Programación 4

SOLUCIÓN EXAMEN DICIEMBRE 2012

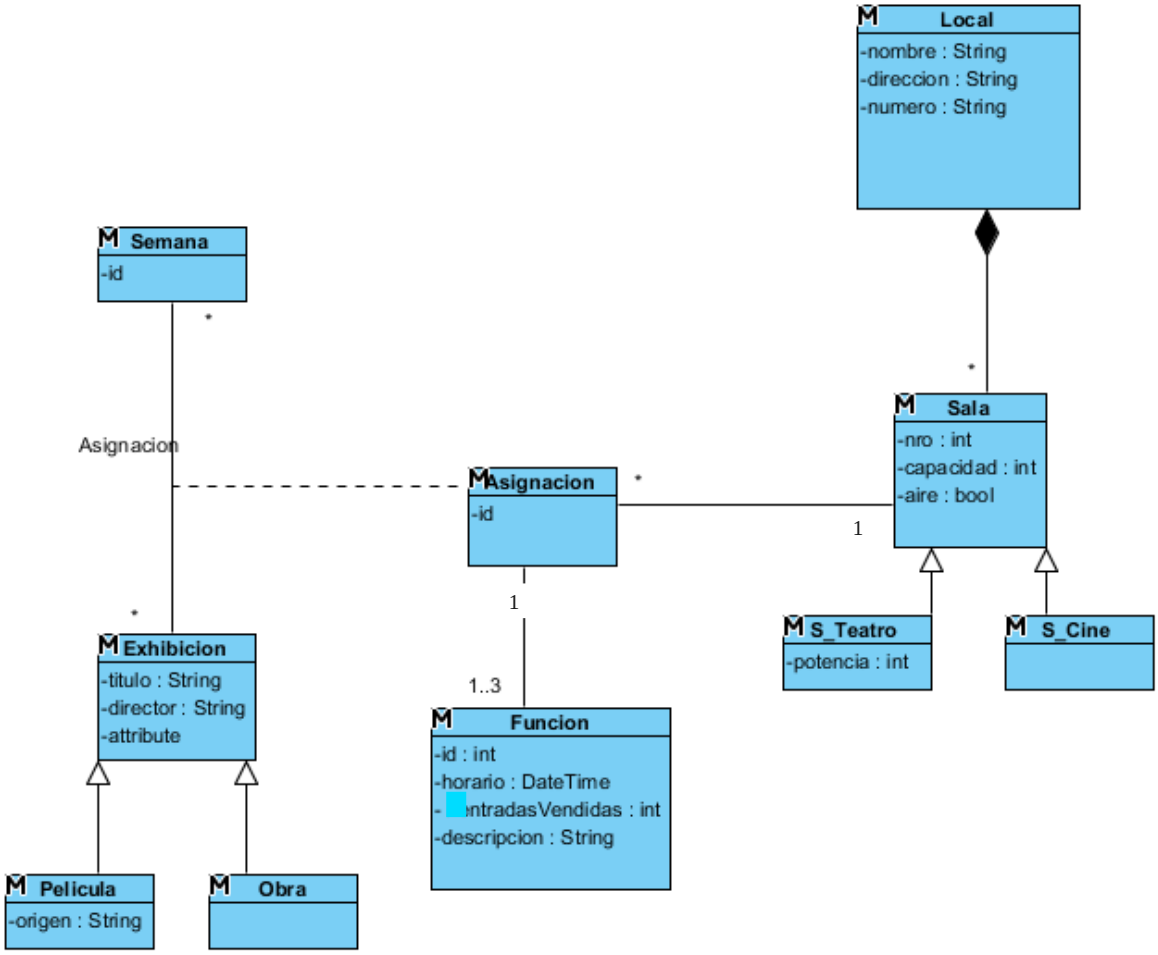
Problema 1 (30 puntos)

a. Indique los componentes necesarios para la especificación de un Caso de Uso Expandido (Ver diapositivas de Requerimientos).

