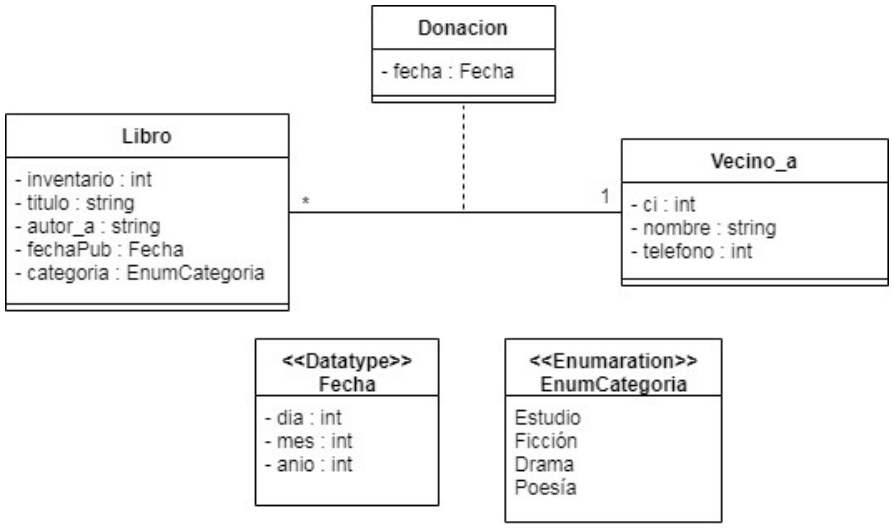
Programación 4

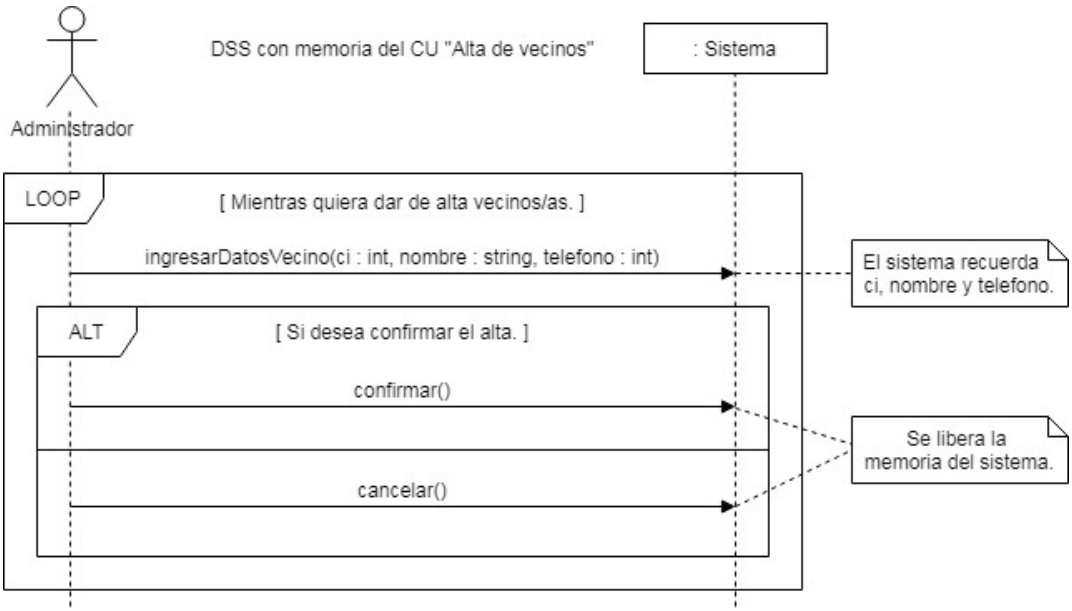
SOLUCIÓN EXAMEN FEBRERO 2021

<u>Problema 1</u> (35 puntos)

