Examen Febrero 2012

Presentar la resolución del parcial:

· Indique su nombre completo y número de cédula en cada hoja.
· Numere todas las hojas e indique la cantidad total de hojas que entrega en la primera.
· Escriba las hojas de un solo lado.
· **Comience cada ejercicio en una hoja nueva.**
· El parcial es individual y sin material. **APAGUE SU CELULAR.**
· **Escriba con lápiz y de forma prolija.**
· Duración: 3 horas.

Ejercicio 1 (30 puntos)

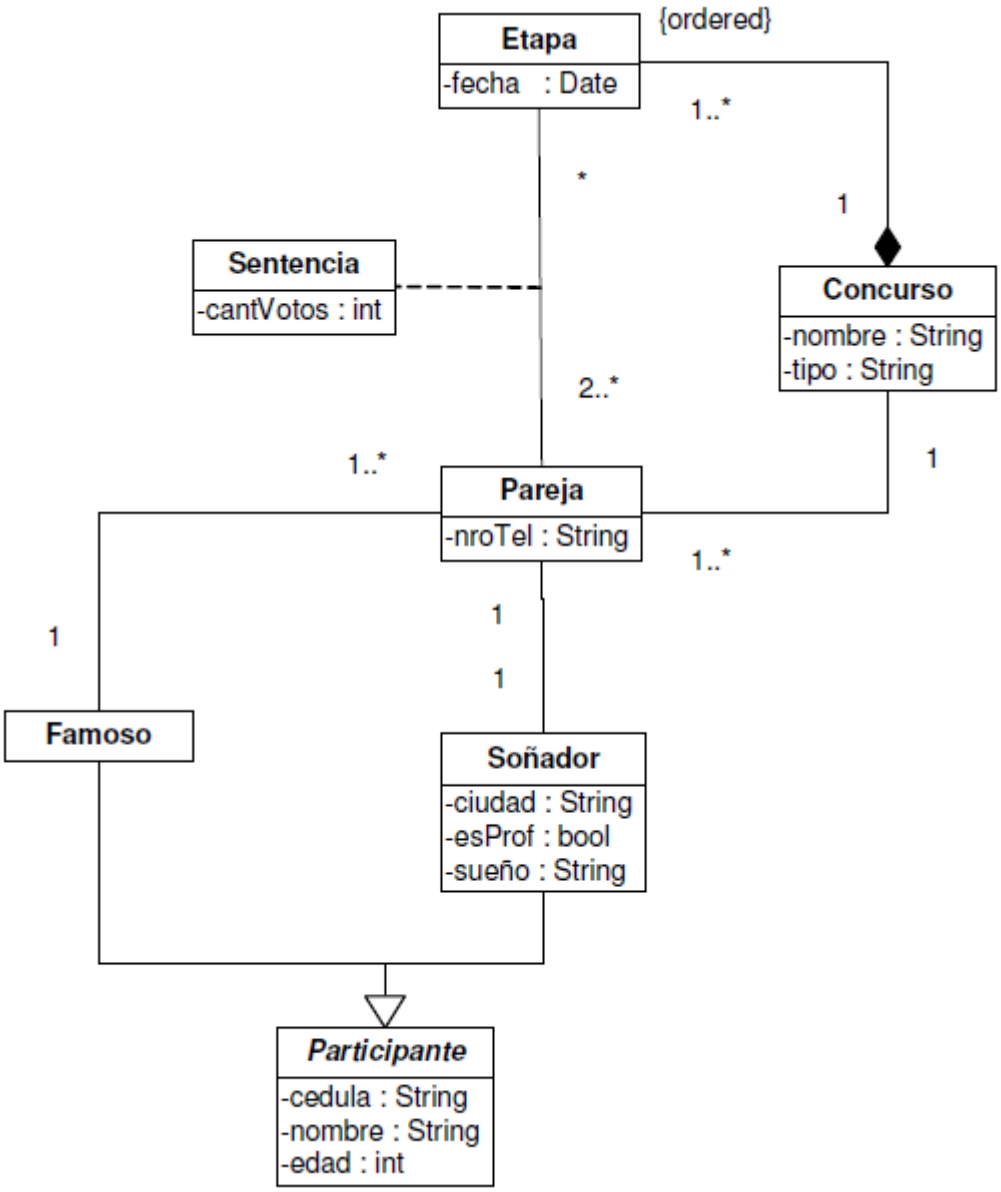
Construya un Diagrama de Modelo de Dominio UML para la siguiente realidad. Las restricciones deben ser expresadas en lenguaje natural.
Modele exclusivamente en base a la información presente en la descripción.

