Programación 4

PARCIAL FINAL EDICIÓN 2009

SOLUCIÓN

Problema 1 (35 puntos)

a) Indique brevemente la relación entre los siguientes elementos: Caso de Uso, Escenario, Operación del Sistema, Contrato y Diagrama de Secuencia del Sistema.

b) En una ciudad extranjera se está a punto de desarrollar un nuevo sistema de transporte colectivo similar al Sistema de Transporte Metropolitano (STM) de la ciudad de Montevideo. El nuevo sistema a desarrollar considerará líneas de transporte (compuestas por tramos) y registrará los viajes realizados sobre esos tramos y la forma en que éstos fueron abonados. Cada tramo se origina en una parada y termina en otra parada (diferente a la original) totalizando cierta cantidad de kilómetros entre ambas. Así por ejemplo la línea número 808 tendrá diez tramos: el primero de 0,8 kilómetros entre las paradas "Plaza JC" (ubicada en la esquina de las calles X y Y) y "Monumento a DC" (ubicado en la esquina de las calles A y B); el segundo… y así sucesivamente. Existirán dos medios de pago en este nuevo sistema de transporte: tickets y tarjetas inteligentes. Ambos permitirán a los pasajeros viajar durante cierto período de tiempo, pero existen restricciones diferentes para cada medio de pago. Mientras que un ticket permite utilizar su tiempo únicamente para 1 o 2 viajes, la tarjeta permite utilizar su tiempo para cualquier cantidad de viajes. En ambos casos no se podrán tomar nuevos viajes si se ha agotado el tiempo del medio de pago. Cada viaje incluirá su duración de forma de poder controlar la validez de los medios de pago.

Se pide:

i) Diagrama de dominio de la realidad planteada con restricciones en lenguaje natural.

ii) Especificar en OCL únicamente las dos siguientes restricciones:
- unicidad del número de línea
- la suma de las duraciones de los viajes sea menor o igual a la duración del medio de pago

iii) Diagrama de Secuencia del Sistema basándose únicamente en el siguiente caso de uso (asumiendo que existe un único escenario para el caso de uso):

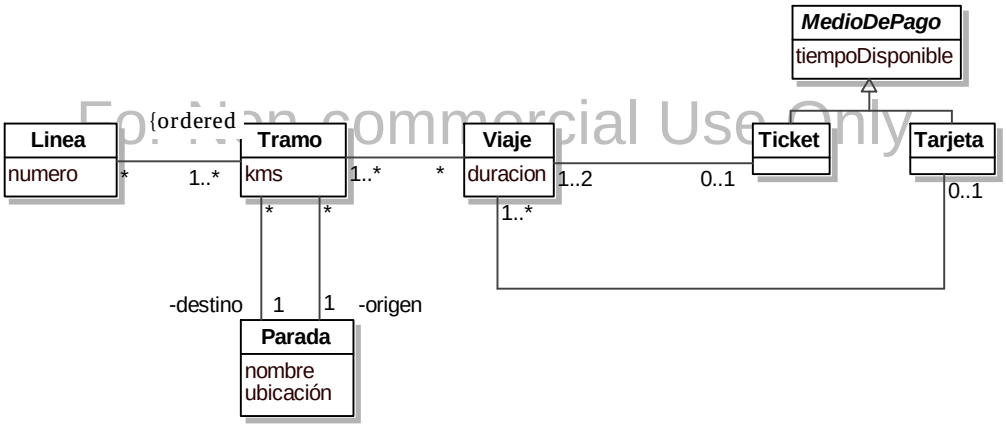
Nombre	Alta de Línea	Actores	Administrador del Sistema
Descripción	El caso de uso comienza cuando el administrador del sistema selecciona la opción de Alta de Línea. El administrador deberá primero ingresar el número de la línea, a lo cual el sistema le devolverá una lista con toda la información de las paradas de las cuales el administrador deberá elegir (mediante su nombre) una de ellas como la próxima parada. Así continuará eligiendo las paradas que conformarán los tramos de la línea hasta que decida terminar el ingreso. Por tanto cada vez que el administrador selecciona una parada, ésta será la parada de destino de un tramo y a su vez la parada de origen del tramo siguiente (si es que éste existe). La distancia (en kilómetros) entre cada par de paradas será calculada automáticamente por el sistema.		

