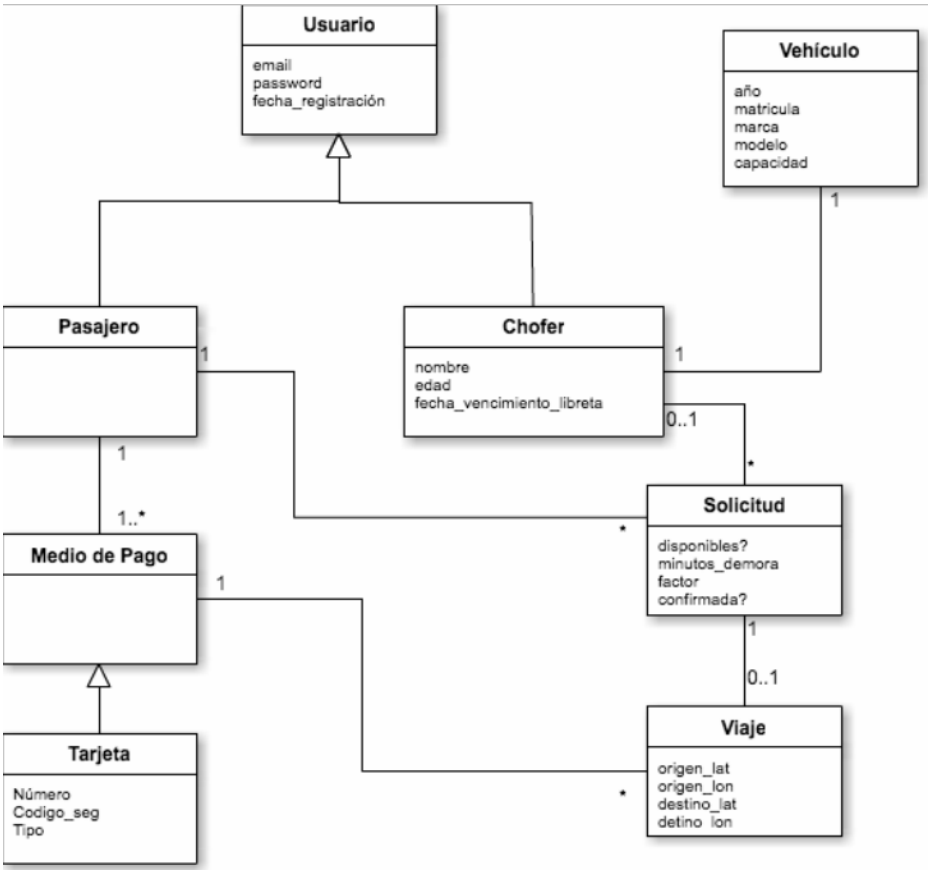
Programación 4

PARCIAL FINAL EDICIÓN 2016 - SOLUCIÓN

Problema 1 (30 puntos)

