Ingeniería en computadores

CE-1103 Algoritmos y estructuras de datos I

Profesor:

Leonardo Andres Araya Martinez

Proyecto 2:

Music Box

Estudiantes:

Fared Alonso Carballo Quesada, 2025085342

Christian Vargas Alvarado, 2022437590

Franklin Santiago Mora Barrantes, 2025097959

Verano

[1] 13/1/2026

1 Introducción

La tecnología ha ampliado sus aplicaciones más allá de los sistemas tradicionales, integrándose también en áreas artísticas como la música. El proyecto *Music Box* busca explorar esta relación mediante la implementación de un reproductor de partituras utilizando estructuras de datos lineales, específicamente una lista doblemente enlazada.

A través de este sistema, se reproducen notas musicales definidas por su frecuencia y duración, aplicando conceptos básicos de teoría musical y programación.

Los requerimientos para la creación del reproductor de partituras es

- Poder guardar las notas deseadas
- Reproducción normal y en reversa
- El codigo esta en C#

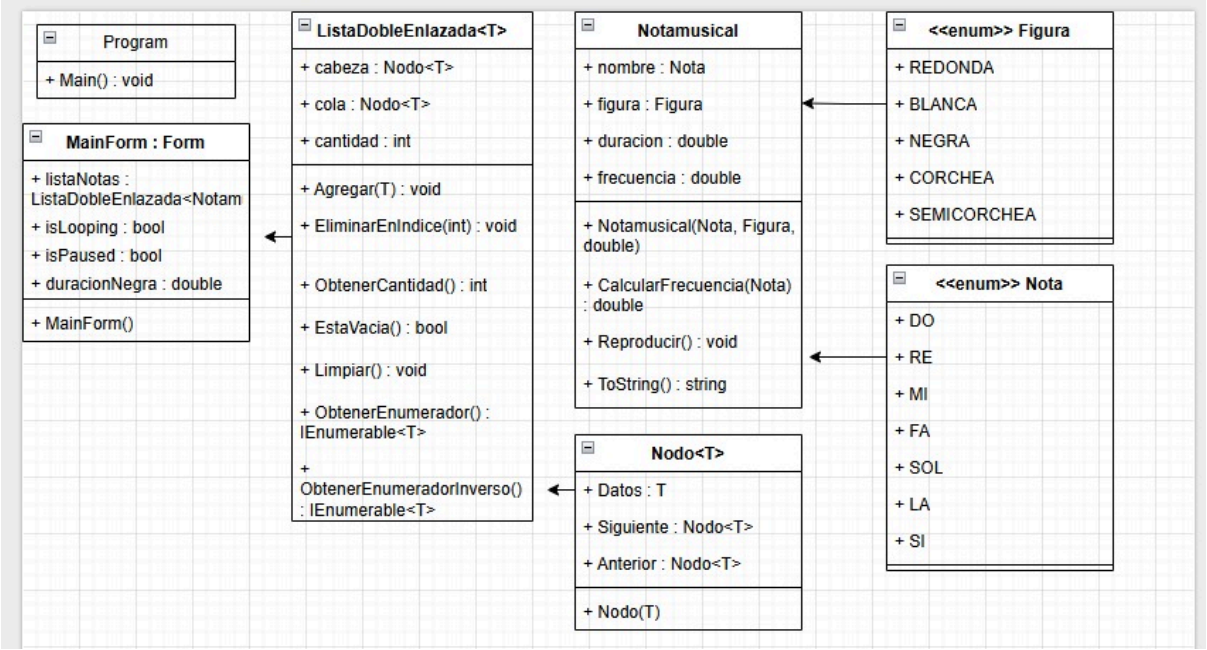
Link Github: https://github.com/Cvargas17/Proyecto2VeranoDatos1

1.2 Problema En este proyecto se requiere de un programa el cual pueda detectar, guardar, y reproducir varias notas musicales en función de una partitura. El cual tiene que también poder reproducir dichas notas en reversa. Ahora, para realizar esto se requiere de una lista doblemente enlazada para así el programa pueda acceder de forma oportuna las notas y así poder cumplir con lo que se le solicita

2.Diseño general

2.1 Diagrama UML

Diagrama UML de las clases utilizadas en el proyecto de MusicBox.



3.Test Plan

El test plan consiste en un conjunto de test unitarios que tienen una funcionalidad sumamente importante posterior al desarrollo de una funcionalidad o de un proyecto. Que básicamente prueba la funcionalidades de manera simple, esto es muy importante a la hora de verificar que todas las funciones agregadas cumplan su debida tarea, y en caso de haber un error, como los test unitarios son tan puntuales entonces este problema puede ser encontrado fácilmente y ser corregido para que no pase a mayores.

