Estructuras de Datos

Venta de entradas en C++

Grado en Ingeniería Informática

Pablo García García
Raúl López Llana

Martes 8:00-10:00h

17 de diciembre de 2021

Índice general

1. Especificación de la interfaz y TAD's **3**
1.1. TAD's implementados 3
1.2. Definición de operaciones 6
1.2.1. Entrada 7
1.2.2. Cliente 7
1.2.3. Pila 7
1.2.4. Cola 8
1.2.5. Lista 8
1.2.6. Lista de entradas 9
1.2.7. Árbol binario 10
1.2.8. Gestor 11

2. Solución adoptada **12**

3. Diseño de relación entre clases **13**
3.1. Diagrama UML 13
3.2. Métodos más destacables 15
3.3. Comportamiento del programa 17

4. Código fuente **18**
4.1. Cabeceras 18
4.2. Clases 25

Bibliografía **59**

DNI de los alumnos:
Pablo García: 03148485S
Raúl López: 03144345S

Índice de código fuente

4.1. Programa principal 18
4.2. Cabecera de entrada 18
4.3. Cabecera de cliente 19
4.4. Cabecera del nodo de pila 20
4.5. Cabecera de la pila 20
4.6. Cabecera del nodo de cola 21
4.7. Cabecera de la cola 21
4.8. Cabecera del nodo de lista 22
4.9. Cabecera de la lista 23
4.10. Cabecera de la lista de entradas 23
4.11. Cabecera del árbol binario 24
4.12. Cabecera del gestor 25
4.13. Implementación de entrada 25
4.14. Implementación de cliente 27
4.15. Implementación del nodo de pila 29
4.16. Implementación de la pila 30
4.17. Implementación del nodo de cola 33
4.18. Implementación de la cola 33
4.19. Implementación del nodo de lista 35
4.20. Implementación de la lista 35
4.21. Implementación de la lista de entradas 41
4.22. Implementación del árbol binario 44
4.23. Implementación del gestor 51

Capítulo 1

Especificación de la interfaz y TAD's

En esta primera parte de la memoria se detallarán los diferentes tipos de datos abstractos utilizados para el correcto funcionamiento de este sistema que simula un gestor de entradas.

1.1. TAD's implementados

En esta sección daremos la especificación de cada uno de los TAD's usados. La parte de `operaciones` de la especificación aparece en la sección 1.2, y se pueden ver implementadas en el lenguaje C++ en el capítulo 4. Vamos a suponer que los nodos de las estructuras (objetos y clases) con sus respectivos punteros son abstracciones que se originan a causa del lenguaje de programación para que su implementación en dicho lenguaje sea posible, y no TAD's en sí, es decir, tomaremos las especificaciones sin usar el "modelo de memoria dinámica" para simplificar las especificaciones.

- **Entrada**: representa lo que sería el pase de acceso a un determinado concierto.

  `espec` ENTRADA
  `usa` string, int, bool, cliente
  `género` entrada
  `fespec` ENTRADA

- **Cliente**: representa a la persona que va a un concierto. Son las personas que almacenamos y de las que gestionamos sus datos.

  `espec` CLIENTE
  `usa` bool, int, string, ENTRADA
  `género` cliente
  `fespec` CLIENTE

- **Pila**: representa una estructura de datos LIFO *(last input - first output)* que almacena los clientes previamente mencionados. De manera "legal" solamente se puede obtener

el cliente que se encuentra en su cima, y como se describe anteriormente, el último en entrar será el primero en salir. Podemos compararlo con una montaña de libros o platos donde solo podemos coger el de arriba del todo "sin accidentes".

`espec` PILA
`usa` string, int, bool, cliente
`género` pila
`fespec` PILA

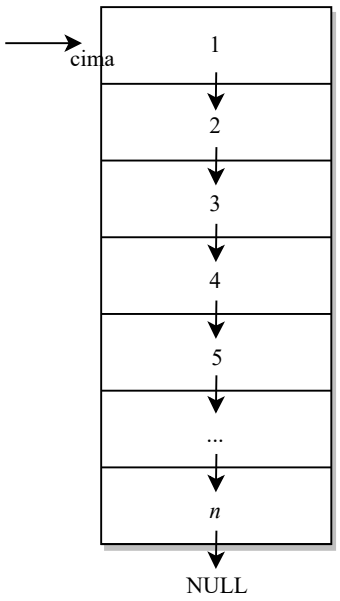

Figura 1.1: Esquema de la estructura pila

- **Cola**: representa una estructura de datos FIFO *(first input - first output)*. Se usarán dos para distinguir entre clientes registrados y no registrados que aún no han accedido a la venta de entradas. En esta estructura de datos solamente se puede consultar de manera legítima el primer elemento que fue insertado. Es similar a una sucesión de personas en una caja de una tienda. Las personas van esperando una detrás de otra para pagar, pero en un instante t_0 solo está siendo atendida una, la que primero había llegado. Se irá y la siguiente comenzará con el pago.

  `espec` COLA[CLIENTE]
  `usa` cliente, bool, int

`género` cola
`fespec` COLA[CLIENTE]

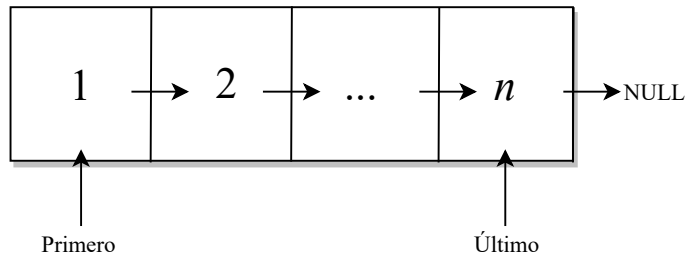

Figura 1.2: Esquema de la estructura cola

- **Lista**: representa una estructura de datos que no tiene un punto fijo de acceso. Se usará una para almacenar todos los clientes con sus entradas ya compradas.

  `espec` LISTA[CLIENTE]
  `usa` cliente, int, bool
  `género` lista
  `fespec` LISTA[CLIENTE]

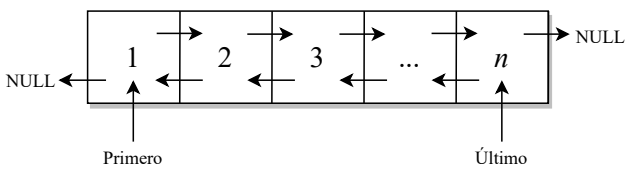

Figura 1.3: Esquema de la estructura lista

- **Lista de entradas**: representa el conjunto de entradas que tiene un cliente, siguiendo la definición de Lista y Entrada.

  `espec` LISTA[ENTRADA]
  `usa` entrada, int, bool
  `género` lista
  `fespec` LISTA[ENTRADA]

- **Árbol binario**: representa una estructura de datos construida por nodos, los cuales tienen un hijo izquierdo y un hijo derecho, donde a su vez, estos hijos son otros árboles (pueden ser vacíos). La raíz es el padre que da lugar a los dos primeros hijos. En cada nodo se almacenará el código del cliente y todas las entradas que tiene asociadas.

  `espec` ARBOL[CLIENTE, ENTRADA]
  `usa` cliente, entrada, int, bool, string
  `género` árbol
  `fespec` ARBOL[CLIENTE, ENTRADA]

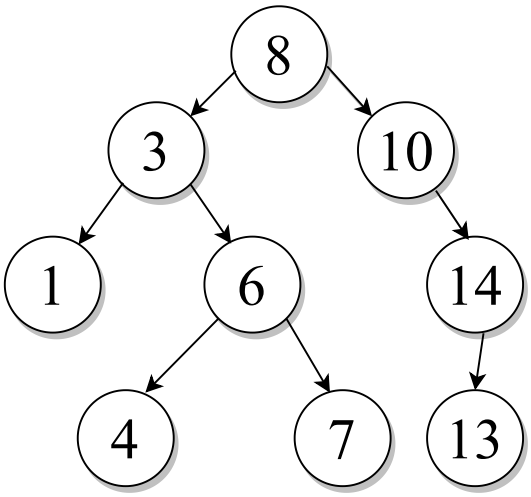

Figura 1.4: Esquema de la estructura árbol binario

1.2. Definición de operaciones

En esta sección se detallan los nombres, parámetros, retorno, y utilidad de las operaciones de los anteriores tipos de datos especificados. Al igual que en la sección anterior, los métodos constructores, destructores, *setters* y *getters* se interpretan como abstracciones del lenguaje C++ para poder implementarlos, pero no operaciones como tal del TAD.

1.2.1. Entrada

- Numero_Aleatorio: a int, b int $\rightarrow$ int. Genera un número $n \in [a, b]$ que será útil para decidir de qué concierto, ubicación, etc, es la entrada.
- Generar_Entrada_Aleatoria: _ $\rightarrow$ Entrada. Genera una entrada con datos aleatorios.
- mostrar: entrada $\rightarrow$ ___. Muestra por por pantalla los datos de esa entrada.
- tiene_entrada: ___ $\rightarrow$ bool. Comprueba si la entrada no es "nula".

1.2.2. Cliente

- Numero_Aleatorio: a int, b int $\rightarrow$ int. Genera un número $n \in [a, b]$ que será útil para asignar un número de DNI.
- mostrarDatos: cliente $\rightarrow$ ___. Muestra por pantalla los datos del cliente.
- contiene_entrada: cliente $\rightarrow$ bool. Comprueba si un cliente contiene una entrada.
- generar_Entrada: cliente $\rightarrow$ ___. Le asigna una entrada al cliente que recibe.

1.2.3. Pila

Vamos a usar la notación de pseudocódigo para los argumentos, es decir, en pseudocódigo pasaríamos nuestra pila como argumento de la operación, sin embargo en C++ simplemente nos referimos a nuestra pila con el puntero *this*, seguido del operador $->$ para aplicar esta función sin pasarle ningún argumento, obteniendo un método sin parámetros y de tipo *void*.

- esVacia: pila $\rightarrow$ bool. Comprueba si una pila está vacía o no.
- apilarAux: cliente, pila $\rightarrow$ pila. Dada una pila y un cliente, pone o apila este en su cima. No se usará como tal, pero otras operaciones funcionarán gracias a esta.
- desapilar: pila $\rightarrow$ pila. Dada una pila elimina el cliente que hay en su cima, y el que estaba debajo pasa a ser la nueva cima.
- mostrar: pila $\rightarrow$ ___. Dada una pila muestra por pantalla su cima en caso de que tenga.
- verEntera: pila $\rightarrow$ ___. Dada una pila muestra todo su contenido por pantalla de forma ilegítima, es decir, con ayuda de punteros y no desapilando.
- contar: pila $\rightarrow$ int. Dada una pila cuenta el total de clientes almacenados en esta.
- fondo: pila $\rightarrow$ cliente. Dada una pila devuelve el cliente que primero se introdujo.
- primero: pila $\rightarrow$ cliente. Dada una pila devuelve el último cliente introducido.
- invertir: pila $\rightarrow$ pila. Dada una pila con k clientes ubicados en las posiciones $p_1, p_2, \cdots, p_k$ hace que el cliente en p_1 pase a la p_k, el cliente en la posición p_2 a la p_{k-1}, $\cdots$, y el de la posición p_k a la p_1.

- montar: pila, pila→ pila. Dadas dos pilas pone una encima de otra. Si tenemos P_a con k clientes, y P_b con q clientes, se devuelve una pila P' donde desde la posición p_1 hasta la p_k están los clientes que estaban en P_a respetando las posiciones, y a continuación entre las posiciones p_{k+1} hasta p_{k+q} se encuentran los clientes que estaban en P_b respetando dichas posiciones pero con k clientes por debajo.
- quitar: int, pila → pila. Dada una pila y un entero desapila n veces su cima.
- eliminarFondo: pila → pila. Dada una pila elimina el primer cliente que entró.
- apilar: cliente, pila → pila. Dada una pila y un cliente, lo apila en esta teniendo en cuenta que si es un cliente previamente registrado, podrá quedarse en la cima de la pila, pero si no se ha registrado, tendrá que quedarse debajo de los clientes registrados.
- borrar_Clientes: pila → pila. Borra todos los clientes de la pila.

1.2.4. Cola

Se usará la misma notación y suposición que en la especificación de operaciones 1.2.3.

