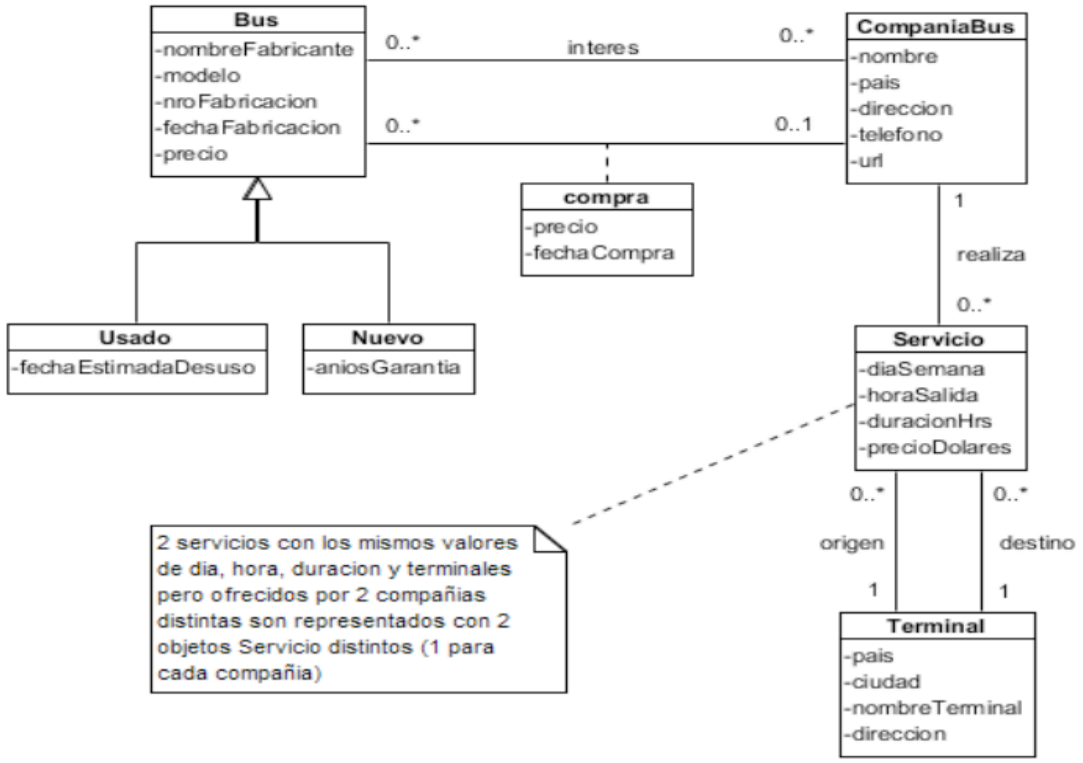
Programación Avanzada

SOLUCIÓN EXAMEN JULIO 2016

Ejercicio 1



Restricciones no estructurales

Identificadores

1. En Bus, (nombreFabricante, modelo, nroFabricacion)
2. En CompaniaBus, (nombre, pais)
3. En Terminal, (pais, ciudad, nombreTerminal)

Valores de atributos

4. Dado un bus b, b.precio > 0
5. Dado un bus nuevo bn, bn.aniosGarantia > 0
6. Dado un servicio s, s.diaSemana ϵ {lunes, martes, miercoles, jueves, viernes, sabado, domingo}, s.horaSalida es un datavalue con 2 atributos: hh y mm, donde hh ϵ [1..24] y mm ϵ [0..59] y s.precioDolares > 0

