Programación 4

EXAMEN FEBRERO 2010

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) Mencione 3 construcciones de los Diagramas de Estructura Estática de UML que NO son pertinentes en los Modelos de Dominio.

1. Operaciones
2. Navegabilidades
3. Dependencias

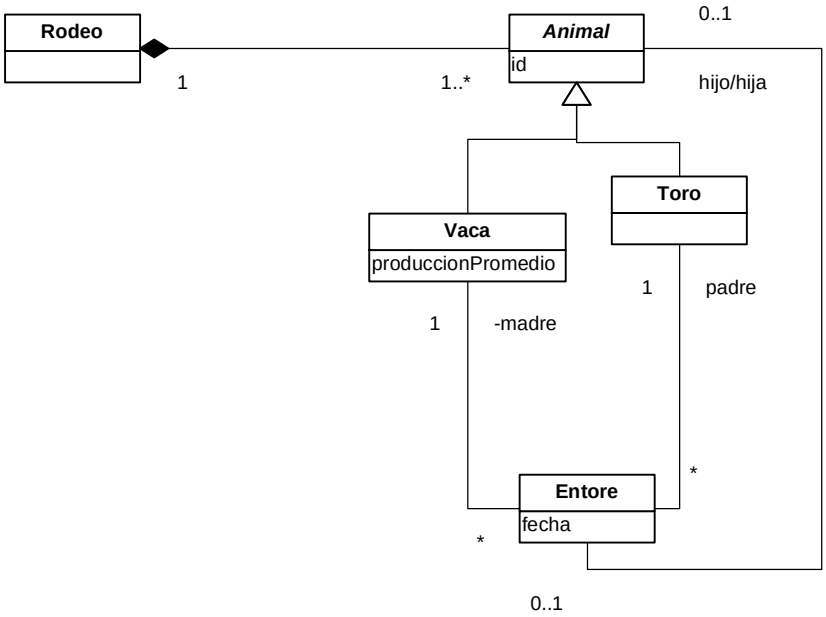
b) Un establecimiento agropecuario le ha encomendado la tarea de desarrollar un software para el manejo de sus rodeos.

Un rodeo es un conjunto disjunto de animales (para el caso de este establecimiento: vacas y toros). Todos los animales se encuentran identificados por su caravana (la cual escaneada con el dispositivo adecuado, emite el número identificatorio del animal). Dado que el establecimiento reporta leche a CONAPROLE, es de interés saber la producción promedio de leche de cada vaca, la cual varía de vaca en vaca. Si bien el establecimiento no cuenta con grandes reproductores, le interesa llevar el control de cuántos toros posee dentro de cada rodeo así como la ascendencia de cada uno de ellos por parte paterna (lo cual luego permite a los expertos identificar los mejores toros para cruzar con las vacas).

Al cruzamiento entre toro y vaca se lo conoce como entore, y el establecimiento desea conocer todos los entores ocurridos, y para cada uno, entre qué animales fue, cuándo ocurrió y qué resultado tuvo. Debido a que el establecimiento mantiene por largos períodos de tiempo a sus animales, puede ocurrir que una vaca sea entorada muchas veces por el mismo toro, siempre del mismo rodeo. El resultado de un entore puede ser nulo (es decir que la vaca no se preñó) o puede ser un nuevo animal (es decir que la vaca efectivamente se preñó y dió a luz un nuevo animal). En este último caso (entore efectivo) interesa también saber qué animal fue producto de qué entore.

Se pide:

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



Nota 1: la asociación entre Entore y Animal posee un mínimo de cero del lado de Entore pues de poseer un 1 todo Animal debe saber de qué entore viene, por lo que se tendrían infinitos Animales para cumplir con el mínimo de 1.

Nota 2: no es necesaria una auto-relación en Toro pues por la relación entre Animal y Entore ya se sabe quién es el padre de cada Toro.