Solución

a) Un caso de uso tiene uno o varios escenarios. Para cada escenario se realiza un DSS el cual contiene operaciones del sistema. Para cada operación del sistema se realiza un contrato.

b)

i)



Restricciones en lenguaje natural:

- El número de línea identifica a cada línea.
- Las paradas de origen y destino de un tramo son diferentes.
- La parada de destino de un tramo es la de origen del tramo siguiente.
- Todos los tramos de un viaje se corresponden con la misma línea.
- Un viaje es pagado por medio de un ticket o por medio de una tarjeta, no por ambos.
- La suma de las duraciones de los viajes realizados con un medio de pago es menor o igual al tiempo disponible de ese medio de pago.

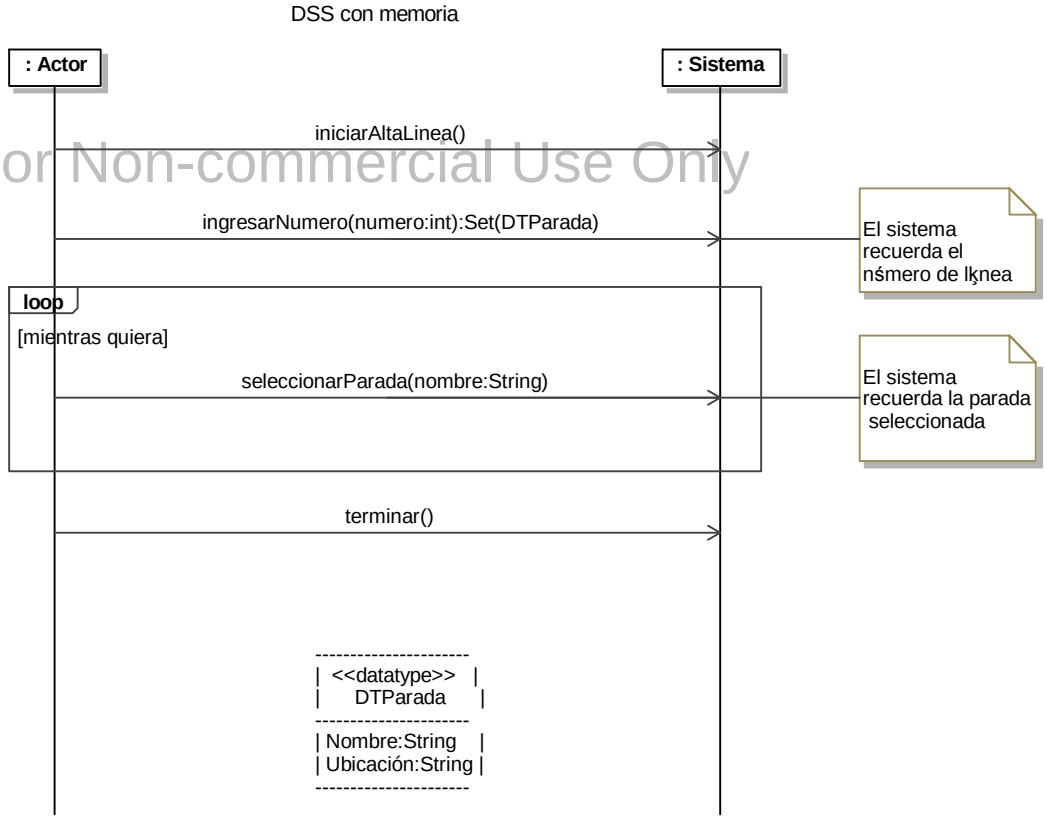
ii)

Restricciones en OCL:

- context Linea inv:
  Linea.allInstances->isUnique(numero)

