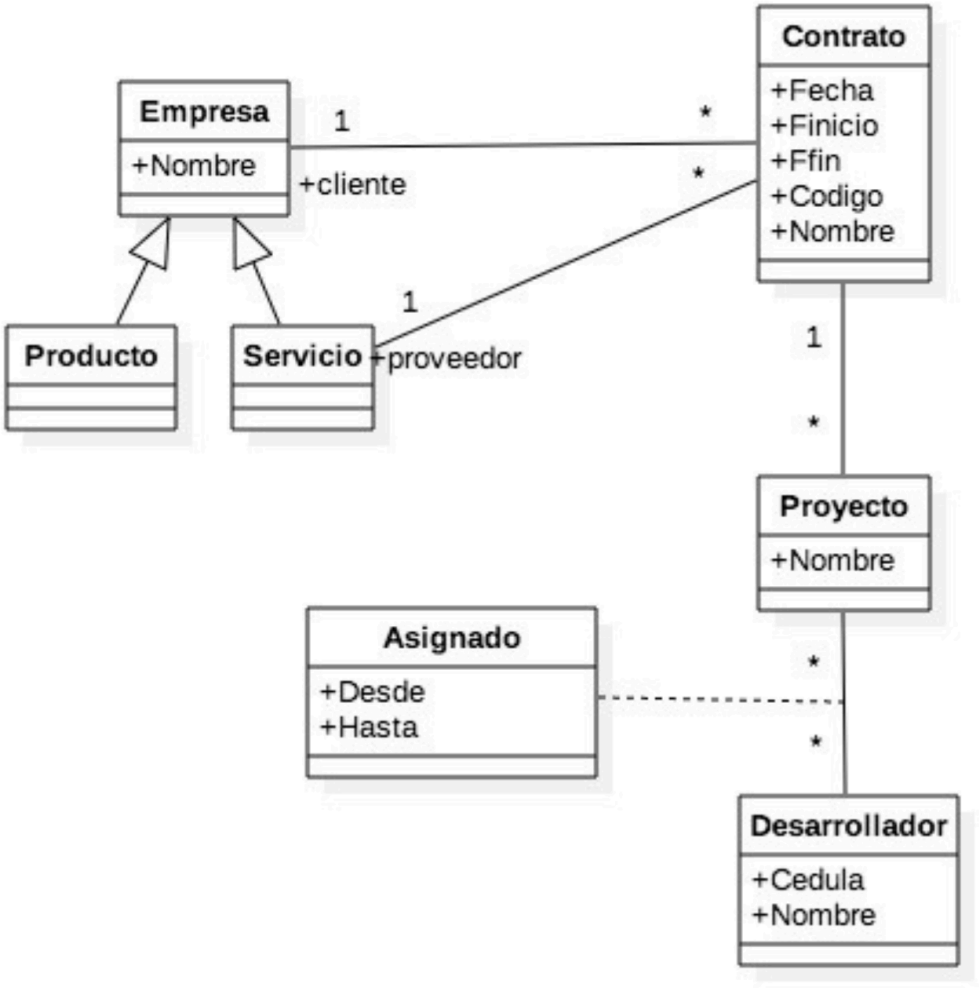
