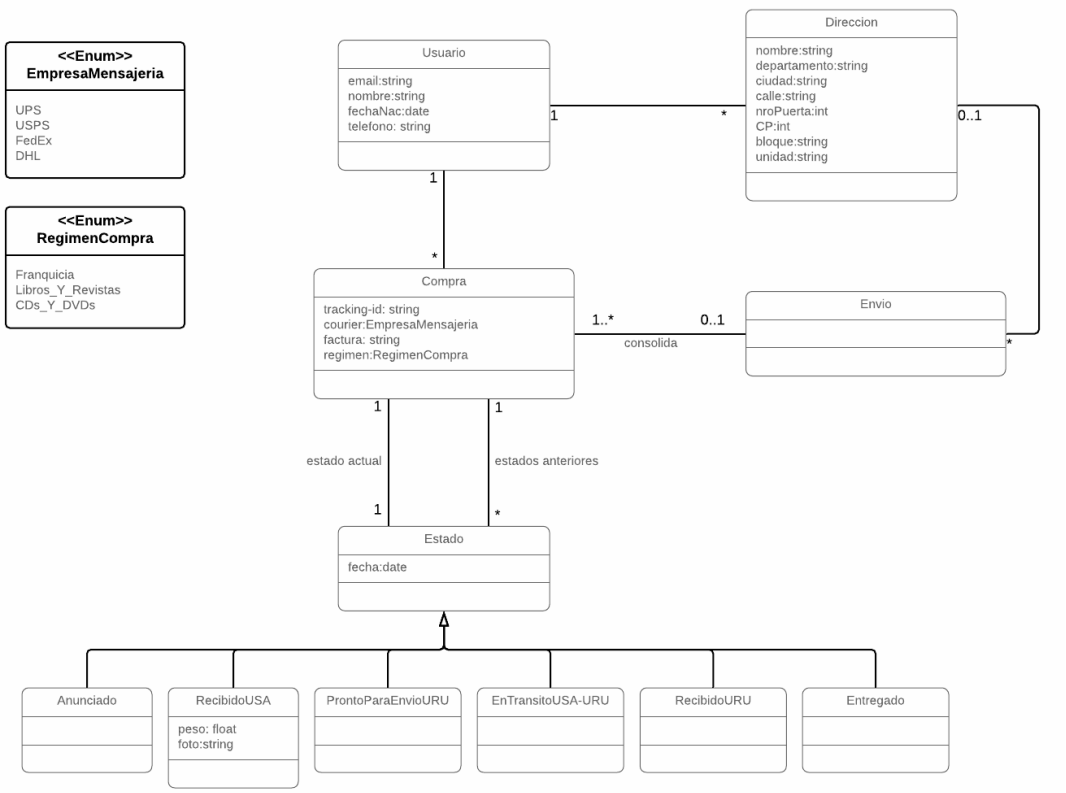
Programación 4

SOLUCIÓN EXAMEN DICIEMBRE 2019

Problema 1:

a)



Restricciones:

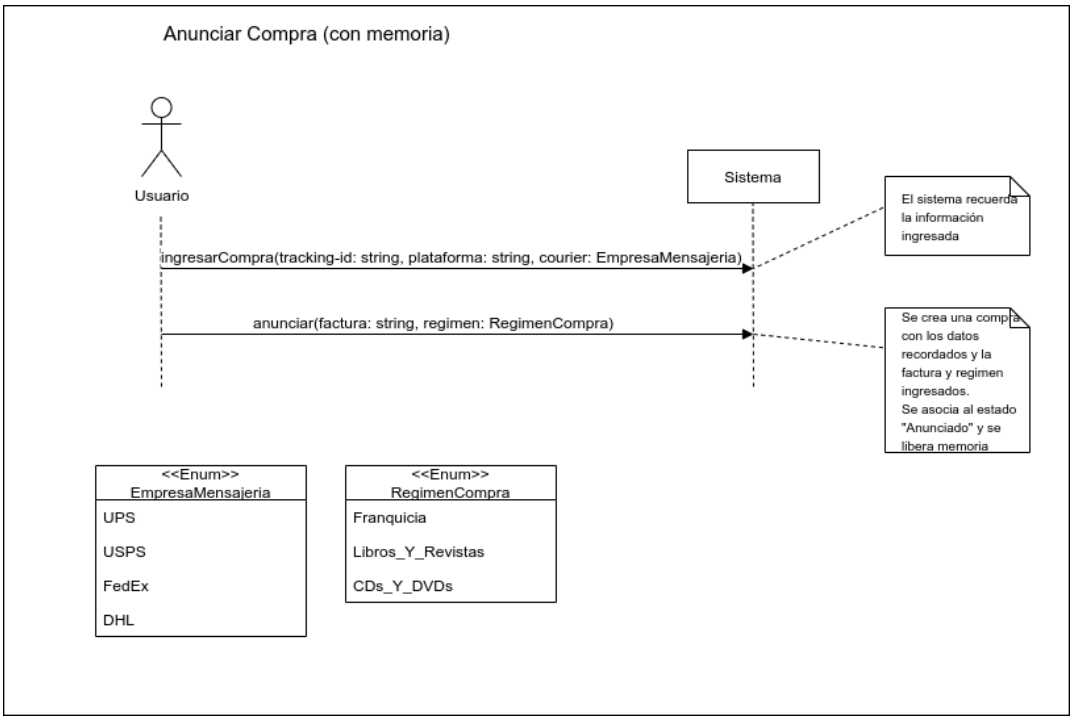
1) Unicidad
- No hay dos usuarios con igual email
- No hay dos compras con igual tracking-id
- Para un mismo usuario, no hay dos direcciones con igual nombre

