Programación 4

SOLUCIÓN EXAMEN JULIO 2012

Por favor siga las siguientes indicaciones:

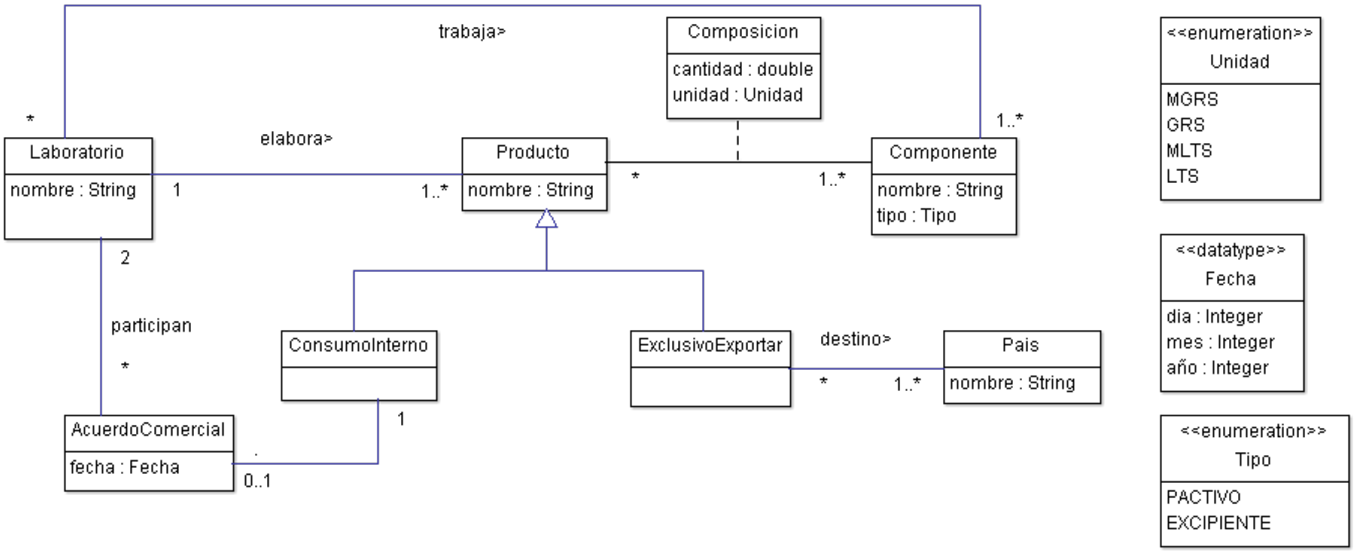
- Escriba con lápiz.
- Escriba las hojas de un solo lado.
- Escriba su nombre y número de documento en todas las hojas que entregue.
- Numere las hojas e indique el total de hojas en la primera de ellas.
- Recuerde entregar su número de examen junto al examen.

Problema 1 (30 puntos)

La División de Laboratorios Veterinarios del Uruguay está interesada en contar con una aplicación de vademécum (catálogo especializado) de productos veterinarios.

Se pide:

1. Modelar la realidad planteada mediante un Diagrama de Modelo de Dominio UML.
2. Expresar todas las restricciones del modelo en **lenguaje natural**.



Restricciones:
R1: el atributo nombre correspondiente laboratorios, productos, componentes y países los identifica.
R2: un laboratorio elabora productos con componentes con los cual trabaja.
R3: un acuerdo comercial es sobre un producto de un laboratorio participante del acuerdo.
R4: la cantidad de cada composición debe ser mayor a 0.

3. Expresar las pre- y post-condiciones, en lenguaje natural, de los contratos de las operaciones del siguiente Diagrama de Secuencia del Sistema (DSS) con memoria, correspondiente a la creación de un acuerdo comercial:

Operación	`iniciarAcuerdoComercial(...)`
Pre	• Existe producto interno **p** de nombre "nomProducto" • Existe laboratorio **l** de nombre labNuevo y no elabora p. • No existe acuerdo comercial para el producto p. • El laboratorio l tiene todos los componentes del producto p.
Post	• Se crea instancia **a** de AcuerdoComercial con fecha "Fecha". Se crean lnks entre a y p, a y l , a y laboratorio elaborador de p. • Se recuerda a.

