Programación 4

SOLUCIÓN EXAMEN JULIO 2011

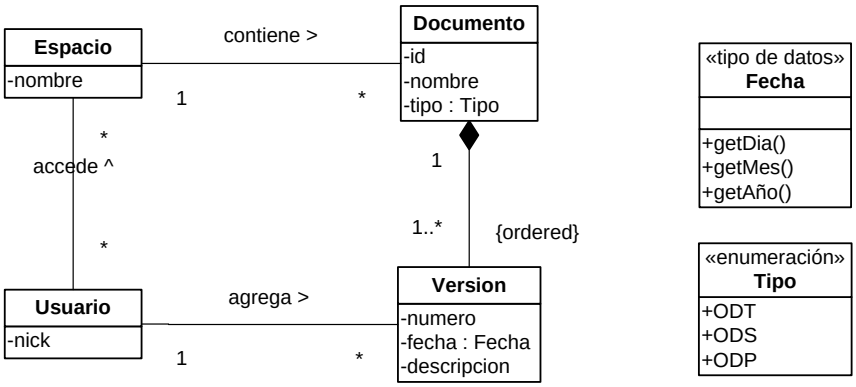
Problema 1 (30 puntos)

a)

Un contrato de software especifica el comportamiento o efecto de una operación, determinando derechos y obligaciones para el proveedor y consumidor de la operación.
Tipos de condiciones que se expresan en un contrato:
- creación de objetos
- eliminación de objetos
- conexión de objetos
- desconexión de objetos
- modificación del valor de los atributos de objetos

b)

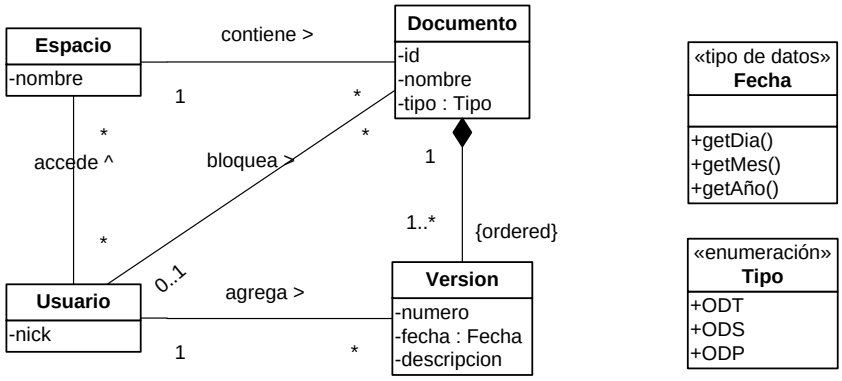
i.



ii.

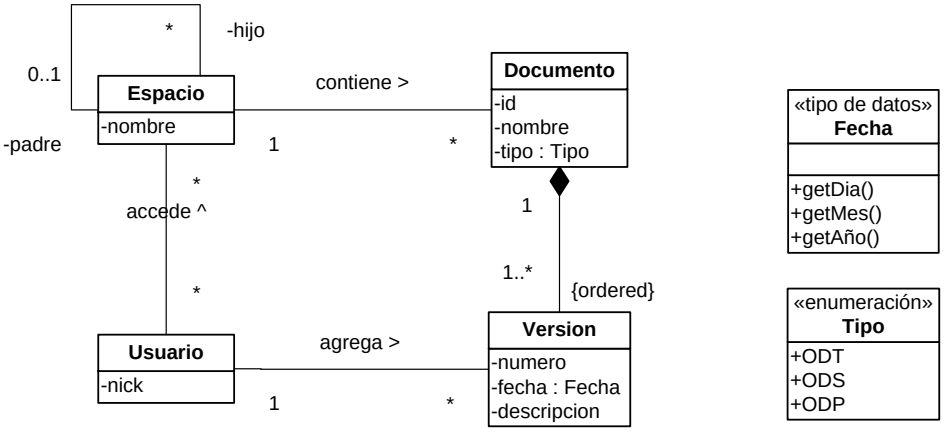
R1. No existen dos espacios con el mismo nombre.
R2. No existen dos usuarios con el mismo nick.
R3. No existen dos documentos con el mismo id.
R4. En el contexto de un documento, no existen dos versiones con el mismo número.
R5. Para un documento, los números de sus versiones son correlativos.
R6. La fecha de una versión de un documento es mayor o igual a las fechas de las versiones con numeración menor de dicho documento.

iii.



R7. El documento bloqueado por un usuario, debe tener una versión del documento agregada por dicho usuario.

