de reproducción>",
       random: true|false, // opcional
       circular: true|false, // opcional
       pistas: [Pista1,Pista2,Pista3,...,Pista_n ] //opcional
    }
```

Los atributos random y circular también son opcionales, por defecto las pistas no serán ni aleatorias ni circulares. También se debe considerar que las pistas no pueden encontrarse más de una vez en una lista, por lo que la repetición de una pista dentro de una lista de reproducción deberá ser marcada como error.

Módulo cliente

El módulo cliente consistirá en una aplicación móvil para el sistema operativo Android, la cual podrá realizar peticiones al módulo servidor de las cuales dependerá su funcionamiento. Ésta aplicación podrá reproducir las pistas almacenadas por el servidor, así como obtener las diversas listas de reproducción.

Lenguaje de comunicación

Solicitud de listas de reproducción

Solicitud: nombre opcional

```
<solicitud>
    <tipo>Lista</tipo>
    <nombre>"nombre_lista"</nombre>
</solicitud>
```

Lista de Listas

```
< listas >
< lista nombre = "tranquilas" pistas = 5>
< lista nombre = "fiesta" pistas = 10>
......
</ listas >
```

Lista de Pistas

```
< lista nombre = "tranquilas" aleatoria ="SI/NO">
< pista nombre = "pista1" duracion = 14500 >
< pista nombre = "pista2" duracion = 60000 >
......
</ lista >
```

Solicitud de pistas

```
<solicitud>
    <tipo>Pista</tipo>
    <nombre>"nombre_pista"</nombre>
</solicitud>
```

Lista de Pistas

```
< pistas >
    < pista nombre = "pista1" duracion = 14500 >
    < pista nombre = "one minute" duracion = 60000 >
    ......
</ pistas >
```

```
< pista nombre = "pista1" >
    < canal numero = 1 >
       < nota duracion = 4000 frecuencia = 32.7 > //do octava 1 4 seg
       < nota duracion = 3500 frecuencia = 73.4 > //re octava 2 3.5 seg
       < nota duracion = 7000 frecuencia = 0 > //esperar 7 seg
    .....
    </ canal >
    < canal numero = 8 >
       < nota duracion = 3500 frecuencia = 73.4 >
       < nota duracion = 7000 frecuencia = 0 >
       < nota duracion = 7000 frecuencia = 0 >
```

Creacion de pistas

Esta función se llevará a cabo mediante el uso del teclado musical el cual capturará la frecuencia de la nota tocada y la reproducirá mientras la nota se sostenga durante un periodo x de tiempo.

```
<solicitud>
    <tipo>pistanueva</tipo>
    <datos>
       <canal >1</canal>
       <nota>Do#</nota>
       <octava>2</octava>
       <duracion>1000</duracion>
    </datos>
    <datos>
       <canal>2</canal>
       ....
    </datos>
    .....
</solicitud>
```