# Estructuras de Datos
# Práctica 2
# Laboratorio

**Realizado por:**
Javier Alonso Lledó 09109855S
Daniel Verduras Gallego 09102432K

# 1)Especificación concreta de la interfaz de los TAD'S implementados

## 1.1)TADS´s creados

Las operaciones se encuentran especificadas en el siguiente apartado

**espec** PEDIDO
**usa** BOOLEANOS, NATURALES, STRING
**var**
p: pedido
nombre, ncliente, direccion, tipo, tarjeta : string
tiempo, prioridad, preparacion: natural
erroneo: booleano
**generos** pedido

**espec** CLIENTE
**usa** LISTA , NATURALES y STRING
**var**
c: cliente
nombre , tipo , tarjeta : string
prioridad: int
listaPedidos: lista
**generos** cliente

**espec** NODO[PEDIDO]
**usa** PEDIDO
**parametro formal**
**generos** pedido
**fparametro**
**var**
n:nodo
ped: pedido
siguiente: puntero a pnodo
**generos** pnodo

**espec** PILA[PEDIDO]
**usa** BOOLEANOS, PEDIDO
**parametro formal**
**generos** pedido
**fparametro**
**var**
p:pila
cima: puntero a pnodo
**generos** pila
**ecuaciones de definitud**
esVacia(p) = F → Def(desapilar())
esVacia(p) = F → Def(mostrarCima())

**espec** COLA[PEDIDO]
**usa** BOOLEANOS, PEDIDO, NATURALES
**parametro formal**
**generos** pedido
**fparametro**
**var**
c:cola
primero, ultimo: puntero a pnodo
longitud: natural
**generos** cola
**ecuaciones de definitud**
esVacia(c) = F → Def(desencolar())
esVacia(c) = F → Def(prim())
esVacia(c) = F → Def(ult())

**espec** LISTA[PEDIDO]
**usa** BOOLEANOS , PEDIDO, NATURALES
**parametro formal**
**generos** pedido
**fparametro**
**var**
l:lista
primero, ultimo: puntero a pnodo

longitud: natural
**generos** lista
esVacia(l) = F → Def(resto())
esVacia(l) = F → Def(eult())
esVacia(l) = F → Def(prim())
esVacia(l) = F → Def(ult())

**espec** NODOARBOL[CLIENTE]
**usa** CLIENTE
**parametro formal**
**generos** cliente
**fparametro**
**var**
n:nodo
c: cliente
izq , der: puntero a nodoA
**generos** nodoA

**espec** ARBOL[CLIENTE]
**usa** CLIENTE , BOOLEANOS , NATURALES
**parametro formal**
**generos** cliente
**fparametro**
**var**
a: arbol
raiz: puntero a nodoA
**generos** arbol

**espec** WEB[PEDIDO]
**usa** BOOLEANOS, NATURALES, STRING ,PEDIDO, PILA, COLA , LISTA y ARBOL
**parametro formal**
**generos** pedido
**fparametro**
**var**

w:web
pilaErroneos: pila
colaReg: cola
colaNR: cola
listaEnviar1 , listaEnviar2: lista
arbolCliente: arbol
**generos** web

## 1.2) Definición de las operaciones del TAD (Nombre, argumentos y retorno)

A continuación se mostrará las operaciones de las TADS divididas por pertenencia:

**PEDIDO:**
Pedido:string → pedido
Pedido:string, string, string, string, string, int→ pedido
Pedido: string , string , int → pedido
getTipo: → string
getPrioridad: → string
toStr: → void
toStr2:→ void
restarPreparacion:→ void

**CLIENTE:**
Cliente: string , string , string , int → cliente
verCliente: → void

**NODO:**
Nodo: pedido, pnodo → pnodo

**PILA:**
Pila: → pila
apilar: pedido→ void
apilarOrdenado:pedido→ void
**parcial** desapilar: → void
**parcial** mostrarCima→ pedido
esVacia:→ booleano
verPila: → void

**COLA:**

Cola:→ cola
encolar: pedido → void
**parcial** desencolar: pedido → void
**parcial** prim: → pedido
**parcial** ult: → pedido
esVacia: → booleano
verCola → void

