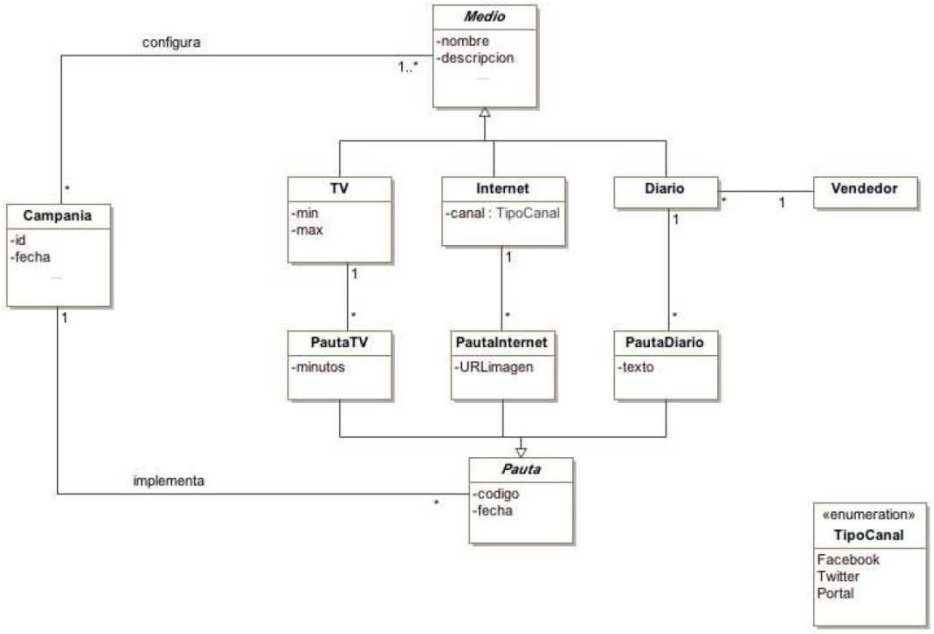
Programación Avanzada

SOLUCIÓN EXAMEN FEBRERO 2015

Problema 1

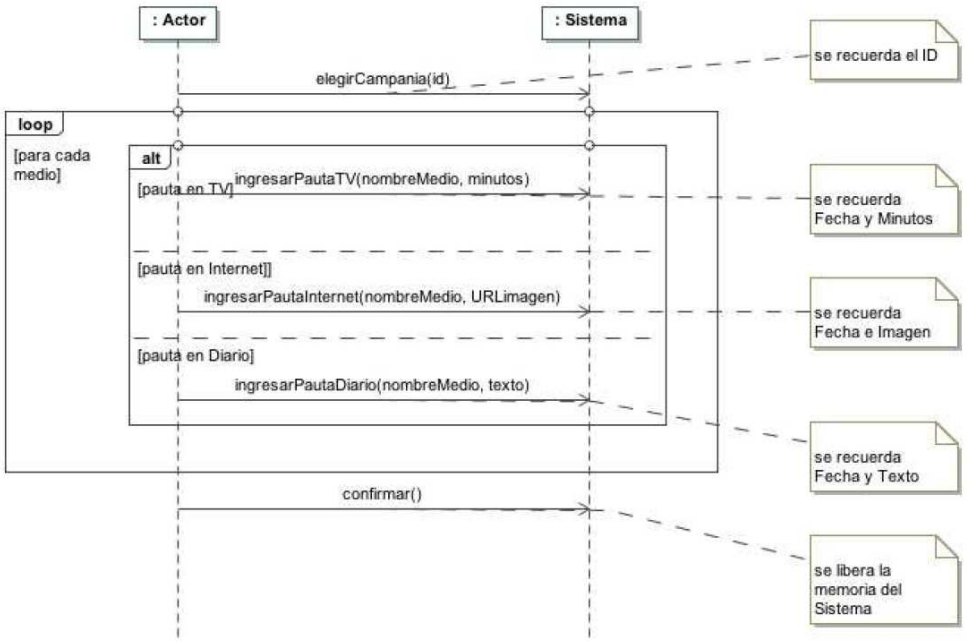
i)



Restricciones:

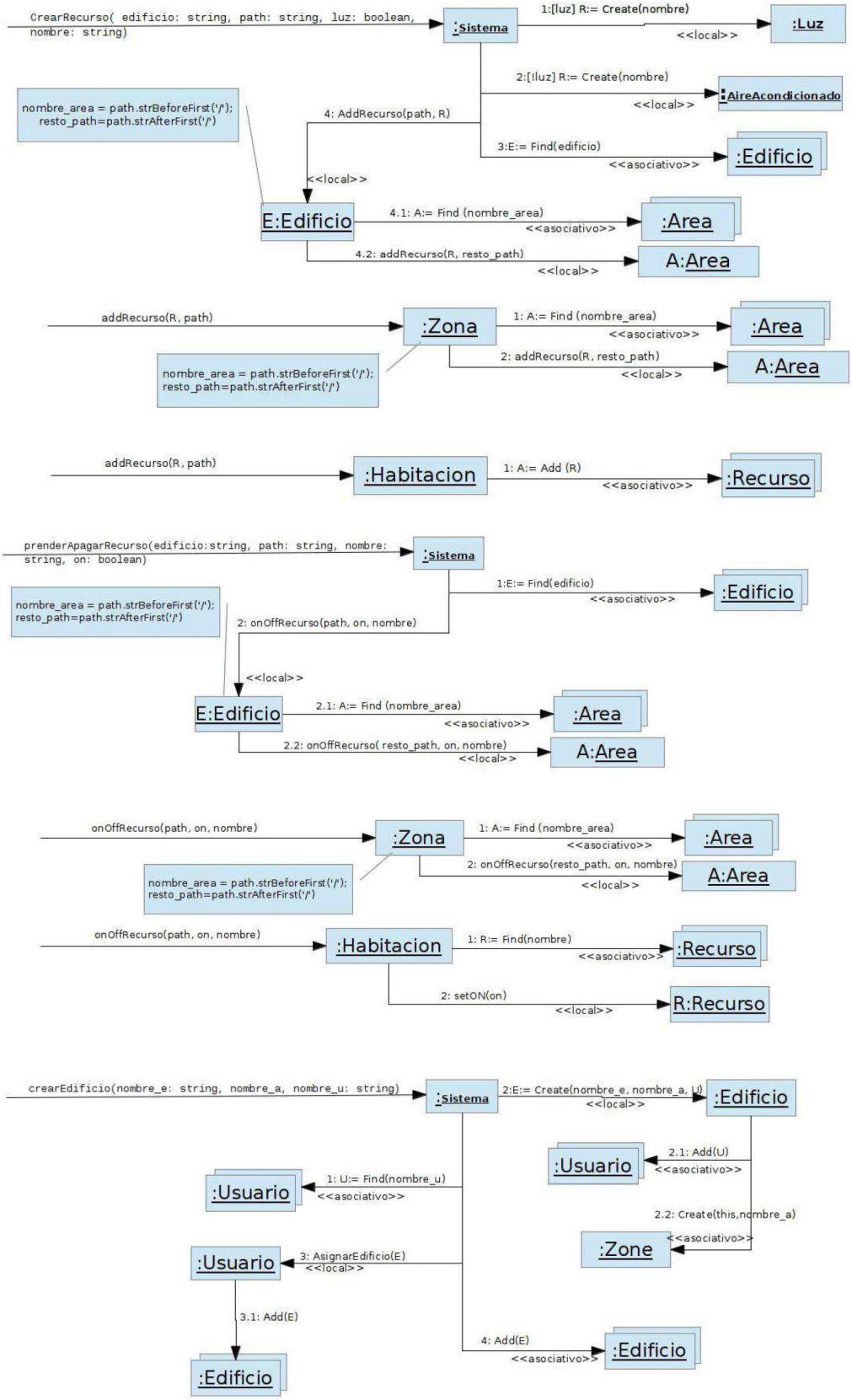
- El ID de la campaña la identifica.
- El nombre del medio lo identifica.
- El código de la pauta la identifica.
- Los minutos de una PautaTV estarán entre el mínimo y máximo definido en su medio correspondiente.
- El medio referenciado por una pauta debe haber sido defindo/configurado en la misma campaña a la que la pauta pertenece.

ii)



Problema 2

i)