- context MedioDePago inv:
  self.viaje.duracion->sum() <= self.tiempoDisponible

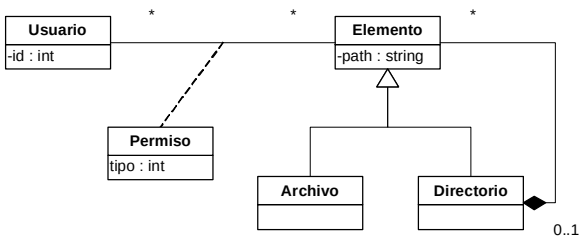
iii)



Problema 2 (35 puntos)

a) Responda en no más de 3 líneas cada una de las siguientes preguntas
   i) Nombre el patrón de diseño que permite que un objeto varíe su comportamiento cuando su estado interno cambia dando la apariencia de que el objeto cambió de clase.
   ii) Nombre el patrón de diseño que define una dependencia 1-n entre objetos y permite que cuando uno lo desee todos los dependientes sean notificados.
   iii) Nombre el patrón de diseño que provee un mecanismo para controlar el acceso a una instancia.
   iv) ¿Cuál es la intención del patrón Template Method?

b) Se analizó un sistema de archivos que permite asignar permisos a usuarios del sistema operativo. El modelo de dominio realizado se puede ver en la figura siguiente. Los elementos del sistema de archivos pueden ser archivos o directorios, quien a su vez puede estar formado por otros elementos. Los elementos se identifican por su dirección absoluta (path). Sobre los elementos se definen permisos de acceso de cierto tipo que se asocian independientemente a cada usuario del sistema.



Además, se definieron dos operaciones que permiten asignar permisos y consultarlos posteriormente. Estas operaciones están definidas por los siguientes contratos reducidos.

Operación	altaDePermiso(idUsr:Integer, pathEl:String, tipoP:Integer)
Pre- y poscondiciones	
Pre: Existe una instancia de Usuario con id igual a idUsr. pre: Existe una instancia de Elemento con path igual a pathEl. pre: No existe una instancia del tipo asociativo Permiso con links a las instancias de Usuario y Elemento identificadas anteriormente. En caso de que la instancia de Elemento sea un Directorio, ninguno de sus Elementos están recursivamente linkeados con instancias de Permiso para ese mismo Usuario. post: Se creó una instancia de Permiso con tipo=tipoP y se asoció a las instancias de Usuario y Elemento identificadas en las precondiciones. Si el elemento era un Directorio entonces para todas las instancias de Elemento que lo componen se creó una instancia de Permiso como se definió antes, y así recursivamente.	

Operación	consultarPermisos(idUsr:Integer):Set(DataPermiso)
Pre- y poscondiciones	
Pre: Existe una instancia de Usuario con id igual a idUsr. post: Para cada instancia de Elemento con el cual esté linkeado la instancia de Usuario, por medio del tipo asociativo Permiso, se define un datavalue de tipo DataPermiso que contiene el tipo de Permiso y path del Elemento relacionado con el Usuario. Se devuelve el conjunto de datavalues en cuestión.	

Se pide:

   i) Realice los Diagramas de Comunicación para las operaciones especificadas en los contratos. No es necesario indicar las visibilidades.

   ii) Realice el Diagrama de Clases de Diseño completo de la solución.

Solución

a)

