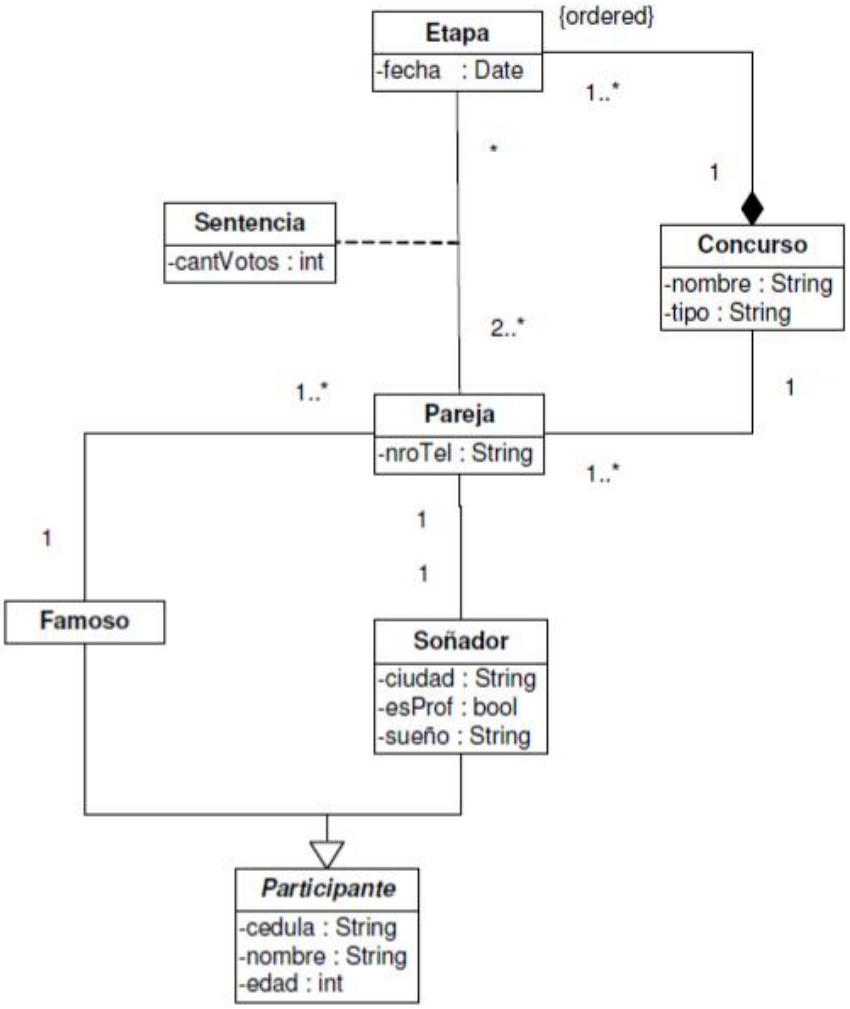
Solución Programación Avanzada Examen Julio 2015

Ejercicio 1)



Restricciones:

context Etapa inv:
-- En una etapa, las parejas sentenciadas deben competir en el concurso al que la etapa pertenece.

```
self.pareja->forAll(p | p.concurso = self.concurso)
```

-- La pareja con menor cantidad de votos en una etapa no participa de una sentencia posterior

```
let eliminada :
Pareja = self.sentencia->select(s1|self.sentencia->forAll(s2|s2 <> s1
implies s1.cantVotos < s2.cantVotos) )->any().pareja in
eliminada.etapa->forAll(e|e.fecha <= self.fecha)
```

context Famoso inv:
-- Un famoso no puede formar parte de más de una pareja en un mismo concurso.

```
self.pareja->forAll(p1, p2 | p1 <> p2 implies p1.concurso <>
p2.concurso)
```

context Concurso inv:
-- Un concurso se identifica por su nombre y tipo.

```
Concurso.allInstances()->forAll(c1, c2| c1 <> c2 implies c1.nombre <>
c2.nombre or c1.tipo <> c2.tipo)
```