Restricciones entre asociaciones

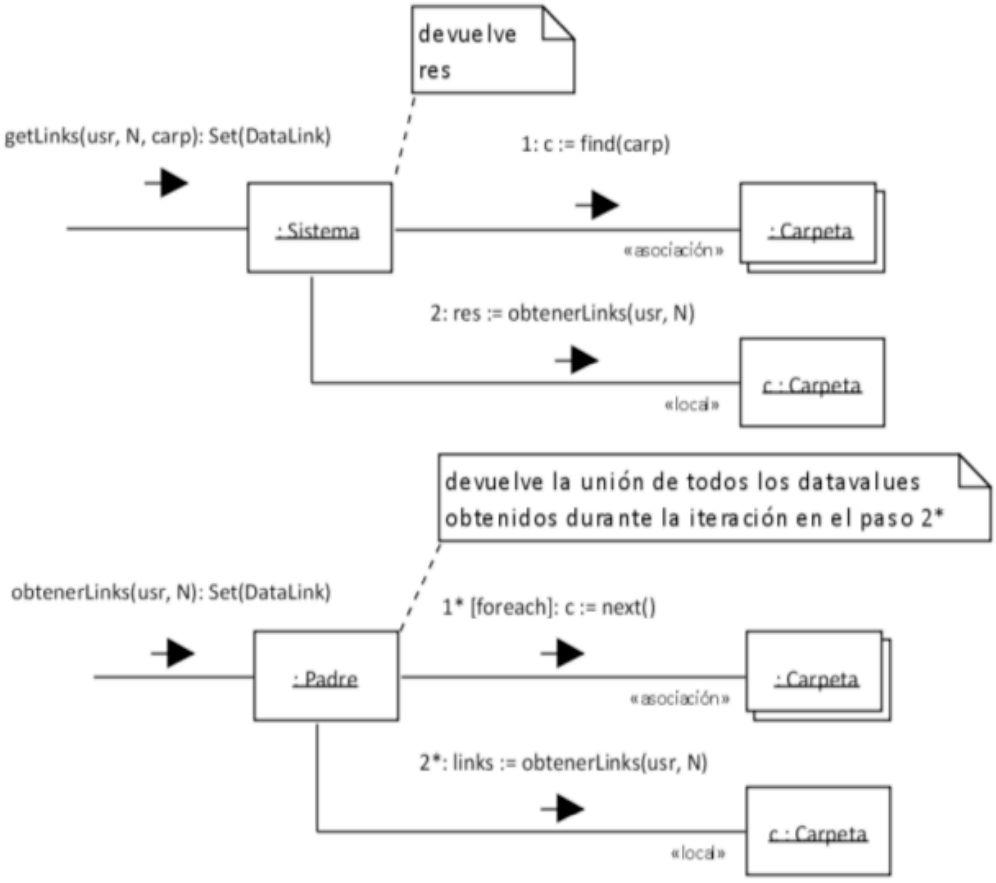
7. La fecha en que realiza una compra debe ser posterior a la fecha de fabricación
8. En el caso de que se realiza la compra de un bus usado, la fecha de la compra deber ser inferior a la fecha estimada de desuso
9. Dado un link de compra entre la compañía c y el bus usado b1 y existe un bus nuevo b2 tal que b1.nombreFabricante = b2.nombreFabricante, entonces debe existir un link de interes entre la compañía c y un bus nuevo b3 tal que b3.nombreFabricante = b1.nombreFabricante
10. Dado un servicio s, s.origen != s.destino
11. En la asociación "realiza" se debe cumplir que cada compañía ofrece desde una misma terminal a lo sumo 2 servicios diarios a una misma terminal destino.

