|  |  |  |
| --- | --- | --- |
|  |  | PEC1  Estructuras de datos. |

# INDICE

* [Datos personales.](#_Datos_personales)
* TAD’s creados:
  + Detalles y justificación de la implementación.
  + Definición de las operaciones del TAD
  + Soluciones adoptadas a las dificultades encontradas
    - [Cola.](#_Cola)
    - [NodoCola.](#_NodoCola)
    - [Lista.](#_Lista)
    - [NodoLista.](#_NodoLista)
    - [Cliente.](#_Cliente)
    - [Gestor.](#_Gestor)
* [Comportamiento del programa.](#_Comportamiento_DEl_Programa)
* [Diagrama UML.](#_Diagrama_UML_2)
* Código fuente.

* + [NodoLista.hpp](#_NodoLista.hpp)
  + [NodoLista.cpp](#_NodoLista.cpp)
  + [Lista.hpp](#_Lista.hpp)
  + [Lista.cpp](#_Lista.cpp)
  + [NodoCola.hpp](#_NodoCola.hpp)
  + [NodoCola.cpp](#_NodoCola.cpp)
  + [Cola.hpp](#_Cola.hpp)
  + [Cola.cpp](#_Cola.cpp)
  + [Cliente.hpp](#_Cliente.hpp)
  + [Cliente.cpp](#_Cliente.cpp)
  + [Gestor.hpp](#_Gestor.hpp)
  + [Gestor.cpp](#_Gestor.cpp)
  + [main.cpp](#_main.cpp)

# Datos personales

## Nombre:

* Juan

## Apellidos:

* Casado Ballesteros

## DNI:

* 09108762A

## Correo:

* jcb7777777@gmail.com

## Asignatura:

* Estructuras de datos GII martes 10:00-12:00
* Laboratorio con Hamid Tayebi de 12:00 a 14:00

# Cola

## Detalles y justificación de la implementación.

Utilizo el TAD cola para implementar los servidores y como estructura para almacenar a los Clientes antes de haberlos introducido en un servidor. Cada cliente está almacenado en un NodoCola que son los elementos de los que la Cola está compuesta.

Cola se comporta como una estructura FIFO en la que el primer elemento que haya entrado será el primero en salir, para hacer esto me apoyo en dos punteros, uno que marca el inicio de la cola llamado primeros que es por donde saco los Clientes y otro llamado último que es por donde introduzco Clientes nuevos.

## Constructor:

En el constructor iniciaré ambos punteros que dan soporte a la estructura de la Cola, primero y último a NULL para indicar que la cola está vacía, también inicio cantidad\_de\_clientes a 0 por el mismo motivo.

## Métodos públicos:

**void insertar (Cliente c);**

* Introduce un Cliente nuevo en la Cola, para ello hará apuntar el nuevo cliente al que está apuntado por el puntero que marca el último y actualiza el valor de ese puntero para que apunte al nuevo introducido que ahora ocupará la última posición.
* Esta función al ser llamada incrementa en uno clientes\_en\_cola de modo que se lleva un recuento activo de ellos.

**Cliente eliminar ();**

* Elimina y retorna el Cliente que entró primero de todos en la Cola, para ello accede al puntero primero lo hace apuntar al siguiente elemento y elimina el que ates era el primero.
* Esta función al ser llamada decrementa en uno clientes\_en\_cola de modo que se lleva un recuento activo de ellos.

**void mostrar ();**

* Lee la Cola de forma iterativa accediendo en cada paso al NodoCola apuntado por el NodoCola anterior por el puntero siguiente ejecutando la función show\_cliente() de cada cliente en cada NodoCola.

**bool es\_vacia ();**

* Retorna primero como un booleano, primero valdrá NULL==0==false cuando la cola esté vacía, en cualquier otro caso retornará true.
* NOTA: es\_vacia() devuelve FALSE cuando es vacía y TRUE cuando está llena, aunque por el nombre de la función parece que debería de devolver los resultados al revés al hacerlo de este modo me evito negar dos veces, una en el return de la función y otra al utilizarla.

**int cantidad\_de\_clientes();**

* Retorna la variable privada clientes\_en\_cola que ya hemos explicado como actualiza su valor que se corresponderá en todo momento con los clientes que haya en la cola.

**Cliente ver\_primero (void);**

* Me devuelve sin borrarlo de la cola al primer cliente que entró en ella y que todavía no ha salido.

## Atributos privados:

## pnodo primero,ultimo;

* Mediante estos punteros a los NodoCola en los que se almacena el primer y el último Cliente gestionamos la cola permitiendo introducir datos en ella o sacándolos.

## Int clientes\_en\_cola;

* En esta variable almacenan dinámicamente según se añaden o se eliminan la cantidad de clientes que hay en la cola en todo momento.

## Soluciones a las dificultades encontradas.

Para poder saber en todo momento y de forma eficiente la cantidad de clientes en cada cola actualizo contantemente la variable clientes\_en\_cola al añadir o eliminar NodoClola a la Cola. Así evito tener que recorrer la lista contándolos para saber cuántos hay.

# NodoCola

## Detalles y justificación de la implementación.

Utilizo este TAD para almacenar los Clientes en una cola. El TAD está compuesto por un Cliente que es el dato que queremos almacenar en la Cola y un puntero que da sustento a la estructura pues apuntará al siguiente NodoCola de la Cola.

A los punteros que apuntan a un NodoCola le damos el nombre de pnodoCola de forma que hacemos el código más legible.

## Constructor:

Iniciamos cada NodoCola con un cliente y con un puntero de tipo NodoCola apuntando a NULL. Esto lo hacemos para poder hacer apuntar ese puntero al NodoCola que corresponda cuando generemos la estructura de cola. El puntero llamado siguiente es un atributo privado, pero el nodo hacemos que sea friend class con Cola para poder desde allí editar su valor.

## Atributos privados:

## Cliente cliente;

* En esta variable almacenaré el valor del cliente que NodoCola debe contener.

## NodoCola \*siguiente;

* Haré apuntar este puntero al NodoCola que corresponda para formar la estructura de Cola.

## Soluciones a las dificultades encontradas.

Con la intención de simplificar el código y hacerlo más visible realizo un typedef a los punteros de tipo NodoCola como pnodoCola, además hago de siguiente un atributo privado, pero permito que Cola lo edite ya que hago un friend class con ella. Por comodidad iniciaré todos los punteros a siguiente NodoCola como NULL para poder darles valores con posterioridad desde la clase Cola.

# Lista

## Detalles y justificación de la implementación.

Utilizo este TAD para gestionar los pnodoLista en los que almacenaré Cliente una vez hayan comprado su entrada. La lista es doblemente enlazada para facilitar la ordenación de los Clientes en función de su código de entrada.

La lista estará gestionada por dos punteros, primero y puntero, a NodosLista, uno que apunte al primer elemento de la lista ordenada y otro que utilizo para poder recorrerla a la hora de ver dónde colocar cada elemento para que esté siempre ordenada.

## Constructor:

Inicio ambos punteros, primero y puntero a NULL para indicar que la cola está vacía, también inicio clientes\_en\_lista a 0 por el mismo motivo.

## Métodos públicos:

## void insertar (Cliente c);

* Introduce un Cliente nuevo en la Lola, para ello puntero irá iterando sobre la lista hasta que o bien el NodoLista siguiente al que está apuntando contenga un Cliente con un identificador de entrada menor que el del Cliente que queremos introducir o que llegue al final de la lista.