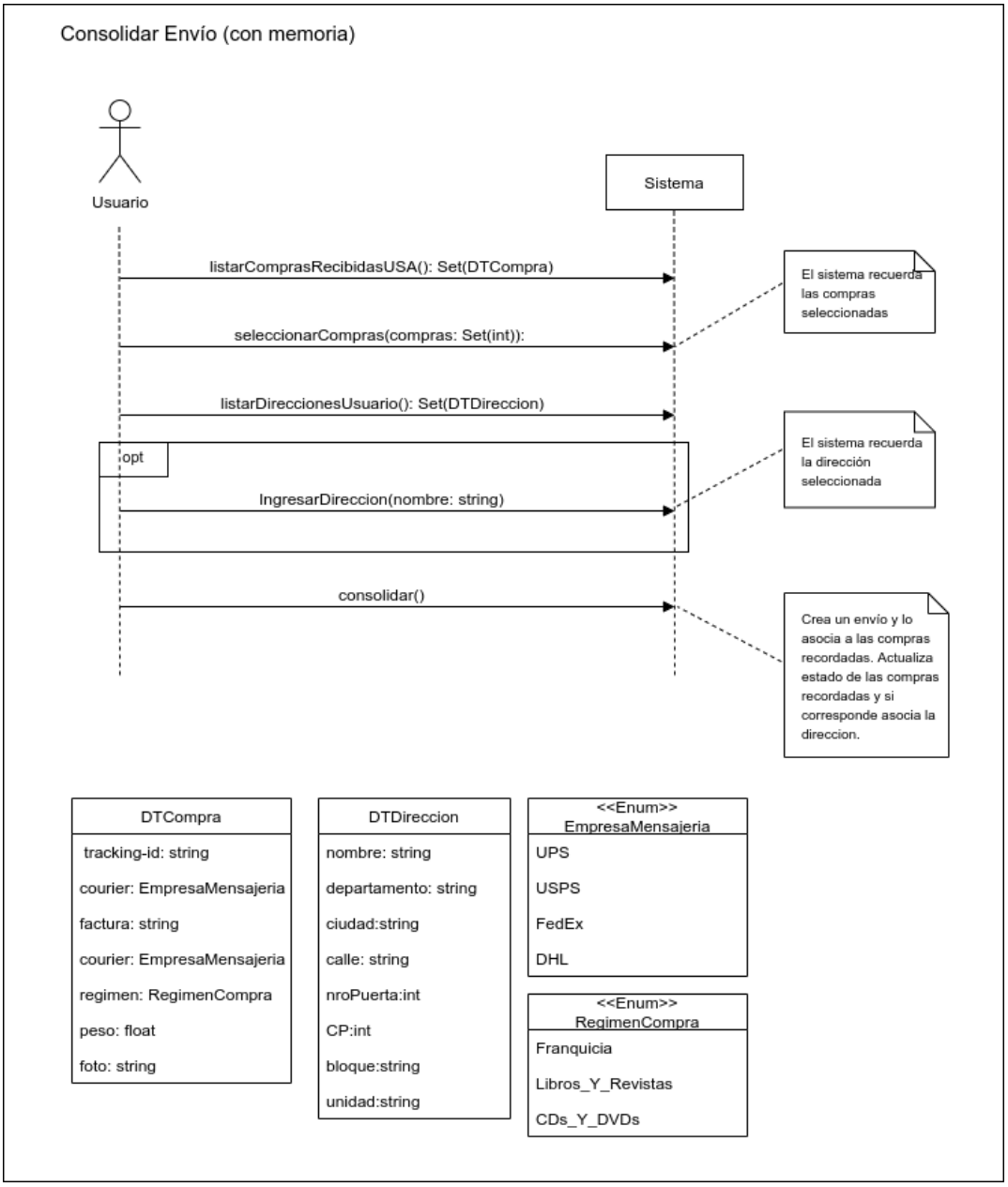
2) Dominio de Atributos
- El peso de una copra es estricto positivo

