Programación 4

SOLUCIÓN EXAMEN DICIEMBRE 2010

Problema 1 (30 puntos)

a) ¿Cuáles son los 5 cambios de estado que se pueden especificar en una post-condición de un contrato de software?

Solución:

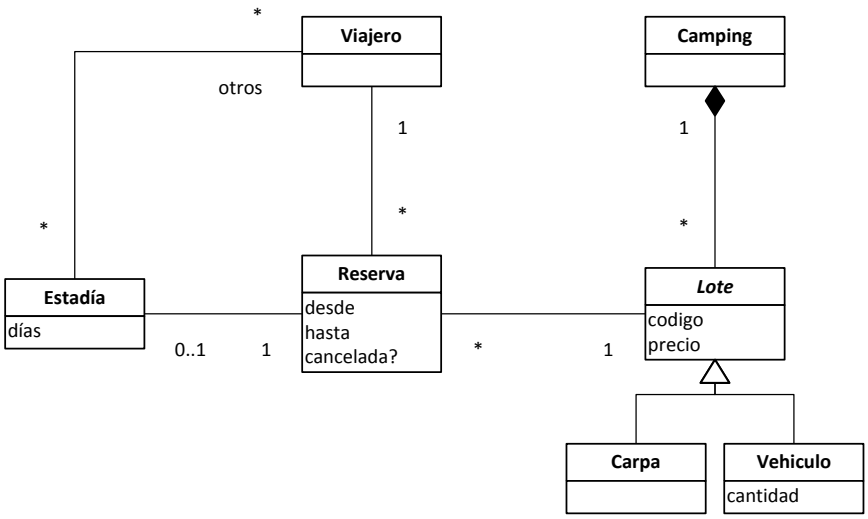
- Creación de instancias
- Destrucción de instancias
- Creación de links
- Destrucción de links
- Modificación de valor de atributo

b) Con la llegada del verano, se le solicita a su equipo el desarrollo de un sistema para el control del acceso a un camping de la costa uruguaya. El control será realizado mediante la lectura de las huellas dactilares. El cliente posee varios campings que desea controlar con un único sistema, en donde cada camping está compuesto de múltiples lotes de dos tipos: lotes para carpas y lotes para vehículos, y para el caso de lotes para vehículo interesa la cantidad de vehículos que entran a la vez. Todos los lotes poseen un código que los identifica y un precio. Un viajero que desee utilizar el camping deberá realizar una reserva sobre un único lote. No se permite la sobre-venta, es decir que se debe poder cumplir con todas las reservas. Cada reserva se realiza para un período de tiempo y para un lote, y una vez que el viajero que realizó la reserva llega al camping, se crea una estadía asociada a la reserva que indica cuántos días efectivamente se ocupó el lote y qué otras personas habitan el lote además de quien lo reservó. En caso de una cancelación, la reserva será guardada pero no estará asociada a ninguna estadía.

Se pide:

i) Modelo de Dominio de la realidad anterior con restricciones en lenguaje natural.

Solución:



Restricciones:

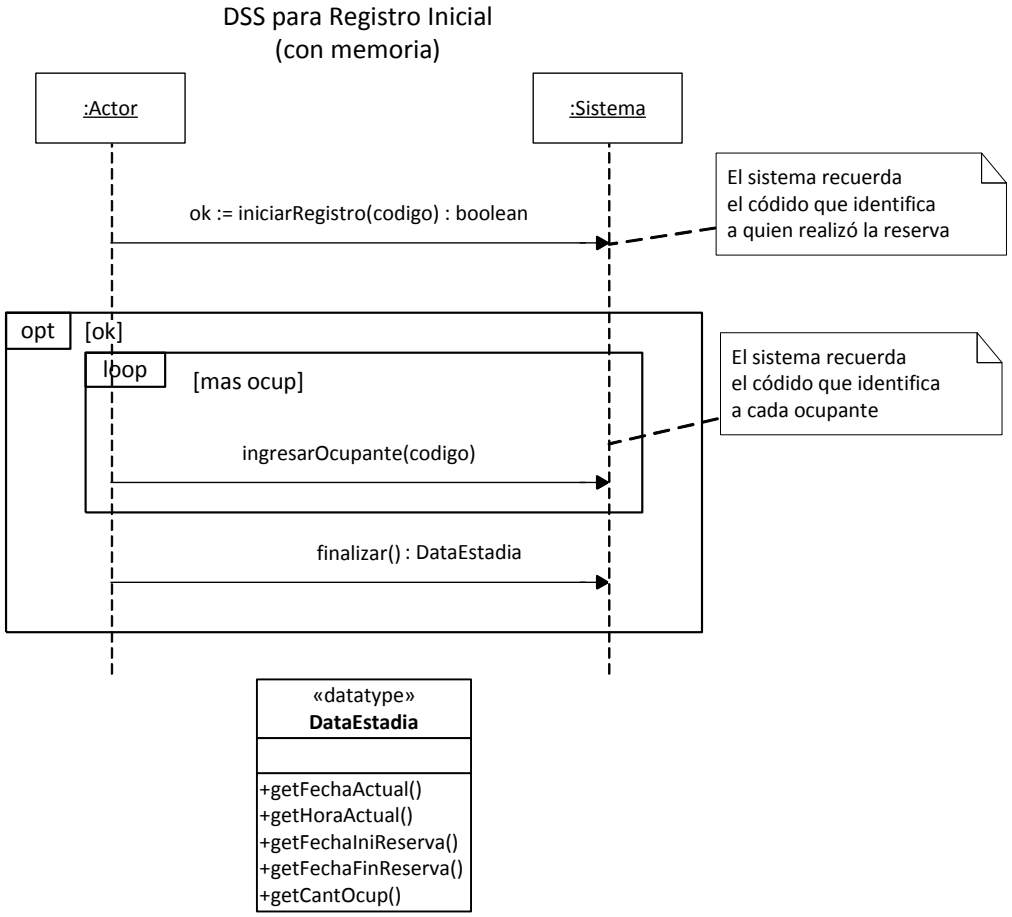
- El viajero que realiza la reserva no es parte de los otros ocupantes de la estadía.
- Un lote no puede estar asociado a dos reservas cuyos períodos se superpongan.
- La reserva podrá estar cancelada sólo si no tiene una estadía asociada.