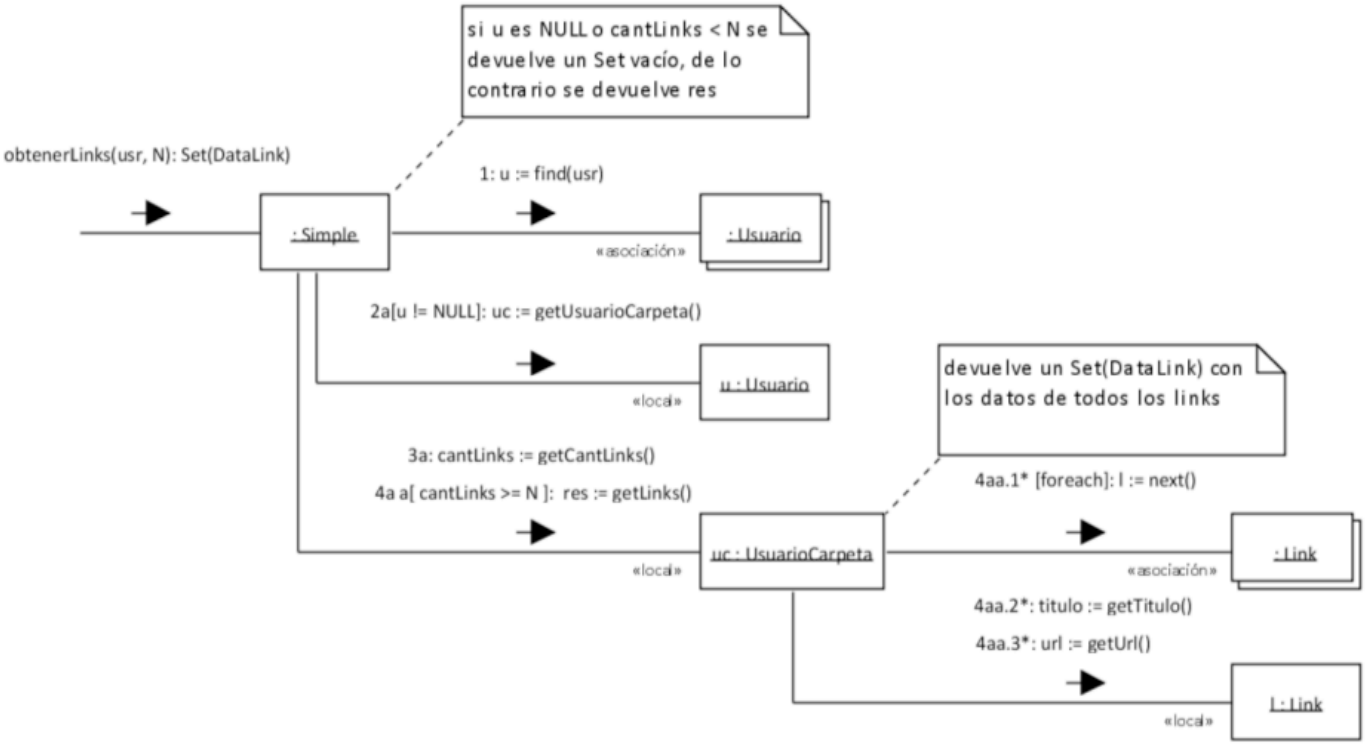
Ejercicio 2

Hay dos alternativas para diseñar la operación getLinks(). Una es obtener los links buscando en la carpeta de nombre carp y luego navegar por el subárbol de los hijos buscando para cada carpeta simple, si el usuario subió N links en cada carpeta que se itera. La segunda alternativa consiste en buscar primero al Usuario y para cada carpeta simple que tenga asociada determinar si subió N links en esa carpeta que se itera y luego determinar si la carpeta tiene nombre carp, o es hija de alguna carpeta de nombre carp navegando hacia el padre.

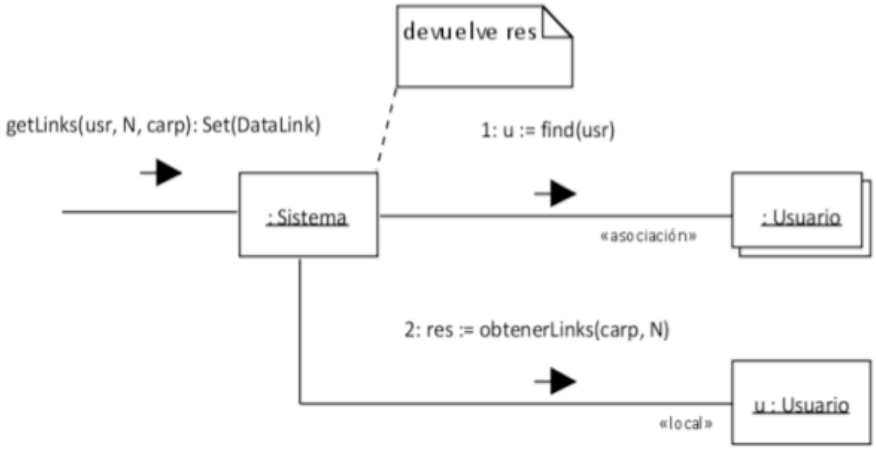
Parte i.