i) State

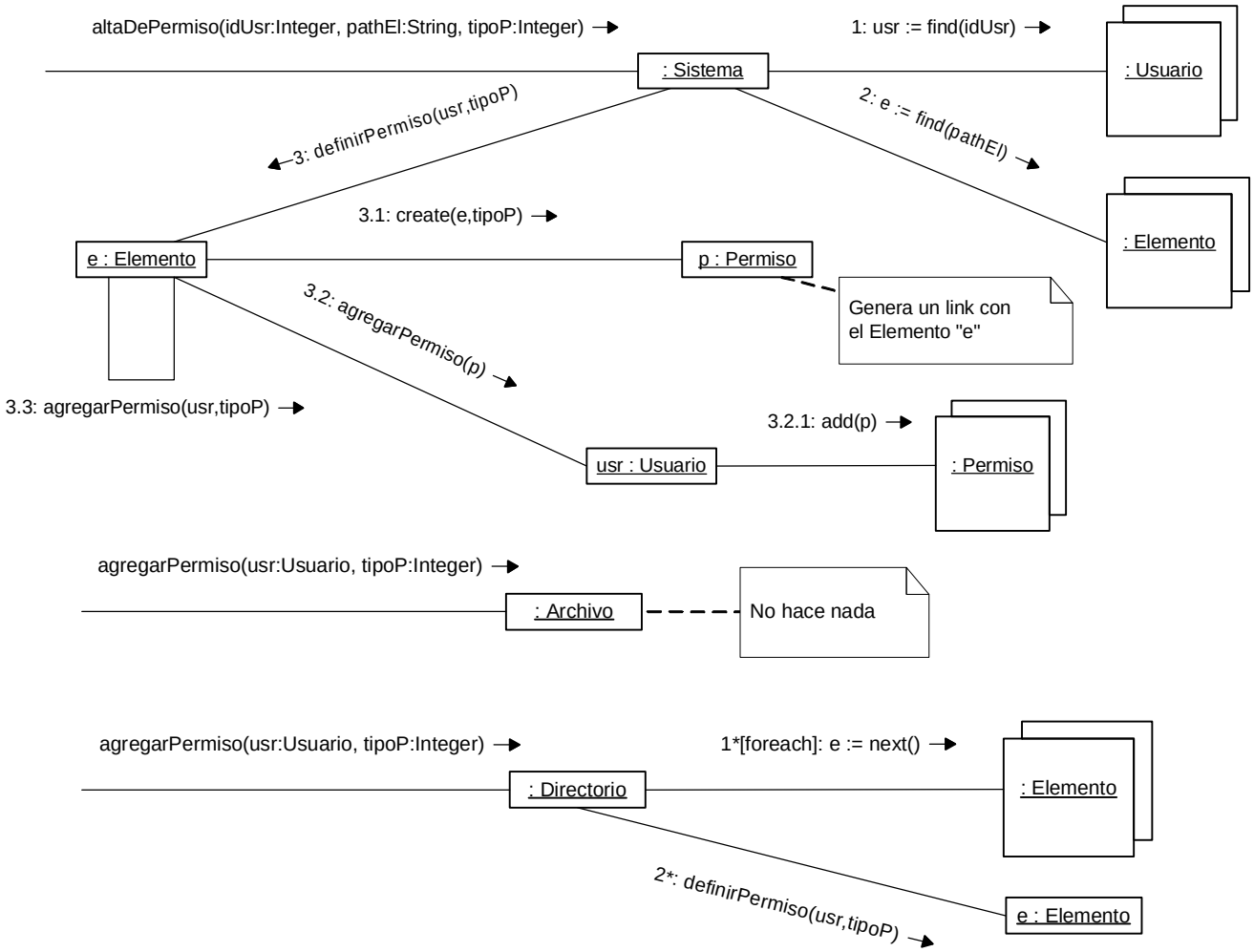
ii) Observer

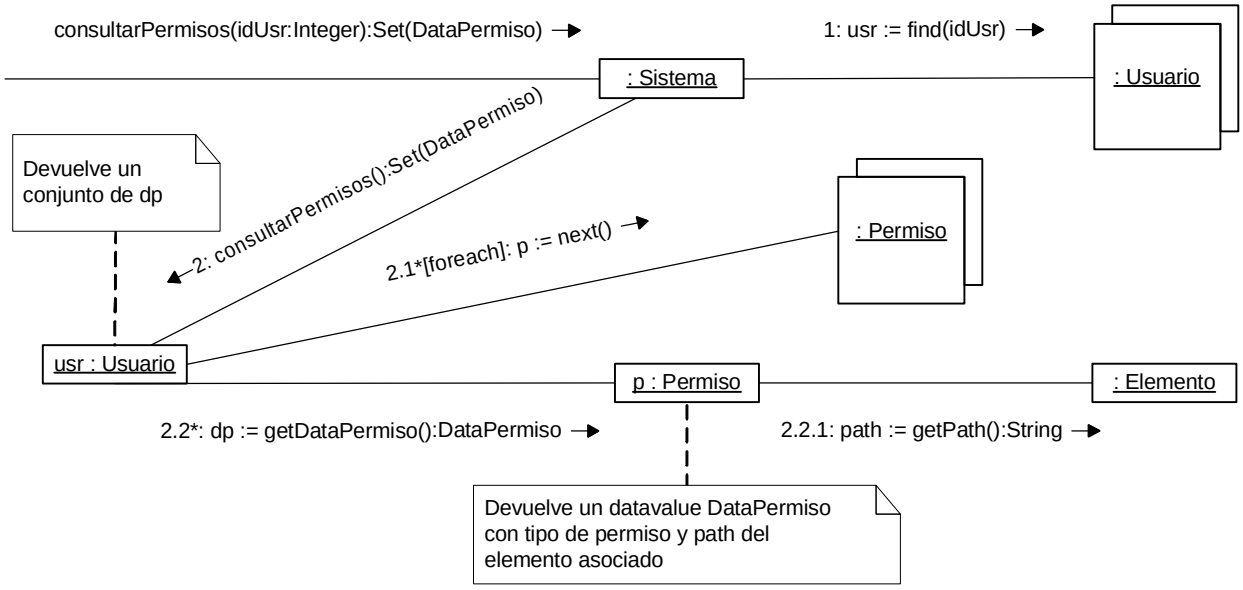
iii) Proxy

iv) Definir el esqueleto de un algoritmo en una