- La fecha “hasta” debe ser mayor que “desde”.
- El código del lote lo identifica.

ii) Realizar el Diagrama de Secuencia del Sistema para el siguiente Caso de Uso, incluyendo datatypes en caso de ser necesario.

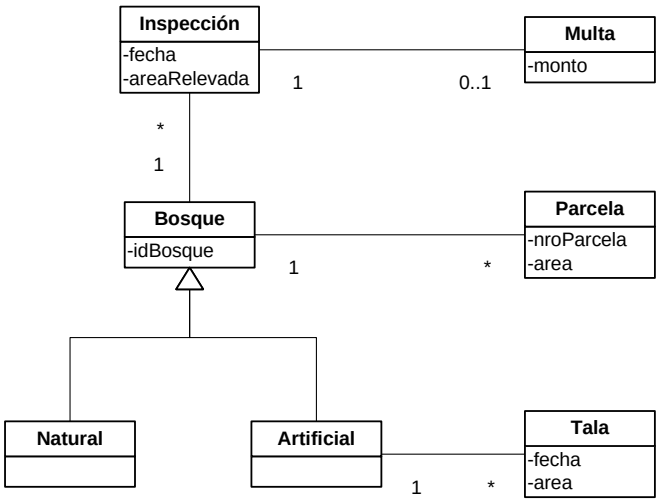
Nombre:	Registro Inicial
Actor:	Viajero que realizó la reserva
Sinopsis:	Este caso de uso comienza cuando el viajero que realizó la reserva coloca su pulgar sobre el sistema de identificación mediante huellas dactilares del camping. Esto envía un código hexadecimal al sistema que permitirá identificar al viajero de aquí en más. En caso de que el viajero no haya realizado una reserva previa, el sistema le notificará de esta situación y el caso de uso termina. En caso de que sí exista una reserva hecha por ese viajero, a continuación el sistema solicita las huellas dactilares de los demás ocupantes del lote, uno por uno. Para finalizar, el último ocupante selecciona una opción de finalizar el registro, a lo cual el sistema responde con la siguiente información: fecha y hora actual, fechas de la reserva y cantidad total de ocupantes.

Solución:



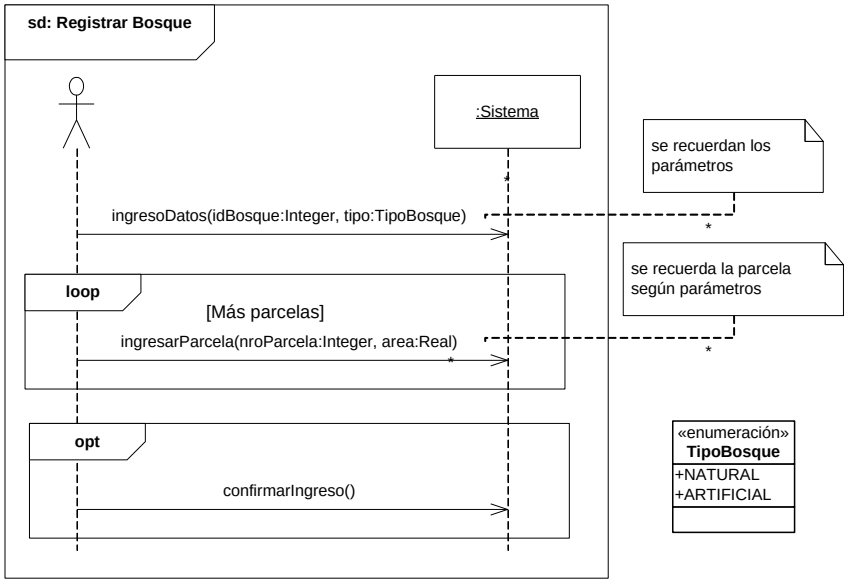
Problema 2 (35 puntos)