Si ha llegado al final de la lista puede ser por dos causas, la lista está vacía por lo que el NodoLista que contiene al Cliente nuevo será el primer elemento de la Lista o porque hemos recorrido toda la lista sin que haya otro Cliente anteriormente introducido en ella con un identificador de entrada de mayor valor en cuyo caso introduciremos nuestro Cliente detrás del NodoLista al que está apuntando puntero. Para realizar esto utilizaremos el puntero a NodoLista que apunta al elemento anterior contenido en cada NodoLista, recordamos que Lista es una esrructura tipo lista doblemente enlazada.

Si se para por haber llegado a un NodoLista cuyo NodoLista siguente contiene un Cliente con un identificador de enetrada de menor valor comprobamos que el NodoLista al que apunta puntero contenga un Cliente con identificador de entrada mayor, de serlo insertamos detrás de él y de no serlo insertaremos el nuevo NodoLista que contiene al nuevo Cliente delante.

* Esta función al ser llamada incrementa en uno clientes\_en\_lista de modo que se lleva un recuento activo de ellos.

## Cliente eliminar ();

* Elimina y retorna el Cliente que entró primero de todos en la Lista, para ello accede al puntero primero lo hace apuntar al siguiente elemento y elimina el que ates era el primero.
* Esta función al ser llamada decrementa en uno clientes\_en\_lista de modo que se lleva un recuento activo de ellos.
* Esta función se utiliza exclusivamente para resetear el programa.

## void mostrar ();

* Lee la Lista de forma iterativa accediendo en cada paso al NodoLista apuntado por el NodoLista anterior por el puntero siguiente ejecutando la función show\_cliente\_completo() de cada cliente en cada NodoCola, mostrando sus datos junto a su identificador de entrada.

## bool es\_vacia ();

* Retorna primero como un booleano, primero valdrá NULL==0==false cuando la cola esté vacía, en cualquier otro caso retornará true.
* NOTA: es\_vacia() devuelve FALSE cuando es vacía y TRUE cuando está llena, aunque por el nombre de la función parece que debería de devolver los resultados al revés al hacerlo de este modo me evito negar dos veces, una en el return de la función y otra al utilizarla.
* Esta función se utiliza exclusivamente para alertar de que la lista no se puede mostrar pues está vacía.

## int cantidad\_de\_clientes();

* En esta variable almacenan dinámicamente según se añaden o se eliminan la cantidad de clientes que hay en la cola en todo momento.
* Esta función se utiliza exclusivamente para mostrar en el menú los clientes que ya hay comprado su entrada y de forma activa al mostrar la lista de entradas compradas durante los procesos de compra.

## Métodos privados:

## void insertar\_delante(pnodoLista n);

* Mediante un puntero a NodoLista auxiliar hago que el NodoLista apuntado por el puntero pasado a la función quede entre el NodoLista al que apunta puntero y el siguiente NodoLista.
* El puntero a siguiente del NodoLista apuntado por puntero apuntará al nuevo NodoLista, el anterior del siguente al nuevo y los punteros siguiente y anterior del nuevo a su siguiente y anterior en la lista de forma que quede incluído en la estructura.

## void insertar\_detras(pnodoLista n);

* Mediante un puntero a NodoLista auxiliar hago que el NodoLista apuntado por el puntero pasado a la función quede entre el NodoLista al que apunta puntero y el anterior NodoLista.
* El puntero a anterior del NodoLista apuntado por puntero apuntará al nuevo NodoLista, el siguiente del anterior al nuevo y los punteros siguiente y anterior del nuevo a su siguiente y anterior en la lista de forma que quede incluído en la estructura.

## void insertar\_primero(pnodoLista n);

* Inserta un NodoLista en la lista haciendo apuntar sus punteros a siguiente y anterior a NULL y apuntando primero y puntero al nuevo NodoLista que será el primero de la misma.

## Atributos privados:

## pnodoLista puntero, primero;

* Mediante estos punteros a los NodoCola en los que se almacena el primer Cliente ordenado y el cliente el el que nos estemos fijando a la hora de ordenar gestionamos la cola permitiendo introducir datos en ella o sacándolos.

## int clientes\_en\_lista;

* En esta variable almacenan dinámicamente según se añaden o se eliminan la cantidad de clientes que hay en la cola en todo momento.

## Soluciones a las dificultades encontradas.

La variable clientes\_en\_lista es tratada del mismo modo que en la Cola para ir almacenando en todo momento el número de clientes que haya en la Lista.

Lista está organizada como una estructura de lista doblemente enlazada, esto facilita en clan medida la inserción de datos de forma ordenada en ella.

# NodoLista

## Detalles y justificación de la implementación.

Utilizo este TAD para almacenar los Clientes en una cola. El TAD está compuesto por un Cliente que es el dato que queremos almacenar en la Lista, un puntero que apuntará al siguiente NodoLista de la Lista y otro que apunta al NodoLista anterior, dando a la estructura una forma de lista doblemente enlazada .

A los punteros que apuntan a un NodoLista le damos el nombre de pnodoLista de forma que hacemos el código más legible.

## Constructor:

Iniciamos cada NodoLista con un cliente y dos punteros de tipo NodoLista apuntando a NULL. Esto lo hacemos para poder hacer apuntar ese puntero al NodoLista que corresponda cuando generemos la estructura de lista doblemente enlazada. El puntero llamado siguiente y el llamado puntero son atributos privados, pero NodoLista hacemos que sea friend class con Lista para poder desde allí editar su valor.

## Atributos privados:

## Cliente cliente;

* En esta variable almacenaré el valor del cliente que NodoLista debe contener.

## NodoLista \*siguiente;

* Haré apuntar este puntero al NodoLista que corresponda para formar la estructura de Lista.

## NodoLista \*siguiente;

* Haré apuntar este puntero al NodoLista que corresponda para formar la estructura de Lista siendo esta una lista doblemente enlazada.

## Soluciones a las dificultades encontradas.

Con la intención de simplificar el código y hacerlo más visible realizo un typedef a los punteros de tipo NodoLista como pnodoLista, además hago de siguiente un atributo privado, pero permito que Lista lo edite ya que hago un friend class con ella. Por comodidad iniciaré todos los punteros a siguiente NodoLista como NULL para poder darles valores con posterioridad desde la clase Lista.

El NodoLista contendrá dos punteros para hacer que la lista sea doblemente enlazada.

# Cliente

## Detalles y justificación de la implementación.

Cliente será un TAD que se corresponderá con una persona que quiere comprar una entrada por internet, esta persona tendrá un nombre, un DNI y será VIP o no en la página de compra de entradas. De ser VIVP tendrá ventajas pues podrá comenzar a comprar su entrada dos horas antes que los clientes normales. Cada persona comenzará a comprar su entrada en una hora determinada y tardará una cantidad de tiempo en hacerlo, en este caso estos datos se corresponderán con valores aleatorios que cumplan los requisitos dados por los profesores, pero podrían ser datos reales.

Los clientes dependiendo de si son VIP o no recorrerán ciertos servidores de venta de entradas u otros hasta acabar todos en una lista ordenados por el número identificador de su entrada, el cual se generará una vez hayan salido de los servidores, es decir una vez hayan finalizado el proceso de compra de su entrada.

## Constructor:

El TAD cliente contará con dos constructores distintos, uno en el que no se iniciarán ninguno de sus atributos y otro en el que se iniciarán la hora\_de\_inicio\_de\_compra y si el Cliente es registrado o no como parámetros y el DNI y el tiempo de compra como valores aleatorios.

En caso de que el Cliente sea no registrado su hora de inicio de compra se incrementará en dos horas ayudándonos de una función auxiliar que hará esto.

Existen dos constructores por comodidad a la hora de declarar Clientes.