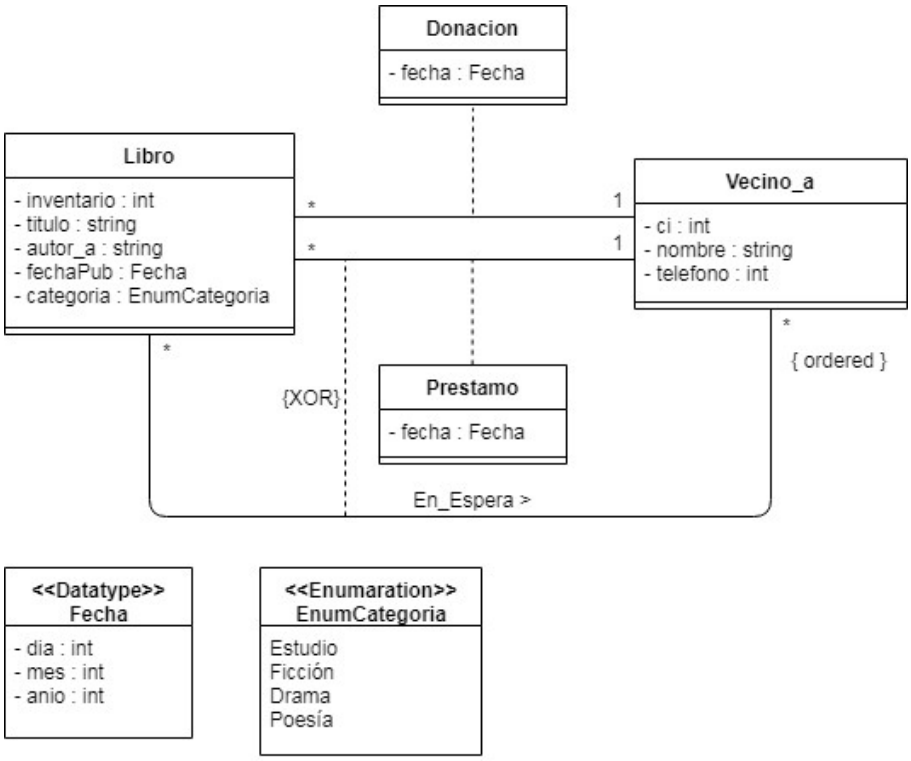
a)



b)



c)



Restricciones:

- El atributo inventario identifica al Libro.
- El atributo ci identifica al Vecino_a.
- La fecha de donación de un Libro tiene que ser mayor o igual a su fecha de publicación.
- Las fechas de los préstamos asociados un Libro tienen que ser mayores o iguales a su fecha de donación.

d)

listarLibros:

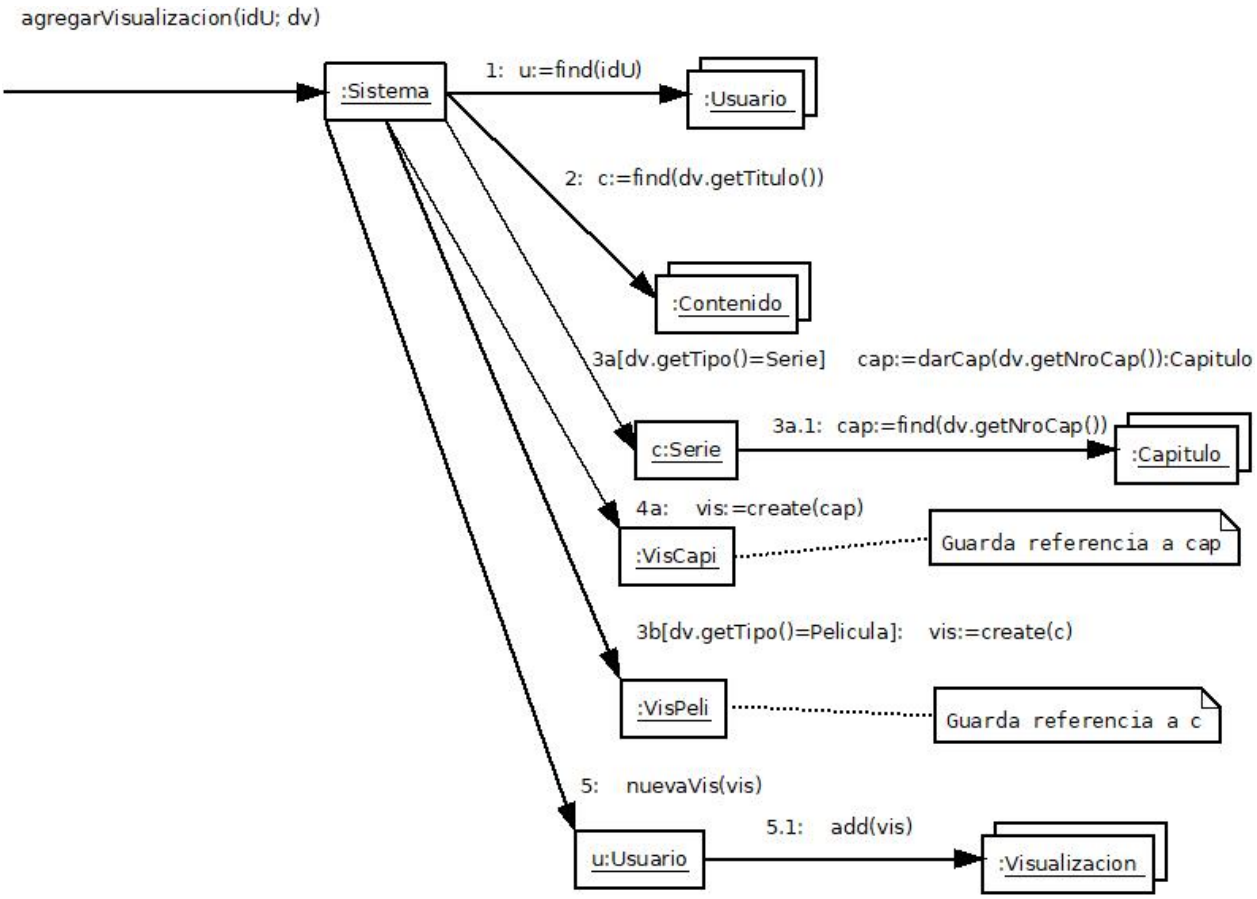
- PRE: -
- POST: Se devuelve un conjunto de datavalues de DTLibro correspondientes a las instancias de Libro del sistema.