3) Restriccionees circulares
- Dado un Usuario, cualquier compra si es Enviada a una Direccion, debe ser una Dirección del mismo Usuario.

4) Reglas de Negocio
- Toda compra se crea con el estado Anunciado

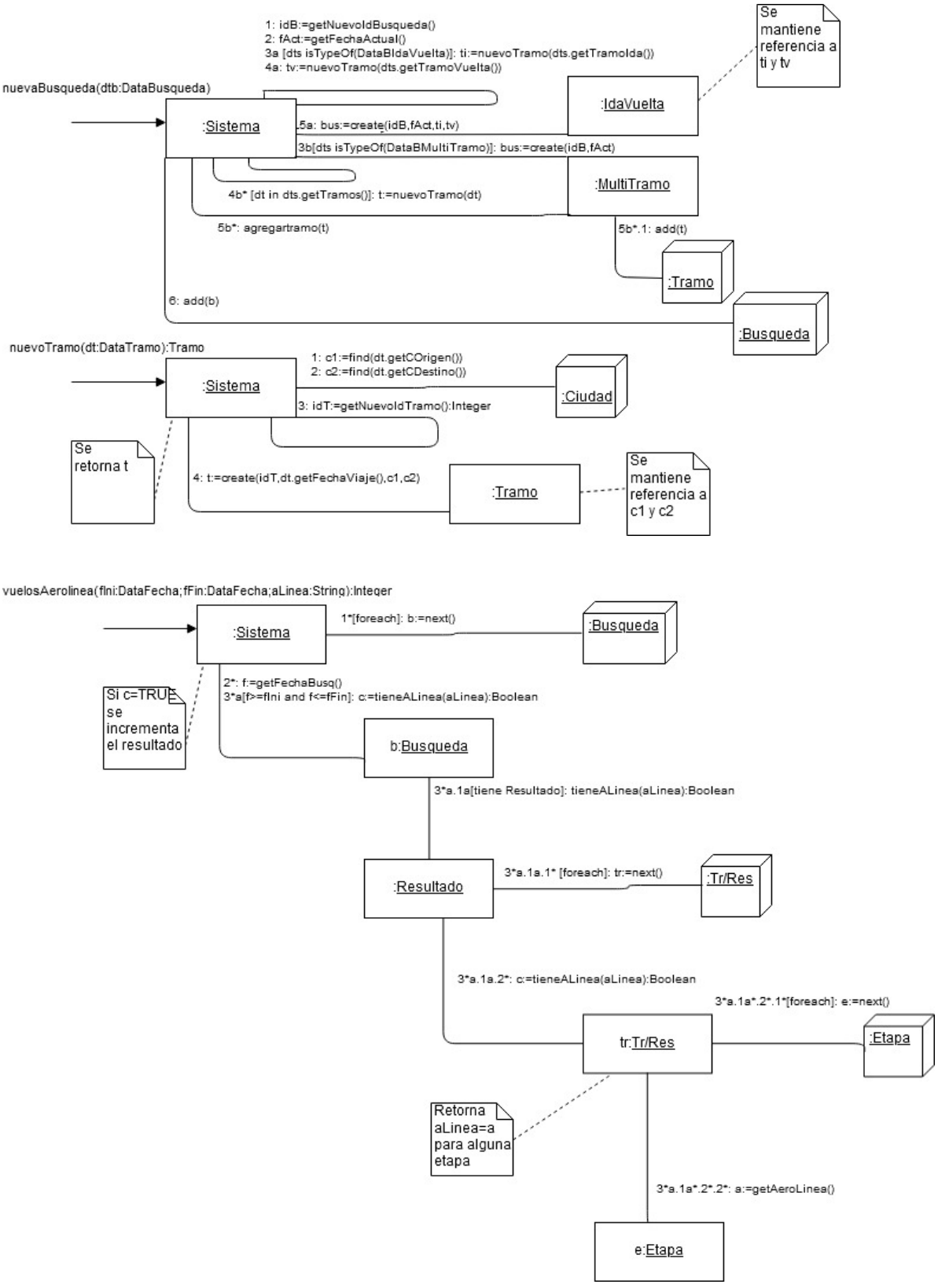
b)


