**ANÁLISIS Y DISEÑO DE ALGORITMOS I**

**PROYECTO DE CURSADA 2022**

**Segunda parte**

Grupo N° 53

Integrantes:

* Nahuel Fabian Flores Medrano - nffloresmedrano@gmail.com
* Abril Valentina Juarez - juarezabrilvalentina@gmail.com

Ayudante:

* Lucas Telesco

[**Introducción**](#_j8imumtjnob) **3**

[**Clase Banco**](#_1dwyd688wfju) **3**

[Implementacion:](#_kja59n7gc5lj) 4

[Detalles de funciones:](#_vp1xbjqsj6rf) 4

[**Clase Cliente**](#_wvxtspphcs09) **5**

[Implementacion:](#_hngxa7m9i6gs) 5

[Detalles de funciones:](#_b9eu8hika369) 6

[**Clase Criterio**](#_fp1d2hemfkns) **6**

[Implementacion:](#_rzdzcw16r83e) 7

[Detalles de funciones:](#_f5o5a8xyc6dr) 7

[**Clase Fila\_Criterio**](#_kcg71ua2xbqb) **7**

[Implementacion:](#_3k4bbgf3hlot) 8

[Detalles de funciones:](#_1v4zawbivp5p) 8

[**Impacto de las estructuras**](#_cwiqlabg3509) **8**

[Modificaciones realizadas respecto a l a primera entrega](#_v2br7s4l1tfh) 9

[**Código fuente**](#_89rfjadjkgzz) **10**

[Banco.h](#_kbqasyd6d7ra) 10

[Banco.cpp](#_ubbubnpyt1x8) 11

[Cliente.h](#_58a7s8y0uurl) 15

[Cliente.cpp](#_du978fhzik25) 16

[Criterio.h](#_n0fubop9bb7k) 17

[Criterio.cpp](#_wkglpy8uypfv) 19

[Fila\_Criterio.h](#_kc4u1i6vegva) 20

[Fila\_Criterio.cpp](#_8ms9cqghlff1) 21

[Fila.h](#_505egt8vp4r) 21

[Fila.cpp](#_4izusxvmbdos) 22

[Lista.h](#_97xe58jotxzc) 24

[Lista.cpp](#_cv95tecmgxtp) 25

[Main.cpp](#_oayjd41yozk7) 29

# Introducción

En el presente informe se aborda la temática de la implementación de TDA a través del proyecto presentado que consiste en simular, de forma muy simplificada, la llegada, espera y atención de los clientes durante su operatoria en un banco.

A continuación se especificarán los tipos de datos nuevos creados para esta entrega, con su implementación mediante las clases y el análisis de cada una y de sus funciones.

# Clase Banco

**Class** Banco

**Imports** Fila[Cliente], Cliente, Criterio, Lista[Cliente],

**Basic Constructors** Banco, IngresarCliente

**Effective**

**Type** Banco

**Operations**

Banco: ->Banco;

IngresarCliente: Cliente x Banco -> Banco;

Abrir\_Cola\_Criterio: string x Boolean x Banco -> Boolean;

Cerrar\_Cola\_Especial: int x Banco - > Boolean;

AtenderProxCliente: int x Banco -> Cliente;

Reencolar\_Clientes: Fila\_criterio x Banco -> Banco;

**End\_class**

## Implementacion:

Para su implementación se definen como atributos de la clase una cola de llegada, de tipo Fila[Cliente], dos colas especiales (pr\_cola\_especial y sg\_cola\_especial) de tipo fila\_criterio y una lista simple de operaciones, de tipo Lista[Cliente]. Además se definen métodos propios de la clase que permiten realizar las operatorias pedidas en un banco: ingresar un cliente, atender al próximo cliente de una cola dada, abrir una nueva cola especial con criterio, cerrar una cola especial, listar las operaciones atendidas según un rango de montos.

Creando esta clase se hace uso de las clases creadas en la primera parte del TPE. A través de la delegación de tareas a ellas el código termina siendo más corto y fácil de entender.

## Detalles de funciones:

* **Banco:** Constructor de la clase. Complejidad temporal: O(1).
* **IngresarCliente:** Agregar elementos a la Cola general o especial abierta en la que pueda acceder. Tiempo de complejidad O(max(l,s1,s2)) siendo l el tamaño de la cola de llegada, s1 el tamaño de la primera cola especial y s2 de la segunda cola especial.
* **AtenderProxCliente:** Obtiene el primer cliente de una cola dada y procesa su transacción. Complejidad O(1) ya que solo saca el primero de la fila.
* **Abrir\_Cola\_Criterio:** Dado un criterio, el banco intenta abrir una cola especial y asignar su criterio. Luego busca en la cola de llegada los Clientes que cumplan con el critrio y los cambia de lista. Complejidad O(n.m) siendo “n” la cantidad de clientes en la cola de llegada y “m” la cantidad de elementos de la nueva cola a agregar.
* **Cerrar\_Cola\_Especial:** Dado un número de cola especial, y solo si se encuentra vacía, actualiza el estado de esta a cerrada. Complejidad temporal de O(1).
* **cantidad\_colas\_abiertas:** Devuelve la cantidad de colas especiales abiertas. Complejidad temporal de O(1).