**LISTA:**

Lista: → lista
insertarIzq : pedido → void
insertarDer: pedido → void
insertarOrdenado : pedido → void
**parcial** resto: → void
**parcial** eult: → void
**parcial** prim: → pedido
**parcial** ult: → pedido
lon: → int
es_vacia: → bool
verLista: → void
verLista2: → void

**NODOARBOL:**

NodoArbol: cliente, nodoA , nodoA → nodoA

**ARBOL:**

ArbolB:→ arbol
ArbolB:cliente→ arbol
vacio:→ booleano
insertar: cliente , nodoA → void
insertarPedido: pedido , nodoA → void
esta: string, nodoA → booleano
buscar: string , nodoA → void
mostrarPreorden: nodoA → void
altura: nodoA → int
cuentaProducto: string , nodoA → int
mostrarVip: nodoA → void

**WEB:**

Web: → web
introducirPedido: pedido → void

introducirTxt: → int
pasarTiempo: → void
mostrarColas: → void
mostrarPila: → void
mostrarLista: → void
webVacia: → bool
buscarCliente: string → void
alturaArbol: → void
verPreorden: → void
cuentaProducto: string → void
mostrarVip: → void

## 2)Solución adoptada: Descripción de las dificultades encontradas

Las mayores dificultades que hemos enfrentado a lo largo del proyecto fueron principalmente relacionadas con la clase Web.
Encontramos que todas ellas iban en torno a la operación encargada de realizar la simulación. Para su correcto funcionamiento fue necesario establecer una serie de operaciones adicionales(introducirPedido y incluirListaEnvios e insertarWeb). Principalmente, los bugs que más tiempo nos tardaron fueron bugs de que no se hacían comprobaciones de si una estructura de datos era vacía o no, y por tanto, el programa crasheaba por utilizar funciones de EEDD que eran parciales.

Con el nuevo planteamiento de esta práctica 2 acerca del paso de los pedidos de las colas a las listas, surgían nuevos problemas a tener en cuenta, a parte de la dificultad principal que es tener dos listas que comprobar individualmente.
La dificultad más relevante en este aspecto fue que, al pasar pedidos cada 2 minutos, y no cada vez que se procesaba un pedido,podía ocurrir la siguiente situación:

Imaginemos que se está procesando un pedido de un cliente no registrado, y lleva 3 minutos de 5 que necesita. Ahora llega un pedido VIP que, al insertarse ordenado en la lista, queda por encima del pedido NR. Lo que habría