- encolar: cliente, cola → cola. Añade un cliente a la cola.
- desencolar: cola → cliente, cola. Quita el cliente que más tiempo llevaba en la cola.
- inicio: cola → cliente. Devuelve el último cliente que menos tiempo lleva en la cola.
- fin: cola → cliente. Devuelve el cliente que más tiempo lleva en la cola.
- es_vacia: cola → bool. Comprueba si la cola tiene algún cliente almacenado o no.
- mostrarCola: cola → ___. Dada una cola muestra todo su contenido por pantalla (se apoya en otro método que va desencolando y encolando).
- invertir: cola → cola. Dada una cola con k clientes ubicados en las posiciones $p_1, p_2, \cdots, p_k$ hace que el cliente en p_1 pase a la p_k, el cliente en la posición p_2 a la p_{k-1}, $\cdots$, y el de la posición p_k a la p_1.
- borrar: cola → cola. Borra todos los clientes de la cola.

1.2.5. Lista

Se usará la misma notación y suposición que en las especificaciones de operaciones 1.2.3 y 1.2.4.

- esvacia: lista → bool. Comprueba si hay algún cliente en la lista.
- eliminar_inicial: lista → lista. Quita de la lista el cliente que se encuentra en la primera posición.

- eliminar_fin: lista → lista. Quita de la lista el cliente que se encuentra en la última posición.
- mostrar: lista → ___. Dada una lista muestra todo su contenido por pantalla .
- insertarIzquierda: cliente, lista → lista. Añade un cliente por la parte izquierda de la lista. No se podrá usar como tal, pero será la base de otras operaciones.
- insertarDerecha: cliente, lista → lista. Añade un cliente por la parte derecha de la lista. No se podrá usar como tal, pero será la base de otras operaciones.
- insertar: cliente, lista → lista. Añade un cliente a la lista teniendo en cuenta que primero estarán los clientes registrados ordenados en función de su hora de llegada (siendo la hora más próxima a las 10:00h la menor), y a continuación los no registrados, siendo el último el cliente no registrado que accedió a la web a la hora más lejana a las 10:00h.
- borrar_Lista: lista → lista. Borra todo el contenido de la lista.
- get_PrimeroNoRegistrado: lista → cliente. Recorre toda la lista hasta encontrar el primer cliente no registrado y lo devuelve.
- borrar_cliente: string → ___. Dado el id de un cliente, recorre la lista hasta encontrar dicho cliente y lo elimina. En caso de no encontrarlo, muestra un mensaje por pantalla.
- info_concierto: string → ___. Muestra por pantalla todos los clientes que tienen una entrada para ese concierto.
- mostrar_info_concierto: lista, string → ___. Dado el ID de un concierto, muestra por pantalla los datos de los clientes que tienen una entrada para asistir a este.
- info_PrimeroRegistrado: lista → ___. Muestra por pantalla los datos del primer cliente registrado en la lista.
- info_PrimeroNoRegistrado: lista → ___. Muestra por pantalla los datos del primer cliente no registrado en la lista.
- inicial: lista → cliente. Devuelve el primer cliente de la lista.
- fin: lista → cliente. Devuelve el último cliente de la lista.

1.2.6. Lista de entradas

Se usará la misma notación y suposición que en las especificaciones de operaciones 1.2.3, 1.2.4 y 1.2.5.

- getLongitud: lista → int. Devuelve el número de entradas en la lista.
- insertarEntrada: entrada, lista → lista. Dada una entrada la añade a la lista dada (por la derecha para simplificar).

- getPrimero: lista → entrada. Devuelve la primera entrada que compró el cliente.
- getUltimo: lista → entrada. Devuelve la última entrada que compró el cliente.
- mostrar: lista → ___. Dada una lista de entradas muestra todo su contenido por pantalla.
- eliminarEntrada: lista, entrada → lista. Dada una lista y una entrada, en caso de estar en la lista, la elimina de esta.
- contiene: lista, string, string → bool. Dada una lista, un ID de un concierto y el tipo del concierto, comprueba si esa entrada está en la lista proporcionada.
- contarEntradas: lista, string, string → int. Dada una lista y un concierto, calcula el número de entradas que hay para dicho concierto.

1.2.7. Árbol binario

Se usará la misma notación[1] y suposición que en las especificaciones de operaciones 1.2.3, 1.2.4, 1.2.5, y 1.2.6.

- insertarCliente (ι): árbol, cliente → árbol. Dado un árbol y un cliente, lo inserta en este teniendo en cuenta que los clientes registrados se posicionan en la parte izquierda del árbol y los no registrados en la derecha.
- insertarClientes: árbol, lista → árbol. Dada una lista de clientes y un árbol, los inserta en el árbol con ayuda de insertarCliente.
- mostrarInordenRegistrados (ι): árbol → ___. Dado un árbol muestra sus clientes ordenados por el código de cliente, siguiendo el recorrido en inorden:
  1. Visitar hijo izquierdo en inorden
  2. Visitar raíz
  3. Visitar hijo derecho en inorden
- mostrarInordenNoRegistrados (ι): árbol → ___. Misma explicación que para la función anterior, pero esta vez al ser no registrados se muestra su DNI.
- mostrarPreorden (ι): árbol → ___. Dado un árbol, muestra por pantalla los datos de todos los clientes almacenados en este, siguendo el recorrido en preorden:
  1. Visitar raíz
  2. Visitar hijo izquierdo en preorden
  3. Visitar hijo derecho en preorden

[1] Cuando exista otro método con mismo nombre pero que recibe diferentes argumentos (sobrecarga de métodos), lo indicaremos con la letra griega ι. Esto sucederá, sobre todo, en aquellos métodos que hagan uso de la recursividad, en los que a veces, se simplifican "separando en dos".

- mostrarInfoEntradas (ι): árbol, string $\rightarrow$ ___. Muestra por pantalla unas estadísticas del número de entradas de cada tipo del concierto dado por su nombre.
- borrarCliente (ι): árbol, string $\rightarrow$ árbol. Dado un árbol y el ID de un cliente, en caso de estar, lo elimina del árbol.
- infoCliente (ι): árbol, string $\rightarrow$ ___. Dado el ID de un cliente y un árbol, en caso de estar, muestra sus datos.
- borrar: árbol $\rightarrow$ árbol. Dado un árbol borra todo su contenido.
- esVacio: árbol $\rightarrow$ bool. Comprueba si un árbol tiene algún cliente.
- maximoCliente: árbol (nodo) $\rightarrow$ string. Encuentra el ID del cliente no registrado "más profundo".

1.2.8. Gestor

Se usará la misma notación y suposición que en las especificaciones de operaciones 1.2.3, 1.2.4, 1.2.5, 1.2.6, y 1.2.7.

- programa: ___ $\rightarrow$ ___. Arranca y lleva la lógica del menú y estructuras de datos de la aplicación.
- menu: ___ $\rightarrow$ char. Muestra por pantalla todas las tareas que se pueden llevar a cabo, pide elegir una de estas y devuelve la letra asociada a dicha tarea.
- crearPrimerosClientes: ^pila $\rightarrow$ ___. Genera de forma aleatoria clientes y los añade a la pila.
- crearSegundosClientes: ^cola, ^cola $\rightarrow$ ___. Crea de forma aleatoria clientes registrados y no registrados y los encola en sus respectivas colas.
- numero_Aleatorio: int, int $\rightarrow$ int. Genera un número aleatorio $n \in [a, b]$.
- reiniciar: ___ $\rightarrow$ ___. Elimina todo el contenido de las estructuras de datos y comienza el programa como si se hubiera vuelto a ejecutar la aplicación.
- get_nuevoID: ___ $\rightarrow$ string. Genera los ID de los usuarios registrados.

Capítulo 2

Solución adoptada

En esta parte se tratarán algunas de las **dificultades** que nos hemos encontrado realizando este trabajo hasta el momento.

En primer lugar, una de las cosas que nos ha complicado el trabajo ha sido que al trabajar en pareja, uno de los integrantes utilizaba Windows como sistema operativo, mientras que otro, una distribución de Linux. Esto generaba errores de compilación en puntos como llamadas a destructores u otras sentencias, como por ejemplo las directivas `#include`. Al parecer, las diferentes versiones del compilador detectan automáticamente o no el resto de archivos necesarios que se han utilizado durante el mismo. Sin embargo, hemos solucionado este problema con ayuda de un repositorio de GitHub donde cada integrante iba subiendo sus ficheros con los `#include` correctos para su compilador y el resto del código actualizado.
Otras dificultades que nos hemos encontrado han sido errores "absurdos", ya que como tal no son cosas complejas, pero si difíciles de detectar cuando hay muchas líneas de código, y además, el proyecto compila pero no hace lo esperado o devuelve un valor distinto de 0, quedando así sin saber en qué parte del código se encuentra el error.
Esto se ha dado hasta el momento en dos ocasiones durante el desarrollo del método `apilar` del TAD `pila` (operaciones 1.2.3). El primero de estos fallos se ocasionaba en una proposición lógica ubicada en la condición de un bucle `while` del tipo $p_1 \wedge p_2$, donde pretendíamos que con que una de las proposiciones fuera falsa no se ejecutara el bucle. El problema era que p_1 comprobaba si la pila era vacía, pero claro, p_2 preguntaba si un determinado objeto cliente estaba registrado. Esto se hacía con ayuda de un puntero, pero desafortunadamente, llegaba un momento donde al estar la pila vacía se intentaba aplicar un método de clase `cliente` al objeto que apuntaba el puntero, que en realidad era `NULL`.
El otro error cometido, y el más absurdo, fue en la parte `main` haciendo pruebas. Se asignaban varios valores a un mismo cliente por error al nombrar los objetos. Llegamos a pensar que estaba fallando el método, lo que nos llevó a diseñar tres algoritmos distintos para implementar el apilar clientes respetando las restricciones dadas, perdiendo bastante tiempo.
Además, al principio de la implementación del árbol binario de búsqueda (operaciones 1.2.7), tuvimos algunos problemas diseñando el método de borrar un cliente del árbol, debido a no tener acceso al padre para borrar el cliente en caso de haberlo encontrado. Finalmente, al pasar de un único main a una clase `Gestor` (código 4.23) como sugirió la profesora en la corrección de la PL1, obtuvimos problemas con las operaciones de cola (operaciones 1.2.4) al no estar implementadas de manera "legítima".

Capítulo 3

Diseño de relación entre clases

3.1. Diagrama UML

En esta sección se presenta un sencillo diagrama UML (figura 3.1) generado mediante técnicas de ingeniería inversa, que ayuda a entender de forma rápida la lógica del proyecto. Ha sido realizado con el software *Enterprise Architect*. Aparece a tamaño completo en la siguiente página.