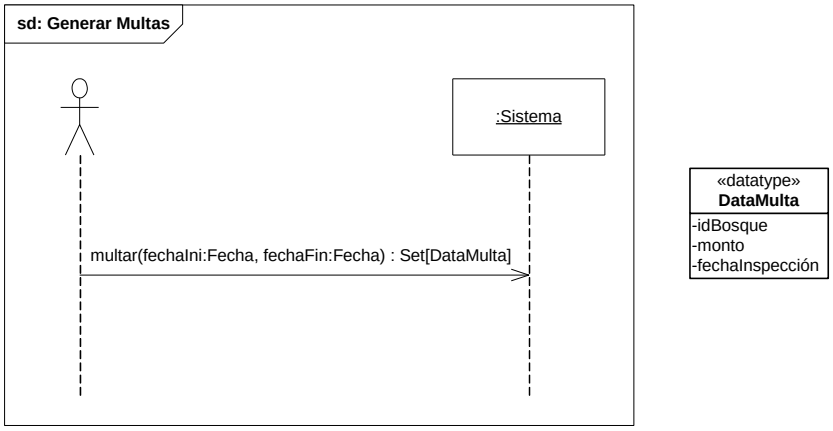
Se ha finalizado la etapa de análisis para el desarrollo de un Sistema que permita registrar y fiscalizar los bosques del territorio nacional. Como muestra el modelo de dominio obtenido, se pueden registrar bosques naturales (son bosques nativos que se deben preservar) y bosques artificiales (son bosques plantados para su tala y posterior comercialización de la madera). El área de un bosque es la suma de las áreas de sus parcelas, con la particularidad que en bosque artificial se deberán restar las áreas del bosque que se talaron. El Sistema permitirá además registrar inspecciones para fiscalizar que los bosques naturales mantienen el área forestada y que los bosques artificiales tienen declarado una tala de árboles correcta (puede ocurrir que no se declare el área a talar para evadir impuestos).



- `idBosque identifica al Bosque y nroParcela identifica a Parcela en el contexto de un Bosque.`

El Sistema permitirá realizar los casos de uso de Registrar Bosque, Inspeccionar, Registrar Tala y Generar Multas. A continuación se muestran los DSS modelados para Registrar Bosque y Generar Multas:





Las operaciones identificadas tienen las siguientes pre y post condiciones:

Operación	ingresoDatos(idBosque:Integer, tipo:TipoBosque)
Pre y post condiciones	
pre: No existe un bosque identificado por idBosque. post: Se recuerdan los parámetros en la memoria del Sistema.	

Operación	ingresarParcela(nroParcela:Integer, area:Real)
Pre y post condiciones	
pre: No existe en la memoria del sistema una instancia de Parcela cuyo atributo nroParcela coincida con el parámetro nroParcela. post: Se crea y recuerda en la memoria del sistema, la instancia de Parcela con atributos nroParcela y area.	

Operación	confirmarIngreso()
Pre y post condiciones	
pre: Existen en la memoria del sistema los parámetros idBosque y tipoBosque, y una colección de instancias de Parcela denominada ps. post: Se crea una instancia de Bosque según tipoBosque (Natural o Artificial) identificada por idBosque. post: Se crea un link entre cada instancia de Parcela en ps y la instancia de Bosque creada.	

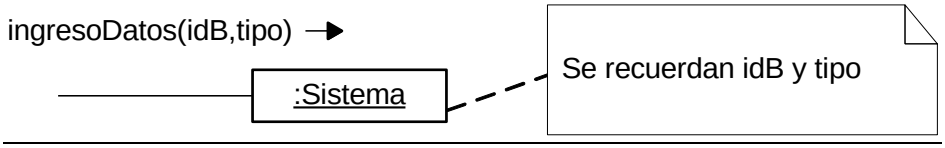
Operación	multar(fechaIni:Fecha, fechaFin:Fecha) : Set[DataMulta]
Pre y post condiciones	
post: Se crea una instancia de multa por cada inspección que tenga fecha dentro del rango especificado por parámetro y que contenga un área relevada inferior al área de un bosque. Se deben tomar aquellas talas con fecha anterior a la inspección. El atributo monto se asigna con un monto preestablecido (dato del Sistema). post: Se crea un link entre cada instancia de multa creada y la instancia de inspección a la que corresponde la multa. post: Se retorna el conjunto de datavalues de DataMulta donde cada datavalue contiene el idBosque del Bosque multado, la fecha de la inspección que ocasiona la multa y el monto de la misma.	

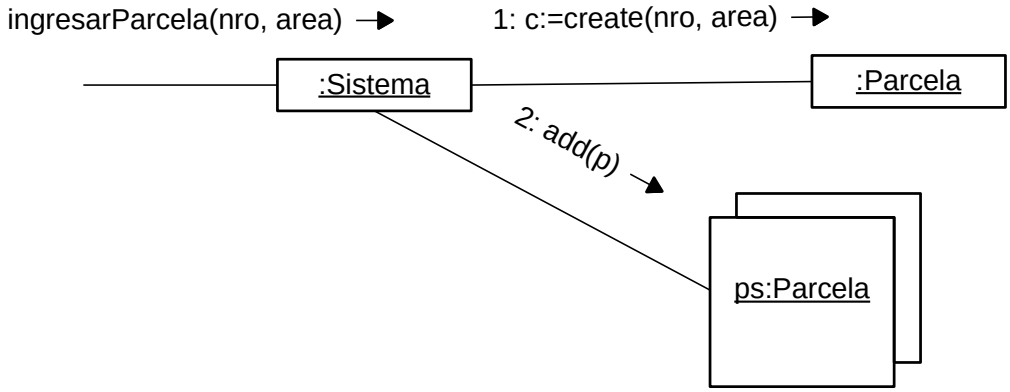
Se pide:

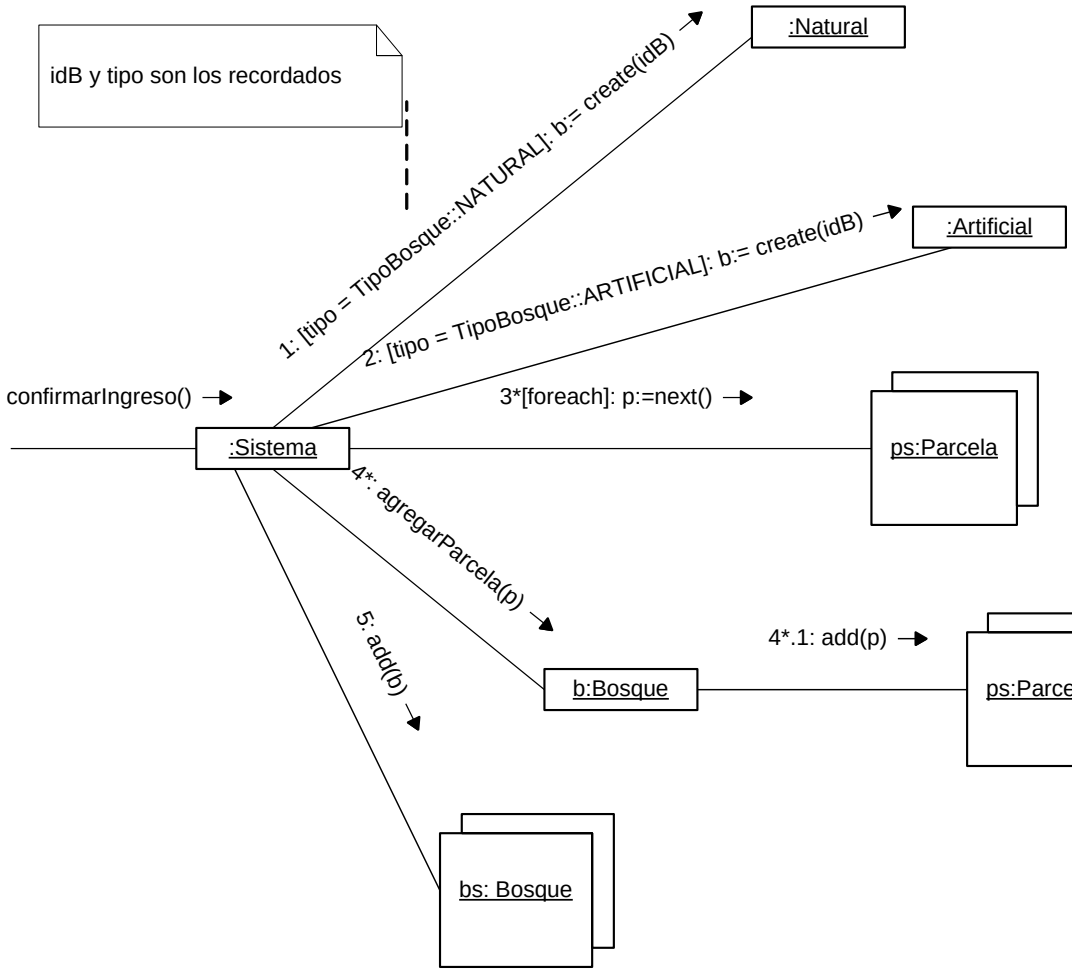
a) Realizar los Diagramas de Comunicación de las cuatro operaciones descritas.
b) Realizar el Diagrama de Clases de Diseño resultante.

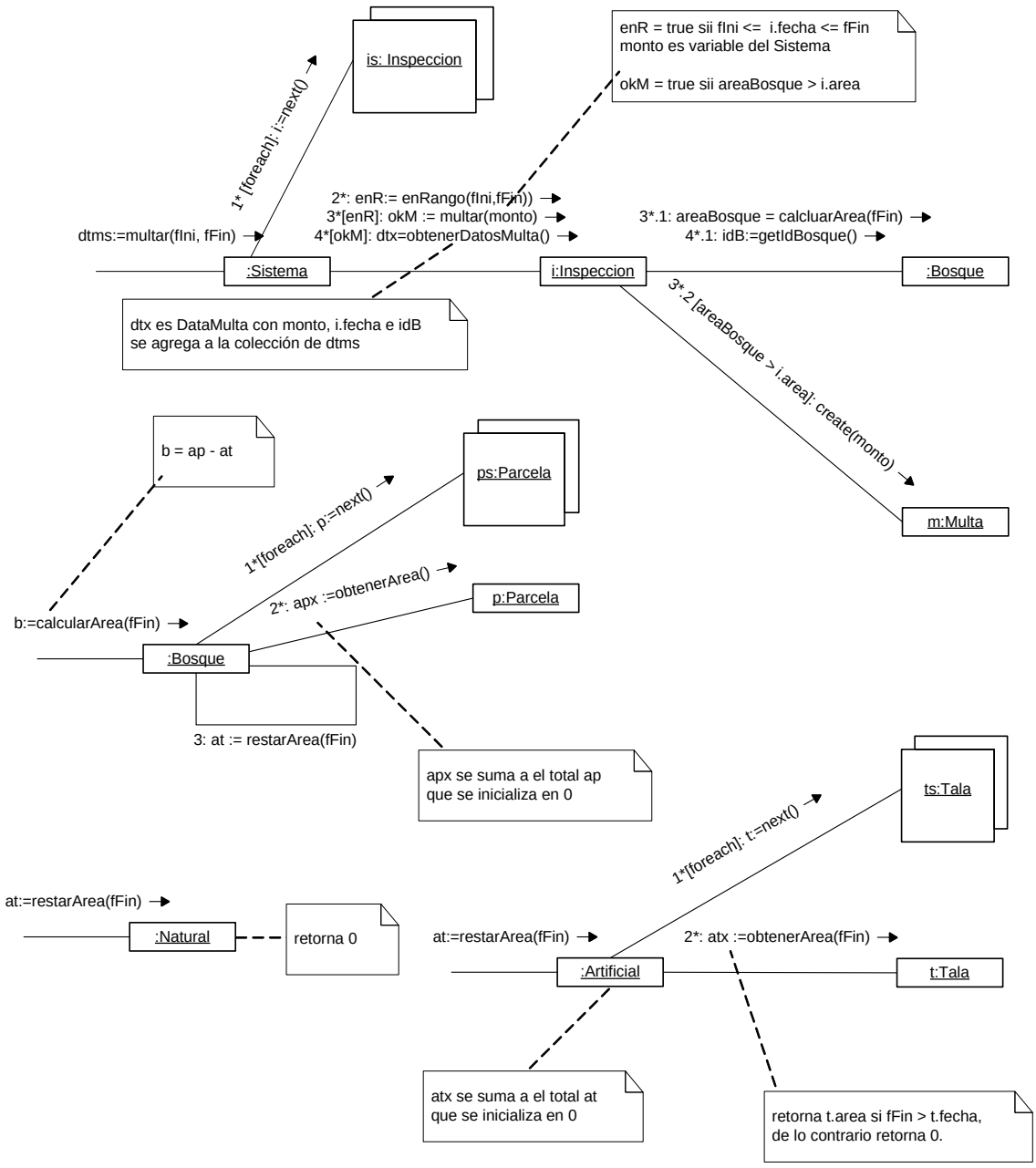
Solución:

a)

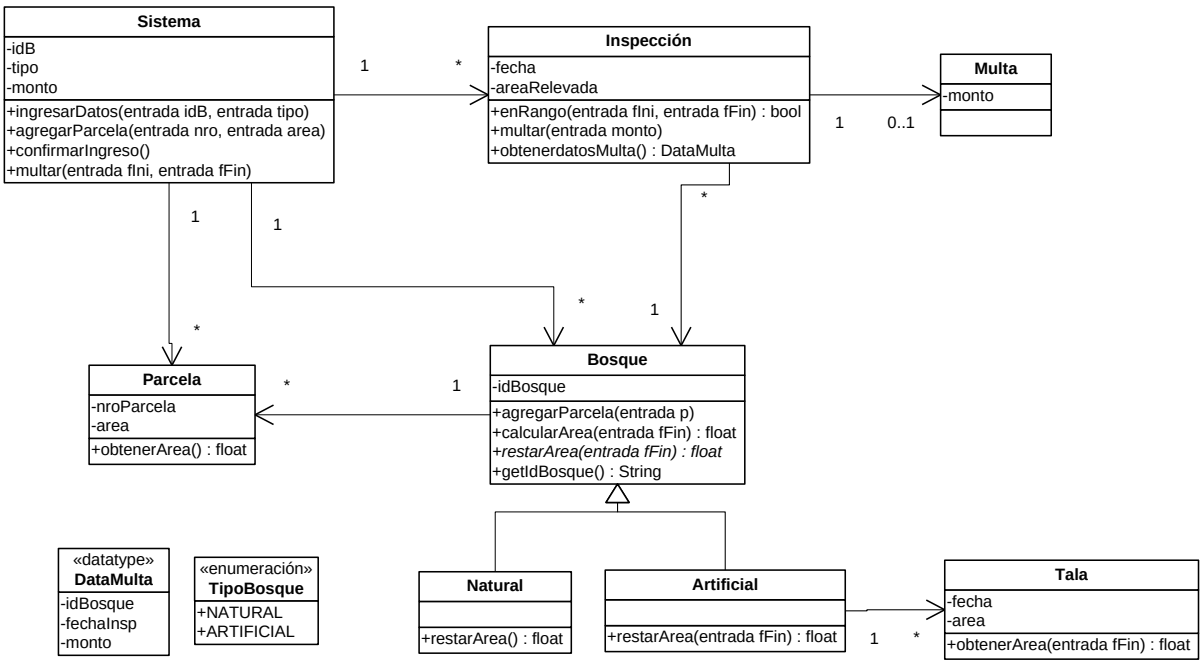

ingresoDatos(idB,tipo)
:Sistema
Se recuerdan idB y tipo


ingresarParcela(nro, area)
1: c:=create(nro, area)
:Sistema
:Parcela
2: add(p)
ps:Parcela


idB y tipo son los recordados
:Natural
1: [tipo = TipoBosque::NATURAL]: b:= create(idB)
:Artificial
2: [tipo = TipoBosque::ARTIFICIAL]: b:= create(idB)
confirmarIngreso()
3*[foreach]: p:=next()
:Sistema
ps:Parcela
4*: agregarParcela(p)
5: add(p)
4*.1: add(p)
b:Bosque
ps:Parcela
bs: Bosque


is: Inspeccion
enR = true sii fIni <= i.fecha <= fFin
monto es variable del Sistema
okM = true sii areaBosque > i.area
1* [foreach]: i:=next()
2*: enR:= enRango(fIni,fFin))
3*[enR]: okM := multar(monto)
4*[okM]: dtx=obtenerDatosMulta()
3*.1: areaBosque = calcluarArea(fFin)
4*.1: idB:=getIdBosque()
dtms:=multar(fIni, fFin)
:Sistema
i:Inspeccion
:Bosque
dtx es DataMulta con monto, i.fecha e idB
se agrega a la colección de dtms
3*.2 [areaBosque > i.area]: create(monto)
m:Multa
b = ap - at
ps:Parcela
1*[foreach]: p:=next()
2*: apx :=obtenerArea()
p:Parcela
b:=calcularArea(fFin)
:Bosque
3: at := restarArea(fFin)
apx se suma a el total ap
que se inicializa en 0
ts:Tala
1*[foreach]: t:=next()
at:=restarArea(fFin)
:Natural
retorna 0
at:=restarArea(fFin)
2*: atx :=obtenerArea(fFin)
:Artificial
t:Tala
atx se suma a el total at
que se inicializa en 0
retorna t.area si fFin > t.fecha,
de lo contrario retorna 0.