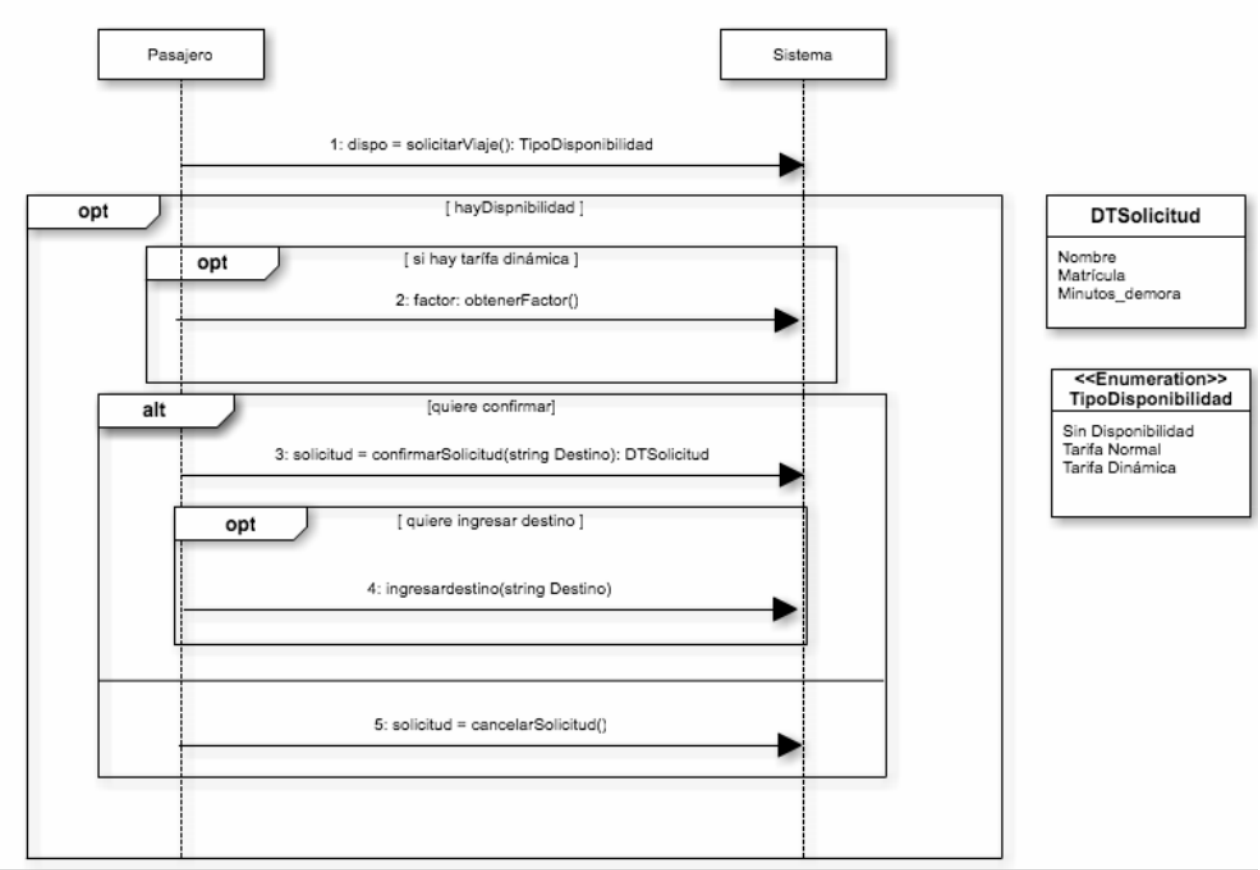
i.



Restricciones:

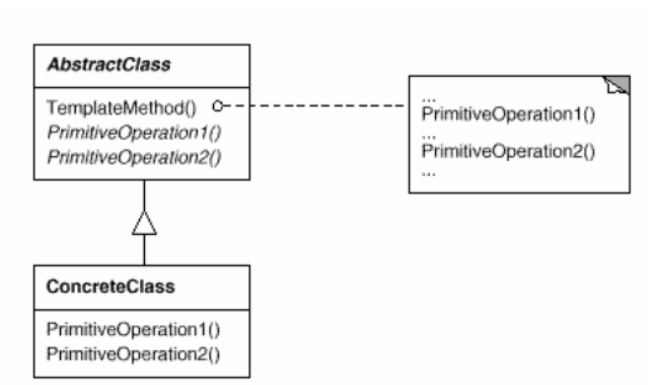
Unicidad de atributos: el mail identifica al Usuario; la matricula identifica al Vehículo.
Dominio de atributos: el año del Vehículo debe ser >= 2011.
Integridad circular: el medio de pago que paga el viaje debe ser del pasajero que solicitó ese viaje.
Atributos calculados: no hay.
Reglas de Negocio: no hay.

ii.


Pasajero
Sistema
1: dispo = solicitarViaje(): TipoDisponibilidad
opt
[hayDispnibilidad]
opt
[si hay tarifa dinámica]
2: factor: obtenerFactor()
alt
[quiere confirmar]
3: solicitud = confirmarSolicitud(string Destino): DTSolicitud
opt
[quiere ingresar destino]
4: ingresardestino(string Destino)
5: solicitud = cancelarSolicitud()
DTSolicitud
Nombre
Matrícula
Minutos_demora
<<Enumeration>>
TipoDisponibilidad
Sin Disponibilidad
Tarifa Normal
Tarifa Dinámica