# Clase Cliente

**Class** Cliente

**Imports** String, Boolean

**Basic Constructors** Cliente, setNombre

**Effective**

**Type** Cliente

**Operations**

Cliente: -> Cliente;

setNombre: Cliente x String -> Cliente;

setEdad: Cliente x Integer -> Cliente;

setEstado: Cliente x Boolean -> Cliente;

setOperacion: Cliente x String -> Cliente;

setDestino:Cliente x String -> Cliente;

setMonto: Cliente x Integer -> Cliente;

getNombre: Cliente -> String;

getEdad: Cliente -Integer;

getEstado: Cliente -> Boolean;

getOperacion: Cliente -> String;

getDestino: Cliente -> String;

getMonto: Cliente -> Integer;

\_==\_: Cliente x Cliente -> Boolean;

**End\_class**

## Implementacion:

Un dato de tipo cliente posee como atributos todos los datos que solicita el banco en el marco del problema: nombre (de tipo string), edad (de tipo integer), operación a realizar (de tipo string), destinatario de la operación (de tipo string) y si es cliente del banco o no (de tipo boolean). A su vez implementa métodos que permiten obtener los valores de cada uno de los atributos (getters) y métodos que permiten modificarlos (setters), y un método especial (operator==), que permite definir la operación de igualdad entre dos instancias del tipo cliente.

## Detalles de funciones:

* **Cliente:** Constructor de la clase. Complejidad temporal: O(1);
* **setNombre, setEdad, setEstado, setOperacion, setDestino:** Asignan el atributo (nombre/edad/estado/operacion/destino) de una instancia la clase. Complejidad temporal de O(1).
* **getNombre, getEdad, getEstado, getOperacion, getDestino:** Obtienen el valor del atributo (nombre/edad/estado/operacion/destino) de una instancia de la clase. Complejidad temporal de O(1).
* **operator==:** Compara los nombres de dos elementos de tipo Cliente y si son iguales, establece su igualdad como objetos Cliente. Complejidad temporal de O(1).

# Clase Criterio

**Class** Criterio

**Imports** String, Boolean, Cliente

**Basic Constructors** Criterio, setCriterio, setCuenta

**Effective**

**Type** Criterio

**Operations**

Criterio: -> Criterio;

setCriterio: Criterio x String -> Criterio;

setCuenta: Criterio x Boolean -> Criterio

getCriterio: Criterio -> String;

cumple\_Criterio: Criterio x Cliente -> Boolean;

**End\_class**

## Implementacion:

La clase Criterio representa al criterio combinado que se utiliza en una cola especial, cuenta con dos atributos, uno (de tipo boolean) que indica si se atienden a clientes del banco o no, y otro atributo (de tipo string) que indica la operación a realizar que toma en cuenta el criterio. Implementa funciones para modificar y obtener estos atributos.

## Detalles de funciones:

* **setCriterio, setCuenta:** Asignan el tipo de operación y si es cliente del banco o no como valores de los atributos. Complejidad temporal de O(1).
* **getCriterio:** Obtiene el atributo que representa el tipo de operación de la instancia de Criterio. Complejidad temporal de O(1).
* **cumple\_criterio:** Dada una instancia de la clase Cliente, devuelve true si la operación coincide con el atributo de operación del Cliente y a su vez el estado del cliente (si es cliente del banco o no) coincide con el atributo de cuenta del criterio. Complejidad temporal de O(1).

# Clase Fila\_Criterio

**Class** Fila\_criterio

**Imports** String, Boolean, Criterio

**Inherits** Fila[Cliente]

**Basic Constructors** Fila\_criterio, setCriterio

**Effective**

**Type** Fila\_criterio

**Operations**

Fila\_criterio: -> Fila\_criterio;

setCriterio: Fila\_criterio x Criterio x String -> Fila\_criterio;

cumple\_Criterio: Fila\_criterio x Criterio x Cliente -> Boolean;

getCriterioText: Fila\_Criterio -> String;

getCriterio: Fila\_criterio -> Criterio;

**End\_class**

## Implementacion:

La clase fila\_criterio es derivada (clase hija) de la clase base Fila<Cliente> (clase padre), así, hereda sus mismos métodos y propiedades, agregando la particularidad de poseer un atributo de tipo Criterio.

