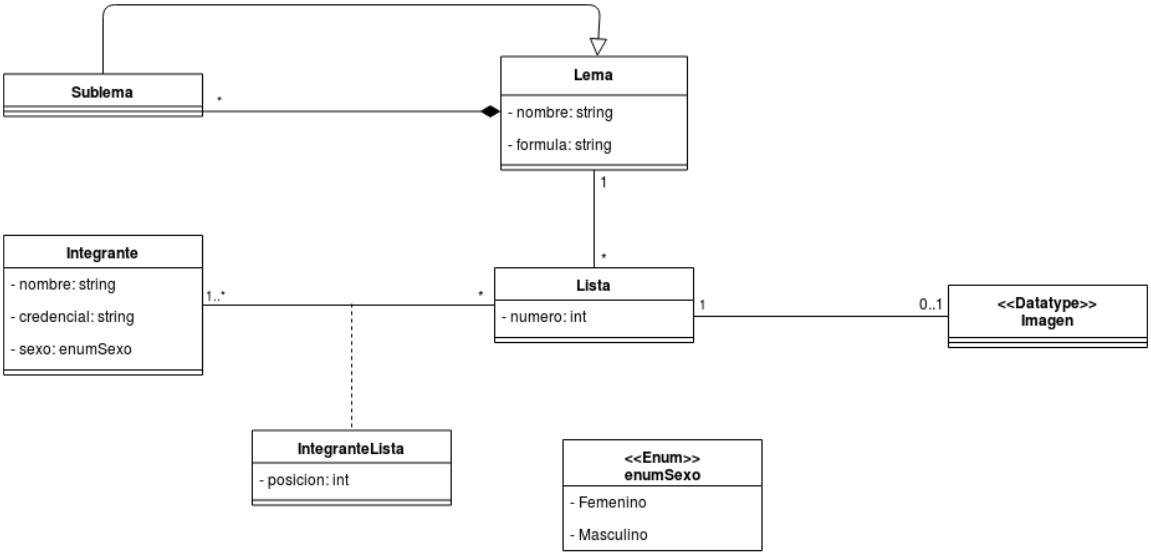
Programación 4

SOLUCION EXAMEN