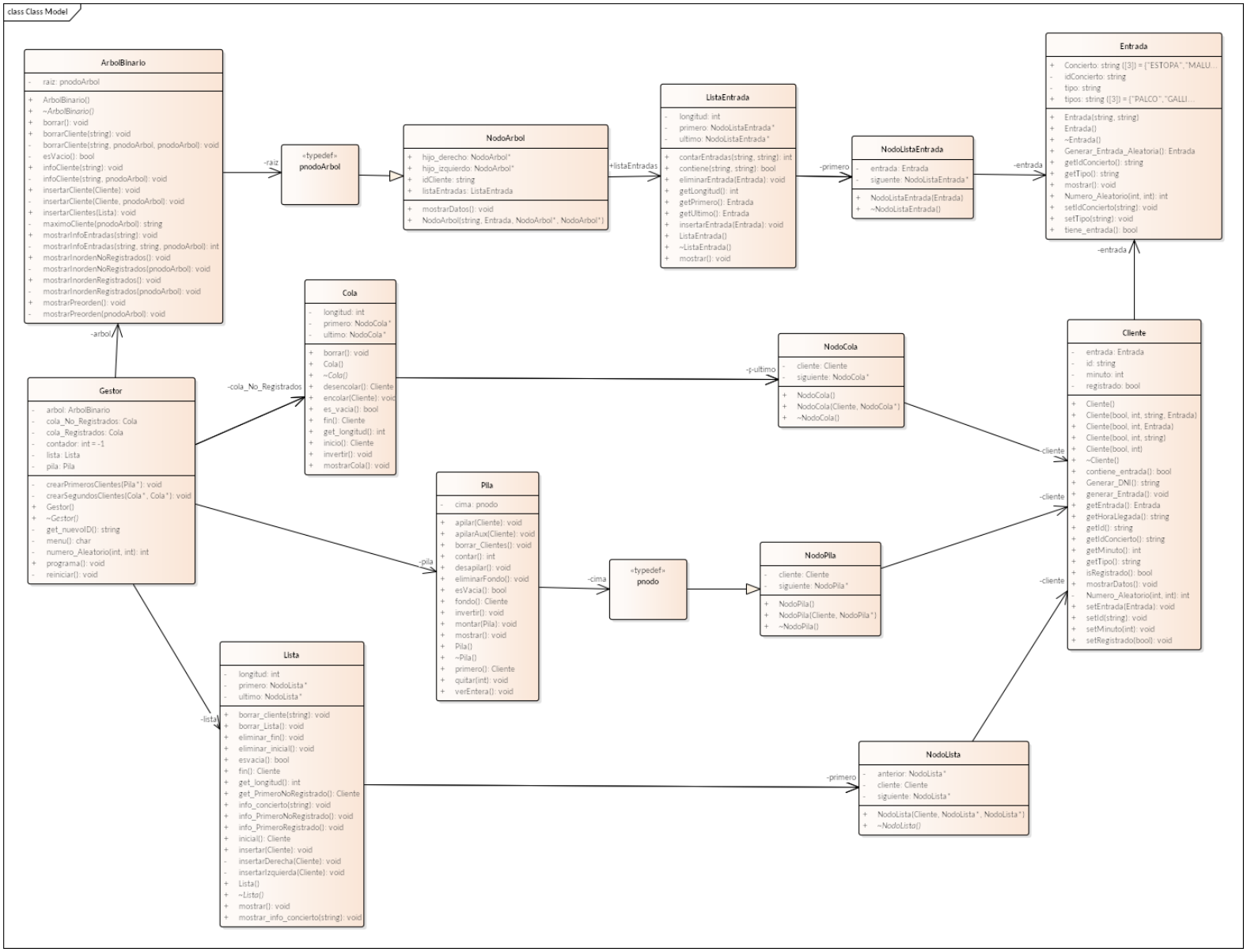

Figura 3.1: Diagrama UML

3.2. Métodos más destacables

Como métodos más destacables se citan a los que añaden clientes a las diferentes estructuras de datos, siendo estos: `encolar`, `apilar`, e `insertar`. Se podría decir que estos métodos son los más complejos, ya que la llamada a estos implica la llamada a otros muchos métodos de dicha clase. Además, como en la mayoría de casos se pide que ya se añadan ordenados según unos determinados criterios, también se necesitan hacer múltiples comprobaciones, haciendo así que el código sea algo más extenso y complejo. También debemos mencionar la inserción y borrado de un determinado cliente en el árbol binario, métodos algo más difíciles de contruir que el resto. A continuación procedemos a explicar de manera resumida que hace cada uno de estos:

- `apilar`: el algoritmo consiste en que, dado un cliente, lo apila en la pila. Si el cliente está registrado debe ir a la cima, si no, debajo de los registrados (Llamemos c al cliente dado). Funciona así:

  1. Si c está registrado, llamamos a la función `apilarAux` que apila un cliente en la cima sin más.
  2. Si c no está registrado, creamos una pila auxiliar `reg` donde almacenaremos temporalmente los clientes registrados que ya teníamos almacenados.
  3. Mientras el puntero que apunta a la cima de la pila apunte a algún cliente que esté registrado, lo llevamos a `reg`.
  4. Una vez estén todos los clientes registrados en `reg`, quiere decir que en la pila solo están los no registrados, pudiendo apilar directamente c.
  5. Una vez apilado c, deshacemos todo lo que hemos hecho, hacemos el algoritmo a la inversa. Mientras el puntero que apunta a la cima de `reg` apunte a algún cliente, apilamos en la pila y desapilamos de esta.
  6. Finalmente, el algoritmo termina con `reg` vacía y la pila principal separada en dos grandes bloques de registrados y no registrados, con c correctamente posicionado.

- `encolar`: dado un cliente c lo añade a la cola, sin necesidad de respetar ningún criterio de ordenación, ya que se crearán dos colas, una de registrados y otra de no registrados. Su funcionamiento es el siguiente:

  1. Se crea un puntero a un nuevo nodo de cola con el cliente.
  2. Se comprueba si la cola es vacía. Si es así, este nodo será el primero y último de la cola, por tanto los punteros de `primero` y `último` apuntarán a este nodo.
  3. Si no estaba vacía, `último` estaría apuntado a un nodo, luego podemos decir que el puntero al siguiente en vez de apuntar a `NULL` apunte a nuestro nodo.
  4. Finalmente, decimos que el puntero `último` apunte a nuestro nodo.

- `insertar`: recibe como parámetro el cliente que se desea insertar, y lo añade de forma ordenada a la lista. Funciona así:

1. Comprueba si c ya tiene asignada una entrada, y en caso de que no la tenga se le asigna.
2. A continuación se revisa si la lista está vacía, ya que en dicho caso no importa si se inserta por la derecha o por la izquierda.
3. En caso de que no esté vacía, se comprueba si el cliente es de tipo registrado o no, para insertarlos en su correspondiente apartado según los criterios marcados.
4. Dentro de su apartado correspondiente, se recorrerán los distintos clientes del apartado hasta encontrar uno que tenga una hora de llegada mayor que c, en ese momento habremos encontrado su posición.
5. Para poder insertarlo, antes tendremos que observar si tiene clientes tanto a su derecha como a su izquierda y modificar los punteros de estos para que apunten a c.

- `insertarCliente`: recibe como parámetros el cliente y el nodo en el que ha de ser insertado (irá cambiando gracias a la recursividad).

1. Se comprueba si el nodo en el que queremos insertar tiene hijos o no.
2. En caso de no tener hijos, si estamos en la raíz del árbol lo colocamos como su hijo izquierdo en caso de estar registrado, o como su hijo derecho en caso contrario.
3. Si no estamos en la raíz, si el código es "menor" se coloca como su hijo izquierdo, y si no, como su hijo derecho. En caso de igualdad, significa que dicho cliente ha comprado otra entrada y se la añadimos.
4. En caso de que el nodo tenga hijos, comprobamos si estamos en la raíz.
5. Si estamos en la raíz y el cliente está registrado, comprobamos si no tiene un hijo izquierdo y en dicho caso lo insertamos.
6. En caso de que sí tenga hijo izquierdo, volvemos al paso 1 pero queriendo situarlo como hijo izquierdo del hijo izquierdo del que nos encontramos.
7. En caso de sí estar registrado volvemos al paso 5 pero jugando con la derecha y no registrado en vez de la izquierda y registrado.
8. Por último, si no estamos en la raíz seguimos la misma lógica del paso 3.

- `borrarCliente`: recibe un cliente, un puntero al padre, y el nodo en el que borrar.

1. Comprueba si el nodo es raíz o no.
2. En caso de que lo sea, comprueba si el cliente que se quiere borrar es registrado o no y en función de eso, se llama al método de borrar de un hijo o de otro.
3. Si no es el nodo raíz, comprueba si el nodo tiene el mismo ID que el cliente que se busca.
4. En caso de ser el que se busca, comprueba primero si carece de hijos, y en dicho caso únicamente se tendrá que quitar el puntero a este nodo pero desde el nodo padre.

5. Si contiene únicamente el hijo izquierdo o el hijo derecho, cambia el puntero del padre para que en vez de apuntar al nodo que queremos borrar, apunte a su hijo directamente.
6. En caso de que contenga tanto el hijo izquierdo como el derecho, se buscará el nodo mayor dentro de hijo izquierdo, se guardará, se borrará de ese árbol, y se sustituirá por el que se quería borrar, actualizando correspondientemente los punteros.
7. Si no coincide el nodo con el que se busca, se compara con el valor ID del nodo y se comprueba si hay que buscarlo por el hijo izquierdo o por el hijo derecho.

3.3. Comportamiento del programa

El programa comienza con la ejecución del archivo principal `Principal.cpp`, que instancia un objeto de la clase `Gestor` e invoca al método `programa` mostrándose lo siguiente por pantalla y recogiendo la letra correspondiente a la tarea a realizar.

```
----------SISTEMA GESTOR DE VENTA DE ENTRADAS----------

a. Generar pila de clientes con reserva de entrada y dos colas de clientes registrados y no registrados
    que van a comprar
b. Generar N clientes con reserva de entrada e incluirlos a la pila
c. Generar N clientes registrados y M no registrados que acceden a la compra de entradas
d. Incluir manualmente un cliente en la pila
e. Incluir manualmente un cliente en la cola
f. Mostrar clientes en la pila
g. Borrar la pila de clientes
h. Mostrar la cola de clientes registrados en espera de compra
i. Mostrar la cola de clientes no registrados en espera de compra
j. Borrar las dos colas
k. Pasar a todos los clientes en espera a la fase de compra
l. Mostrar los datos de todos los clientes de la lista
m. Buscar en la lista y mostrar los datos del primer cliente registrado que ha comprado una entrada
n. Buscar en la lista y mostrar los datos del primer cliente no registrado que ha comprado entrada
o. Buscar en la lista y mostrar, dado el identificador de un concierto, los datos de todos los
    clientes que han comprado una entrada
p. Borrar los datos de un cliente de la lista, dado su identificador (DNI o codigo de cliente)
q. Borrar la lista de clientes
r. Simular que se finaliza la compra de entradas, sacando los clientes de la lista e insertándolos
    en el árbol binario de búsqueda en el orden indicado
s. Insertar un cliente manualmente en el árbol
t. Mostrar los datos de todos los clientes registrados, ordenados por código de cliente (inorden)
u. Mostrar los datos de todos los clientes no registrados, ordenados por DNI (inorden)
v. Borrar los datos de un cliente en el árbol, dado su identificador (DNI o código de cliente)
w. Buscar un cliente, en el árbol, y mostrar los datos de las entradas que ha comprado
x. Mostrar los datos de todos los clientes almacenados en el arbol (recorrido en preorden)
y. Mostrar una estadística del total de entradas vendidas para un concierto dado, desglosadas por
    tipo
z. Reiniciar el programa
$. Salir
      Introduzca tarea:
```

Capítulo 4

Código fuente

Nota: para la realización del código no se han utilizado tildes ni acentos en las palabras que deberían llevarlos (ya bien sea en comentarios o cadenas de caracteres), ya que, además de que no es una buena práctica (normalmente se programa en inglés), el paquete que permite insertar código en documentos generados con este software, no soporta caracteres que no estén incluidos en el sistema de codificación UTF-8 básico. A continuación se muestra el programa principal:

Código 4.1: Programa principal

```
1  #include "PracticaLab/Gestor.h"
2
3
4  #include <iostream>
5  #include <string>
6
7  using namespace std;
8
9
10 int main(){
11     Gestor gestor = Gestor();
12     gestor.programa();
13
14     return 0;
15 }
```

4.1. Cabeceras

Código 4.2: Cabecera de entrada

```
1  #ifndef ENTRADA_H_INCLUDED
2  #define ENTRADA_H_INCLUDED
3
4  #include <string>
5
6  using namespace std;
7
```

```
8  class Entrada{
9      private:
10         string tipo;
11         string idConcierto;
12
13         friend class Cliente;
14
15     public:
16         Entrada(string tipo, string idConcierto); //Constructor
17         Entrada(); //Sobrecarga de constructor
18         ~Entrada(); //Destructor
19
20         //Getters
21         string getTipo();
22         string getIdConcierto();
23
24         //Setters
25         void setTipo(string tipo);
26         void setIdConcierto(string idConcierto);
27
28         Entrada Generar_Entrada_Aleatoria();
29         int Numero_Aleatorio(int DESDE, int HASTA);
30         void mostrar();
31         bool tiene_entrada();
32         string tipos[3] = {"PALCO","GALLINERO","PATIO"} ;
33         string Concierto[3] = {"ESTOPA","MALU","MELENDI"};
34 };
35
36
37 #endif // ENTRADA_H_INCLUDED
```

Código 4.3: Cabecera de cliente

```
1  #ifndef CLIENTE_H_INCLUDED
2  #define CLIENTE_H_INCLUDED
3
4  #include <string>
5  #include "Entrada.h"
6
7  using namespace std;
8
9  class Cliente{
10     private:
11         bool registrado;
12         int minuto;
13         string id;
14         Entrada entrada;
15         int Numero_Aleatorio(int DESDE, int HASTA);
16
17
18     public:
19         Cliente(); //Constructor
20         Cliente(bool registrado, int minuto, string id, Entrada entrada);
21         Cliente(bool registrado, int minuto, Entrada entrada);
```

```
22          Cliente(bool registrado, int minuto, string id);
23          Cliente(bool registrado, int minuto);
24          ~Cliente(); //Destructor
25
26          Entrada getEntrada();
27          bool isRegistrado();
28          int getMinuto();
29          string getId();
30          string getTipo();
31          string getIdConcierto();
32          string getHoraLlegada();
33
34          void generar_Entrada();
35          void setRegistrado(bool estado);
36          void setMinuto(int minuto);
37          void setId(string id);
38          void setEntrada(Entrada entrada);
39          void mostrarDatos();
40          bool contiene_entrada();
41          string Generar_DNI();
42  };
43
44  #endif
```

