PROYECTO

201801539 – Brenda Paola Gramajo Paniagua

202100692- Josue Ricardo Carias Ordoñez

201900081 - José Luis Saloj Julajuj

Resumen

El proyecto realizado se basó en la búsqueda de una solución al problema que se planteó, mediante la lógica y conocimientos que se han adquirido durante la clase y que han sido aplicados en el laboratorio. Este problema consistió en la elaboración de un reproductor de música. El programa fue desarrollado en el lenguaje de programación de Python y se utilizaron Listas Enlazadas para la creación del algoritmo y almacenamiento de datos. El problema planteado representa funciones que se pueden observar en el medio en el que vivimos, y la solución se puede utilizar como recurso de optimización en los casos similares, al tener un programa que realice tareas como estas, se mejora el entorno de trabajo y, por lo tanto, la producción. La solución planteada es la muestra de conocimientos adquiridos y es preparación para los problemas que se puedan presentar en un futuro.

Palabras clave

Listas, Nodo, Música, Canciones, Circular

Abstract

The project carried out was based on the search for a solution to the problem that was raised, through the logic and knowledge that have been acquired during the class and that have been applied in the laboratory. This problem consisted of the development of a console program that will consist of implementing a grouping methodology for the solution to achieve compressing audio signals. The program was developed in the Python programming language and linked lists were used for

algorithm creation and data storage. The problem posed represents functions that can be observed in the environment in which we live, and the solution can be used as a resource for optimization in similar cases, having a program that performs tasks like these, improves the working environment and therefore, the production. The proposed solution is a sample of acquired knowledge and a preparation for future problems.

Keywords

Lists, Node, Music, Songs, Time

Introducción

El proyecto por realizar describe la solución para construir un programa en lenguaje Python que consiste en un reproductor de música el cual cuenta con una interfaz de usuario amigable e intuitiva que permitirá al usuario ordenar la música que encuentra en su ordenador, así como mostrar estadísticas con relación a sus canciones. El programa realizado podrá realizar varias operaciones como crear las listas de reproducción que desee, podrá seleccionar las canciones por agregar para luego darle un nombre a la lista, cargar un archivo XML que contenga la información de la biblioteca, entre otras operaciones. Una de las principales funciones es la carga de archivos XML, así como el uso de estructuras Lineales creadas personalmente para el almacenamiento de la información, esto implementando nodos y listas simples enlazadas y listas circularmente enlazadas. También se implementaron soluciones de forma gráfica con la herramienta de graphviz y por último se implementaron reportes en formato HTML. Al combinar todas estas herramientas se logró implementar y dar una solución al problema planteado.

Desarrollo del tema

El desarrollo del proyecto consistió en dos partes importantes, la primera parte se desarrolló la lectura del archivo de entrada XML, y la extracción de los datos contenidos en el documento. Toda la segunda parte se centró en la creación de las respectivas clases para trabajar con la POO y los TDA´s necesarios para brindarle una solución al problema.

Se creó una clase “Canciones” donde se almacenarían los atributos de cada una de los canciones cargados en el sistema, como la lista de instrucciones para emitir el puntero, que almacenaba los atributos respectivos de cada Sistema individual, así como su lista de caracteres.

También se implementó para la solución la creación de una clase Nodo, en la que se tenían los atributos necesarios para la creación de varios Nodos que se incluirían en la clase de Lista Enlazada y una lista Doblemente Enlazada. La clase de Lista enlazada contenía las funciones necesarias para la creación de las listas, como puede ser la función insertar, buscar, recorrer, etc. Se crearon los respectivos objetos 'Mensaje' extrayendo los datos del archivo XML brindado por el usuario y se almacenaron en el respectivo Nodo de la Lista Enlazada. Con esto se pudieron crear funciones como lo son el insertar nodos y el recorrer. Para la función de graficar, se utilizó la librería de Python llamada Graphviz. Esta se implementó como una solución practica para graficar los sistemas de drones cargados y la serie de instrucciones para emitir los mensajes. El proyecto incluye un menú en consola agradable al usuario. Este tiene funciones como lo son:

1. Cargar Archivo:

Función que permite cargar al usuario el archivo XML solicitando la ruta del archivo.

2. Generar XML de salida:

Genera un archivo XML de salida con las lista de cancion escuchas

3. Gestionar canciones:

Muestra gráficamente el listado de canciones.

4. Gestión de ListaDeCanciones:

Esta opción permite ver el listado de canciones

5. Inicializar Sistema:

Esta opción permite que el sistema pueda inicializarse sin información previa.

6. Salir:

Termina la ejecución del programa

Figura 1. Menú en consola.

Fuente: elaboración propia.

El programa se visualiza a través de una interfaz creada con Tkinter además se desarrolló con base de Programación Orientada a Objetos y la aplicación de las Listas Enlazadas Dobles y Circulares.

