Programación 4

EXAMEN - JULIO 2007

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 examen junto al examen

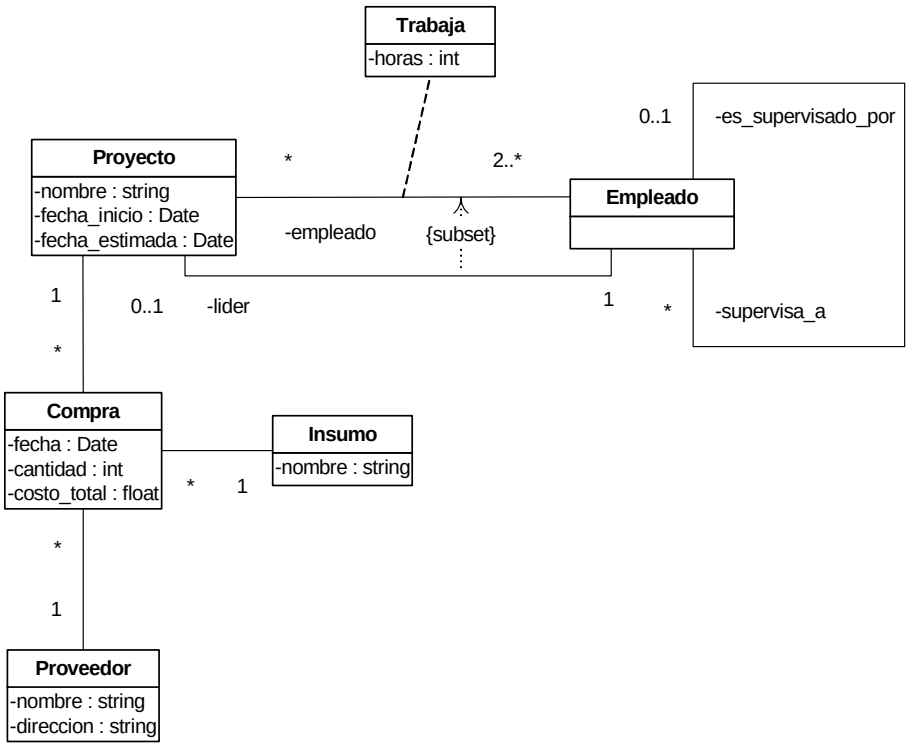
Problema 1 (25 puntos)

a) Mencione los subtipos del tipo `Collection` de OCL y explique en **no más de 5 líneas** en qué se diferencian.

Se distinguen tres subtipos de *Collection*: *Set*, *Sequence* y *Bag*. Se corresponden con los tipos abstractos *Conjunto*, *Secuencia* y *Bolsa*, respectivamente.

Un valor de *Set* es una colección de elementos donde sus elementos no se repiten y no existe un orden entre ellos. Un valor de *Sequence* es una colección de elementos donde sus elementos se pueden repetir y existe un orden lineal entre ellos. Un valor de *Bag* es una colección de elementos donde sus elementos se pueden repetir pero no existe un orden entre ellos.

b) Construya un Diagrama de Modelo de Dominio UML para la siguiente realidad. Las restricciones deben ser expresadas en lenguaje natural y en OCL. Modele **exclusivamente** en base a la información presente en la descripción.



```
context Proyecto inv:
-- El nombre de los proyectos es identificador.
Proyecto.allInstances->isUnique(nombre)

context Empleado inv:
-- Un empleado no puede estar asignado más de 10 horas entre todos
-- los proyectos en los que participa.
self.trabaja.horas->sum() <= 10
```

context Empleado **inv**:
-- Un empleado si es líder no trabaja en más de un proyecto

```
self.lider->notEmpty() implies self.empleado->size() = 1
```

context Empleado **inv**:
-- Un empleado líder no es supervisado por alguien más.

```
self.lider->notEmpty() implies self.es_supervisado_por->empty()
```