JULIO 2019

Problema 1

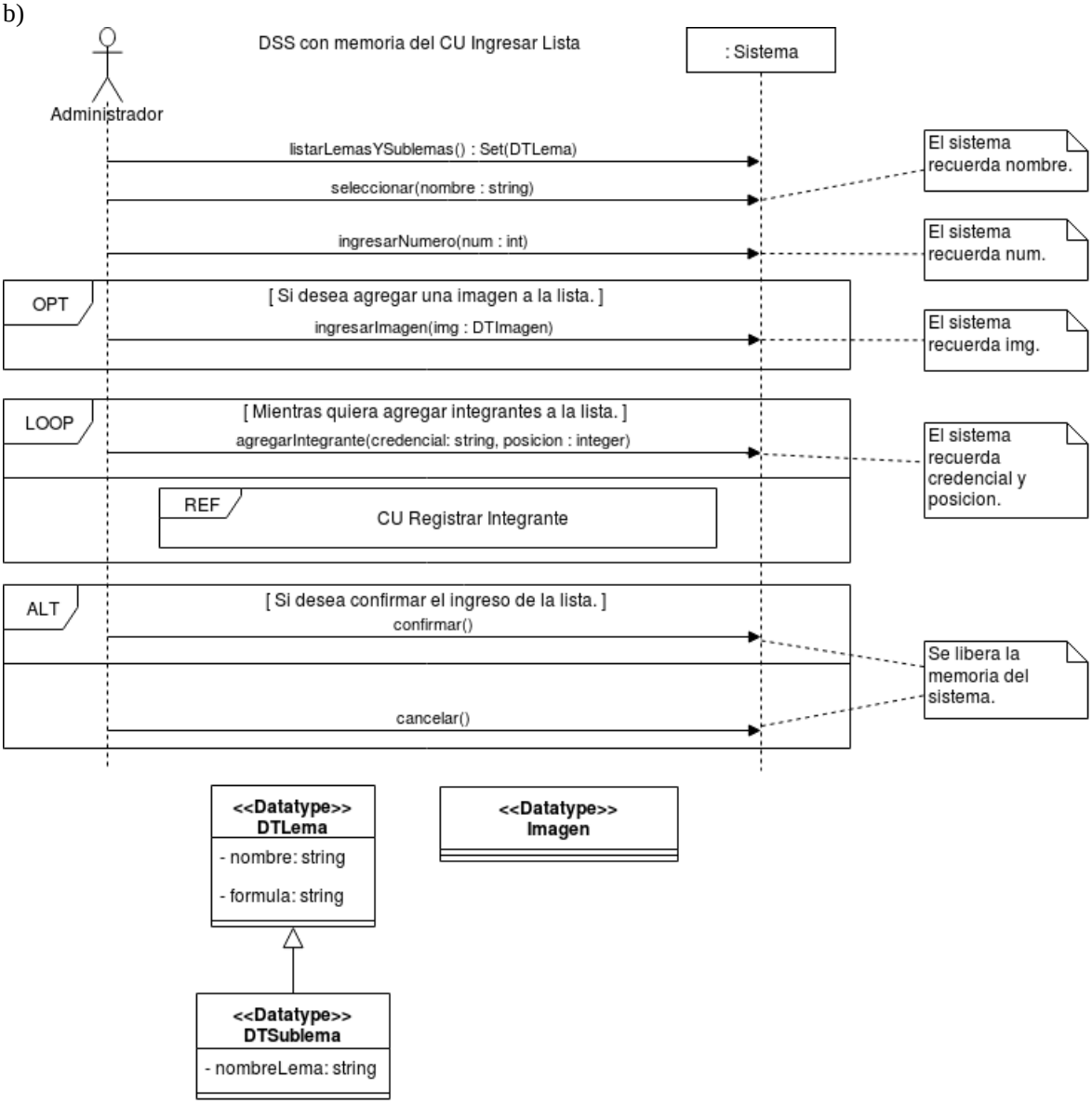
a)