Una Empresa Productora de Televisión desea realizar un software que le permita gestionar varios concursos que la misma produce. Cada uno de estos concursos está formado por un conjunto de parejas que compite en una disciplina determinada (baile, canto, etc.), pudiendo existir varios concursos para una misma disciplina.

Un concurso se identifica por el nombre y tipo de disciplina, es de carácter eliminatorio y se realiza en distintas etapas que se llevan a cabo en una fecha determinada. Cabe notar que no puede haber dos etapas de un mismo concurso con la misma fecha. Como resultado de cada etapa, dos o más parejas quedarán sentenciadas, lo que habilitará a una votación del público. La votación se realiza por vía telefónica, por lo que a cada pareja se le asignará un número telefónico (que la identifica) al comenzar el concurso.
La pareja sentenciada con menor cantidad de llamados será eliminada del concurso.

Cada pareja participante está formada por un soñador y un famoso identificados por su cédula. Otros datos de interés son el nombre y la edad de los participantes. Para los soñadores se necesita saber además su ciudad natal, si es profesional y una descripción del sueño a cumplir en caso de ganar el concurso. Las reglas establecen que a un soñador sólo se le permitirá participar en un concurso mientras que los famosos pueden participar en más de uno. No está permitido que un famoso pueda participar con más de un soñador de un mismo concurso.

Solucion ejercicio 1



Restricciones:

context Etapa inv:

-- En una etapa, las parejas sentenciadas deben competir en el concurso al que la etapa pertenece.

```
self.pareja->forAll(p | p.concurso = self.concurso)
```

-- La pareja con menor cantidad de votos en una etapa no participa de una sentencia posterior

```
let eliminada :
Pareja = self.sentencia->select(s1|self.sentencia->forAll(s2|s2 <> s1
implies s1.cantVotos < s2.cantVotos) )->any().pareja in
eliminada.etapa->forAll(e|e.fecha <= self.fecha)
```

context Famoso inv:

-- Un famoso no puede formar parte de más de una pareja en un mismo concurso.

```
self.pareja->forAll(p1, p2 | p1 <> p2 implies p1.concurso <>
p2.concurso)
```

context Concurso inv:
-- Un concurso se identifica por su nombre y tipo.

```
Concurso.allInstances()->forAll(c1, c2| c1 <> c2 implies c1.nombre <>
c2.nombre or c1.tipo <> c2.tipo)
```

-- Para un concurso no puede haber dos etapas con una misma fecha.

```
self.etapa->isUnique(fecha)
```

-- Para un concurso no puede haber dos parejas con el mismo número de teléfono.

```
self.pareja->isUnique(nroTel)
```

context Participante inv:

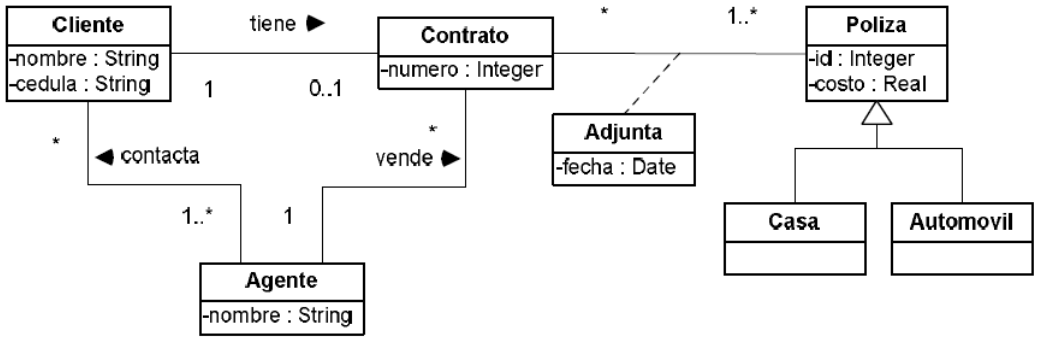
-- Un participante se identifica por su cédula.

```
Participante.allInstances()->isUnique(cedula)
```

Ejercicio 2 (35 puntos)

a) Responda en no más de 3 líneas cada una de las siguientes preguntas
   i) ¿A qué se le llama visibilidad?
   ii) ¿Qué tipos de visibilidad se vieron en el curso?
   iii) Describa dos tipos de visibilidad.

b) Durante la construcción de un sistema de pólizas de seguros se generó el siguiente modelo de dominio. Básicamente, existen agentes que contactan a clientes para venderles pólizas de seguros. Los clientes firman un contrato con la empresa y en dicho contrato adjuntan las pólizas que deseen tener (casa, automóvil, etc.), las cuales pueden agregarse y quitarse del contrato a lo largo del tiempo.



Dentro de los casos de uso a soportar por el sistema, se encuentran las siguientes operaciones:

Nombre	Incluir póliza en contrato existente de un cliente
Operación	agregarPoliza(idP:Integer, ced:String)
Descripción	Agrega la póliza de identificador idP al contrato del cliente de cédula ced utilizando la fecha actual del sistema.

Nombre	Consultar clientes con póliza de automóvil
Operación	consultarPolizaDeAutomovil():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

Se pide:

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

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 toda la información contenida en los diagramas.

Solución Ejercicio 2

a)

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)

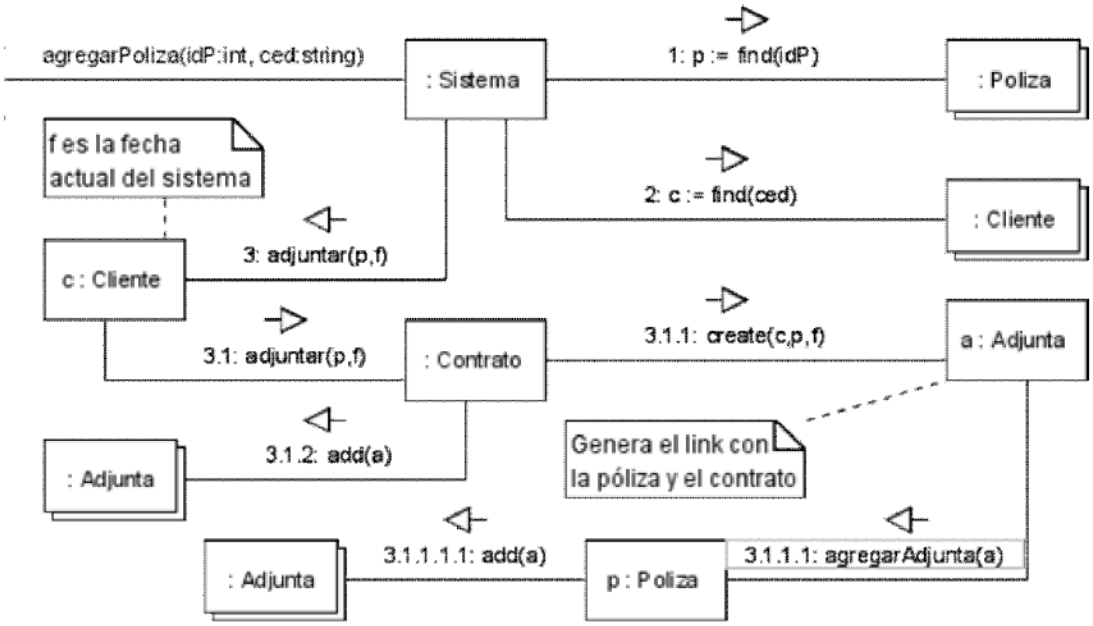
i)

