Programación Avanzada

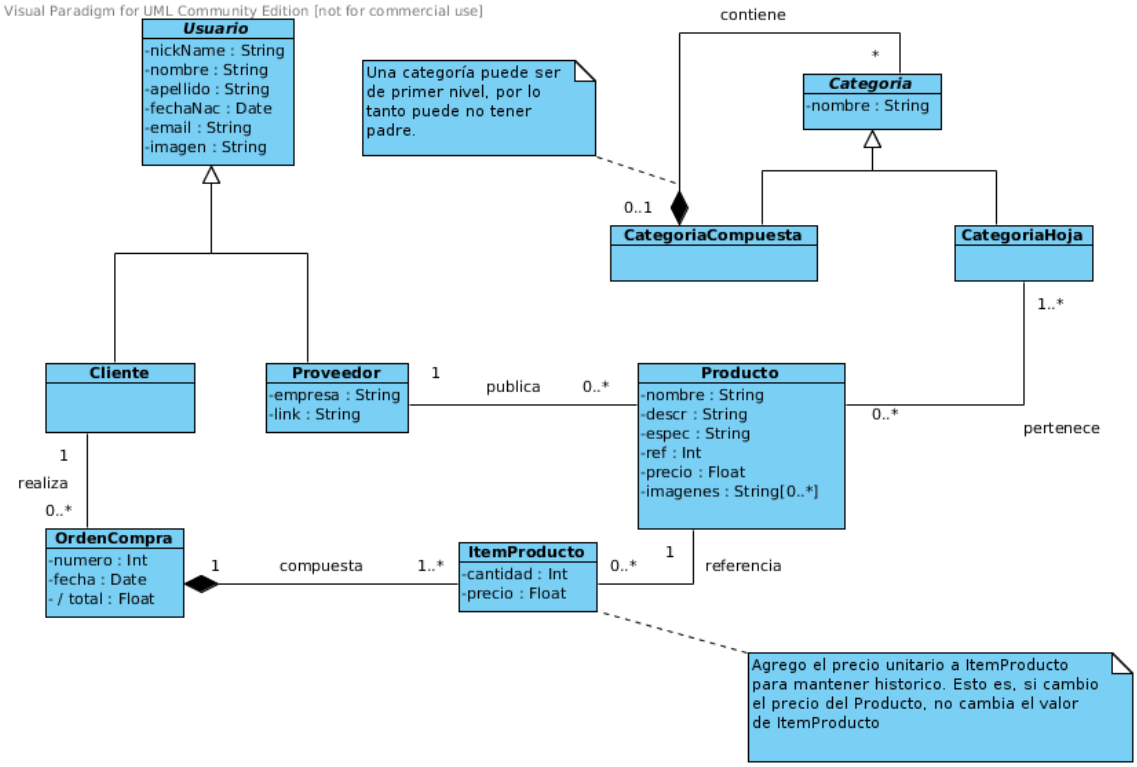
EXAMEN DICIEMBRE 2014

Por favor siga las siguientes indicaciones:

- Escriba con 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.

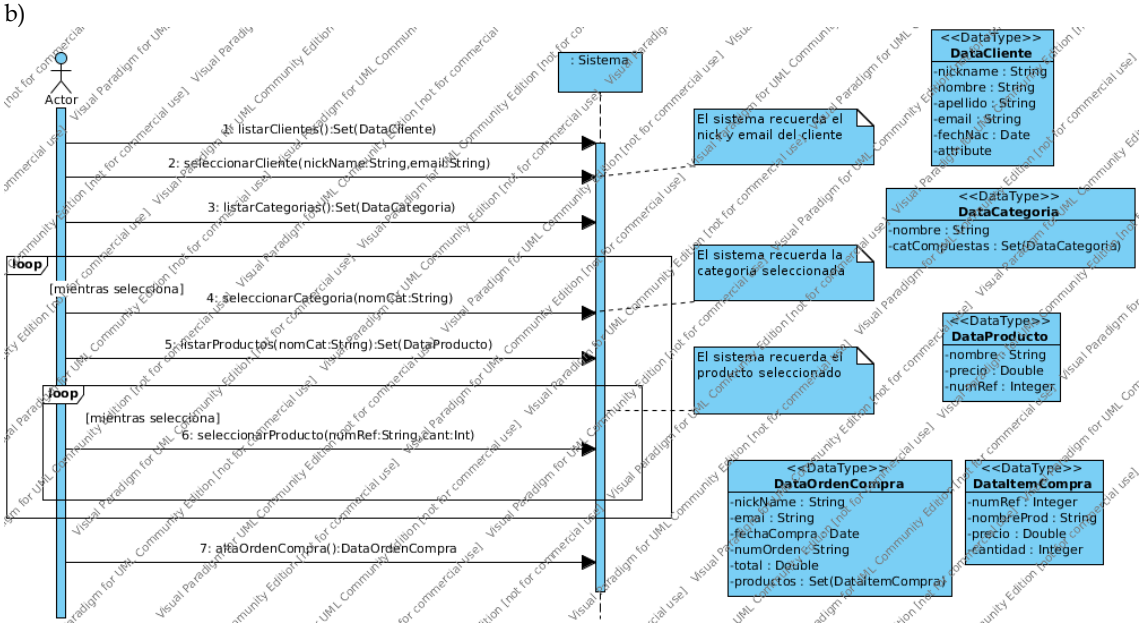
Problema 1 (40 puntos)

a)