Restricciones:

- El atributo nombre identifica al Lema.
- El atributo numero identifica a la Lista.
- El atributo credencial identifica al/ a la Integrante.
- Asociado a una Lista existen N instancias de IntegranteLista cuyos atributos posición conforman una secuencia de enteros de que va de 1 hasta N.

b)



c)

listarLemasYSublemas:
PRE: -
POST: Se devuelve un conjunto de datavalues de DTLema y DTSublema correspondientes a las instancias de Lema y Sublema del sistema.

seleccionar:
PRE: Existe en el sistema una instancia de Lema o Sublema con atributo nombre igual al parámetro de la operación.
POST: Se recuerda el parámetro nombre.

ingresarNumero:
PRE: -
POST: Se recuerda el parámetro num.

ingresarImagen:
PRE: -
POST: Se recuerda el parámetro img.

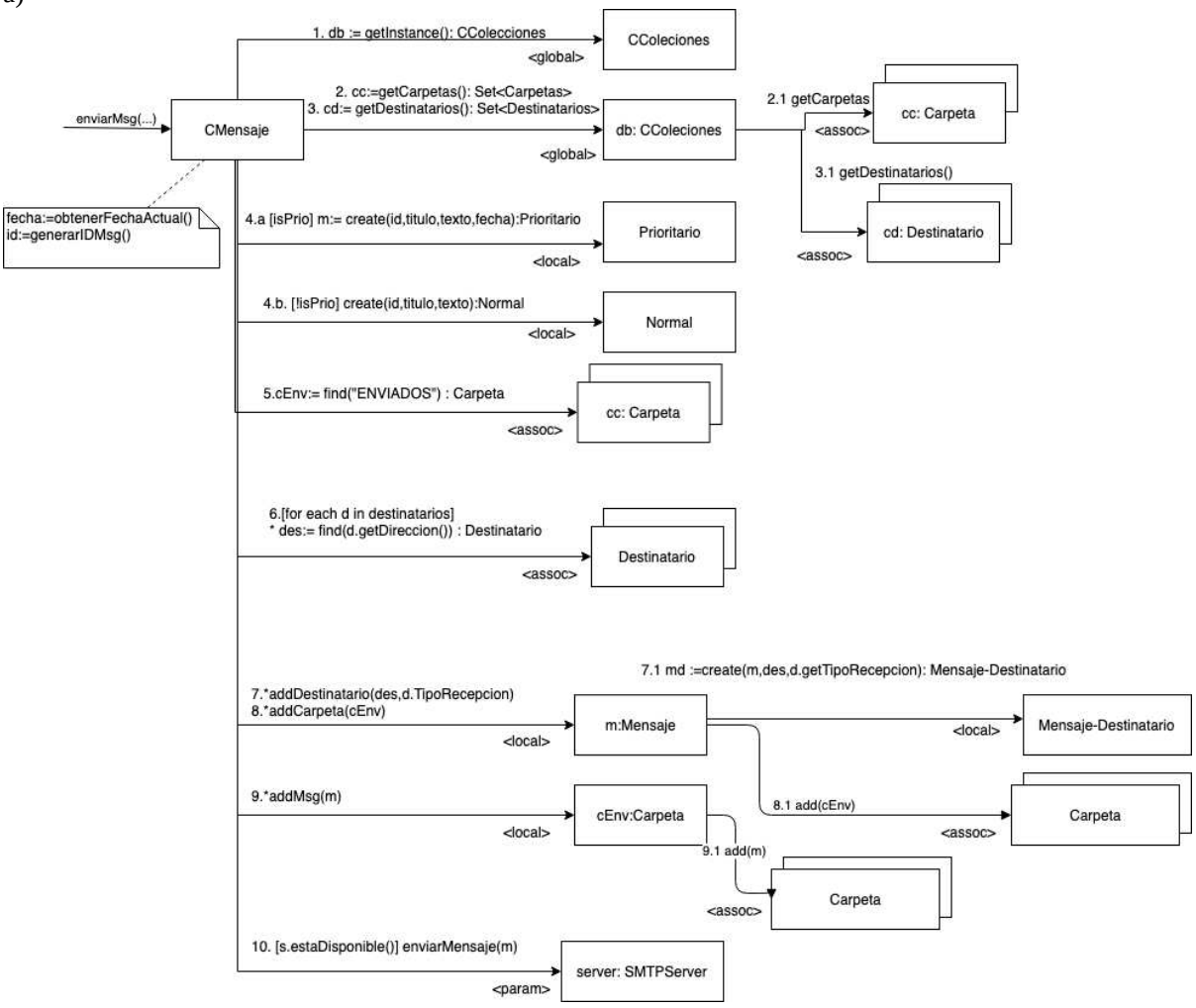
agregarIntegrante:
PRE: -
POST: Se recuerdan los pares credencial-posicion.

confirmar:
PRE: El sistema recuerda “nombre”, “num” y opcionalmente “img”. Además se recuerdan los pares “credencial-posicion” de los integrantes que se desean agregar.
POST: Se crea en el sistema una instancia “lista” de Lista con atributo numero con valor “num” y se le asocia la “img” si corresponde. Se asocia “lista” a la instancia de Lema con atributo nombre con valor “nombre”. Finalmente para cada par “credencial-posicion”: se asocia “lista” a la instancia “integrante” de Integrante con atributo credencial con valor “credencial”, se crea una instancia “intList” de IntegranteLista con atributo posicion con valor “posicion”, y se asocia a “intList” tanto a “lista” como a “integrante”.
POST2: Se libera la memoria del sistema.

cancelar:
PRE: -
POST: Se libera la memoria del sistema.

Problema 2

a)