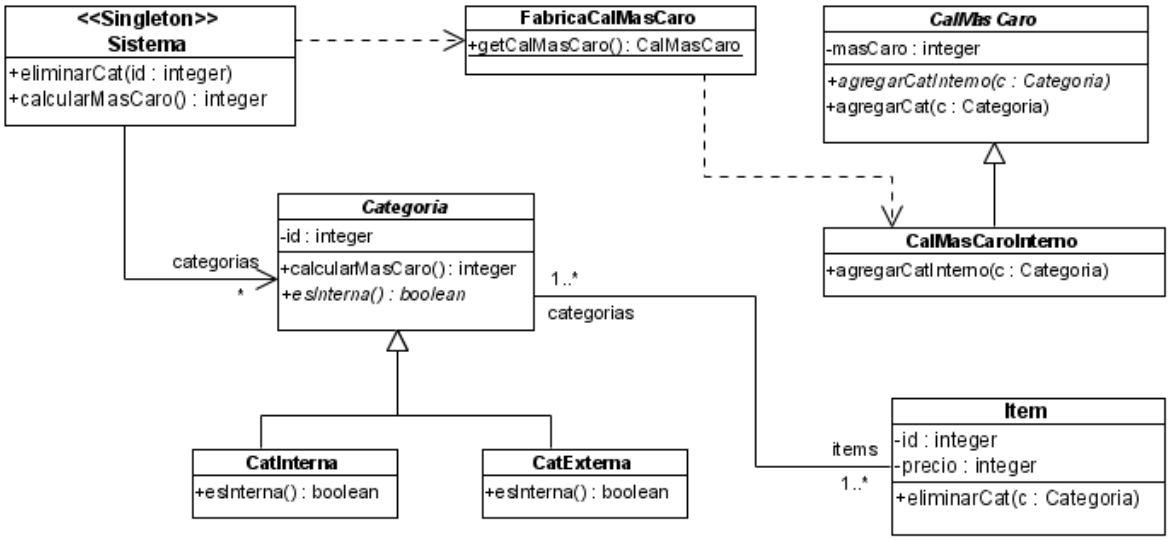
b)


Sistema
-idB
-tipo
-monto
+ingresarDatos(entrada idB, entrada tipo)
+agregarParcela(entrada nro, entrada area)
+confirmarIngreso()
+multar(entrada fIni, entrada fFin)
Inspección
-fecha
-areaRelevada
+enRango(entrada fIni, entrada fFin) : bool
+multar(entrada monto)
+obtenerdatosMulta() : DataMulta
Multa
-monto
Parcela
-nroParcela
-area
+obtenerArea() : float
Bosque
-idBosque
+agregarParcela(entrada p)
+calcularArea(entrada fFin) : float
+restarArea(entrada fFin) : float
+getIdBosque() : String
«datatype»
DataMulta
-idBosque
-fechaInsp
-monto
«enumeración»
TipoBosque
+NATURAL
+ARTIFICIAL
Natural
+restarArea() : float
Artificial
+restarArea(entrada fFin) : float
Tala
-fecha
-area
+obtenerArea(entrada fFin) : float
1
*
1
0..1

Problema 3 (35 puntos)

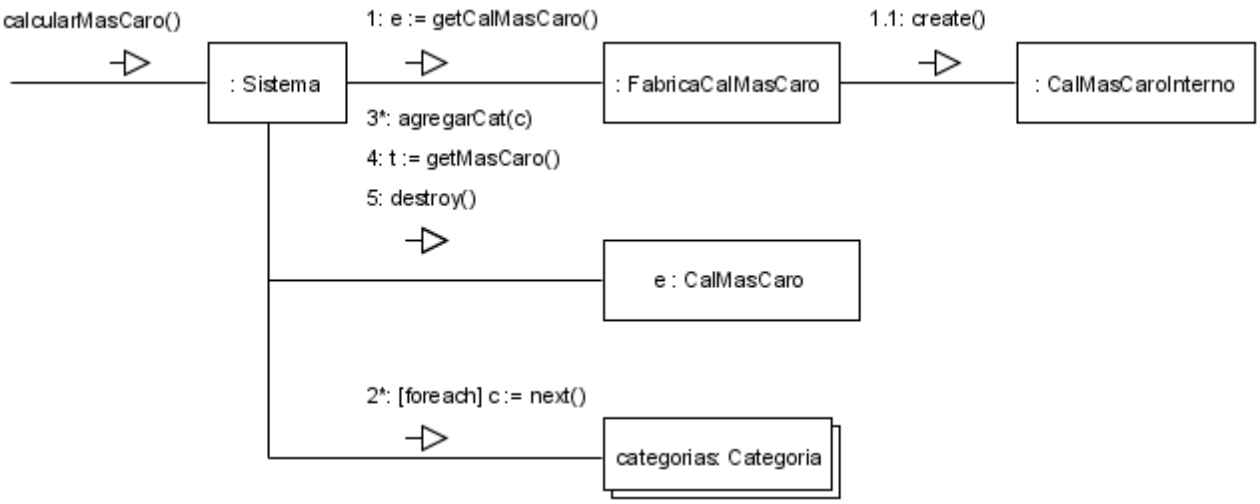
Es de interés la implementación de un sistema de gestión de inventario de un sitio web de *e-commerce* que comercializa ítems tanto del sitio web original como de otros sitios de comercio electrónico externos, cuyo tratamiento dentro de la página web es especial dado que sus datos son extraídos de manera diferente. Los ítems pertenecen a varias categorías, que pueden ser internas (*CatInterna*), que son las que contienen exclusivamente ítems que comercializa directamente el sitio web, o externas (*CatExterna*), que contienen ítems de sitios web externos.