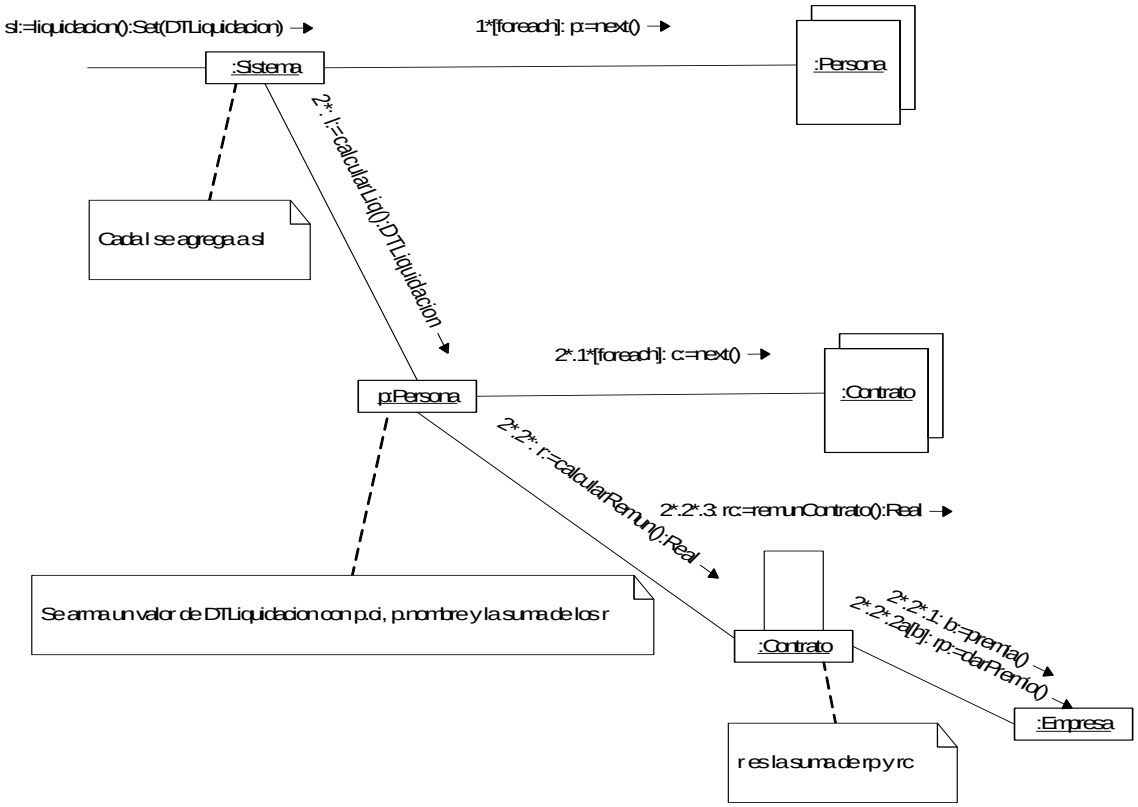
iv.

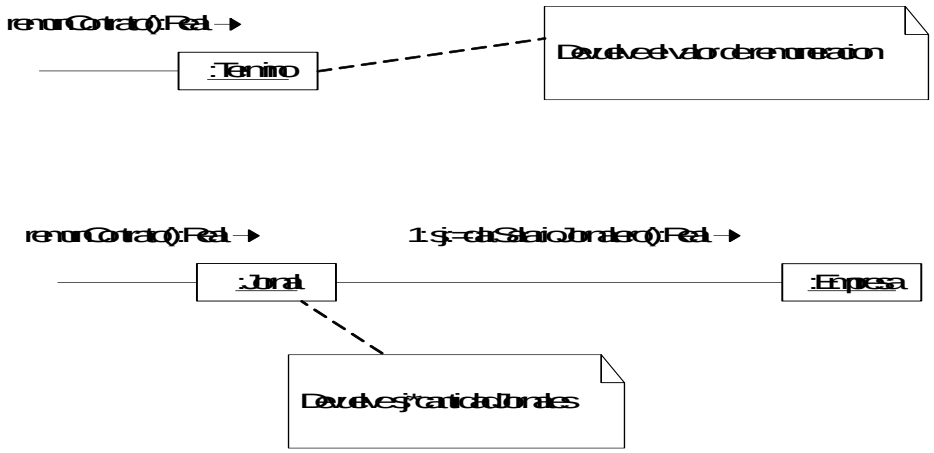


Problema 2 (35 puntos)