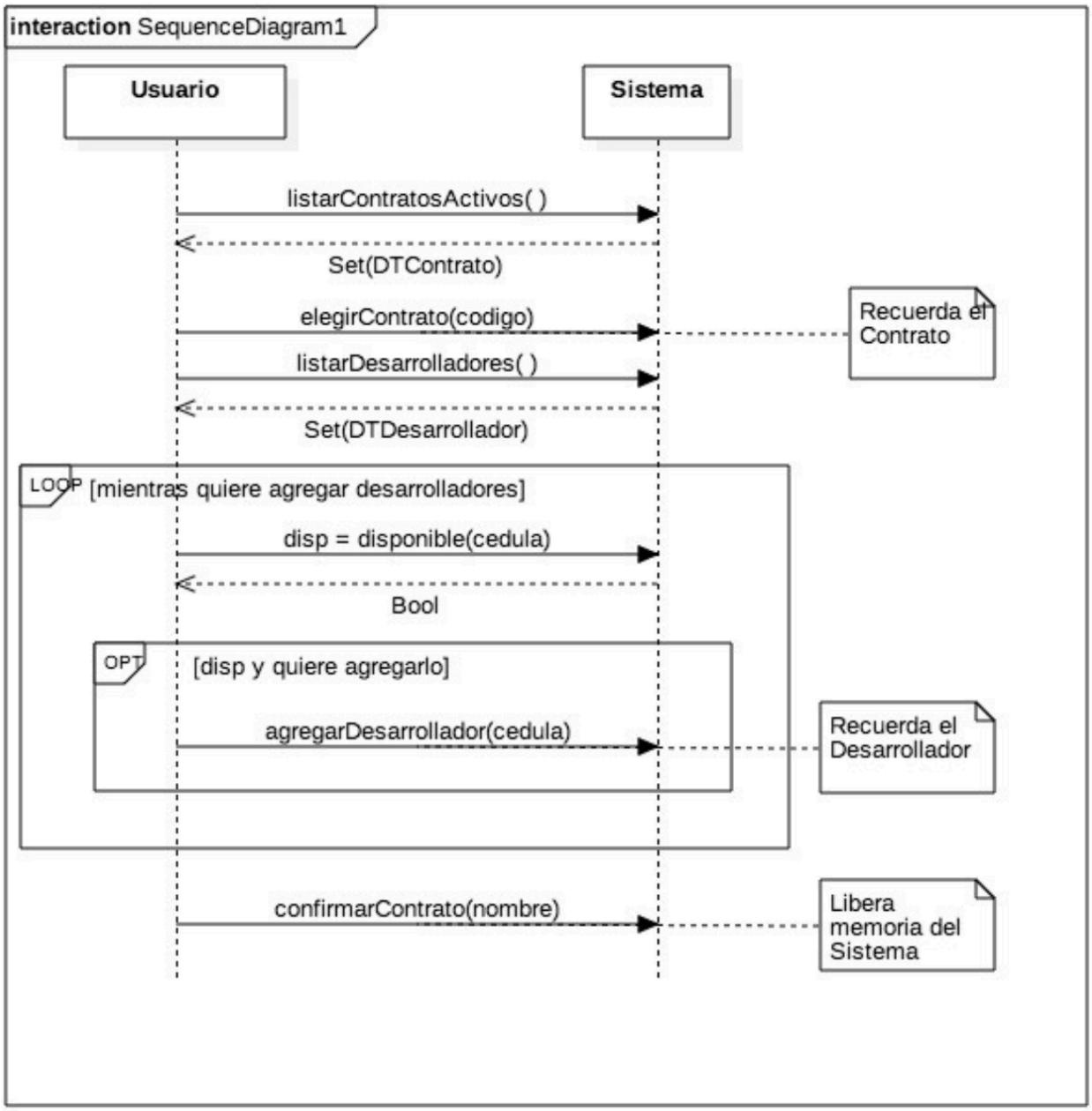
CORRECCIÓN PARCIAL FINAL PA 2020