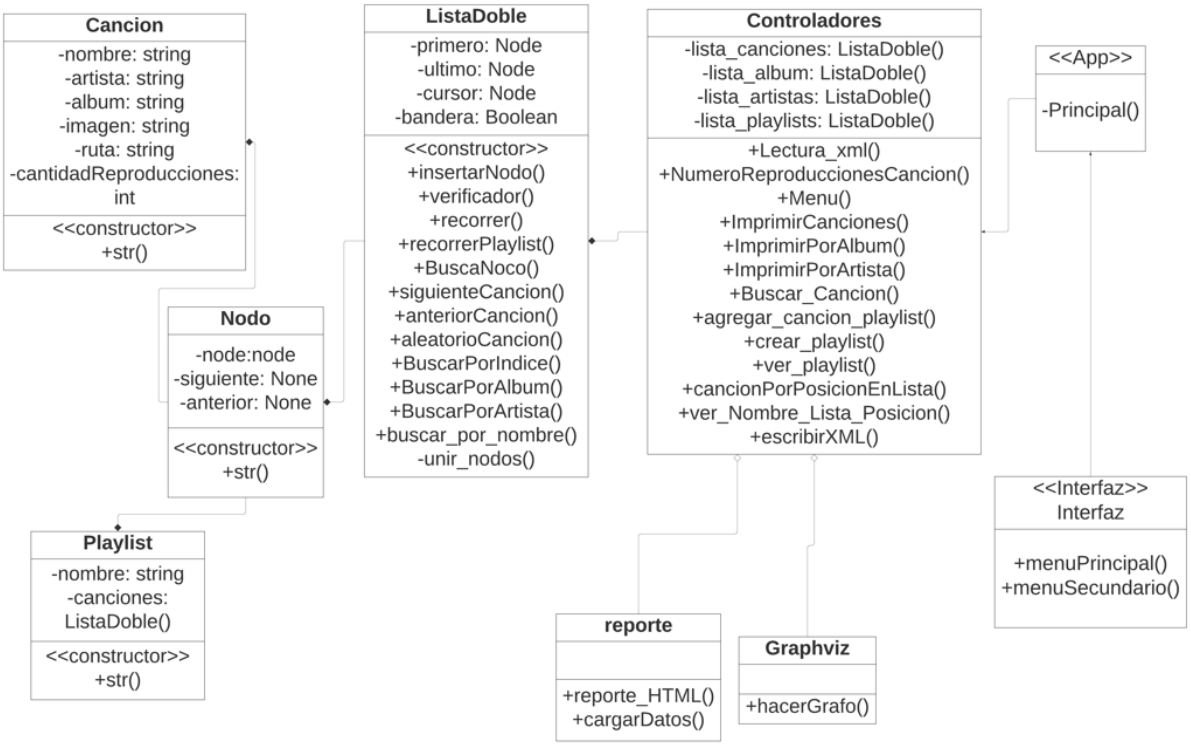


Figura 2. Diagrama de Clases.

Fuente: elaboración propia.

Conclusiones

- La programación orientada a objetos se basa en el concepto de crear un modelo del problema de destino en sus programas. La programación orientada a objetos disminuye los errores y promociona la reutilización del código. Python es un lenguaje orientado a objetos.

- Las listas enlazadas funcionan mediante la conexión de nodos en una secuencia lineal, donde cada nodo contiene un valor y una referencia (puntero) al siguiente nodo en la secuencia. Estas listas son dinámicas y eficiente.

  La memoria dinámica se refiere a la asignación y liberación de memoria en tiempo de ejecución según sea necesario esto genera que sea eficiente y efectiva para el almacenamiento.

- Los archivos XML implementan una forma de almacenar datos de manera legible y jerárquica y pueden albergar diferentes tipos de datos.

Anexos

Clase app.py:

```python
from LecturaXml import *
from reporte import reporte_HTML


def Principal():
    Menu()
    opcion = int(input ("\nIngrese el número de la acción que desea realizar:\n"))
    while 1<= opcion and opcion <=12:
        if opcion == 1:
            Lectura_xml("Entrada.xml")
        elif opcion == 2:
            ImprimirCanciones()
        elif opcion == 3:
            nombreAlbum = input("Ingrese el nombre del album para la clasificación:\n")
            ImprimirPorAlbum(nombreAlbum)
        elif opcion == 4:
            nombreArtista = input("Ingrese el nombre del artista para la clasificación:\n")
            ImprimirPorArtista(nombreArtista)
        elif opcion == 5:
            playlist = input('Ingrese nombre de la playlist a crear: >')
            crear_playlist(playlist)
        elif opcion == 6:
            play = input('Ingrese el nombre de la playlist: >')
            cancion = input('Ingrese el nombre de la cancion: ')
            agregar_cancion_playlist(play, cancion)
        elif opcion == 7:
            ver_playlist()
        elif opcion == 8:
            nombreCancion = input('Ingrese el nombre de la canción:\n')
            NumeroReproduccionesCancion(nombreCancion)
        elif opcion == 9:
            indice = int(input('Ingrese la posicion de la playlist a buscar:\n'))
            ver_unaLista_Posicion(indice)
        elif opcion == 10:
            indiceLista = int(input('Ingrese la posicion de la playlist a buscar:\n'))
            indiceCancion = int(input('Ingrese la posicion que ocupa la cancion en la lista:\n'))
            cancion = cancionPorPosicionEnLista(indiceLista,indiceCancion)
            print(f'Nombre: {cancion.nombre}')
            print(f'Album: {cancion.album}')
            print(f'Artista: {cancion.artista}')
            print(f'Número de reproducciones: {cancion.CantidadReproducciones}')
        elif opcion == 11:
            reporte_HTML()
        elif opcion == 12:
            escribirXML()
        elif opcion == 13:
            print("A salido del menú")
            break
        Menu()
```

Clase cancion.py:

```python
class Cancion:

    def __init__(self):
        self.nombre = None
        self.artista = None
        self.album = None
        self.imagen = None
        self.ruta = None
        self.CantidadReproducciones=0

    def __str__(self) -> str:
        return "{" + str(self.nombre) + ", " + str(self.artista) + ',' + str(self.album) + ',' + str(self.imagen) + ',' + str(self.ruta)
```