operación, delegando algunos pasos a sus subclases. Permite redefinir ciertos pasos de un algoritmo sin modificar la estructura del mismo.

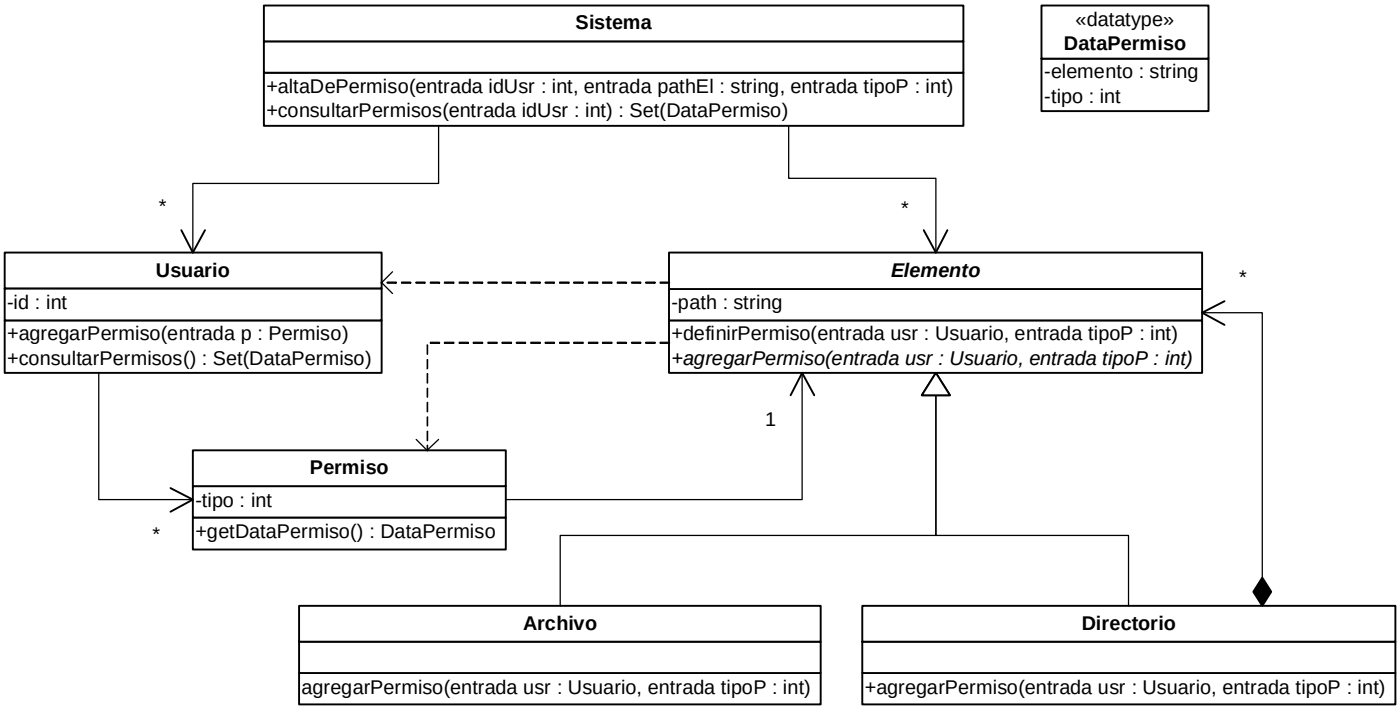
b)

