Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Laboratorio de Algoritmos y Estructuras II
Enero-Marzo 2016

Proyecto 2: Implementación de una aplicación para la reproducción de música

1. Introducción

El objetivo del proyecto es la de hacer una implementación de un reproductor de música, que soporte reproducir una lista de varias canciones en un determinado orden. En su implementación usted tendrá la oportunidad de usar varias estructuras de datos que se han estudiado en su curso de teoría de algoritmos 2.

La aplicación podrá reproducir una lista de canciones que serán importadas de un archivo de texto plano, adicionalmente se podrán hacer búsquedas de canciones por autor o por género y operaciones de agregación, eliminación y ordenamiento sobre la lista de canciones.

2. TADs a considerar

2.1. Marco teórico sobre representación de TADs

La especificación de los TADs que usaremos en este proyecto se harán usando el estilo basado en la teoría de modelos que se encuentra en las notas del Prof. Ravelo [1]. Un modelo en el mundo de la lógica, se refiere a un tipo de objeto matemático que satisface una lista de requerimientos (a estos requerimientos se le llaman generalmente axiomas). Por ejemplo, si informalmente decimos que los requerimientos para que algún objeto sea una pila de tipo T son:

1. Es un contenedor de objetos de tipo T
2. Los elemento se pueden insertar de uno en uno
3. Los elementos se sacan en orden inverso al cual se insertaron

Entonces un modelo para estos requerimientos o axiomas es cualquier tipo de objeto junto con unas operaciones que satisfaga estos tres items. Por ejemplo, las secuencias s de elementos en T junto con dos operaciones de empilar y desempilar que agregan y eliminan elementos de uno en uno sólo al final de la secuencia, es un modelo de Pila. Si los tres axiomas o requerimientos anteriores se escriben en un lenguaje formal expresado en forma de ecuaciones, entonces se le llama especificación algebraica del TAD, en cambio las secuencias s junto con las operaciones se le llama especificación por modelo del TAD.

La especificación por modelo siempre es un tipo de objeto junto con unas operaciones, el tipo de objeto sólo sin tomar en cuenta las operaciones se denomina modelo de representación. Si el modelo de representación cumple con una condición para todos los objetos del TAD, entonces esta condición se denomina *invariante de representación*.

En el ejemplo anterior, el modelo de representación eran las secuencias s, en otras palabras, por definición de secuencia, el modelo de representación eran las función $s : [0..N) \to T$, lo cual es un objeto abstracto que vive en el mundo de las matemáticas.

Cuando los objetos que se usan para el modelo del TAD viven en el mundo abstracto de las matemáticas, entonces se dice que el modelo de representación es abstracto, en cambio cuando el modelo de representación usa objetos que son provisto directamente por el lenguaje de programación que usted va a usar, se denomina modelo de representación concreto. Por ejemplo la secuencia s anterior con sus operaciones se considera un modelo abstracto, en cambio si la pila se hubiese modelado con un arreglo en lugar que con una secuencia y con dos operaciones empilar y desempilar sobre él, entonces se consideraría como un modelo concreto.

En este enunciado se especificaran el comportamiento de los métodos de los TADs que usted usará, usando modelos abstractos. Esto es, porque es más simple la especificación de esta forma que con los modelos de representación concretos. Seguidamente de la especificación abstracta de cada TAD se mostrará el modelo de representación concreto con la estructura de datos específica que usted debe usar para la implementación de cada TAD.

Nota. *Usted usará el modelo abstracto sólo para entender el comportamiento de los métodos, pero debe programar teniendo en mente el modelo concreto, es decir la estructura de datos que específicamente se le solicita*

A continuación daremos la especificación de cada uno de estos TAD. Las siguientes especificación son lo mínimo que se requiere, pero usted es libre de agregar nuevos TADs y métodos a los TADs descritos si lo considera necesario.

2.2. TAD Canción

Es el TAD que representa a una canción a reproducir, contiene los atributos título, género, cantante y archivo (de tipo mp3, AVI o OGG).

