Programación 4

PARCIAL FINAL EDICIÓN 2011

SOLUCIÓN

Por favor siga las siguientes indicaciones:

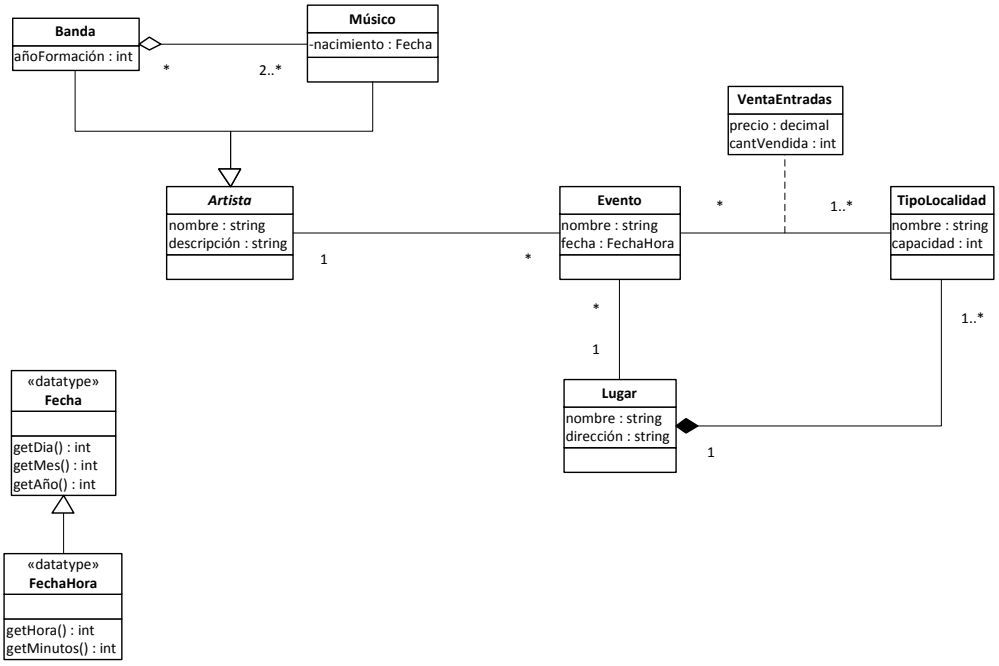
- Escriba con lápiz
- Escriba las hojas de un solo lado
- Escriba su nombre y número de documento en todas las hojas que entregue
- Numere las hojas e indique el total de hojas en la primera de ellas
- Recuerde entregar su numero de parcial junto al parcial

Problema 1 (30 puntos)

a) Las asociaciones que son de comprensión no son indispensables para que el modelo conceptual sea correcto. Lo contrario sucede con las asociaciones need-to-know cuya ausencia hace que el modelo no refleje cierta información del modelo que sí está presente en la realidad que se quiere representar.

b)

i.

