Programación 4

PARCIAL FINAL EDICIÓN 2008

SOLUCIÓN

Problema 1 (30 puntos)

a) Responda en no más de 3 líneas cada una de las siguientes preguntas

i) ¿Cuál es el objetivo y qué actividades se realizan en la etapa de Análisis?

El objetivo de la etapa de análisis es modelar el dominio del problema y especificar el comportamiento del sistema como caja negra. Para ello se realizan las actividades de Modelado de Dominio y Especificación del Comportamiento del Sistema, respectivamente.

ii) ¿Qué es una agregación y qué tipos de agregaciones existen?

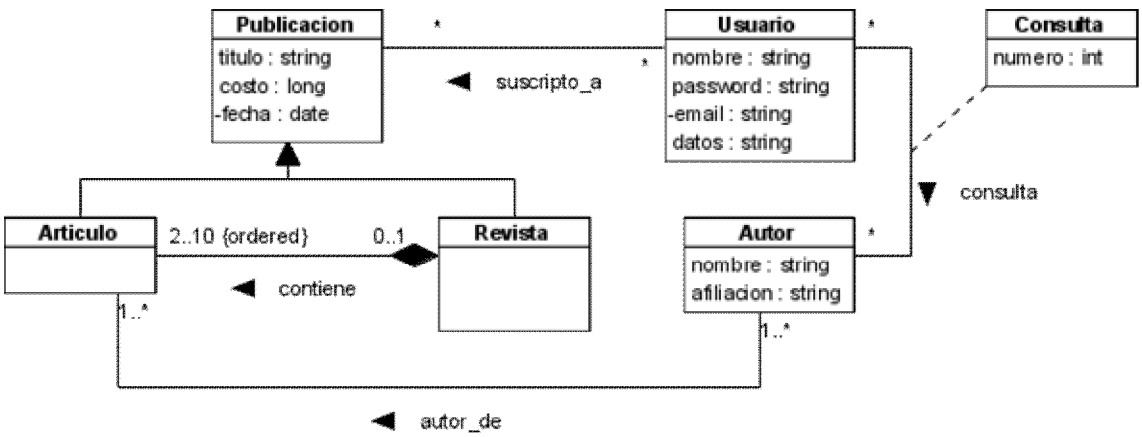
Es una forma más fuerte de asociación que significa que un elemento es parte de otro. Existen dos variantes: Agregación Compartida (agregación) y Agregación Compuesta (composición).

iii) ¿Qué es un evento del sistema?

Un evento del sistema es un estímulo externo, generado por un actor, ante el cual el sistema debe reaccionar.

b) Se está desarrollando un sitio web con un sistema de suscripción a publicaciones electrónicas.

i) Construir el Modelo de Dominio para la realidad descripta y presentarlo en un diagrama utilizando UML. Las restricciones deben ser expresadas en lenguaje natural y en OCL. Modelar exclusivamente en base a la información presente en la descripción y los casos de uso.