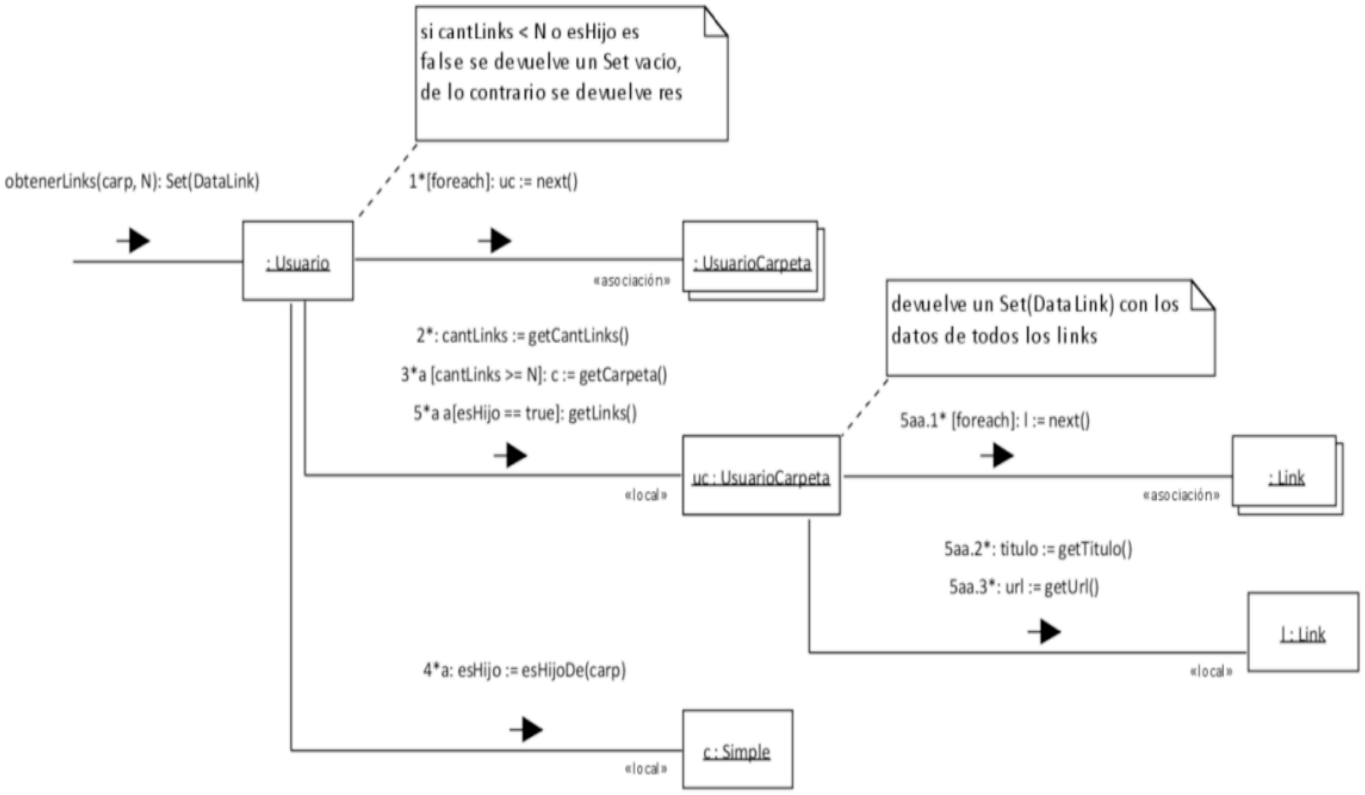
Solución 1

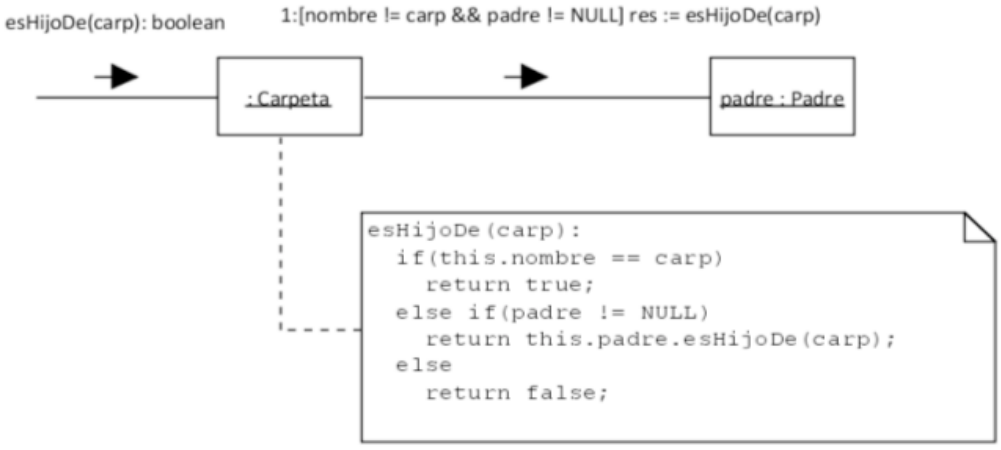



si u es NULL o cantLinks < N se devuelve un Set vacío, de lo contrario se devuelve res
obtenerLinks(usr, N): Set(DataLink)
1: u := find(usr)
: Simple
«asociación»
: Usuario
2a[u != NULL]: uc := getUsuarioCarpeta()
«local»
u : Usuario
devuelve un Set(DataLink) con los datos de todos los links
3a: cantLinks := getCantLinks()
4a a[cantLinks >= N]: res := getLinks()
4aa.1* [foreach]: l := next()
«local»
uc : UsuarioCarpeta
«asociación»
: Link
4aa.2*: titulo := getTitulo()
4aa.3*: url := getUrl()
«local»
l : Link