i) Diagramas de Comunicación





ii) Diagrama de Clases de Diseño

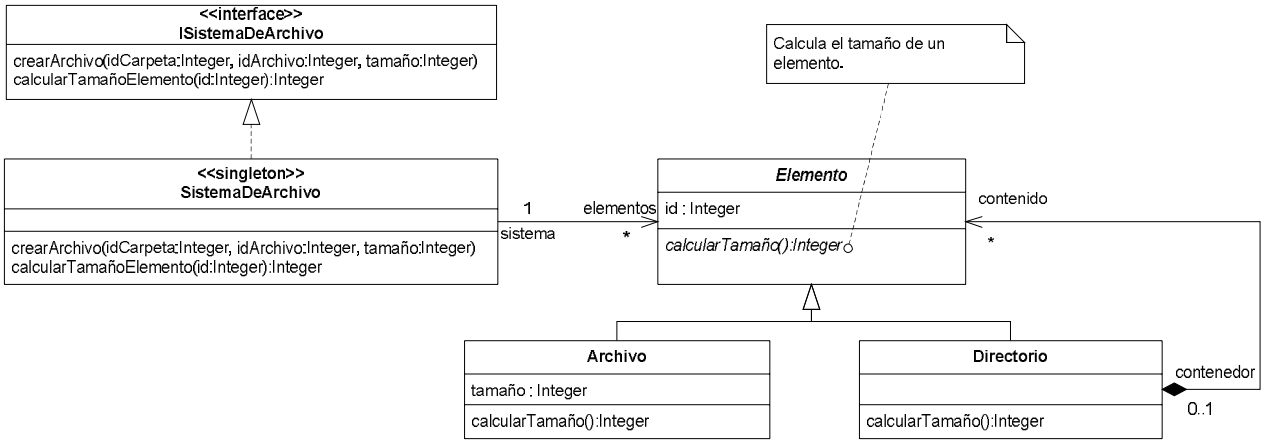


Problema 3 (30 puntos)

a) Enumere los diferentes tipos de relaciones entre elementos que pueden aparecer en un diagrama de clases de diseño e indique como se implementan en C++.

b) Se le ha encomendado la tarea de implementar un prototipo de sistema de archivos. El sistema de archivos contendrá elementos, los cuales tendrán un identificador y podrán ser de dos tipos: directorios, que a su vez podrán contener a otros elementos; o archivos que registrarán el tamaño ocupado. En particular, se le ha pedido implementar las operaciones para crear un archivo en un directorio y calcular el espacio total ocupado por un elemento del sistema.

Se le ha entregado el siguiente diseño parcial que usted deberá respetar:



Se le ha entregado también las siguientes especificaciones para las operaciones pedidas:

Operación	SistemaDeArchivo::crearArchivo(idDirectorio, idArchivo, tam)
Descripción	Crea el archivo con el atributo id = idArchivo y tamaño = tam. Asocia el archivo creado con el directorio y sistema de archivos que lo contienen.

Operación	SistemaDeArchivo::calcularTamañoElemento(idElemento)
Descripción	Calcula el tamaño del elemento cuyo atributo id = idElemento. Si el elemento es un archivo, el resultado es su tamaño; si es un directorio es la suma de los tamaños de los elementos que contiene.