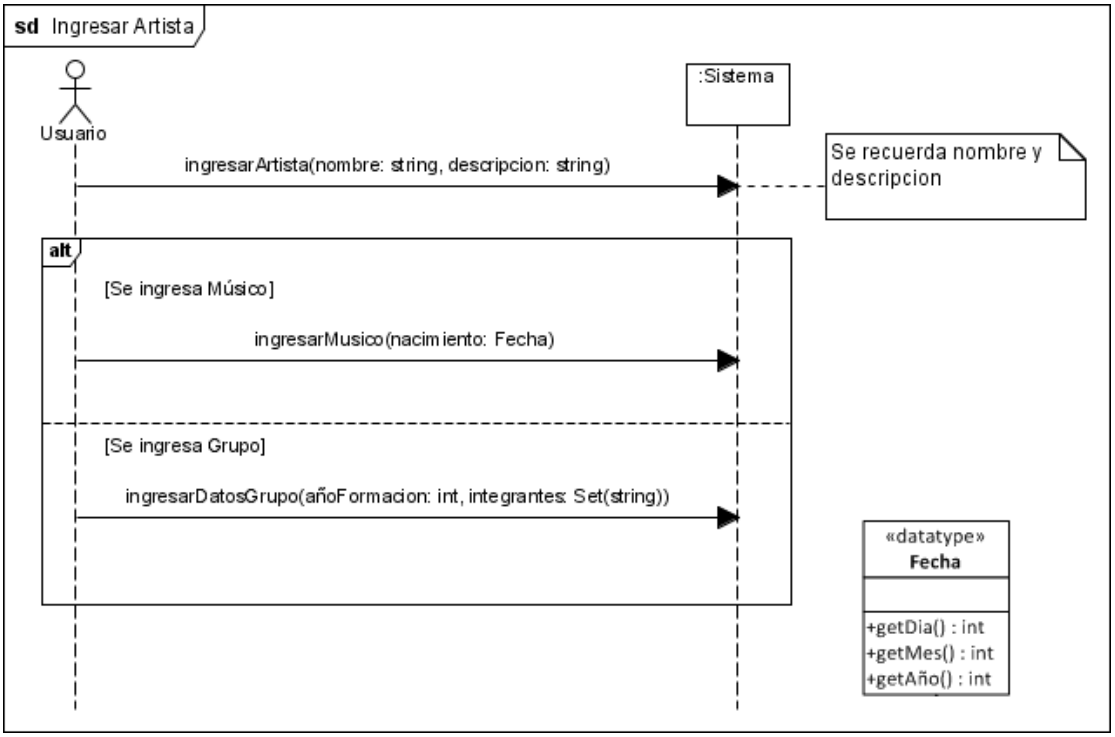


Restricciones:

- No hay dos instancias de artistas distintas con el mismo nombre
- Para cada instancia de Banda, no existe ninguna instancia de Solista cuya fecha de nacimiento sea posterior al año de formación de la banda
- No hay dos instancias de Evento distintas con el mismo nombre

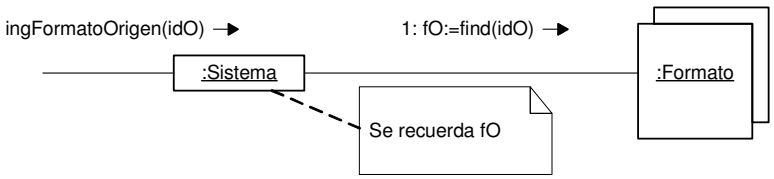
- No hay dos instancias de Lugar distintas con el mismo nombre
- No hay dos instancias de TipoLocalidad distintas asociadas al mismo Evento y con el mismo nombre
- Para cada instancia de VentaEntradas cantVendida es menor o igual al atributo capacidad de la instancia TipoLocalidad que tiene asociada.
- Para cada instancia de Evento las instancias de TipoLocalidad que tiene asociados son exactamente las mismas que a las que tiene asociadas la instancia Lugar del evento.

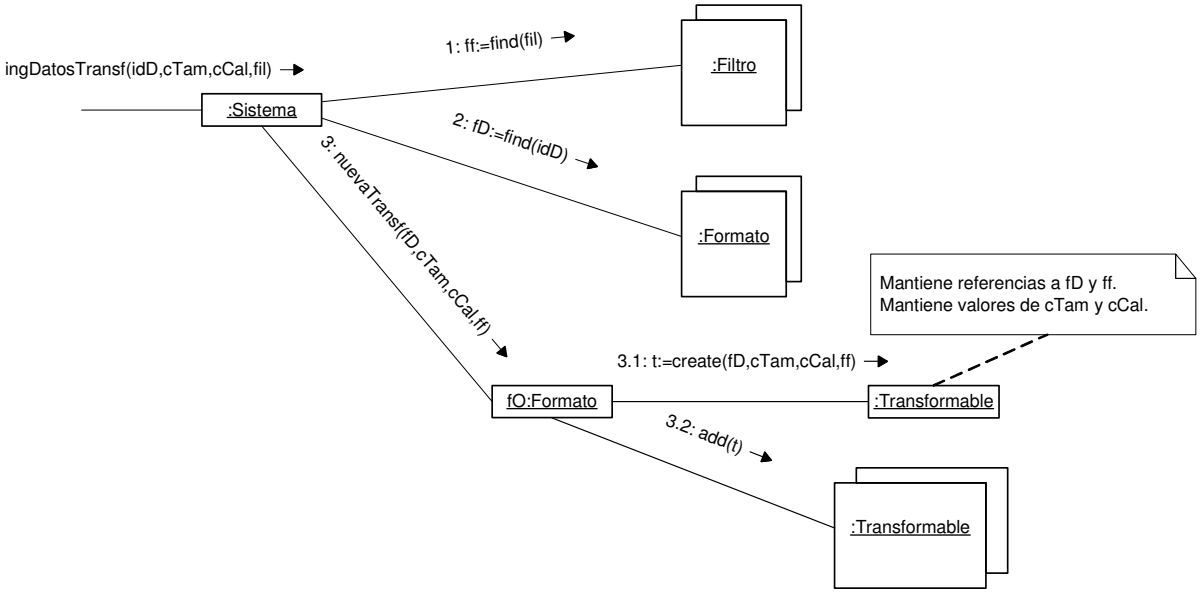
ii.