-- Para un concurso no puede haber dos etapas con una misma fecha.
`self.etapa->isUnique(fecha)`

-- Para un concurso no puede haber dos parejas con el mismo número de teléfono.
`self.pareja->isUnique(nroTel)`

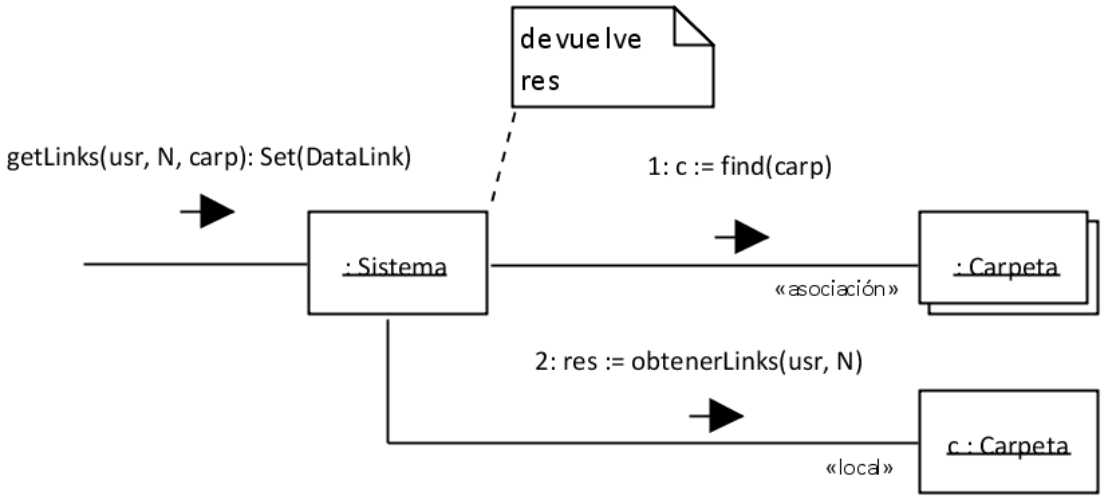
context Participante inv:
-- Un participante se identifica por su cédula.
`Participante.allInstances()->isUnique(cedula)`

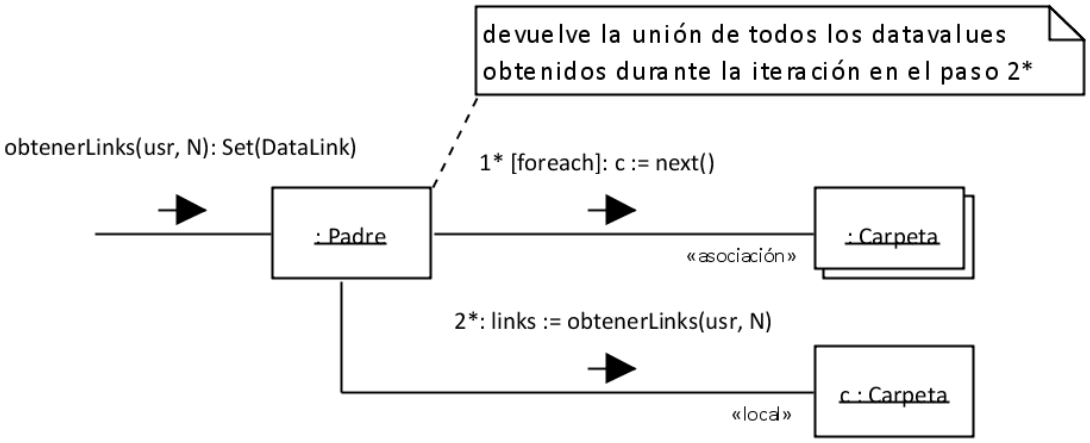
Ejercicio 2)