Clase Graficas.py:

```python
from graphviz import Digraph
from listaDobleEnlazada import *

from LecturaXml import *


def hacerGrafo():
    # Crear un gráfico DOT
    dot = Digraph(comment='Canciones')
    largoLista = int(lista_canciones.cantidadElementos())

    dot.node("TITULO", "Biblioteca")

    # Agregar nodos con forma ovalada para los nombres de las canciones
    for i in range(largoLista):
        cancion = lista_canciones.BuscarPorIndice(i)
        dot.node(str(i), str(cancion.nombre))

    # Agregar bordes entre los nodos de los nombres de las canciones
    for i in range(largoLista - 1):
        dot.edge(str(i), str(i + 1))

    # Agregar nodos con forma cuadrada para los álbumes
    for i in range(largoLista):
        cancion = lista_canciones.BuscarPorIndice(i)
        dot.node(str(largoLista + i), str(cancion.album))

    # Agregar bordes entre los nodos de los álbumes
    for i in range(largoLista - 1):
        dot.edge(str(largoLista + i), str(largoLista + i + 1))

    for i in range(largoLista):
        cancion = lista_canciones.BuscarPorIndice(i)
        dot.node(str(largoLista*2 + i), str(cancion.artista))

    for i in range(largoLista - 1):
        dot.edge(str(largoLista*2 + i), str(largoLista*2 + i + 1))

    dot.edge("TITULO","0")
    dot.edge("TITULO",str(largoLista*1))
    dot.edge("TITULO",str(largoLista*2))

    # Generar el archivo DOT
    dot.render('productos_grafo', format='png', view=True)
```