## Detalles de funciones:

* **setCriterio:** Asigna el tipo de operación del atributo de tipo Criterio. Complejidad temporal de O(1).
* **cumple\_Criterio:** Dado un cliente (de tipo Cliente) retorna true si cumple con el criterio de la instancia de fila\_criterio. Complejidad temporal de O(1).
* **getCriterioText:** Obtiene el tipo de operación del atributo de tipo Criterio de la clase. Complejidad temporal de O(1).
* **getCriterio:** Obtiene el criterio de la instancia de fila\_criterio. Complejidad temporal de O(1).

# Impacto de las estructuras

Para la implementación de las colas se hizo uso de la clase Fila que a su vez deriva de la clase Lista, que es una estructura basada en punteros a nodos. La clase Fila también podría haberse implementado mediante un arreglo en lugar de una lista de nodos, pero esto no supondría una mejoría ya que sería un mayor costo espacial, al no ser de tamaño dinámico, y en cuanto a lo temporal, acceder a los elementos sería más rápido, por otra parte, por ejemplo, el eliminar un elemento de una Fila estructurada como un arreglo implicaría realizar un corrimiento sobre todo el arreglo. Por ende, se opta como mejor opción la estructura de lista de nodo con restricciones sobre las operaciones, como el tipo Fila.

La creación de una clase Criterio permite que cada instancia de esta clase pueda decidir por su cuenta si un cliente cumple con el propio criterio o no, lo que facilita luego la implementación de las colas especiales en las que se agregan clientes solo si estos cumplen con el criterio de la cola, así la clase fila\_criterio sólo se encarga de agregar el elemento a la cola, mientras que el criterio es el que decide si el cliente cumple o no con las condiciones necesaria.

El tipo de dato Cliente es necesario para poder representar todos los datos de cada cliente que va a operar en el banco, además añade la funcionalidad de determinar si dos clientes son iguales o no, que se requiere para poder utilizarla en otras clases.

Por último la clase Banco es una estructura conformada por colas especiales (fila\_criterio), una cola de llegada (fila) y una lista de operaciones atendidas (lista), con esta estructura puede realizar las operatorias necesarias en el contexto de atender a los clientes, abrir y cerrar colas especiales, listar las operaciones, entre otras, sin tener que preocuparse, por ejemplo, de cómo una cola especial (en realidad, su criterio) determina si un cliente nuevo puede agregarse a esa cola o no, ya que esto es responsabilidad de la clase fila\_especial.

Mejoras que podrían hacerse

En la implementación de las colas con criterio se podría usar una lista de colas para un futuro crecimiento en las colas de ser necesario.

Dado que el problema se trata de cola de Clientes, se puede transformar la lista y la fila a manejar solo de clase clientes y no usarlas parametrizadas, ya que no se espera otro dato.

Conclusiones del análisis

Luego de poner en práctica las clases TDA se concluyó que permiten ignorar los detalles específicos del problema y encapsularlos en una clase, permitiéndonos heredar de la misma para agregar funciones más específicas, como la fila con criterio del problema resuelto en este trabajo.

## Modificaciones realizadas respecto a l a primera entrega

El código de la primera entrega se modificó de modo que la clase Fila sea como la clase Lista pero con restricciones en sus métodos. Esto es, tiene como atributo a una lista y sus operaciones de agregar y eliminar se ven modificadas de manera que se agrega un elemento al final de la fila y se elimina el primer elemento de la fila.

# Código fuente

## Banco.h

| **#ifndef BANCO\_H** **#define BANCO\_H**  **#include "../fila.h"** **#include "fila\_criterio.h"** **#include "Cliente.h"** **#include "Criterio.h"** **#include "../lista.h"**  **#include <string>**  **class** **Banco** {  **public**:  Banco();  **virtual** ~Banco();   **protected**:   **private**:   Fila<Cliente> \* cola\_llegada = **new** Fila<Cliente>();  fila\_criterio \* pr\_cola\_especial;  fila\_criterio \* sg\_cola\_especial;  Lista<Cliente> \* operaciones;    **public**:  **void** **reencolar\_clientes**(fila\_criterio \* cola);  **void** **IngresarCliente**(Cliente cl);  Cliente **Atender\_Prox\_Cliente**(**int** opcion);  **bool** **Abrir\_Cola\_Criterio**(std::string crit,**bool** tieneCuenta);  **bool** **Cerrar\_Cola\_Especial**(**const** **int** cola);  **bool** **pr\_cola\_abierto**();  **bool** **sg\_cola\_abierto**();  std::string **pr\_cola\_getcrit**();  std::string **sg\_cola\_getcrit**();  **void** **iniciar\_iterador**();  **void** **avanzar\_iterador**();  Cliente **elemento\_iterador**();  **bool** **final\_iterador**();  **int** **cantidad\_colas\_abiertas**();   };  **#endif *// BANCO\_H*** |
| --- |