La especificación del TAD es la siguiente:

Modelo de representación

var *titulo*: String
var *genero*: String
var *artista*: String
var *archivo*: File

Invariante de Representación

$titulo \neq$ "" $\wedge genero \neq$ "" $\wedge cantante \neq$ "" $\wedge EsArchivoDeMusica(archivo)$

Nota. *El predicado* $EsArchivoDeMusica(archivo)$ *es verdadero si y sólo si archivo es un archivo de música en formato mp3, AVI o OGG*

Concretamente para implementar este TAD en Python usted definirá una clase de nombre Cancion que poseerá los atributos título, genero, cantante y archivo

Operaciones

Los métodos se definirán en notación GCL

proc *crear*(**in** $titulo : String,\ genero : String,\ artista : String,\ archivo : File$; **out** $self : Cancion$)
{**Pre**: $titulo \neq$ "" $\wedge genero \neq$ "" $\wedge artista \neq$ "" $\wedge archivo \neq$None}
{**Post**: $self.titulo = titulo \wedge self.genero = genero \wedge self.artista = artista \wedge self.archivo = archivo$}

Este método está escrito en notación GCL, sin embargo corresponde al constructor de la clase en Python cuya firma es

$$\_\_init\_\_(self, titulo, genero, artista, archivo)$$

proc *es_igual*(**in** $self : Cancion,\ cancion : Cancion$; **out** $igual : Bool$)
{**Pre**: $True$}

{**Post**: $igual \equiv self.titulo = cancion.titulo \wedge self.artista = cancion.artista$}

En lenguaje natural esta especificación dice que dos canciones son iguales si y sólo si sus títulos y artistas son iguales. La colección de los pares de canciones tales que son iguales, forman una relación que es de equivalencia, en otras palabras, la relación que se forma es reflexiva, simétrica y transitiva.

proc *esmenor_artista*(**in** *self* : *Cancion*, *cancion* : *Cancion*; **out** *esMenor* : *Bool*)
{**Pre**: *True*}
{**Post**: $esMenor \equiv self.artista \sqsubset_{lex} cancion.artista \vee$
$(self.artista = cancion.artista \wedge self.titulo \sqsubseteq self.titulo)$}

Nota. *El símbolo* $\sqsubset_{lex}$ *denota la relación de orden lexicográfico estrico, que es la relación de orden que se usa para ordenar las palabras en un diccionario. La notación* $a \sqsubseteq_{lex} b$ *denota* $(a = b \vee a \sqsubset_{lex} b)$

La última especificación dice que esmenor_artista devuelve el valor de verdad de si el artista de la canción *self*, es menor según el orden lexicográfico al artista de *cancion* y si los artistas son iguales, devuelve el valor de verdad de si el título de *self* es menor lexicográficamente al título de *cancion*.

proc *esmenor_titulo*(**in** *self* : *Cancion*, *cancion* : *Cancion*; **out** *esMenor* : *Bool*)
{**Pre**: *True*}
{**Post**: $esMenor \equiv self.titulo \sqsubset_{lex} cancion.titulo \vee$
$(self.titulo = cancion.titulo \wedge self.artista \sqsubseteq self.artista)$}

Esta especificación dice que esmenor_artista devuelve el valor de verdad de si el título de la canción *self* es menor según el orden lexicográfico al título de *cancion* y si los títulos son iguales, devuelve el valor de verdad de si el artista de *self* es menor lexicográficamente al artista de *cancion*.

proc *get_titulo*(**in** *self* : *Cancion*, **out** *titulo* : *String*)
{**Pre**: *True*}
{**Post**: $self.titulo = titulo$}

proc *get_genero*(**in** *self* : *Cancion*, **out** *genero* : *String*)
{**Pre**: *True*}
{**Post**: $self.genero = genero$}

proc *get_artista*(**in** *self* : *Cancion*, **out** *artista* : *String*)
{**Pre**: *True*}
{**Post**: $self.artista = artista$}

proc *get_archivo*(**in** *self* : *Cancion*, **out** *archivo* : *File*)
{**Pre**: *True*}
{**Post**: $self.archivo = archivo$}

2.3. TAD Lista de reproducción

Corresponde a la lista de canciones que se van a reproducir. Las canciones que se ven en el panel de reproducción corresponden a una lista de reproducción.

La especificación del TAD Lista de reproducción es la siguiente:

Modelo abstracto de representación
var *contenido*: secuencia de tipo Canción

Invariante de Representación

$\neg(\exists\, i, j \mid 0 \leq i < j < len(contenido) : contenido[i].es\_igual(contenido[j]))$

El invariante anterior dice que no hay canciones repetidas en la secuencia *contenido* la cual representa la secuencia de canciones a reproducir.

Por otro lado para que se pueda navegar hacia adelante y atrás en la lista de reproducción, y para que cuando termine la última canción, se comience de nuevo desde la primera, entonces la implementación del TAD lista de reproducción concretamente debe hacerse con una lista circular doblemente enlazada.