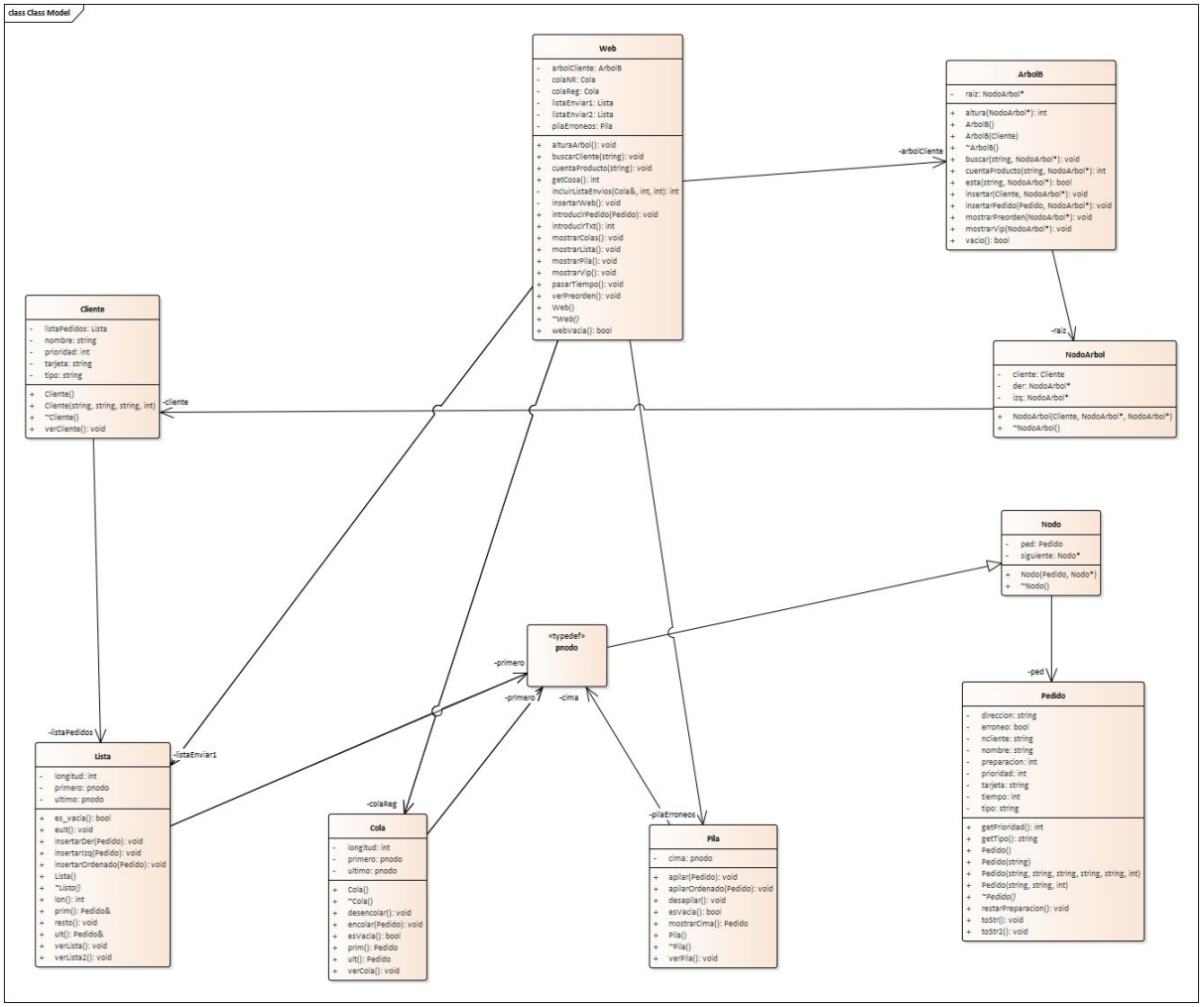
que hacer es empezar a procesar el pedido que acaba de llegar, pero guardar esos 3 minutos del pedido NR, para que cuando se vuelva a procesar le queden solo 2 minutos.

Después de pensarlo mucho, decidimos crear un nuevo atributo en pedido, llamado procesamiento, que es el tiempo de procesamiento real que le queda. Cuando llega a 0 se envía el pedido.

A parte de las dificultades con la clase Web, otras dificultades surgieron con temas del propio C++, lenguaje que aún no dominamos. Por ejemplo, el cin >> en un bucle actuaba solo en la primera iteración. Tuvimos que documentarnos por qué pasaba esto y solucionarlo(era porque había que limpiar el buffer).

# 3) Diseño de la relación entre las clases de los TAD implementados

## 3.1)Diagrama UML




## 3.2)Explicación de los métodos más destacados

En las estructuras de datos, los métodos más importantes son:

- En las listas, el método insertar ordenado. Cuando se pasan los pedidos de las colas a la lista para procesarlos, este método los inserta según su tipo(VIP los primeros,NVIP y NR los últimos).
- En las pilas, el método apilar ordenado. Apila los pedidos según su tipo. Encima de la pila los VIP, luego los NVIP y al fondo los NR.
- En la clase árbol, los dos métodos más importantes son insertar y insertarPedido. Ambas funciones se encargan de añadir nuevos clientes y pedidos al sistema respectivamente. En insertarPedido primero busca si existe el cliente al que pertenece el pedido. Si el cliente se encuentra en el

árbol se añade el pedido a su lista de pedidos y si no, añade un nuevo cliente al árbol.

- En la clase pedido, el método más a destacar es el constructor de Pedido, al que se le pasa el string en bruto y lo tiene que limpiar (las //) y depurar, por si es un pedido que le falten datos, que el tiempo esté mal introducido o que sea un tipo distinto de VIP, NVIP y NR. Básicamente tiene un contador que si no cuenta los 6 elementos necesarios, detecta que es un pedido erróneo automáticamente y genera los datos que faltan. Si el tiempo es, por ejemplo, un string (que no se puede convertir a int), genera el tiempo aleatoriamente entre 1 y 10 y lo marca como erróneo. Lo mismo si detecta un tipo distinto a los preestablecidos.

