Programación 4

EXAMEN DICIEMBRE 2009

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 número de examen junto al examen.

Problema 1 (30 puntos)

a) Describa muy brevemente la diferencia entre una asociación y un link.

Una asociación describe una relación entre clasificadores (clases o datatypes) en tanto un link es una tupla que referencia instancias concretas de clasificadores. Un link es una instancia de una asociación.

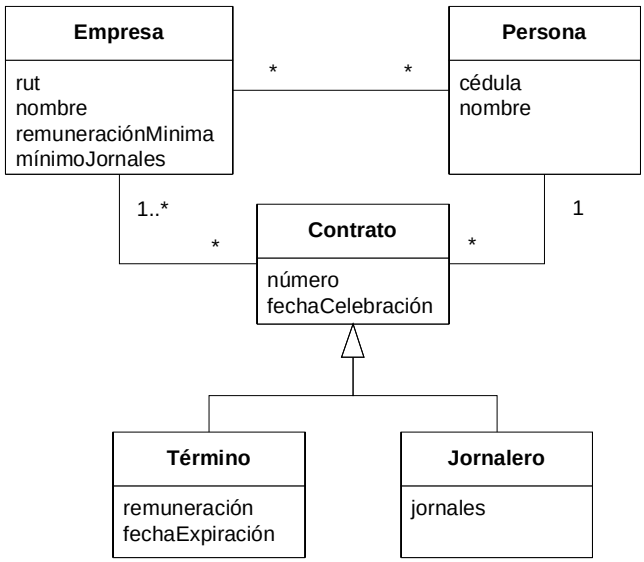
b) Se desea construir un sistema para una gestoría que permita llevar un registro de las empresas que gestiona y la información sobre las contrataciones que las mismas realizan.

De las empresas interesa registrar su número de RUT (que las identifica), su nombre y la remuneración y cantidad de jornales mínima por las que puede contratar (ambas cantidades mayores que 0). De las personas contratadas interesa su número de cédula (que las identifica) y su nombre. De los contratos vigentes interesa su número (que los identifica) y la fecha en la que fueron celebrados. Para los contratos firmados a término, interesa además la remuneración acordada (que debe ser siempre superior a la remuneración mínima de todas las empresas que lo suscriben) y su fecha de expiración (que debe ser mayor a la fecha de celebración). En cambio, para los contratos de jornaleros interesa además de la información general del contrato, la cantidad de jornales contratados (que debe ser mayor que la cantidad mínima de jornales de todas las empresas que lo suscriben).

Una empresa puede tener muchos contratos vigentes. Una persona también. Un contrato siempre es suscrito por una persona y por una o más empresas que se comprometen solidariamente a cumplir las obligaciones del mismo. Finalmente, para cada empresa, el sistema deberá registrar todas las personas que han sido contratadas alguna vez por la misma. En particular, este registro debe incluir a aquellas personas con contratos vigentes actualmente.

Se pide:

i) Modelar la realidad planteada mediante un Diagrama de Modelo de Dominio UML.



ii) Expresar todas las restricciones del modelo en lenguaje natural.

- RUT identifica Empresa, cédula identifica Persona y número identifica contrato (unicidad)
- Remuneración mínima y mínimo de jornales mayores que 0 (dominio de atributos)
- Si una empresa tiene un contrato vigente con una persona, esa persona está incluida en el registro histórico de personas contratadas. (circular)
- Para todo contrato a término su remuneración es mayor que la remuneración mínima de todas las empresas involucradas. (regla de negocios)
- Para todo contrato de jornalero, la cantidad de jornales es mayor que la cantidad de jornales mínima de todas las empresas involucradas. (regla de negocios).
- La fecha de expiración debe ser mayor que la fecha de celebración de un contrato.

iii) A partir del modelo definido en la primera parte, exprese las siguientes restricciones en OCL:

- Si una empresa tiene un contrato vigente con una persona, esa persona está incluida en el registro de todas las personas que han sido contratadas alguna vez por la misma.