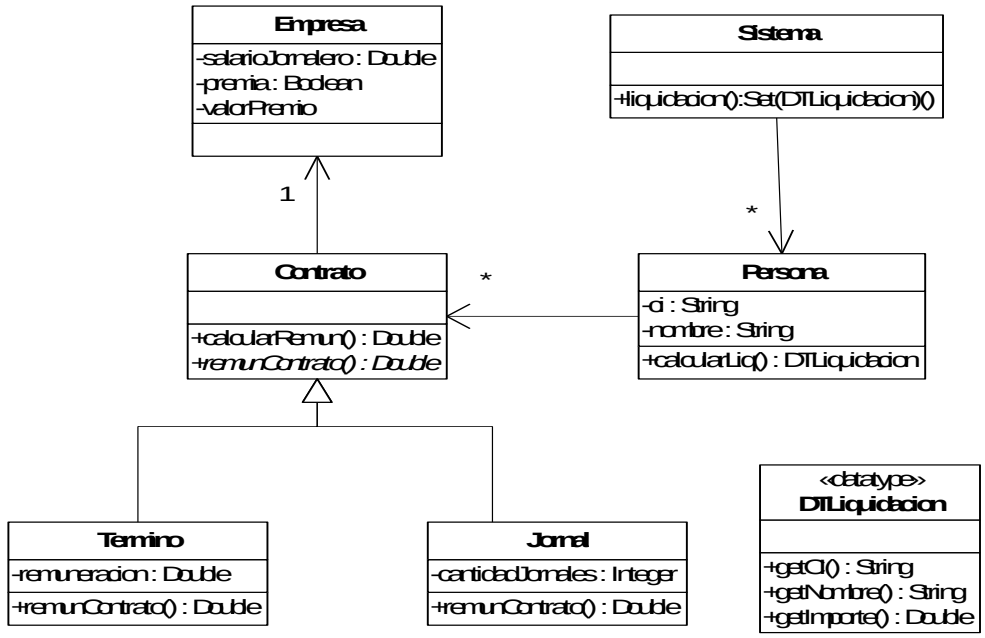
Parte 1

i.



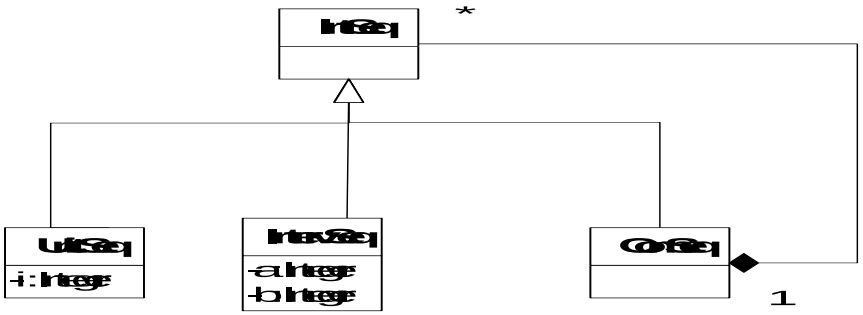


ii



Parte 2

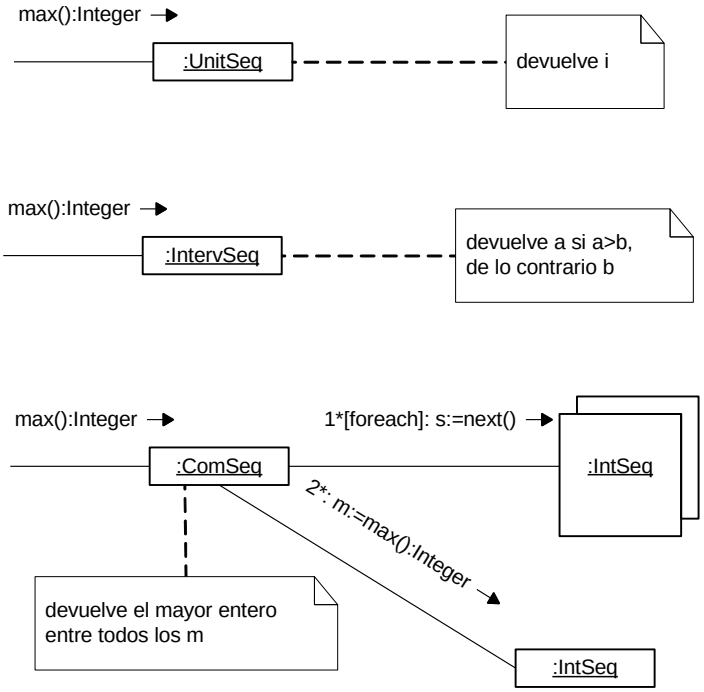
a.