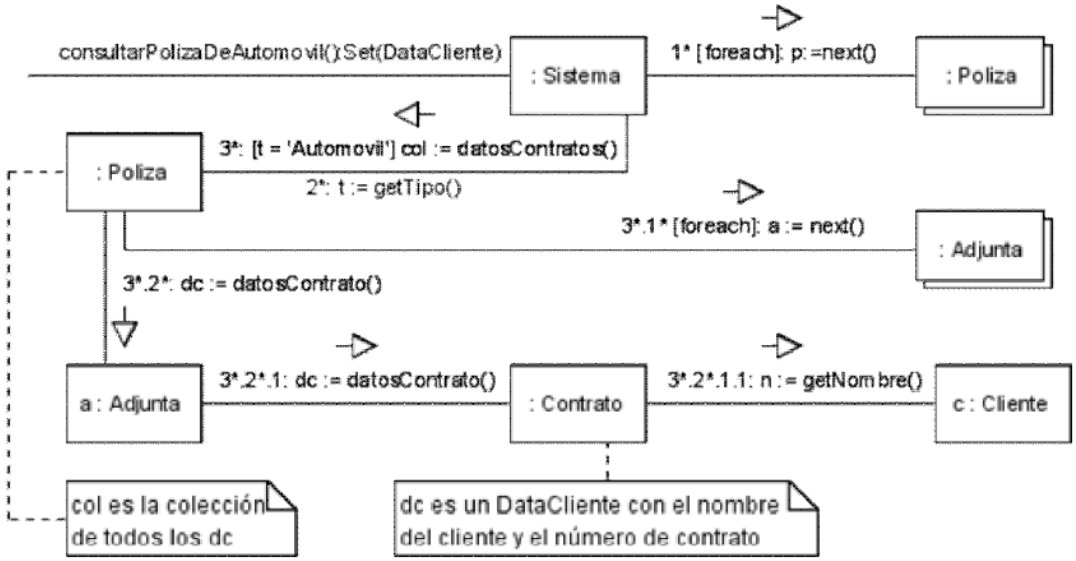
Nombre	**Incluir póliza en contrato de un cliente**
Operación	`agregarPoliza(idP:int, ced:string)`
Descripción	Agrega la póliza de identificador `idP` al contrato del cliente de cédula `ced` utilizando la fecha actual del sistema.
Pre	- La póliza con identificador idP existe - El cliente con cédula ced 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

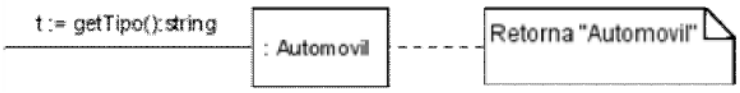
Nombre	**Consultar clientes con póliza de automóvil**
Operación	`consultarPolizaDeVida():Set(DataCliente)`
Descripción	Retorna un conjunto de datavalues compuestos por el número de contrato y el nombre de cliente que posean alguna póliza de automóvil
Pre	
Post	- Retorna una colección de DataCliente con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