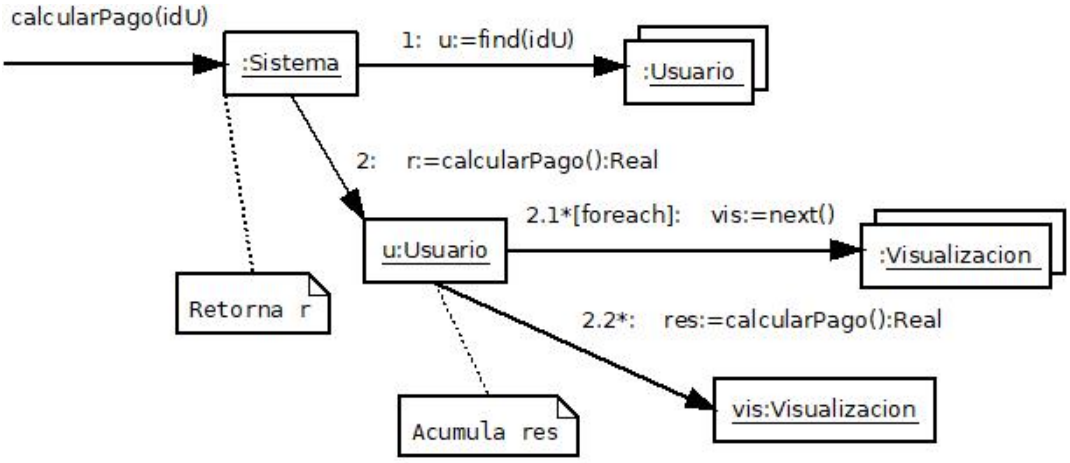
solicitarLibro:

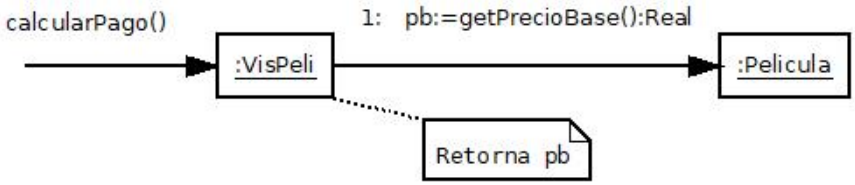
- PRE1: Existe en el sistema una instancia de Vecino_a "v" recordada.
- PRE2: Existe en el sistema una instancia de Libro "l" con atributo inventario igual al parámetro inventario de esta operación.
- POST1: Si "l" no está asociada a ninguna instancia de prestamo entonces se crea una instancia de préstamo "p" colocando en el atributo fecha el valor de la fecha del Sistema. Además, se crean un link entre "l" y "p", y uno entre "p" y "v". En caso contrario se agrega una instancia de la asociación "En_espera" que relacione a "l" y "v".
- POST2: Si el libro se encontraba disponible se devuelve el mensaje "Préstamo otorgado.", y sino el mensaje "Libro no disponible. Se le agrego a la lista de espera.".