  El otro gran método es la función pasarTiempo en la clase Web. Es básicamente la función que simula todo el programa. Principalmente, la función se encarga de procesar los pedidos, es decir, simula el paso del tiempo para que, cuando se termine un pedido, se elimina de las listas y cree el cliente y el producto reducidos para añadirlos al árbol (estas funciones pertenecen a la clase ArbolB y son llamadas por la función pasarTiempo). Cada 2 minutos añade dos pedidos nuevos a las listas. La función no se termina hasta que el sistema se queda vacío o el usuario decida cancelar la ejecución. Esta función viene ayudada por la función auxiliar introducirListaEnvios, que lo que hace es, de una cola, pasar los pedidos a la lista de envíos que corresponda (la 1 o la 2), pero si el pedido es erróneo, lo pasa a la pila.
  También viene ayudada por la función insertarWeb, que es llamada cada 2 minutos por pasarTiempo y lo que hace es saber a qué tipos de pedidos (vips,erroneos + vips,nr...) les toca pasar a las listas, y añadirlos.

  Todas estas operaciones de insertar en la lista o apilar en la pila se hacen con los métodos nombrados anteriormente(apilarOrdenado e insertarOrdenado).

## 3.3)Explicación del comportamiento del programa

El programa comienza mostrando los nombres de los creadores y, a continuación, pregunta al usuario si quiere que los pedidos sean leidos de un fichero de texto o cogerlos del propio código(en el código hay implementados 10 pedidos, dos de ellos erróneos, para así tener el mínimo requerido en caso de que no se decida leer del fichero de texto.)

Si se elige leer del fichero de texto, si hay menos de 10 pedidos en el fichero de texto se introducirán pedidos del código hasta llegar al mínimo que son 10.

Una vez teniendo todos los pedidos se introducen al sistema Web y se muestra por pantalla la situación de las estructuras de datos del mismo. Como es lógico, en este punto sólo tendrán pedidos las colas, ya que aún no ha comenzado la simulación y no se han pasado los pedidos erróneos a la pila ni los pedidos normales a las listas de envíos.

En este punto ya comienza la simulación. Se muestra el estado de las EEDD una vez más, pero esta vez las listas de envíos ya tienen pedidos y la pila puede que tenga o no.

A partir de aquí se empiezan a procesar los primeros pedidos, simulando el paso del tiempo. Cada vez que se pulse una tecla del teclado avanzará el tiempo en 1 minuto.Cada dos minutos se añaden dos nuevos pedidos a las listas (uno a cada una) siguiendo con el criterio del enunciado. Cuando ha pasado el tiempo necesario para que se procese el pedido actual, te muestra los datos del pedido, se elimina de la lista y, a partir de él, se crea un objeto cliente y un pedido con atributos reducidos. Si el cliente no estaba en el árbol previamente se añade al árbol. Si estaba en el árbol entonces simplemente se añade el producto a la lista de productos de cliente, en el correspondiente nodo del árbol.

Cada vez que se envíe un pedido el usuario tiene la opción de terminar con el proceso, aunque aún queden pedidos por procesar.

Una vez terminado el proceso, te muestra todas las opciones con respecto al árbol de clientes creado.  Verlo,buscar un nombre,ver su altura,buscar un producto para ver cuántos pedidos de ese producto ha habido y mostrar los

clientes VIP. Este proceso es un bucle hasta que el usuario escoja el número 6, que es salir del sistema.

Cabe destacar que en múltiples puntos del programa se usa la función system("cls") para borrar la consola y que no se almacenen demasiados datos a la vez, ya que podría ser confuso.

# 4)Código fuente

El código fuente viene adjunto con la memoria pero añadimos el enlace de Github que utilizamos para tener registro de nuestro trabajo y poder acceder a todos los cambios introducidos en cualquier momento.
El repositorio de GitHub se ha mantenido privado hasta el momento del envío, para prevenir plagios.

https://github.com/javierxxal/PL2ED

# 5)Bibliografía

Diapositivas de Árboles Binarios de Búsqueda
StackOverflow(en general)