## Banco.cpp

| *#include "Banco.h"* *#include "../fila.cpp"* *#include "../lista.cpp"*  **int** Banco::cantidad\_colas\_abiertas(){  **int** cant = 0;  **if** (this->pr\_cola\_especial) cant++;  **if** (this->sg\_cola\_especial) cant++;  **return** cant; } std::string Banco::pr\_cola\_getcrit(){  **return** this->pr\_cola\_especial->getCriterioText(); }  std::string Banco::sg\_cola\_getcrit(){  **return** this->sg\_cola\_especial->getCriterioText(); }   void Banco::reencolar\_clientes(fila\_criterio \* cola){  cola\_llegada->iniciar\_iterador();  **while** (!(cola\_llegada->final\_iterador())){  Cliente aux = cola\_llegada->elemento\_iterador();  **if** (cola->cumple\_criterio(aux)){  cola->agregar(aux);  cola\_llegada->eliminar\_elem\_lista(aux);  }  cola\_llegada->avanzar\_iterador();  } };  **bool** Banco::Abrir\_Cola\_Criterio(**const** std::string crit, **bool** tieneCuenta){  Criterio criter;  criter.setCriterio(crit);  criter.setCuenta(tieneCuenta);  **if** (this->pr\_cola\_especial == **NULL**){  this->pr\_cola\_especial = **new** fila\_criterio(criter);  this->reencolar\_clientes(pr\_cola\_especial);  **return** **true**;  }  **if** (this->sg\_cola\_especial == **NULL** && crit != pr\_cola\_especial->getCriterioText()) {  this->sg\_cola\_especial = **new** fila\_criterio(criter);  this->reencolar\_clientes(sg\_cola\_especial);  **return** **true**;  }  **return** **false**; } **bool** Banco::pr\_cola\_abierto(){  **if** (this->pr\_cola\_especial == **NULL**) **return** **false**;  **return** **true**; } **bool** Banco::sg\_cola\_abierto(){  **if** (this->sg\_cola\_especial == **NULL**) **return** **false**;  **return** **true**; } */\* \* @param opcion contiene el nro de cola a tomar el cliente \*/* Cliente Banco::Atender\_Prox\_Cliente(**int** opcion){  Cliente aux;  **if** (opcion == 1){  **if** (cola\_llegada->cantidad\_elementos() > 0){  aux = this->cola\_llegada->obtenerElemento();  this->operaciones->agregar(aux);  cola\_llegada->eliminar\_elemento();  **return** aux;  }  }  **if** (opcion == 2){  **if** (pr\_cola\_especial->cantidad\_elementos() > 0){  aux = this->pr\_cola\_especial->obtenerElemento();  this->operaciones->agregar(this->pr\_cola\_especial->obtenerElemento());  pr\_cola\_especial->eliminar\_elemento();  **return** aux;  }  }  **if** (opcion == 3){  **if** (sg\_cola\_especial->cantidad\_elementos() > 0){  aux = this->sg\_cola\_especial->obtenerElemento();  this->operaciones->agregar(this->sg\_cola\_especial->obtenerElemento());  sg\_cola\_especial->eliminar\_elemento();  **return** aux;  }  }  **return** aux; };  **bool** Banco::Cerrar\_Cola\_Especial(**const** **int** cola){  **if** (cola == 1){  **if** (this->pr\_cola\_especial->cantidad\_elementos() == 0){  delete(pr\_cola\_especial);  this->pr\_cola\_especial = **NULL**;  **return** **true**;  }  } **else** {  **if** (this->sg\_cola\_especial->cantidad\_elementos() == 0){  delete(sg\_cola\_especial);  this->sg\_cola\_especial = **NULL**;  **return** **true**;  }  }  **return** **false**; };  void Banco::IngresarCliente(Cliente cl){  */// INGRESAN COLAS CREADAS*  **if** ( (pr\_cola\_especial) && pr\_cola\_especial->getCriterio().cumple\_criterio(cl)){  pr\_cola\_especial->agregar(cl);  **return**;  }  **if** ( (sg\_cola\_especial) && sg\_cola\_especial->getCriterio().cumple\_criterio(cl)){  sg\_cola\_especial->agregar(cl);  **return**;  }  this->cola\_llegada->agregar(cl); }  void Banco::iniciar\_iterador() {  this->operaciones->iniciar\_iterador(); }  void Banco::avanzar\_iterador () {  this->operaciones->avanzar\_iterador(); }  Cliente Banco::elemento\_iterador () {  **return** this->operaciones->elemento\_iterador(); }  **bool** Banco::final\_iterador () {  **return** this->operaciones->final\_iterador(); }  Banco::Banco() {  this->pr\_cola\_especial = **NULL**;  this->sg\_cola\_especial = **NULL**;  this->operaciones = **new** Lista<Cliente>(); }  Banco::~Banco() {  delete(this->cola\_llegada);  delete(this->pr\_cola\_especial);  delete(this->sg\_cola\_especial);  delete(this->operaciones); } |
| --- |