En una lista circular doblemente enlazada los nodos tienen los campos *next* y *prev* que apuntan al próximo y anterior nodo respectivamente, adicionalmente los nodos están enlazados formando un círculo, es decir, el campo *next* del último nodo apunta al primero y el campo *prev* del primer nodo apunta al último (ver Figura 1).

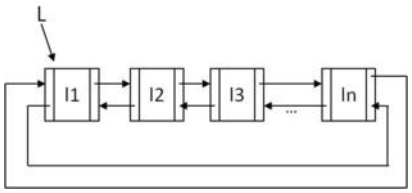

Figura 1: Lista circular doblemente enlazada

En el modelo de representación concreto no usaremos la secuencia contenido ya que es un objeto matemático, en su lugar sustituiremos la variable contenido por una variable *cabeza* que apunta al primer nodo de la lista. Con el apuntador a la cabeza podemos recorrer y recuperar todos los elementos de la lista, siguiendo los enlaces de sus nodos.

Como expusimos anteriormente definimos el modelo de representación concreto del TAD Lista de reproducción de la siguiente forma:

Modelo concreto de representación

var *cabeza*: NodoLista

Nota. *NodoLista es un tipo de dato que almacena la canción y los apuntadores next y prev a los otros nodos de la lista.*

Para implementar este TAD en Python usted debe definir una clase llamada *ListaReproduccion* que debe tener como atributo al nodo *cabeza*

Operaciones

La definición formal de las operaciones se hace en función de la representación abstracta en notación GCL y en lenguaje natural los comentarios referentes a la implementación en concreto

proc *crear*(**out** *self* : *Lista de reproduccion*)
{**Pre**: *True*}
{**Post**: $self.contenido = \langle\rangle$}

Este método está escrito en notación GCL, sin embargo corresponde al constructor de la clase en Python cuya firma es

$$\_\_init\_\_(self)$$

proc *agregar*(**in-out** *self* : *Lista de reproduccion*; **in** *cancion* : *Cancion*)
{**Pre**: *True*}
{**Post**: $self.contenido = \langle cancion \rangle \;||\; self_0.contenido$}

Nota. *La notación $self_0$ denota a la variable de especificación que contiene el estado de la variable self antes de de ejecutar el procedimiento.*

Como usted vio en su curso de Algoritmos 1 $\langle cancion \rangle$ significa secuencia de longitud 1 con único elemento *cancion*, y el operador ||, significa concatenación de secuencias. En lenguaje natural esta especificación significa que el método agregar de la lista l toma una Canción *cancion* y la agrega al inicio de la lista.

proc *eliminar*(**in-out** *self* : *Lista de reproduccion*; **in** c : *Cancion*)
{**Pre**: $self.contenido \neq \langle\rangle \wedge esSecuencia(s_0) \wedge esSecuencia(s_0') \wedge self.contenido = s_0 \ || \ \langle c \rangle \ || \ s_0'$}
{**Post**: $self.contenido = s_0 \ || \ s_0'$}

En lenguaje natural, esta especificación significa que el método eliminar de la lista l toma una canción c y si está en la lista la elimina.

proc *ordenar_titulo*(**in-out** *self* : *Lista de reproducion*)
{**Pre**: *True*}
{**Post**: $set(self.contenido) = set(self_0.contenido) \wedge$
$(\forall\, i \mid 0 \leq i < len(self.contenido) - 1 \mid self.contenido[i].esmenor\_titulo(self.contenido[i+1]))$}

Nota. *La notación $set(self.contenido)$ se refiere al conjunto de las canciones que se encuentran en $self.contenido$*

En lenguaje natural la última especificación significa que la lista resultante estará ordenada por título de canción.

proc *ordenar_artista*(**in-out** *self* : *Lista de reproduccion*)
{**Pre**: *True*}
{**Post**: $set(self.contenido) = set(self_0.contenido) \wedge$
$(\forall\, i \mid 0 \leq i < len(self.contenido) - 1 \mid self.contenido[i].esmenor\_artista(self.contenido[i+1]))$}

En lenguaje natural esta especificación significa que la lista resultante estará ordenada por artista de canción.