Código 4.4: Cabecera del nodo de pila

```
1   #ifndef NODOPILA_H_INCLUDED
2   #define NODOPILA_H_INCLUDED
3
4   #include "Cliente.h"
5
6   class NodoPila{
7       private:
8           //Atributos
9           Cliente cliente;
10          NodoPila *siguiente;  //Puntero al elemento que esta "debajo"
11
12          friend class Pila;
13
14      public:
15          //Constructores
16          NodoPila();
17          NodoPila(Cliente cliente, NodoPila *s=NULL);
18
19          //Destructor
20          ~NodoPila();
21  };
22
23  typedef NodoPila *pnodo;
24
25  #endif // NODOPILA_H_INCLUDED
```

Código 4.5: Cabecera de la pila

```
1   #ifndef PILA_H_INCLUDED
```

```
2  #define PILA_H_INCLUDED
3
4  #include "NodoPila.h"
5
6  class Pila{
7      private:
8          //Atributos
9          pnodo cima;
10
11     public:
12         //Constructor y destructor
13         Pila();
14         ~Pila();
15
16         //Metodos
17         bool esVacia();
18         void apilarAux(Cliente cliente);
19         void desapilar();
20         void mostrar();
21         void verEntera();
22         int contar();
23         Cliente fondo();
24         Cliente primero();
25         void invertir();
26         void montar(Pila pila);
27         void quitar(int n);
28         void eliminarFondo();
29         void apilar(Cliente c);
30         void borrar_Clientes();
31
32 };
33
34 #endif // PILA_H_INCLUDED
```

Código 4.6: Cabecera del nodo de cola

```
1  #ifndef NODOCOLA_H
2  #define NODOCOLA_H
3  #include <iostream>
4  #include "Cliente.h"
5  class NodoCola{
6      private:
7          Cliente cliente;
8          NodoCola* siguiente;
9          friend class Cola;
10
11     public:
12         NodoCola();
13         NodoCola(Cliente cliente, NodoCola *sig = NULL);
14         ~NodoCola();
15 };
16
17 #endif // NODOCOLA_H
```

Código 4.7: Cabecera de la cola

```
#ifndef COLA_H
#define COLA_H

#include "NodoCola.h"

#include <iostream>

class Cola{
    private:
        NodoCola *primero;
        NodoCola *ultimo;
        int longitud;

    public:
        Cola();
        void encolar(Cliente);
        Cliente desencolar();
        Cliente inicio();
        Cliente fin();
        bool es_vacia();
        int get_longitud();
        void mostrarCola();
        void mostrarDatosCola();
        virtual ~Cola();
        void invertir();
        void borrar();


};

#endif // COLA_H
```

Código 4.8: Cabecera del nodo de lista

```
#ifndef NODOLISTA_H
#define NODOLISTA_H

#include <iostream>
#include "Cliente.h"
class NodoLista{
    public:
        NodoLista(Cliente clien, NodoLista *sig=NULL, NodoLista *ant=NULL)
            ;
        virtual ~NodoLista();
        friend class Lista;

    private:
         Cliente cliente;
         NodoLista* siguiente;
         NodoLista* anterior;
};

#endif // NODOLISTA_H
```

Código 4.9: Cabecera de la lista

```
#ifndef LISTA_H
#define LISTA_H

#include "NodoLista.h"
#include "Cliente.h"

#include <iostream>

class Lista{
    public:
        Lista();

        virtual ~Lista();
        bool esvacia();
        int get_longitud();
        void insertar(Cliente clien);
        Cliente inicial();
        Cliente fin();

        Cliente get_PrimeroNoRegistrado();

        void eliminar_inicial();
        void eliminar_fin();

        void mostrar();
        void borrar_Lista();
        void info_PrimeroRegistrado();
        void info_PrimeroNoRegistrado();
        void borrar_cliente(string id);
        void info_concierto(string id_concierto);
        void mostrar_info_concierto(string id_concierto);

    private:
        NodoLista* primero;
        NodoLista* ultimo;
        int longitud;

        void insertarIzquierda(Cliente clien);
        void insertarDerecha(Cliente clien);

};

#endif // LISTA_H
```

Código 4.10: Cabecera de la lista de entradas

```
#ifndef LISTAENTRADA_H_INCLUDED
#define LISTAENTRADA_H_INCLUDED

#include "NodoListaEntrada.h"

class ListaEntrada{
    public:
```

```
8           ListaEntrada();
9           ~ListaEntrada();
10          int getLongitud();
11          void insertarEntrada(Entrada entrada);
12          Entrada getPrimero();
13          Entrada getUltimo();
14          void mostrar();
15          void eliminarEntrada(Entrada entrada);
16          bool contiene(string idConcierto, string tipo);
17          int contarEntradas(string idConcierto, string tipo);
18      private:
19          NodoListaEntrada *primero;
20          NodoListaEntrada *ultimo;
21          int longitud;
22  };
23
24
25  #endif // LISTAENTRADA_H_INCLUDED
```

Código 4.11: Cabecera del árbol binario

```
1   #ifndef ARBOLBINARIO_H
2   #define ARBOLBINARIO_H
3
4   #include "Lista.h"
5   #include "NodoArbol.h"
6
7
8   class ArbolBinario
9   {
10      public:
11          ArbolBinario();
12          virtual ~ArbolBinario();
13          void insertarCliente(Cliente cliente);
14          void insertarClientes(Lista lista);
15          void mostrarInordenRegistrados();
16          void mostrarInordenNoRegistrados();
17          void mostrarPreorden();
18          void mostrarInfoEntradas(string idConcierto);
19          void borrarCliente(string id);
20          void infoCliente(string id);
21          void borrar();
22      private:
23          void mostrarPreorden(pnodoArbol nodo);
24          void infoCliente(string id, pnodoArbol nodo);
25          void borrarCliente(string id,pnodoArbol nodo, pnodoArbol padre);
26          void insertarCliente(Cliente cliente,pnodoArbol nodo);
27          int mostrarInfoEntradas(string idConcierto,string tipo, pnodoArbol
                nodo);
28          pnodoArbol raiz;
29          bool esVacio();
30          string maximoCliente(pnodoArbol nodo);
31          void mostrarInordenRegistrados(pnodoArbol nodo);
32          void mostrarInordenNoRegistrados(pnodoArbol nodo);
```

```
};

#endif // ARBOLBINARIO_H
```

Código 4.12: Cabecera del gestor

```
#ifndef GESTOR_H
#define GESTOR_H

#include "Pila.h"
#include "Cola.h"
#include "Lista.h"
#include "ArbolBinario.h"

class Gestor
{
    public:
        Gestor();
        virtual ~Gestor();
        void programa();
    private:
        Pila pila;
        Cola cola_Registrados;
        Cola cola_No_Registrados;
        Lista lista;
        ArbolBinario arbol;
        int contador=-1;
        string get_nuevoID();
        char menu();
        void crearSegundosClientes(Cola *cola_Regis, Cola *cola_No_Regis);
        void crearPrimerosClientes(Pila *pila);
        void reiniciar();
        int numero_Aleatorio(int DESDE, int HASTA);
};

#endif // GESTOR_H
```

4.2. Clases

Código 4.13: Implementación de entrada

```
#include <iostream>
#include <string>

#include "Entrada.h"

using namespace std;

string tipos[3] = {"PALCO","GALLINERO","PATIO"};
string Concierto[3] = {"ESTOPA","MALU","MELENDI"};

Entrada::Entrada(string tipo, string idConcierto){
```

```
12      this->tipo=tipo;
13      this->idConcierto=idConcierto;
14  }
15
16  Entrada::Entrada(){
17  }
18
19  Entrada::~Entrada(){
20  }
21
22  string Entrada::getTipo(){
23      return this->tipo;
24  }
25
26  string Entrada::getIdConcierto(){
27      return this->idConcierto;
28  }
29
30  void Entrada::setTipo(string tipo){
31      this->tipo=tipo;
32  }
33
34  void Entrada::setIdConcierto(string idConcierto){
35      this->idConcierto=idConcierto;
36  }
37
38  void Entrada::mostrar(){
39      char caracter=192;
40      cout<<"   "<<caracter<<"Tipo: "<<this->tipo<<", idConcierto: "<<this->
            idConcierto<<endl;
41  }
42
43  int Entrada::Numero_Aleatorio(int DESDE, int HASTA){
44    return rand()%(HASTA-DESDE+1)+DESDE;
45
46  }
47
48  bool Entrada::tiene_entrada(){
49      if(this->idConcierto!="" && this->tipo!=""){
50          return true;
51      }
52      else{
53          return false;
54      }
55  }
56
57  Entrada Entrada::Generar_Entrada_Aleatoria(){
58      Entrada entrada;
59
60      entrada.setTipo(tipos[Numero_Aleatorio(0,2)]);
61      entrada.setIdConcierto(Concierto[Numero_Aleatorio(0,2)]);
62
63      return entrada;
64  }
```

Código 4.14: Implementación de cliente

```cpp
#include <iostream>
#include <string>
#include <stdio.h>

#include "Cliente.h"
#include "Entrada.h"
#include "Entrada.cpp"



using namespace std;

Cliente::Cliente(bool registrado, int minuto, string id, Entrada entrada){
    this->registrado=registrado;
    this->minuto=minuto;
    this->id=id;
    this->entrada=entrada;
}

Cliente::Cliente(bool registrado, int minuto, Entrada entrada){
    this->registrado=registrado;
    if(!this->isRegistrado()){
        this->id=Generar_DNI();
    }
    else{
        this->id="CODREG000";
    }
    this->minuto=minuto;
    this->entrada=entrada;
}

Cliente::Cliente(bool registrado, int minuto, string id){
    this->registrado=registrado;
    this->minuto=minuto;
    this->id=id;
}

Cliente::Cliente(bool registrado, int minuto){
    this->registrado=registrado;
    if(!this->isRegistrado()){
        this->id=Generar_DNI();
    }
    else{
        this->id="CODREG000";
    }
    this->minuto=minuto;
}

Cliente::Cliente(){
    this->id="CODREG000";
    this->entrada=Entrada();
}
```

```
53
54  Cliente::~Cliente(){
55  }
56
57  bool Cliente::isRegistrado(){
58      return this->registrado;
59  }
60
61  int Cliente::getMinuto(){
62      return this->minuto;
63  }
64
65  string Cliente::getHoraLlegada(){
66      string cadena = "10:"+to_string(this->minuto);
67      return cadena;
68  }
69
70  string Cliente::getId(){
71      return this->id;
72  }
73
74  Entrada Cliente::getEntrada(){
75      return this->entrada;
76  }
77
78  string Cliente::getTipo(){
79      return this->getEntrada().getTipo();
80  }
81
82  string Cliente::getIdConcierto(){
83      return this->getEntrada().getIdConcierto();
84  }
85
86  void Cliente::setRegistrado(bool estado){
87      this->registrado=estado;
88  }
89
90  void Cliente::setMinuto(int minuto){
91      this->minuto=minuto;
92  }
93
94  void Cliente::setId(string id){
95      this->id=id;
96  }
97
98  void Cliente::setEntrada(Entrada entrada){
99      this->entrada=entrada;
100 }
101
102 int Cliente::Numero_Aleatorio(int DESDE, int HASTA){
103   return rand()%(HASTA-DESDE+1)+DESDE;
104 }
105
106 bool Cliente::contiene_entrada(){
```

```
107     bool resul=false;
108     if(this->entrada.tiene_entrada()){
109         resul=true;
110     }
111     return resul;
112 }
113 
114 void Cliente::generar_Entrada(){
115     this->entrada=this->entrada.Generar_Entrada_Aleatoria();
116 }
117 
118 void Cliente::mostrarDatos(){
119 
120         string cadena ="ID: "+ this->getId() + " Hora de llegada: "+this->
               getHoraLlegada();
121         string entrada_cliente;
122         if(this->contiene_entrada()){
123                 entrada_cliente=" concierto: " + this->getIdConcierto()+"
                       tipo: "+this->getTipo();
124         }else{
125             entrada_cliente="";
126         }
127         cout<< cadena << entrada_cliente<<endl;
128 
129 }
130 
131 string Cliente::Generar_DNI(){
132     char letras[]="TRWAGMYFPDXBNJZSQVHLCKE\0";
133     string DNI="";
134     int numero,posicion_array;
135     numero=Numero_Aleatorio(10000000,99999999);
136     posicion_array=numero - (numero / 23 * 23);
137     DNI = to_string(numero)+letras[posicion_array];
138     return DNI;
139 }
```