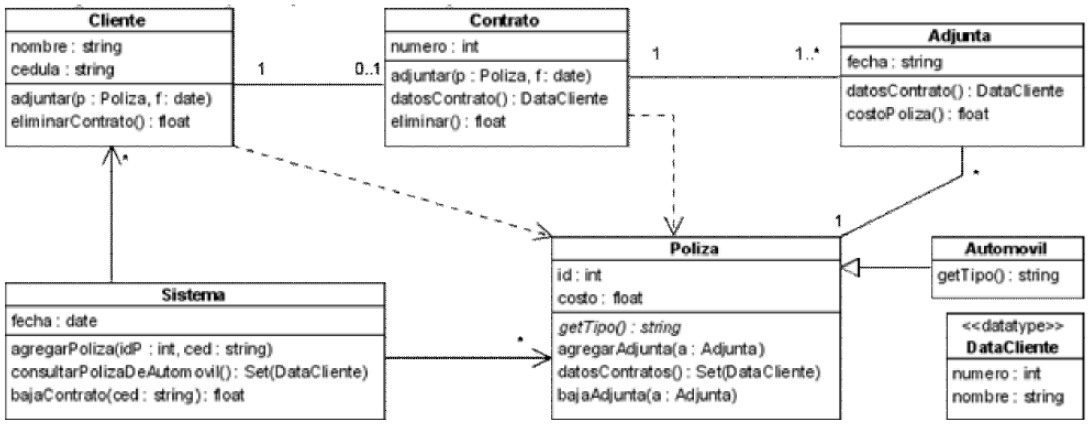
ii)


agregarPoliza(idP:int, ced:string)
: Sistema
1: p := find(idP)
: Poliza
f es la fecha actual del sistema
2: c := find(ced)
: Cliente
c : Cliente
3: adjuntar(p,f)
3.1: adjuntar(p,f)
: Contrato
3.1.1: create(c,p,f)
a : Adjunta
: Adjunta
3.1.2: add(a)
Genera el link con la póliza y el contrato
: Adjunta
3.1.1.1.1: add(a)
p : Poliza
3.1.1.1: agregarAdjunta(a)


consultarPolizaDeAutomovil():Set(DataCliente)
: Sistema
1* [foreach]: p:=next()
: Poliza
: Poliza
3*: [t = 'Automovil'] col := datosContratos()
2*: t := getTipo()
3*.1* [foreach]: a := next()
: Adjunta
3*.2*: dc := datosContrato()
a : Adjunta
3*.2*.1: dc := datosContrato()
: Contrato
3*.2*.1.1: n := getNombre()
c : Cliente
col es la colección de todos los dc
dc es un DataCliente con el nombre del cliente y el número de contrato

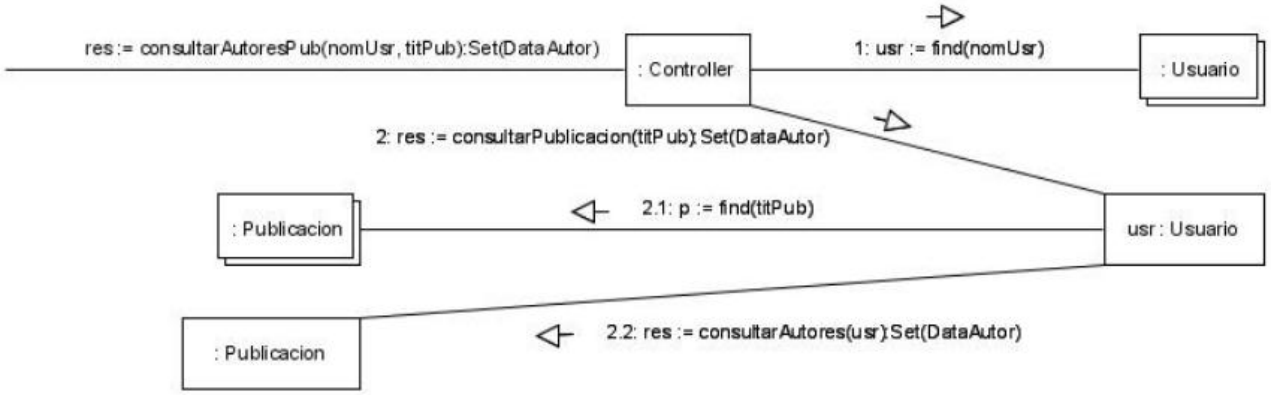

t := getTipo():string
: Automovil
Retorna "Automovil"