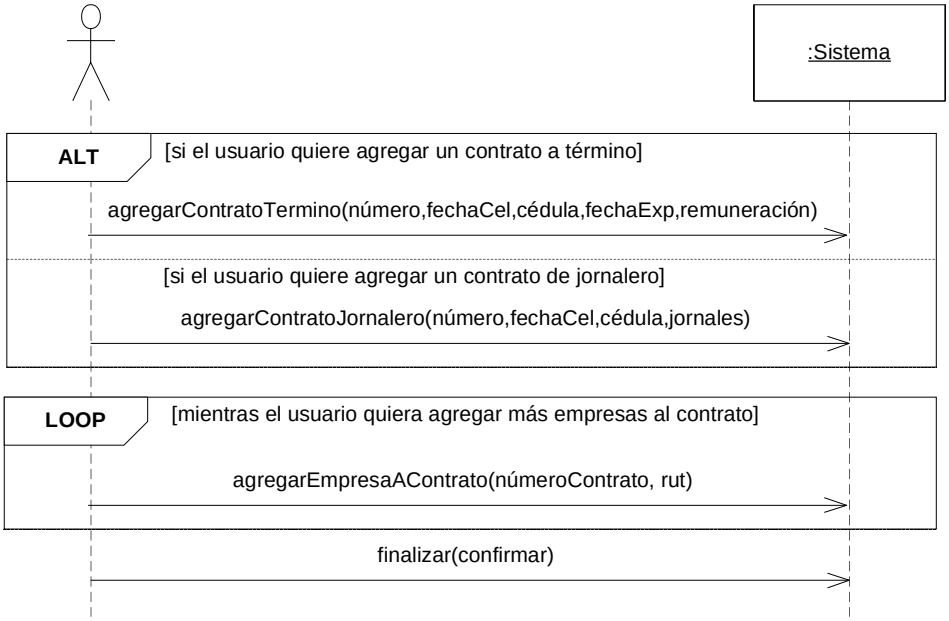
```
context Empresa inv:
  self.persona->includesAll(self.contrato.persona)
```

- Para todo contrato a término, su remuneración es mayor que la remuneración mínima de todas las empresas que lo suscriben.

```
context Termino inv:
  self.empresa.remuneracionMinima->forAll(r | r < self.remuneracion)
```

c) Realice el Diagrama de Secuencia del Sistema para el siguiente caso de uso:

Nombre	Agregar un nuevo contrato	Actores	Usuario
Descripción	El caso de uso comienza cuando el usuario indica al sistema el número de contrato a agregar, su fecha de celebración, la cédula del contratado y el tipo de contrato. En caso de ser un contrato a término, el usuario indica también la remuneración y la fecha de expiración. En caso contrario, indica solamente la cantidad de jornales contratados. Luego, el usuario selecciona una a una de una lista las empresas que suscriben el contrato. Para finalizar, el usuario confirma o cancela el alta al sistema.		

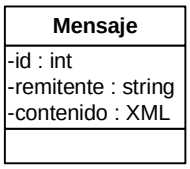


Problema 2 (35 puntos)

a) Mencione las actividades y artefactos de la etapa de Diseño de la metodología Iterativa & Incremental vista en el curso.

Actividad	Artefacto
Diseño de Interacciones	Diagrama de Comunicación
Diseño de Estructura	Diagrama de Estructura Estática (Diagrama de Clases)

b) Una cola de mensajes es una estructura de datos que recibe un mensaje a publicar (por parte de una aplicación cliente), lo almacena en la estructura (típicamente de tipo FIFO) y lo pone a disposición (de cualquier aplicación cliente, típicamente otra diferente a quien remitió el mensaje). El acceso a la cola será en este caso atómico y serial (no existirá acceso en paralelo a la misma). Las colas de mensajes permiten la distribución de aplicaciones pero manteniendo una comunicación asincrónica y desacoplada entre ellas mediante el envió de mensajes a la cola (de forma que las aplicaciones cliente no tengan que conocerse entre sí). Los mensajes a encolar tienen la siguiente estructura:



La forma de colocar un mensaje en una cola es invocando la operación `put(m:Mensaje)`. Esta operación encola el mensaje recibido por parámetro y avisa a las aplicaciones cliente de la cola que se ha encolado un mensaje, enviándoles a éstas el ID del mensaje.