- Nombre que identifica al caso de uso
- Actores participantes en el caso de uso
- Sinopsis que describe brevemente suobjetivo
- Curso típico de eventos que narra la"historia" más común de los actores durante el uso del sistema
- Cursos alternativos de eventos que narran las variantes de uso del sistema

b. Se desea implementar un software de gestión para una empresa de salas de entretenimiento. La empresa cuenta con múltiples locales que se identifican por un nombre y de los cuales se registra su dirección y número de teléfono. Cada local cuenta con múltiples salas de cine y teatro. Se conoce la capacidad de cada una de las salas, si poseen aire acondicionado o no y, para las salas de teatro, la potencia del sonido medida en KW. La asignación de las obras y películas exhibidas en cada sala (teatro y cine, respectivamente) se realiza en base semanal. Cada semana es conocida por un número del 1 al 52. Una obra o película en una semana se asigna a una única sala pudiéndose exhibirse en ella hasta en 3 funciones por semana. Una función es la exhibición de una obra o película en un día y horario determinado. Para cada una de las funciones interesa registrar la cantidad de entradas vendidas. De cada película y obra se conoce su titulo y director y en particular para las películas interesa saber su origen.

Se pide: Modelo de Dominio de la realidad anterior con restricciones en lenguaje natural.



Restricciones:

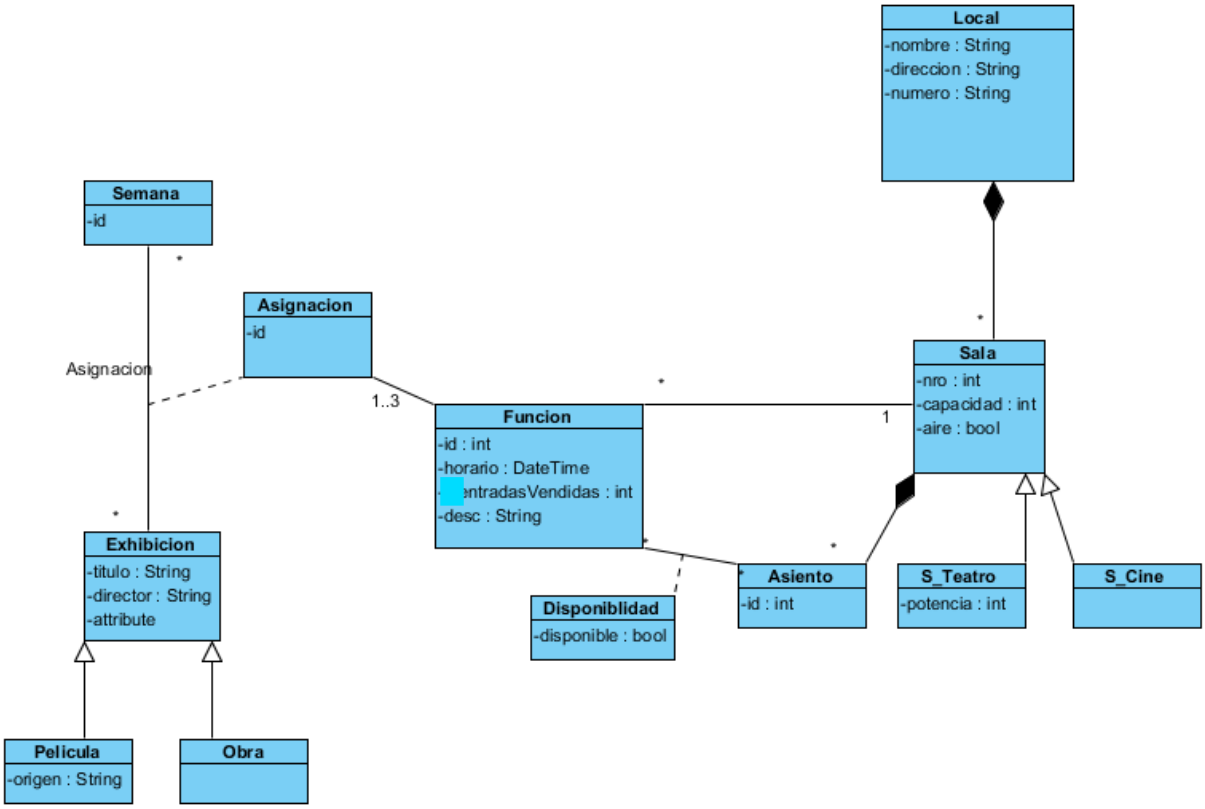
-Unicidad de Semana.id ([1..52]) y Local.nombre. No exigidas: Asignacion.id,Exhibicion.titulo, Funcion.id y para cada Local l, unicidad de Sala.id

-La cantidad de entradas vendidas en una función no puede superar la capacidad de la sala asignada.