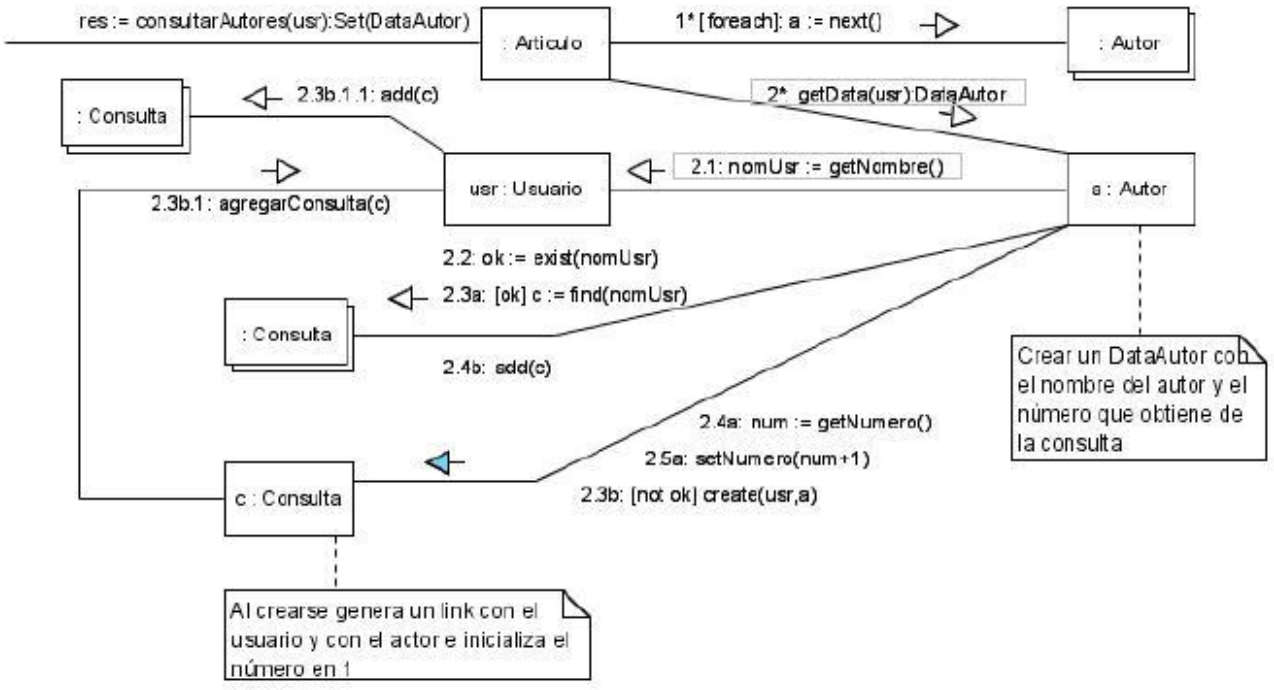
iii)