Solución 2

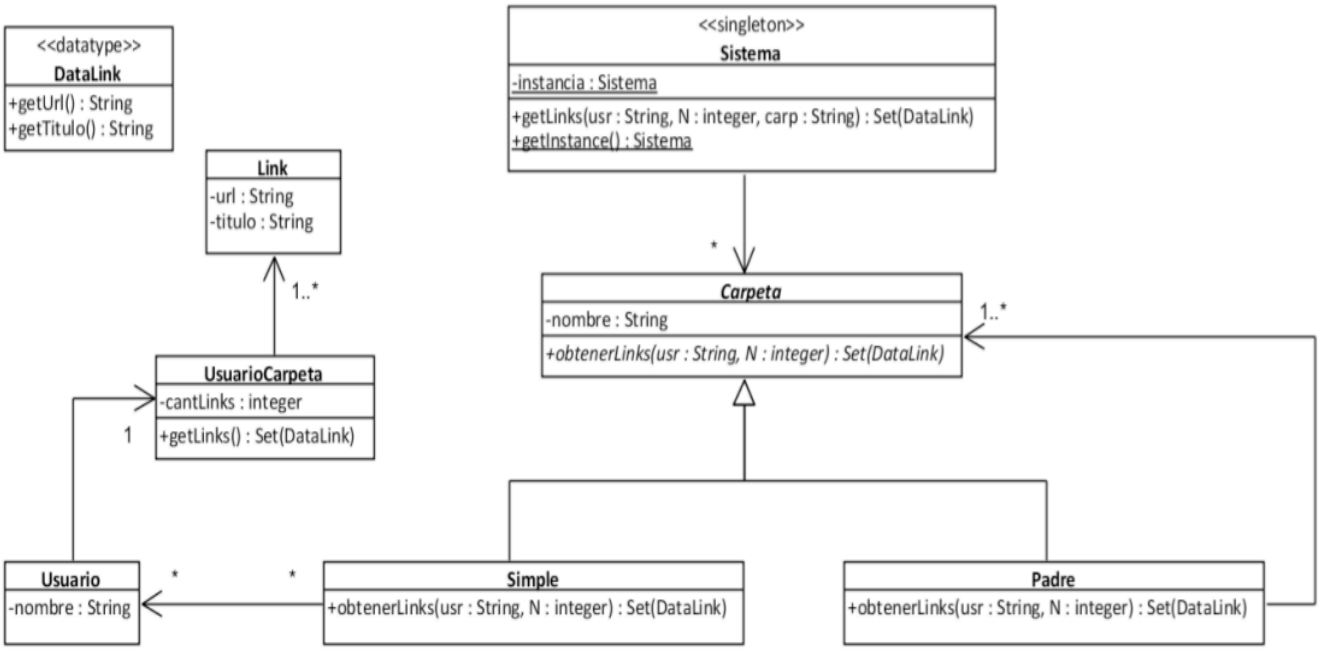

devuelve res
getLinks(usr, N, carp): Set(DataLink)
1: u := find(usr)
: Sistema
«asociación»
: Usuario
2: res := obtenerLinks(carp, N)
«local»
u : Usuario


si cantLinks < N o esHijo es
false se devuelve un Set vacio,
de lo contrario se devuelve res
obtenerLinks(carp, N): Set(DataLink)
1*[foreach]: uc := next()
: Usuario
«asociación»
: UsuarioCarpeta
devuelve un Set(DataLink) con los
datos de todos los links
2*: cantLinks := getCantLinks()
3*a [cantLinks >= N]: c := getCarpeta()
5*a a[esHijo == true]: getLinks()
5aa.1* [foreach]: l := next()
«local»
uc : UsuarioCarpeta
«asociación»
: Link
5aa.2*: titulo := getTitulo()
5aa.3*: url := getUrl()
l : Link
«local»
4*a: esHijo := esHijoDe(carp)
«local»
c : Simple