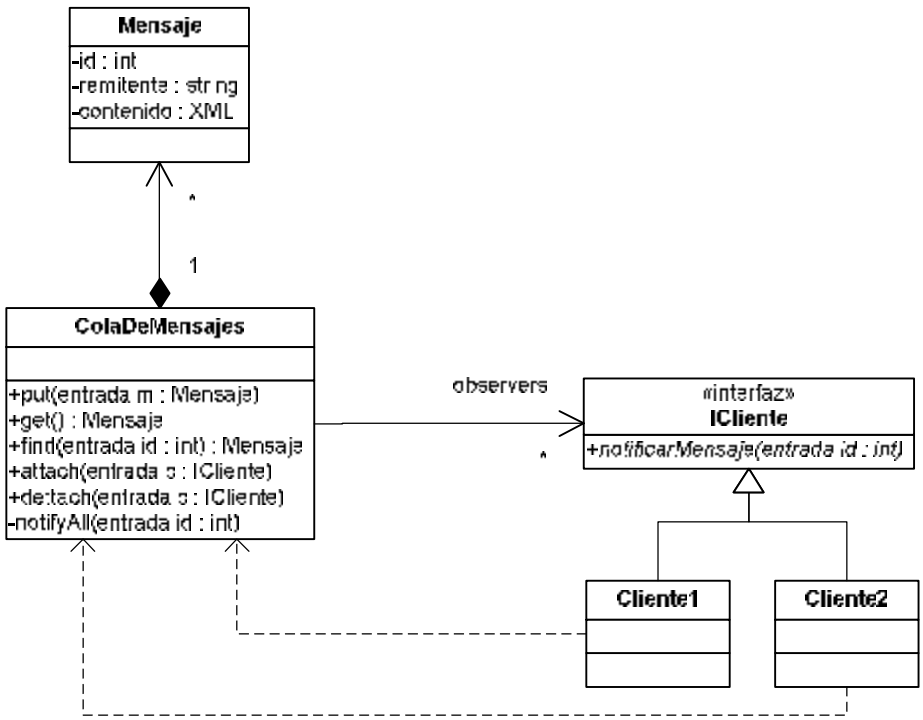
La forma de obtener un mensaje de la cola es invocando la operación `get():Mensaje` que devuelve el primer mensaje de la cola y lo elimina de la misma. En case de no haber mensajes la operación devuelve un mensaje vacío con ID cero para indicar el error.

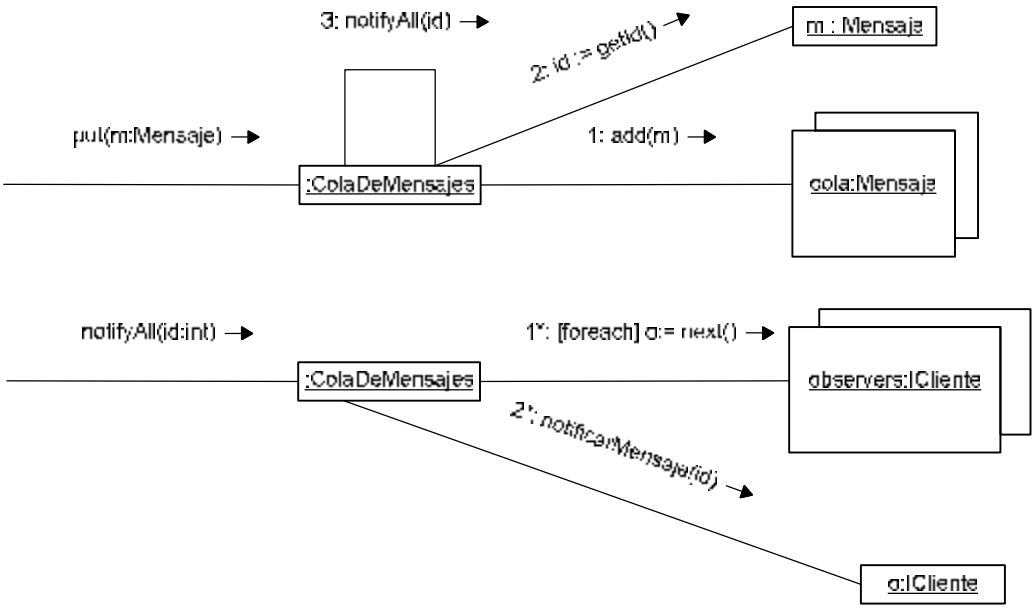
Otro servicio que brinda la cola es la de buscar un mensaje específico mediante la operación `find(id:int):Mensaje` que devuelve el primer mensaje que encuentre cuyo identificador coincida con el pasado por parámetro y lo elimina de la cola. En case de no encontrarlo la operación devuelve un mensaje vacío con ID cero para indicar el error.