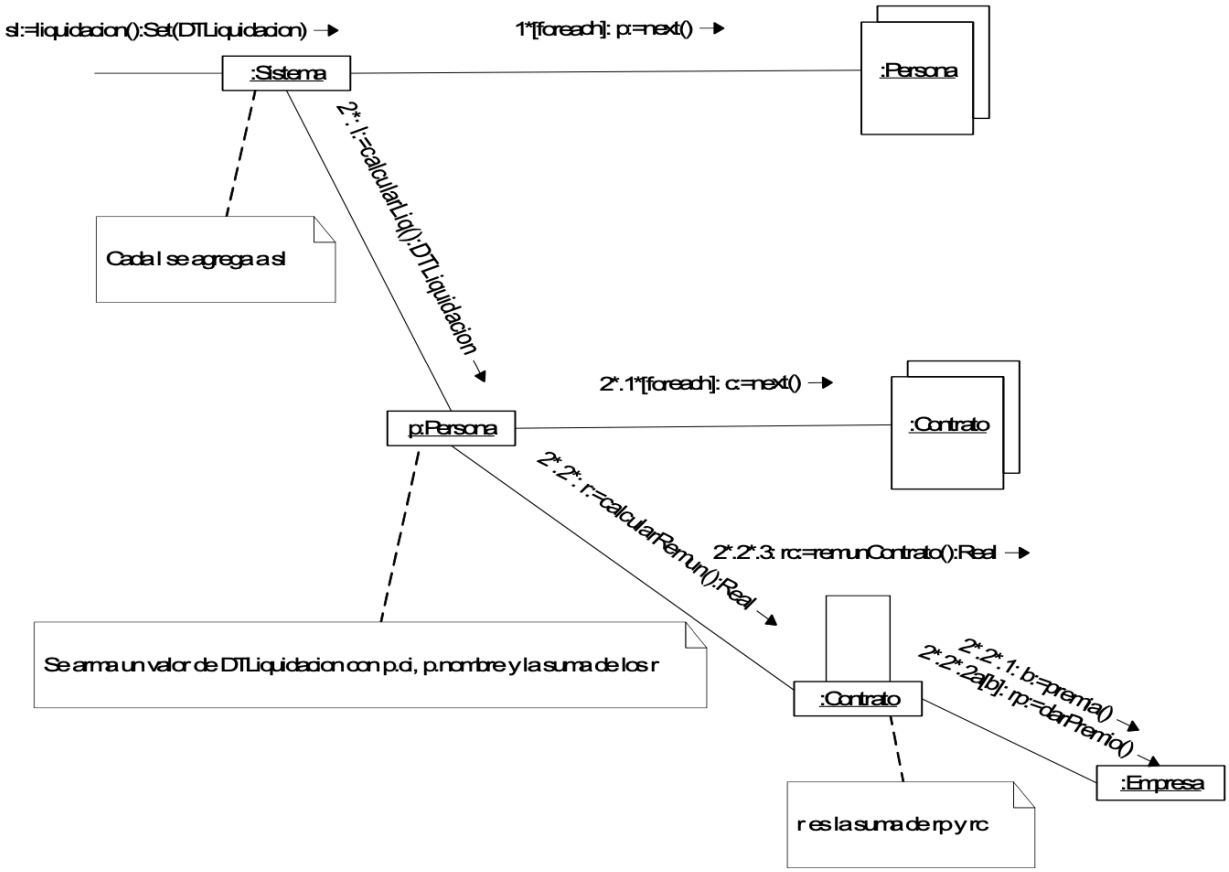
Ejercicio 1)