context Empleado **inv**:
-- Todo empleado no líder debe tener un supervisor

```
self.lider->empty() implies self.es_supervisado_por->notEmpty()
```

context Empleado **inv**:
-- Un empleado no puede ser supervisor directo de él mismo.

```
self.supervisa_a->select(e | e = self)->empty()
```

context Compra **inv**:
-- Un mismo proyecto puede comprar el mismo insumo al mismo proveedor solamente en
-- fechas diferentes.

```
Compra.allInstances->forAll(c1, c2 | c1 <> c2 implies
        c1.proyecto <> c2.proyecto or c1.proveedor <> c2.proveedor
        or c1.insumo <>c2.insumo or c1.fecha <> c2.fecha)
```

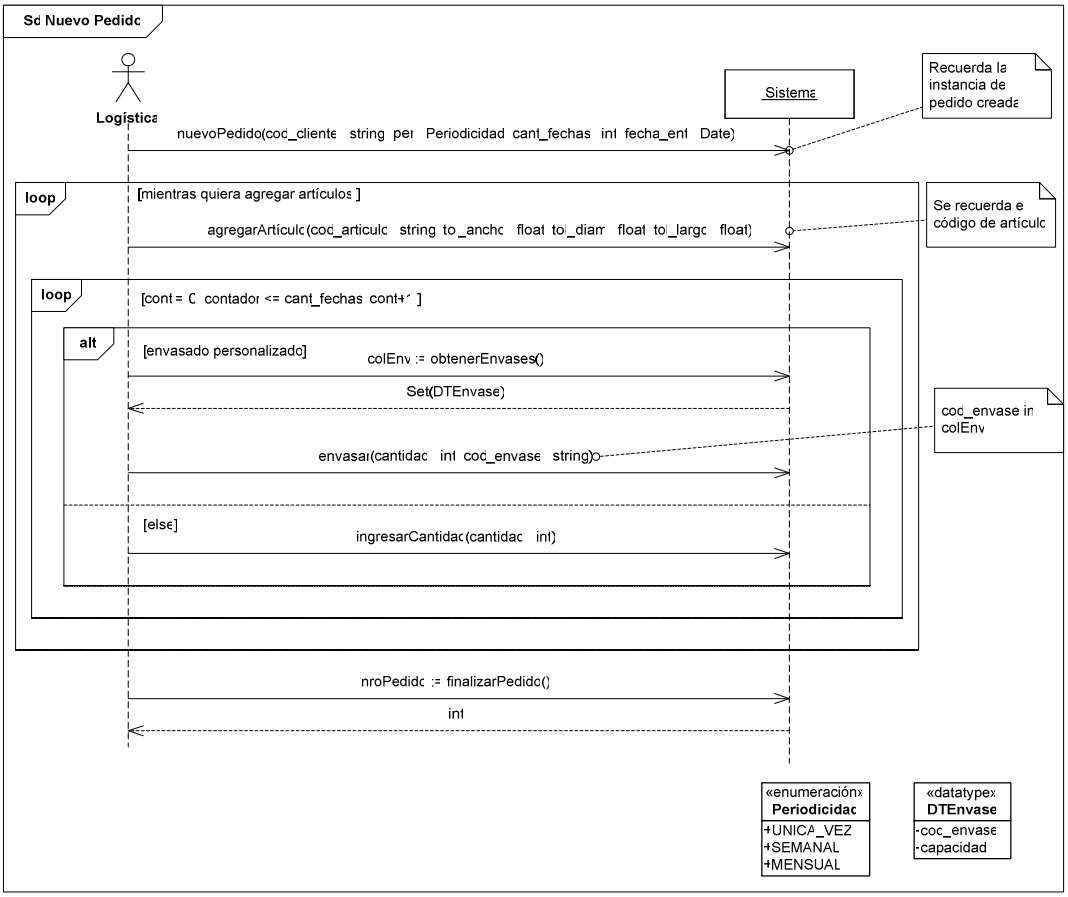
Problema 2 (25 puntos)

a) Definir Contrato de Software e indicar **brevemente** cuáles son las responsabilidades del proveedor y el consumidor del mismo.