-La exhibiciones del tipo película y obra solo podrán estar asociadas a Salas de cine y teatro respectivamente.

c. Se quiere optimizar el uso de las salas por lo que se desea asignar distintas salas a las funciones semanales asignadas a una obra o película y no en una única sala asignada semanalmente como se restringe en (c). Además debido a las numerosas quejas por colas innecesarias se desea numerar todas las entradas vendidas por lo que el sistema deberá permitir registrar y consultar la disponibilidad de cada asiento para cada función.

Se pide: Modelo Dominio con modificaciones necesarias para cumplir con la realidad anterior y restricciones en lenguaje natural.



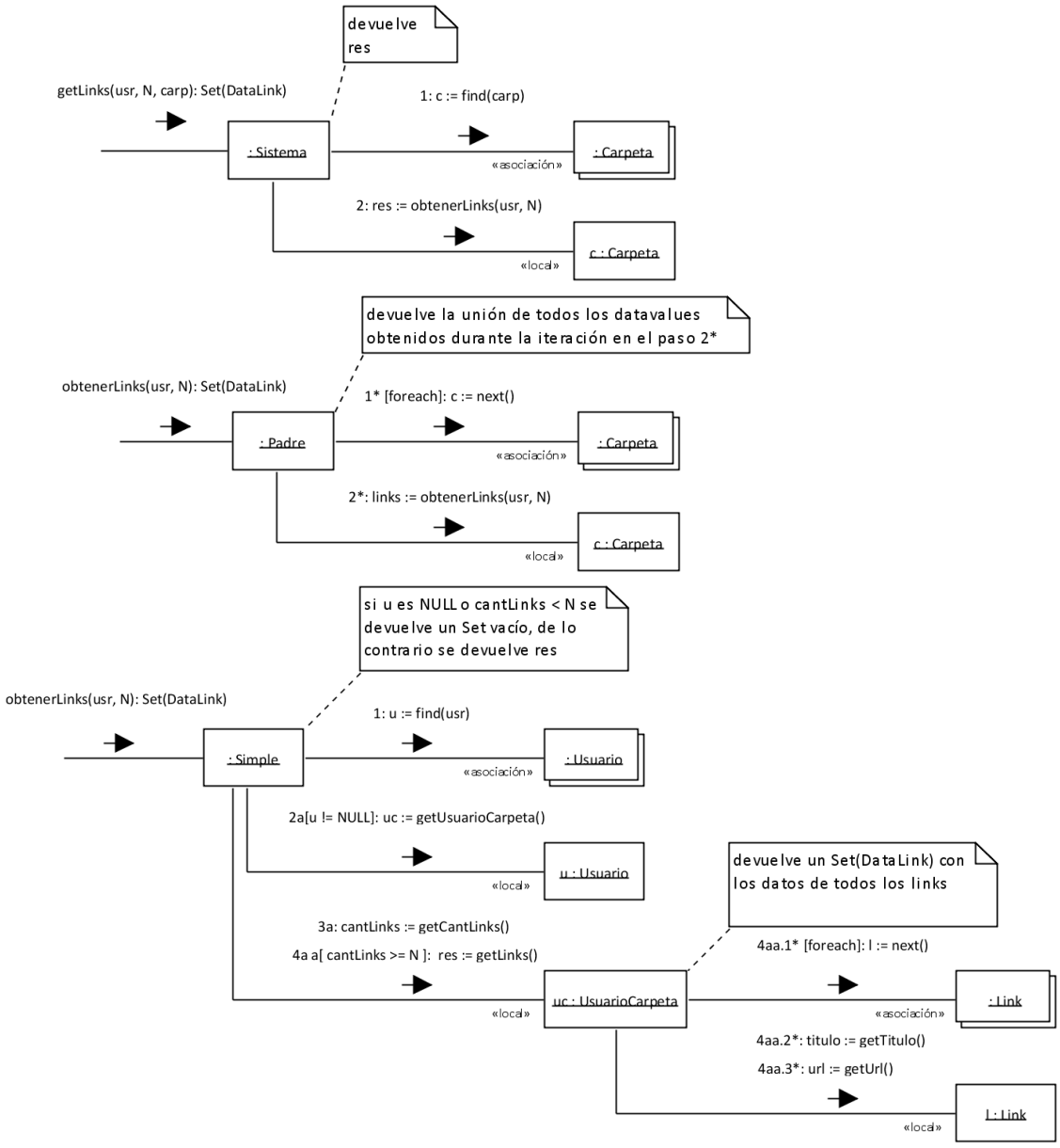
Se agregan las siguientes restricciones:
-La cantidad de asientos asociados a una función no puede superar la capacidad de la sala asignada.
-Si la asociación de Función-Sala se incluye por comprensión, entonces todos los asientos asignados a la Función deberán pertenecer a la misma sala asignada a la función.