2.4. TAD Indice

El TAD Indice es un diccionario de canciones para poder hacer búsquedas por género o por artista.
La especificación abstracta del TAD Indice es la siguiente:
Modelo abstracto de representación
var *tabla*: $String \nrightarrow$ *secuencia de Cancion*

Nota. *La notación $\nrightarrow$ en la línea anterior indica que tabla es una función parcial de los String a las secuencias de Canción.*

Invariante de Representación
True

Como queremos buscar de forma eficiente, por título de artista o por género, entonces se requiere usar tablas de Hash para la implementación de este TAD. Las tablas de Hash resolverán las colisiones usando encadenamiento (chaining) y como artista y género son String, entonces usaremos a los Strings como las claves del universo de canciones para las tablas de Hash. Como pueden haber más de una canción por clave (artista o género), entonces el par clave/valor de los nodos de nuestra tabla son de tipo $\langle String, lista\ de\ canciones \rangle$
Se usará la función de Hash (para claves de tipo String) de nombre *fnv*, que es una función

$$fnv : String \to [0, .., N)$$

que cumple con todas las propiedades matemáticas necesarias enunciadas en Cormen et al [2], que debe cumplir una función de Hash. Sin embargo, *fnv* no puede ser usada directamente, ya que si el número de

buckets de la tabla es m, y es menor a N, este hash pudiera mandar claves a índices fuera del arreglo, esto se soluciona fácilmente usando la variante

$$f_m(s) := fnv(s) \ mod \ m$$

que redimensiona los valores de fnv a un intervalo de índices dentro del arreglo, de esta forma $f_m(s)$ retorna el ídice del arreglo a usar para la clave s.

Si en cierto momento nuestra tabla guarda n elementos y tenemos m buckets, entonces llamaremos a $\alpha := \frac{n}{m}$ como el *factor de carga*. Como está demostrado que una búsqueda promedio en este tipo de tablas tiene un costo de $O(1+\alpha)$, entonces para asegurar la eficiencia de la estructura, mantendremos el valor de α menor a 0,8, en otras palabras, si la tabla crece en su número de elementos guardados a n', de manera que $\alpha = \frac{n'}{m} \geq 0{,}8$, entonces redimensionamos la tabla aumentando el número de buckets a $m' = 2m + 1$ para que el nuevo $\alpha = \frac{n'}{m'}$ sea menor a 0,8 y la volvemos a llenar con los mismos elementos pero usando la nueva función de Hash $f_{m'}$ que se ajusta al nuevo tamaño. Esta operación se conoce como rehash.

Modelo concreto de representación

const *size*: entero
var *buckets*: *Arreglo de Listas de pares* $(String, lista\ de\ canciones)$

Invariante de Representación

$$len(buckets) \geq 5 \wedge \tfrac{size}{len(buckets)} < 0{,}8 \wedge size = \sum_{i=0}^{len(buckets)-1} len(buckets[i])$$

La primera condición del invariante dice que el número mínimo de buckets es 5, la segunda condición del invariante dice que el factor de carga α es menor siempre a 0,8 y la tercera condición dice que *size* siempre es igual a la cantidad de elementos que están almacenados en la tabla.

En Python usted definirá una clase de nombre Indice con los atributos size y buckets como lo indica el modelo de representación concreto.

Operaciones

Las operaciones son especificadas formalmente en base al modelo de representación abstracto y en lenguaje natural los comentarios referentes a la implementación en concreto

proc *crear*(**out** *self* : *Indice*)
{**Pre**: *True*}
{**Post**: $self.tabla = \varnothing$}

Este método está escrito en notación GCL, sin embargo corresponde al constructor de la clase en Python cuya firma es

$$\_\_init\_\_(self)$$

Nota. *En términos del modelo concreto de representación, se requiere que adicionalmente, el constructor debe retornar la estructura creada tal que* $len(buckects) = 5$*, pero con las listas de pares vacías en cada bucket.*

proc *agregar*(**in-out** *self* : *Indice*; **in** s : *String*, c : *Cancion*)
{**Pre**: *True*}
{**Post**: $(s \notin Dom(self.tabla) \Rightarrow self.tabla = self_0.tabla \cup \{(s, \langle c \rangle)\}) \wedge$
$(s \in Dom(self.tabla) \Rightarrow self.tabla = self_0.tabla \cup \{\ (s, self_0.tabla(s)\ \ ||\ \ \langle c \rangle)\ \})\ \}$

En lenguaje natural esta especificación dice que se coloca un nuevo par clave/valor $(s, \langle c \rangle)$ al dicionario donde $\langle c \rangle$ es una lista con una sola canción c si estamos en el caso que la clave s no es clave de ningún elemento del diccionario, pero en el caso de que exista una lista de canciones almacenada con la clave s, entonces la canción c se colocará al final de la lista del par ya existente que tiene como clave a s.

