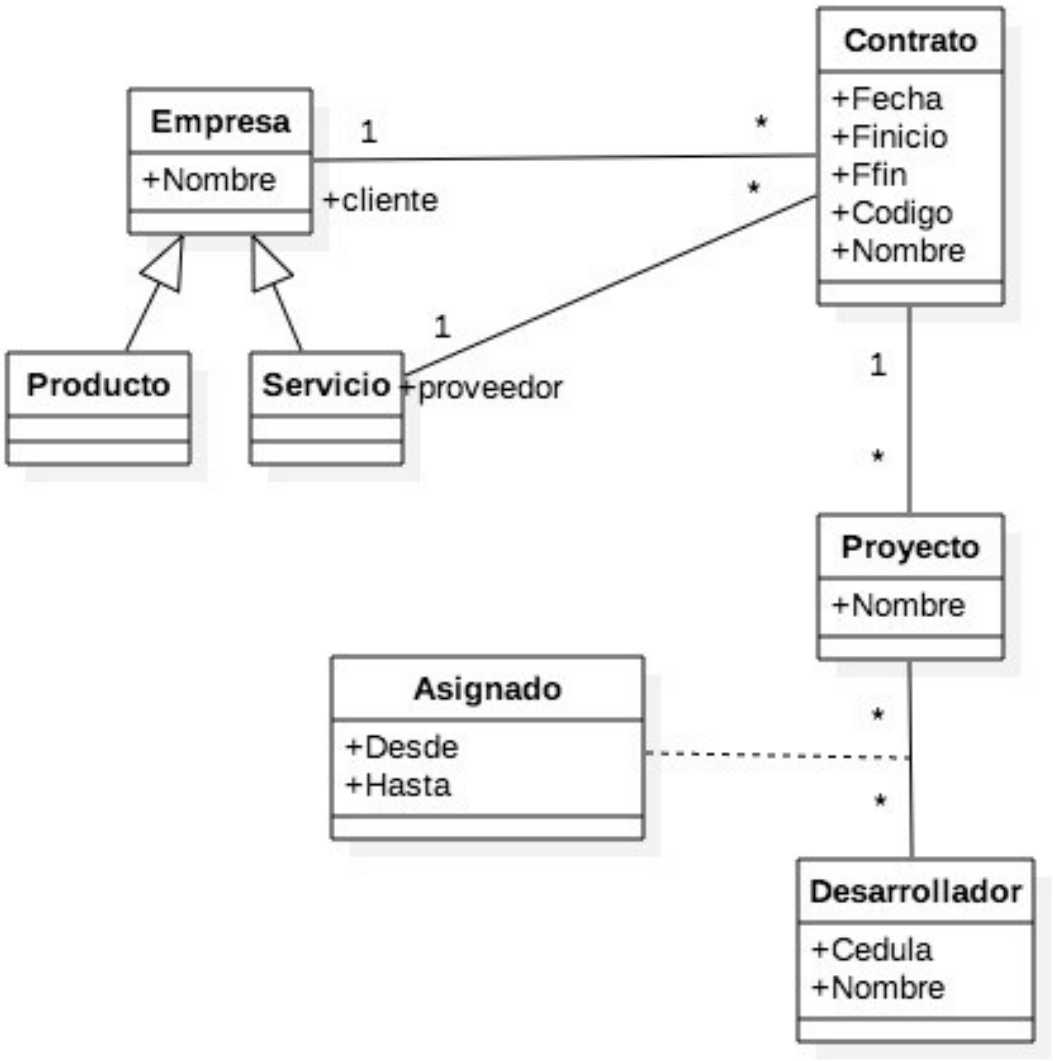
Programación 4

Examen Julio 2015 – Solución

Problema 1

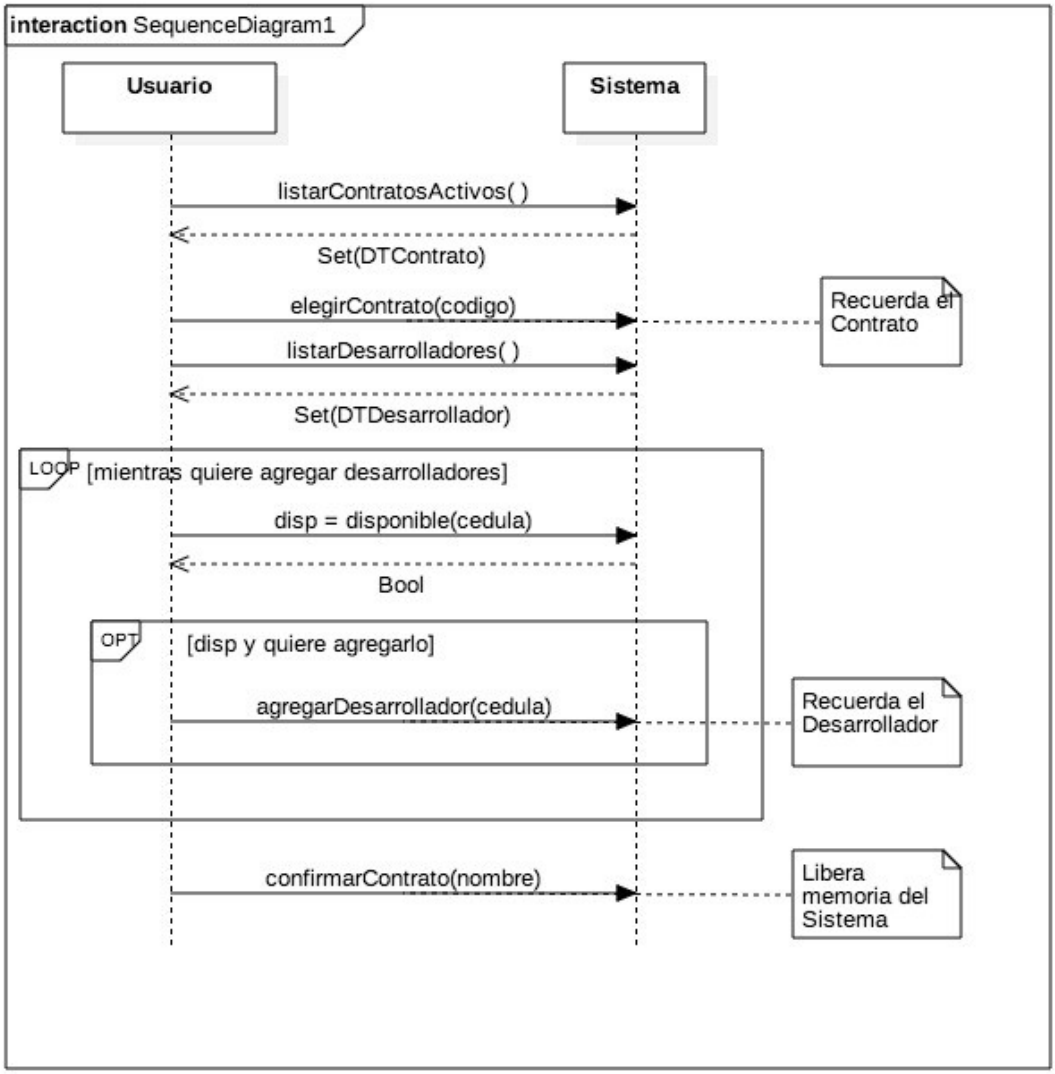
a) Si, mediante el uso de interfaces.

b) i)



Restricciones:
* El codigo del contrato es unico.
* El nombre del proyecto es unico.
* La cedula del desarrollador es unica.
* La fecha de inicio del contrato es menor a la fecha de fin.
* Las fechas Desde y Hasta de asignacion de un desarrollador estan dentro de las fechas de Inicio y Fin de un contrato.
* El cliente es diferente del proveedor de un contrato (una empresa no trabaja para si misma).

b) ii)