Problema 2 (35 puntos)

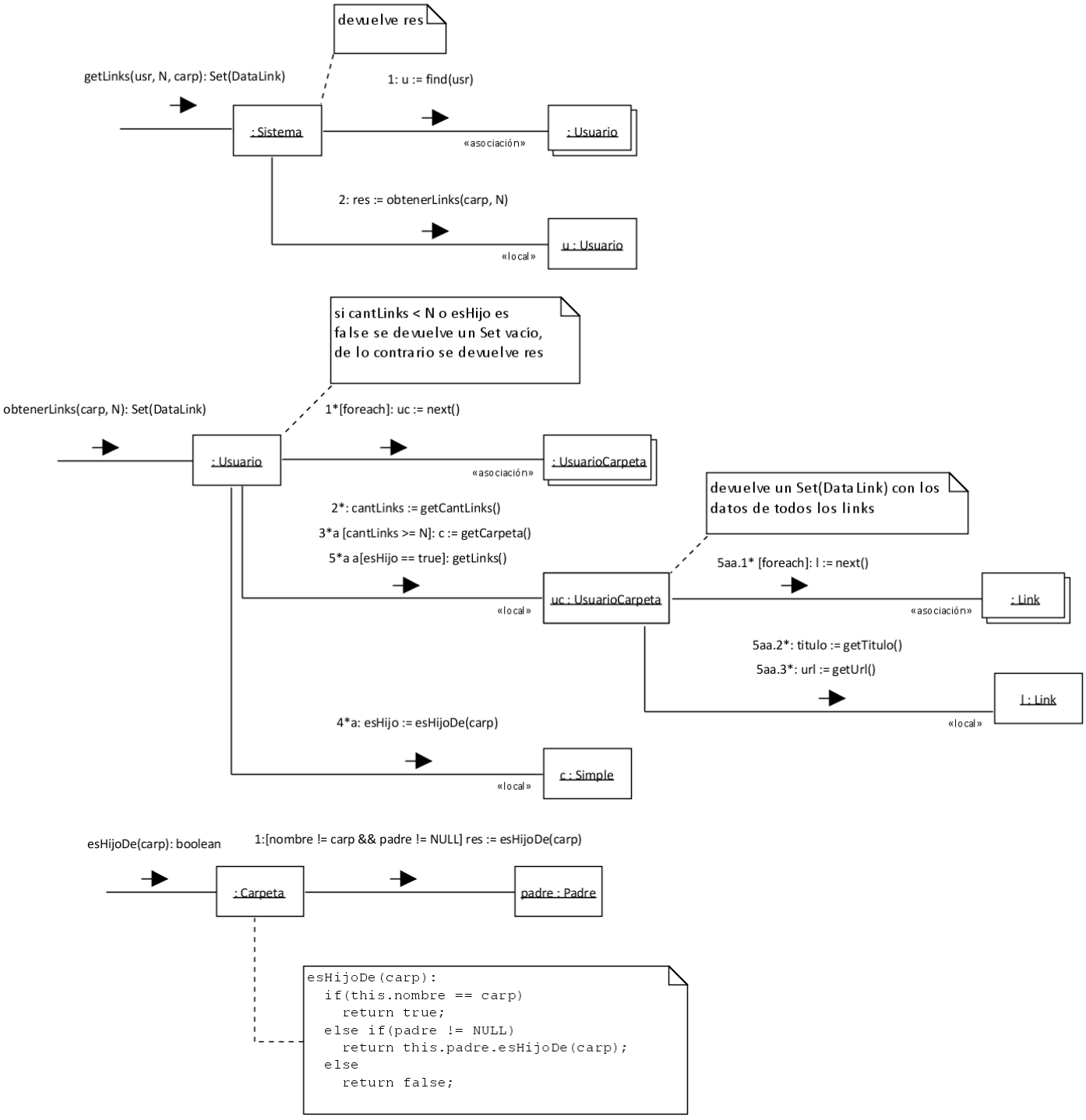
Hay dos alternativas para diseñar la operación `getLinks()`. Una es obtener los links buscando en la carpeta de nombre `carp` y luego navegar por el subárbol de los hijos buscando para cada carpeta simple, si el usuario subió N links en cada carpeta que se itera. La segunda alternativa consiste en buscar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