## Cliente.h

| **#ifndef CLIENTE\_H** **#define CLIENTE\_H**  **#include <string>** **class** **Cliente** {  **public**:  Cliente();  **virtual** ~Cliente();  **void** **setNombre**(std::string name);  **void** **setEdad**(**int** age);  **void** **setEstado**(**bool** estado);  **void** **setOperacion**(std::string oper);  **void** **setDestino**(std::string oper);  **void** **setMonto**(**int** monto);  std::string **getNombre**();  **int** **getEdad**();  **bool** **getEstado**();  std::string **getOperacion**();  std::string **getDestino**();  **int** **getMonto**();  **bool** **operator**==(Cliente cl2);    **protected**:   **private**:  std::string nombre;  **int** edad;  **bool** es\_cliente;  std::string tipo\_operacion;  std::string destino\_operacion;  **int** monto; };  **#endif *// CLIENTE\_H*** |
| --- |

## Cliente.cpp

| **#include "Cliente.h"**   **void** Cliente::setNombre(std::string name){  **this**->nombre = name; }  **void** Cliente::setEdad(**int** age){  **this**->edad = age; }  **void** Cliente::setEstado(**bool** estado){  **this**->es\_cliente = estado; } **void** Cliente::setOperacion(std::string oper){  **this**->tipo\_operacion = oper; } **void** Cliente::setMonto(**int** monto){  **this**->monto = monto; } **void** Cliente::setDestino(std::string destino){  **this**->destino\_operacion = destino; }  std::string Cliente::getNombre(){  **return** **this**->nombre; }  **int** Cliente::getEdad(){  **return** **this**->edad; }  **bool** Cliente::getEstado(){  **return** **this**->es\_cliente; }  std::string Cliente::getOperacion(){  **return** **this**->tipo\_operacion; }  **int** Cliente::getMonto(){  **return** **this**->monto; }  std::string Cliente::getDestino(){  **return** **this**->destino\_operacion; }  **bool** Cliente::**operator**==(Cliente cl2){  **return** **this**->nombre == cl2.getNombre(); }  Cliente::Cliente() {  *//ctor* }  Cliente::~Cliente() {  *//dtor* } |
| --- |

## Criterio.h

| **#include "Cliente.h"**   **void** Cliente::setNombre(std::string name){  **this**->nombre = name; }  **void** Cliente::setEdad(**int** age){  **this**->edad = age; }  **void** Cliente::setEstado(**bool** estado){  **this**->es\_cliente = estado; } **void** Cliente::setOperacion(std::string oper){  **this**->tipo\_operacion = oper; } **void** Cliente::setMonto(**int** monto){  **this**->monto = monto; } **void** Cliente::setDestino(std::string destino){  **this**->destino\_operacion = destino; }  std::string Cliente::getNombre(){  **return** **this**->nombre; }  **int** Cliente::getEdad(){  **return** **this**->edad; }  **bool** Cliente::getEstado(){  **return** **this**->es\_cliente; }  std::string Cliente::getOperacion(){  **return** **this**->tipo\_operacion; }  **int** Cliente::getMonto(){  **return** **this**->monto; }  std::string Cliente::getDestino(){  **return** **this**->destino\_operacion; }  **bool** Cliente::**operator**==(Cliente cl2){  **return** **this**->nombre == cl2.getNombre(); }  Cliente::Cliente() {  *//ctor* }  Cliente::~Cliente() {  *//dtor* } |
| --- |

## Criterio.cpp

| **#include "Criterio.h"** **#include <iostream>**  **void** Criterio::setCriterio(**const** std::string crit){  **this**->operacion = crit; };  **void** Criterio::setCuenta(**bool** cuenta){  **this**->cuenta = cuenta; };  **bool** Criterio::cumple\_criterio(Cliente cli){  **if** ((**this**->operacion == cli.getOperacion()) && (**this**->cuenta == cli.getEstado())) **return** true;  **return** false; };  std::string Criterio::getCriterio(){  **return** **this**->operacion; }  Criterio::Criterio() {  *//ctor* }  Criterio::~Criterio() {  *//dtor* } |
| --- |

## Fila\_Criterio.h

| **#ifndef FILA\_CRITERIO\_H** **#define FILA\_CRITERIO\_H**  **#include "../fila.h"** **#include "Cliente.h"** **#include "Criterio.h"** **#include <string>**  **class** **fila\_criterio**: **public** Fila<Cliente> {   **private**:  Criterio criter;  **bool** abierta = false;   **public**:  fila\_criterio();  fila\_criterio(Criterio criter);  **virtual** ~fila\_criterio();   **void** **setCriterio**(**const** std::string crit){  criter.setCriterio(crit);  }   **bool** **cumple\_criterio**(Cliente cli){  **if** (criter.cumple\_criterio(cli)) **return** true;  **return** false;  }   std::string **getCriterioText**(){  **return** **this**->criter.getCriterio();  }   Criterio **getCriterio**(){  **return** **this**->criter;  } };  **#endif *// FILA\_CRITERIO\_H*** |
| --- |