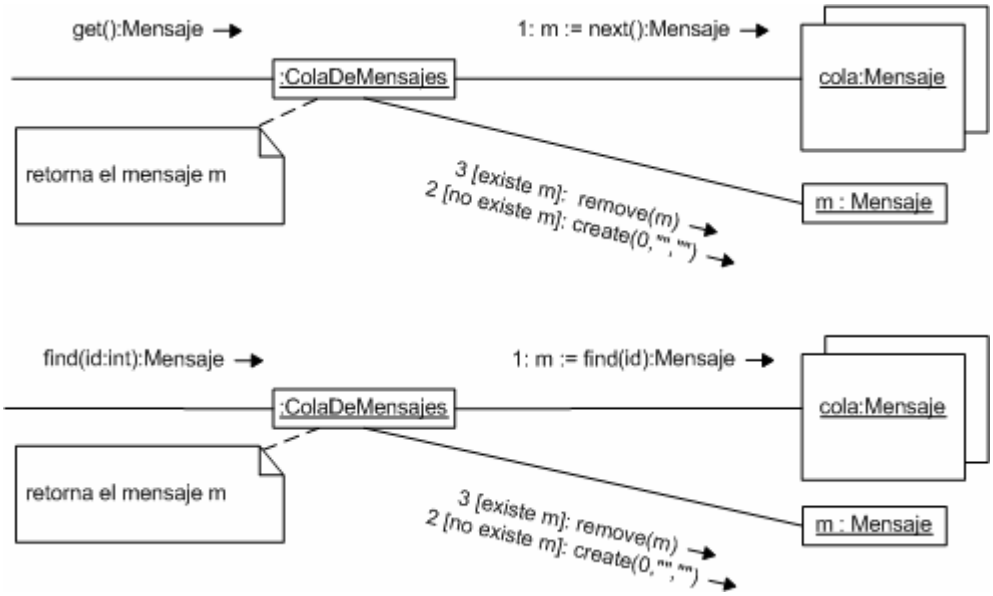
Se pide:

i. Diseñe la cola de mensajes así como el mecanismo para colocar, avisar y obtener mensajes de dicha cola, utilizando patrones de diseño. Se espera un Diagrama de Clases de Diseño (DCD) de la solución así como el diseño de las operaciones put, get y find (Diagrama de Comunicación).

- El mecanismo debe considerar la posibilidad de agregar nuevas aplicaciones cliente en el futuro sin afectar el diseño de la cola.
- Se debe explicitar en los diagramas la creación de instancias.


Mensaje
-id : int
-remitente : string
-contenido : XML
*
1
ColaDeMensajes
+put(entrada m : Mensaje)
+get() : Mensaje
+find(entrada id : int) : Mensaje
+attach(entrada o : ICliente)
+detach(entrada o : ICliente)
-notifyAll(entrada id : int)
observers
*
«interfaz»
ICliente
+notificarMensaje(entrada id : int)
Cliente1
Cliente2


3: notifyAll(id)
2: id := getId()
m : Mensaje
put(m:Mensaje)
1: add(m)
:ColaDeMensajes
cola:Mensaje
notifyAll(id:int)
1*: [foreach] o:= next()
:ColaDeMensajes
observers:ICliente
2*: notificarMensaje(id)
o:ICliente



ii. Explique qué patrón(es) de diseño utilizó indicando (para cada patrón) las clases participantes y sus roles.