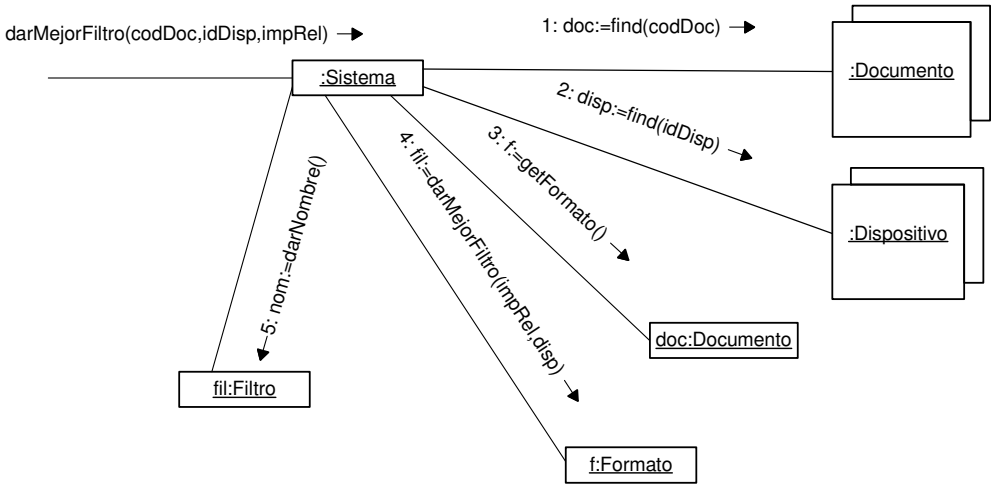


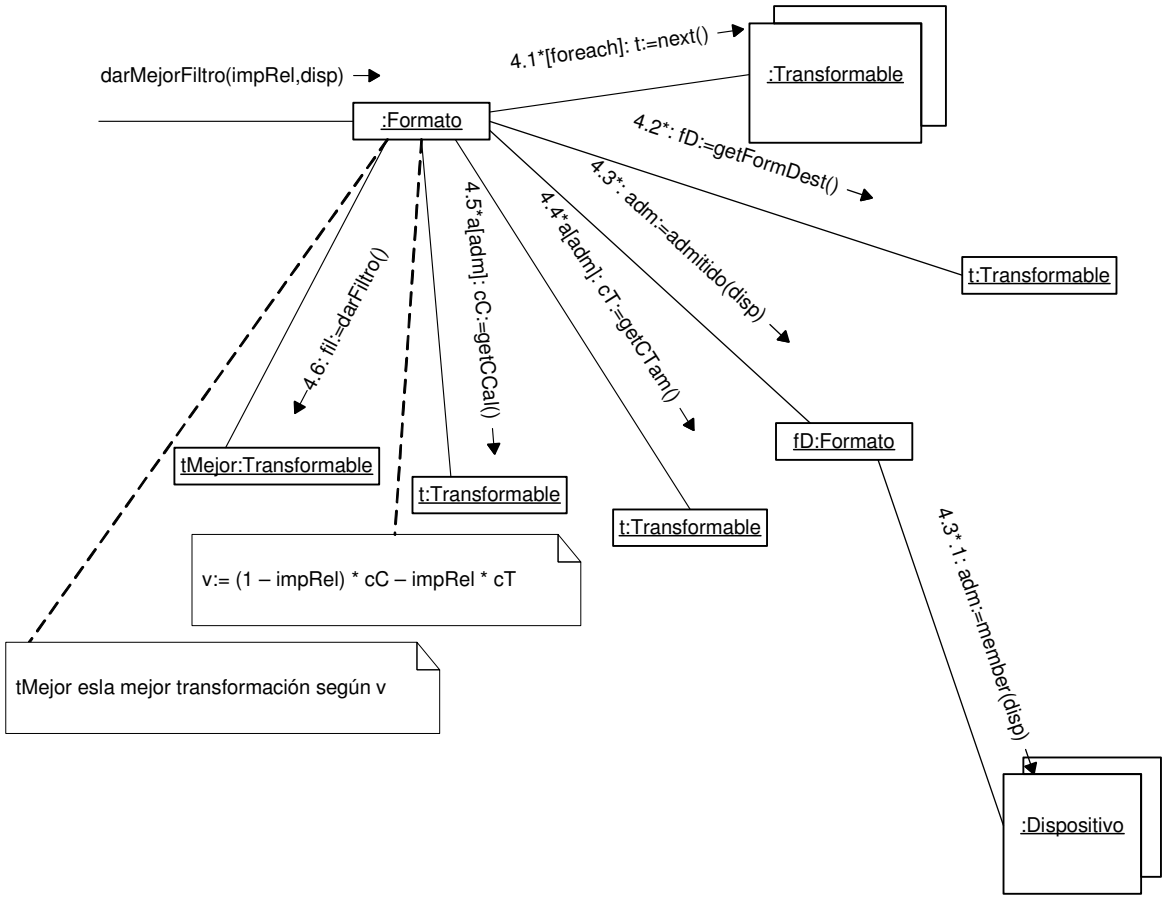
Problema 2 (35 puntos)