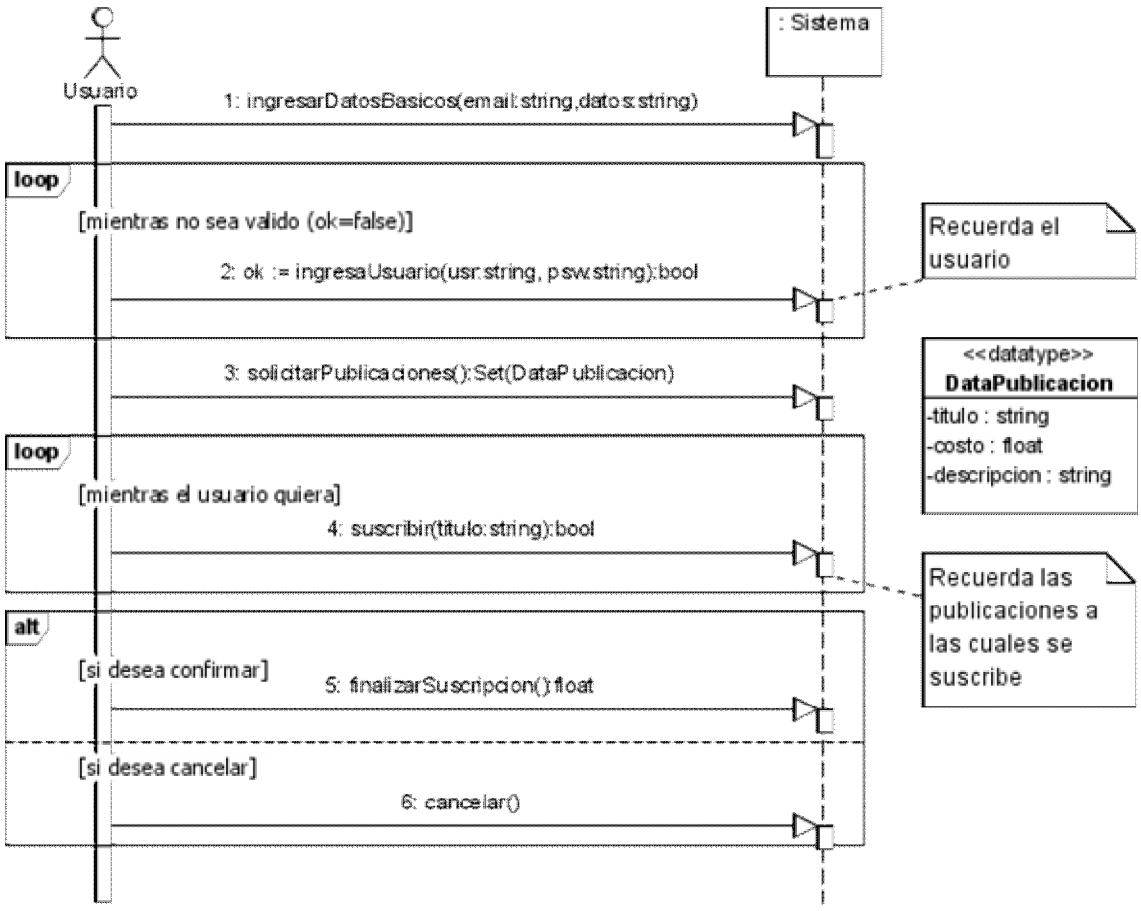


Restricciones

```
-- El título de la publicación es único.
context Publicacion inv:
   Publicacion.allInstances()->isUnique(titulo)
```

```
-- El nombre del usuario es único.
context Usuario inv:
   Usuario.allInstances()->isUnique(nombre)

-- La fecha de publicación de un artículo es la misma de la
-- revista en la que aparece
context Revista inv:
   self.articulo->forAll(a:Articulo | a.fecha = self.fecha)


-- El costo de una revista es siempre menor o igual a la
-- suma del costo de los artículos que contiene
context Revista inv:
   self.costo <= self.articulo.costo->sum()


-- Un usuario no puede estar al mismo tiempo suscripto a un
-- artículo y a la revista que lo contiene
context Usuario inv:
   self.publicacion->select(p:Publicación |
      p.oclIsTypeOf(Articulo))->forAll(a:Publicacion |
         not p.oclAsType(Articulo).revista.usuario->includes(self))
```

ii) Realice <u>un único</u> Diagrama de Secuencia de Sistema para el caso de uso Suscripción a Publicaciones, incluyendo <u>toda</u> la información contenida en el mismo.



Problema 2 (40 puntos)

a) Responda en no más de 3 líneas cada una de las siguientes preguntas

i) ¿A qué se le llama ***visibilidad***?

La visibilidad es la capacidad de un objeto de tener una referencia a otro.

ii) ¿Qué tipos de visibilidad se vieron en el curso?

Se vieron 4 tipos: local, global, por parámetro y por atributo.

iii) Describa dos tipos de visibilidad.