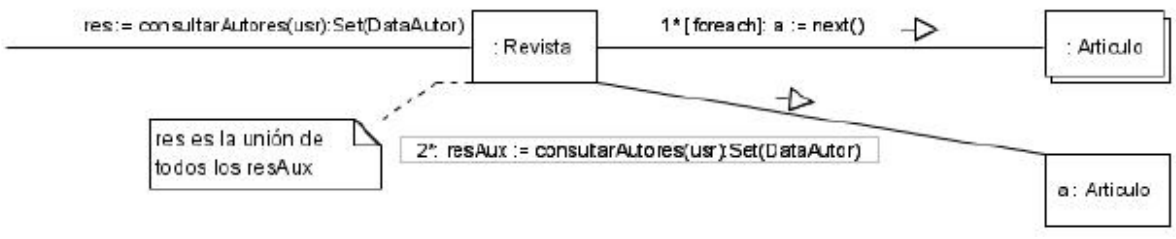

Cliente
nombre : string
cedula : string
adjuntar(p : Poliza, f : date)
eliminarContrato() : float
1
0..1
Contrato
numero : int
adjuntar(p : Poliza, f : date)
datosContrato() : DataCliente
eliminar() : float
1
1..*
Adjunta
fecha : string
datosContrato() : DataCliente
costoPoliza() : float
*
*
1
Poliza
id : int
costo : float
getTipo() : string
agregarAdjunta(a : Adjunta)
datosContratos() : Set(DataCliente)
bajaAdjunta(a : Adjunta)
Automovil
getTipo() : string
Sistema
fecha : date
agregarPoliza(idP : int, ced : string)
consultarPolizaDeAutomovil() : Set(DataCliente)
bajaContrato(ced : string) : float
*
<<datatype>>
DataCliente
numero : int
nombre : string

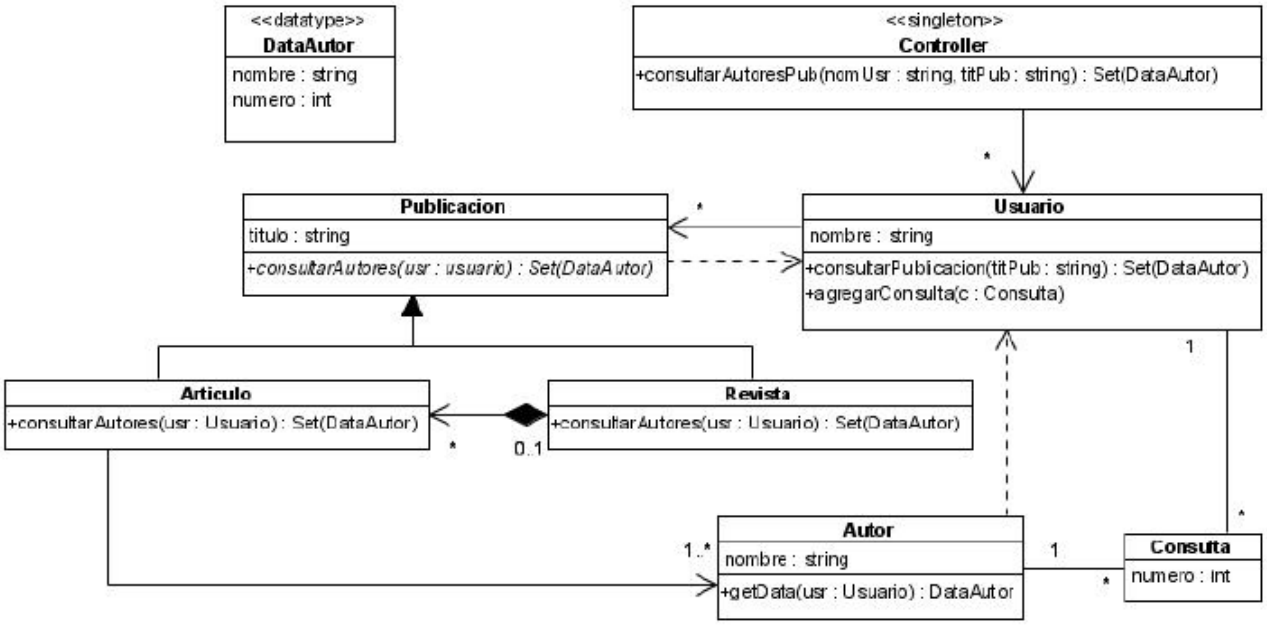
Ejercicio 3 (35 puntos)

Se está desarrollando un sitio web con un sistema de suscripción a publicaciones electrónicas. El sitio permite el registro de usuarios para acceder a dichas publicaciones, la consulta de información sobre las mismas y el registro de información estadística sobre la cantidad de consultas que cada usuario realiza. El diseño parcial del sistema incluye el Diagrama de Comunicación y el Diagrama de Clases de Diseño que se presentan a continuación.