Clase Interfaz.py:

```python
import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageTk
from LecturaXml import *
import os
from reporte import *
from Graficas import hacerGrafo

def menuPrincipal():
    # Crear la menu principal
    menu = tk.Tk()

    # Establecer título y tamaño de la menu
    menu.title("IPCMusic ")
    menu.geometry("720x480")
    menu.resizable(False, False)
    menu.config(bg="gray")

    # frame donde estan los botones y la barra de busqueda
    frame = tk.Frame(menu, width=700, height=50, bg="lightgrey")
    frame.grid(column=0,row=0,padx=10, pady=10,)

    # boton para cargar el archivo
    btnArchivo = tk.Button(frame, text="Archivo",font="arial", command=menuSecundario)
    btnArchivo.grid(column=0,row=0,padx=5,pady=5)

    def hacerReporte():
        reporte_HTML()
        hacerGrafo()
        escribirXML()
```

```python
#boton para generar el reporte
btnReporte = tk.Button(frame, text="Reporte",font="arial", command=hacerReporte)
btnReporte.grid(column=1,row=0,padx=5,pady=5)

#textbox para buscar
txtBarra = tk.Text(frame, width=57, height=2)
txtBarra.grid(column=3,row=0,padx=5,pady=5)

#carga la ruta de la imagen de adelante
buscarImagen = 'Imgenes/buscar.png'
img = Image.open(buscarImagen)
buscar = ImageTk.PhotoImage(img)

#Funcion para buscar cancion por nombre
def buscarPorCancion():
    txtBuscar.delete('1.0', tk.END)
    txtBuscar.insert(tk.END, (formatoBuscarCancion(txtBarra.get("1.0", "end-1c"))))
```

```python
def cargar_imagen(ruta):
    img = None  # Inicializa img con un valor predeterminado
    if os.path.exists(ruta):
        img = Image.open(ruta)
    else:
        print("La imagen no se encontró. Cargando imagen por defecto...")
        ruta_por_defecto = os.path.join('Imgenes', 'Imgenes\default.png')
        if os.path.exists(ruta_por_defecto):
            img = Image.open(ruta_por_defecto)
        else:
            print("La imagen por defecto no se encontró.")

    if img is not None:
        Album = ImageTk.PhotoImage(img)
        label.config(image=Album)
        label.image = Album  # Actualizar la referencia para evitar que la imagen sea eliminada por el recolector de basura
    else:
        # Manejar el caso en el que no se ha podido cargar ninguna imagen
        print("No se pudo cargar ninguna imagen.")

#carga la ruta de la imagen de play
playImagen = 'Imgenes\playPequeño.png'
img = Image.open(playImagen)
play = ImageTk.PhotoImage(img)

#Funcion para dar formato a la infomacion
def textoDeInformacion():
    dato=lista_canciones.BuscarPorIndice(0)
    salida="Nombre: "+dato.nombre+"\n"
    salida+="Artista: "+dato.artista+"\n"
    salida+="Album: "+dato.album+"\n"
    txtBuscar.delete('1.0', tk.END)
    txtBuscar.insert(tk.END, (str(salida)))
    print(dato.ruta+"--------")
    cargar_imagen((dato.imagen))
```

```python
#funcion encargada de dar formato a la informacion de salida
def formatoBuscarCancion(nombre):
    datos = lista_canciones.buscar_por_nombre(nombre)
    if datos is not None:
        salida = "Nombre: " + datos.nombre + "\n"
        salida += "Artista: " + datos.artista + "\n"
        salida += "Album: " + datos.album + "\n"
    else:
        salida = "La canción no se encontró"
    return salida


#boton para generar el buscar
btnBuscar = tk.Button(frame, image=buscar,text="Reporte",font="arial", command=buscarPorCancion)
btnBuscar.grid(column=4,row=0,padx=5,pady=5)

#frame de abajo, contiene las imagnes y los botones
frameReproductor = tk.Frame(menu, width=800, height=100, bg="lightgrey")
frameReproductor.grid(column=0,row=1,padx=10)

#carga la ruta de la imagen
albunImagen = 'Imgenes\defaulAlbumPequeño.png'
img = Image.open(albunImagen)
Album = ImageTk.PhotoImage(img)

#es la imagen del album
label = tk.Label(frameReproductor, image=Album, height=250,width=250)
label.grid(column=0,rowspan=5,row=0,padx=10,pady=10)
```

```python
#crea el boton de play
btnPlay = tk.Button(frameReproductor,image=play,font="arial", command=textoDeInformacion, width=60,height=60,)
btnPlay.grid(column=2,row=0,padx=2,pady=2)

#carga la ruta de la imagen de pausa
pauseImagen = 'Imgenes\pausaPequeño.png'
img = Image.open(pauseImagen)
pause = ImageTk.PhotoImage(img)

#crea el boton de pausa
btnPausa = tk.Button(frameReproductor,image=pause,font="arial", command="LLamar a la funcion pausa", width=60,height=60)
btnPausa.grid(column=3,row=0,padx=2,pady=2)

#carga la ruta de la imagen de detener
detenerImagen = 'Imgenes\detenerPqueño.png'
img = Image.open(detenerImagen)
detener = ImageTk.PhotoImage(img)

#crea el boton de detener
btnDetener = tk.Button(frameReproductor,image=detener,font="arial", command="LLamar a la funcion detener", width=60,height=60)
btnDetener.grid(column=4,row=0,padx=2,pady=2)

#textbox para informacion
txtBuscar = tk.Text(frameReproductor, width=45, height=10)
txtBuscar.grid(column=1,columnspan=5,row=1,rowspan=3,padx=10,pady=5)

#carga la ruta de la imagen de atras
atrasImagen = 'Imgenes/atras.png'
img = Image.open(atrasImagen)
atras = ImageTk.PhotoImage(img)

#Funcion para adelantar cancion
def atrasarCancion():
    txtBuscar.delete('1.0', tk.END)
    txtBuscar.insert(tk.END, (str(formatoAtrasar())))
```

```
#crea el boton de atras
btnAtras = tk.Button(frameReproductor,image=atras,font="arial", command=atrasarCancion, width=60,height=60)