Un objeto A tiene visibilidad por atributo sobre un objeto B si B es un pseudoatributo de A.
Un objeto A tiene visibilidad por parámetro sobre un objeto B si B es un parámetro de un método de A.
Un objeto A tiene visibilidad local sobre un objeto B si B es declarado localmente en un método de A.
Un objeto A tiene visibilidad global sobre un objeto B si B es visible en forma global.

b)

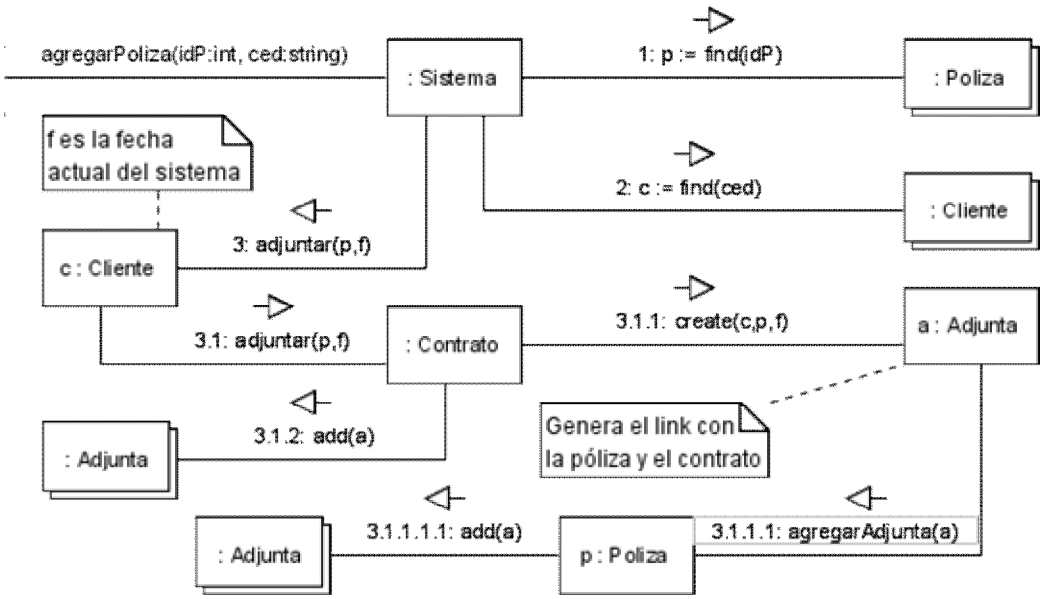
i) Completar los contratos de las operaciones incluyendo las pre y post condiciones correspondientes. Considere exclusivamente la información dada en el problema.

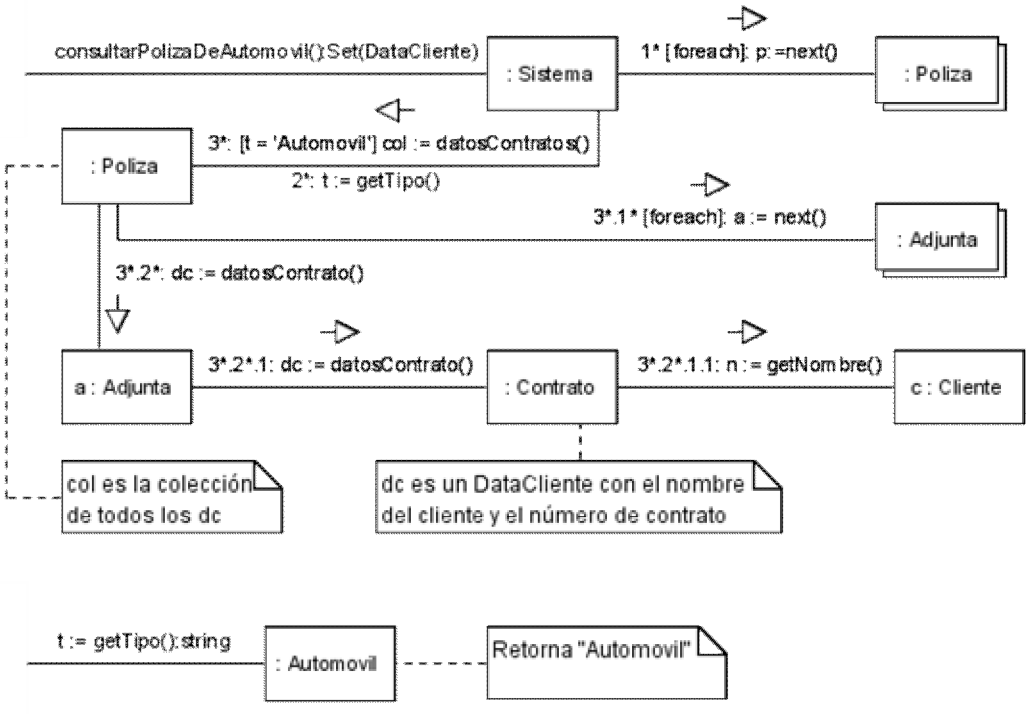
Nombre	Incluir póliza en contrato de un cliente
Operación	`agregarPoliza(idP:int, ced:string)`
Descripción	Agrega la póliza de identificador `idP` al contrato del cliente de cédula `ced` utilizando la fecha actual del sistema.
Pre	- La póliza con identificador idP existe - El cliente con cédula ced existe y tiene un contrato asociado - El contrato del cliente no tiene dicha póliza asociada
Post	- Se creó una instancia de la clase de asociación Adjunta con la fecha actual como atributo y se generó el link entre el contrato del cliente y la póliza por medio de dicha clase de asociación