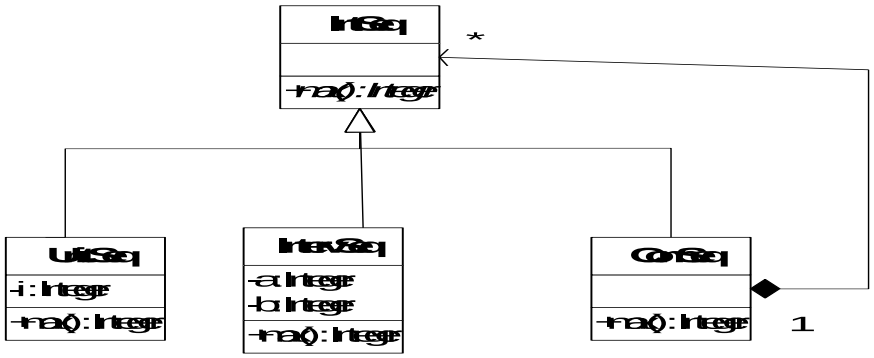
Una instancia de ComSeq no puede estar asociada a sí misma.

b.

i.



ii.



iii.

```
class IntSeq : public ICollectible {
      public:
            virtual int max()=0;
}

class UnitSeq : public IntSeq {
      private:
            int i;
      public:
            UnitSeq(int);
            void setI(int);
            int getI();
            int max();
}
```

```
class IntervSeq : public IntSeq {
      private:
            int a,b;
      public:
            IntervSeq(int,int);
            void setA(int);
            void setB(int);
            int getA();
            int getB();
            int max();
}

class ComSeq : public IntSeq {
      private:
            ICollection *seqs;
      public:
            ComSeq();
            ICollection * getSeqs();
            int max();
}
```

Problema 3 (35 puntos)

i.

El patrón de diseño utilizado es Observer, de la siguiente forma:

- Subject: ControladorProductos
- Observer: IProveedor
- Concrete Observer: Proveedor1, Proveedor2

ii.

ControladorProductos.h

```
class ControladorProductos
{
private:
        static ControladorProductos* instance;
        IDictionary* productos;
        ICollection* proveedores;
        ControladorProductos();
        void enviarAlertaStock();

public:
        static ControladorProductos* getInstance();
        void vender(int idProd);
        void agregarProveedor(IProveedor* pr);
        void quitarProveedor(IProveedor* pr);
        void agregarProducto(Producto* p);
        virtual ~ControladorProductos();
};
```

ControladorProductos.cpp

```
ControladorProductos* ControladorProductos::instance = 0;

ControladorProductos::ControladorProductos(){
```

```
	productos = new Dictionary();
	proveedores = new Collection();
}

ControladorProductos* ControladorProductos::getInstance(){
	if (instance==0){
		instance = new ControladorProductos();
	return instance;
}

void ControladorProductos::enviarAlertaStock(int idProd){
	IIterator* it();
	for(it = proveedores->getIterator(); it->hasCurrent(); it-
>next()){
		IProveedor* p = (IProveedor*) it->current();
		p->alertaStock(idProd);
	}
	delete it;
}

void ControladorProductos::vender(int idProd){
	IKey* k = new IntegerKey(idProd);
	Producto* p = (Producto*)productos->find(k);

	int s = p->getStock();
	if(s == 1)
		this->enviarAlertaStock(idProd);
	delete k;
}

void ControladorProductos::agregarProveedor(IProveedor* pr){
	proveedores->add(pr);
}

void ControladorProductos::quitarProveedor(IProveedor* pr){
	proveedores->remove(pr);
}

void ControladorProductos::agregarProducto(Producto* p){
	IKey* k = new IntegerKey(p->getId());
	productos->add(k,p);
}

ControladorProductos::~ControladorProductos(){
	IIterator* it;
	for(it = productos->getIterator(); it->hasCurrent(); it-
>next()){
		Producto* p = (Producto*) it->current();
		delete p;
	}
	delete this->productos;
	delete it;

	for(it = proveedores->getIterator(); it->hasCurrent(); it-
>next()){
		IProveedor* prov = (IProveedor*) it->current();
		delete prov;
	}
	delete this->proveedores;
	delete it;
```

```
      this->instance = 0;
}
```

iii.

IProveedor.h

```
class IProveedor : public ICollectible{
public:
      virtual void alertaStock(int idProd) = 0;
      virtual ~IProveedor();
};
```

Proveedor1.h

```
class Proveedor1 : IProveedor{
public:
      void alertaStock(int idProd);
      virtual ~Proveedor1();
};
```