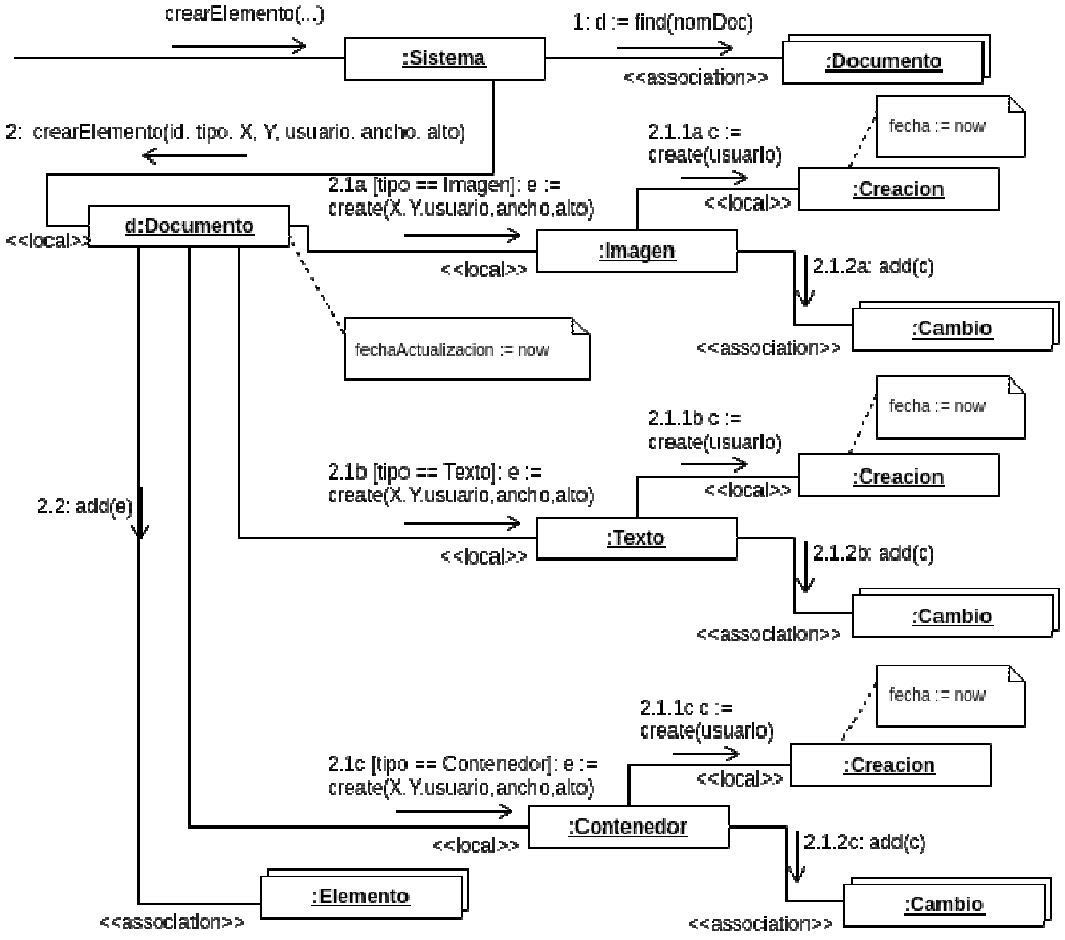
Problema 2 (35 puntos)

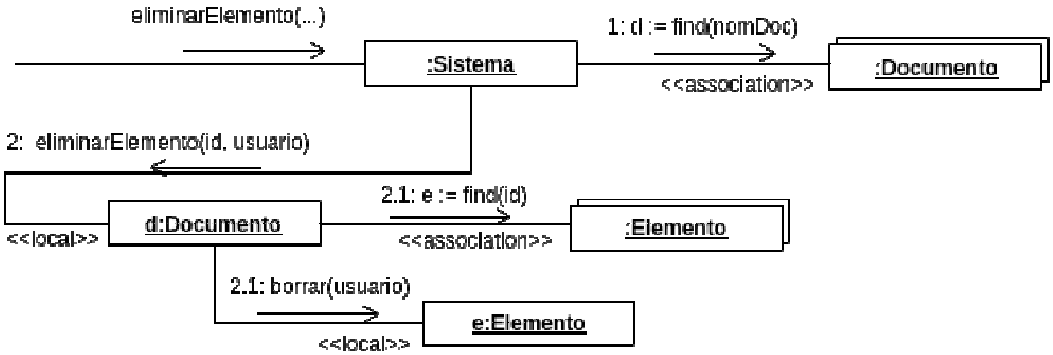
Parte a)