i.

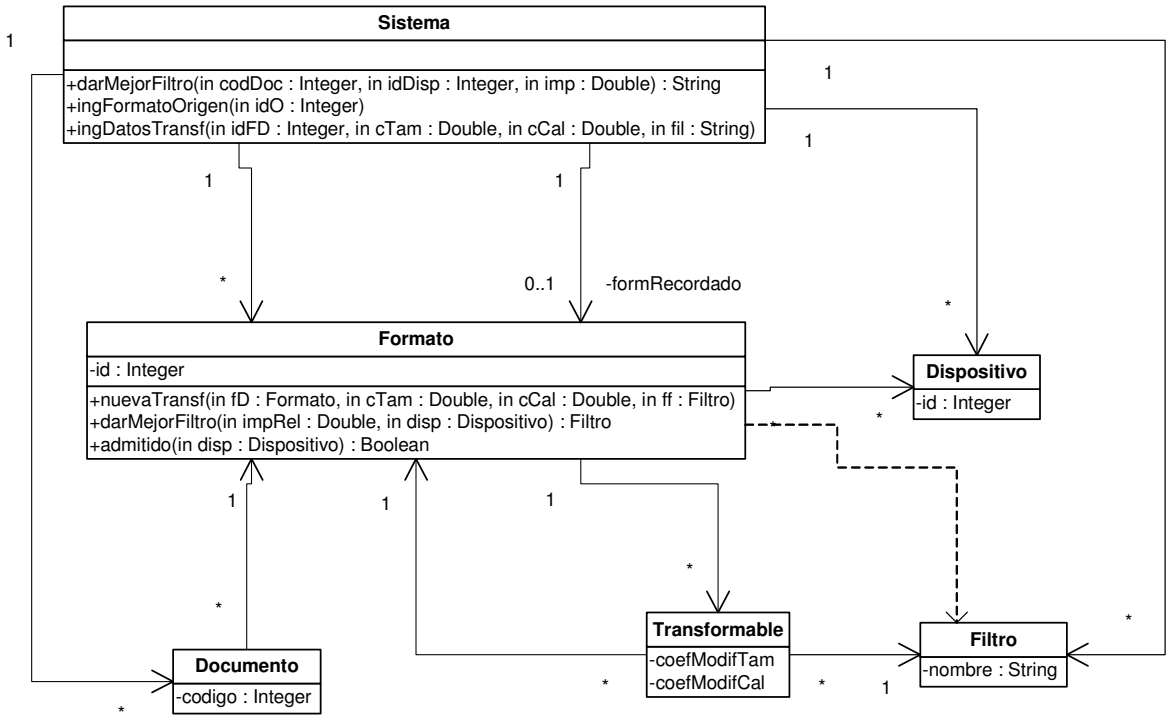



ingDatosTransf(idD,cTam,cCal,fil)
:Sistema
1: ff:=find(fil)
:Filtro
2: fD:=find(idD)
:Formato
3: nuevaTransf(fD,cTam,cCal,ff)
Mantiene referencias a fD y ff.
Mantiene valores de cTam y cCal.
3.1: t:=create(fD,cTam,cCal,ff)
fO:Formato
:Transformable
3.2: add(t)
:Transformable