## Métodos públicos:

## void show\_cliente();

* Llama a otros métodos privados que explicaremos próximamente de modo tal que muestra por pantalla todos los datos del cliente excepto su identificador de entrada aplicándoles un formato.

## void show\_cliente\_completo();

* Llama a los mismos métodos que show\_cliente() y les aplica el mismo formato a la hora de mostrar los atributos del Cliente y además muestra con formato el identificador de entrada.

## void generar\_identificador\_de\_entrada();

* Dota al cliente de identificador de entrada, 4 letras aleatorias, este método es utilizado cuando los clientes salen de los servidores.

## long get\_identificador\_de\_entrada();

* Transforman el identificador de entrada en un long y lo retornan, este método es empleado para poder ordenar los Clientes en la lista por su identificador.

## int get\_hora\_de\_inicio\_de\_compra();

* Devuelve la hora de inicio de compra del cliente de modo que se pueda saber cuándo este debe de entrar en un servidor para comprar su entrada.

## int get\_tiempo\_de\_compra();

* Devuelve el tiempo que elCliente tardad en comprar su entrada, de este modo podremos saber cuándo sacarlo del servidor.

## Atributos privados:

## char DNI [9];

* Aquí se almacenan las ocho letras y el número de DNI que son generados de forma aleatoria. El número es generado a partir de las letras como en los DNI reales.

## int hora\_de\_inicio\_de\_compra;

* Aquí se almacena la hora a la que el Cliente debe entrar en los servidores a comprar su entrada.

## int tiempo\_de\_compra;

* Aquí se almacena el tiempo que el Cliente debe permanecer dentro de un servidor, es lo que tarda en comprar su entrada.

## bool registrado;

* Dice si el cliente está registrado o no, a la hora de la implementación estar registrado supone tener un tiempo de inicio de compra dos horas menor y entrar en un servidor formado por una cola única en vez de en uno formado por cuatro colas.

## char identificador\_de\_entrada [4];

* Cuatro letras aleatorias que representan la identificación de la entrada.

## long numero\_de\_entrada;

* En esta variable almaceno el identificador de entrada transformado en un número de modo tal que pueda comprarlos con otros identificadores.

## Métodos Privados:

## long transformar\_identificador\_de\_entrada();

* Función que transforma el identificador de entrada de cuatro letras a un número, multiplica el valor ASCII de cada letra por 100 y lo suma en un acumulador, el resultado se guarda en numero\_de\_entrada.
* Esta función es llamada solo por generar\_identificador\_de\_entrada();

## void show\_identificador\_de\_entrada();

* Muestra las 4 letras del identificador de entrada en pantalla sin formato.
* Esta función es llamada solo por las que muestran el cliente con formato.

## void generar\_DNI();

* Genera 8 letras aleatorias y las guarda en DNI, con ellas genera un número tal y como se hace con los DNI reales y guarda todo en DNI.
* Esta función es llamada solo por el constructor.

## void show\_DNI();

* Muestra el DNI sin formato.
* Esta función es llamada solo por las que muestran el cliente con formato.

## void generar\_hora\_de\_inicio\_de\_compra();

* Suma dos horas a la hora de inicio de compra en el caso de que el Cliente sea no registrado.
* Esta función es llamada solo por el constructor.

## void show\_hora\_de\_inicio\_de\_compra();

* Muestra la hora de inicio de compra con formato parcial, en vez de mostrar el número en el que se guardan los minutos del inicio de la compra la muestra como hh:mm.
* Esta función es llamada solo por las que muestran el cliente con formato.

## void generar\_tiempo\_de\_compra();

* Da un número aleatorio de 1 a 10 al tiempo de compra.
* Esta función es llamada solo por el constructor.

## void show\_tiempo\_de\_compra();

* Muestra sin formato el tiempo que el cliente deberá de pasar en el servidor.
* Esta función es llamada solo por las que muestran el cliente con formato.

## Soluciones a las dificultades encontradas.

Por comodidad todos los procesos que se realizan dentro del Cliente están subdivididos en funciones que se llaman unas a otras, de esta forma resulta más fácil ir probando cada apartado de código por separado, también resulta más fácil realizar modificaciones sobre él.

Ha curiosidades en el diseño como que la letra de DNI se genera de una forma realista o que el identificador de entrada lo hago equivaler a un número para poder compararlo.

Evito crear dos clases distintas de Cliente (VIP y NO\_REGISTRADO) mediante el uso de un booleano que los diferencia, así no es necesario realizar herencia. Esta variable se emplea para saber si en cliente debe de ir a servidor VIP o a los cuatro servidores habilitados para los no registrados.

La hora a la que los clientes no registrados comienzan a comprar entradas y el rango de duración de la compra de la misma está definido como:

#define HORA\_DE\_INICIO\_NO\_REGISTRADOS 120

#define TIEMPO\_MAXIMO\_DE\_COMPRA 10

de modo que resultaría fácil modificarlo, pero sin que estén dentro de ninguna variable.

# Gestor

## Detalles y justificación de la implementación.

Utilizo el TAD Gestor como interface para controlar los servidores mediante la ejecución de funciones desde el main. En gestor se localiza una Cola que actúa como servidor para los Clientes VIP y otra Cola que los almacena antes de entrar al servidor. También se encuentra un array con otras cuatro Colas para representar a los cuatro servidores de los clientes NO REGISTRADOS, con otra Cola más para almacenarlos antes de hacerlos pasar por los servidores. Por último, hay una Lista que como ya hemos visto es doblemente enlazada y se encuentra siempre ordenada por los identificadores de entrada de los Clientes que contiene.

Gestor permite generar una cantidad determinada (está declarada en un define) de Clientes VIP y otra de Clientes NO REGISTRADOS, se pueden ver o borrar ambos tipos. También se puede hacer una simulación de su paso a través de los servidores. Se iniciará un reloj y los Clientes entrarán en ellos cuando llegue su hora y saldrán cuando hayan estado su tiempo de compra siendo los primeros de la cola. En el caso de los Clientes NO REGISTRADOS hay un algoritmo que determina en qué Cola colocarse, la primera menos ocupada siguiendo el orden del array. Cuando los Clientes salen de las Colas se van a la Lista la cual se puede visualizar.

## Constructor.

No se inicia ningún valor, el llenado de las Colas y la generación o borrado de Clientes se realiza por parte del usuario desde el main.

## Métodos públicos:

## void generar\_solicitudes\_de\_clientes\_VIP(void);

* Genera clientes tipo VIP asignándoles a cada uno un tiempo de inicio de compra que será igual al inicio de compra del cliente anterior más un aleatorio entre cero y cinco, guarda los clientes en una cola ordenados por su hora de inicio. SI la Cola de Clientes VIP en espera está llena y se vuelve a solicitar su llenado esta se reescribe.

## void mostrar\_clientes\_VIP(void);

* Muestra la cola de los Clientes VIP que todavía no han entrado al servidor.

## void borrar\_clientes\_VIP(bool texto);

* Elimina la Cola de los Clientes VIP que todavía no han entrado al servidor.

## void generar\_solicitudes\_de\_clientes\_no\_registrados(void);

* Genera un número definido de Clientes NO REGISTRADOS y los almacena en una Cola, si la Cola está llena la reescribe. El proceso de llenado de cola es igual que para los VIP, el ajuste de la hora de inicio de compra se hace dentro del TAD Cliente.

## void mostrar\_clientes\_no\_registrados(void);

* Muestra la cola de los Clientes NO REGISTRADOS que todavía no han entrado al servidor.