Un *Contrato de Software* es un documento que especifica el comportamiento o efecto de una operación. El *proveedor* debe satisfacer las postcondiciones en caso que se hallan cumplan las precondiciones e la operación. El *consumidor* debe satisfacer las precondiciones, por lo tanto debe saber en qué momento invocar la operación.

b)

i. Realizar un único Diagrama de Secuencia de Sistema para el caso de uso anterior. Incluya los tipos de los parámetros y valores de retorno de las operaciones.



ii. Indicar las precondiciones y postcondiciones de los contratos de software de las operaciones del sistema identificadas en el DSS de la parte anterior, exceptuando lo que corresponde al envasado automático de los artículos.

void nuevoPedido(cod_cliente : string, per : Periodicidad, cant_fechas : int, fecha_ent : Date)

pre:

- Existe una instancia de cliente con el código = *cod_cliente*.

post:

- Se crea una instancia de Pedido con la periodicidad, fecha de primera entrega y cantidad de fechas especificadas por parámetro. El valor del número de pedido es generado por el sistema.
- Se crea un link entre la instancia de Pedido y la instancia de Cliente de código *cod_cliente*.
- Se crean instancias de Entrega para cada fecha de entrega. El valor fecha de cada instancia se setea con las fechas de entrega calculadas por el sistema.
- Se crea un link entre la instancia de pedido creada y cada una de las instancias de Entrega creadas.
- El sistema crea un link a la instancia de Pedido identificada por el número generado, identificado como el pedido actual.

void agregarArtículo(cod_articulo : string, tol_ancho : float, tol_diam : float, tol_largo : float)

pre:

- El sistema tiene una instancia de pedido como pedido actual.
- Existe una instancia de EspArticulo de código = cod_articulo.
- No existe un link entre una instancia de ArticuloPedido del pedido actual y la instancia de EspArticulo de código = *cod_artículo*.

post:

- Se crea una instancia de ArticuloPedido con los valores en los atributos de tolerancias según se especifica por parámetro.
- Se crea un link entre el pedido actual y la instancia de ArticuloPedido creada.
- Se crea un link entre la instancia de EspArticulo de código = *cod_artículo* y la instancia de ArtículoPedido creada.
- El sistema mantiene en memoria el código de artículo *cod_articulo*.

Set(DTEnvase) obtenerEnvases()

pre:

- El sistema tiene en memoria el código de artículo.

post:

- Se retorna un conjunto de *datavalues* de DTEnvase, cada uno con los valores de código y capacidad de las instancias de Envase linkeadas a la instancia de TipoArticluo que está linkeada a su vez con la instancia de EspArticulo de código recordado por el sistema.

void envasar(cantidad : int, cod_envase : string)

pre:

- El sistema tiene una instancia de pedido como pedido actual.
- El sistema tiene en memoria el código de artículo cod_articulo.
- Existe una instancia de Envase identificada por *cod_envase*.

post:

- Se crea una instancia de ArticuloFecha con la cantidad especificada por parámetro, linkeandose a la instancia de ArticuloPedido de código cod_articulo y a la instancia de Entrega con valor de fecha de entrega que corresponde a la secuencia del pedido actual.
- Se crea un link entre la instancia de ArticuloFecha creada y la instancia de Envase de código *cod_envase*.
- El sistema deja de recordar el código de artículo.

int finalizarPedido()

pre:

- El sistema tiene una instancia de pedido como pedido actual.

post:

- Se retorna el número de pedido del pedido actual.
- Se elimina el link entre el sistema y el pedido actual.

Problema 3 (25 puntos)

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

i. ¿Qué es un controlador?

Un controlador es una clase que implementa las operaciones del sistema.

ii. ¿Qué tipos de controladores existen y qué operaciones contienen?