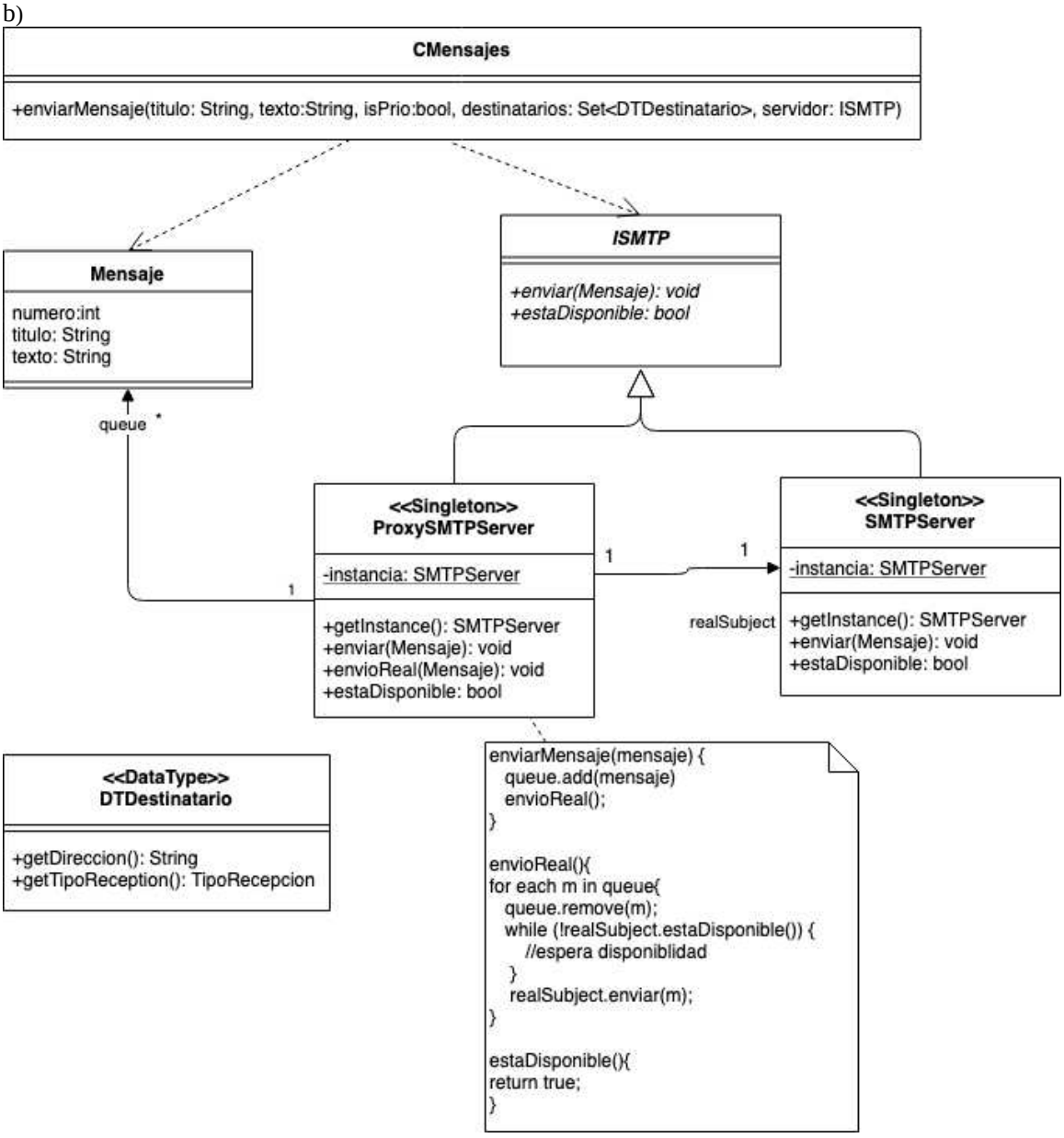
b)

Se implementa una solución basada en Proxy donde CMensajes cumple el rol de contexto pero sin asociarse sino que recibiendo el subject por parámetro.

Correspondencia de Roles:
- CMensajes: Context
- ISMTP: Subject
- SMTPServer: RealSubject
- ProxySMTPServer: Proxy

De este modo el chequeo de disponibilidad en el Proxy siempre retorna true, encolando los mensajes hasta que el envío pueda ser realizado una vez que el RealSubject indique disponibilidad, como se describe en los pseudocódigos de las operaciones.

b)


CMensajes
+enviarMensaje(titulo: String, texto:String, isPrio:bool, destinatarios: Set<DTDestinatario>, servidor: ISMTP)
Mensaje
numero:int
titulo: String
texto: String
ISMTP
+enviar(Mensaje): void
+estaDisponible: bool
queue
*
1
<<Singleton>>
ProxySMTPServer
-instancia: SMTPServer
+getInstance(): SMTPServer
+enviar(Mensaje): void
+envioReal(Mensaje): void
+estaDisponible: bool
1
1
realSubject
<<Singleton>>
SMTPServer
-instancia: SMTPServer
+getInstance(): SMTPServer
+enviar(Mensaje): void
+estaDisponible: bool
<<DataType>>
DTDestinatario
+getDireccion(): String
+getTipoReception(): TipoRecepcion
enviarMensaje(mensaje) {
queue.add(mensaje)
envioReal();
}
envioReal(){
for each m in queue{
queue.remove(m);
while (!realSubject.estaDisponible()) {
//espera disponiblidad
}
realSubject.enviar(m);
}
estaDisponible(){
return true;
}

