Programación Avanzada

SOLUCION EXAMEN DICIEMBRE 2015

Por favor siga las siguientes indicaciones:

- 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.
- El examen tiene un total de 100 puntos, para aprobar es necesario tener 60 puntos o más.
- La duración del examen es de 3 horas.

Problema 1 (35 puntos)

Se desea desarrollar un sistema de pólizas de seguros para una determinada empresa. Para ello, se cuenta con la siguiente descripción de la realidad:

En la empresa trabajan agentes que pueden contactar a clientes con el objetivo de venderles pólizas de seguros. Los agentes ofrecen contratos, los cuales tienen un número que los identifica. Un cliente puede firmar un único contrato con la empresa. Un contrato es de un único cliente y es vendido por un único agente.

De los clientes se conoce la cédula que los identifica, el nombre, un teléfono de contacto y su dirección de correo. De los agentes se conoce la cédula de identidad que los identifica y la fecha en que comenzó a trabajar como vendedor de pólizas de seguros.

Los contratos tienen al menos una póliza de seguro. Con el correr del tiempo se pueden agregar o remover pólizas del contrato. Es de interés registrar la fecha en que se agrega una póliza en el contrato.

Una póliza puede estar en contratos distintos. Existen dos tipos de pólizas, pólizas de casa y pólizas de automóvil. De las pólizas se conoce un entero que las identifica y el costo en pesos uruguayos.

Además, considere la siguiente operación:

Nombre	Incluir póliza en el contrato de un cliente
Operación	*incluirPolizaEnContrato(idP: Integer, ciC: String)*
Descripción	Agrega la póliza de identificador *idP* en el contrato del cliente de cédula *ciC*, utilizando la fecha actual del sistema

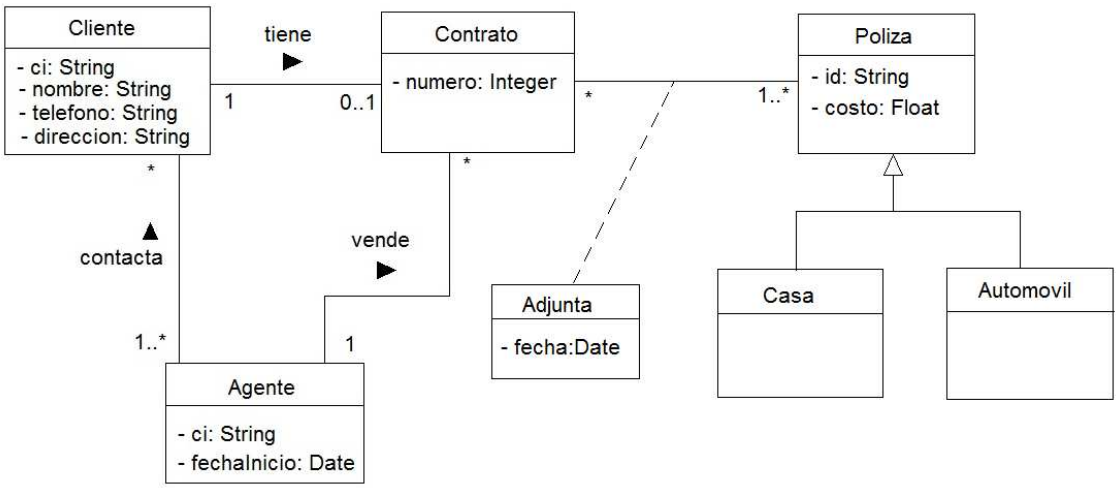
Se pide:

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.

ii. Completar el contrato de la operación *incluirPolizaEnContrato*, incluyendo las pre- y post-condiciones correspondientes.

Solución:

i. Modelo de Dominio

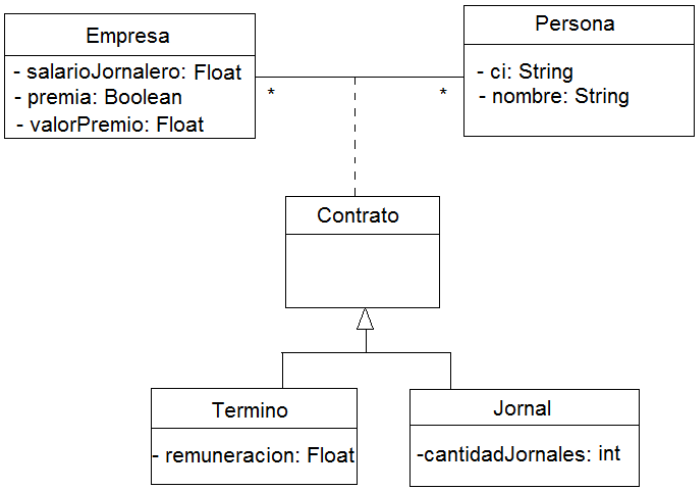


ii. Contrato

Nombre	Incluir póliza en el contrato de un cliente
Operación	*incluirPolizaEnContrato(idP: Integer, ciC: String)*
Descripción	Agrega la póliza de identificador *idP* en el contrato del cliente de cédula *ciC*, utilizando la fecha actual del sistema
Pre	- La póliza con identificador *idP* existe - El cliente con cédula *ciC* 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

Problema 2 (30 puntos)

Considere el siguiente modelo de dominio que representa los contratos de personal de distintas empresas:



Se quiere diseñar la operación que realiza la *liquidación de las remuneraciones de todas las personas*. Esta operación no recibe parámetros y devuelve un conjunto de valores. Cada uno de estos valores tiene tres atributos: *cédula de la persona, nombre de la persona* y el *importe de su remuneración*.

La base del cálculo de la liquidación se ha delegado en la clase *Contrato*, que es el experto de la información. Calcular el importe de remuneración de un contrato consiste en la *suma* entre el *premio que otorga la empresa* (si es que corresponde) y el cálculo en sí de la *remuneración que depende del tipo de contrato*. Si es a *término* la remuneración es la fijada en la definición del contrato. Si es *jornalero* la remuneración se calcula multiplicando el valor del salario de jornalero de la empresa y la cantidad de jornales acordada en la definición del contrato.

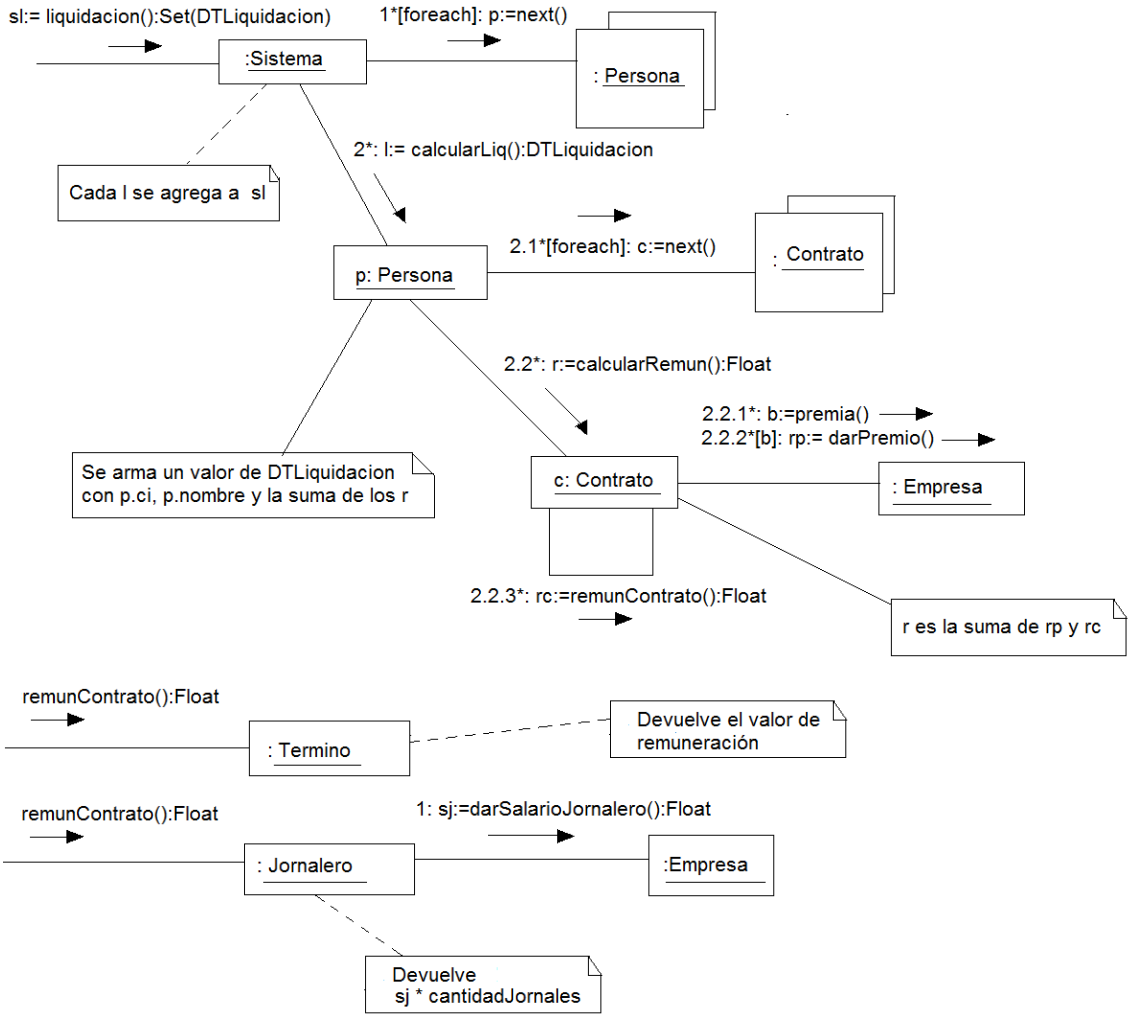
Se pide:

i. Realice el Diagrama de Comunicación completo de la operación *liquidación*. Indicar claramente los parámetros y el tipo del resultado de todas las operaciones involucradas en su solución. No es necesario indicar las visibilidades.

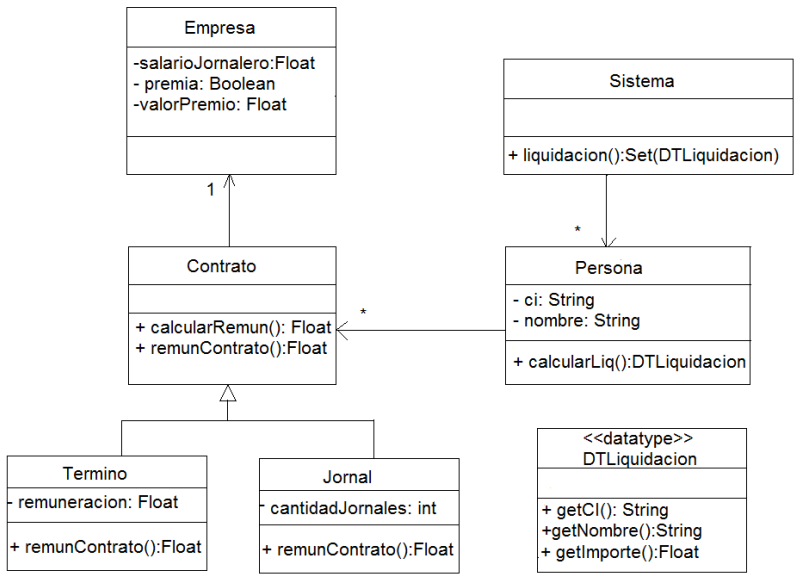
ii. Realice el Diagrama de Clases de Diseño (DCD) correspondiente.

Solución:

i. Diagrama de Comunicación

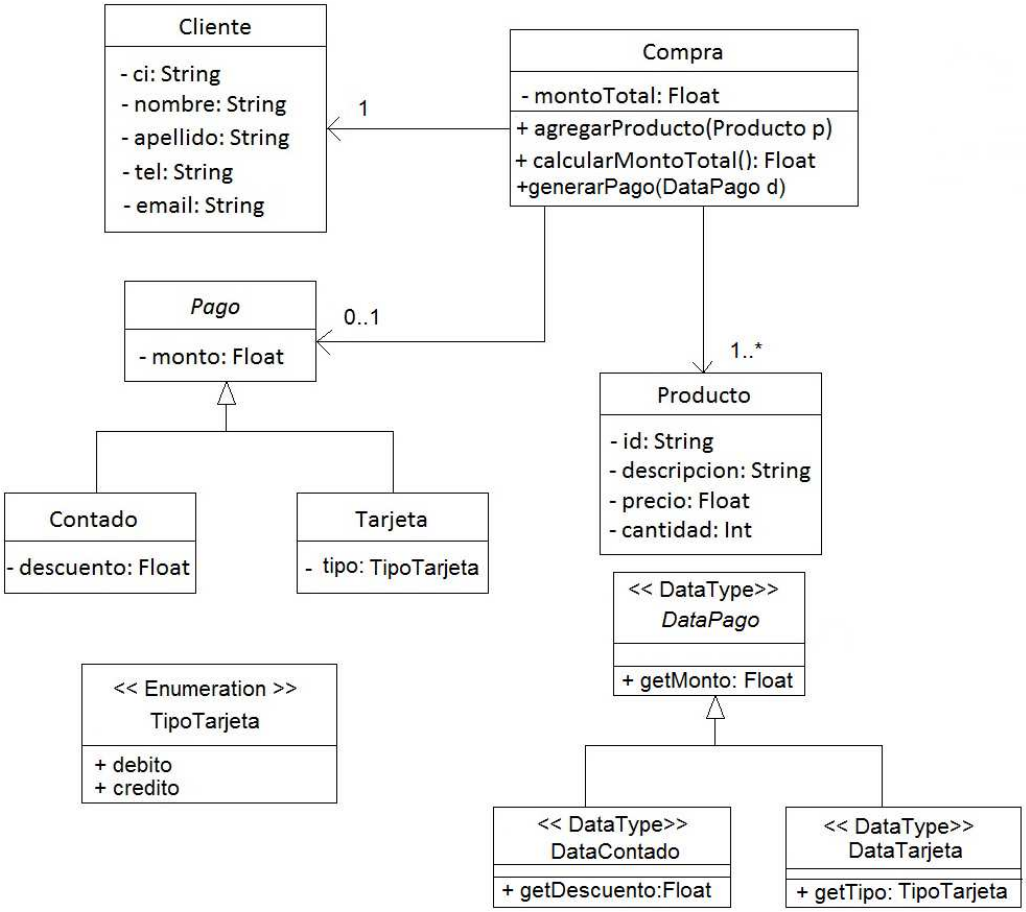


ii. DCD



Problema 3 (35 puntos)

Considere el siguiente diagrama de clases, correspondiente a una parte de un sistema de compras de productos por parte de clientes.



Se pide:

Implementar completamente en C++ todas las clases presentes en el diagrama anterior.

Observaciones:

- Se deberá implementar los *constructores*, *destructores*, *setters* y *getters* de las distintas clases, así como las operaciones de la clase *Compra*:
    - La operación *agregarProducto* recibe un producto y lo agrega a la colección de productos de la compra.
    - La operación *calcularMontoTotal*, deberá recorrer todos sus productos para obtener el monto total de la compra.
    - La operación *generarPago* genera el pago de la compra a partir de *d*, que es una instancia de *DataPago*.
- No implementar *datatypes* ni *enumerados*.
- No incluir directivas al precompilador.
- Puede suponer la existencia de la interface *ICollectible* e implementaciones de *ICollection* (clase *List*) e *IIterator* según sea necesario.
- Es posible utilizar las clases *set<T>* o *vector<T>* de la *Standard Template Library* (STL).