Se utilizó Observer de la siguiente forma:

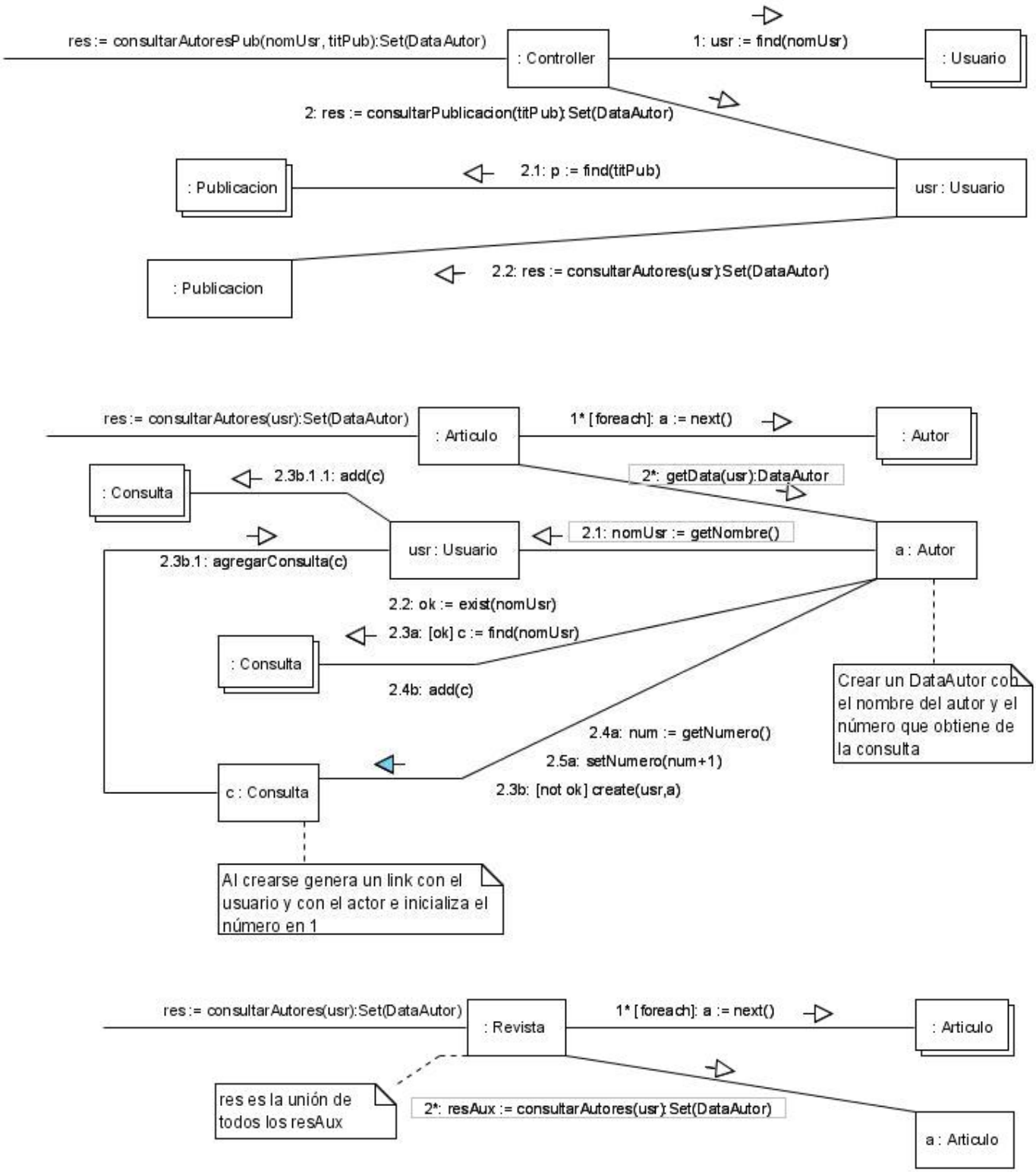
Subject: ColaDeMensajes
Observer: ICliente
ConcreteObserver: Cliente1, Cliente2