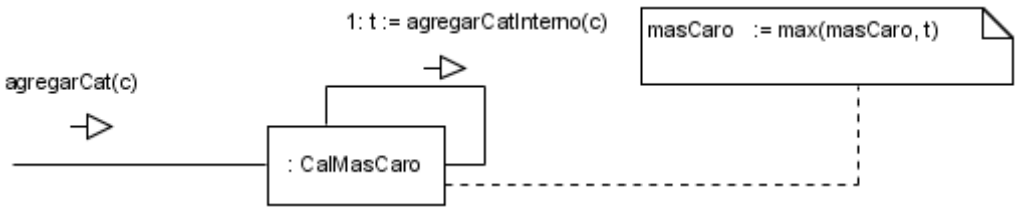
Durante la etapa de diseño se generó el siguiente diagrama de clases de diseño que se deberá respetar durante la implementación. La clase *CalMasCaro* se utiliza para la implementación de la operación *calcularMasCaro()* de la clase *Sistema* y se obtiene utilizando un objeto Fábrica representado por la clase *FabricaCalMasCaro*.

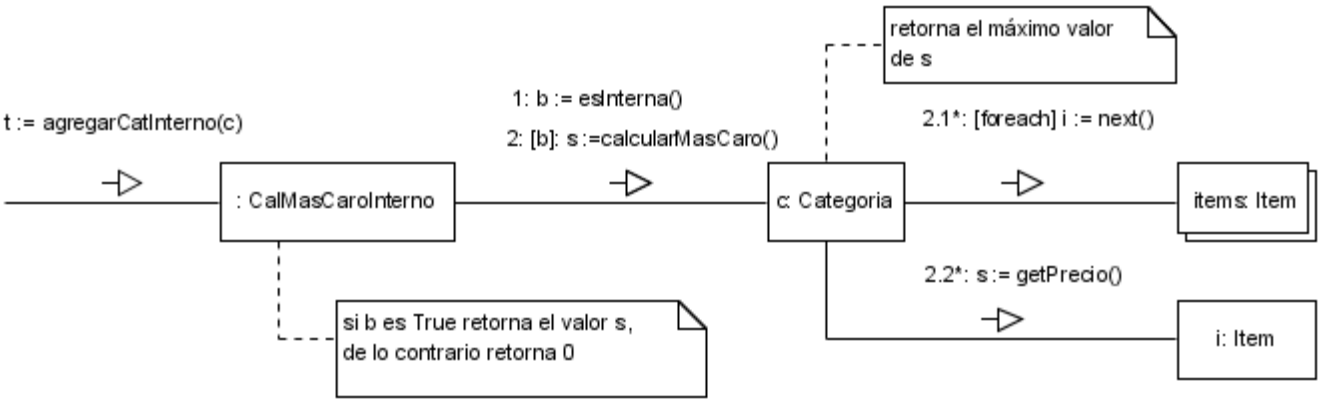


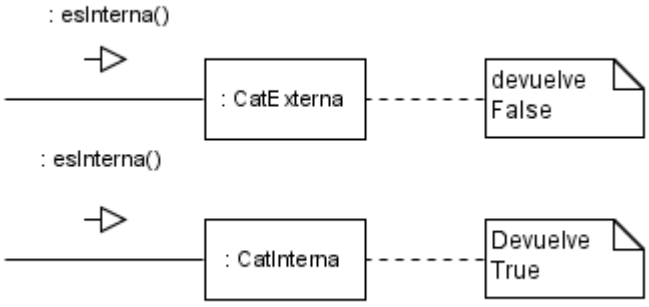
El sistema brinda dos operaciones: *eliminarCat()* se encarga de eliminar una categoría del sistema. Aquellos ítems que solo pertenecían a la categoría eliminada también se destruyen. La segunda operación denominada *calcularMasCaro()* devuelve el precio del ítem más caro del sitio web teniendo en cuenta solamente aquellos que pertenecen a categorías internas. A continuación se dan diagramas de comunicación para ambas operaciones.

calcularMasCaro():