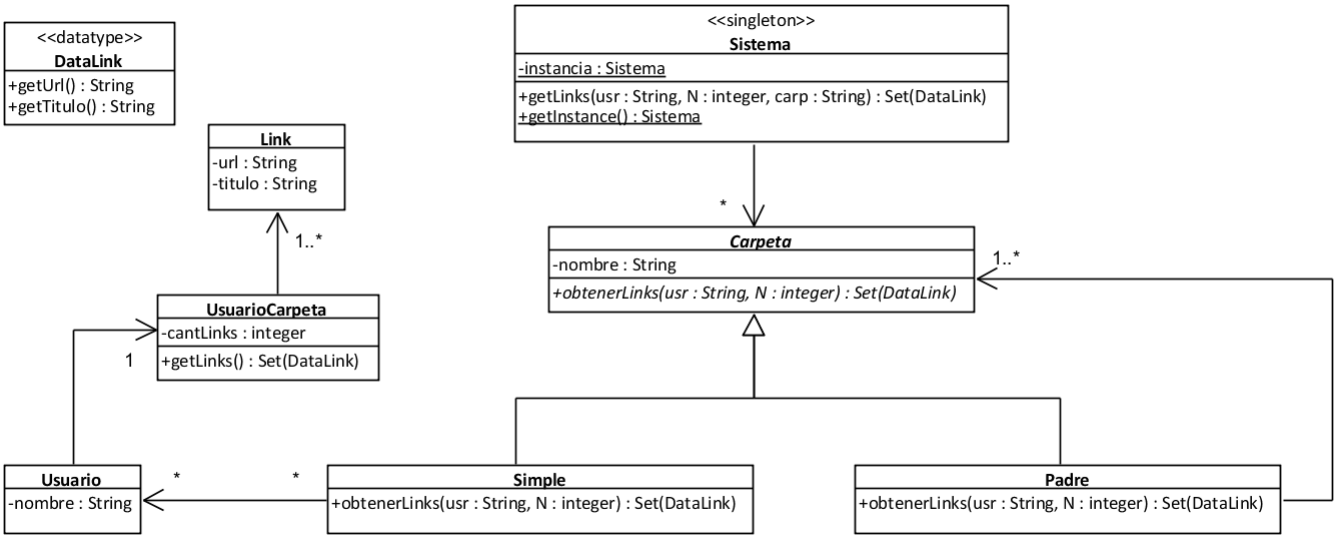


Solución 2

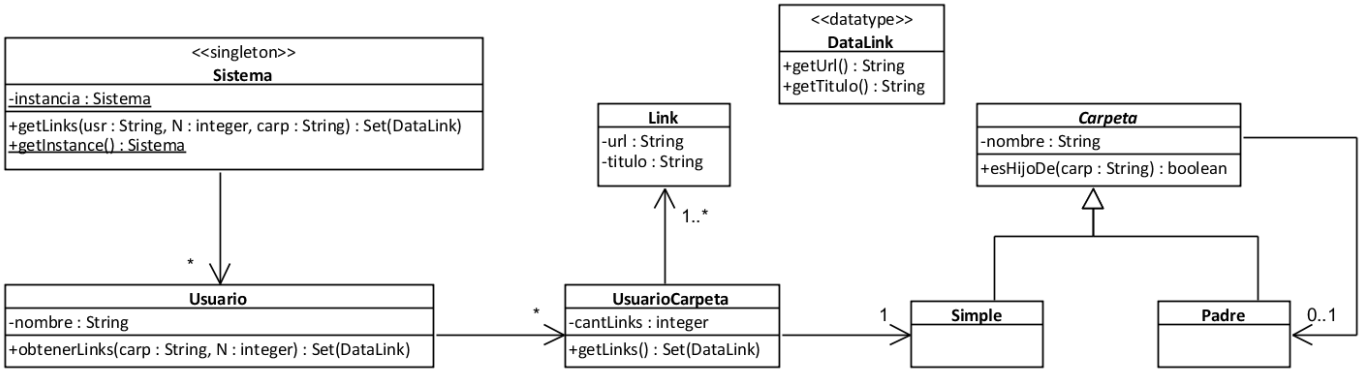

devuelve res
getLinks(usr, N, carp): Set(DataLink)
1: u := find(usr)
: Sistema
«asociación»
: Usuario
2: res := obtenerLinks(carp, N)
«local»
u : Usuario
si cantLinks < N o esHijo es false se devuelve un Set vacío, de lo contrario se devuelve res
obtenerLinks(carp, N): Set(DataLink)
1*[foreach]: uc := 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)



Problema 3 (35 puntos)

a)

```
// CacheManager.h
class CacheManager : public ServerBase
{
  private:
    Server *server;
    IDictionary *resources;

  public:
    CacheManager(Server* server);
    String get(String url);
    String find(String url);
    void update(String url, String content);

    bool cache_hit(String url);

    virtual ~CacheManager();
}

// CacheManager.cpp

CacheManager::CacheManager(Server* server)
{
  this->server = server;
  this->resources = new Dictionary();
}

String CacheManager::get(String url)
{
  if(this->cache_hit(url)){
    return this->find(url);
  }
  else{
    String content = server->get(url);
    this->update(url, content);
    return content;
  }
}

String CacheManager::find(String url)
{
  IKey *key = new KeyString(url);

    CachedResource *resource = (CachedResource*) this->resources-
>find(key);

  delete key;
  return resource->getContent();
}

void CacheManager::update(String url, String content)
{
```

```
  IKey *key = new KeyString(url);

  bool cached = this->resources->member(key);