Nota. *En su implementación concreta usted debe hacer rehash después de realizar esta operación en caso de que el factor de carga supere el valor* 0,8

proc *buscar*(**in** *self* : *Indice*, *s* : *String*; **out** *l* : *secuencia de Cancion*)
{**Pre**: $s \in Dom(self.tabla)$}
{**Post**: $l = self.tabla(s)$}

En lenguaje natural esta especificación dice que el método buscar devuelve la lista de canciones asociada a la clave *s*.

2.5. TAD Reproductor

Corresponde al concepto de reproductor abstrayendo su interfaz gráfica, es decir el concepto de reproductor considerando sólo sus operaciones play, pause, stop, next, etc.

La especificación del TAD Reproductor es el siguiente:

Modelo abstracto de representación
var *lista*: Lista de reproducción
var *cancion_actual*: Entero

Invariante de Representación
$cancion\_actual \in Dom(lista.contenido) \wedge lista.contenido \neq \varnothing$

El invariante anterior dice que *cancion_actual* es un índice dentro de la secuencia *lista.contenido* y que *lista.contenido* nunca es una lista vacía.

La variable *cancion_actual* representa el índice de la canción que se esta reproduciendo y la que se muestra en el panel como canción actual. *lista* es la lista de canciones a reproducir

En el modelo de representación concreto indexaremos la lista con los nodos y no con enteros, por lo que sustituiremos la variable *cancion_actual* de tipo entero por tipo *NodoLista*. Adicionalmente la lista abstracta *lista* del modelo de representación abstracto lo sustituiremos por su versión concreta en Python de tipo ListaReproduccion.

Como expusimos anteriormente definimos el modelo de representación concreto del TAD Reproductor de la siguiente forma:

Modelo concreto de representación
const *phonon*: PyQt4.phonon.MediaObject
var *lista*: ListaReproduccion
var *cancion_actual*: NodoLista

Concretamente en Python usted definirá una clase de nombre Reproductor que tendrá como atributo *lista* de tipo ListaReproduccion, *cancion_actual* de tipo NodoLista y un atributo adicional de tipo PyQt4.phonon.MediaObject, que es un objeto Python que permite reproducir música e implementa las operaciones play, stop y pause. Vea la próxima sección para más información sobre como reproducir música en Python.

Operaciones

Los métodos se definirán en notación GCL en base al modelo de representación abstracto. Para facilitar la especificación definiremos los predicados *reproduciendo*(r) y *estaEnPause*(r) que son ciertos si y sólo si se está reproduciendo y está en pausa, la canción actual del reproductor r respectivamente.

proc *crear*(**in** *lista* : *Lista de reproduccion*; **out** *self* : *Reproductor*)
{**Pre**: $lista.contenido \neq \varnothing$}
{**Post**: $self.cancion\_actual = 0 \wedge self.lista = lista$}

Este método está escrito en notación GCL, sin embargo corresponde al constructor de la clase en Python cuya firma es

$$\_\_init\_\_(self, lista)$$

proc *play*(**in-out** *self* : *Reproductor*)
{**Pre**: $\neg reproduciendo(self) \vee estaEnPause(self)$}
{**Post**: $reproduciendo(self)$}

El método play reproduce la canción actual del reprocuctor.

proc *stop*(**in-out** *self* : *Reproductor*)
{**Pre**: $reproduciendo(self)$}
{**Post**: $\neg reproduciendo(self)$}

El método stop para la reproducción de la canción actual del reproductor.
proc *pause*(**in-out** *self* : *Reproductor*)
{**Pre**: $reproduciendo(self)$}
{**Post**: $estaEnPause(self)$}

EL método pause pone en pausa la reproducción de la canción actual del reproductor.
proc *siguiente*(**in-out** *self* : *Reproductor*; **in** c : *Cancion*)
{**Pre**: $\neg estaEnPause(self)$}
{**Post**: $self_0.lista = self.lista \wedge self.cancion\_actual = self_0.cancion\_actual + 1 \ mod \ len(self.lista)$}

En lenguaje natural esta especificación significa que la *cancion_actual* se actualiza a la siguiente en el orden circular de la lista de canciones.

proc *atras*(**in-out** *self* : *Reproductor*)
{**Pre**: $\neg estaEnPause(self)$}
{**Post**: $self_0.lista = self.lista \wedge self.cancion\_actual = self_0.cancion\_actual - 1 \ mod \ len(self.lista)$}