CrearRecurso(edificio: string, path: string, luz: boolean, nombre: string)
:Sistema
1:[luz] R:= Create(nombre)
<<local>>
:Luz
2:[!luz] R:= Create(nombre)
<<local>>
:AireAcondicionado
nombre_area = path.strBeforeFirst('/'); resto_path=path.strAfterFirst('/')
4: AddRecurso(path, R)
3:E:= Find(edificio)
<<asociativo>>
:Edificio
<<local>>
E:Edificio
4.1: A:= Find (nombre_area)
<<asociativo>>
:Area
4.2: addRecurso(R, resto_path)
<<local>>
A:Area
addRecurso(R, path)
:Zona
1: A:= Find (nombre_area)
<<asociativo>>
:Area
nombre_area = path.strBeforeFirst('/'); resto_path=path.strAfterFirst('/')
2: addRecurso(R, resto_path)
<<local>>
A:Area
addRecurso(R, path)
:Habitacion
1: A:= Add (R)
<<asociativo>>
:Recurso
prenderApagarRecurso(edificio:string, path: string, nombre: string, on: boolean)
:Sistema
1:E:= Find(edificio)
<<asociativo>>
:Edificio
nombre_area = path.strBeforeFirst('/'); resto_path=path.strAfterFirst('/')
2: onOffRecurso(path, on, nombre)
<<local>>
E:Edificio
2.1: A:= Find (nombre_area)
<<asociativo>>
:Area
2.2: onOffRecurso(resto_path, on, nombre)
<<local>>
A:Area
onOffRecurso(path, on, nombre)
:Zona
1: A:= Find (nombre_area)
<<asociativo>>
:Area
nombre_area = path.strBeforeFirst('/'); resto_path=path.strAfterFirst('/')
2: onOffRecurso(resto_path, on, nombre)
<<local>>
A:Area
onOffRecurso(path, on, nombre)
:Habitacion
1: R:= Find(nombre)
<<asociativo>>
:Recurso
2: setON(on)
<<local>>
R:Recurso
crearEdificio(nombre_e: string, nombre_a, nombre_u: string)
:Sistema
2:E:= Create(nombre_e, nombre_a, u)
<<local>>
:Edificio
:Usuario
2.1: Add(U)
<<asociativo>>
1: U:= Find(nombre_u)
<<asociativo>>
:Usuario
2.2: Create(this,nombre_a)
<<asociativo>>
:Zone
:Usuario
3: AsignarEdificio(E)
<<local>>
3.1: Add(E)
4: Add(E)
<<asociativo>>
:Edificio
:Edificio

Problema 3

i)

```
class A {

private:
	int dato;
	static A *instance;
	A();

public:
	static A *getInstance();
	int getDato();
	void setDato(int);
}

A *A::instance = NULL;

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

int A::getDato() {
	return dato;
}

void A::setDato(int d) {
	dato = d;
}
```

ii)

```
class Paquete : public ICollectible {

public:
	virtual double calcularPeso() = 0;
	virtual double calcularVolumen() = 0;
	virtual ~Paquete();
}

Paquete::~Paquete() {
}

class Sencillo : public Paquete {

private:
	 double peso, volumen;

public:
	Sencillo(double, double);
	double getPeso();
	void setPeso(double);
	double getVolumen();
	void setVolumen(double);
	double calcularPeso();
	double calcularVolumen();
}

Sencillo::Sencillo(double p, double v) {
	peso = p;
	volumen = v;
}

double Sencillo::getPeso() {
	return peso;
}

void Sencillo::setPeso(double p) {
	peso = p;
```

```
}

double Sencillo::getVolumen() {
	return volumen;
}

void Sencillo::setVolumen(double v) {
	volumen = v;
}

double Sencillo::calcularPeso() {
	return peso;
}

double Sencillo::calcularVolumen() {
	return volumen;
}

class Complejo : public Paquete

private:
	ICollection *componentes;
	OptimizadorVolumen *optimizador;



public:
	Complejo(ICollection *, OptimizadorVolumen *);
	~Complejo();
	double calcularPeso();
	double calcularVolumen();
	void setOptVol(OptimizadorVolumen *);
}

Complejo::Complejo(ICollection *comps, OptimizadorVolumen *opt) {
	componentes = new List;
	IIterator *it = comps->getIterator();
	while (it->hasCurrent()) {
		componentes->add(it->getCurrent());
		it->next();
	}
	delete it;
	optimizador = opt;
}

Complejo::~Complejo() {
	IIterator *it = componentes->getIterator();
	ICollectible *elem;
	while (it->hasCurrent()) {
		elem = it->getCurrent()
		it->next();
		componentes->remove(elem);
		delete elem;
	}
	delete it;
	delete componentes;
}

double Complejo::calcularPeso() {
	IIterator *it = componentes->getIterator();
	double result = 0;
	while (it->hasCurrent()) {
		result = result + ((Paquete *)it->getCurrent())->getPeso();
		it->next();
	}
	delete it;
	return result;
}

double Complejo::calcularVolumen() {
	return opt->volOptimo(componentes);
}
```

```
void Complejo::setOptVol(OptimizadorVolumen *opt) {
	optimizador = opt;
}

class OptimizadorVolumen {

public:
	virtual double valOptimo(ICollection *) = 0;
	virtual ~OptimizadorVolumen();
}

OptimizadorVolumen::~OptimizadorVolumen() {
}
```