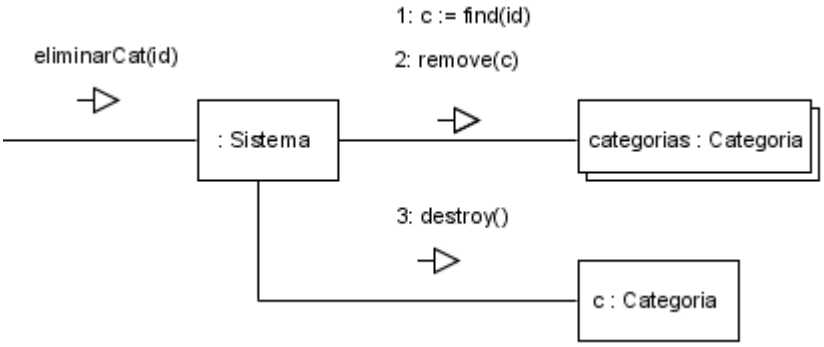


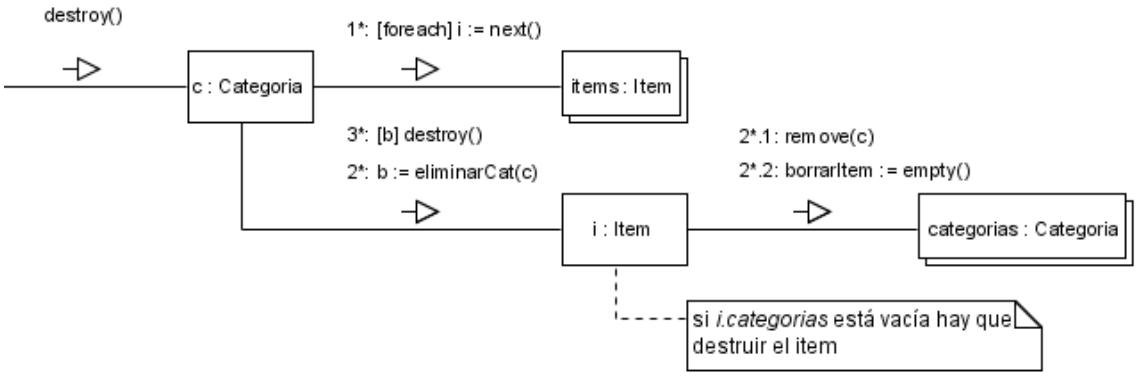






eliminarCat():





Se pide:

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

sistema.h

```
class Sistema
{
private:
	static Sistema *instance;
	Sistema();
	Col *categorias;
public:
	static Sistema *getInstance();
	void eliminarCat(int id);
	int calcularMasCaro();
};
```

sistema.cpp

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

Sistema::Sistema()
{
	categorias = new Col();
}

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

void Sistema::eliminarCat(int id)
{
	KeyInt *k = new KeyInt(id);
	Categoria *c = (Categoria *) (categorias->find(k));
	categorias->remove(c);
	delete c;
	delete k;
}

int Sistema::calcularMasCaro()
{
	CalMasCaro *e = FabricaCalMasCaro::getCalMasCaro();
```

```
	IIterator *it = categorias->getIterator();
	while(it->hasCurrent()){
		e->agregarCat((Categoria *) (it->getCurrent()));
		it->next();
	}
	delete it;

	int res = e->getMasCaro();

	delete e;
	return res;
}
```

ii) Implemente en C++ los .h y .cpp de las clases CalMasCaro y CalMasCaroInterno.

calmascaro.h

```
class CalMasCaro {
private:
	int masCaro;
public:
	CalMasCaro();
	virtual int agregarCatInterno(Categoria *c) = 0;
	void agregarCat(Categoria *c);
	virtual ~CalMasCaro();
};
```

calmascaro.cpp

```
CalMasCaro::CalMasCaro() {
	masCaro = 0;
}

void CalMasCaro::agregarCat(Categoria *c)
{
	int m = agregarCatInterno(c);
	if(masCaro < m)
		masCaro = m;
}

CalMasCaro::~CalMasCaro() {

}
```

calmascarointerno.h

```
class CalMasCaroInterno: public CalMasCaro {
public:
	int agregarCatInterno(Categoria *c);

};
```

calmascarointerno.cpp

```
int CalMasCaroInterno::agregarCatInterno(Categoria *c)
{
	if(c->esInterna())
		return 0;
	else
		return c->calcularMasCaro();
```

```
}
```

iii) Implemente en C++ los .h y .cpp de las clases Categoria y CatInterna.

categoria.h

```
class Categoria: public ICollectible {
private:
	int id;
	Col *items;
protected:
	Categoria(int id);
public:
	int calcularMasCaro();
	virtual bool esInterna() = 0;
	virtual ~Categoria();
};
```

categoria.cpp

```
Categoria::Categoria(int id)
{
	this->id = id;
	items = new Col();
}

Categoria::~Categoria() {
	IIterator *it = items->getIterator();
	while(it->hasCurrent()){
		Item *i = (Item *) (it->getCurrent());
		if(i->eliminarCat(this) == true)
			delete i;
		it->next();
	}
	delete it;
	delete items;
}

int Categoria::calcularMasCaro()
{
	IIterator *it = items->getIterator();
	int res = 0;
	while(it->hasCurrent()){
		int p = ((Item *) (it->getCurrent()))->getPrecio();
		if(p > res)
			res = p;
	}
	delete it;
	return res;
}
```

catinterna.h

```
class CatInterna: public Categoria {
public:
	CatInterna(int id);
	bool esInterna();
	virtual ~CatInterna();
};
```

catinterna.cpp

```
CatInterna::CatInterna(int id):Categoria(id)
{
}

CatInterna::~CatInterna() {
}

bool CatInterna::esInterna()
{
	return true;
}
```