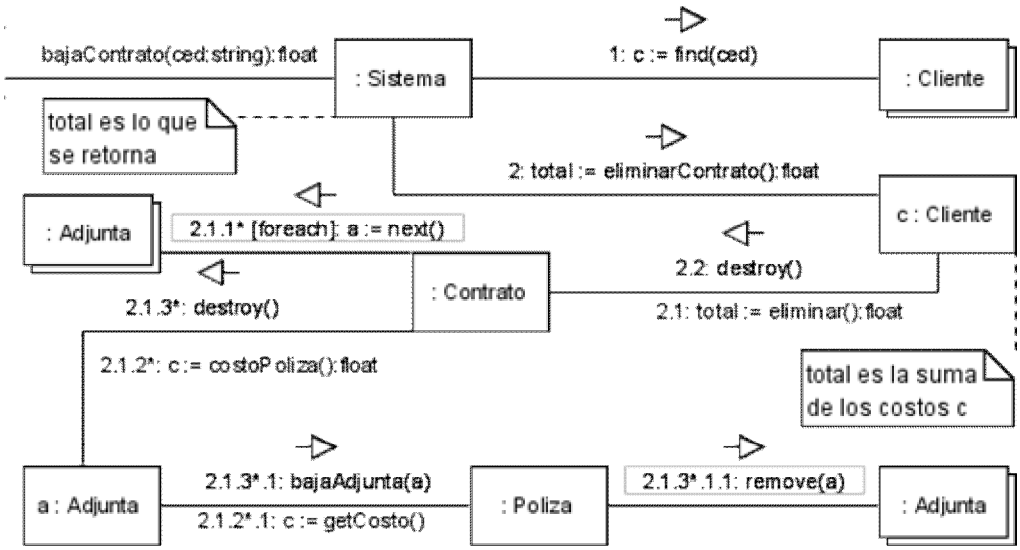
Nombre	Consultar clientes con póliza de automóvil
Operación	`consultarPolizaDeVida():Set(DataCliente)`
Descripción	Retorna un conjunto de datavalues compuestos por el número de contrato y el nombre de cliente que posean alguna póliza de automóvil
Pre	
Post	- Retorna una colección de DataCliente con el número de contrato y el nombre de cliente que posean alguna póliza de automóvil

Nombre	Dar de baja un contrato
Operación	`bajaContrato(ced:string):flota`
Descripción	Elimina el contrato del cliente de cédula `ced` y a su vez retorna el costo total de las pólizas que incluía dicho contrato.
Pre	- El cliente con cédula ced existe y tiene un contrato asociado
Post	- Se eliminó la instancia de contrato asociada con el cliente - Se eliminó el link entre el cliente y el contrato - Se eliminaron las instancias de Adjunta asociadas con el contrato - Se eliminaron los links entre las instancias de Adjunta y las pólizas - Se retornó el monto total de las pólizas que incluía el contrato