Código 4.15: Implementación del nodo de pila

```
1  #include <iostream>
2  #include "NodoPila.h"
3  
4  using namespace std;
5  
6  NodoPila::NodoPila(){
7      this->cliente=Cliente();
8      siguiente=NULL;
9  }
10 
11 NodoPila::NodoPila(Cliente cliente, NodoPila *s){
12     this->cliente=cliente;
13     siguiente=s;
14 }
15 
16 NodoPila::~NodoPila(){
```

```
17 }
```

Código 4.16: Implementación de la pila

```
1  #include <iostream>
2
3  #include "Pila.h"
4  #include "NodoPila.h"
5  #include "NodoPila.cpp"
6
7
8  using namespace std;
9
10 Pila::Pila(){
11     //Constructor
12     this->cima=NULL;
13 }
14
15 Pila::~Pila(){
16     //Destructor
17     while(this->cima){
18         desapilar();
19     }
20 }
21
22 bool Pila::esVacia(){
23     //Comprueba si la pila esta vacia
24     return this->cima==NULL;
25 }
26
27 void Pila::apilarAux(Cliente cliente){
28     //Apila un cliente en la pila
29     pnodo n=new NodoPila(cliente, this->cima); //Un nuevo nodo almacenara
           al cliente y su elemento anterior sera la cima actual
30     this->cima=n; //Ahora la cima es el nuevo nodo
31 }
32
33 void Pila::desapilar(){
34     //Desapila un cliente de la pila
35     pnodo nodo;
36     if(this->cima){
37         nodo=this->cima;
38         this->cima=nodo->siguiente;
39         //delete nodo;
40         //Si hay elemento por debajo, la cima pasa a ser el elemento de
               debajo
41     }
42 }
43
44 void Pila::mostrar(){
45     //Muestra la cima de la pila en caso de que no este vacia
46     if(this->esVacia()){
47         cout<<"\nPila vacia"<<endl;
48     }
```

```
49      else{
50          cout<<"\nCima de la pila: "<<this->cima->cliente.getId()<<endl;
51      }
52  }
53
54  void Pila::verEntera(){
55      Pila aux;
56      aux.cima=this->cima;
57      if(aux.cima){
58          while(aux.cima){
59              aux.cima->cliente.mostrarDatos();
60              aux.desapilar();
61          }
62      }else{
63          cout << "Pila vacia"<<endl;
64      }
65  }
66
67  int Pila::contar(){
68      //Cuenta cuantos clientes hay en una pila
69      Pila aux;
70      aux.cima=this->cima;
71      int clientes=0;
72
73      while(aux.cima){
74          clientes++;
75          aux.desapilar();
76      }
77
78      return clientes;
79  }
80
81  Cliente Pila::fondo(){
82      //Devuelve el cliente del fondo
83      Pila aux;
84      aux.cima=this->cima;
85
86      while(aux.contar()>1){
87          aux.desapilar();
88      }
89
90      return aux.cima->cliente;
91  }
92
93  Cliente Pila::primero(){
94      return this->cima->cliente;
95  }
96
97  void Pila::invertir(){
98      //Invierte la pila de clientes
99      Pila aux;
100     while(!this->esVacia()){
101         aux.apilarAux(this->cima->cliente);
102         this->desapilar();
```

```
103     }
104
105     this->cima=aux.cima;
106 }
107
108 void Pila::montar(Pila pila){
109     //Pone una pila de clientes encima de otra
110     Pila aux=pila;
111     aux.invertir();
112
113     while(!aux.esVacia()){
114         this->apilarAux(aux.cima->cliente);
115         aux.desapilar();
116     }
117 }
118
119 void Pila::quitar(int n){
120     //Elimina los n primeros clientes de la pila
121     for(int i=0; i<n; i++){
122         if(!this->esVacia()){
123             this->desapilar();
124         }
125     }
126 }
127
128 void Pila::borrar_Clientes(){
129     while(!this->esVacia()){
130         this->desapilar();
131     }
132     cout<<"Se ha borrado el contenido de la pila"<<endl;
133 }
134
135 void Pila::eliminarFondo(){
136     this->invertir();
137     this->desapilar();
138     this->invertir();
139 }
140
141 void Pila::apilar(Cliente c){
142     if(c.isRegistrado()){
143         this->apilarAux(c);
144     }
145     else{
146         Pila reg;
147         while(this->cima && this->cima->cliente.isRegistrado()){
148             reg.apilarAux(this->cima->cliente);
149             this->desapilar();
150         }
151
152         this->apilarAux(c);
153         while(reg.cima){
154             this->apilarAux(reg.cima->cliente);
155             reg.desapilar();
156         }
```

```
157     }
158 }
```

Código 4.17: Implementación del nodo de cola

```
1  #include<iostream>
2
3  #include "NodoCola.h"
4
5  NodoCola::NodoCola(){
6      this->siguiente=NULL;
7      this->cliente = Cliente();
8
9  }
10
11 NodoCola::NodoCola(Cliente cliente, NodoCola* sig){
12     this->cliente = cliente;
13     this->siguiente = sig;
14 }
15
16
17 NodoCola::~NodoCola(){
18
19 }
```

Código 4.18: Implementación de la cola

```
1  #include <iostream>
2  #include <string>
3  #include <sstream>
4
5
6  #include "Cola.h"
7  #include "NodoCola.h"
8  #include "Cliente.h"
9  #include "Cliente.cpp"
10
11 using namespace std;
12
13 Cola::Cola(){
14     //Constructor
15     primero=NULL;
16     ultimo=NULL;
17     longitud=0;
18 }
19
20 Cola::~Cola(){
21 }
22
23 void Cola::encolar(Cliente cliente){
24     //Encola un lciente en la cola
25     NodoCola *nodo=new NodoCola(cliente);
26     if(es_vacia()){
27         this->primero=nodo;
28         this->ultimo=nodo;
```

```
29      }
30      else{
31          this->ultimo->siguiente=nodo;
32          this->ultimo=nodo;
33      }
34      longitud++;
35  }
36
37  Cliente Cola::desencolar(){
38      //Desencola un cliente de la cola
39      Cliente  cliente;
40      if(!this->es_vacia()){
41          cliente=primero->cliente;
42          NodoCola *aux=this->primero;
43          if(this->primero==this->ultimo){
44              this->primero=NULL;
45              this->ultimo=NULL;
46              aux->siguiente=NULL;
47              delete(aux);
48          }
49          else{
50              this->primero=this->primero->siguiente;
51              aux->siguiente=NULL;
52              delete(aux);
53          }
54          longitud--;
55      }
56      return cliente;
57  }
58
59  bool Cola::es_vacia(){
60      return this->longitud!=0?false:true;
61  }
62
63  Cliente Cola::inicio(){
64      return this->primero->cliente;
65  }
66
67  Cliente Cola::fin(){
68      return this->ultimo->cliente;
69  }
70
71  int Cola::get_longitud(){
72      return this->longitud;
73  }
74
75  void Cola::borrar(){
76      while(!this->es_vacia()){
77          this->desencolar();
78      }
79      cout<<"Se ha borrado el contenido de la cola"<<endl;
80  }
81
82  void Cola::mostrarCola(){
```

```
83      if(this->primero){
84          this->mostrarDatosCola();
85          this->invertir();
86      }else
87          cout<<"La cola esta vacia"<<endl;
88  }
89
90  void Cola::mostrarDatosCola(){
91      NodoCola *actual=this->primero;
92
93     if(actual){ //Miro si la cola no esta vacia
94          Cliente a=actual->cliente; //Me guardo el primer cliente
95          actual->cliente.mostrarDatos(); //Muestro sus datos
96          desencolar(); //Lo elimino
97          mostrarDatosCola(); //Llamo al mismo metodo para que muestre el
                resto de los clientes
98          encolar(a); //Vuelvo insertar los clientes para dejar la cola
                intacta
99      }
100 }
101
102 void Cola::invertir(){
103     if(!this->es_vacia()){
104         Cliente c=this->inicio();
105         this->desencolar();
106         this->invertir();
107         this->encolar(c);
108     }
109 }
```

Código 4.19: Implementación del nodo de lista

```
1  #include<iostream>
2  #include "NodoLista.h"
3
4  NodoLista::NodoLista(Cliente clien,NodoLista* sig,NodoLista* ant)
5  {
6      this->cliente=clien;
7      this->siguiente=sig;
8      this->anterior=ant;
9  }
10
11 NodoLista::~NodoLista(){
12 }
```