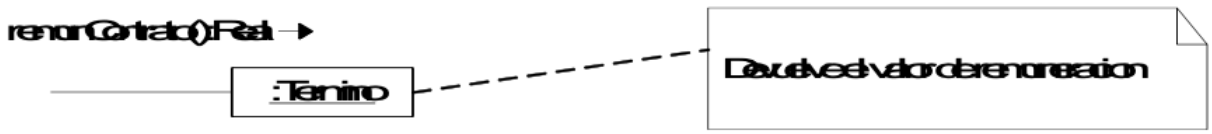


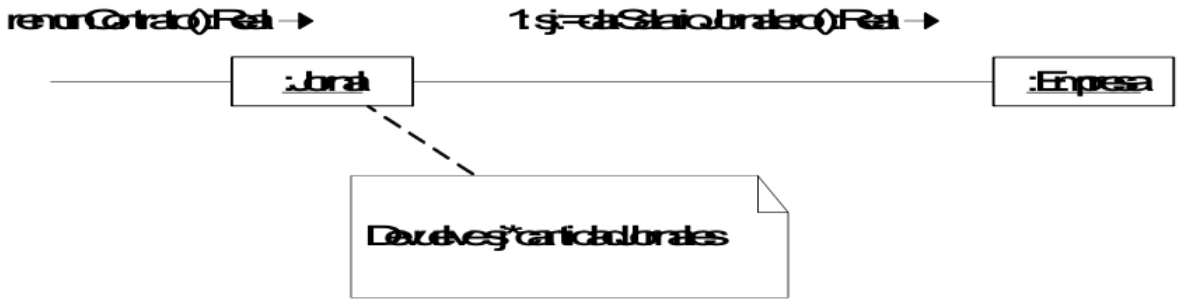
Restricciones:
* El codigo del contrato es unico.
* El nombre del proyecto es unico.
* La cedula del desarrollador es unica.
* La fecha de inicio del contrato es menor a la fecha de fin.
* Las fechas Desde y Hasta de asignacion de un desarrollador estan dentro de las fechas de Inicio y Fin de un contrato.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