## Fila\_Criterio.cpp

| **#include "fila\_criterio.h"**    fila\_criterio::fila\_criterio(Criterio criter) {  **this**->criter = criter; }  fila\_criterio::fila\_criterio() {  *//ctor* }  fila\_criterio::~fila\_criterio() {  *//dtor* } |
| --- |

## Fila.h

| **#ifndef FILA\_EJEMPLO\_H** **#define FILA\_EJEMPLO\_H**  **#include "lista.h"** **#include "Cliente.h"** **#include "Criterio.h"**  **template** <**typename** elemL> **class** **Fila**{   **private**:   Lista<elemL> lista;   **public**:   Fila();  ~Fila();   **void** **agregar**(elemL elemento);   **int** **cantidad\_elementos**();   elemL **verificar**();   elemL **obtenerElemento**();   **bool** **es\_vacia**();   **void** **eliminar\_elemento**();   **void** **eliminar\_elem\_lista**(elemL L);   **void** **iniciar\_iterador**();   **void** **avanzar\_iterador**();   **bool** **final\_iterador**();   elemL **elemento\_iterador**();   elemL **obtener\_elemento\_it**();  };  **#endif** |
| --- |

## Fila.cpp

| **#include "fila.h"**  **#include "Cliente.h"** **#include <string>** **#include <stdlib.h>**  **#include <iostream>**  **using** **namespace** std;  **template** <**typename** elemL> Fila<elemL>::Fila() {  }  **template** <**typename** elemL> Fila<elemL>::~Fila() {  }  **template** <**typename** elemL> **void** Fila<elemL>::iniciar\_iterador(){  **this**->lista.iniciar\_iterador(); };  **template** <**typename** elemL> elemL Fila<elemL>::elemento\_iterador(){  **return** **this**->lista.elemento\_iterador(); };  **template** <**typename** elemL> **void** Fila<elemL>::avanzar\_iterador(){  **this**->lista.avanzar\_iterador(); };  **template** <**typename** elemL> elemL Fila<elemL>::obtener\_elemento\_it(){  **return** **this**->lista.obtener\_elemento\_it(); };  **template** <**typename** elemL> **bool** Fila<elemL>::final\_iterador(){  **return** **this**->lista.final\_iterador(); };  **template** <**typename** elemL> **void** Fila<elemL>::agregar(elemL elemento){  **this**->lista.agregar(elemento); }  **template** <**typename** elemL> **int** Fila<elemL>::cantidad\_elementos(){  **return** **this**->lista.cantidad\_elementos(); }  **template** <**typename** elemL> elemL Fila<elemL>::obtenerElemento(){  elemL aux = **this**->lista.primer\_elemento();  **return** aux; }  **template** <**typename** elemL> **bool** Fila<elemL>::es\_vacia(){  **return** (**this**->lista.es\_vacia()); }  **template** <**typename** elemL> **void** Fila<elemL>::eliminar\_elemento(){  elemL aux = **this**->lista.primer\_elemento();  **this**->lista.eliminar\_elemento(aux); }  **template** <**typename** elemL> **void** Fila<elemL>::eliminar\_elem\_lista(elemL L){  **this**->lista.eliminar\_elemento(L); }     **template** **class** **Fila**<unsigned int>; **template** **class** **Fila**<float>; **template** **class** **Fila**<string>; **template** **class** **Fila**<char>; **template** **class** **Fila**<bool>; **template** **class** **Fila**<Cliente>; |
| --- |

## Lista.h

| **#ifndef LISTA\_EJEMPLO\_H** **#define LISTA\_EJEMPLO\_H**  **#include <stdlib.h>** **#include "Cliente.h"**  **template** <**typename** elemL> **class** **Lista**{   **private**:  **struct** **nodo\_lista** {  elemL elemento;  nodo\_lista \* sig;  };    **int** cant;  **void** **vaciar**(nodo\_lista \* & primero);  **void** **remover\_it\_nodo**();  nodo\_lista \* Primero, \*it, \*it\_previo;   **public**:   Lista();  ~Lista();    **void** **agregar**(elemL elemento , **int** posicion = 0);   **int** **cantidad\_elementos**();   **bool** **verificar**( **const** elemL elemento);   **bool** **es\_vacia**() **const**;   **bool** **eliminar\_elemento**(elemL elemento);   **void** **iniciar\_iterador**();   **void** **avanzar\_iterador**();   elemL **elemento\_iterador**();   elemL **obtener\_elemento\_it**();   **bool** **final\_iterador**();   elemL **primer\_elemento**();  };  **#endif** |
| --- |