Problema 3

IObserver.hpp

```
class IObserver {
	public:
		virtual void notificar(DtNotificacion notificacion) = 0;
}
```

Estado.hpp

```
class Estado {
	public:
		virtual Estado *darSiguienteEstado() = 0;
		virtual TipoEstado getTipoEstado() = 0;
}
```

TorreControl.hpp

```
class TorreControl: public IObserver {
	private:
		set<DtNotificacion> notificaciones;
	public:
		TorreControl();
		~TorreControl();
		void notificar(DtNotificacion notificacion);
}
```

TorreControl.cpp

```
TorreControl::TorreControl(){}

TorreControl::~TorreControl(){}

void TorreControl::notificar(DtNotificacion notificacion){
	notificaciones.insert(notificacion);
}
```

Repartidor.hpp

```
class Repartidor {
	private:
		int id;
		int bateria;
		int capacidad;
		set<IObserver*> observadores;
		set<Producto*> productos;
		Estado* estado;
		DtFecha obtenerFecha();
		DtNotificacion generarNotificacion();
		void notificar(DtNotificacion notif);
	protected:
		Repartidor(int id, bateria int, int capacidad);
	public:
		virtual ~Repartidor();
		void comenzarReparto(set<Producto*>);
		void finalizarReparto();
		void agregarObservador(IObserver *);
		void removerObservador(IObserver *);
}
```

```
Repartidor.cpp
--------------

DtNotificacion Repartidor::generarNotificacion(){
      return         DtNotificacion(id,         estado->getTipoEstado(),
obtenerFecha());
}

Repartidor::Repartidor(int id, bateria int, int capacidad){
      this->id = id;
      this->bateria = bateria;
      this->capacidad = capacidad;
      this->estado = new Disponible();
}

Repartidor::~Repartidor(){
      if (estado!=NULL){
            delete estado;
      }
      for(set<Producto*>::iterator
iter=productos.begin();iter!=productos.end();++iter){
            delete (*iter);
      }
}

void Repartidor::comenzarReparto(set<Producto*> productos){
      this->productos = productos;
      Estado* nuevo = estado->darSiguienteEstado();
      delete estado;
      estado = nuevo;
      notificar(generarNotificacion());
}

void Repartidor::finalizarReparto(){
      for(set<Producto*>::iterator
iter=productos.begin();iter!=productos.end();++iter){
            delete (*iter);
      }
      Estado* nuevo = estado->darSiguienteEstado();
      delete estado;
      estado = nuevo;
      notificar(generarNotificacion());

}

void Repartidor::notificar(DtNotificacion notif){
      for(set<IObserver*>::iterator
iter=observadores.begin();iter!=observadores.end();++iter){
            (*iter)->notificar(notif);
      }
}

void Repartidor::agregarObservador(IObserver* obs){
      observadores.insert(obs);
}

void Repartidor::removerObservador(IObserver* obs){
      observadores.erase(obs);
}
```

```
Drone.hpp
-------------
class Drone: public Repartidor {
      private:
            int cantHelices;
      public:
            Drone(int id, bateria int, int capacidad, int cantHelices);
            ~Drone();
}


Drone.cpp
-------------
Drone::Drone(int    id,    bateria    int,    int    capacidad,    int
cantHelices):Repartidor(id,bateria,capacidad){
      this->cantHelices=cantHelices;
}

Drone::~Drone(){}


Disponible.hpp
-------------
class Disponible: public Estado {
      public:
            Disponible();
            ~Disponible();
            Estado *darSiguienteEstado();
            TipoEstado getTipoEstado();
}


Disponible.cpp
-------------
Estado *Disponible::darSiguienteEstado(){
      return new Repartiendo();
}

TipoEstado Disponible::getTipoEstado(){
      enum TipoEstado tipo = Disponible;
      return tipo;
}

Disponible::Disponible(){}

Disponible::~Disponible(){}
```