El proyecto emplea 10 test unitarios que prueban diversas funcionalidades del código principal, como la frecuencia de todas las notas o la duración de las

figuras. Mismos tests van a ser descritos con la siguiente tabla, que va a ejemplificar cómo se desarrolla el proceso de testeo y qué hace cada test en concreto:

1. **Qué hace el test**
2. **Cuál es la entrada que se le ingresa al código probado**
3. **Cuál es la salida que se espera**
4. **Un screenshot del test corriendo**

- Test 1: Pruebas de notas musicales:
  - Este test determina que todas las frecuencias de las notas correspondan a las frecuencias predefinidas de cada una de ellas. En caso de que alguna o varias no correspondan, el test falla.
  - La entrada que se le ingresa es la frecuencia esperada de cada una de las notas, seguida del objeto de la nota con su nombre, figura (estándar se usa negra), y el tiempo de la negra. También se establece una variación de 0.01 entre frecuencias.
  - Para la salida se esperan igualmente esas mismas frecuencias para cada nota.

```
0 references
public class NotamusicalTests
{
    //test 1: frecuencias correctas de todas las notas
    [Fact]
    0 references
    public void FrecuenciaCorrecta_DeTodasLasNotas()
    {
        Assert.Equal(261.63, new Notamusical(Nota.DO, Figura.NEGRA, 1.0).frecuencia, 2);

        Assert.Equal(293.66, new Notamusical(Nota.RE, Figura.NEGRA, 1.0).frecuencia, 2);

        Assert.Equal(329.63, new Notamusical(Nota.MI, Figura.NEGRA, 1.0).frecuencia, 2);

        Assert.Equal(349.23, new Notamusical(Nota.FA, Figura.NEGRA, 1.0).frecuencia, 2);

        Assert.Equal(392.00, new Notamusical(Nota.SOL, Figura.NEGRA, 1.0).frecuencia, 2);

        Assert.Equal(440.00, new Notamusical(Nota.LA, Figura.NEGRA, 1.0).frecuencia, 2);

        Assert.Equal(493.88, new Notamusical(Nota.SI, Figura.NEGRA, 1.0).frecuencia, 2);
```

- Test 2: Pruebas de figuras musicales:
  - El test determina la duración de cada una de las figuras musicales, siendo estas dependientes a la duración de la negra. Entonces determina si su duración corresponde a la esperada (x4 duración para

la redonda, x2 duración para la blanca, x1 duración para la negra, x1/2 duración para la corchea y x1/4 duración para la semicorchea).

- La entrada que se le ingresa es primeramente la definición del tiempo estándar de la negra (1 segundo), y luego se va calculando según la figura. Luego se crea el objeto para cada figura, con la nota DO como nota estándar.
- Se espera una salida con la duración esperada del cálculo con la negra anteriormente establecida.

```
//test 2: duración correcta de todas las figuras
[Fact]
0 references
public void DuracionCorrecta_DeTodasLasFiguras()
{
    double negra = 1.0;

    Assert.Equal(4.0, new Notamusical(Nota.DO, Figura.REDONDA, negra).duracion, 2);

    Assert.Equal(2.0, new Notamusical(Nota.DO, Figura.BLANCA, negra).duracion, 2);

    Assert.Equal(1.0, new Notamusical(Nota.DO, Figura.NEGRA, negra).duracion, 2);

    Assert.Equal(0.5, new Notamusical(Nota.DO, Figura.CORCHEA, negra).duracion, 2);

    Assert.Equal(0.25, new Notamusical(Nota.DO, Figura.SEMICORCHEA, negra).duracion, 2);

}
```

- Test 3: Prueba que la lista empiece vacía:
  - El código valida que siempre se empiece con una lista vacía.
  - Se crea un objeto de lista vacía.
  - Si la lista está vacía entonces el test tiene éxito (no se espera retornar nada).

```
//test 3: la lista inicia vacía
[Fact]
0 references
public void Lista_IniciaVacia()
{
    var lista = new ListaDobleEnlazada<int>();
    Assert.True(lista.EstaVacia());
}
```

- Test 4: Obtener cantidad de elementos:
    - Prueba que no haya elementos, o que los elementos en la lista sean cero, para que en el próximo test se determine el aumento de esa cantidad.
    - El código crea una lista con cero elementos.
    - El código retorna la cantidad de elementos en la lista (0).}

```
//test 4: obtenerCantidad inicia en cero
[Fact]
0 references
public void ObtenerCantidad_IniciaEnCero()
{
    var lista = new ListaDobleEnlazada<int>();
    Assert.Equal(0, lista.ObtenerCantidad());
}
```