btnAtras.grid(column=2,row=4,padx=2,pady=2)

#carga la ruta de la imagen de aleatorio
aleatorioImagen = 'Imgenes/aleatorio.png'
img = Image.open(aleatorioImagen)
aleatorio = ImageTk.PhotoImage(img)


#Funcion para cancion aleatoria
def cancionAleatoria():
    txtBuscar.delete('1.0', tk.END)
    lista_canciones.aleatorioCancion()
    txtBuscar.insert(tk.END, (str("Se ha activado el modo aleatorio")))

#crea el boton de aleatorio
btnAleatorio = tk.Button(frameReproductor,image=aleatorio,font="arial", command=cancionAleatoria, width=60,height=60)
btnAleatorio.grid(column=3,row=4,padx=2,pady=2)

#carga la ruta de la imagen de adelante
adelanteImagen = 'Imgenes/adelante.png'
img = Image.open(adelanteImagen)
adelante = ImageTk.PhotoImage(img)

#Funcion para adelantar cancion
def adelantarCancion():
    txtBuscar.delete('1.0', tk.END)
    txtBuscar.insert(tk.END, (str(formatoAdelantar())))
```

```
#funcion encargada de dar formato a la informacion de salida
def formatoAdelantar():
    datos=lista_canciones.siguienteCancion()
    salida="Nombre: "+datos.nombre+"\n"
    salida+="Artista: "+datos.artista+"\n"
    salida+="Album: "+datos.album+"\n"
    cargar_imagen((datos.imagen))
    return salida

#crea el boton de adelante
btnAdelante = tk.Button(frameReproductor,image=adelante,font="arial", command=adelantarCancion, width=60,height=60)
btnAdelante.grid(column=4,row=4,padx=2,pady=2)


# Mostrar la menu
menu.mainloop()
```

```
def menuSecundario():
    menu = tk.Tk()

    # Establecer título y tamaño de la menu
    menu.title("IPCMusic ")
    menu.geometry("720x480")
    menu.resizable(False, False)
    menu.config(bg="gray")

    frameDeArriba = tk.Frame(menu, width=700, height=50, bg="lightgrey")
    frameDeArriba.grid(padx=10, pady=10)

    # funcion para buscar el archivo usando windows
    def open_file_dialog():
        archivo = filedialog.askopenfile(filetypes=[("Archivos de texto", "*.xml"), ("Todos los archivos", "*.*")])

        if archivo:
            ruta=archivo.name
            Lectura_xml("Entrada.xml")
            lista_canciones.recorrer()

    # boton para cargar el archivo
    btnArchivo = tk.Button(frameDeArriba, text="Abrir Archivo",font="arial", command=open_file_dialog)
    btnArchivo.grid(column=0,row=0,padx=5,pady=5)

    def verLasPlayList():
        txtVerPlaylist.delete('1.0', tk.END)
        txtVerPlaylist.insert(tk.END, (str(ver_playlist())))

    # boton para ver Playlist
    btnVerPlaylist = tk.Button(frameDeArriba, text="Ver Playlists",font="arial", command=verLasPlayList)
    btnVerPlaylist.grid(column=1,row=0,padx=5,pady=5)

    #frame de abajo, contiene las imagnes y los botones
    frameReproductor = tk.Frame(menu, width=700, height=100, bg="lightgrey")
    frameReproductor.grid(column=0,row=1,padx=10)
```

```python
labelNombreLista=tk.Label(frameReproductor,text="Ingrese el nombre de la playList", bg="lightgrey")
labelNombreLista.grid(column=3,row=0,padx=5,pady=5)

#textbox para buscar
txtNombrePlaylist = tk.Text(frameReproductor, width=40, height=2)
txtNombrePlaylist.grid(column=4,row=0,padx=5,pady=5)

#textbox para informacion
txtVerPlaylist = tk.Text(frameReproductor, width=80, height=5)
txtVerPlaylist.grid(column=0,columnspan=5,row=1,rowspan=3,padx=10,pady=10)

def crearPlaylist():
    crear_playlist(str(txtNombrePlaylist.get("1.0", "end-1c")))
    txtNombrePlaylist.delete('1.0', tk.END)

#crea el boton de adelante
btnCrarLista = tk.Button(frameReproductor,text="Crear playlist",font="arial",command=crearPlaylist)
btnCrarLista.grid(column=4,row=4,padx=2,pady=2)

#frame de abajo, contiene las imagnes y los botones
frameMegusta = tk.Frame(menu, width=700, height=100, bg="lightgrey")
frameMegusta.grid(column=0,row=2,padx=10,pady=5)

#textbox para buscar
txtNombreCancion = tk.Text(frameMegusta, width=40, height=2)
txtNombreCancion.grid(column=4,row=0,padx=5,pady=5)
```

```python
def buscarPorCancion():
    txtInformacionDeLaCancionCreada.delete('1.0', tk.END)
    txtInformacionDeLaCancionCreada.insert(tk.END, (formatoBuscarCancion(str(txtNombreCancion.get("1.0", "end-1c")))))

def formatoBuscarCancion(nombre):
    datos = lista_canciones.buscar_por_nombre(nombre)
    if datos is not None:
        salida = "Nombre: " + datos.nombre + "\n"
        salida += "Artista: " + datos.artista + "\n"
        salida += "Album: " + datos.album + "\n"
    else:
        salida = "La canción no se encontró"
    return salida

#boton para generar el buscar
btnBuscarCancion = tk.Button(frameMegusta, text="Buscar Cancion", font="Arial",command=buscarPorCancion)
btnBuscarCancion.grid(column=3, row=0, padx=5, pady=5)

#textbox para informacion
txtInformacionDeLaCancionCreada = tk.Text(frameMegusta, width=80, height=5)
txtInformacionDeLaCancionCreada.grid(column=0,columnspan=5,row=1,rowspan=3,padx=10,pady=10)

def agregarCancionPlaylist():
    agregar_cancion_playlist(str(txtNombrePlaylist.get("1.0", "end-1c")), str(txtNombreCancion.get("1.0", "end-1c")))
    txtNombreCancion.delete('1.0', tk.END)
    txtInformacionDeLaCancionCreada.delete('1.0', tk.END)

#crea el boton de adelante
btnAgregarCancion = tk.Button(frameMegusta,text="Agregar Cancion",font="arial",command=agregarCancionPlaylist)
btnAgregarCancion.grid(column=4,row=5,padx=2,pady=2)

menu.mainloop()
```