darMejorFiltro(codDoc,idDisp,impRel)
:Sistema
1: doc:=find(codDoc)
:Documento
2: disp:=find(idDisp)
:Dispositivo
3: f:=getFormato()
doc:Documento
4: fil:=darMejorFiltro(impRel,disp)
5: nom:=darNombre()
fil:Filtro
f:Formato


darMejorFiltro(impRel,disp)
:Formato
4.1*[foreach]: t:=next()
:Transformable
4.2*: fD:=getFormDest()
t:Transformable
4.3*: adm:=admitido(disp)
4.4*a[adm]: cT:=getCTam()
4.5*a[adm]: cC:=getCCal()
4.6: fil:=darFiltro()
fD:Formato
tMejor:Transformable
t:Transformable
t:Transformable
v:= (1 – impRel) * cC – impRel * cT
4.3*.1: adm:=member(disp)
tMejor esla mejor transformación según v
:Dispositivo

ii.


Sistema
+darMejorFiltro(in codDoc : Integer, in idDisp : Integer, in imp : Double) : String
+ingFormatoOrigen(in idO : Integer)
+ingDatosTransf(in idFD : Integer, in cTam : Double, in cCal : Double, in fil : String)
0..1
-formRecordado
Formato
-id : Integer
+nuevaTransf(in fD : Formato, in cTam : Double, in cCal : Double, in ff : Filtro)
+darMejorFiltro(in impRel : Double, in disp : Dispositivo) : Filtro
+admitido(in disp : Dispositivo) : Boolean
Dispositivo
-id : Integer
Transformable
-coefModifTam
-coefModifCal
Filtro
-nombre : String
Documento
-codigo : Integer

Problema 3 (35 puntos)

i.

SpecialIterator.h

```
class SpecialIterator : public Iterator{
      private:
      int stepSize;
      int forward;
      Node * current;
      SpecialIterator();

      public:
      SpecialIterator(Node*);
      virtual int hasNext();
      virtual Object* next();
      void setForward();
      void setBackward();
      void setStepSize(int);
      virtual ~SpecialIterator();

};
```

SpecialIterator.cpp

```
SpecialIterator::SpecialIterator(Node *current){
      this->stepSize = 1;
      this->forward = 1;
      this->current = current;
}

SpecialIterator::SpecialIterator(){
      this->stepSize = 1;
      this->forward = 1;
      this->current = 0;
}

void SpecialIterator::setForward(){
      this->forward = 1;
}

void SpecialIterator::setBackward(){
      this->forward = 0;
}

void SpecialIterator::setStepSize(int sSize){
      this->stepSize = sSize;
}


int SpecialIterator::hasNext(){
      if (current == 0)
            return 0;
      int ret = 1;
      Node * tmp = current;
```

```
	for(int i = 1; i <= stepSize; i++){
		if(this->forward() == 1){
			if(tmp->hasNext()){
				tmp = tmp->getNext();
			}else{
				ret = 0;
				break;
			}
		}else{
			if(tmp->hasPrevious()){
				tmp = tmp->getPrevious();
			}else{
				ret = 0;
				break;
			}
		}
	}
	return ret;
}

Object* SpecialIterator::next(){
	if (current == 0)
		throw std::domain_error("don't have next item.");
	Object* ret = 0;
	Node * tmp = current;
	for(int i = 1; i <= stepSize; i++){
		if(this->forward() == 1){
			if(tmp->hasNext()){
				tmp = tmp->getNext();
			}else{
				throw std::domain_error("don't have next
item.");
				break;
			}
		}else{
			if(tmp->hasPrevious()){
				tmp = tmp->getPrevious();
			}else{
				throw std::domain_error("don't have previous
item.");
				break;
			}
		}
	}
	ret = tmp->getItem();
	current = tmp;
	return ret;
}

SpecialIterator::~SpecialIterator(){
}
```

ii.

```
DoubleLinkedList.cpp

Iterator* DoubleLinkedList::iterator(){
	Iterator* i = new SpecialIterator(this->first);
	return i;
}
```