## Lista.cpp

| **#include "lista.h"**   **#include <iostream>** **#include <string>** **#include <stdlib.h>**  **using** **namespace** std;  **template** <**typename** elemL> Lista<elemL>::Lista() {   **this**->Primero = NULL;  **this**->cant = 0;  }  **template** <**typename** elemL> Lista<elemL>::~Lista() {  vaciar(**this**->Primero); }  **template** <**typename** elemL> **void** Lista<elemL>::vaciar(nodo\_lista \* & primero) {   nodo\_lista \* aux;  aux = primero;  **while** (aux != NULL) {  primero = primero->sig;  **delete** aux;  aux = primero;  } }  **template** <**typename** elemL> **void** Lista<elemL>::agregar(elemL elemento,**int** posicion){  nodo\_lista \* aux = **new** nodo\_lista();  aux->elemento = elemento;  aux->sig = NULL;  nodo\_lista \* l = **this**->Primero;  **if** (l != NULL){  **if** (posicion == 0){  nodo\_lista \* cursor = l;  **while** (cursor->sig != NULL){  cursor = cursor->sig;  }  cursor->sig = aux;  }  **else**{  **if** (posicion == 1){  aux->sig = l;  **this**->Primero = aux;  }  **else**{  **while** (posicion > 1){  l = l->sig;  posicion--;  }  aux->sig = l->sig;  l->sig = aux;  }  }  }  **else**{  **this**->Primero = aux;  }  **this**->cant++; }  **template** <**typename** elemL> **int** Lista<elemL>::cantidad\_elementos(){  **return** **this**->cant; }  **template** <**typename** elemL> **bool** Lista<elemL>::verificar( **const** elemL elemento){  nodo\_lista \* l = **this**->Primero;  **while** (l != NULL){  **if** (l->elemento == elemento){  **return** true;  }  l = l->sig;  }  **return** false; }  **template** <**typename** elemL> **bool** Lista<elemL>::es\_vacia() **const**{  **if** (Primero != NULL) **return** false;  **return** true; }  **template** <**typename** elemL> **bool** Lista<elemL>::eliminar\_elemento(elemL elemento){  nodo\_lista \* cursor = Primero;  nodo\_lista \* aux = cursor;  **bool** found = false;  **while** ((cursor != NULL) && (!found)){   **if** (cursor->elemento == elemento){  **if** (cursor == Primero) {  aux = cursor;  Primero = cursor->sig;  **delete** aux;  }  **else**{  aux->sig = cursor->sig;  **delete** cursor;  }  cant--;  found = true;  }  **else**{  aux = cursor;  cursor = cursor->sig;  }  }  **return** found; }  **template** <**typename** elemL> **void** Lista<elemL>::iniciar\_iterador(){  it = Primero; }  **template** <**typename** elemL> **void** Lista<elemL>::avanzar\_iterador(){  **if** (it){  it\_previo = it;  it = it->sig;  }  **else** {  it\_previo = it;  it = NULL;  } }  **template** <**typename** elemL> elemL Lista<elemL>::elemento\_iterador(){  **return** it->elemento; }  **template** <**typename** elemL> **bool** Lista<elemL>::final\_iterador(){  **return** (it == NULL); }  **template** <**typename** elemL> elemL Lista<elemL>::primer\_elemento(){  **return** **this**->Primero->elemento; }  **template** <**typename** elemL> elemL Lista<elemL>::obtener\_elemento\_it(){  elemL aux = it->elemento;  it = it->sig;  it\_previo->sig = it;  **return** aux; }  **template** <**typename** elemL> **void** Lista<elemL>::remover\_it\_nodo(){  nodo\_lista \*aux = it;  it = it->sig;  it\_previo->sig = it;  **delete** aux; }   **template** **class** **Lista**<unsigned int>; **template** **class** **Lista**<float>; **template** **class** **Lista**<string>; **template** **class** **Lista**<char>; **template** **class** **Lista**<bool>; **template** **class** **Lista**<Cliente>; |
| --- |