En lenguaje natural esta especificación significa que la *cancion_actual* se actualiza a la anterior en el orden circular de la lista de canciones.

proc *set_lista*(**in-out** *self* : *Reproductor*; **in** *lista* : *Lista de reproduccion*)
{**Pre**: $lista.contenido \neq \varnothing$}
{**Post**: $self.cancion\_actual = 0 \wedge self.lista = lista$}

Este método actualiza el valor de la lista de reproducción por la nueva *lista* que se da por parámetro y además se coloca la *cancion_actual* como la primera de la *lista*

2.6. TAD Interfaz Gráfica

Corresponde al TAD que maneja los métodos y atributos para el manejo de la interfaz gráfica, usted implementará este TAD haciendo uso de una librería gráfica de Python llamada PyQt4

La interfaz gráfica constará de un panel con los botones del reproductor (que son play, stop, pause, siguiente y atrás) y un panel con la lista de canciones a reproducir y el título y artista de la canción actual. La canción actual es la que esta tocándose o la que es la próxima a tocar si se aprieta play. Es de notar que no se mostrarán todas las canciones cargadas por archivo, sino sólo las que pase el cliente para ser reproducidas. (ver operaciones del TAD cliente)

Los únicos eventos que manejará la interfaz gráfica son el click sobre un botón y el refrescar el panel de lista de reproducción y el de la canción actual.

3. Aplicación Cliente

El cliente debe poder leer archivos en texto plano, donde se encontraran listados la dirección de varios archivos de audio en formato mp3, AVI o OGG. Estos archivos son leídos y cargados como una lista de reproducción para el reproductor, que deben visualizarse en un panel estático de una interfaz gráfica destinada para la lista de canciones a reproducir.

El formato del archivo de texto tendrá una canción por cada línea y en cada línea los campos se diferenciaran por tabulaciones a modo de columnas de la siguiente forma:

$$< titulo > \ [TAB] \ < artista > \ [TAB] \ < genero > \ [TAB] \ < ruta \ al \ archivo \ de \ audio >$$

El usuario manejará una parte de la aplicación por líneas de comandos y otra parte por interfaz gráfica. La ejecución de la aplicación debe hacerse con el comando

$$> \ python \ reproductor.py \ < archivo >$$

el cual debe generar un menú por consola que permita interactuar al usuario con la aplicación y debe desplegar una interfaz gráfica con los botones del reproductor de música y un panel con la lista de canciones listadas en el archivo de texto plano $< archivo >$,

Las opciones que el menú por consola debe permitir al usuario ejecutar son las siguientes: Importar canciones de archivo de texto, eliminar canción, ordenar por título, ordenar por artista, buscar por género, buscar por artista, restaurar lista de reproducción original.

Los botones que la interfaz gráfica debe tener son las siguientes: play, pause, stop, siguiente y atrás.

Nuestro reproductor será de forma tal que bloquea el menú de la consola, de modo que usted no podrá usar el menú mientras se está reproduciendo una canción.

El cliente hace uso del TAD Reproductor y el TAD InterfazGrafica, como veremos a continuación donde se encuentra la lista mínima de variables que debe usar este cliente.

Variables del programa Cliente
const *lista*: ListaReproduccion
var *reproductor*: Reproductor
var *interfaz*: Interfaz Gráfica
var *indice_artista*: Indice
var *indice_genero*: Indice

Operaciones

proc *inicializar_cliente*(**in** *lista* : *Lista de reproduccion*; **out** *reproductor* : *Reproductor*, *interfaz* : *Interfaz*, *indice_artista* : *Indice*, *indice_genero* : *Indice*)
{**Pre**: $lista \neq \varnothing \wedge$}
{**Post**: $reproductor.lista = lista \wedge$
$indice\_artista = IndArtista(lista) \wedge indice\_genero = IndGenero(lista)$}

Nota. *La notación* $IndArtista(lista)$ *denota al Indice resultante de aplicar a un Indice vacío* i *la operación* $agregar(i, a, c)$ *por cada canción* c *y su correspondiente artista* a*, de la lista de canciones lista. Por otro lado, La notación* $IndGenero(lista)$ *denota al Indice resultante de aplicar a un Indice vacío* j *la operación* $agregar(j, g, c)$ *por cada canción* c *y su correspondiente género* g*, de la lista de canciones lista.*