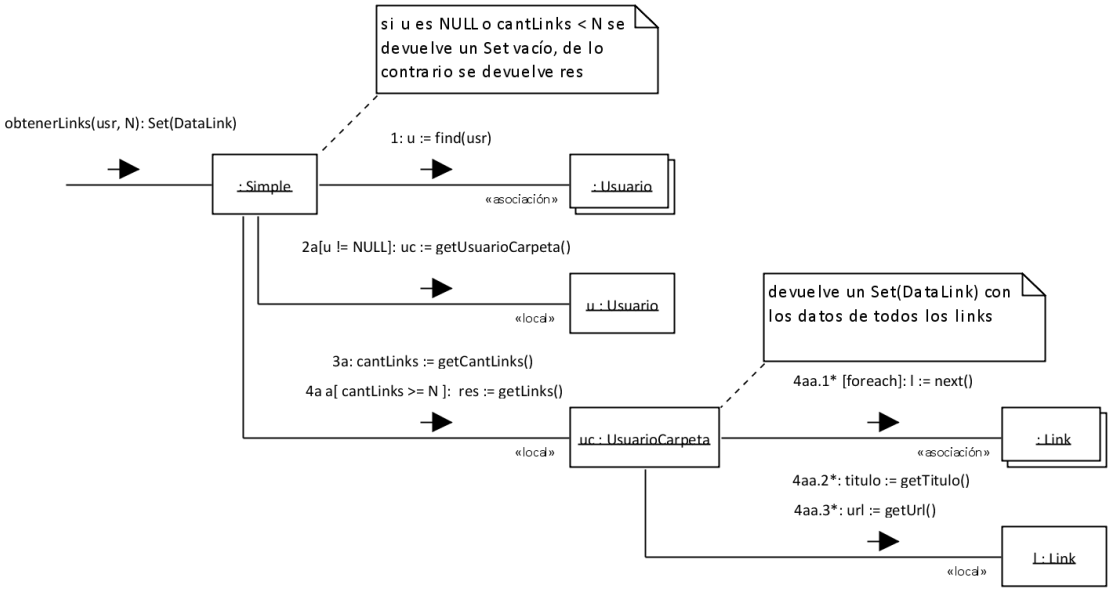
Hay dos alternativas para diseñar la operación `getLinks()`. Una es obtener los links buscando en la carpeta de nombre `carp` y luego navegar por el subárbol de los hijos buscando para cada carpeta simple, si el usuario subió N links en cada carpeta que se itera. La segunda alternativa consiste en buscar 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.