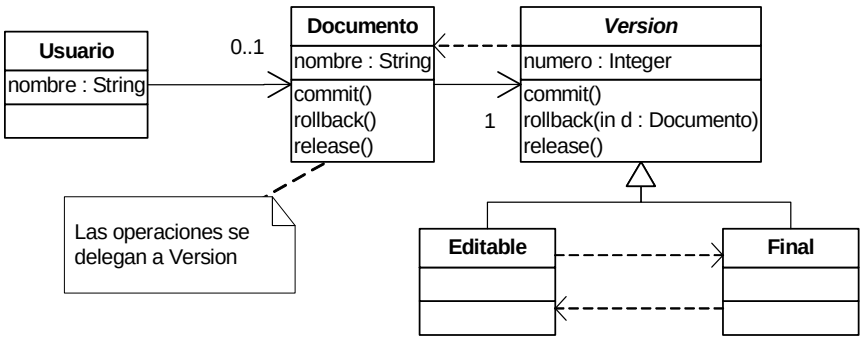
Controladores de Fachada: contiene todas las operaciones del sistema
Controlador de Caso de Uso: contiene todas las operaciones de un mismo caso de uso
Controlador de Mini Fachada: contiene operaciones de varios casos de uso relacionados

iii. ¿Qué relación existe entre controladores e interfaces del sistema?

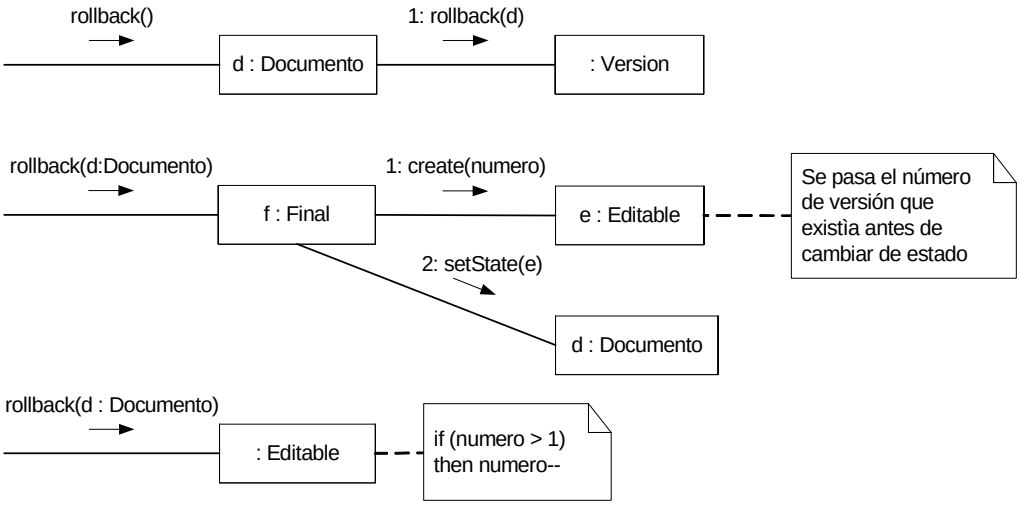
Las interfaces del sistema contienen las operaciones del sistema, las cuales son implementadas por los controladores.

b)

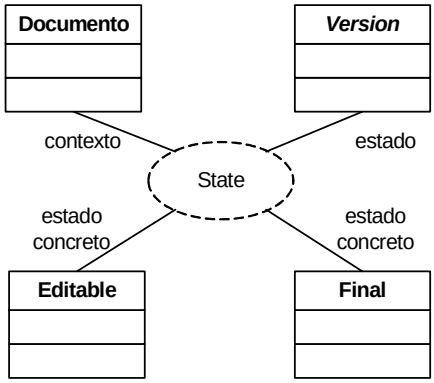
i. Realice el Diagrama de Clases de Diseño de la solución al problema planteado.



ii. Realice el Diagrama de Comunicación correspondientes a la operación `rollback()`



iii. Identifique el patrón de diseño utilizado indicando las clases participantes y sus roles.