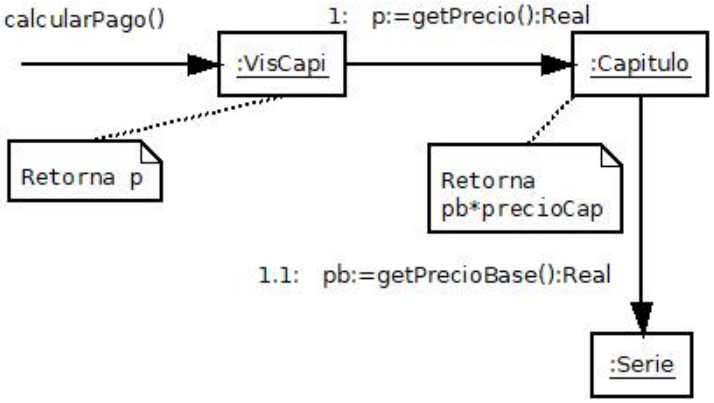
Problema 2 (30 puntos)

a)

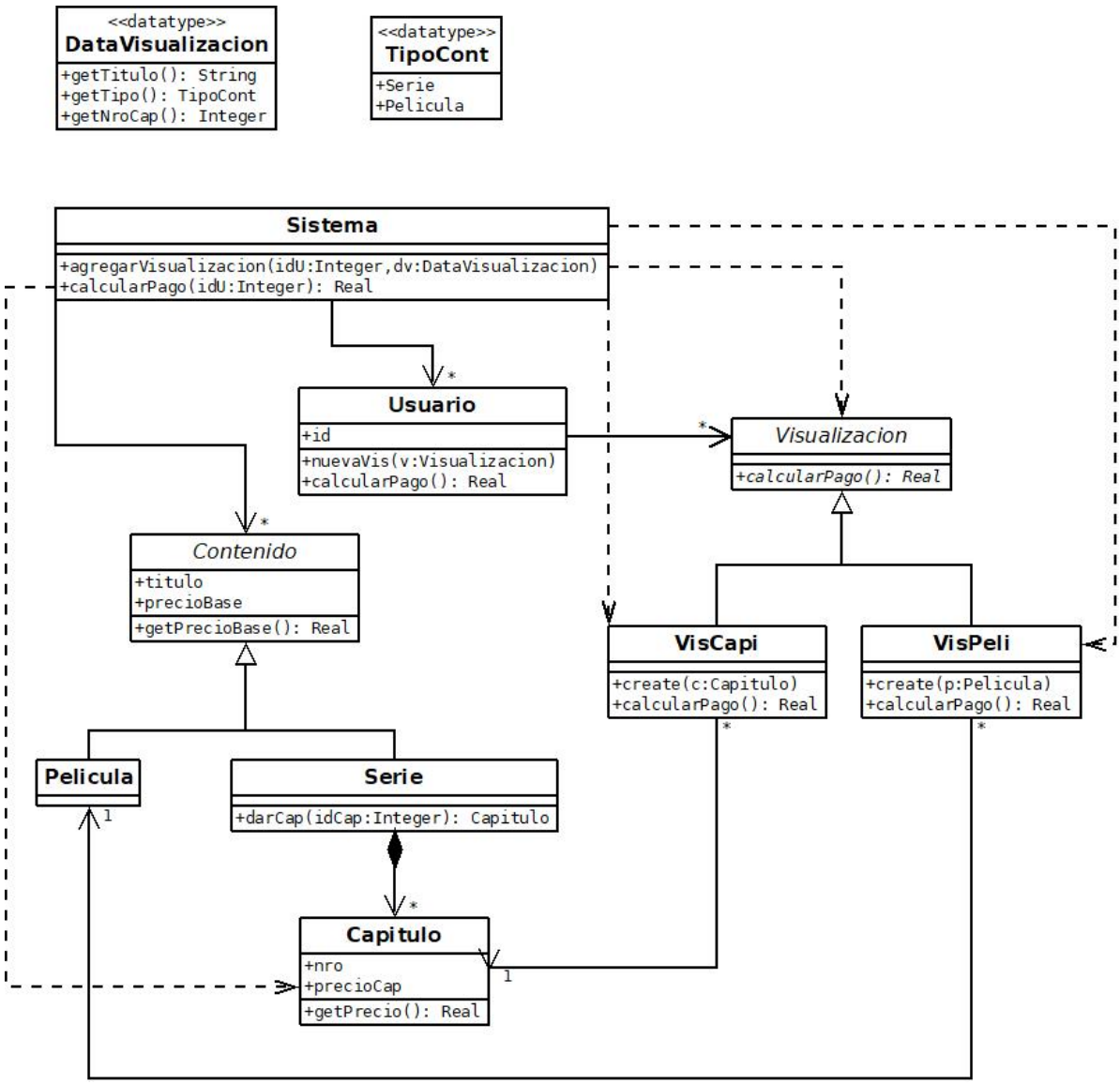



calcularPago(idU)
:Sistema
1: u:=find(idU)
:Usuario
2: r:=calcularPago():Real
u:Usuario
2.1*[foreach]: vis:=next()
:Visualizacion
Retorna r
2.2*: res:=calcularPago():Real
Acumula res
vis:Visualizacion


calcularPago()
1: pb:=getPrecioBase():Real
:VisPeli
:Pelicula
Retorna pb


calcularPago()
1: p:=getPrecio():Real
:VisCapi
:Capitulo
Retorna p
Retorna
pb*precioCap
1.1: pb:=getPrecioBase():Real
:Serie

b)