- No existen dos usuarios con mismo nickname.
- No existen dos usuarios registrados con el mismo nickname.
- No existen dos productos con el mismo nombre.
- No existen dos productos con igual numero ref.
- No existen dos categorias con el mismo nombre.
- El total de una orden de compra se calcula como la sumatoria de los precios asociados a un item de producto
- Una imagen asociada a un producto no puede estar asociada a un usuario.
- Una imagen asociada a un usuario no puede estar asociada a un producto.

b)

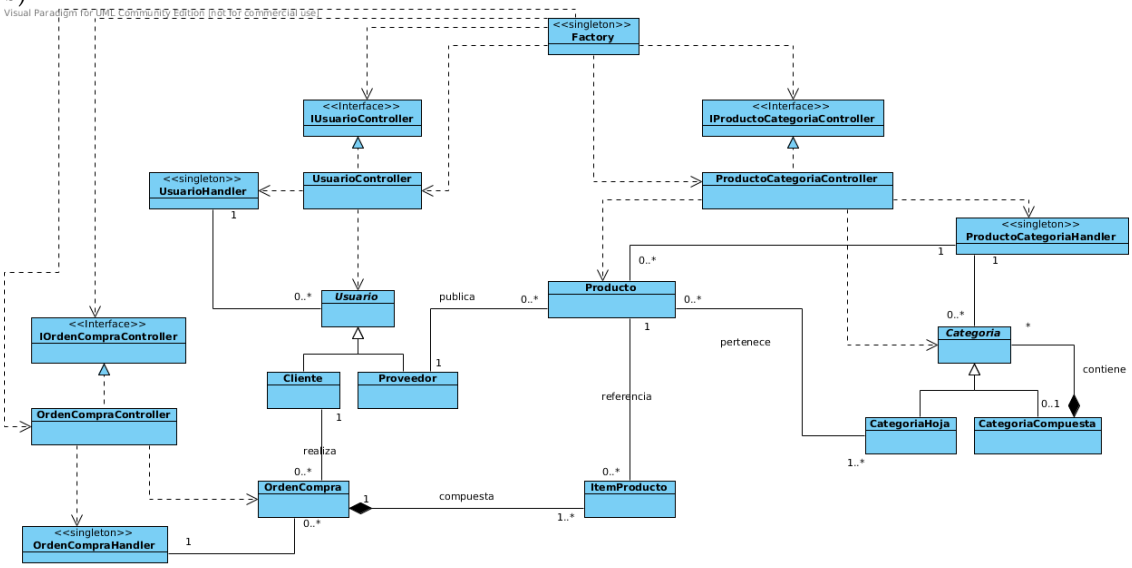

Actor
: Sistema
1: listarClientes():Set(DataCliente)
2: seleccionarCliente(nickName:String,email:String)
3: listarCategorias():Set(DataCategoria)
loop
[mientras selecciona]
4: seleccionarCategoria(nomCat:String)
5: listarProductos(nomCat:String):Set(DataProducto)
loop
[mientras selecciona]
6: seleccionarProducto(numRef:String,cant:Int)
7: altaOrdenCompra():DataOrdenCompra
El sistema recuerda el nick y email del cliente
El sistema recuerda la categoria seleccionada
El sistema recuerda el producto seleccionado
<<DataType>>
DataCliente
-nickname : String
-nombre : String
-apellido : String
-email : String
-fechNac : Date
-attribute
<<DataType>>
DataCategoria
-nombre : String
-catCompuestas : Set(DataCategoria)
<<DataType>>
DataProducto
-nombre : String
-precio : Double
-numRef : Integer
<<DataType>>
DataOrdenCompra
-nickName : String
-email : String
-fechaCompra : Date
-numOrden : String
-total : Double
-productos : Set(DataItemCompra)
<<DataType>>
DataItemCompra
-numRef : Integer
-nombreProd : String
-precio : Double
-cantidad : Integer

Problema 2 (30 puntos)

Dada la realidad planteada en el Ejercicio 1 se pide:

a) Elabore el **Diagrama de Comunicación** correspondiente al **Caso de Uso Generar Orden de Compra.**

b)