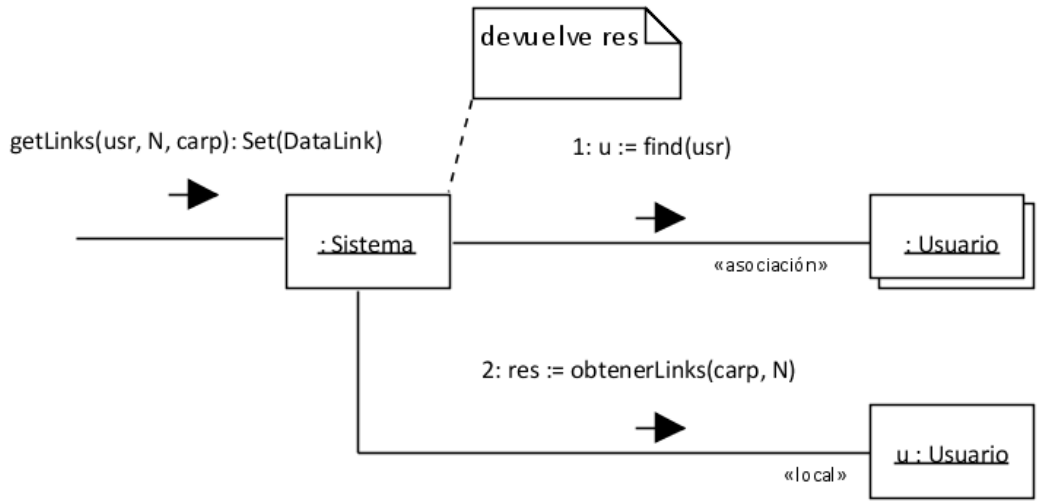
Solución 1

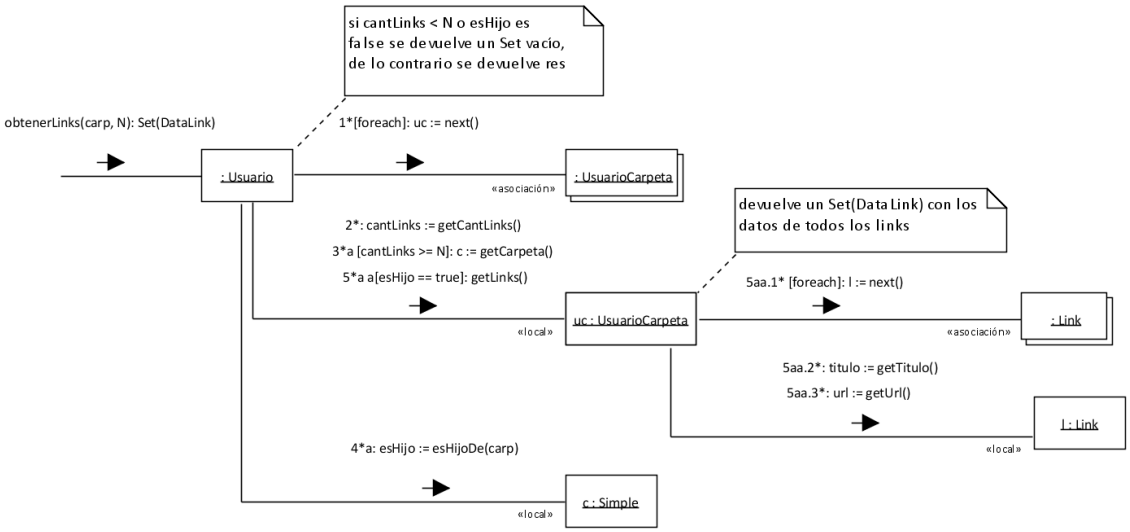


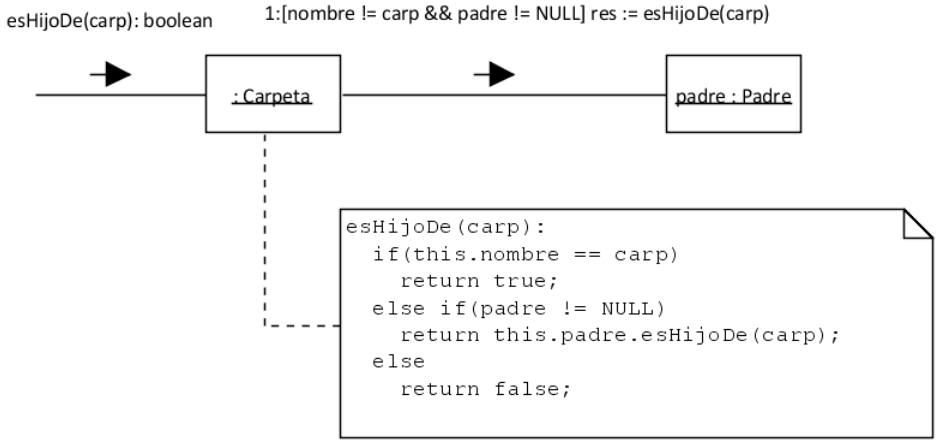

devuelve la unión de todos los datavalues obtenidos durante la iteración en el paso 2*
obtenerLinks(usr, N): Set(DataLink)
1* [foreach]: c := next()
: Padre
: Carpeta
«asociación»
2*: links := obtenerLinks(usr, N)
c : Carpeta
«local»