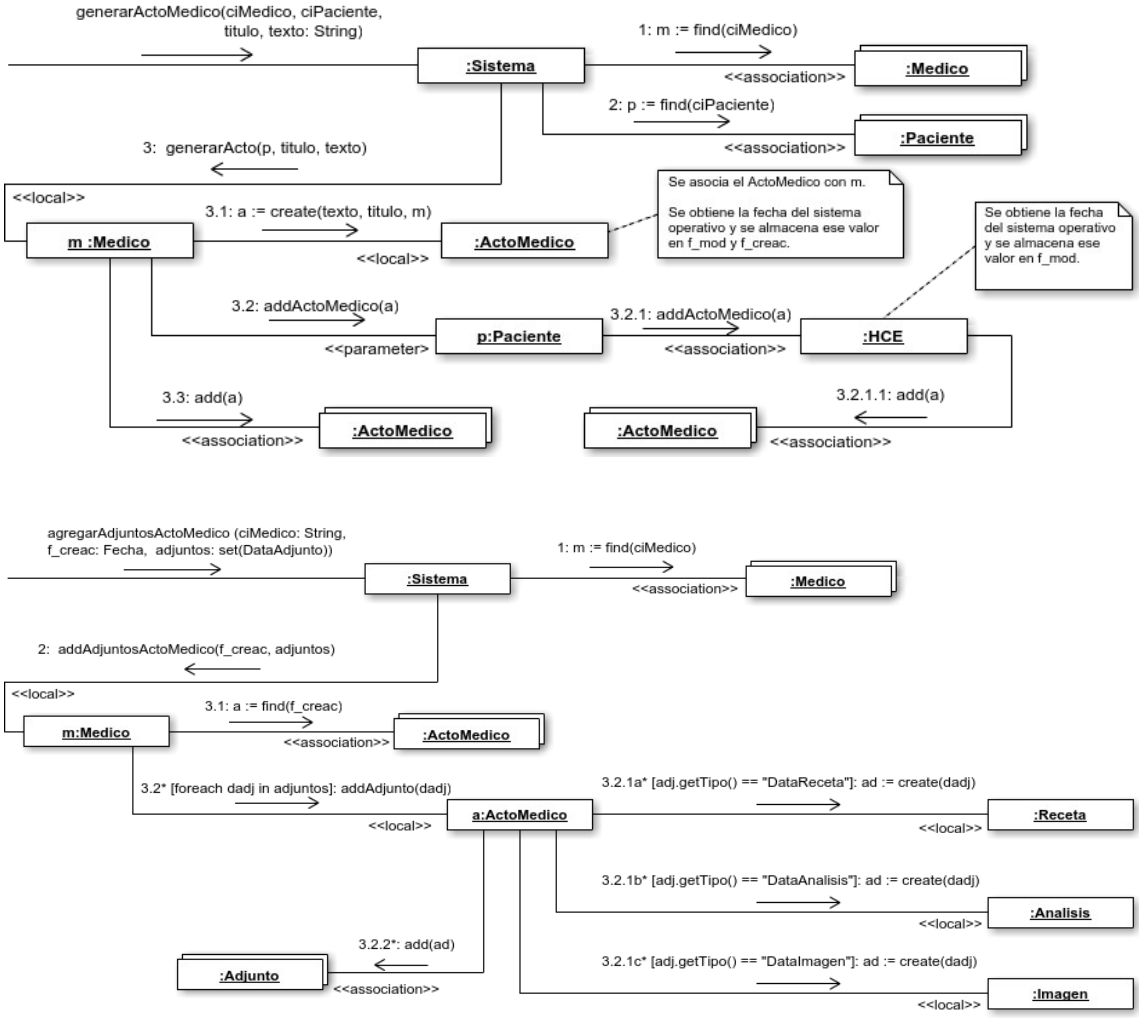

interaction SequenceDiagram1
Usuario
Sistema
listarContratosActivos( )
Set(DTContrato)
elegirContrato(codigo)
Recuerda el Contrato
listarDesarrolladores( )
Set(DTDesarrollador)
LOOP [mientras quiere agregar desarrolladores]
disp = disponible(cedula)
Bool
OPT [disp y quiere agregarlo]
agregarDesarrollador(cedula)
Recuerda el Desarrollador
confirmarContrato(nombre)
Libera memoria del Sistema

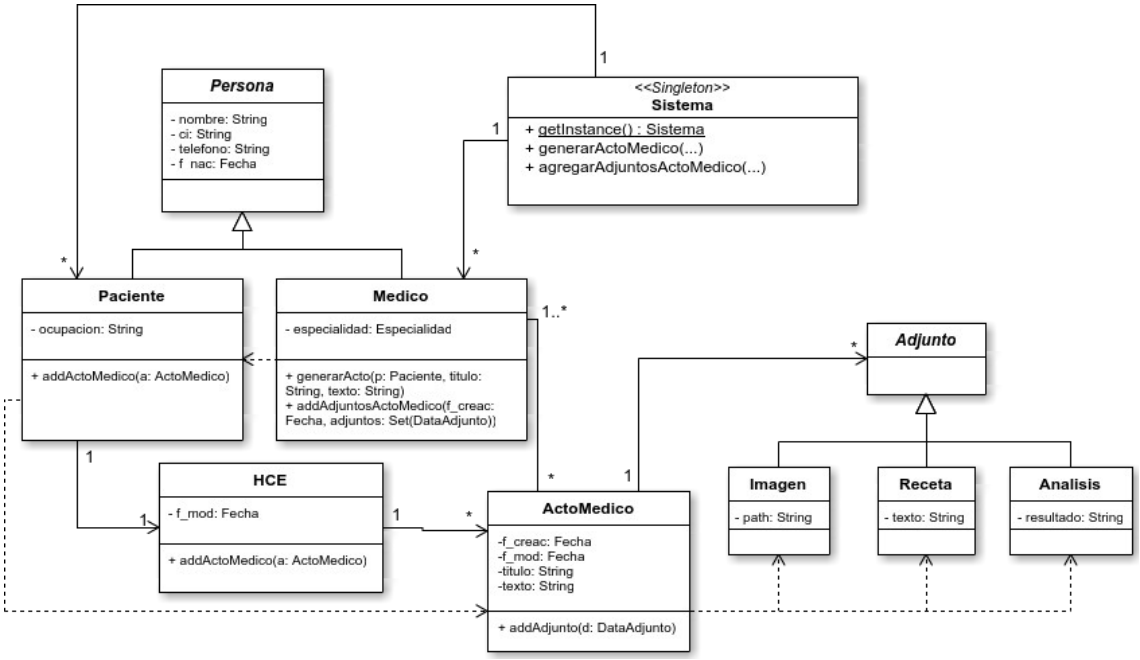
Problema 2

Parte a)

i. Interesa que mencionen que son criterios para una buena asignación de responsabilidades.
ii. Expert, creator, bajo acoplamiento, alta cohesión, no hablar con extraños, controller.