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

- id identifica al animal
- un animal no puede ser descendiente de si mismo
- no puede haber un entore de dos animales de diferente rodeo

iii) A partir del modelo definido en la primera parte, exprese la siguiente restricción en OCL: "no hay dos entores de la misma vaca y toro en la misma fecha".

```
context Entore inv:
Entore.allInstances()->forAll(e1, e2 | (e1 <> e2 and e1.madre =
e2.madre and e1.padre = e2.padre) implies e1.fecha <> e2.fecha)
```

iv) ¿Qué modificaciones haría al Modelo de Dominio si el resultado de un entore efectivo fueran muchos animales y no uno solo?

Colocando una asociación entre Entore y Rodeo en lugar de entre Entore y Animal.

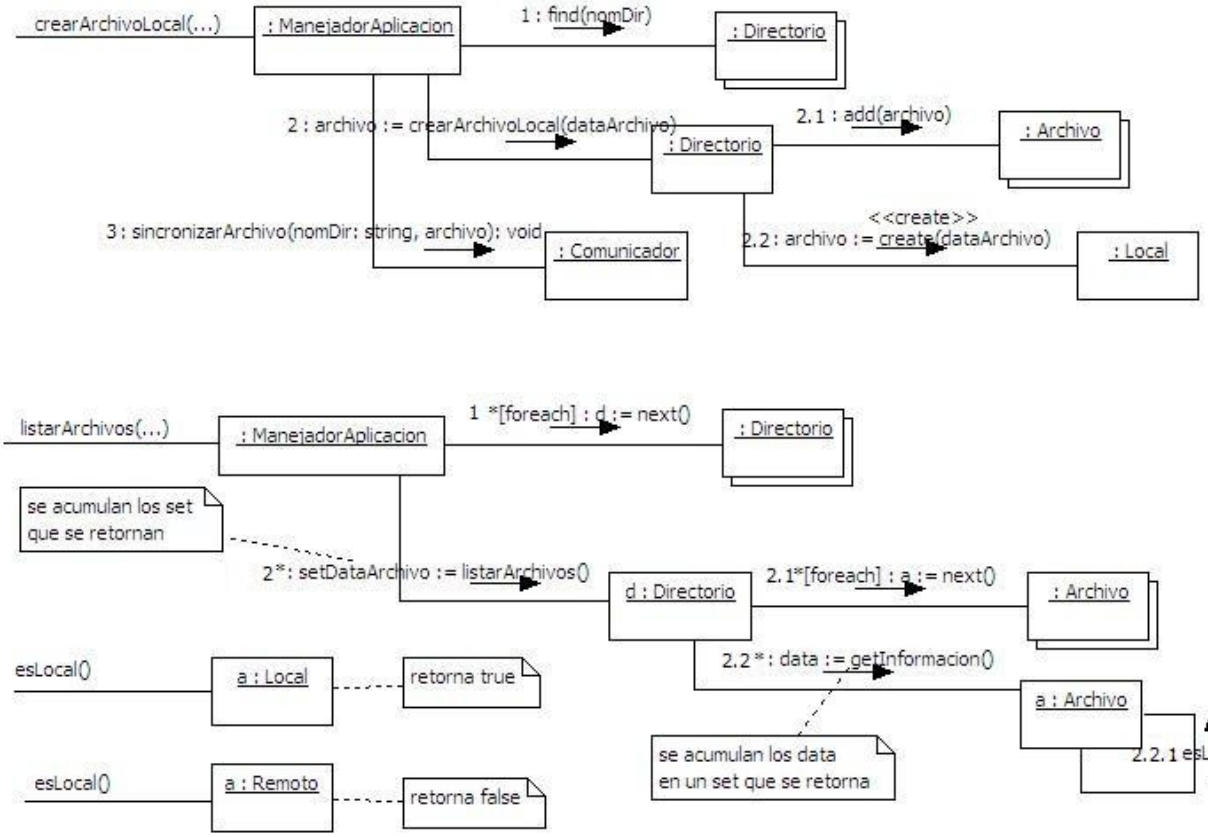
v) ¿Qué modificaciones haría al Modelo de Dominio de la parte i) si se desea saber cuáles fueron todos los toros con los cuales fue entorada una vaca, independientemente si el entore fue efectivo o no?