## void borrar\_clientes\_no\_registrados(bool texto);

* Elimina la Cola de los Clientes NO REGISTRADOS que todavía no han entrado al servidor.

## void simular\_venta\_a\_clientes\_VIP(void);

* Hace pasar a los Clientes VIP por su servidor dedicado, cuando han estado tanto tiempo como indica su tiempo de compra siendo los primeros de la Cola les saca de servidor, asigna un identificador de entrada e introduce en la Lista ordenada.

## void simular\_venta\_a\_clientes\_no\_registrados(void);

* Selecciona el servidor correcto de los cuatro disponibles para los Clientes NO REGISTRADOS según los criterios indicados y cuando han estado tanto tiempo como indica su tiempo de compra siendo los primeros de la Cola les saca de servidor, asigna un identificador de entrada e introduce en la Lista ordenada.

## void mostrar\_entradas\_vendidas(void);

* Muestra la Lista ordenada de los Clientes con su identificador de entrada.

## void reiniciar(void);

* Borra todas las Colas de los Clientes en espera y la Lista de clientes con entradas compradas si es que no estban vacías ya.

## int cantidad\_no\_registrados();

* Devuelve el número de Clientes NO REGISTRADOS en espera.

## int recuento\_VIP();

* Devuelve el número de Clientes VIP en espera.

## int recuento\_entradas();

* Devuelve el número de Clientes que ya han comprado su entrada.

## Atributos privados:

## Cola VIP\_en\_espera;

* En esta cola se almacenan los Clientes VIP cuando se generan a la espera de entrar a su servidor, está ordenada por hora de inicio de compra.

## Cola servidor\_VIP;

* Esta Cola representa al servidor en el que los Clientes VIP pueden comprar las entradas.

## Cola no\_registrado\_en\_espera;

* En esta cola se almacenan los Clientes NO REGISTRADOS cuando se generan a la espera de entrar a su servidor, está ordenada por hora de inicio de compra.

## Cola servidor\_no\_registrado[SERVIDORES\_NO\_REGISTRADOS];

* Con estas 4 Colas se representan los servidores en los que los Clientes NO REGISTRADOS pueden comprar sus entradas.

## Lista clientes\_terminados;

* En esta Lista se almacenen los Clientes sin importar su tipo que ya hayan comprado su entrada, está ordenada por el identificador de entrada.

## Métodos privados:

## void show\_hora(int t);

* Se utiliza esta función para dar formato al reloj que controla cuando los CLientes entran o salen del servidor antes de mostrarloe en pantalla. El formato dado será hh:mm que coincide con el formato en el que se muestr el tiempo de inicio de compra de los Clientes.

## int recuento\_no\_registrados();

* Cuenta Cuantos Clientes NO REGISTRADOS hay en total entre los cuatro servidores en los que pueden comprar las entradas. Esta función se utiliza para saber cuándo todos los Clientes han terminado de comprar su entrada y poder dar por terminada la venta de las mismas.

## void actualizar\_pantalla\_VIP(int tiempo, int tiempo\_en\_proceso);

* Muestra el estado de la Cola de Clientes VIP en espera, del servidor dedicado y de la Lista en un tiempo concreto. Esta función existe por hacer más legible el código y separar los cálculos de si los Clientes deben entrar o salir de los servidores de la fase de mostrar el estado del proceso.

## void actualizar\_pantalla\_no\_registrado(int tiempo, int \* tiempo\_en\_proceso);

* Muestra el estado de la Cola de Clientes NO REGISTRADOS en espera, de sus servidores y de la Lista en un tiempo concreto. Esta función existe por hacer más legible el código y separar los cálculos de si los Clientes deben entrar o salir de los servidores de la fase de mostrar el estado del proceso.

## Soluciones a las dificultades encontradas.

Para saber cuándo un Cliente lleva tanto tiempo como marca su tiempo de compra siendo el primero de la Cola que implementa el servidor en el que está utilizo una variable auxiliar que llamo tiempo en proceso. Esta variable la incremento en uno si hay un Cliente en la Cola y la pongo a cero cuando ha alcanzado a tiempo de compra y por tanto cuando he sacado al Cliente del servidor.

En el caso de los Clientes NO REGISTRADOS en vez de utilizar solo una variable auxiliar utilizo un array de tantas variables como servidores haya para realizar el mismo proceso.

El número de Clientes de cada tipo, la velocidad a la que pasa el tiempo, el número de servidores para Clientes NO REGISTRADOS o la distancia entre el tiempo de compra de un Cliente y otro están determinadas por constantes en defines, por lo que son fácilmente modificables antes de ejecutar el programa.

Las funciones de actualizar pantalla, aunque sean una solución un poco fea ya que les paso los tiempos por parámetro son útiles porque simplifican el código, lo hacen más legible y fácil de modificar y mantener. Hacer de los temporizadores Atributos de la Clase no me pareció una solución adecuada ya que son específicos de las funciones de simulación y no definen al Gestor.

# Comportamiento DEl Programa

El usuario se encuentra con un conjunto de acciones que puede realizar. Cada acción corresponde a una letra que introducirá por pantalla. De este modo el usuario generará Clientes VIP o NO REGISTRADOS pudiendo borrarlos, volverlos a generar o pudiendo visualizarlos. Simulará el proceso de compra de los Clientes haciendo pasar a los VIP por una Cola de la que tendrán que ser primeros tanto tiempo como indique su tiempo de comprar para poder salir. En el caso de los NO REGISTRADOS habrá cuatro colas y se colocarán en la primera más vacía. Una vez finalizado el proceso de compra de cada Cliente a este se le asignará un identificador de entrada y será introducido en una Lista ordenada por él. Finalmente, el usuario podrá decidir si ver la Lista, si reiniciar el programa o si generar más Clientes para repetir el proceso.

El programa está a prueba de fallos de ejecución, no hay posibles acciones de usuario que lleven a sacar un Cliente o a mostrarlo cuando la Lista está vacía o acciones que pueda realizar que sean incompatibles con el programa.

Los procesos más detallados sobre la implementación de cada acción están reflejados en el TAD correspondiente en su método correspondiente.

# Diagrama UML

|  |
| --- |
| Cliente |
| Publico: void show\_cliente();  void show\_cliente\_completo();  void generar\_identificador\_de\_entrada();  long get\_identificador\_de\_entrada();  int get\_hora\_de\_inicio\_de\_compra();  int get\_tiempo\_de\_compra(); |
| Privado: char DNI [9];  int hora\_de\_inicio\_de\_compra;  int tiempo\_de\_compra;  bool registrado;  char identificador\_de\_entrada [4];  long numero\_de\_entrada;  void show\_identificador\_de\_entrada();  long transformar\_identificador\_de\_entrada();  void generar\_DNI();  void show\_DNI();  void generar\_hora\_de\_inicio\_de\_compra();  void show\_hora\_de\_inicio\_de\_compra();  void generar\_tiempo\_de\_compra();  void show\_tiempo\_de\_compra(); |

|  |
| --- |
| NodoLista friend class Lista; |
| Privado: Cliente cliente;  NodoLista \*siguiente;  NodoLista \*anterior; |

|  |
| --- |
| Lista |
| Publico: void insertar (Cliente c);  Cliente eliminar ();  void mostrar ();  bool es\_vacia ();  int cantidad\_de\_clientes(); |
| Privado: nodoLista puntero, primero;  int clientes\_en\_lista;  void insertar\_delante(pnodoLista n);  void insertar\_detras(pnodoLista n);  void insertar\_primero(pnodoLista n); |

|  |
| --- |
| NodoCola friend class Cola; |
| Privado: Cliente cliente;  NodoCola \*siguiente; |