<<datatype>>
DataVisualizacion
+getTitulo(): String
+getTipo(): TipoCont
+getNroCap(): Integer
<<datatype>>
TipoCont
+Serie
+Pelicula
Sistema
+agregarVisualizacion(idU:Integer,dv:DataVisualizacion)
+calcularPago(idU:Integer): Real
Usuario
+id
+nuevaVis(v:Visualizacion)
+calcularPago(): Real
Visualizacion
+calcularPago(): Real
Contenido
+titulo
+precioBase
+getPrecioBase(): Real
VisCapi
+create(c:Capitulo)
+calcularPago(): Real
VisPeli
+create(p:Pelicula)
+calcularPago(): Real
Pelicula
Serie
+darCap(idCap:Integer): Capitulo
Capitulo
+nro
+precioCap
+getPrecio(): Real
*
1

Problema 3 (35 puntos)

a)

Singleton:

- Clase ControladorDeQuejas.
- Problema tipo: Asegurar que una clase tenga una sola instancia y proveer un acceso global a ella.

State:

- Roles: Queja es el Contexto, Etapa es el Estado, Enviada y Procesada son los Estados Concretos.
- Problema tipo: Permitir que un objeto varíe su comportamiento cuando su estado interno cambie. El objeto parecerá haber cambiado de clase.

b)

```
//Etapa.h

class Etapa {
public:
      virtual Etapa* darSiguienteEtapa(string texto) = 0;
      virtual string verComentario() = 0;
      Etapa(){};
      virtual ~Etapa(){}
}

//Enviada.h
class Enviada: public Etapa {
public:
      Enviada()
      Etapa* modificarEtapa(string txt);
      string verComentario();
}

//Enviada.cpp
Enviada::Enviada(){}

Etapa* Enviada::darSiguienteEtapa(string txt){
      Etapa* nueva = new Procesada(txt);
      return nueva;
}

string Etapa::verComentario(){
      return "Enviada: mensaje en breve";
}


//Queja.h
class Queja{
private:
      int id;
      string texto;
      Etapa* etapa;
public:
```

```
      Queja(int id, string texto);
       ~Queja();
      void cambiarEtapa(string);
      DtQueja getDT();
}

//Queja.cpp

Queja::Queja(int id, string texto){
      this->id = id;
      this->texto = texto;
      this->etapa = new Enviada();
}

Queja::~Queja() {

      delete etapa;
}

void Queja::cambiarEtapa(string texto){
      Etapa* temp = this->etapa;
      this->etapa = this->etapa->darSiguienteEtapa(texto);
      delete temp;
}


//ControladorQuejas.h
class ControladorQuejas{
private:
      ControladorQuejas();
      static ControladorQuejas* instancia;
      map<int,Usuario*> usuarios;
      map<int,Queja*> quejas;
public:
      static ControladorQuejas* getInstance();

      void crearUsuario(int id, string nombre, DtFecha fechita);
      void eliminarUsuario(int id);
      void  modificarQueja(int  idQueja,  string  Comentario,  set<int>
afectados);
}

//ControladorQuejas.cpp

ControladorQuejas* ControladorQuejas::instancia = NULL;

ControladorQuejas* ControladorQuejas::getInstance(){
      if (instancia == NULL)
            instancia = new ControladorQuejas();
      return instancia;
}

void  ControladorQuejas::crearUsuario(int  id,  string  nombre,  DtFecha
fechita){
      Usuario* u = new Usuario(id, nombre, fechita);
      usuarios[id] = u;
}

void ControladorQuejas::eliminarUsuario(int id){
      Usuario* u = usuarios[id];
      if(u!= NULL){
```

```
            usuarios.erase(id);
            delete u;
      }
}

void ControladorQuejas::modificarQueja(int id, string comentario,
set<int> idAfectados){
      Queja* q = quejas[id];
      if (q!=NULL){
            q->cambiarEtapa(comentario);
            DtQueja dt = q->getDT();

            std::set<int>::iterator it;
            for (it = idAfectados.begin(); it != idAfectados.end();
++it) {
                  Usuario* u = usuarios[*it];
                  u->actualizarQueja(dt);
            }
      }
}
```