  int expiration = TimestampClock::expiration_time();

  CachedResource* resource;

  if(cached)
  {
    resource = (CachedResource*) this->resources->find(key);
    resource->setContent(content);
    resource->setExpiration(expiration);

    delete key;
  }

  else
  {
    resource = new CachedResource(url, content, expiration);
    this->resources->add(key, resource);
  }
}

bool CacheManager::cache_hit(String url)
{
  IKey *key = new KeyString(url);

  bool cached = this->resources->member(key);
  if(!cached){
    delete key;
    return false;
  }
  else
  {
    CachedResource* resource = (CachedResource*) this->resources-
>find(key);

    delete key;

    int expiration = resource->getExpiration();
    int now = TimestampClock::now();
    return (expiration>now);
  }
}

CacheManager::~CacheManager()
{
  IIterator *elems = this->resources->getElemIterator();
  IIterator *keys = this->resources->getKeyIterator();

  while(elems->hasCurrent()){

   CachedResource *c = (CachedResource*) elems->getCurrent();
```

```
    delete c;
  }

  delete elems;

  while(keys->hasCurrent()){
    IKey *k = keys->getCurrent();

    delete k;
  }

  delete keys;
  delete resources;
}

//ServerBase.h

class ServerBase
{
  public:
    virtual String get(String url) = 0;
    virtual ~ServerBase();
};


//ServerBase.cpp

ServerBase::~ServerBase()
{}
```

b)
Se utiliza Proxy, con los siguientes roles.
Subject: ServerBase
Proxy: CacheManager
Real Subject: Server