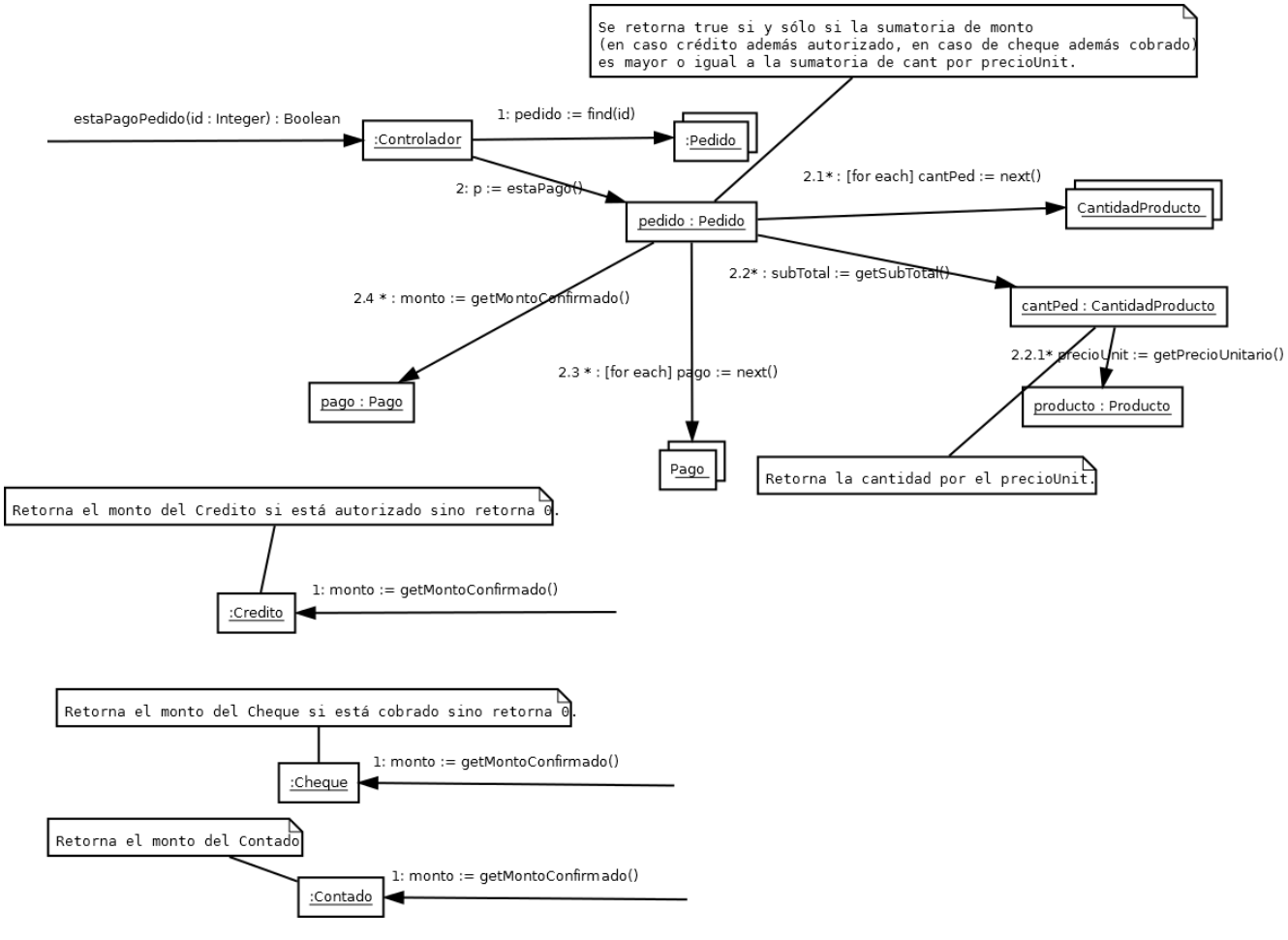
Operación	`agregarComponenteExcipiente(...)`
Pre	• Existe un componente c de tipo excipiente de nombre "nomComp". • El componente c no está como componente del producto p linkeado al acuerdo comercial en memoria. • El laboratorio l linkeado al acuerdo comercial que no tiene el producto p, trabaja con el componente c • El atributo cantidad es mayor a 0.
Post	• Se recuerdan en la memoria del sistema los datos ingresados por parámetro