Ninguna modificación; ya que está contemplado en el modelo.

Problema 2 (35 puntos)

Se pide:

i. Realizar los Diagramas de Comunicación de las operaciones definidas anteriormente.



ii. Si utilizó algún patrón de diseño, especifique su nombre, clases participantes y roles.

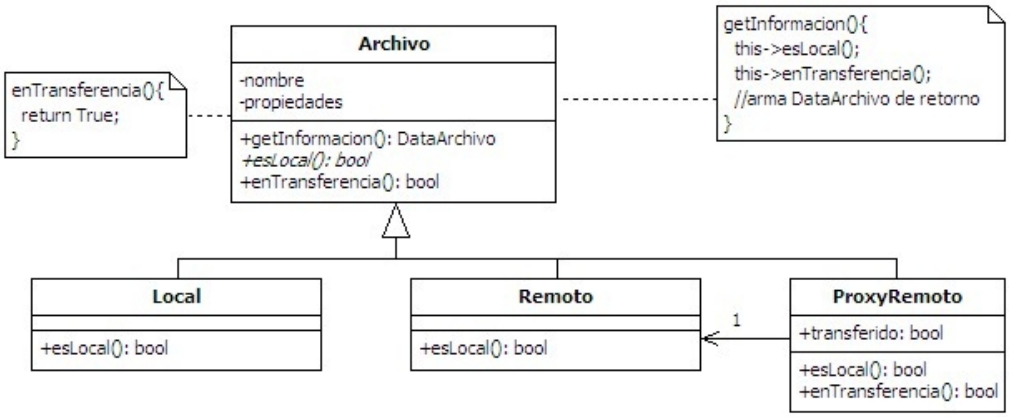
Patrón **Singleton**: `ManejadorAplicacion`

Patrón **Template Method**:

- Template Method: `Archivo::getInformacion(…)`
- Primitivos: `Local::esLocal(), Remoto::esLocal()`

Se pide:

iii. Modificar el Diagrama de Clases de Diseño original de manera tal de soportar este nuevo requerimiento con la restricción de que no se puede alterar las clases Local ni Remoto.



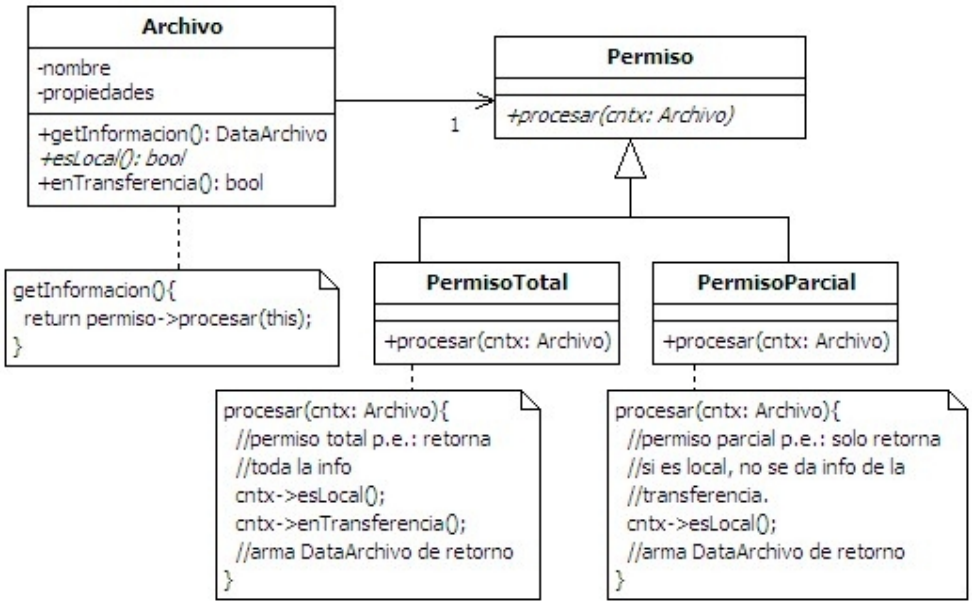
iv. Indique qué patrón de diseño utilizó, especifique su nombre, clases participantes y roles.