|  |
| --- |
| Main.cpp |

|  |
| --- |
| Gestor |
| Publico: void generar\_solicitudes\_de\_clientes\_VIP(void);  void mostrar\_clientes\_VIP(void);  void borrar\_clientes\_VIP(bool texto);  void generar\_solicitudes\_de\_clientes\_no\_registrados(void);  void mostrar\_clientes\_no\_registrados(void);  void borrar\_clientes\_no\_registrados(bool texto);  void simular\_venta\_a\_clientes\_VIP(void);  void simular\_venta\_a\_clientes\_no\_registrados(void);  void mostrar\_entradas\_vendidas(void);  void generar\_arbol(void);  void buscar\_cliente(void);  void mostrar\_arbol\_en\_pre\_orden(void);  void mostrar\_arbol\_en\_post\_orden(void);  void mostrar\_arbol\_en\_in\_orden(void);  void dibujar\_arbol(void);  void reiniciar(void);  int cantidad\_no\_registrados();  int recuento\_VIP();  int recuento\_entradas(); |
| Privado: Cola VIP\_en\_espera;  Cola servidor\_VIP;  Cola no\_registrado\_en\_espera;  Cola servidor\_no\_registrado[SERVIDORES\_NO\_REGISTRADOS];  void show\_hora(int t);  int recuento\_no\_registrados();  void actualizar\_pantalla\_VIP(int tiempo, int tiempo\_en\_proceso);  void actualizar\_pantalla\_no\_registrado(int tiempo, int \* tiempo\_en\_proceso); |

|  |
| --- |
| Cola |
| Publico: void insertar (Cliente c);  Cliente eliminar ();  void mostrar ();  bool es\_vacia ();  int cantidad\_de\_clientes();  Cliente ver\_primero (void); |
| Privado: pnodoCola primero, ultimo;  int clientes\_en\_cola; |

# CóDIGO FUENTE

## NodoLista.hpp

#ifndef NODOLISTA\_HPP

#define NODOLISTA\_HPP

#include <iostream>

#include "Cliente.hpp"

using namespace std;

class NodoLista

{

public:

NodoLista(Cliente c, NodoLista \*sig = NULL, NodoLista \*ant = NULL);

~NodoLista();

private:

Cliente cliente;

NodoLista \*siguiente;

NodoLista \*anterior;

friend class Lista;

};

typedef NodoLista\* pnodoLista;

#endif // NODOLISTA\_HPP

## NodoLista.cpp

#include "NodoLista.hpp"

NodoLista::NodoLista(Cliente c, NodoLista\* sig, NodoLista\* ant)

{

cliente = c;

siguiente = sig;

anterior = ant;

}

NodoLista::~NodoLista()

{

}

## Lista.hpp

#ifndef LISTA\_HPP

#define LISTA\_HPP

#include "NodoLista.hpp"

class Lista

{

public:

Lista();

~Lista();

void insertar (Cliente c);

Cliente eliminar ();

void mostrar ();

bool es\_vacia ();

int cantidad\_de\_clientes();

private:

pnodoLista puntero, primero;

int clientes\_en\_lista;

void insertar\_delante(pnodoLista n);

void insertar\_detras(pnodoLista n);

void insertar\_primero(pnodoLista n);

};

#endif // LISTA\_HPP

## Lista.cpp

#include "Lista.hpp"

Lista::Lista()

{

primero = NULL;

puntero = NULL;

clientes\_en\_lista = 0;

}

void Lista::insertar(Cliente c)

{

pnodoLista nuevo;

nuevo = new NodoLista(c);

long identificador\_cliente = c.get\_identificador\_de\_entrada();

puntero = primero;

if (!primero)

primero = nuevo;

else

{

while(puntero->siguiente and puntero->siguiente->cliente.get\_identificador\_de\_entrada() < identificador\_cliente)

puntero = puntero->siguiente;

if(!puntero->siguiente)

if(puntero->cliente.get\_identificador\_de\_entrada() > identificador\_cliente)

insertar\_detras(nuevo);

else

insertar\_primero(nuevo);

else

if(puntero->cliente.get\_identificador\_de\_entrada() > identificador\_cliente)

insertar\_detras(nuevo);

else

insertar\_delante(nuevo);

}

clientes\_en\_lista ++;

}

void Lista::insertar\_delante(pnodoLista n)

{

pnodoLista aux;

aux = puntero->siguiente;

puntero->siguiente = n;

n->siguiente = aux;

n->anterior = puntero;

}

void Lista::insertar\_detras(pnodoLista n)

{

pnodoLista aux;

aux = puntero->anterior;

puntero->anterior = n;

n->anterior = aux;

n->siguiente = puntero;

primero = n;

}

void Lista::insertar\_primero(pnodoLista n)

{

puntero->siguiente = n;

n->anterior = puntero;

}

Cliente Lista::eliminar()

{

pnodoLista nodo;

Cliente c;

nodo = primero;

primero = nodo -> siguiente;

c = nodo -> cliente;

delete nodo;

clientes\_en\_lista --;

return c;

}

bool Lista::es\_vacia()

{

return primero;

}

void Lista::mostrar()

{

pnodoLista aux = primero;

while(aux)

{

aux -> cliente.show\_cliente\_completo();

aux = aux -> siguiente;

}

}

int Lista::cantidad\_de\_clientes()

{

return clientes\_en\_lista;

}

Lista::~Lista()

{

while (primero)

{

eliminar();

clientes\_en\_lista --;

}

}

## NodoCola.hpp

#ifndef NODO\_HPP

#define NODO\_HPP

#include <iostream>

#include "Cliente.hpp"

using namespace std;

class NodoCola

{

public:

NodoCola(Cliente c, NodoCola \*sig = NULL);

~NodoCola();

private:

Cliente cliente;

NodoCola \*siguiente;

friend class Cola;

};

typedef NodoCola\* pnodoCola;

#endif // NODO\_HPP

## NodoCola.cpp

#include "NodoCola.hpp"

NodoCola::NodoCola(Cliente c, NodoCola\* sig)

{

cliente = c;

siguiente = sig;

}

NodoCola::~NodoCola()

{

}

## Cola.hpp

#ifndef COLA\_HPP

#define COLA\_HPP

#include "NodoCola.hpp"

class Cola

{

public:

Cola();

~Cola();

void insertar (Cliente c);

Cliente eliminar ();

void mostrar ();

bool es\_vacia ();

int cantidad\_de\_clientes();

Cliente ver\_primero (void);

private:

pnodoCola primero, ultimo;

int clientes\_en\_cola;

};

#endif // COLA\_HPP

## Cola.cpp

#include "Cola.hpp"

Cola::Cola()

{

primero = NULL;

ultimo = NULL;

clientes\_en\_cola = 0;

}

void Cola::insertar(Cliente c)

{

pnodoCola nuevo;

nuevo = new NodoCola(c);

if (ultimo)

ultimo -> siguiente = nuevo;

ultimo = nuevo;

if (!primero)

primero = nuevo;

clientes\_en\_cola ++;

}

bool Cola::es\_vacia()

{

return primero;

}

Cliente Cola::eliminar()

{

pnodoCola nodo;

Cliente c;

nodo = primero;

primero = nodo -> siguiente;

c = nodo -> cliente;

delete nodo;

if (!primero)

ultimo = NULL;

clientes\_en\_cola --;

return c;

}

Cliente Cola::ver\_primero()

{

return primero->cliente;

}

void Cola::mostrar()

{

pnodoCola aux = primero;

while(aux)

{

aux -> cliente.show\_cliente();

aux = aux -> siguiente;