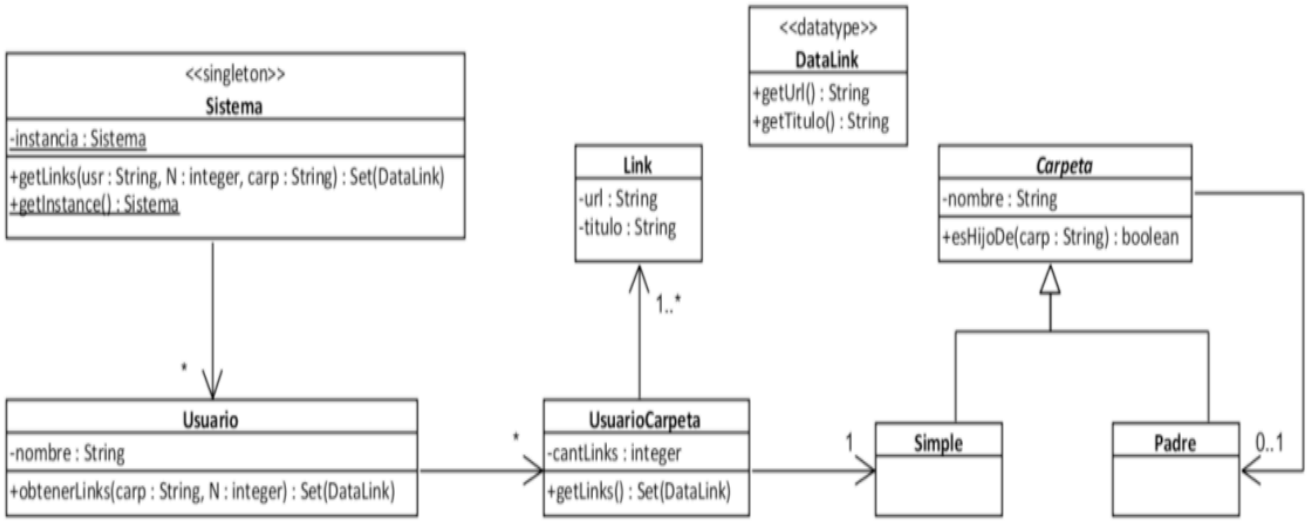

esHijoDe(carp): boolean
1:[nombre != carp && padre != NULL] res := esHijoDe(carp)
: Carpeta
padre : Padre
esHijoDe(carp):
if(this.nombre == carp)
return true;
else if(padre != NULL)
return this.padre.esHijoDe(carp);
else
return false;

Parte ii.

<u>Solución 1</u> (correspondiente a la solución 1 de la parte i)



<u>Solución 2</u> (correspondiente a la solución 2 de la parte i)



Ejercicio 3

i. Implementar en C++ los .h de las clases Sistema, Artículo, Categoría y Compuesta.

```
// Sistema.h
class Sistema {
        private:
                IDictionary *categorias;
                IDictionary *articulos;
        public:
I               InfoArticulo *procesarCategoria(String cat);
};

// Articulo.h
class Articulo {
        private:
                String codigo;
                float precio;
                static float precioMinimo;
                int stock;
        public:
                InfoArticulo *getInfo();
                float getPrecio();
                void setStock(int stock);
};

// Categoria.h
class Categoria: public ICollectible {
        private:
                String nombre;
        public:
                virtual ICollection *getAllCodigos() = 0;
};

// Compuesta.h
class Compuesta: public Categoria {
        private:
                ICollection hijos;
        public:
                ICollection *getAllCodigos();
};
```

ii. Implementar en C++ los .cpp de las clases Sistema y Compuesta.

```
// Sistema.cpp
InfoArticulo *Sistema::procesarCategoria(String cat) {
	InfoArticulo *ret = NULL;
	float precioMinimo = getPrecioMinimo();
	Categoria *c = (Categoria*) categorias->find(cat);
	if(cat == NULL)
		throw std::invalid_argument("No existe la categoria");
	ICollection *codigos = c->getAllCodigos();
	IIterator *it;
	for(it = codigos->getIterator(); it->hasCurrent(); it->next()){
		String *cod = (String *)(it->getCurrent());
		Articulo *a = (Articulo *) articulos->find(cod);
		float precio = a->getPrecio();
		if(ret == NULL || precio > ret->getPrecio)
			ret = a->getInfo();
		if(precio < precioMinimo)
			a->setStock(0);
	}
	delete it;
	// borra la coleccion de codigos devueltos
	for(it = c->getIterator(); it->hasCurrent(); it->next()){
		String *cod = (String *)(it->getCurrent());
		it->removeCurrent();
		delete cod;
	}
	delete it;
	delete codigos;
}

// Compuesta.cpp
ICollection *Compuesta::getAllCodigos()
{
	ICollection *res = new Col;
	IIterator *it = hijos->getIterator();
	for(it = hijos->getIterator(); it->hasCurrent(); it->next()){
		Categoria *c = (Categoria *)(it->getCurrent());
		IIterator *it2;
		ICollection *allCodigos = c->getAllCodigos();
		for(it2 = c->getIterator(); it2->hasCurrent(); it2->next())
			{
			res->add(it2->getCurrent());
			}
		delete allCodigos;
		delete it2;
	}
	delete it;
```