Clase lecturaXML.py

```python
import xml.etree.ElementTree as ET
from listaDobleEnlazada import ListaDoble
from cancion import Cancion
from playlist import Playlist

lista_canciones = ListaDoble()
lista_album = ListaDoble()
lista_artista = ListaDoble()
lista_playlists = ListaDoble()

def Lectura_xml(ruta):
    raiz = ET.parse(ruta).getroot()


    for canciones in raiz.findall('cancion'):

        nuevaCancion = Cancion()

        nombreCancion = canciones.get('nombre')
        nuevaCancion.nombre = nombreCancion

        for artistaCancion in canciones.findall('artista'):
            artista = artistaCancion.text
            nuevaCancion.artista = artista

        for albumCancion in canciones.findall('album'):
            album = albumCancion.text
            nuevaCancion.album = album

        for imagenCancion in canciones.findall('imagen'):
            imagen = imagenCancion.text
            nuevaCancion.imagen = imagen

        for rutaCancion in canciones.findall('ruta'):
            ruta = rutaCancion.text
            nuevaCancion.ruta = ruta

        lista_canciones.insertarNodo(nuevaCancion)
```

```python
def NumeroReproduccionesCancion(nombreCancion):
    cancion = lista_canciones.buscar_por_nombre(nombreCancion)
    if cancion:
        print(f'La canción: {cancion.nombre} tiene: {cancion.CantidadReproducciones} reproducciones')
    else:
        print('La cancion no existe')

def Menu():
    print("==========Proyecto1_IPC_2=========="
          +"\n\t1. Cargar archivo xml"
          +"\n\t2. Ver todas las canciones"
          +"\n\t3. Clasificar por album"
          +"\n\t4. Clasificar por artistas"
          +"\n\t5. Crear playlist"
          +"\n\t6. Agregar canciones a la playlist"
          +"\n\t7. ver playlist"
          +"\n\t8. ver cantidad de reproducciones de una canción"
          +"\n\t9. ver una lista segun posicion"
          +"\n\t10. ver una cancion en una lista segun posicion"
          +"\n\t11. reporte de HTML"
          +"\n\t12. Generar XML"
          +"\n\t13. Salir")

def ImprimirCanciones():
    lista_canciones.recorrer() #Imprime la lista enlazada


def ImprimirPorAlbum(album):
    if lista_canciones.cantidadElementos()!=0:
        if lista_canciones.BuscarPorAlbum(album):
            for i in range(lista_canciones.cantidadElementos()):
                if album == lista_canciones.BuscarPorIndice(i).album:
                    lista_album.insertarNodo(lista_canciones.BuscarPorIndice(i))
                i = i + 1
            print()
            lista_album.recorrer()
        else:
            print("El album ingresado no existe")
    else:
        print("No hay elementos en la lista")
```

```python
def ImprimirPorAlbum(album):
    if lista_canciones.cantidadElementos()!=0:
        if lista_canciones.BuscarPorAlbum(album):
            for i in range(lista_canciones.cantidadElementos()):
                if album == lista_canciones.BuscarPorIndice(i).album:
                    lista_album.insertarNodo(lista_canciones.BuscarPorIndice(i))
                i = i + 1
            print()
            lista_album.recorrer()
        else:
            print("El album ingresado no existe")
    else:
        print("No hay elementos en la lista")

def ImprimirPorArtista(artista):
    if lista_canciones.cantidadElementos()!=0:
        if lista_canciones.BuscarPorArtista(artista):
            for i in range(lista_canciones.cantidadElementos()):
                if artista == lista_canciones.BuscarPorIndice(i).artista:
                    lista_artista.insertarNodo(lista_canciones.BuscarPorIndice(i))
                i = i + 1
            print()
            lista_artista.recorrer()
        else:
            print("El artista ingresado no existe")
    else:
        print("No hay elementos en la lista")


def Buscar_Cancion(cancion_buscada):
    cancion = lista_canciones.buscar_por_nombre(cancion_buscada)
    print('Cancion: ' + cancion.nombre)
    print('Artista: ' + cancion.artista)
    print('Album: ' + cancion.album)
    print('Imagen: ' + cancion.imagen)
    print('Ruta: ' + cancion.ruta)
    print('\n')
```

```python
def agregar_cancion_playlist(nombre_playlist, cancion):
    global lista_playlists
    global lista_canciones

    playlist = lista_playlists.buscar_por_nombre(nombre_playlist)
    cancion_encontrada = lista_canciones.buscar_por_nombre(cancion)

    playlist.canciones.insertarNodo(cancion_encontrada)



def crear_playlist(nombre):
    global lista_playlists

    nueva_playlist = Playlist()
    nueva_playlist.nombre = nombre

    lista_playlists.insertarNodo(nueva_playlist)


def ver_playlist():
    return lista_playlists.recorrer_playlist()


def ver_unaLista_Posicion(indice):
    print(lista_playlists.BuscarPorIndice(indice).nombre)
    lista_playlists.BuscarPorIndice(indice).canciones.recorrer()
    print('\n')


# Devuelve el nodo de la cancion
def cancionPorPosicionEnLista(indiceLista, indiceCancion):
    Lista = lista_playlists.BuscarPorIndice(indiceLista)
    cancion_En_Lista = Lista.canciones.BuscarPorIndice(indiceCancion)
    return cancion_En_Lista

def ver_Nombe_Lista_Posicion(indice):
    nombre = lista_playlists.BuscarPorIndice(indice).nombre
    return nombre
```

```python
#Escribe el XML
def escribirXML():
    global lista_playlists
    root = ET.Element("ListasReproduccion")

    actual = lista_playlists.primero
    while actual:
        lista = ET.SubElement(root, 'lista', attrib={"nombre": str(actual.node.nombre)})

        aux = actual.node.canciones.primero
        while aux:
            cancion = ET.SubElement(lista, 'cancion', attrib={"nombre": str(aux.node.nombre)})
            artista = ET.SubElement(cancion, 'artista')
            artista.text = aux.node.artista
            album = ET.SubElement(cancion, 'album')
            album.text = aux.node.album
            veces = ET.SubElement(cancion, 'vecesReproducida')
            veces.text = aux.node.CantidadReproducciones
            imagen = ET.SubElement(cancion, 'imagen')
            imagen.text = aux.node.imagen
            ruta = ET.SubElement(cancion, 'ruta')
            ruta.text = aux.node.ruta

            aux = aux.siguiente
            if aux == actual.node.canciones.primero:
                break

        actual = actual.siguiente
        if actual == lista_playlists.primero:
            break

    tree = ET.ElementTree(root)
    tree.write("listasReproduccion.xml")
```

```python
if __name__ == "__main__":
    leer_xml = Lectura_xml("Entrada.xml")
    #ImprimirCanciones()
```