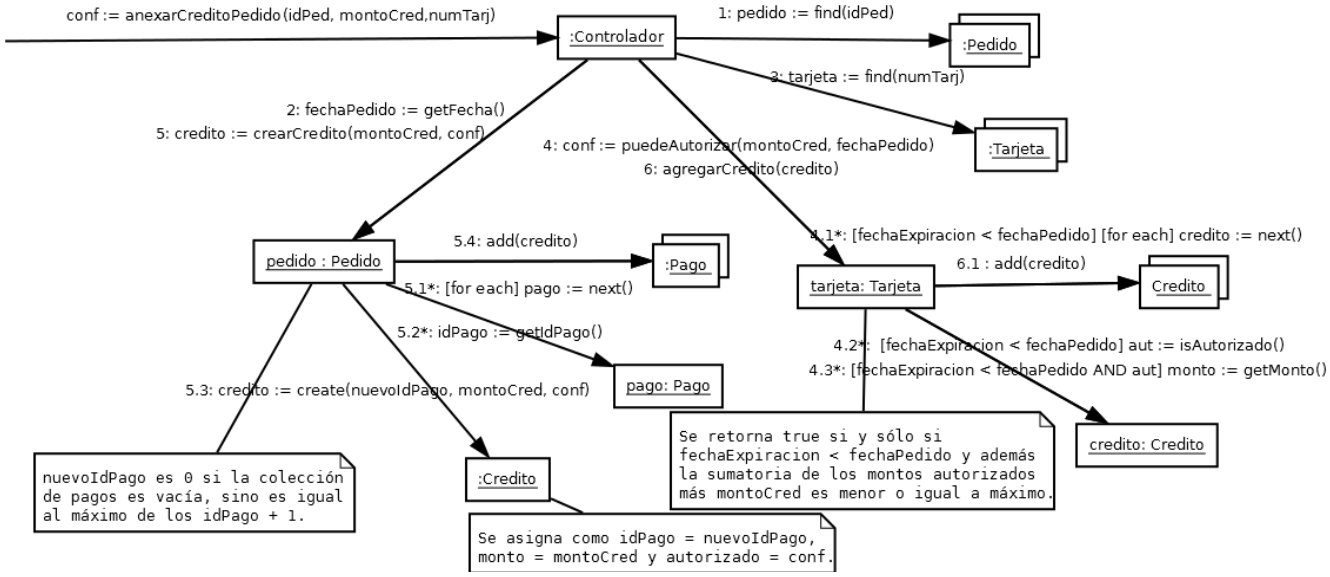
Operación	`finalizarAcuerdoComercial(...)`
Pre	• Existe en memoria un acuerdo comercial a. • Existe en memoria los datos de los componentes agregados (0 o más).
Post	• Screa el tipo asociativo composición para cada componente de nombre recordado y el producto p, con los atributos cantidad y unidad recordados. • Se quita el link entre el producto p del acuerdo comercial y el laboratorio linkeado al producto. • Se crea el link entre el otro laboratorio del acuerdo comercial y el producto p. • Se libera la memoria del sistema

Problema 2 (35 puntos)