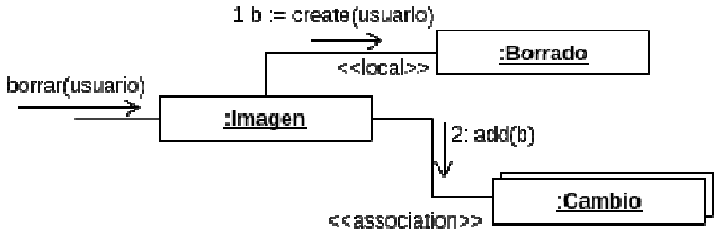


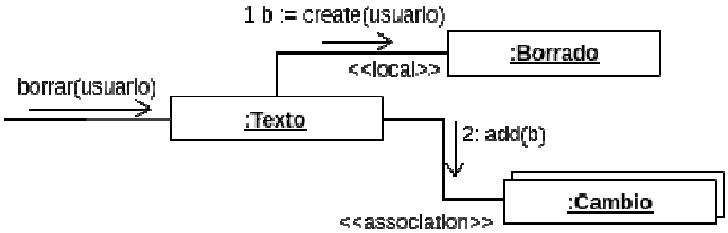
Parte b)

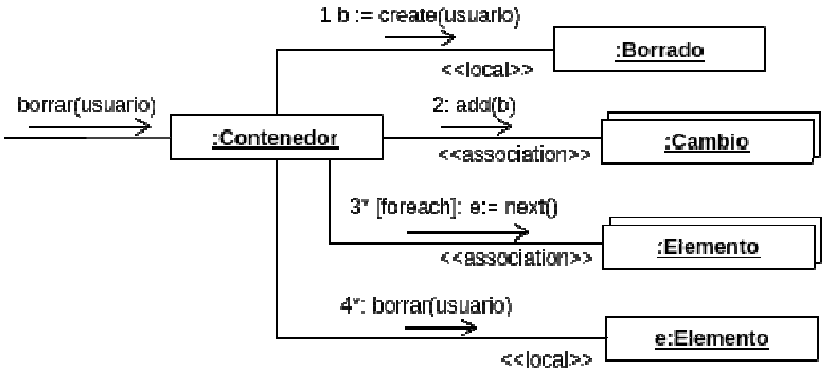
i.