}

}

int Cola::cantidad\_de\_clientes()

{

return clientes\_en\_cola;

}

Cola::~Cola()

{

while (primero)

{

eliminar();

clientes\_en\_cola --;

}

}

## Cliente.hpp

#ifndef CLIENTE\_HPP

#define CLIENTE\_HPP

#define HORA\_DE\_INICIO\_NO\_REGISTRADOS 120

#define TIEMPO\_MAXIMO\_DE\_COMPRA 10

#include <iostream>

#include <ctime>

#include <math.h>

#include <stdlib.h>

#include <time.h>

#include "windows.h"

using namespace std;

class Cliente

{

public:

Cliente();

Cliente(int t,bool r);

~Cliente();

void show\_cliente();

void show\_cliente\_completo();

void generar\_identificador\_de\_entrada();

long get\_identificador\_de\_entrada();

int get\_hora\_de\_inicio\_de\_compra();

int get\_tiempo\_de\_compra();

private:

char DNI [9];

int hora\_de\_inicio\_de\_compra;

int tiempo\_de\_compra;

bool registrado;

char identificador\_de\_entrada [4];

//Identificador de entrada

long numero\_de\_entrada;

long transformar\_identificador\_de\_entrada();

void show\_identificador\_de\_entrada();

//DNI

void generar\_DNI();

void show\_DNI();

//Hora de inicio de compra

void generar\_hora\_de\_inicio\_de\_compra();

void show\_hora\_de\_inicio\_de\_compra();

//Tiempo de compra

void generar\_tiempo\_de\_compra();

void show\_tiempo\_de\_compra();

};

#endif // CLIENTE\_HPP

## Cliente.cpp

#include "Cliente.hpp"

Cliente::Cliente()

{

}

Cliente::Cliente(int t, bool r)

{

this->hora\_de\_inicio\_de\_compra = t;

this->registrado = r;

generar\_DNI();

generar\_hora\_de\_inicio\_de\_compra();

generar\_tiempo\_de\_compra();

}

//DNI

void Cliente::generar\_DNI()

{

char letra [23] = {'T','R','W','A','G','M','Y','F','P','D','X','B','N','J','Z','S','Q','V','H','L','C','K','E'};

long dni = 0;

for (int x = 0; x < 8; x++)

{

DNI[x]=rand()%10;

dni+=DNI[x]\*pow(10,x);

}

dni %=23;

DNI[8] = letra [dni];

}

void Cliente::show\_DNI()

{

for (int x = 0; x < 8; x++)

cout << (char)(DNI[x]+48);

cout << '-' << DNI[8];

}

//HORA DE INICIO DE COMPRA

void Cliente::generar\_hora\_de\_inicio\_de\_compra()

{

if (!registrado)

hora\_de\_inicio\_de\_compra += HORA\_DE\_INICIO\_NO\_REGISTRADOS;

}

void Cliente::show\_hora\_de\_inicio\_de\_compra()

{

int hora = hora\_de\_inicio\_de\_compra/60, minutos = hora\_de\_inicio\_de\_compra%60;

if (hora < 10)

cout << '0';

cout << hora << ':';

if (minutos < 10)

cout << '0';

cout << minutos;

}

int Cliente::get\_hora\_de\_inicio\_de\_compra()

{

return hora\_de\_inicio\_de\_compra;

}

//TIEMPO DE COMPRA

void Cliente::generar\_tiempo\_de\_compra()

{

tiempo\_de\_compra = 1 + rand()%TIEMPO\_MAXIMO\_DE\_COMPRA;

}

void Cliente::show\_tiempo\_de\_compra()

{

cout << tiempo\_de\_compra;

if (tiempo\_de\_compra < 10)

cout << " ";

}

int Cliente::get\_tiempo\_de\_compra()

{

return tiempo\_de\_compra;

}

//CLIENTE

void Cliente::show\_cliente()

{

cout << "\t\tDNI: ";

show\_DNI();

cout << " - ";

cout << "Hora de inicio de compra: ";

show\_hora\_de\_inicio\_de\_compra();

cout << " - ";

cout << "Tiempo de compra: ";

show\_tiempo\_de\_compra();

cout << '\n';

}

//IDENTIFICADOR DE ENTRADA

void Cliente::generar\_identificador\_de\_entrada()

{

for(int x = 0; x < 4; x++)

identificador\_de\_entrada[x] = rand()%26 + 65;

numero\_de\_entrada = transformar\_identificador\_de\_entrada();

}

void Cliente::show\_identificador\_de\_entrada()

{

for(int x = 0; x < 4; x++)

cout << identificador\_de\_entrada[x];

}

long Cliente::transformar\_identificador\_de\_entrada()

{

long acumulador = 0;

for (int x = 0; x < 4; x++)

acumulador += (identificador\_de\_entrada[x])\*pow(100,(3-x));

return acumulador;

}

long Cliente::get\_identificador\_de\_entrada()

{

return numero\_de\_entrada;

}

void Cliente::show\_cliente\_completo()

{

cout << "\t\tDNI: ";

show\_DNI();

cout << " - ";

cout << "Hora de inicio de compra: ";

show\_hora\_de\_inicio\_de\_compra();

cout << " - ";

cout << "Tiempo de compra: ";

show\_tiempo\_de\_compra();

cout << " - ";

cout << "Identificador de entrada: ";

show\_identificador\_de\_entrada();

cout << '\n';

}

Cliente::~Cliente()

{

}

## Gestor.hpp

#ifndef GESTOR\_HPP

#define GESTOR\_HPP

#include <unistd.h>

#include "Cola.hpp"

#include "Lista.hpp"

#include "Cliente.hpp"

#define DIFERENCIA\_LLEGADA 5

#define VIP 15

#define NO\_REGISTRADOS 20

#define SERVIDORES\_NO\_REGISTRADOS 4

#define TIEMPO 5000000

using namespace std;

class Gestor

{

public:

Gestor();

~Gestor();

void generar\_solicitudes\_de\_clientes\_VIP(void);

void mostrar\_clientes\_VIP(void);

void borrar\_clientes\_VIP(bool texto);

void generar\_solicitudes\_de\_clientes\_no\_registrados(void);

void mostrar\_clientes\_no\_registrados(void);

void borrar\_clientes\_no\_registrados(bool texto);

void simular\_venta\_a\_clientes\_VIP(void);

void simular\_venta\_a\_clientes\_no\_registrados(void);

void mostrar\_entradas\_vendidas(void);

void generar\_arbol(void);

void buscar\_cliente(void);

void mostrar\_arbol\_en\_pre\_orden(void);

void mostrar\_arbol\_en\_post\_orden(void);

void mostrar\_arbol\_en\_in\_orden(void);

void dibujar\_arbol(void);

void reiniciar(void);

int cantidad\_no\_registrados();

int recuento\_VIP();

int recuento\_entradas();

private:

Cola VIP\_en\_espera;

Cola servidor\_VIP;

Cola no\_registrado\_en\_espera;

Cola servidor\_no\_registrado[SERVIDORES\_NO\_REGISTRADOS];

void show\_hora(int t);

int recuento\_no\_registrados();

void actualizar\_pantalla\_VIP(int tiempo, int tiempo\_en\_proceso);

void actualizar\_pantalla\_no\_registrado(int tiempo, int \* tiempo\_en\_proceso);

Lista clientes\_terminados;

};

#endif // GESTOR\_HPP

## Gestor.cpp

#include "Gestor.hpp"

Gestor::Gestor()

{

}

Gestor::~Gestor()

{

}

void Gestor::generar\_solicitudes\_de\_clientes\_VIP()