- Test 5: Aumenta la cantidad cuando se introduce un elemento:
    - Como su nombre lo indica, es el test encargado de validar que, cuando se ingresa un elemento (Nota, Figura), sea añadido a la lista y la cantidad de elementos suba.
    - Se crea una lista con y se le asignan 1 elemento.
    - Se espera que los elementos se vean reflejados dentro de la lista.

```
//test 5: agregar un elemento aumenta la cantidad
[Fact]
0 references
public void Agregar_Elemento_AumentaCantidad()
{
    var lista = new ListaDobleEnlazada<int>();
    lista.Agregar(10);
    Assert.Equal(1, lista.ObtenerCantidad());
}
```

- Test 6: Orden de elementos:
    - Se prueba que los elementos de la lista se encuentren en el orden con el que fueron introducidos.
    - Se introducen tres elementos genéricos y se determina si están en el orden con el que fueron introducidos.
    - Se debe retornar el orden correcto de los elementos, de lo contrario dará error.

```
//test 6: enumerador devuelve elementos en orden
[Fact]
0 references
public void Enumerador_DevuelveElementosEnOrden()
{
    var lista = new ListaDobleEnlazada<int>();
    lista.Agregar(1);
    lista.Agregar(2);
    lista.Agregar(3);

    var resultado = new List<int>(lista.ObtenerEnumerador());

    Assert.Equal(new List<int> { 1, 2, 3 }, resultado);
}
```

- Test 7: Frecuencia nunca es cero:
    - Se determina que la frecuencia de una nota nunca será cero, de lo contrario no será apta para el programa.
    - Se crea un objeto con la nota DO y la figura estándar (negra), con una duración de 1 segundo, y se determina que su frecuencia mientras se están reproduciendo nunca será cero.
    - Se analiza esta frecuencia y se retorna el resultado de la prueba.

```
0 references
public class NotamusicalExtraTests
{
    //test 7: la frecuencia nunca es cero
    [Fact]
    0 references
    public void Frecuencia_EsMayorQueCero()
    {
        var nota = new Notamusical(Nota.DO, Figura.NEGRA, 1.0);
        Assert.True(nota.frecuencia > 0);
    }
```

- Test 8: Duración nunca es cero:
    - Este test (parecido al previo) determina que la duración de la nota (la figura utilizada) nunca será cero.
    - Se crea un objeto con la nota DO y la figura CORCHEA con el tiempo establecido de 1 segundo.
    - Se espera que aunque la nota no sea una NEGRA, sino una fracción de esta, nunca llegue a ser cero, de lo contrario el test falla.

```
//test 8: la duración siempre es positiva
[Fact]
0 references
public void Duracion_EsMayorQueCero()
{
    var nota = new Notamusical(Nota.DO, Figura.CORCHEA, 1.0);
    Assert.True(nota.duracion > 0);
}
```

- Test 9: String no es vacío:
    - Se determina que al agregarse una nota no surja un error como que en lugar de la nota y figura se agregue un valor o espacio vacío en la lista.
    - Se introduce el objeto estándar (DO y NEGRA), con la duración de 1 segundo. Se espera que se agregue en la lista y no sea un conjunto vacío.
    - Se debe retornar el mismo objeto y no un conjunto vacío. En caso de que se retorne vacío, el test falla.

```
//test 9: toString no devuelve texto vacío
[Fact]
0 references
public void ToString_NoEstaVacio()
{
    var nota = new Notamusical(Nota.DO, Figura.NEGRA, 1.0);
    Assert.False(string.IsNullOrWhiteSpace(nota.ToString()));
}
```

- Test 10: Enumerador inverso:
    - Como la lista es una lista doblemente enlazada, esta prueba determina que cuando se establezca el recorrido al revés los valores se inviertan correctamente.
    - Se crea una lista en la que se agregan elementos genéricos y luego se recorre la lista al revés.
    - Se espera que se retornen los valores en orden invertido, de lo contrario el test falla.

```
//test 10: enumerador inverso devuelve el orden correcto
[Fact]
0 references
public void EnumeradorInverso_DevuelveOrdenInverso()
{
    var lista = new ListaDobleEnlazada<int>();
    lista.Agregar(1);
    lista.Agregar(2);
    lista.Agregar(3);

    var resultado = new List<int>(lista.ObtenerEnumeradorInverso());

    Assert.Equal(new List<int> { 3, 2, 1 }, resultado);
}
```

Finalmente cuando todos los test se ejecuten correctamente, en la terminal se presentará un resumen de las pruebas, si estas fueron exitosas entonces devolverá la cantidad de pruebas ejecutadas, cuales tuvieron un error y cuales fueron exitosas (en nuestro caso todas serán exitosas).

```
  MusicBox.Tests net10.0-windows realizado correctamente prueba (0,8s)

Resumen de pruebas: total: 10; con errores: 0; correcto: 10; omitido: 0; duración: 0,8 s
Compilación realizado correctamente en 2,0s
```