Problema 3 (30 puntos)

a) Ver teórico

b)

```
// ManejadorArchivos.hh
class ManejadorArchivos {
private:
      static ManejadorArchivos* instance;
      Elemento* raiz;
      ManejadorArchivos();
public:

      static ManejadorArchivos* getInstance();

      bool agregarArchivo(const DataRuta&, const String&);

      bool borrar(const DataRuta&);

};
```

```
// ManejadorArchivos.cc
ManejadorArchivos* ManejadorArchivos::instance = NULL;
ManejadorArchivos::ManejadorArchivos() {

       this->raiz = new Directorio("/");
}

ManejadorArchivos* ManejadorArchivos::getInstance() {

if (ManejadorArchivos::instance == NULL)

      ManejadorArchivos::instance = new ManejadorArchivos();
      return ManejadorArchivos::instance;

}

bool ManejadorArchivos::agregarArchivo(const DataRuta& ruta,
const String& contenido) {

      return this->raiz->agregarArchivo(ruta, contenido, 1);
```

```
}

bool ManejadorArchivos::borrar(const DataRuta& ruta) { // No se
puede borrar la raíz

_____
      if (ruta.getCantidadPartes() <= 0)
            return false;
      return this->raiz->borrar(ruta, 1);
           }


/ Elemento.hh
class Elemento : public ICollectible {
private:
      String nombre;
public:
      Elemento(const String&);

      String getNombre();

      virtual bool agregarArchivo(const DataRuta&, const String&,
int);

      virtual bool borrar(const DataRuta&, int);

      virtual ~Elemento();

};

// Elemento.cc
Elemento::~Elemento() {}
String Elemento::getNombre() {
       return this->nombre;}
bool Elemento::agregarArchivo(const DataRuta& ruta, const
String& contenido, int i) { // Implementación vacía por defecto
para el caso de Archivo

return false;}
```

```
bool Elemento::borrar(const DataRuta& ruta, int i) {//
Implementación vacía por defecto para el caso de Archivo

return false;}


// Archivo.hh
class Archivo : public Elemento {
private:
      String contenido;
public:
      Archivo(const String&, const String&);

      ~Archivo();

};

// Archivo.cc
Archivo::Archivo(const String& n, const String& c) : Elemento(n),
contenido(c) {}

Archivo::~Archivo() {

      UtilidadesIO::borrar(this);
}

// Directorio.hh
class Directorio : public Elemento {
private:
      IDictionary* hijos;
public:
      Directorio(const String&);

      bool agregarArchivo(const DataRuta&, const String&, int);

      bool borrar(const DataRuta&, int);

      ~Directorio();};
```

```
// Directorio.cc
Directorio::Directorio(const String& n) : Elemento(n), hijos(new
List()) {}

bool Directorio::agregarArchivo(const DataRuta& ruta, const
String& contenido, int i) {

StringKey k(StringKey(ruta.getParte(i)));

Elemento* e = (Elemento*)this->hijos->find(&k);

bool hijoDirecto = ruta.getCantidadPartes() == i;

if (e == NULL) {// Se crea el directorio o archivo
      String parte = ruta.getParte(i);
      IKey* key = new StringKey(parte); if (hijoDirecto) {

      if (hijoDirecto) {// Archivo

            Archivo* a = new Archivo(parte, contenido);

            this->hijos->add(key, a);

            UtilidadesIO::crear(a);

            }

      else {// Directorio

            Directorio* d = new Directorio(parte);
            this->hijos->add(key, d);
            UtilidadesIO::crear(d);
            // Esta invocación nunca va a fallar dado que el
            directorio está vacío

            d->agregarArchivo(ruta, contenido, i + 1);

            }

      return true;}

else {

      if (hijoDirecto) {// Ya existía un elemento con ese nombre
```

```
        return false;
}

else {// Invocación recursiva

return e->agregarArchivo(ruta, contenido, i + 1);

      }

}

}
```

```
bool Directorio::borrar(const DataRuta& ruta, int i) {

StringKey key(ruta.getParte(i));

Elemento* e = (Elemento*)this->hijos->find(&key);

        // Ruta inválida
        if (e == NULL)
                return false;
        if (ruta.getCantidadPartes() == i) {
                // Se borra un hijo directo
            delete this->hijos->remove(&key);
            return true;

} else {

            // Se borra un hijo indirecto, invocación recursiva

            return e->borrar(ruta, i + 1);
      }

}

Directorio::~Directorio() {
}

// Primero se eliminan los hijos para que no queden descolgados
// al momento de borrarlos de disco, además no se pueden borrar
// directorio no vacíos

IIterator* it;

for (it = this->hijos->getIterator(); it->hasCurrent(); it-
>next())

      delete it->getCurrent();

delete it;
delete this->hijos;
// Se elmina físicamente este directorio que ahora está vacío

UtilidadesIO::borrar(this);
}
```