Código 4.20: Implementación de la lista

```
1  #include <iostream>
2  #include <string>
3  #include <sstream>
4
5
6  #include "Lista.h"
7  #include "Cliente.h"
8  #include "NodoLista.h"
```

```
9
10
11  Lista::Lista(){
12      this->primero=NULL;
13      this->ultimo=NULL;
14      this->longitud=0;
15  }
16
17  Lista::~Lista(){
18  }
19
20  void Lista::insertarIzquierda(Cliente clien){
21      NodoLista *nodo = new NodoLista(clien);
22      if(this->esvacia()){
23          this->primero=nodo;
24          this->ultimo=nodo;
25      }
26      else{
27          nodo->siguiente=this->primero;
28          nodo->anterior=NULL;
29          this->primero->anterior=nodo;
30          this->primero=nodo;
31      }
32      this->longitud++;
33  }
34
35  void Lista::insertarDerecha(Cliente clien){
36      NodoLista *nodo = new NodoLista(clien);
37      if(this->esvacia()){
38          this->primero=nodo;
39          this->ultimo=nodo;
40      }
41      else{
42          nodo->anterior=this->ultimo;
43          nodo->siguiente=NULL;
44          this->ultimo->siguiente=nodo;
45          this->ultimo=nodo;
46      }
47      this->longitud++;
48  }
49
50  int Lista::get_longitud(){
51      return this->longitud;
52  }
53
54  void Lista::insertar(Cliente clien){
55      if(!clien.contiene_entrada()){
56          clien.generar_Entrada();
57      }
58      if(this->esvacia()){
59          this->insertarIzquierda(clien);
60      }
61      else{
62          if(clien.isRegistrado()){//Clientres Registrados
```

```
63              NodoLista *aux=this->primero;
64
65              while(aux->siguiente!= NULL && aux->cliente.isRegistrado()==
                    true && clien.getMinuto()>aux->cliente.getMinuto()){
66                  aux=aux->siguiente;
67              }
68
69
70              if(aux->siguiente == NULL && clien.getMinuto()>aux->cliente.
                    getMinuto()){
71                  this->insertarDerecha(clien);
72              }
73              else{
74
75                   if(aux->anterior==NULL && clien.getMinuto() <= aux->
                        cliente.getMinuto()){
76                      this->insertarIzquierda(clien);
77                  }
78                  else{
79                      NodoLista *nodo = new NodoLista(clien);
80
81                      nodo->anterior=aux->anterior;
82                      nodo->siguiente=aux;
83
84                      if(aux->anterior != NULL){
85                          aux->anterior->siguiente=nodo;
86                      }else{
87                          this->primero=nodo;
88                      }
89                      aux->anterior=nodo;
90                      this->longitud++;
91                  }
92              }
93          }
94          else{//Clientes NO registrados
95              NodoLista *aux=this->primero;
96
97              while (aux->siguiente!= NULL && aux->cliente.isRegistrado()==
                    true){//paso los que estan registrados
98                  aux=aux->siguiente;
99              }
100             while(aux->siguiente!= NULL && clien.getMinuto()>aux->cliente.
                    getMinuto()){
101                 aux=aux->siguiente;
102             }
103             if(aux->siguiente == NULL && aux->cliente.isRegistrado()){
104                 this->insertarDerecha(clien);
105             }
106             else{
107
108                 if(aux->siguiente == NULL && clien.getMinuto()>aux->
                        cliente.getMinuto()){
109                     this->insertarDerecha(clien);
110                 }
```

```
111                 else{
112
113                     if(aux->anterior==NULL && clien.getMinuto() <= aux->
                            cliente.getMinuto()){
114                         this->insertarIzquierda(clien);
115                     }
116                     else{
117                         NodoLista *nodo = new NodoLista(clien);
118
119                         nodo->anterior=aux->anterior;
120                         nodo->siguiente=aux;
121
122                         if(aux->anterior != NULL){
123                             aux->anterior->siguiente=nodo;
124                         }else{
125                             this->primero=nodo;
126                         }
127                         aux->anterior=nodo;
128                         this->longitud++;
129                     }
130                 }
131             }
132
133         }
134
135     }
136
137 }
138
139 Cliente Lista::inicial(){
140     return this->primero->cliente;
141 }
142
143 Cliente Lista::fin(){
144     return this->ultimo->cliente;
145 }
146
147 void Lista::eliminar_inicial(){
148     if(!this->esvacia()){
149         this->primero = this->primero->siguiente;
150         if(this->primero==NULL){
151             this->ultimo=NULL;
152         }
153         else{
154             this->primero->anterior=NULL;
155         }
156         this->longitud=longitud-1;
157     }
158 }
159
160 void Lista::eliminar_fin(){
161     if(!this->esvacia()){
162         this->ultimo=this->ultimo->anterior;
163         if(this->ultimo==NULL){
```

```
164             this->primero=NULL;
165         }
166         else{
167             this->ultimo->siguiente=NULL;
168         }
169         this->longitud--;
170     }
171 }
172
173 bool Lista::esvacia(){
174     return this->longitud==0?true:false;
175 }
176
177 void Lista::mostrar(){
178     if(this->primero){
179        NodoLista *aux=this->primero;
180
181        while(aux->siguiente){
182             aux->cliente.mostrarDatos();
183             aux=aux->siguiente;
184         }
185             aux->cliente.mostrarDatos();
186     }
187     else{
188         cout<<"Lista vacia"<<endl;
189     }
190
191 }
192
193 Cliente Lista::get_PrimeroNoRegistrado(){
194     NodoLista *aux=this->primero;
195
196     while(aux->siguiente && aux->cliente.isRegistrado()){
197         aux=aux->siguiente;
198     }
199     return aux->cliente;
200 }
201
202 void Lista::info_PrimeroRegistrado(){
203     if(!this->esvacia()){
204         Cliente cliente = this->inicial();
205         if(cliente.isRegistrado())
206             cliente.mostrarDatos();
207         else
208             cout << "La lista no contiene ningun cliente registrado"<<endl
                    ;
209     }
210     else{
211         cout <<"Lista vacia, no puedo mostrar informacion del primer
                cliente"<<endl;
212     }
213 }
214
215 void Lista::info_PrimeroNoRegistrado(){
```

```
216     if(!this->esvacia()){
217         Cliente cliente = this->get_PrimeroNoRegistrado();
218         if(!cliente.isRegistrado())
219             cliente.mostrarDatos();
220         else
221             cout<<"Error"<<endl;
222     }
223     else{
224         cout <<"Lista vacia, no puedo mostrar informacion del primer
                cliente"<<endl;
225     }
226 }
227
228 void Lista::borrar_Lista(){
229
230     while(!this->esvacia()){
231         this->eliminar_inicial();
232     }
233     cout<<"Se ha borrado el contenido de la lista"<<endl;
234 }
235
236 void Lista::borrar_cliente(string id){
237     if(this->primero){
238     NodoLista *aux=this->primero;
239     bool cliente_borrado=false;
240     while(aux->siguiente&& !cliente_borrado){
241             if(aux->cliente.getId()==id){
242
243
244                 if(aux->anterior==NULL && aux->siguiente==NULL){
245                     this->primero=NULL;
246                     this->ultimo=NULL;
247                 }else
248
249                 if(aux->anterior==NULL){
250                     this->primero=aux->siguiente;
251                     this->primero->anterior=NULL;
252                 }else
253
254                 if(aux->siguiente==NULL){
255                     this->ultimo=aux->anterior;
256                     this->ultimo->siguiente=NULL;
257                 }
258                 else{
259                     aux->siguiente->anterior=aux->anterior;
260                     aux->anterior->siguiente=aux->siguiente;
261                     }
262
263                 this->longitud=longitud-1;
264                 cliente_borrado=true;
265
266             }
267             else{
268                 aux=aux->siguiente;
```

```
269             }
270         }
271 
272         if(cliente_borrado){
273             cout<<"Se ha borrado el cliente con el id introducido"<<endl;
274         }
275         else{
276             cout<<"No se ha encontrado un cliente con ese id"<<endl;
277         }
278     }
279     else{
280         cout<<"No existe dicho cliente"<<endl;
281     }
282 }
283 
284 void Lista::mostrar_info_concierto(string id_concierto){
285 
286     if(this->primero){
287         NodoLista *aux=this->primero;
288 
289        while(aux->siguiente){
290                 if(aux->cliente.getIdConcierto()==id_concierto){
291                     aux->cliente.mostrarDatos();
292                   }
293                 aux=aux->siguiente;
294 
295             }
296             if(aux->cliente.getIdConcierto()==id_concierto){
297               aux->cliente.mostrarDatos();
298             }
299     }
300     else{
301         cout<<"No hay datos"<<endl;
302     }
303 
304 }
305 
306 void Lista::info_concierto(string id_concierto){
307     if(id_concierto=="ESTOPA" ||id_concierto=="MALU" ||id_concierto=="
        MELENDI"){
308                 mostrar_info_concierto(id_concierto);
309     }
310     else{
311         cout <<"No se ha introducido un concierto correcto"<<endl;
312     }
313 }
```

Código 4.21: Implementación de la lista de entradas

```
1 #include "ListaEntrada.h"
2 #include "NodoListaEntrada.h"
3 #include "NodoListaEntrada.cpp"
4 
5 ListaEntrada::ListaEntrada(){
```

```
6       this->primero=NULL;
7       this->ultimo=NULL;
8       this->longitud=0;
9   }
10
11  ListaEntrada::~ListaEntrada(){
12  }
13
14  int ListaEntrada::getLongitud(){
15      return this->longitud;
16  }
17
18  Entrada ListaEntrada::getPrimero(){
19      return this->primero->entrada;
20  }
21
22  Entrada ListaEntrada::getUltimo(){
23      return this->ultimo->entrada;
24  }
25
26  void ListaEntrada::insertarEntrada(Entrada entrada){
27      NodoListaEntrada *nodo=new NodoListaEntrada(entrada);
28
29      if(this->getLongitud()==0){
30          this->primero=nodo;
31          this->ultimo=nodo;
32      }
33      else{
34          this->ultimo->siguente=nodo;
35          this->ultimo=nodo;
36      }
37
38      longitud++;
39  }
40
41  bool ListaEntrada::contiene(string idConcierto, string tipo){
42      bool tiene=false;
43        NodoListaEntrada *aux=this->primero;
44
45      while(aux){
46          if(aux->entrada.getIdConcierto()==idConcierto && aux->entrada.
                getTipo()==tipo)
47              tiene=true;
48          aux=aux->siguente;
49      }
50      return tiene;
51  }
52
53  void ListaEntrada::mostrar(){
54      NodoListaEntrada *aux=this->primero;
55
56      while(aux){
57          aux->entrada.mostrar();
58          aux=aux->siguente;
```

```
59      }
60  }
61
62  void ListaEntrada::eliminarEntrada(Entrada entrada){
63      bool encontrado=false;
64      NodoListaEntrada *aux=this->primero;
65      NodoListaEntrada *posAnt=NULL;
66
67      if((this->primero->entrada.getIdConcierto()==entrada.getIdConcierto())
            && (this->primero->entrada.getTipo()==entrada.getTipo())){
68          encontrado=true;
69      } //Miro si la primera entrada es la que estoy buscando
70      else{
71          while(aux->siguente && !encontrado){
72              if((aux->siguente->entrada.getIdConcierto()==entrada.
                    getIdConcierto())&&(aux->siguente->entrada.getTipo()==
                    entrada.getTipo())){
73                  encontrado=true;
74                  posAnt=aux;
75              }
76          aux=aux->siguente;
77          }
78      }//Si no, busco entre las n-1 entradas donde esta
79
80      if(!encontrado){
81          cout<<"No he podido borrar esa entarda de este cliente porque no
                la tiene. "<<endl;
82      }
83      else{ //Procedo a eliminar
84          NodoListaEntrada *aux2;
85          if(encontrado && !posAnt){
86              this->primero=this->primero->siguente;
87          }
88          else{
89              aux2=posAnt->siguente;
90              posAnt->siguente=aux2->siguente;
91              aux2->siguente=NULL;
92          }
93          longitud--;
94          delete(aux2);
95      }
96  }
97
98
99  int ListaEntrada::contarEntradas(string idConcierto, string tipo){
100     int total=0;
101
102     NodoListaEntrada *aux=this->primero;
103
104     while(aux){//Recorro la lista de entradas
105         //Si coincide el nombre del concierto y el tipo dados, sumo 1
106         if((aux->entrada.getIdConcierto()==idConcierto) && (aux->entrada.
                getTipo()==tipo))
107             total+=1;
```

```
108         aux=aux->siguente;
109     }
110 
111     return total;
112 }
```