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
: Usuario
«asociación»
2a[u != NULL]: uc := getUsuarioCarpeta()
u : Usuario
«local»
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()
uc : UsuarioCarpeta
«local»
: Link
«asociación»
4aa.2*: titulo := getTitulo()
4aa.3*: url := getUrl()
l : Link
«local»

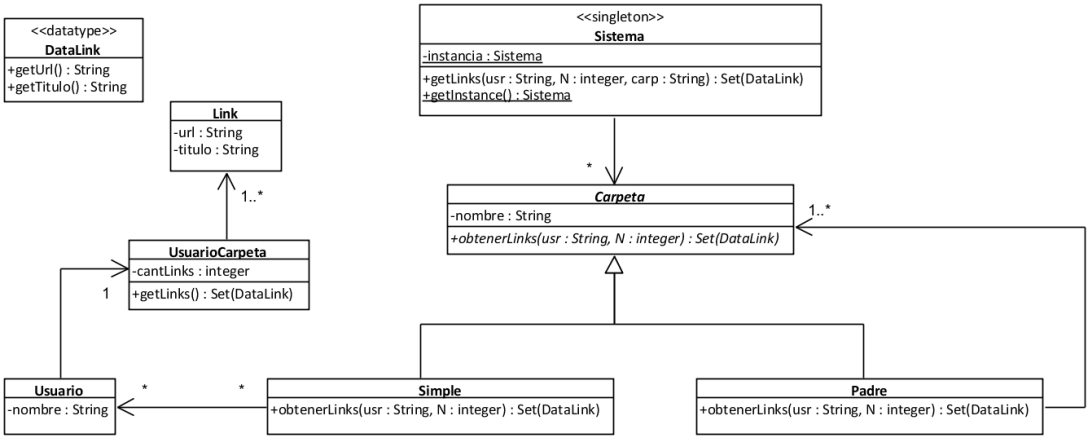
Solución 2


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

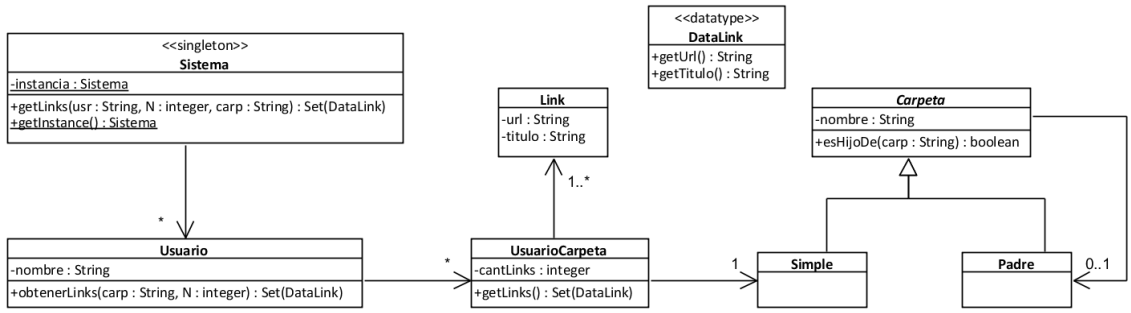

si cantLinks < N o esHijo es
false se devuelve un Set vacío,
de lo contrario se devuelve res
obtenerLinks(carp, N): Set(DataLink)
1*[foreach]: uc := next()
: Usuario
«asociación»
: UsuarioCarpeta
devuelve un Set(DataLink) con los
datos de todos los links
2*: cantLinks := getCantLinks()
3*a [cantLinks >= 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.
Solución 1 (correspondiente a la solución 1 de la parte i)



Solución 2 (correspondiente a la solución 2 de la parte i)



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() {
}
```