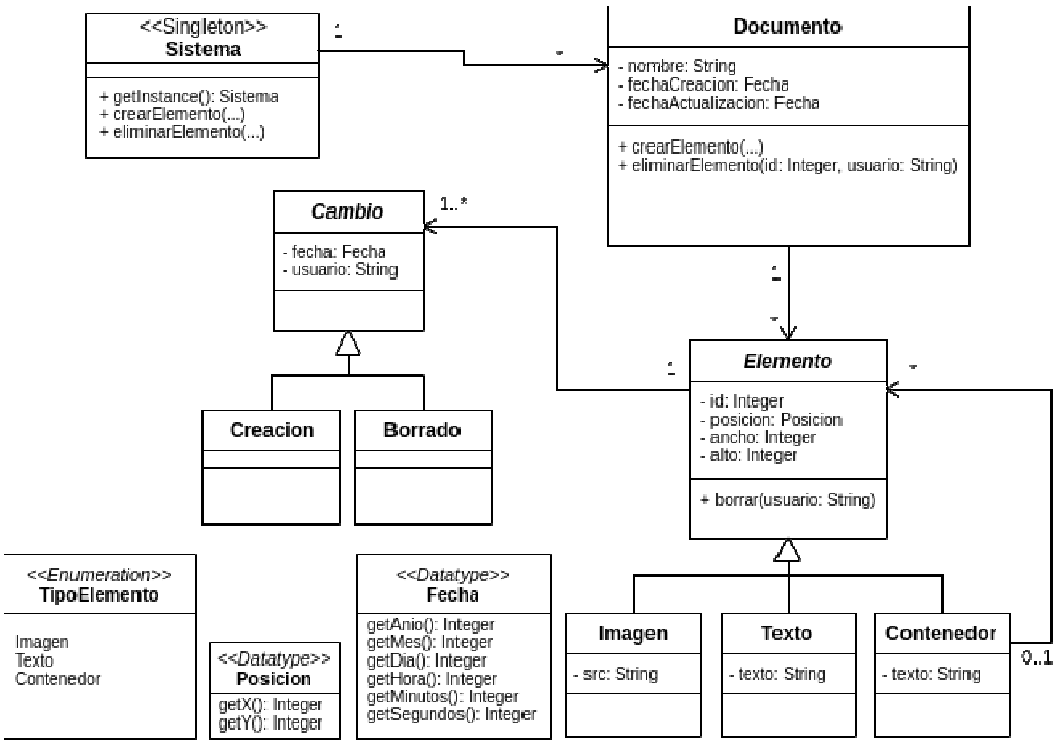

eliminarElemento(...)
:Sistema
1: d := find(nomDoc)
<<association>>
:Documento
2: eliminarElemento(id, usuario)
d:Documento
<<local>>
2.1: e := find(id)
<<association>>
:Elemento
2.1: borrar(usuario)
<<local>>
e:Elemento


1 b := create(usuario)
<<local>>
:Borrado
borrar(usuario)
:Imagen
2: add(b)
:Cambio
<<association>>


1 b := create(usuario)
<<local>>
:Borrado
borrar(usuario)
:Texto
2: add(b)
:Cambio
<<association>>


1 b := create(usuario)
<<local>>
:Borrado
borrar(usuario)
:Contenedor
2: add(b)
<<association>>
:Cambio
3* [foreach]: e:= next()
<<association>>
:Elemento
4*: borrar(usuario)
<<local>>
e:Elemento

ii.


<<Singleton>>
Sistema
+ getInstance(): Sistema
+ crearElemento(...)
+ eliminarElemento(...)
Documento
- nombre: String
- fechaCreacion: Fecha
- fechaActualizacion: Fecha
+ crearElemento(...)
+ eliminarElemento(id: Integer, usuario: String)
Cambio
- fecha: Fecha
- usuario: String
1..*
Elemento
- id: Integer
- posicion: Posicion
- ancho: Integer
- alto: Integer
+ borrar(usuario: String)
Creacion
Borrado
<<Enumeration>>
TipoElemento
Imagen
Texto
Contenedor
<<Datatype>>
Posicion
getX(): Integer
getY(): Integer
<<Datatype>>
Fecha
getAnio(): Integer
getMes(): Integer
getDia(): Integer
getHora(): Integer
getMinutos(): Integer
getSegundos(): Integer
Imagen
- src: String
Texto
- texto: String
Contenedor
- texto: String
0..1

Problema 3 (35 puntos)