{

int tiempo\_cliente\_anterior = 0;

borrar\_clientes\_VIP(false);

VIP\_en\_espera.insertar(Cliente (0,true));

for(int x = 1; x < VIP; x++)

{

tiempo\_cliente\_anterior += rand()%6;

VIP\_en\_espera.insertar(Cliente (tiempo\_cliente\_anterior,true));

}

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole,10);

cout << endl << "\tClientes VIP generados!" << endl << endl;

}

void Gestor::mostrar\_clientes\_VIP()

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

if (VIP\_en\_espera.es\_vacia())

{

SetConsoleTextAttribute(hConsole,10);

cout << endl <<"\tClientes VIP: " << "\t\t\t"<< VIP\_en\_espera.cantidad\_de\_clientes() << endl << endl;

SetConsoleTextAttribute(hConsole,15);

VIP\_en\_espera.mostrar();

}

else

{

SetConsoleTextAttribute(hConsole,12);

cout << endl << "\tNo hay Clientes VIP en espera!" << endl << endl;

}

}

void Gestor::borrar\_clientes\_VIP(bool texto)

{

while(VIP\_en\_espera.es\_vacia())

VIP\_en\_espera.eliminar();

if (texto)

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole,13);

cout << endl << "\tClientes VIP borrados!" << endl << endl;

}

}

void Gestor::generar\_solicitudes\_de\_clientes\_no\_registrados()

{

int tiempo\_cliente\_anterior = 0;

borrar\_clientes\_no\_registrados(false);

no\_registrado\_en\_espera.insertar(Cliente (0,false));

for(int x = 1; x < NO\_REGISTRADOS; x++)

{

tiempo\_cliente\_anterior += rand()%6;

no\_registrado\_en\_espera.insertar(Cliente (tiempo\_cliente\_anterior,false));

}

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole,10);

cout << endl << "\tClientes NO REGISTRADOS generados!" << endl << endl;

}

void Gestor::mostrar\_clientes\_no\_registrados()

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

if (no\_registrado\_en\_espera.es\_vacia())

{

SetConsoleTextAttribute(hConsole,10);

cout << endl<< "\tClientes NO REGISTRADOS: " << "\t\t\t"<< VIP\_en\_espera.cantidad\_de\_clientes() << endl << endl;

SetConsoleTextAttribute(hConsole,15);

no\_registrado\_en\_espera.mostrar();

}

else

{

SetConsoleTextAttribute(hConsole,12);

cout << endl << "\tNo hay Clientes NO REGISTRADOS en espera!" << endl << endl;

}

}

void Gestor::borrar\_clientes\_no\_registrados(bool texto)

{

while(no\_registrado\_en\_espera.es\_vacia())

no\_registrado\_en\_espera.eliminar();

if (texto)

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole,13);

cout << endl << "\tClientes NO REGISTRADOS borrados!" << endl << endl;

}

}

void Gestor::simular\_venta\_a\_clientes\_VIP()

{

int tiempo = -1;

int tiempo\_en\_proceso = 0;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

if(VIP\_en\_espera.cantidad\_de\_clientes()>0)

{

Cliente c;

do

{

if (servidor\_VIP.cantidad\_de\_clientes()>0)

{

if (servidor\_VIP.ver\_primero().get\_tiempo\_de\_compra() == tiempo\_en\_proceso )

{

tiempo\_en\_proceso = 0;

c = servidor\_VIP.eliminar();

c.generar\_identificador\_de\_entrada();

clientes\_terminados.insertar(c);

}

else

tiempo\_en\_proceso++;

}

while((VIP\_en\_espera.cantidad\_de\_clientes() > 0) and (VIP\_en\_espera.ver\_primero().get\_hora\_de\_inicio\_de\_compra() == tiempo))

servidor\_VIP.insertar(VIP\_en\_espera.eliminar());

actualizar\_pantalla\_VIP(tiempo, tiempo\_en\_proceso);

tiempo ++;

usleep(TIEMPO);

system("cls");

}while(servidor\_VIP.cantidad\_de\_clientes() > 0 or VIP\_en\_espera.cantidad\_de\_clientes() > 0);

SetConsoleTextAttribute(hConsole,10);

cout << "\t\tVENTA DE ENTRADAS VIP TERMINADA" << endl;

actualizar\_pantalla\_VIP(tiempo, tiempo\_en\_proceso);

}

else

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole,12);

cout << endl << "\tNo hay clientes VIP" << endl << endl;

}

}

void Gestor::actualizar\_pantalla\_VIP(int tiempo, int tiempo\_en\_proceso)

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole,1115);

cout << endl <<"\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t\t Tiempo:";

show\_hora(tiempo);

cout << " "<< endl;

cout << endl;

cout << "\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl<<endl;

SetConsoleTextAttribute(hConsole,15);

cout <<"----------------------------------------------------------------------------------------------------------------------------------------------------"<< endl;

SetConsoleTextAttribute(hConsole,11);

cout << "\tClientes en espera: " << "\t\t\t"<< VIP\_en\_espera.cantidad\_de\_clientes() << endl << endl;

SetConsoleTextAttribute(hConsole,7);

VIP\_en\_espera.mostrar();

SetConsoleTextAttribute(hConsole,15);

cout << endl <<"----------------------------------------------------------------------------------------------------------------------------------------------------"<< endl;

SetConsoleTextAttribute(hConsole,14);

cout << "\tServidor VIP: " << "\t\t\t"<< " Tiempo en proceso: " << tiempo\_en\_proceso << endl << endl;

SetConsoleTextAttribute(hConsole,7);

servidor\_VIP.mostrar();

SetConsoleTextAttribute(hConsole,15);

cout << "----------------------------------------------------------------------------------------------------------------------------------------------------"<< endl;

SetConsoleTextAttribute(hConsole,2);

cout << "\tEntradas vendidas: "<< "\t\t\t"<< clientes\_terminados.cantidad\_de\_clientes() << endl << endl;

SetConsoleTextAttribute(hConsole,7);

clientes\_terminados.mostrar();

SetConsoleTextAttribute(hConsole,15);

cout <<"----------------------------------------------------------------------------------------------------------------------------------------------------"<< endl;

}

void Gestor::simular\_venta\_a\_clientes\_no\_registrados()

{

int tiempo = 119;

int tiempo\_en\_proceso[SERVIDORES\_NO\_REGISTRADOS];

for (int x = 0; x < SERVIDORES\_NO\_REGISTRADOS; x++)

tiempo\_en\_proceso[x] = 0;

if(no\_registrado\_en\_espera.cantidad\_de\_clientes()>0)

{

Cliente c;

do

{

for (int x = 0; x < SERVIDORES\_NO\_REGISTRADOS; x++)

{

if (servidor\_no\_registrado[x].cantidad\_de\_clientes() > 0)

{

if (servidor\_no\_registrado[x].ver\_primero().get\_tiempo\_de\_compra() == tiempo\_en\_proceso[x])

{

tiempo\_en\_proceso[x] = 0;

c = servidor\_no\_registrado[x].eliminar();

c.generar\_identificador\_de\_entrada();

clientes\_terminados.insertar(c);

}

else

tiempo\_en\_proceso[x]++;

}

}

while((no\_registrado\_en\_espera.cantidad\_de\_clientes() > 0) and (no\_registrado\_en\_espera.ver\_primero().get\_hora\_de\_inicio\_de\_compra() == tiempo))

{

int mejor\_servidor;

int menor = NO\_REGISTRADOS+1;

for(int x = 0; x < SERVIDORES\_NO\_REGISTRADOS; x ++)