Se pide:

i) Implementar los .h de la interfaz ISistemaDeArchivo y de las clases SistemaDeArchivo, Elemento, Archivo y Directorio.

ii) Implementar en C++ las operaciones especificadas anteriormente y todas aquellas operaciones necesarias para su implementación que pertenezcan a las clases SistemaDeArchivo, Elemento, Archivo y Directorio.

Observaciones:

- Asuma que existe una implementación estándar de las interfaces ICollectible, ICollection, IIterator, IDictionary e IKey.

- Asuma que existe una clase Lista que realiza las interfaces ICollection e IDictionary y una clase KeyInteger que realiza la interfaz IKey.
- No defina colecciones concretas.
- No incluya directivas de preprocesador.

- Se debe respetar el diseño parcial dado.

Solución

a)

- Dependencia.
  Se implementa utilizando la directiva de preprocesador #include:
  #include"ClaseRelacionada.h"
- Asociación con multiplicidad mayor que 1.
  Se implementa mediante la utilización de colecciones o diccionarios:
  ICollection* nombreDeRol;
- Asociación con multiplicidad 1 o 0..1.
  Se implementa mediante atributos de tipo puntero a la clase relacionada:
  ClaseRelacionada* nombreDeRol;
- Generalización.
  Se implementa mediante la utilización de la herencia pública de C++:
  class A : public B
- Realización.
  Se implementa mediante la utilización de la herencia pública de C++:
  class A : public IA

b)

```
class ISistemaDeArchivo {
	public:
		virtual void crearArchivo(int idDir, int idArch, int tam) = 0;
		virtual int calcularTamanoElemento(int id) = 0;

		virtual ~ISistemaDeArchivo();
};

class SistemaDeArchivo : public ISistemaDeArchivo {
	public:
		static SistemaDeArchivo* getInstance();
		~SistemaDeArchivo();
		void crearArchivo(int idDir, int idArch, int tam);
		int calcularTamanoElemento(int id);
	private:
		static SistemaDeArchivo* instancia;
		IDictionary* elementos;
		SistemaDeArchivo();
};

class Elemento: public ICollectible {
	public:
		Elemento(int id);
```

```
		~Elemento();
		virtual int calcularTamano() = 0;
	private:
		int id;
};

class Archivo: public Elemento {
	public:
		Archivo(int id, int tam);
		int calcularTamano();
	private:
		int tamano;
};

class Directorio : public Elemento {
	public:
		Directorio(int id);
		~Directorio();
		int calcularTamano();
		Archivo* crearArchivo(int idArch, int tam);
	private:
		IDictionary* contenido;
};

void SistemaDeArchivo::crearArchivo(int idDir, int idArch, int tam) {
	IKey* directoryKey = new KeyInteger(idDir);
	Directorio* directorio = dynamic_cast<Directorio*>(elementos-
>find(directoryKey));
	delete directoryKey;

	if (directorio != NULL) {
		Archivo* archivo = directorio->crearArchivo(idArch, tam);
		elementos->add(new KeyInteger(idArch), archivo);
	} else {
		throw invalid_argument("Directorio no válido");
	}
}

int SistemaDeArchivo::calcularTamanoElemento(int id) {
	IKey* ek = new KeyInteger(id);
	Elemento* elem = (Elemento*) elementos->find(ek);
	delete ek;

	if (elem != NULL)
		return elem->calcularTamano();
	else
		throw invalid_argument("No existe elemento con el id pasado");
```

```
}

Elemento::Elemento(int id) : id(id) {
}

Elemento::~Elemento() {
}

int Directorio::calcularTamano() {
	IIterator* it = this->contenido->getIterator ();
	Elemento* elemento;
	int tam = 0;
	while(it->hasCurrent ()) {
		elemento = (Elemento*) it->getCurrent();
		tam += elemento->calcularTamano();
		it->next();
	}
	delete it;
	return tam;
}

Archivo* Directorio::crearArchivo(int idArch, int tam) {
	Archivo* archivo = new Archivo(idArch, tam);
	contenido->add(new KeyInteger(idArch), archivo);
	return archivo;
}

Archivo::Archivo(int id, int tam) :Elemento(id), tamano(tam) {
}

int Archivo::calcularTamano() {
	return this->tamano;
}
```