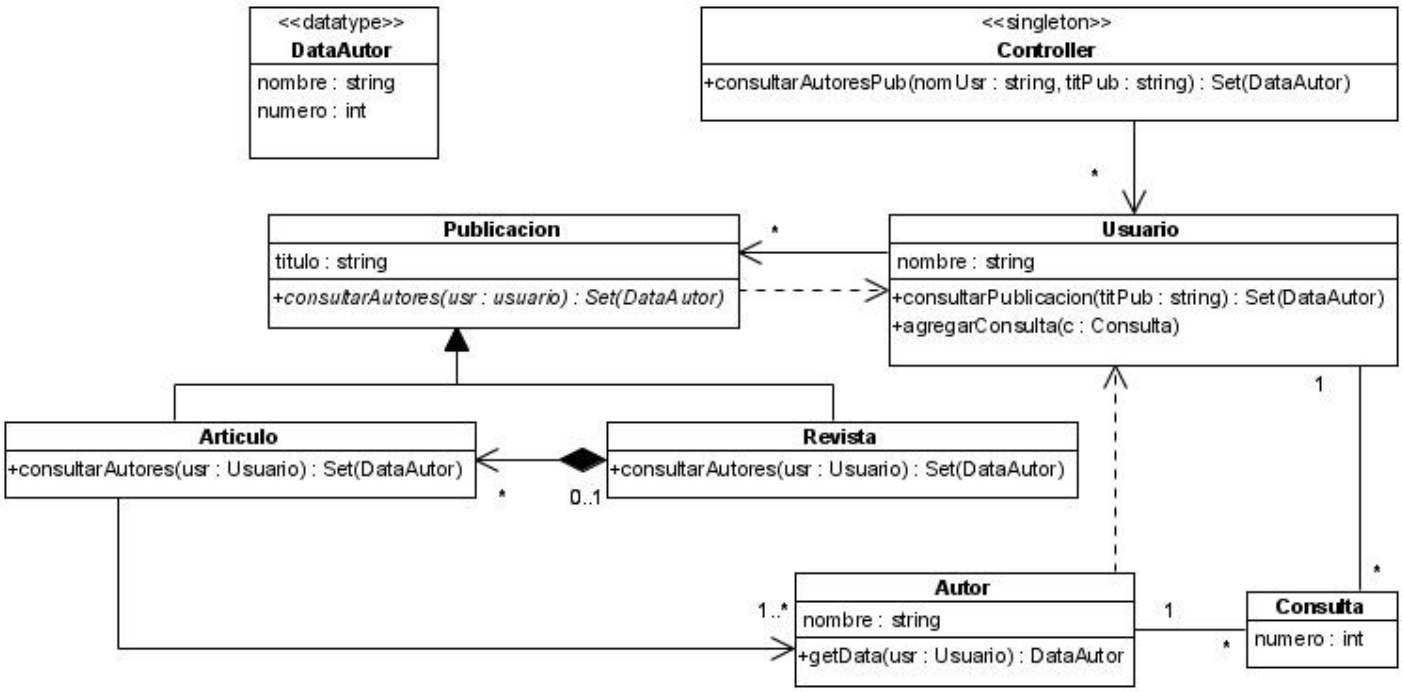
iii. ¿Qué modificaciones haría al diseño en caso de incorporar una prioridad a cada mensaje, modificando la operación `get` para que reciba un parámetro y obtenga el primer mensaje cuya prioridad sea la pasada por parámetro? No es necesario rehacer los diagramas.

Simplemente agregar un atributo prioridad de tipo entero en la clase Mensaje y hacer que el `get(prioridad:int):Mensaje` de `ColaDeMensajes` busque en su colección de mensajes el primero con la prioridad pasada por parámetro.

Problema 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`

```
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`. No incluya constructores, destructores ni operaciones `set` y `get` de atributos.

```
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);
	}
}
```

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.