Patrón **Proxy**:

- Client: `Directorio`
- Subject: `Archivo`
- Real Subject: `Remoto`
- Proxy: `ProxyRemoto`

Se pide:

v. Modificar el Diagrama de Clases de Diseño original de manera tal de soportar este nuevo requerimiento.



vi. Indique qué patrón de diseño utilizó, especifique su nombre, clases participantes y roles.

Patrón **Strategy**:

- Client: `Directorio`
- Context: `Archivo`
- Strategies: `PermisoTotal, PermisoParcial`

Problema 3 (35 puntos)

Durante la construcción de un sistema de gestión de pólizas de seguros se logró el diseño parcial que se presenta a continuación, que incluye los Diagramas de Comunicación y el Diagrama de Clases de Diseño. El sistema registra los contratos firmados por clientes donde se adjuntan diferentes pólizas de seguros.

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

```
class Sistema {
private:
	static Sistema * instance;
	Sistema();

	IDictionary * clientes;
	IDictionary * polizas;

public:
	static Sistema * getInstance();
	DataPoliza consultarPoliza(int nro);
	float bajaContrato(String ced);

};
```

ii) Implemente en C++ el `.cc` ***de la clase*** `Sistema`***, salvo la operación*** `consultarPoliza()`***. Incluya código de manejo de excepciones en la operación*** `bajaContrato()`***para el caso en que no exista el cliente cuya cédula se utiliza en la eliminación.***

```
Sistema * Sistema::instance = NULL;

Sistema::Sistema() {
	this->polizas = new Lista();
	this->clientes = new Lista();
}

Sistema * Sistema::getInstance(){
  if (instance == NULL)
	instance = new Sistema();
  return instance;
}

float Sistema::bajaContrato(String ced){
	KeyString * cedKey = new KeyString(ced);
	Cliente * cli = (Cliente *) clientes->find(cedKey);

	if (cli == NULL)
		throw new Exception("El cliente no existe");
	else
		return cli->eliminarContrato();

}
```

iii) Implemente en C++ los *.h* ***de la jerarquía de clases compuesta por*** *Poliza*, *Automovil* ***y*** *Casa*. ***No incluya constructores, destructores ni operaciones*** *set* ***y*** *get* ***de atributos.***

```
class Poliza : public ICollectible{
private:
	int id;
	float costo;
	ICollection * adjuntas;

public:
	virtual String tipoPoliza() = 0;
	DataPoliza datosPoliza();
	void bajaAdjunta(Adjunta *);
	float getCosto();
};

class Automovil : public ICollectible{

public:
	String tipoPoliza();
};

class Casa : public ICollectible{

public:
	String tipoPoliza();
};
```

iv) Implemente en C++ la operación de la clase *Poliza DataPoliza datosPoliza()*.

```
DataPoliza Poliza::datosPoliza(){
	ICollection * adj = new Lista();
	String tipo = this->tipoPoliza();

	IIterator * iter = adjuntas->getIterator();

	while (iter->hasCurrent()){
		Adjunta * adjunta = (Adjunta *) iter->current();
		DataAdjunta * data = new DataAdjunta(
                                        adjunta->datosAdjunta());
		adj->add(data);
		iter->next();
	}

	return DataPoliza(tipo,this->costo,adj);
}
```

v) Implemente en C++ el destructor de la clase `Contrato`.

```
Contrato::~Contrato(){
	IIterator * iter = adjuntas->getIterator();

	while (iter->hasCurrent()){
		Adjunta * adjunta = (Adjunta *) iter->current();
		adjuntas->remove(adjunta);
		delete adjunta;
		iter->next();
	}

	delete adjuntas;

}
```