{

if (servidor\_no\_registrado[x].cantidad\_de\_clientes() < menor)

{

menor = servidor\_no\_registrado[x].cantidad\_de\_clientes();

mejor\_servidor = x;

}

}

servidor\_no\_registrado[mejor\_servidor].insertar(no\_registrado\_en\_espera.eliminar());

}

actualizar\_pantalla\_no\_registrado(tiempo, tiempo\_en\_proceso);

system("cls");

tiempo ++;

usleep(TIEMPO);

}while(recuento\_no\_registrados() > 0 or no\_registrado\_en\_espera.cantidad\_de\_clientes() > 0);

cout << "\t\tVENTA DE ENTRADAS NO REGISTRADOS TERMINADA" << endl;

actualizar\_pantalla\_no\_registrado(tiempo, tiempo\_en\_proceso);

}

else

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole,12);

cout << endl << "\tNo hay clientes NO\_REGISTRADOS" << endl << endl;

}

}

void Gestor::actualizar\_pantalla\_no\_registrado(int tiempo, int \* tiempo\_en\_proceso)

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole,1115);

cout << endl <<"\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t\t Tiempo:";

show\_hora(tiempo);

cout << " "<< endl;

cout << "\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl<<endl;

SetConsoleTextAttribute(hConsole,15);

cout << endl <<"----------------------------------------------------------------------------------------------------------------------------------------------------"<< endl;

SetConsoleTextAttribute(hConsole,11);

cout << "\tClientes en espera: " << "\t\t\t"<< no\_registrado\_en\_espera.cantidad\_de\_clientes() << endl << endl;

SetConsoleTextAttribute(hConsole,7);

no\_registrado\_en\_espera.mostrar();

for (int x = 0; x < SERVIDORES\_NO\_REGISTRADOS; x++)

{

SetConsoleTextAttribute(hConsole,15);

cout <<"----------------------------------------------------------------------------------------------------------------------------------------------------"<< endl;

SetConsoleTextAttribute(hConsole,14);

cout << "\tServidor: "<< x << "\t\t\tTiempo en proceso: "<< tiempo\_en\_proceso[x] << endl << endl;

SetConsoleTextAttribute(hConsole,7);

servidor\_no\_registrado[x].mostrar();

}

SetConsoleTextAttribute(hConsole,15);

cout <<"----------------------------------------------------------------------------------------------------------------------------------------------------"<< endl;

SetConsoleTextAttribute(hConsole,10);

cout << "\tEntradas vendidas: "<< "\t\t\t"<< clientes\_terminados.cantidad\_de\_clientes() << endl << endl;

SetConsoleTextAttribute(hConsole,7);

clientes\_terminados.mostrar();

SetConsoleTextAttribute(hConsole,15);

cout <<"----------------------------------------------------------------------------------------------------------------------------------------------------"<< endl;

}

void Gestor::mostrar\_entradas\_vendidas()

{

if(clientes\_terminados.cantidad\_de\_clientes())

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole,10);

cout << endl << "\tEntradas vendidas: " << endl << endl;

SetConsoleTextAttribute(hConsole,15);

clientes\_terminados.mostrar();

}

else

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole,12);

cout << endl << "\tNo hay ENTRADAS vendidas" << endl;

}

}

## main.cpp

#include <iostream>

#include <ctime>

#include <math.h>

#include <iomanip>

#include "windows.h"

#include <Gestor.hpp>

using namespace std;

int main(int argc, char \*\*argv)

{

Gestor gestor;

char opcion;

ShowWindow(GetConsoleWindow(),SW\_MAXIMIZE);

srand(time(NULL));

do

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole,74);

cout << endl << endl;

cout << "\t---------------------------------------------------------------------------------------------\t\t\t" << endl;

cout << "\t MENU \t\t\t" << endl;

cout << "\t---------------------------------------------------------------------------------------------\t\t\t" << endl;

SetConsoleTextAttribute(hConsole,47);

cout << "\tNumero de clientes VIP: " << setw(4)<< gestor.recuento\_VIP() << " \t\t\t"<< endl;

cout << "\tNumero de clientes no registrados: " << setw(4)<< gestor.cantidad\_no\_registrados() << " \t\t\t" << endl;

cout << "\tEntradas vendidas: "<< setw(4) << gestor.recuento\_entradas() << " \t\t\t"<< endl << endl;

SetConsoleTextAttribute(hConsole,7);

cout << "\t\t A. Generar solicitudes de entrada de los clientes VIP." << endl;

cout << "\t\t B. Mostrar la cola de solicitudes en espera de los clientes VIP." << endl;

cout << "\t\t C. Borrar la cola de solicitues en espera de los clientes VIP." << endl;

cout << "\t\t D. Generar solicitudes de entrada de los clientes no registrados." << endl;

cout << "\t\t E. Mostrar la cola de solicitudes en espera de los clientes no registrados." << endl;

cout << "\t\t F. Borrar la cola de solicitues en espera de los clientes no registrados." << endl;

cout << "\t\t G. Simular el proceso de compra de los clientes VIP." << endl;

cout << "\t\t H. Simular el proceso de compra de los clientes no registrados." << endl;

cout << "\t\t I. Mostrar la lista de entradas vendidas." << endl;

cout << "\t\t J. Reiniciar el programa." << endl << endl;

cout << "\t\t K. Crear un arbol binario de busqueda ordenado por el DNI." << endl;

cout << "\t\t L. Buscar un cliente en el ABB, dado su DNI y mostrar sus datos." << endl;

cout << "\t\t M. Mostrar el ABB en pre-orden." << endl;

cout << "\t\t N. Mostrar el ABB en post-orden." << endl;

cout << "\t\t O. Mostrar el ABB en in-orden." << endl;

cout << "\t\t P. Dibujar el ABB." << endl << endl;

cout << "\t\t S. Salir del programa" << endl << endl;

SetConsoleTextAttribute(hConsole,15);

cout << "\tIndique la opcion deseada: ";

cin >> opcion ;

opcion = toupper(opcion);

system("cls");

switch(opcion)

{

case 'A':

gestor.generar\_solicitudes\_de\_clientes\_VIP();

break;

case 'B':

gestor.mostrar\_clientes\_VIP();

break;

case 'C':

gestor.borrar\_clientes\_VIP(true);

break;

case 'D':

gestor.generar\_solicitudes\_de\_clientes\_no\_registrados();

break;

case 'E':

gestor.mostrar\_clientes\_no\_registrados();

break;

case 'F':

gestor.borrar\_clientes\_no\_registrados(true);

break;

case 'G':

gestor.simular\_venta\_a\_clientes\_VIP();

break;

case 'H':

gestor.simular\_venta\_a\_clientes\_no\_registrados();

break;

case 'I':

gestor.mostrar\_entradas\_vendidas();

break;

case 'J':

gestor.reiniciar();

break;

case 'K':

gestor.generar\_arbol();

break;

case 'L':

gestor.buscar\_cliente();

break;

case 'M':

gestor.mostrar\_arbol\_en\_pre\_orden();

break;

case 'N':

gestor.mostrar\_arbol\_en\_post\_orden();

break;

case 'O':

gestor.mostrar\_arbol\_en\_in\_orden();

break;

case 'P':

gestor.dibujar\_arbol();

break;

case 'S':

SetConsoleTextAttribute(hConsole,13);

cout << endl<< endl <<"\tSaliendo del programa..." << endl << endl << endl;

break;

default:

SetConsoleTextAttribute(hConsole,12);

cout << endl <<"\tOpcion incorrecta!" << endl << endl;

break;

}

}

while(opcion != 'S');

return 0;

}