Parte b)





Problema 3

Publicacion.h:

```
class Publicacion {
public:
    Publicacion();
    Publicacion(Usuario*, int fid, string mensaje);

    void darMeGusta(Usuario*);
    void agregarComentario(Comentario*);
    void seguirPublicacion(IObserver*);
    void dejarDeSeguirPublicacion(IObserver*);
    virtual DataPublicacion* obtenerDataPublicacion() = 0;

    virtual ~Publicacion();
private:
    void generarNotificacion();
    Usuario* creador;
    ICollection* meGusta;
    ICollection* comentarios;
    ICollection* seguidores;
    int fid;
    string mensaje;
};
```

```
/*******************************************************/

Foto.h:

class Foto :public Publicacion {
public:
    Foto();
    Foto(Usuario* creador, int fid, string mensaje, string fotoPath);

    virtual DataPublicacion* obtenerDataPublicacion();

    virtual ~Foto();
private:
    string fotoPath;

};

/*******************************************************/

IObserver.h:

class IObserver : public ICollectible {
public:
    IObserver();

    virtual void notificar(Notificacion*) = 0;

    virtual ~IObserver();
private:

};

/*******************************************************/

Usuario.h

class Usuario : public IObserver {
public:
    Usuario();
    Usuario(string mail, string password, string fotoPerfilPath);

    void notificar(Notificacion*);

    virtual ~Usuario();
private:
    string mail;
    string password;
    string fotoPerfilPath;
};

/*******************************************************/

NotificationFactory.h

class NotificationFactory {
public:
    static NotificationFactory* getInstance();
    NotificationFactory(const NotificationFactory& orig);

    Notificacion* generarNotificacion(DataPublicacion*);
```

```
    virtual ~NotificationFactory();
private:
    NotificationFactory();
    static NotificationFactory* instance;
};

/*******************************************************/

Publicacion.cpp

Publicacion::Publicacion() {
}

Publicacion::Publicacion(Usuario* creador, int fid, string mensaje){
    this->creador = creador;
    this->fid = fid;
    this->mensaje = mensaje;
    this->comentarios = new List();
    this->meGusta = new List();
    this->seguidores = new List();
    this->seguirPublicacion(creador);
}

void Publicacion::darMeGusta(Usuario* usuario){
    this->comentarios->add(usuario);
    this->generarNotificacion();
    this->seguirPublicacion(usuario);
}

void Publicacion::agregarComentario(Comentario* comentario){
    this->comentarios->add(comentario);
    this->generarNotificacion();
    this->seguirPublicacion(comentario->getUsuario());
}

void Publicacion::seguirPublicacion(IObserver* observador){
    this->seguidores->add(observador);
}

void Publicacion::dejarDeSeguirPublicacion(IObserver* observador){
    this->seguidores->remove(observador);
}

void Publicacion::generarNotificacion(){
    DataPublicacion* dp = this->obtenerDataPublicacion();
    NotificationFactory* nf = NotificationFactory::getInstance();
    Notificacion* notificacion = nf->generarNotificacion(dp);

    IIterator* it = this->seguidores->getIterator();

    while(it->hasCurrent()){
        IObserver* observador = (IObserver*) it->getCurrent();
        observador->notificar(notificacion);
        it->next();
    }

    delete it;
}

Publicacion::~Publicacion() {
    IIterator* it = this->comentarios->getIterator();
```

```
    while(it->hasCurrent()){
        delete it->getCurrent();
        it->next();
    }

    delete it;
}

/*****************************************************************/

Foto.cpp

Foto::Foto() {
}

Foto::Foto(Usuario* creador, int fid, string mensaje, string fotoPath)
: Publicacion(creador, fid, mensaje) {
    this->fotoPath = fotoPath;
}

DataPublicacion* Foto::obtenerDataPublicacion(){
       return new DataFoto(this->getFid(), this->getMensaje(), this-
>getFotoPath());
}

Foto::~Foto() {
}


/*************************************************************/

NotificationFactory.cpp

NotificationFactory* NotificationFactory::instance = NULL;

NotificationFactory::NotificationFactory() {
}

NotificationFactory* NotificationFactory::getInstance() {
    if (!instance) {
        instance = new NotificationFactory();
    }
    return instance;
}

NotificationFactory::~NotificationFactory() {
}
```