## Main.cpp

| **#include "lista.cpp"** **#include "fila.cpp"** **#include "Banco.h"**  **#include <iostream>**  **using** **namespace** std;  Banco \* bnc = **new** Banco(); **int** opcion = -1;  **void** **ingresar\_usuario**(){  system("cls");  */// Destinos: persona, banco o impuesto*  */// Operaciones: Retiro, Depósito, Transferencia o Pago*  **int** nro = rand() % 100+1;   Cliente nuevoC;  string nombre = "cliente ";  nombre+=std::to\_string(nro);  nuevoC.setNombre(nombre);  cout << nombre<< endl;  nuevoC.setEdad(nro);  nuevoC.setMonto(rand() % 10000+1);  *// TO DO generar ingresos random*  nuevoC.setDestino("persona");  nuevoC.setOperacion("retiro");  nuevoC.setEstado(true);  bnc->IngresarCliente(nuevoC);  cout << "Nuevo cliente ingresado" << endl;  system("pause"); };  **void** **abrir\_cola**(){  **if** (bnc->cantidad\_colas\_abiertas() < 2){  */\*\* No se realiza el pedido de los criterios para ahorrar tiempo y codigo  \*/*  string criterio = "retiro";  **bool** tieneCuenta = true;  bnc->Abrir\_Cola\_Criterio(criterio,tieneCuenta);  } **else** {  cout << "Cantidad maxima de colas alcanzadas" << endl;  }  system("pause"); };  **void** **proxClienteCola**(){  Cliente clAtendido;  **if** (bnc->cantidad\_colas\_abiertas() == 0){  clAtendido = bnc->Atender\_Prox\_Cliente(1);  } **else** {  **int** opcion\_cola = 1;  cout << "Seleccione cola para atender cliente" << endl;  cout << "1 - Cola general" << endl;  **if** (bnc->pr\_cola\_abierto()) cout << "2 - Cola con Criterio: " << bnc->pr\_cola\_getcrit() << endl;  **if** (bnc->sg\_cola\_abierto()) cout << "3 - Cola con Criterio: " << bnc->sg\_cola\_getcrit() << endl;  cout << "---> ";  cin >> opcion\_cola;  clAtendido = bnc->Atender\_Prox\_Cliente(opcion\_cola);   }  cout << "Se atendio al cliente: " << endl;  cout << clAtendido.getNombre() << endl;  cout << clAtendido.getEdad() << endl;  cout << clAtendido.getOperacion() << endl;  cout << clAtendido.getDestino() << endl;  cout << clAtendido.getMonto() << endl;  **if** (clAtendido.getEstado()) { cout << "Es cliente del banco" << endl; }  **else** { cout << "No es cliente del banco" << endl; }  system("pause"); }; **void** **cerrar\_cola**(){  **int** cola;  **if** ((!bnc->pr\_cola\_abierto()) && (!bnc->sg\_cola\_abierto())){  cout << "Ninguna Cola especila Abierta" << endl;  } **else** {  cout << "Seleccion Cola a cerrar" << endl;  **if** (bnc->pr\_cola\_abierto()) cout << "1 - Cola con criterio " << bnc->pr\_cola\_getcrit() << endl;  **if** (bnc->sg\_cola\_abierto()) cout << "2 - COla con criterio " << bnc->sg\_cola\_getcrit() << endl;  cout << "---> ";  cin >> cola;  **if** (bnc->Cerrar\_Cola\_Especial(cola)) cout << "Cola cerrada" << endl;  **else** cout << "La cola no se encuentra vacia" << endl;  system("pause");  } }; **void** **listar\_operaciones**(){  bnc->iniciar\_iterador();  **float** suma = 0;  **int** contador = 0;  **int** minimo, maximo;  cout << "Ingresar monto minimo: " << endl;  cin >> minimo;  cout << "Ingresar monto maximo: " << endl;  cin >> maximo;  **while** (**not** bnc->final\_iterador()) {  **if** ((bnc->elemento\_iterador().getMonto() >= minimo) && (bnc->elemento\_iterador().getMonto() <= maximo)) {  cout << bnc->elemento\_iterador().getNombre() << endl;  cout << bnc->elemento\_iterador().getEdad() << endl;  cout << bnc->elemento\_iterador().getOperacion() << endl;  cout << bnc->elemento\_iterador().getDestino() << endl;  cout << bnc->elemento\_iterador().getMonto() << endl;  **if** (bnc->elemento\_iterador().getEstado()) {cout << "Es cliente del banco" << endl;}  **else** {cout << "No es cliente del banco" << endl;}  suma += bnc->elemento\_iterador().getEdad();  contador++;  }  bnc->avanzar\_iterador();  }  cout << "El promedio de edad de los clientes del listado es " << (suma / contador) << endl;  system("pause"); };   **int** **main**(**int** argc, **char** \*argv[]) {  **bool** uso\_sistema = true;   **while**(uso\_sistema){  system("cls");  cout << "Sistema Banco" << endl;  cout <<"--Menu-" <<endl;  cout <<"1. Ingresar Cliente" <<endl;  cout <<"2. Atender Proximo Cliente" <<endl;  cout <<"3. Abrir Cola con criterio" <<endl;  cout <<"4. Cerrar Cola especial" <<endl;  cout <<"5. Listar Operacion por Monto" <<endl;  cout <<"0. Salir" <<endl;  cout << endl;  cout << "Ingrese Opcion:" << endl;  */// No se chequea si no es un nro el valor ingresado*  cin >> opcion;  **if** (opcion == 1) ingresar\_usuario();  **if** (opcion == 2) proxClienteCola();  **if** (opcion == 3) abrir\_cola();  **if** (opcion == 4) cerrar\_cola();  **if** (opcion == 5) listar\_operaciones();  **if** (opcion == 0){  uso\_sistema = false;  }  system("cls");  }   **return** 0; } |
| --- |