i.

```
class Cancion {
public:
    Cancion(int id, string nombre, string autor, string album, string
bit_rate, char* contenido);

    int getId();
    string getNombre();
    string getAutor();
    string getAlbum();
    string getBitRate();
    char* getContenido();

    virtual ~Cancion();
private:
    int id;
    string nombre;
    string autor;
    string album;
    string bit_rate;
    char* contenido;
};

Cancion::Cancion(int id, string nombre, string autor, string album,
string bit_rate, char* contenido) {
    this->id = id;
    this->nombre = nombre;
    this->autor = autor;
    this->album = album;
    this->bit_rate = bit_rate;
    this->contenido = contenido;
}

Cancion::~Cancion() {}

class AdmOperacionalUsuario {
public:
    AdmOperacionalUsuario();

    list<DtCancion>    ordernar_favoritas(string    email,    EnumOrden
criterio);
    char* descargar_cancion(string email, int id);
    void agregar_favorita(string email, int id);

    virtual ~AdmOperacionalUsuario();
private:
    map<string, Usuario*> usuarios;
    map<int, Cancion*> canciones;
};

AdmOperacionalUsuario::AdmOperacionalUsuario() {
}

list<DtCancion>        AdmOperacionalUsuario::ordernar_favoritas(string
email, EnumOrden criterio) {
    list<DtCancion> ldc;
```

```
    list<Cancion*>      canciones      =      this->usuarios[email]-
>ordenar_favoritas(criterio);

    for(list<Cancion*>::iterator   it   =   canciones.begin();   it   !=
canciones.end(); it++) {
        Cancion* c = *it;
        DtCancion  dc(c->getId(),  c->getNombre(),  c->getAutor(),  c-
>getAlbum(), c->getBitRate(), c->getContenido());

        ldc.push_back(dc);
    }

    return ldc;
}

char* AdmOperacionalUsuario::descargar_cancion(string email, int id) {
    return this->usuarios[email]->descargar_cancion(id);
}

void AdmOperacionalUsuario::agregar_favorita(string email, int id) {
    this->usuarios[email]->agregar_favorita(this->canciones[id]);
}

AdmOperacionalUsuario::~AdmOperacionalUsuario() {
}

class Usuario {
public:
    Usuario(string email, string nombre, string password);

    virtual list<Cancion*> ordenar_favoritas(EnumOrden criterio) = 0;
    char* descargar_cancion(int id);
    void agregar_favorita(Cancion* cancion);

    map<int, Cancion*> getFavoritas();

    virtual ~Usuario();
private:
    string email;
    string nombre;
    string password;
    map<int, Cancion*> favoritas;
    Cancion* enReproduccion;
};

Usuario::Usuario(string email, string nombre, string password) {
    this->email = email;
    this->nombre = nombre;
    this->password = password;
    enReproduccion = NULL;
}

char* Usuario::descargar_cancion(int id) {
    if (this->favoritas.find(id) == this->favoritas.end()) {
        throw invalid_argument("El usuario no tiene la cancion en sus
favoritas");
    }

    return this->favoritas[id]->getContenido();
}
```

```
void Usuario::agregar_favorita(Cancion* cancion) {
    this->favoritas[cancion->getId()] = cancion;
}

map<int, Cancion*> Usuario::getFavoritas() {
    return this->favoritas;
}

Usuario::~Usuario() {}

class Gratis : public Usuario {
public:
    Gratis(string email, string nombre, string password);

    virtual list<Cancion*> ordenar_favoritas(EnumOrden criterio);

    virtual ~Gratis();
private:

};

Gratis::Gratis(string email, string nombre, string password) :
Usuario(email, nombre, password) {
}

list<Cancion*> Gratis::ordenar_favoritas(EnumOrden criterio) {
    throw std::invalid_argument("No puede ordenar favoritos");
}

Gratis::~Gratis() {}

class Premium : public Usuario {
public:
    Premium(string email, string nombre, string password);

    virtual list<Cancion*> ordenar_favoritas(EnumOrden criterio);

    virtual ~Premium();
private:

};

Premium::Premium(string email, string nombre, string password) :
Usuario(email, nombre, password) {
}

list<Cancion*> Premium::ordenar_favoritas(EnumOrden criterio) {

    map<int, Cancion*> favoritas = this->getFavoritas();
    list<Cancion*> canciones;

    for(map<int, Cancion*>::iterator it = favoritas.begin(); it !=
favoritas.end(); it++) {
        canciones.push_back(it->second);
    }
    return       CancionUtils::getInstance()->ordenar_lista(canciones,
criterio);

}

Premium::~Premium() {}
```

ii.

```
class CancionUtils {
public:
    static CancionUtils* getInstance();
    list<Cancion*>  ordenar_lista(list<Cancion*>  canciones,  EnumOrden
criterio);
    virtual ~CancionUtils();
private:
    CancionUtils();
    static CancionUtils* instance;
};

CancionUtils *CancionUtils::instance = NULL;

CancionUtils::CancionUtils() {}

CancionUtils* CancionUtils::getInstance() {
    if (instance==NULL)
        instance = new CancionUtils();
    return instance;
}

list<Cancion*>  CancionUtils::ordenar_lista(list<Cancion*>  canciones,
EnumOrden criterio) {
}

CancionUtils::~CancionUtils() {}
```

iii.

```
class DtCancion {
public:
    DtCancion(int  id,  string  nombre,  string  autor,  string  album,
string bit_rate, char* contenido);

    int getId();
    string getNombre();
    string getAutor();
    string getAlbum();
    string getBitRate();
    char* getContenido();
    virtual ~DtCancion();
private:
    int id;
    string nombre;
string autor;
    string album;
    string bit_rate;
    char* contenido;
};

DtCancion::DtCancion(int  id,  string  nombre,  string  autor,  string
album, string bit_rate, char* contenido) {
    this->id = id;
    this->nombre = nombre;
    this->autor = autor;
    this->album = album;
    this->bit_rate = bit_rate;
```

```
    this->contenido = contenido;
}

int DtCancion::getId() {
    return this->id;
}

string DtCancion::getNombre() {
    return this->nombre;
}

string DtCancion::getAutor() {
    return this->autor;
}

string DtCancion::getAlbum() {
    return this->album;
}

string DtCancion::getBitRate() {
    return this->bit_rate;
}

char* DtCancion::getContenido() {
    return this->contenido;
}

DtCancion::~DtCancion() {}
```

iv.

```
enum EnumOrden { Nombre, Autor };
```