Solución:

```
//Cliente.h

#include “String.h”

class Cliente {

private:

   String ci;
   String nombre;
   String apellido;
   String tel;
   String email;

public:

   Cliente();
   Cliente(String ci, String nombre, String apellido, String tel, String email);
   void setCI(String ci);
   void setNombre(String nombre);
   void setApellido(String apellido);
   void setTel(String tel);
   void setEmail(String email);
   String getCI();
   String getNombre();
   String getApellido();
   String getTel();
   String getEmail();
   virtual ~Cliente();

};

//Cliente.cpp

#include “Cliente.h”

Cliente::Cliente() {

}

Cliente::Cliente(String ci, String nombre, String apellido, String tel, String email)
{
   this->ci = ci;
   this->nombre = nombre;
   this->apellido = apellido;
   this->tel = tel;
   this->email = email;
}

void Cliente::setCI(String ci) {
   this->ci = ci;
}

void Cliente::setNombre(String nombre) {
   this->nombre = nombre;
}

void Cliente::setApellido(String apellido) {
   this->apellido = apellido;
}

void Cliente::setTel(String tel) {
   this->tel = tel;
}
```

```
void Cliente::setEmail(String email) {
   this->email = email;
}

String Cliente::getCI() {
   return this->ci;
}

String Cliente::getNombre() {
   return this->nombre;
}

String Cliente::getApellido() {
   return this->apellido;
}

String Cliente::getTel() {
   return this->tel;
}

String Cliente::getEmail() {
   return this->email;
}

Cliente::~Cliente() {

}

//Producto.h

#include “String.h”
#include “ICollectible.h”

class Producto: public ICollectible {

private:

   String id;
   String descripcion;
   float precio;
   int cantidad;

public:

   Producto();
   Producto(String id, String descripcion, float precio, int cantidad);
   void setId(String id);
   void setDescripcion(String descripcion);
   void setPrecio(float precio);
   void setCantidad(int cantidad);
   String getId();
   String getDescripcion();
   float getPrecio();
   int getCantidad();
   virtual ~Producto();

};
```

```
//Producto.cpp

#include “Producto.h”

Producto::Producto() {

}

Producto::Producto(String id, String descripcion, float precio, int cantidad) {

   this->id = id;
   this->descripcion = descripcion;
   this->precio = precio;
   this->cantidad = cantidad;

}

void Producto::setId(String id) {
   this->id = id;
}

void Producto::setDescripcion(String descripcion) {
   this->descripcion = descripcion;
}

void Producto::setPrecio(float precio) {
   this->precio = precio;
}

void Producto::setCantidad(int cantidad) {
   this->cantidad = cantidad;
}

String Producto::getId() {
   return this->id;
}

String Producto::getDescripcion() {
   return this->descripcion;
}

float Producto::getPrecio() {
   return this->precio;
}

int Producto::getCantidad() {
   return this->cantidad;
}

Producto::~Producto() {

}

//Pago.h

class Pago {

private:

   float monto;

public:

   Pago(float monto);
   virtual ~Pago();
};
```

```
//Pago.cpp

#include “Pago.h”

Pago::Pago(float monto) {
   this->monto = monto;
}

Pago::~Pago() {

}

//Contado.h

#include “Pago.h”

class Contado: public Pago {

private:

   float descuento;

public:

   Contado(float monto, float descuento);
   virtual ~Contado();

};

//Contado.cpp

#include “Contado.h”

Contado::Contado(float monto, float descuento): Pago(monto) {

   this->descuento = descuento;
}

Contado::~Contado() {

}

//Tarjeta.h

#include “Pago.h”
#include “TipoTarjeta.h”

class Tarjeta: public Pago {

private:

   TipoTarjeta tipo;

public:

   Tarjeta(float monto, TipoTarjeta tipo);
   virtual ~Tarjeta();

};
```

```
//Tarjeta.cpp

#include “Tarjeta.h”

Tarjeta::Tarjeta(float monto, TipoTarjeta tipo): Pago(monto) {

   this->tipo = tipo;
}

Tarjeta::~Tarjeta() {

}

//Compra.h

#include “Cliente.h”
#include “Producto.h”
#include “Pago.h”
#include “DataPago.h”
#include “ICollection.h”

class Compra {

private:

   float montoTotal;
   ICollection *productos;
   Cliente *cliente;
   Pago *pago;

public:

   Compra();
   Compra(Cliente *cliente, Producto *producto);
   void agregarProducto(Producto *producto);
   void calcularMontoTotal();
   void generarPago(DataPago *pago);
   virtual ~Compra();

};

//Compra.cpp

#include “Compra.h”
#include “IIterator.h”
#include “List.h”

Compra::Compra() {

}

Compra::Compra(Cliente *cliente, Producto* producto) {

   this->productos = new List();
   this->productos->add(producto);
   this->montoTotal = 0;
   this->cliente = cliente;
   this->pago = NULL;
}

void Compra::agregarProducto(Producto* producto) {
   this->productos->add(producto);
}
```

```
void Compra::calcularMontoTotal() {

   float parcial = 0;
   IIterator * it = this->productos->getIterator();
   while (it->hasCurrent()) {
        Producto * producto = (Producto *)it->getCurrent();
        parcial += producto->getCantidad() * producto->getPrecio();
        it->next();
   }
   this->montoTotal = parcial;
}

void Compra::generarPago(DataPago *pago) {

    Pago * p;
    DataContado * c;
    DataTarjeta * t;
    c = dynamic_cast<DataContado*> (pago);
    if (c != NULL){
       p = new Contado(c->getMonto(), c->getDescuento());
    }
    else
    {
       t = dynamic_cast<DataTarjeta*> (pago);
       p = new Tarjeta(t->getMonto(), t->getTipo());
    }
    this->pago = p;
}
```