Clase listaDobleEnlazada.py

```python
import random
from nodo import Nodo

class ListaDoble:

    def __init__(self):
        self.primero = None
        self.ultimo = None
        self.Cursor = None
        self.bandera = True


    def insertarNodo(self, node):
        if self.primero is None:
            self.primero = self.ultimo = self.Cursor = Nodo(node)
            # self.Cursor.node.CantidadReproducciones = 1 # Cuenta la primera cancion que se repoduce
        else:
            actual = self.ultimo
            self.ultimo = actual.siguiente = Nodo(node)
            self.ultimo.anterior = actual
        self .__unir_nodos()

    # Verifica la existencia de elemntos en una lista
    def verificador(self):
        actual = self.primero
        if self.primero != None:
            return True
        else:
            return False
```

```python
def recorrer(self):
    actual = self.primero
    if self.primero != None:
        while actual:
            print('Cancion: ' + actual.node.nombre)
            print('Artista: ' + actual.node.artista)
            print('Album: ' + actual.node.album)
            print('Imagen: ' + actual.node.imagen)
            print('Ruta: ' + actual.node.ruta)
            print('\n')
            actual = actual.siguiente
            if actual == self.primero:
                break
    else:
        print('La lista está vacía\n')


def recorrer_playlist(self):
    actual = self.primero
    salida=""
    while actual:
        print('Nombre de playlist: ' + actual.node.nombre)
        print('Canciones: ')
        aux = actual.node.canciones.primero
        salida+="-"+str(actual.node.nombre)+"\n"
        while aux:
            print(aux.node.nombre)
            aux = aux.siguiente
            salida+=str(aux.node.nombre)+"\n"
            if aux == actual.node.canciones.primero:
                break
        actual = actual.siguiente
        print('\n')
        if actual == self.primero:
            break
    return salida
```

```python
# Esta funcion sirve para la función aleatorio
def BuscaNodo(self, indice):
    contador = 0
    aux = self.primero
    if aux != None:
        while aux != None:
            if contador == indice:
                return aux
            else:
                aux = aux.siguiente
                contador = contador + 1
                if aux == self.primero:
                    return None
    return None

def siguienteCancion(self):
    if self.bandera:
        self.Cursor = self.Cursor.siguiente
        cancion = self.Cursor.node
    else:
        indice = random.randint(0, int(self.cantidadElementos())-1)
        self.Cursor = self.BuscaNodo(indice)
        cancion = self.Cursor.node
    cancion.CantidadReproducciones = int(cancion.CantidadReproducciones)+1 # Contador de número de reproducciones
    return cancion

def anteriorCancion(self):
    if self.bandera:
        self.Cursor = self.Cursor.anterior
        cancion = self.Cursor.node
    else:
        indice = random.randint(0, int(self.cantidadElementos())-1)
        self.Cursor = self.BuscaNodo(indice)
        cancion = self.Cursor.node
    cancion.CantidadReproducciones = int(cancion.CantidadReproducciones)+1 # Contador de número de reproducciones
    return cancion
```

```python
def anteriorCancion(self):
    if self.bandera:
        self.Cursor = self.Cursor.anterior
        cancion = self.Cursor.node
    else:
        indice = random.randint(0, int(self.cantidadElementos())-1)
        self.Cursor = self.BuscaNodo(indice)
        cancion = self.Cursor.node
    cancion.CantidadReproducciones = int(cancion.CantidadReproducciones)+1 # Contador de número de reproducciones
    return cancion

def aleatorioCancion(self):
    if self.bandera:
        self.bandera = False
        print('Entre en aleatorio')
    else:
        self.bandera = True
        print('He salido de aleatorio')
    return


def cantidadElementos(self):
    contador = 0
    aux = self.primero
    if aux != None:
        while aux != None:
            contador = contador + 1
            aux = aux.siguiente
            if aux == self.primero:
                return contador

def BuscarPorIndice(self, indice):
    contador = 0
    aux = self.primero
    while aux and contador != indice:
        aux = aux.siguiente
        contador += 1
    return aux.node if aux else None
```

```python
def BuscarPorArtista(self, artista):
    aux = self.primero
    if aux != None:
        while aux != None:
            if aux.node.artista == artista:
                return True
            else:
                aux = aux.siguiente
                if aux == self.primero:
                    return False
    return False

def buscar_por_nombre(self, nombre):
    if self.primero is None:  # Verificar si la lista está vacía
        return None

    actual = self.primero  # Empezar desde el primer nodo
    while actual:
        if actual.node.nombre == nombre:  # Verificar si el nombre coincide
            return actual.node  # Devolver el nodo si se encuentra el nombre

        actual = actual.siguiente  # Moverse al siguiente nodo
        if actual == self.primero:  # Si volvemos al inicio, hemos recorrido toda la lista
            break  # Romper el bucle para evitar un bucle infinito si no se encuentra el nombre

    return None

def __unir_nodos(self):
    if self.primero!=None:
        self.primero.anterior=self.ultimo
        self.ultimo.siguiente=self.primero
```