La especificación anterior dice que con la lista de reproducción *lista*, se construye un Reproductor que se almacena en el atributo *reproductor* y los dos Indices por artista y género. Para construir los índice se comienza creando uno vacío y luego se aplica el método agregar por cada canción de la lista acompañado de su respectiva clave dependiendo si el índice es por artista o por género.

proc *restaurar_lista_original*(**in** *lista* : *ListaReproduccion* **in-out** *reproductor* : *Reproductor*)
{**Pre**: *True*}
{**Post**: $reproductor.lista = lista$}

Este procedimiento restablece en el reproductor, la lista de reproducción original (que se almacena en la variable *lista* del cliente) que existía justo después de la última importación de archivo. Esta lista se restablece también en el panel gráfico.

proc *eliminar_cancion*(**in-out** *reproductor* : *Reproductor*; **in** *c* : *Cancion*)
{**Pre**: $esSecuencia(s_0) \wedge esSecuencia(s_0') \wedge reproductor.lista.contenido = s_0 \ || \ \langle c \rangle \ || \ s_0'$}
{**Post**: $reproductor.lista.contenido = s_0 \ || \ s_0'$}

Este procedimiento elimina la canción *c* de la lista de canciones del reproductor y adicionalmente refresca el panel gráfico con la lista resultante.

Nota. *El usuario introducirá por consola el título y artista de la canción a eliminar y una vez hecho esto, se creará una canción c con estos campos que se usaŕa para llamar este procedimiento y eliminar la canción de la lista de reproducción del reproductor.*

proc *ordenar_por_titulo*(**in-out** *reproductor* : *Reproductor*)
{**Pre**: $reproductor.lista.contenido = lista_0$}
{**Post**: $set(reproductor.lista.contenido) = set(lista_0) \wedge$
$estaOrdenadaPorTitulo(reproductor.lista.contenido)$}

Este procedimiento ordena por título de canción, la lista de reproducción interna del reproductor y la despliega en este orden en el panel gráfico.

proc *ordenar_por_artista*(**in-out** *reproductor* : *Reproductor*)
{**Pre**: $reproductor.lista.contenido = lista_0$}
{**Post**: $set(reproductor.lista.contenido) = set(lista_0) \wedge$
$estaOrdenadaPorArtista(reproductor.lista.contenido)$}

Este procedimiento ordena por artista la lista de reproducción interna del reproductor y la despliega en este orden en el panel gráfico.

proc *buscar_por_genero*(**in-out** *reproductor* : *Reproductor*; **in** *genero* : *String*, *indice_genero* : *Indice*)
{**Pre**: *True*}
{**Post**: $reproductor.lista = indice\_genero(genero)$}

Este procedimiento genera una nueva lista interna del reproductor con las canciones sólo del género escogido, de modo que esta lista se despliega en el panel gráfico. La lista se genera haciendo una búsqueda en el Indice *indice_genero* usando como clave al género

proc *buscar_por_artista*(**in-out** *reproductor* : *Reproductor*; **in** *artista* : *String*, *indice_artista* : *Indice*)
{**Pre**: *True*}
{**Post**: $reproductor.lista = indice\_artista(artista)$}

Este procedimiento genera una nueva lista interna del reproductor con las canciones sólo del artista escogido, de modo que esta lista se actualiza en el panel gráfico. La lista se genera haciendo una búsqueda en el Indice *indice_artista* usando como clave al artista

proc *importar*(**in-out** *reproductor* : *Reproductor*; **in** *archivo* : *String*)
{**Pre**: $reproductor = reproductor_0$}
{**Post**: $reproductor.lista = reproductor_0.lista \ ||< c \ for \ c \ in \ archivo >$}

Este procedimiento lee un archivo de lista de canciones, cada canción listada en el archivo se agregan a la lista de reproducción interna del reproductor y a la lista del panel gráfico.

proc *play*(**in-out** *reproductor* : *Reproductor*)

{**Pre**: $\neg reproduciendo(reproductor) \vee estaEnPause(reproductor)$}
{**Post**: $reproduciendo(reproductor)$}

Este procedimiento se ejecuta luego de apretar el botón de play de la interfaz gráfica, el cual inicia la reproducción de la lista de canciones del reproductor. Junto a los botones debe aparecer en el panel gráfico, el artista y título de la canción que se esta reproduciendo

proc *pause*(**in-out** *reproductor* : *Reproductor*)
{**Pre**: $reproduciendo(reproductor)$}
{**Post**: $estaEnPause(reproductor)$}