Se pide:

i) Implemente en C++ el .h de la clase Controller

ii) Implemente en C++ el .cc de la clase Controller. Incluya código de manejo de excepciones en la operación consultarAutoresPub() para el caso en que no exista el usuario cuyo nombre se utiliza en la búsqueda de autores.

iii) Implemente en C++ los .h de la jerarquía de clases compuesta por Publicación, Articulo y Revista. No incluya constructores, destructores ni operaciones set y get de atributos.

iv) Implemente en C++ la operación de la clase Artículo
ICollection * consultarAutores (Usuario*).

v) Implemente en C++ la operación de la clase Autor
DataAutor * getData (Usuario *).

Observaciones:

- Asuma que existe una implementación estándar de las interfaces ICollectible, ICollection, IIterator, Idictionary e Ikey y que existe una clase Lista que realiza las interfaces Icollection e Idictionary y una clase KeyString que realiza la interfaz IKey. No defina colecciones concretas.
- Asuma la existencia de una clase String
- No incluya directivas de preprocesador.

Solución ejercicio 3

i) Implemente en C++ el `.h` de la clase `Controller`

```
class Controller {

private:

      static Controller * instance;

      Controller();


      IDictionary * usuarios;


public:

      static Controller * getInstance();

      ICollection * consultarAutoresPub(String nomUsr, String
titPub);

};
```

ii) Implemente en C++ el `.cc` de la clase `Controller`. Incluya código de manejo de excepciones en la operación `consultarAutoresPub()` para el caso en que no exista el usuario cuyo nombre se utiliza en la búsqueda de autores.

```
Controller * Controller::instance = NULL;


Controller::Controller() {

      usuarios = new Lista();

}


Controller * Controller::getInstance(){

  if (instance == NULL)

        instance = new Controller();
```

```
  return instance;
}

ICollection * Controller::consultarAutoresPub(String nomUsr,
                                              String titPub){
      KeyString *ks = new KeyString(nomUsr);
      ICollectible *usr = usuarios->find(ks);

      if (usr == NULL)
            throw new Exception("El usuario no existe");
      else
      {
            Usuario * usuario = (Usuario *) usr;
            return usuario->consultarPublicacion(titPub);
      }
}
```

iii) Implemente en C++ los `.h` de la jerarquía de clases compuesta por `Publicación`, `Articulo` y `Revista`.

```
class Publicacion : public ICollectible {
private:
	String titulo;

public:
	virtual ICollection * consultarAutores(Usuario * usr) = 0;
};

class Revista : public Publicacion {
private:
	ICollection * articulos;

public:
	ICollection * consultarAutores(Usuario * usr);
};

class Articulo: public Publicacion {
private:
	ICollection * autores;

public:
	ICollection * consultarAutores(Usuario * usr);
};
```

iv) Implemente en C++ la operación de la clase `Artículo` `ICollection * consultarAutores (Usuario *)`.

```
ICollection * Articulo::consultarAutores(Usuario * usr){
	IIterator * iter = autores->getIterator();
	ICollection * res = new Lista();

	while (iter->hasCurrent()){
		Autor * autor = (Autor *) iter->current();
		DataAutor * data = autor->getData(usr);
		res->add(data);
		iter->next();
	}

	return res;
}
```

v) Implemente en C++ la operación de la clase `Autor` `DataAutor * getData (Usuario *)`.

```
DataAutor * Autor::getData(Usuario * usr) {
	String nomUsr = usr->getNombre();
	KeyString * ks = new KeyString(nomUsr);

	ICollectible *cons = consultas->find(ks);
	if (cons == NULL){
		Consulta * consulta = new Consulta(usr,this);
		consultas->add(ks,consulta);
		return new DataAutor(this->nombre,1);
	}
	else{
		Consulta * consulta = (Consulta *) cons;
		int num = consulta->getNumero();
		consulta->setNumero(num+1);
		return new DataAutor(this->nombre,num+1);
	}
}
```