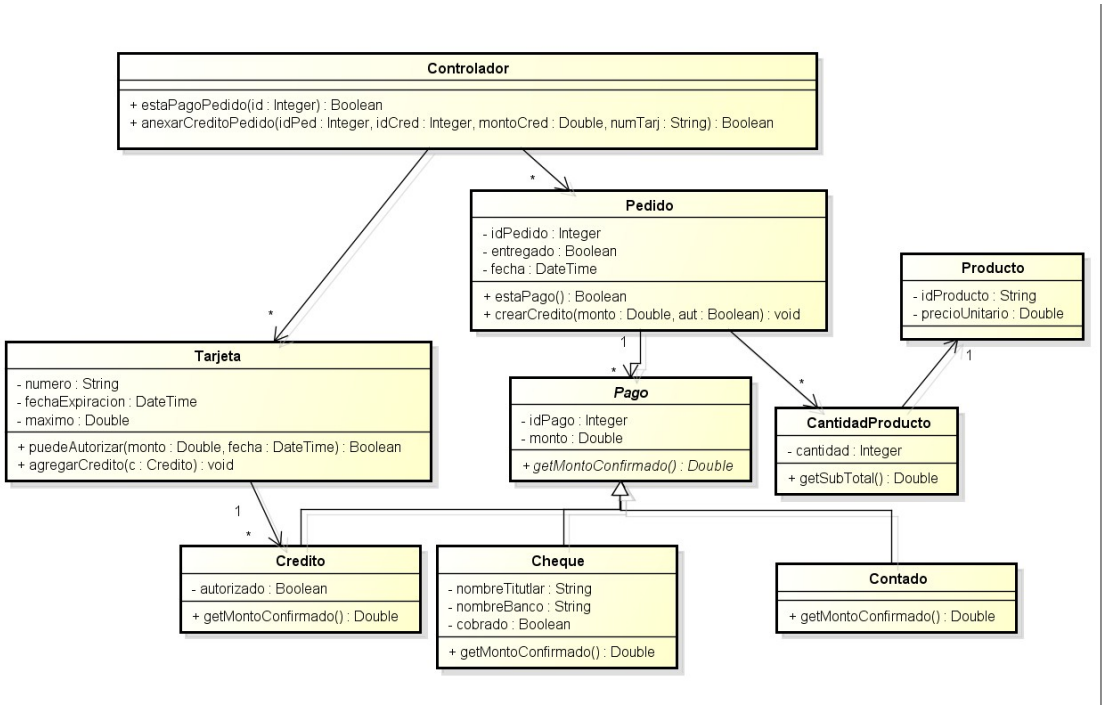
Se desea diseñar un sistema de pedidos partiendo del siguiente diagrama de modelo de dominio.

i. Realizar el Diagrama de Comunicación completo de las siguientes operaciones del sistema. Indicar claramente los parámetros y el tipo del resultado de todas las operaciones involucradas en su solución. No se permite agregar atributos que puedan ser calculados por la información ya brindada en el modelo de dominio. Se pueden agregar DataTypes en caso de que lo requiera.




conf := anexarCreditoPedido(idPed, montoCred,numTarj)
:Controlador
1: pedido := find(idPed)
:Pedido
3: tarjeta := find(numTarj)
:Tarjeta
2: fechaPedido := getFecha()
5: credito := crearCredito(montoCred, conf)
4: conf := puedeAutorizar(montoCred, fechaPedido)
6: agregarCredito(credito)
pedido : Pedido
5.4: add(credito)
:Pago
5.1*: [for each] pago := next()
4.1*: [fechaExpiracion < fechaPedido] [for each] credito := next()
tarjeta: Tarjeta
6.1 : add(credito)
Credito
5.2*: idPago := getIdPago()
pago: Pago
4.2*: [fechaExpiracion < fechaPedido] aut := isAutorizado()
4.3*: [fechaExpiracion < fechaPedido AND aut] monto := getMonto()
5.3: credito := create(nuevoIdPago, montoCred, conf)
credito: Credito
nuevoIdPago es 0 si la colección de pagos es vacía, sino es igual al máximo de los idPago + 1.
:Credito
Se retorna true si y sólo si fechaExpiracion < fechaPedido y además la sumatoria de los montos autorizados más montoCred es menor o igual a máximo.
Se asigna como idPago = nuevoIdPago, monto = montoCred y autorizado = conf.

ii. Realizar el Diagrama de Clases de Diseño (DCD) correspondiente para las operaciones de i.



Problema 3 (35 puntos)

Se desea implementar un sistema que gestiona el inventario de artículos de una tienda.

Se pide:

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

```
// Sistema.h
class Sistema {
private:
      IDictionary *categorias;
      IDictionary *articulos;
      static Sistema *instancia;
public:
      static Sistema *getInstance();
      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 IKey, public ICollectible
{
private:
      String nombre;
public:
      bool equals(IKey *k);
      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
Sistema *Sistema::instancia = NULL;

Sistema *Sistema::getInstance()
{
      if(instancia == NULL)
            instancia = new Sistema();
      return instancia;
}

InfoArticulo *Sistema::procesarCategoria(String cat)
{
      InfoArticulo *ret = NULL;

      float precioMinimo = getPrecioMinimo();

      Categoria *c = categorias->find(cat);
      if(cat == NULL)
            throw std::invalid_argument("No existe la categoria");

      ICollection *codigos = c->getAllCodigos();
      IIterator *it;
      for(it = c->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)
                  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;
}
```