Problema 4 (25 puntos)

Implemente un `ManejadorSeñal` concreto, las interfaces `IParameter` e `ISubscriptor` y todas las clases que defina para resolver el funcionamiento parcial del sistema descrito.

```
/** IParameter.hh **/

class IParameter {
      public virtual ~IParameter;
}

/** IParameter.cc **/

IParameter::~IParameter () {}


/** ISubscriptor.hh **/

class ISubscriptor : public Object {
      public:
            virtual void accionAlerta(IParameter * param) = 0;
            virtual ~ISubscriptor();
}

/** ISubscriptor.cc **/

ISubscriptor::~ISubscriptor(){}

/** IntParameter.hh **/

//si el valor es 1 el mensaje corresponde
//a accionar las camaras fijas.
//si es valor es 2 el mensaje corresponde
//a accionar las camaras moviles.

class IntParameter : public IParameter {
      private:
            int param;
      public:
            IntParameter(int p);
            int getParam();
}


/** IntParameter.cc **/

IntParameter::IntParameter (int p) : param(p){}

int IntParameter::getParam() {return param;}

/** SubscriptorCamaraFija.hh **/

class SubscriptorCamaraFija : public ISubscriptor {
      private:
            Fija * camara;
      public:
            SubscriptorCamaraFija(Fija *);
            void accionAlerta(IParameter *);
            ~SubscriptorCamaraFija();
}
```

```
/** SubscriptorCamaraFija.cc **/

SubscriptorCamaraFija::SubscriptorCamaraFija(Fija * cam){
      camara = cam;
}

void SubscriptorCamaraFija::accionAlerta(IParameter * param){
      IntParameter * param = dynamic_cast<IntParameter*>(p);
      if (p != 0 && p->getParam == 1)
            camara->fotografiar();
}

SubscriptorCamaraFija::~SubscriptorCamaraFija(){
      delete camara;
}


/** SubscriptorCamaraMovil.hh **/

class SubscriptorCamaraMovil : public Object {
      private:
            Movil * camara;
      public:
            SubscriptorCamaraMovil(Movil*);
            void accionAlerta(IParameter*);
            ~SubscriptorCamaraMovil();
}


/** SubscriptorCamaraMovil.cc **/

SubscriptorCamaraMovil::SubscriptorCamaraMovil(Movil * cam){
      camara=cam;
}

void SubscriptorCamaraMovil::accionAlerta(IParameter * param){
      IntParameter * param = dynamic_cast<IntParameter*>(p);
      if (p != 0 && p->getParam == 2)
            camara->fotografiar();
}

SubscriptorCamaraMovil::~SubscriptorCamaraMovil(){delete camara;}


/** ManejadorSeñal.hh **/

class ManejadorSeñal {
      private:
            ICollection * subs;
      public:
            ManejadorSeñal();
            void alertaSeñal(int vel);
            void agregarSubscriptor(ISubscriptor * sub);
            ~ManejadorSeñal();
}
```

```
/** ManejadorSeñal.cc **/

ManejadorSeñal::ManejadorSeñal(){
	subs=new List();
}

ManejadorSeñal::alertaSeñal(int vel){
	IIterator *it = subs->getIterator(); ISubscriptor *sub;

	//primero se alertan las camaras fijas
	while(it->hasCurrent){
		sub = (ISubscriptor*)it->current();
		sub->accionAlerta(new IParameter(1));
		it->next();
	}
	delete it;
	//Espero un tiempo en función de la velocidad registrada
	Timer::sleep(vel);

	it = subs->getIterator();

	//ahora alerto las camaras moviles
	while(it->hasCurrent){
		sub = (ISubscriptor*)it->current();
		sub->accionAlerta(new IParameter(2));
		it->next();
	}
	delete it;
}

ManejadorSeñal::agregarSubscriptor(ISubscriptor * sub){
	subs->add(sub);
}

ManejadorSeñal::~ManejadorSeñal(){
	delete subs;
}
```