Este procedimiento se ejecuta luego de apretar el botón de pause de la interfaz gráfica, el cual pausa la reproducción de una canción. Este procedimiento no modifica el campo de artista y título de canción actual que se visualiza en el panel gráfico

proc *stop*(**in-out** *reproductor* : *Reproductor*)
{**Pre**: $reproduciendo(reproductor)$}
{**Post**: $\neg reproduciendo(reproductor)$}

Este procedimiento se ejecuta luego de apretar el botón de stop de la interfaz gráfica, el cual para la reproducción de una canción.

proc *siguiente*(**in-out** *reproductor* : *Reproductor*)
{**Pre**: $len(reproductor.lista) \neq \varnothing$}
{**Post**: $reproductor.cancion\_actual = reproductor.cancion\_actual + 1 \ mod \ len(reproductor.lista)$}

Este procedimiento se ejecuta luego de apretar el botón siguiente de la interfaz gráfica, el cual hace saltar a la siguiente canción de la lista de reproducción. Este procedimiento actualiza el campo de artista y título en el panel gráfico de la canción que se está reproduciendo.

proc *atras*(**in-out** *reproductor* : *Reproductor*)
{**Pre**: $len(reproductor.lista) \neq \varnothing$}
{**Post**: $reproductor.cancion\_actual = reproductor.cancion\_actual - 1 \ mod \ len(reproductor.lista)$}

Este procedimiento se ejecuta luego de apretar el botón atrás de la interfaz gráfica, el cual hace saltar a la canción anterior de la lista de reproducción. Este procedimiento actualiza el campo de artista y título en el panel gráfico de la canción que se está reproduciendo

4. Librerias gráficas de Python

para realizar la interfaz de su reproductor usted deberá usar la librería PyQt4 de python, que servirá para que usted realice la ventana del reproductor con sus diferentes botones (play, stop, pause, siguiente, atrás) y el panel de canción actual y lista de canciones. Para una breve explicación de uso de la librería PyQt4 para este proyecto puede visitar [3] y [4].

5. Librería para la reproducción de música en Python

Para reproducir la música en su reproductor usted usará la clase Phonon del módulo phonon de PyQt4, en esta clase se encuentran los métodos que permiten reproducir, parar y pausar una canción en formato mp3, AVI o OGG. Para más información visite [5], [6] y [7].

6. Condiciones de entrega

El programa que entregue debe poderse ejecutarse en Linux usando Python 2.7. Si un programa no se ejecuta, el equipo tiene cero como nota del proyecto.

Se considerará para su evaluación los aspectos de modularidad, estilo y documentación del código Python, según las normas de la guía de estilo publicada en la sección de Bibliografía del curso en Aula Virtual.

El trabajo es por equipos de laboratorio. Debe entregar los códigos fuentes de sus programas, en un archivo comprimido llamado *Proyecto2-X-Y.tar.gz*, donde X y Y son los números de carné de los integrantes del grupo. La entrega se realizará por aula virtual *antes* de la 1 : 00 pm del lunes 29 de Marzo de 2016. Cada grupo debe entregar firmada a su encargado de laboratorio, la Declaración de Autenticidad para Entregas, cuya planilla se encuentra en la página del curso. Ambos integrantes del equipo deben trabajar en el proyecto. El no cumplimiento de algunos de los requerimientos podrá resultar en el rechazo de su entrega.

7. Referencias

[1] RAVELO, J. Especificación e Implementación de Tipos Abstractos de Datos http://ldc.usb.ve/ jravelo/docencia/algoritmos/material/tads.pdf

[2] CORMEN, T., LEISERSON, C., RIVEST, R., AND STAIN, C, *Introduction to algorithms*, 3rd ed. MIT press, 2009.

[3] Intro/basic GUI - PyQt with Python GUI Programming tutorial https://www.youtube.com/watch?v=JBME1ZyHiP8&list=PLQVvvaa0QuDdVpDFNq4FwY9APZPGSUyR4

[4] PyQT4 Tutorials https://pythonspot.com/pyqt4/

[5] Phonon Module http://pyqt.sourceforge.net/Docs/PyQt4/phonon-module.html

[6] Muy rápido, muy fácil, reproductor de audio con Python http://notoquesmicodigo.blogspot.com/2013/07/muy-rapido-muy-facil-reproductor-de.html

[7] Reproductor de música en 72 líneas https://www.youtube.com/watch?v=sm3SSVSDuHo