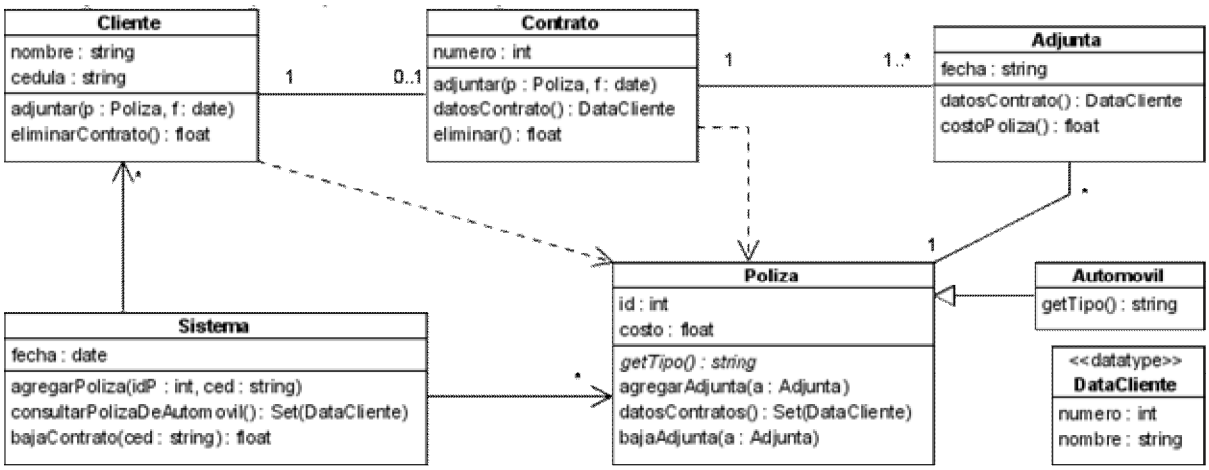
ii) Realizar un Diagrama de Comunicación para cada una de las operaciones respetando los contratos descritos. Explicitar la eliminación de instancias.







iii) Realizar el Diagrama de Clases de Diseño incluyendo <u>toda</u> la información contenida en los diagramas.



Problema 3 (30 puntos)

PARTE A:

Se pide:

i) Implementar en C++ la interfaz de un iterador de recursos para IColRecursos (siguiendo el patrón de diseño iterator). No es necesario incluir operaciones para remover elementos en el iterador.

```
// IIterator.h
class IIterator
{
public:
   virtual Recurso* getCurrent() = 0;
   virtual bool hasCurrent() = 0;
   virtual void next() = 0;
   virtual ~IIterator();
};

// IIterator.cc
IIterator::~IIterator(){}
```

ii) Implementar en C++ una realización de la interfaz del iterador (definida en el punto i) para la clase VectorRecursos. No se puede modificar la clase VectorRecursos, con la salvedad de que se deberá agregar la operación que devuelve el iterador, ni asumir nada sobre su definición más allá de lo presentado en el diagrama.

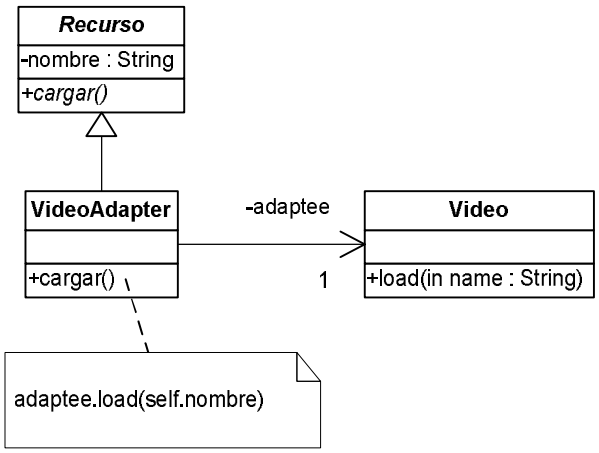
```
// VectorIterator.h
class VectorIterator: public IIterator
{
private:
   Vector* vector;
   int actual;
public:
   VectorIterator(Vector*);
   Recurso* getCurrent();
   bool hasCurrent();
   void next();
   ~VectorIterator();
};

// VectorInterator.cc
VectorIterator::VectorIterator(Vector* v): vector(v), actual(0) {};
Recurso* VectorIterator::getCurrent()
{
   return this->vector->getElement(this->actual);
```

```
};
bool VectorIterator::hasCurrent()
{
   return this->actual < this->vector->getSize();
};
void VectorIterator::next() { this->actual++; };
VectorIterator::~VectorIterator() {};
```

PARTE B:

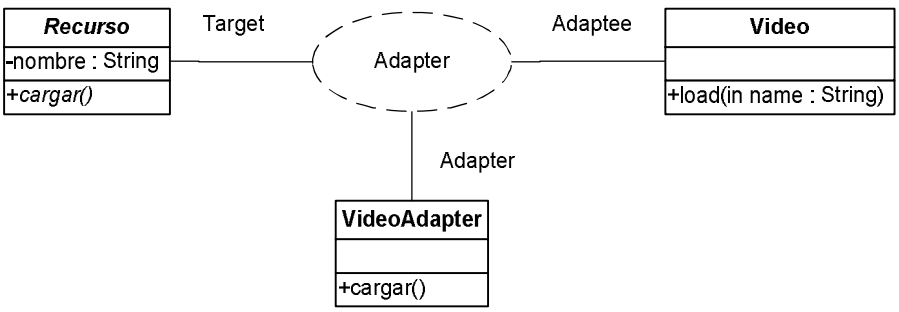
Se pide:

i) Realizar un DCD para el mecanismo mencionado aplicando patrones de diseño.



ii) Explicar qué patrón(es) de diseño utilizó indicando (para cada patrón) las clases participantes y sus roles.

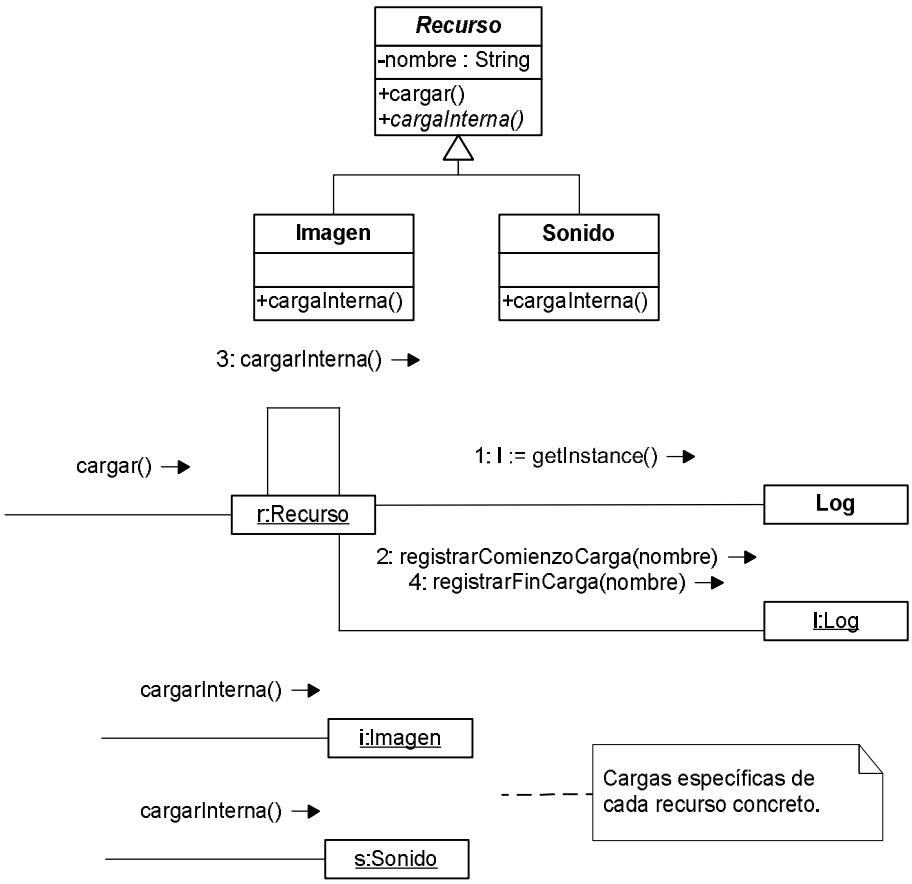
Se aplicó el patrón Adapter, ya que el tipo de la clase Video no es compatible con Recurso y la clase Video no puede modificarse.



PARTE C:

Se pide:

i) Realizar el diseño completo (DCD y comunicaciones) del mecanismo mencionado aplicando patrones de diseño.



ii) Explicar qué patrón(es) de diseño utilizó indicando (para cada patrón) las clases participantes y sus roles.

Se aplicó Template Method ya que el mecanismo de log pedido agrega una parte común al algoritmo de carga de todo recurso concreto.