Código 4.22: Implementación del árbol binario

```
1  #include <iostream>
2  #include <string>
3  
4  #include "ArbolBinario.h"
5  #include "NodoArbol.cpp"
6  #include "Cliente.h"
7  #include "NodoArbol.h"
8  
9  using namespace std;
10 
11 ArbolBinario::ArbolBinario()
12 {
13     this->raiz= new  NodoArbol("$$$$$$$$$");
14 
15 }
16 
17 ArbolBinario::~ArbolBinario(){}
18 
19 void ArbolBinario::insertarCliente(Cliente cliente, pnodoArbol nodo){
20     string codigoCliente = cliente.getId();
21     if(nodo->hijo_izquierdo==NULL && nodo->hijo_derecho==NULL){ //Si el
           nodo en el que queremos insertar no tiene hijos
22         if(nodo->idCliente == "$$$$$$$$$"){ //En caso de estar en la raiz
23             if(cliente.isRegistrado()){ //Si esta registrado se va a la
                   izquierda
24                 nodo->hijo_izquierdo = new NodoArbol(cliente.getId(),
                       cliente.getEntrada());
25             }else{ //Si no a la derecha
26                 nodo->hijo_derecho = new NodoArbol(cliente.getId(),cliente
                       .getEntrada());
27             }
28         }else{ //Si no estamos en la raiz
29             if(codigoCliente < nodo->idCliente){ //Si el codigo del
                   cliente que queremos insertar es "menor" se va a la
                   izquierda
30                 nodo->hijo_izquierdo = new NodoArbol(cliente.getId(),
                       cliente.getEntrada());
31             }else if(codigoCliente > nodo->idCliente){ //Si es codigo de
                   cliente es "mayor" se va a la derecha
32                 nodo->hijo_derecho = new NodoArbol(cliente.getId(),cliente
                       .getEntrada());
33             }else{ //Si ya esta en el arbol, se anade la entrada a su
                   lista
34                 nodo->listaEntradas.insertarEntrada(cliente.getEntrada());
35             }
36         }
```

```
37      }else{ //En caso de tener hijos
38          if(nodo->idCliente == "$$$$$$$$$"){ //Si estamos en la raiz
39              if(cliente.isRegistrado()){ //Y el cliente es registrado, se
                    almacena en la izquierda
40                      if(nodo->hijo_izquierdo==NULL)  //si no tenemos ningun
                            cliente registrado almacenado
41                          nodo->hijo_izquierdo = new NodoArbol(cliente.getId
                                (),cliente.getEntrada());
42                      else
43                          insertarCliente(cliente,nodo->hijo_izquierdo); //
                                Insertamos en la izquierda del siguiente
44              }else{ //Si no,lo inserto por la derecha
45                   if(nodo->hijo_derecho==NULL)//Si aun no habia insertado
                        ningun cliente por la derecha
46                          nodo->hijo_derecho = new NodoArbol(cliente.getId()
                                ,cliente.getEntrada());
47                      else
48                          insertarCliente(cliente,nodo->hijo_derecho);
49              }
50          }else{ //Si no estamos en la raiz
51              if(codigoCliente < nodo->idCliente){//Si el cliente a insertar
                     tiene un codigo menor que el del nodo en el que estamos,
                    inserto por la izquierda
52                   if(nodo->hijo_izquierdo==NULL)//Si no tiene insertado
                        ningun cliente por la izquierda
53                          nodo->hijo_izquierdo = new NodoArbol(cliente.getId
                                (),cliente.getEntrada());
54                      else
55                          insertarCliente(cliente,nodo->hijo_izquierdo);
56              }else if(codigoCliente>nodo->idCliente){//Codigo de cliente
                    mayor que el que estamos, inserto por la derecha
57                  if(nodo->hijo_derecho==NULL)//Si no tiene insertado ningun
                        cliente por la derecha
58                          nodo->hijo_derecho = new NodoArbol(cliente.getId()
                                ,cliente.getEntrada());
59                      else
60                          insertarCliente(cliente,nodo->hijo_derecho);
61              }else{//Si el cliente por casualidad entra y ya estaba
                    insertado
62                    nodo->listaEntradas.insertarEntrada(cliente.getEntrada())
                          ;
63              }
64          }
65      }
66
67  }
68
69  void ArbolBinario::insertarCliente(Cliente cliente){
70      this->insertarCliente(cliente,this->raiz);
71  }
72
73  void ArbolBinario::insertarClientes(Lista lista){
74      if(this->raiz && !lista.esvacia()){ //Bucle mientras la lista siga
            teniendo clientes
```

```
75          while(!lista.esvacia()){
76              this->insertarCliente(lista.inicial(),this->raiz); //Inserto
                    el primer cliente de la lista
77              lista.eliminar_inicial();
78          }
79          cout << "Se han insertado todos los clientes de la lista al arbol"
                << endl;
80      }
81  }
82
83  void ArbolBinario::mostrarInordenRegistrados(){
84      if(!esVacio())
85          if(this->raiz->hijo_izquierdo!=NULL)
86              mostrarInordenRegistrados(this->raiz->hijo_izquierdo);
87          else
88              cout << "El arbol no contiene clientes registrados"<<endl;
89      else
90          cout << "Arbol vacio"<<endl;
91  }
92
93  void ArbolBinario::mostrarInordenRegistrados(pnodoArbol nodo){
94      if(nodo == NULL){
95              cout << "";
96      }else{
97
98          if(!nodo->idCliente.find("CODREG")){ // si es registrado entra
99              mostrarInordenRegistrados(nodo->hijo_izquierdo);
100             if(nodo->idCliente!="$$$$$$$$$")
101                 nodo->mostrarDatos();
102             mostrarInordenRegistrados(nodo->hijo_derecho);
103         }
104     }
105 }
106
107 void ArbolBinario::mostrarInordenNoRegistrados(){
108    if(!esVacio())
109         if(this->raiz->hijo_derecho!=NULL)
110             mostrarInordenNoRegistrados(this->raiz->hijo_derecho);
111         else
112             cout <<"El arbol no contiene clientes no registrados"<< endl;
113     else
114         cout << "Arbol vacio"<<endl;
115 }
116
117 void ArbolBinario::mostrarInordenNoRegistrados(pnodoArbol nodo){
118     if(nodo == NULL)
119         cout << "";
120     else
121     {
122         if(nodo->idCliente.find("CODREG")){
123             mostrarInordenNoRegistrados(nodo->hijo_izquierdo);
124             nodo->mostrarDatos();
125             mostrarInordenNoRegistrados(nodo->hijo_derecho);
126         }
```

```
127     }
128 }
129
130 void ArbolBinario::mostrarPreorden(){
131     if(!esVacio())
132         mostrarPreorden(this->raiz);
133     else
134         cout << "Arbol vacio"<<endl;
135 }
136
137 void ArbolBinario::mostrarPreorden(pnodoArbol nodo){
138     if(nodo==NULL)
139         cout << "";
140     else{
141         if(nodo->idCliente!="$$$$$$$$$")
142             nodo->mostrarDatos();
143         mostrarPreorden(nodo->hijo_izquierdo);
144         mostrarPreorden(nodo->hijo_derecho);
145     }
146 }
147
148 void ArbolBinario::mostrarInfoEntradas(string idConcierto){
149     cout <<endl;
150     int palco=0,gallinero=0,patio=0;
151     //Miro cuantas entradas se han comprado de cada tipo del concierto
        dado tanto clientes registrados como no
152     palco = mostrarInfoEntradas(idConcierto,"PALCO",this->raiz->
        hijo_izquierdo) + mostrarInfoEntradas(idConcierto,"PALCO",this->
        raiz->hijo_derecho);
153     patio = mostrarInfoEntradas(idConcierto,"PATIO",this->raiz->
        hijo_izquierdo) + mostrarInfoEntradas(idConcierto,"PATIO",this->
        raiz->hijo_derecho);
154     gallinero = mostrarInfoEntradas(idConcierto,"GALLINERO",this->raiz->
        hijo_izquierdo) + mostrarInfoEntradas(idConcierto,"GALLINERO",this
        ->raiz->hijo_derecho);
155     //preparo para mostrar los resultados por pantalla
156     cout << idConcierto<<endl;
157     char caracter= 192;
158     cout <<"    "<<caracter<<"Palco: "<<palco<<endl;
159     cout <<"    "<<caracter<<"Patio: "<<patio<<endl;
160     cout <<"    "<<caracter<<"Gallinero: "<<gallinero<<endl;
161     cout <<"-------------------------------------"<<endl;
162     cout <<"     \tTotal: "<<(patio+palco+gallinero)<<endl;
163 }
164
165 int ArbolBinario::mostrarInfoEntradas(string idConcierto,string tipo,
    pnodoArbol nodo){
166     if(nodo!=NULL){
167             if(nodo->listaEntradas.contiene(idConcierto, tipo)){//Si el
                nodo tiene entradas compradas de ese tipo
168                 return nodo->listaEntradas.contarEntradas(idConcierto, tipo
                    ) + mostrarInfoEntradas(idConcierto,tipo,nodo->
                    hijo_izquierdo) +  mostrarInfoEntradas(idConcierto,tipo,
                    nodo->hijo_derecho);
```

```
169             }else{
170                 return  mostrarInfoEntradas(idConcierto,tipo,nodo->
                        hijo_izquierdo) +  mostrarInfoEntradas(idConcierto,tipo
                        ,nodo->hijo_derecho);
171             }
172     }
173     return 0;
174 }
175
176 void ArbolBinario::borrarCliente(string id){
177     if(id.length()==9){
178         borrarCliente(id,this->raiz,NULL);
179     }else{
180         cout <<"No existe un usuario con ese ID" << endl;
181     }
182 }
183
184 //Utilizare unicamente el puntero "padre" cuando encuentre el nodo que
        tengo que borrar
185 void ArbolBinario::borrarCliente(string id, pnodoArbol nodo, pnodoArbol
        padre){
186       if(nodo->idCliente == "$$$$$$$$$"){//Si estoy en el nodo raiz
187         if(!id.find("CODREG")){//Si cliente a borrar es registrado
188             if(nodo->hijo_izquierdo!=NULL)//miro si no es vacio el arbol
                    que contiene a los clietes registrados
189                 borrarCliente(id,nodo->hijo_izquierdo,nodo);
190             else
191                 cout <<"No existe un usuario con ese ID" << endl;
192         }else{//En caso de estar buscando un cliente no registrado
193             if(nodo->hijo_derecho!=NULL)
194                 borrarCliente(id,nodo->hijo_derecho,nodo);
195             else
196                 cout <<"No existe un usuario con ese ID" << endl;
197         }
198       }else{//Estamos en un nodo que no es la raiz
199         if(nodo->idCliente==id){//He encontrado el cliente que tengo que
                borrar
200             if(nodo->hijo_derecho==NULL && nodo->hijo_izquierdo==NULL){//
                    Miro si no tiene ningun hijo
201                 //Miro cual es el nodo que tengo que borrar desde el padre
                        y lo borro
202                 if(padre->hijo_derecho==nodo){
203                     cout<<"Se ha eliminado correctamente al usuario"<<endl
                            ;
204                     padre->hijo_derecho=NULL;
205                 }
206                 else{
207                     cout<<"Se ha eliminado correctamente al usuario"<<endl
                            ;
208                     padre->hijo_izquierdo=NULL;
209                 }
210
211             }else {//El cliente a borrar contiene algun hijo
212                 if(nodo->hijo_izquierdo==NULL){//Solo tiene Hijo Derecho
```

```
213                     //Miro cual es el nodo que tengo que borrar desde el
                            padre y lo borro
214                     if(padre->hijo_derecho==nodo){
215                         cout<<"Se ha eliminado correctamente al usuario"<<
                                endl;
216                         padre->hijo_derecho=nodo->hijo_derecho;
217                     }
218                     else{
219                         cout<<"Se ha eliminado correctamente al usuario"<<
                                endl;
220                         padre->hijo_izquierdo=nodo->hijo_derecho;
221                     }
222
223                 }else if(nodo->hijo_derecho==NULL){//Solo tiene Hijo
                        Izquierdo
224                      //Miro cual es el nodo que tengo que borrar desde el
                            padre y lo borro
225                     if(padre->hijo_derecho==nodo){
226                         cout<<"Se ha eliminado correctamente al usuario"<<
                                endl;
227                         padre->hijo_derecho=nodo->hijo_izquierdo;
228                     }
229                     else{
230                         cout<<"Se ha eliminado correctamente al usuario"<<
                                endl;
231                         padre->hijo_izquierdo=nodo->hijo_izquierdo;
232                     }
233
234                 }else{//Tiene los 2 hijos
235                     //Busco el cliente mayor id de la rama izquierda del
                            que quiero borrar
236                     string clienteMaximo = maximoCliente(nodo->
                            hijo_izquierdo);
237                     //borro el cliente que acabo de buscar del arobl
238                     borrarCliente(clienteMaximo, nodo->hijo_izquierdo,nodo
                            );
239                     //sustituyo por el que primeramente queria borrar
240                     nodo->idCliente = clienteMaximo;
241                 }
242             }
243         }else{//No he encontrado el cliente buscado, tengo que seguir
                buscando el cliente en los hijos
244               if(nodo->idCliente<= id){//El numero del cliente es menor y
                      he de buscar por la izquierda
245                  if(nodo->hijo_derecho!=NULL)
246                     borrarCliente(id,nodo->hijo_derecho,nodo);
247                  else
248                     cout <<"No existe un usuario con ese ID" << endl;
249               }else{//El numero del cliente es mayor y he de buscar por la
                      derecha
250                   if(nodo->hijo_izquierdo!=NULL)
251                     borrarCliente(id,nodo->hijo_izquierdo,nodo);
252                  else
253                     cout <<"No existe un usuario con ese ID" << endl;
```

```
254             }
255         }
256      }
257 }
258
259 bool ArbolBinario::esVacio(){
260     if(this->raiz->idCliente=="$$$$$$$$$" &&(this->raiz->hijo_derecho==
           NULL && this->raiz->hijo_izquierdo==NULL))
261         return true;
262     else
263         return false;
264 }
265
266 //Encontrar el cliente mayor de un arbol
267 string ArbolBinario::maximoCliente(pnodoArbol nodo){
268     string  maximo = nodo->idCliente;
269     //Si existe un hijo derecho, entonces ese sera mayor que el actual
270     if(nodo->hijo_derecho!=NULL){
271         maximo = maximoCliente(nodo->hijo_derecho);
272     }
273     return maximo;
274 }
275
276 void ArbolBinario::infoCliente(string id){
277     cout<<endl;
278     if(!id.find("CODREG")){
279         if(this->raiz->hijo_izquierdo==NULL)
280             cout << "No existe ese cliente" <<endl;
281         else
282             infoCliente(id, this->raiz->hijo_izquierdo);
283    }else{
284         if(this->raiz->hijo_derecho==NULL)
285             cout << "No existe ese cliente"<<endl;
286         else
287             infoCliente(id, this->raiz->hijo_derecho);
288    }
289 }
290
291 void ArbolBinario::infoCliente(string id, pnodoArbol nodo){
292    if(nodo->idCliente==id){
293         nodo->mostrarDatos();
294    }else{
295
296       if(nodo->idCliente<= id){
297                 if(nodo->hijo_derecho!=NULL)
298                     infoCliente(id,nodo->hijo_derecho);
299                 else
300                     cout <<"No existe un usuario con ese ID" << endl;
301             }else{
302                  if(nodo->hijo_izquierdo!=NULL)
303                     infoCliente(id,nodo->hijo_izquierdo);
304                 else
305                     cout <<"No existe un usuario con ese ID" << endl;
306             }
```

```
307     }
308 }
309
310 void ArbolBinario::borrar(){
311     this->raiz->hijo_derecho=NULL;
312     this->raiz->hijo_izquierdo=NULL;
313     cout <<"Se ha borrado el contenido de arbol"<< endl;
314 }
```