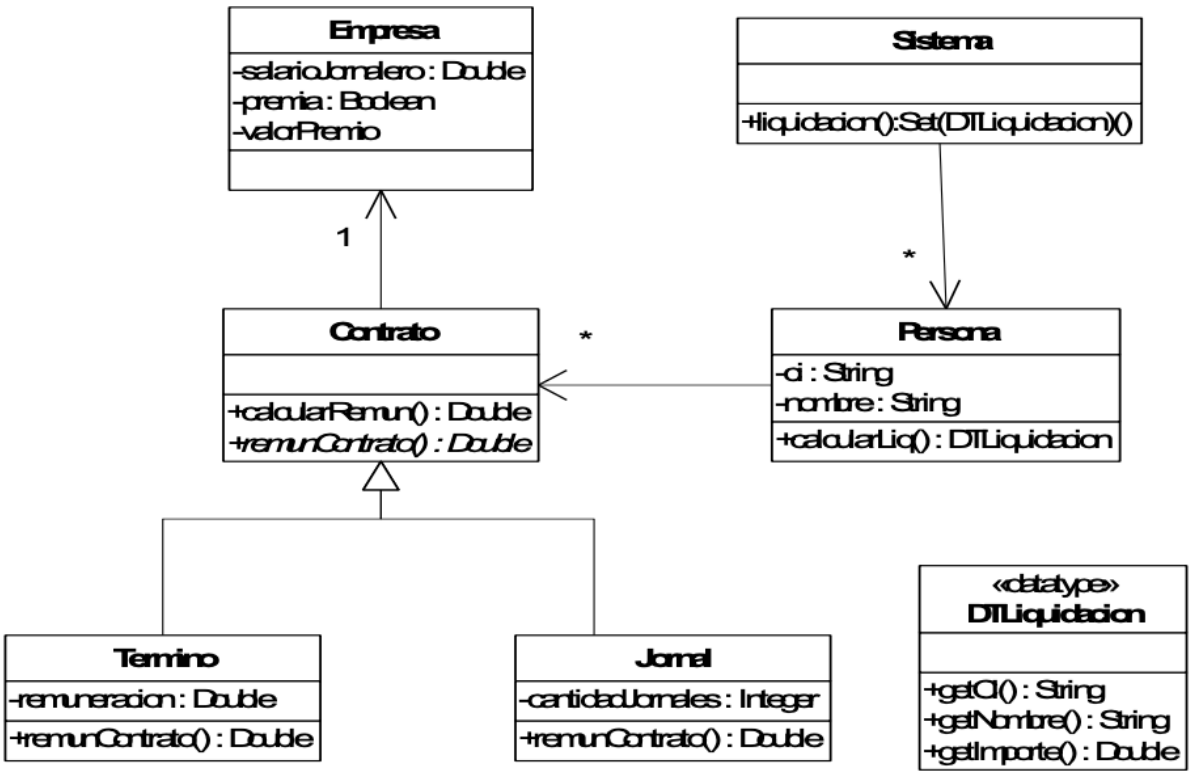

interaction SequenceDiagram1
Usuario
Sistema
listarContratosActivos( )
Set(DTContrato)
elegirContrato(codigo)
Recuerda el Contrato
listarDesarrolladores( )
Set(DTDesarrollador)
LOOP
[mientras quiere agregar desarrolladores]
disp = disponible(cedula)
Bool
OPT
[disp y quiere agregarlo]
agregarDesarrollador(cedula)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

Ejercicio 2)


sl:=liquidacion():Set(DTLiquidacion)
1*[foreach]: p:=next()
:Sistema
:Persona
2*.1:=calcularLiq():DTLiquidacion
Cada l se agrega a sl
2*.1*[foreach]: c:=next()
p:Persona
:Contrato
2*.2*: r:=calcularRemun():Real
2*.2*.3: rc:=remunContrato():Real
Se arma un valor de DTLiquidacion con p.ci, p.nombre y la suma de los r
:Contrato
2*.2*.1: b:=premia()
2*.2*.2a[b]: rp:=darPremio()
:Empresa
r es la suma de rp y rc


:Termino
Devuelve el valor de enumeracion


:Jornal
:Empresa
Devuelve sj*cantidadJornales



Ejercicio 3)

```
class Paquete : public ICollectible {
  public:
    virtual double calcularPeso() = 0;
    virtual double calcularVolumen() = 0;
    virtual ~Paquete();
}

Paquete::~Paquete() {
}

class Sencillo : public Paquete {
  private:
    double peso, volumen;
  public:
    Sencillo(double, double);
    double getPeso();
    void setPeso(double);
    double getVolumen();
    void setVolumen(double);
    double calcularPeso();
    double calcularVolumen();
}
```

```
Sencillo::Sencillo(double p, double v) {
  peso = p;
  volumen = v;
}

double Sencillo::getPeso() {
  return peso;
}

void Sencillo::setPeso(double p) {
  peso = p;
}

double Sencillo::getVolumen() {
  return volumen;
}

void Sencillo::setVolumen(double v) {
  volumen = v;
}

double Sencillo::calcularPeso() {
  return peso;
}

double Sencillo::calcularVolumen() {
  return volumen;
}

class Complejo : public Paquete {
  private:
    ICollection *componentes;
    OptimizadorVolumen *optimizador;
  public:
    Complejo(ICollection *, OptimizadorVolumen *);
    ~Complejo();
    double calcularPeso();
    double calcularVolumen();
    void setOptVol(OptimizadorVolumen *);

Complejo::Complejo(ICollection *comps, OptimizadorVolumen *opt) {
  componentes = new List;
  IIterator *it = comps->getIterator();
  while (it->hasCurrent()) {
    componentes->add(it->getCurrent());
    it->next();
  }
  delete it;
  optimizador = opt;
}
```

```
Complejo::~Complejo() {
  IIterator *it = componentes->getIterator();
  ICollectible *elem;
  while (it->hasCurrent()) {
    elem = it->getCurrent();
    it->next();
    componentes->remove(elem);
    delete elem;
  }
  delete it;
  delete componentes;
}

double Complejo::calcularPeso() {
  IIterator *it = componentes->getIterator();
  double result = 0;
  while (it->hasCurrent()) {
    result = result + ((Paquete *)it->getCurrent())->getPeso();
    it->next();
  }
  delete it;
  return result;
}

double Complejo::calcularVolumen() {
  return opt->volOptimo(componentes);
}

void Complejo::setOptVol(OptimizadorVolumen *opt) {
  optimizador = opt;
}

class OptimizadorVolumen {
  public:
    virtual double valOptimo(ICollection *) = 0;
    virtual ~OptimizadorVolumen();
}

OptimizadorVolumen::~OptimizadorVolumen() {
}
```