Clase nodo.py

```python
class Nodo:

    def __init__(self, node):
        self.node = node
        self.siguiente = None
        self.anterior = None

    def __str__(self) -> str:
        return str(self.node)
```

Clase playlist.py

```python
from listaDobleEnlazada import ListaDoble

class Playlist:

    def __init__(self):
        self.nombre = None
        self.canciones = ListaDoble()

    def __str__(self) -> str:
        aux = self.canciones.primero
        cancion = '\n'
        while aux:
            cancion += str(aux.node) + '\n'
            aux = aux.siguiente

        return "{" + str(self.nombre) + ", "  + cancion +  '}'
```

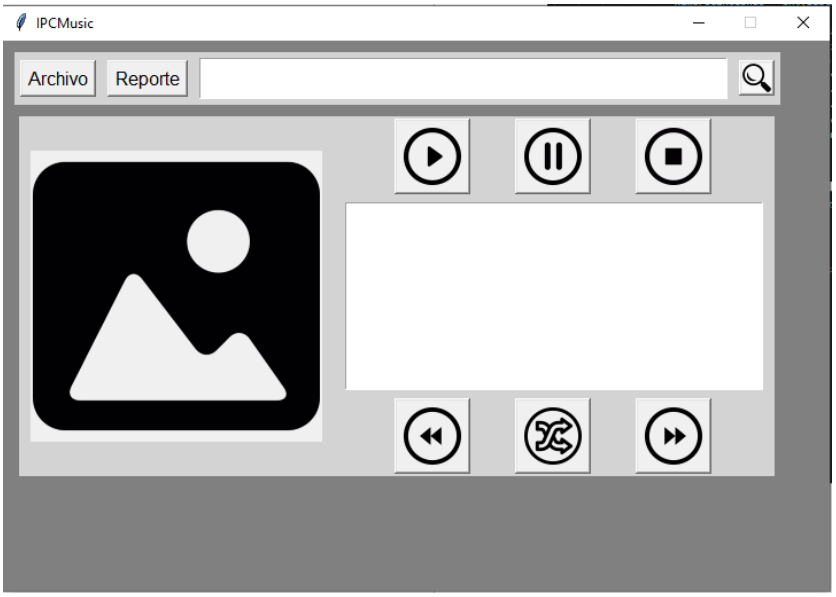
Clase reporte.py

```python
import os
from string import Template
from LecturaXml import *

def reporte_HTML():
    if lista_canciones.verificador()==True:
        cantidadCanciones = int(lista_canciones.cantidadElementos())
        if cantidadCanciones > 0:
            for i in range(cantidadCanciones):
                nombreLista = "Biblioteca"
                reproducido = lista_canciones.BuscarPorIndice(i).CantidadReproducciones
                if int(reproducido > 0):
                    nombreCancion = lista_canciones.BuscarPorIndice(i).nombre
                    nombreArtista = lista_canciones.BuscarPorIndice(i).artista
                    nombreAlbum = lista_canciones.BuscarPorIndice(i).album
                    cargarDatos(nombreLista, nombreCancion, nombreArtista, nombreAlbum, reproducido)
                i+=1
    if lista_playlists.verificador() == True:
        numeroListas = int(lista_playlists.cantidadElementos())
        if numeroListas > 0:
            for i in range(numeroListas):
                if lista_playlists.BuscarPorIndice(i).canciones.verificador() == True:
                    numeroCanciones = int(lista_playlists.BuscarPorIndice(i).canciones.cantidadElementos())
                    if numeroCanciones > 0:
                        for j in range(numeroCanciones):
                            reproducido = cancionPorPosicionEnLista(i,j).CantidadReproducciones
                            if int(reproducido) > 0:
                                nombreLista = ver_Nombe_Lista_Posicion(i)
                                nombreCancion = cancionPorPosicionEnLista(i,j).nombre
                                nombreArtista = cancionPorPosicionEnLista(i,j).artista
                                nombreAlbum = cancionPorPosicionEnLista(i,j).album
                                cargarDatos(nombreLista, nombreCancion, nombreArtista, nombreAlbum, reproducido)
                            j+=1
                i+=1
```

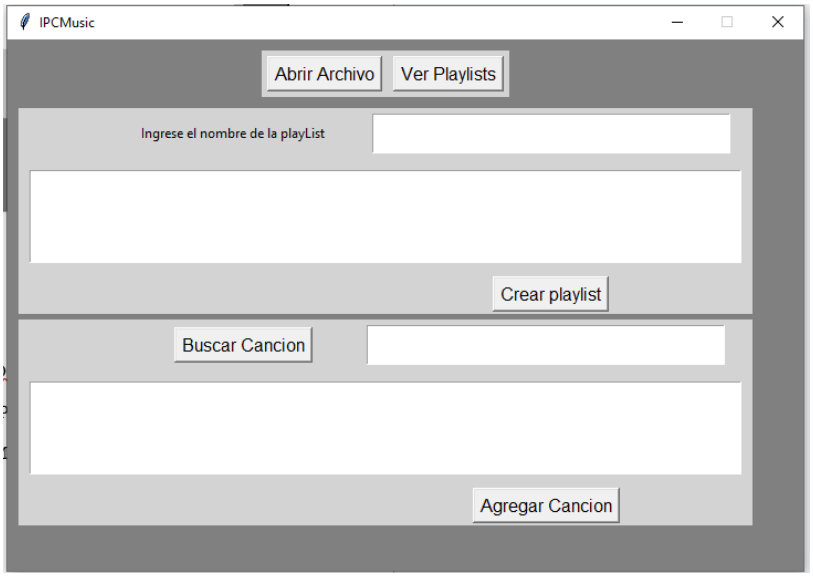
```python
def cargarDatos(nombreLista, nombreCancion, nombreArtista, nombreAlbum, reproducido):
    archivo = open("FormatoHTML\\formato.html")

    src = Template(archivo.read())

    cancion = {'nombreLista':nombreLista, 'nombreCancion':nombreCancion, 'nombreArtista':nombreArtista, 'nombreAlbum':nombreAlbum,

    result = src.substitute(cancion)

    try:
        os.mkdir('Cancion')
        archivo2 = open("Reporte.html",'a')
        archivo2.writelines(result)
    except OSError:
        if os.path.exists('Cancion'):
            archivo2 = open("Reporte.html",'a')
            archivo2.writelines(result)

    os.system(r"")

if __name__=="__main__":
    reporte_HTML()
```

Funcionamiento

Ventana principal: nos permite recorrer entre canciones generar el reporte, que incluye la gráfica, el xml de salida y buscar una cancion en especifico



ventana secundaria: nos permite crear listas de canciones, agregar canciones a las mismas y cargar el xml de entrada



Referencias bibliográficas

W. R. McKinney, (2016). Python for Data Analysis. O'Reailly, Inc.

A. J. Aho, (2005). Estructuras de datos y algoritmos. Prentice-Hall

L. M. Joyanes, (2010). Estructura de Datos en Java. Mc Graw Hill, Inc.

Link repositorio: https://github.com/JosCarias/IPC2_Proyecto1Diciembre_-Grupo-13.git