Código 4.23: Implementación del gestor

```
1  #include <ctime>
2  #include <iostream>
3  #include <string>
4  #include <windows.h>
5
6  #include "Gestor.h"
7  #include "Pila.h"
8  #include "Pila.cpp"
9  #include "Cola.h"
10 #include "Lista.h"
11 #include "ArbolBinario.h"
12 #include "ArbolBinario.cpp"
13
14 Gestor::Gestor()
15 {
16     this->pila=Pila();
17     this->cola_No_Registrados=Cola();
18     this->cola_Registrados=Cola();
19     this->lista=Lista();
20     this->arbol=ArbolBinario();
21 }
22
23 Gestor::~Gestor(){}
24
25 string Gestor::get_nuevoID(){
26     //Aumento el contador
27     contador++;
28     //Transformo el numero a string
29     string idUltimo=to_string(contador);
30     //Doy formato para que el numero contenga 3 caracteres
31     switch(idUltimo.length()){
32             case 1:
33                 idUltimo="00"+idUltimo;
34                 break;
35             case 2:
36                 idUltimo="0"+idUltimo;
37                 break;
38     }
39     return "CODREG"+idUltimo;
40 }
41
42 int Gestor::numero_Aleatorio(int DESDE, int HASTA){
43   return rand()%(HASTA-DESDE+1)+DESDE;
```

```
44
45 }
46
47 void Gestor::reiniciar(){
48     this->pila.borrar_Clientes();
49     this->cola_No_Registrados.borrar();
50     this->cola_Registrados.borrar();
51     this->lista.borrar_Lista();
52     this->arbol.borrar();
53 }
54
55 void Gestor::crearPrimerosClientes(Pila *pila){
56     int aleatorio=numero_Aleatorio(5,15);
57
58       for(int i=0; i<aleatorio; i++){ //Clientes registrados
59         if(i%2==0)
60             pila->apilar(Cliente(true,numero_Aleatorio(0,59),get_nuevoID()
                ));
61         else
62             pila->apilar(Cliente(false,numero_Aleatorio(0,59)));
63     }
64
65 }
66
67 void Gestor::crearSegundosClientes(Cola *cola_Regis, Cola *cola_No_Regis){
68     int aleatorio=numero_Aleatorio(5,15);
69
70
71     for(int i=0; i<aleatorio; i++){ //Clientes registrados
72         cola_Regis->encolar(Cliente(true,numero_Aleatorio(0,59),
               get_nuevoID()));
73     }
74
75
76     aleatorio=numero_Aleatorio(5,15);
77     for(int i=0; i<aleatorio; i++){ //Clientes NO registrados
78         cola_No_Regis->encolar(Cliente(false,numero_Aleatorio(0,59)));
79     }
80
81 }
82
83 char Gestor::menu(){
84     char tarea;
85
86     cout<<"\n\t\t-----------SISTEMA GESTOR DE VENTA DE ENTRADAS
           -----------\n"<<endl;
87     cout<<"a. Generar pila de clientes con reserva de entrada y dos colas
           de clientes registrados y no registrados que van a comprar"<<endl;
88     cout<<"b. Generar N clientes con reserva de entrada e incluirlos a la
           pila"<<endl;
89     cout<<"c. Generar N clientes registrados y M no registrados que
           acceden a la compra de entardas"<<endl;
90     cout<<"d. Incluir manualmente un cliente en la pila"<<endl;
91     cout<<"e. Incluir manualmente un cliente en la cola"<<endl;
```

```
92      cout<<"f. Mostrar clientes en la pila"<<endl;
93      cout<<"g. Borrar la pila de clientes"<<endl;
94      cout<<"h. Mostrar la cola de clientes registrados en espera de compra"
            <<endl;
95      cout<<"i. Mostrar la cola de clientes no registrados en espera de
            compra"<<endl;
96      cout<<"j. Borrar las dos colas"<<endl;
97      cout<<"k. Pasar a todos los clientes en espera a la fase de compra"<<
            endl;
98      cout<<"l. Mostrar los datos de todos los clientes de la lista"<<endl;
99      cout<<"m. Buscar en la lista y mostrar los datos del primer cliente
            registrado que ha comprado una entrada"<<endl;
100     cout<<"n. Buscar en la lista y mostrar los datos del primer cliente no
             registrado que ha comprado entrada"<<endl;
101     cout<<"o. Buscar en la lista y mostrar, dado el identificador de un
            concierto, los datos de todos los clientes que han comprado una
            entrada"<<endl;
102     cout<<"p. Borrar los datos de un cliente de la lista, dado su
            identificador (DNI o codigo de cliente)"<<endl;
103     cout<<"q. Borrar la lista de clientes"<<endl;
104     cout<<"r. Simular que se finaliza la compra de entradas, sacando los
            clientes de la lista e insertandolos en el arbol binario de
            busqueda en el orden indicado. El arbol puede estar o no vacio."<<
            endl;
105     cout<<"s. Insertar un cliente, manualmente, en el arbol."<<endl;
106     cout<<"t. Mostrar los datos de todos los clientes registrados,
            ordenados por codigo de cliente (inorden)"<<endl;
107     cout<<"u. Mostrar los datos de todos los clientes no registrados,
            ordenados por DNI (inorden)."<<endl;
108     cout<<"v. Borrar los datos de un cliente en el arbol, dado su
            identificador (DNI o codigo de cliente)."<<endl;
109     cout<<"w. Buscar un cliente, en el arbol, y mostrar los datos de las
            entradas que ha comprado."<<endl;
110     cout<<"x. Mostrar los datos de todos los clientes almacenados en el
            arbol (recorrido en preorden)."<<endl;
111     cout<<"y. Mostrar una estadistica del total de entradas vendidas para
            un concierto dado, desglosadas por tipo."<<endl;
112     cout<<"z. Reiniciar el programa."<<endl;
113
114
115
116     cout<<"$. Salir"<<endl;
117
118     cout<<"\nIntroduzca tarea: ";
119     cin>>tarea;
120
121     return tarea;
122 }
123
124 void Gestor::programa(){
125     ShowWindow(GetConsoleWindow(), SW_MAXIMIZE); //Pantalla completa
126
127     char opcion;
128     srand(time(NULL)); //Para generar numeros aleatorios
```

```
129
130     while(true){
131         opcion=menu();
132         switch(opcion){
133             case 'a':
134                 crearPrimerosClientes(&this->pila);
135                 crearSegundosClientes(&this->cola_Registrados, &this->
                        cola_No_Registrados);
136                 break;
137
138             case 'b':{
139                     int p;
140                     cout<<"Cantidad de clientes a generar: ";
141                     cin>>p;
142
143                     for(int i=0; i<p; i++){
144                         this->pila.apilar(Cliente(true, numero_Aleatorio
                                (0,59), get_nuevoID()));
145                     }
146                 }
147                 break;
148
149             case 'c':
150                 {
151                     int p;
152                     cout<<"Cantidad de clientes registrados a generar: ";
153                     cin>>p;
154
155                     for(int i=0; i<p; i++){
156                        this->cola_Registrados.encolar(Cliente(true,
                               numero_Aleatorio(0,59), get_nuevoID()));
157                     }
158
159                     cout<<"Cantidad de clientes no registrados a generar:
                            ";
160                     cin>>p;
161
162                     for(int i=0; i<p; i++){
163                        this->cola_No_Registrados.encolar(Cliente(false,
                               numero_Aleatorio(0,59)));
164                     }
165                 }
166                 break;
167
168             case 'd':
169                 {
170                     //bool registrado=true;//true -> "CODREG000"; false ->
                            "03488795T"
171                     bool registrado=false;
172                     string id="03488795T";
173                     int hora_llegada=numero_Aleatorio(0,59);
174                     //Entrada entrada=Entrada(Entrada().tipos[2],Entrada()
                            .Concierto[0]);
175                     Entrada entrada=Entrada(Entrada().tipos[(rand()
```

```
                        %(2-0+1)+0)], Entrada().Concierto[(rand()%(2-0+1)
                        +0)]);
176
177                     this->pila.apilar(Cliente(registrado, hora_llegada, id
                        , entrada));
178                 }
179                 break;
180
181             case 'e':
182                 {
183                     bool registrado=true;//true -> "CODREG000"; false ->
                        "03488795T"
184                     int hora_llegada=numero_Aleatorio(0,59);
185                     //Entrada entrada=Entrada(Entrada().tipos[2],Entrada()
                        .Concierto[0]);
186                     Entrada entrada=Entrada(Entrada().tipos[(rand()
                        %(2-0+1)+0)], Entrada().Concierto[(rand()%(2-0+1)
                        +0)]);
187
188                     if(registrado){
189                         this->cola_Registrados.encolar(Cliente(registrado,
                            hora_llegada, get_nuevoID(), entrada));
190                     }else{
191                         this->cola_No_Registrados.encolar(Cliente(
                            registrado, hora_llegada, entrada));
192                     }
193                 }
194                 break;
195
196             case 'f':
197                 this->pila.verEntera();
198
199                 break;
200
201             case 'g':
202                 this->pila.borrar_Clientes();
203
204                 break;
205
206             case 'h':
207                 this->cola_Registrados.mostrarCola();
208
209                 break;
210
211             case 'i':
212                 this->cola_No_Registrados.mostrarCola();
213
214                 break;
215
216             case 'j':
217                 this->cola_Registrados.borrar();
218                 this->cola_No_Registrados.borrar();
219
220                 break;
```

```
221
222             case 'k':
223                 {
224                     int total=this->cola_Registrados.get_longitud();
225                     for(int i=0;i<total;i++){
226                         this->lista.insertar(this->cola_Registrados.
                            desencolar());
227                     }
228
229
230                     total=this->cola_No_Registrados.get_longitud();
231                     for(int i=0;i<total;i++){
232                         this->lista.insertar(this->cola_No_Registrados.
                            desencolar());
233                     }
234
235                     total=this->pila.contar();
236                     for(int i=0;i<total;i++){
237                         this->lista.insertar(this->pila.primero());
238                         this->pila.desapilar();
239                     }
240                 }
241                 break;
242
243             case 'l':
244                 {
245                 this->lista.mostrar();
246                 }
247                 break;
248
249             case 'm':
250                 {
251                 this->lista.info_PrimeroRegistrado();
252                 }
253                 break;
254
255             case 'n':
256                 {
257                 this->lista.info_PrimeroNoRegistrado();
258                 }
259                 break;
260
261             case 'o':
262                 {
263                     string p;
264                     cout<<"Nombre del concierto (ESTOPA, MALU, MELENDI): "
                        ;
265                     cin>>p;
266                     this->lista.info_concierto(p);
267                 }
268                 break;
269
270             case 'p':
271                 {
```

```
272                     string p;
273                     cout<<"ID del cliente que quieres borrar: ";
274                     cin>>p;
275                     this->lista.borrar_cliente(p);
276                 }
277                 break;
278 
279             case 'q':
280                 {
281                 this->lista.borrar_Lista();
282                 }
283                 break;
284 
285             case 'r':
286                 {
287                     this->arbol.insertarClientes(lista);
288                 }
289                 break;
290 
291             case 's':
292                 {
293                     //bool registrado=true;//true -> "CODREG000"; false ->
                            "03488795T"
294                     bool registrado=false;//true -> "CODREG000"; false ->
                        "03488795T"
295                     string id= "03488795T";
296                     int hora_llegada=numero_Aleatorio(0,59);
297                    //Entrada entrada=Entrada(Entrada().tipos[2],Entrada().
                        Concierto[0]);
298                     Entrada entrada=Entrada(Entrada().tipos[(rand()
                        %(2-0+1)+0)], Entrada().Concierto[(rand()%(2-0+1)
                        +0)]);
299 
300                     if(registrado){
301                         this->arbol.insertarCliente(Cliente(registrado,
                            hora_llegada, get_nuevoID(), entrada));
302                     }else{
303                         this->arbol.insertarCliente(Cliente(registrado,
                            hora_llegada, id, entrada));
304                     }
305                 }
306                 break;
307 
308             case 't':
309                 {
310                     this->arbol.mostrarInordenRegistrados();
311                 }
312                 break;
313 
314             case 'u':
315                 {
316                     this->arbol.mostrarInordenNoRegistrados();
317                 }
318                 break;
```

```
319
320             case 'v':
321                 {
322                     string id;
323                     cout<<"Id del cliente que quieres borrar: ";
324                     cin>>id;
325                     this->arbol.borrarCliente(id);
326                 }
327                 break;
328
329             case 'w':
330                 {
331                     string id;
332                     cout<<"Id del cliente que quieres saber sus entradas:
                        ";
333                     cin>>id;
334                     this->arbol.infoCliente(id);
335                 }
336                 break;
337
338             case 'x':
339                 {
340                     this->arbol.mostrarPreorden();
341                 }
342                 break;
343
344             case 'y':
345                 {
346                     string id;
347                     cout<<"Id del concierto que quieres saber informacion:
                         ";
348                     cin>>id;
349                     this->arbol.mostrarInfoEntradas(id);
350                 }
351                 break;
352
353             case 'z':
354                 {
355                     this->reiniciar();
356                 }
357                 break;
358
359
360             case '$':
361                 exit(0);
362                 break;
363
364             default:
365                 cout<<"Su tarea no puede realizarse. "<<endl;
366                 break;
367         }
368     }
369 }
```

Bibliografía

[1] Foro de dudas StackOverFlow. `https://es.stackoverflow.com/`.

[2] Mª José Domínguez. Introducción a los TAD's, 2021. Universidad de Alcalá.

[3] Mª José Domínguez. Listas, 2021. Universidad de Alcalá.

[4] Mª José Domínguez. Pilas y colas en C++, 2021. Universidad de Alcalá.

[5] Mª José Domínguez. Árboles binarios, 2021. Universidad de Alcalá.