Anunciar Compra (con memoria)
Usuario
Sistema
El sistema recuerda la información ingresada
ingresarCompra(tracking-id: string, plataforma: string, courier: EmpresaMensajeria)
anunciar(factura: string, regimen: RegimenCompra)
Se crea una compra con los datos recordados y la factura y regimen ingresados.
Se asocia al estado "Anunciado" y se libera memoria
<<Enum>>
EmpresaMensajeria
UPS
USPS
FedEx
DHL
<<Enum>>
RegimenCompra
Franquicia
Libros_Y_Revistas
CDs_Y_DVDs

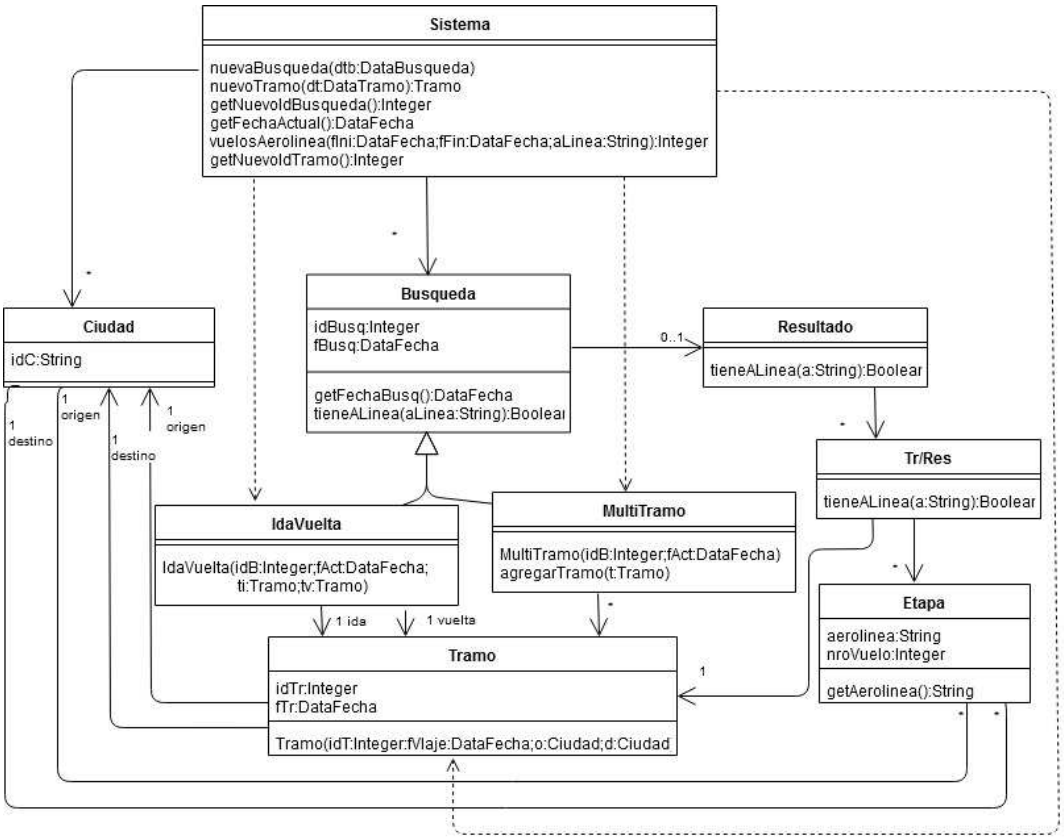

Consolidar Envío (con memoria)
Usuario
Sistema
listarComprasRecibidasUSA(): Set(DTCompra)
El sistema recuerda las compras seleccionadas
seleccionarCompras(compras: Set(int)):
listarDireccionesUsuario(): Set(DTDireccion)
opt
El sistema recuerda la dirección seleccionada
IngresarDireccion(nombre: string)
consolidar()
Crea un envío y lo asocia a las compras recordadas. Actualiza estado de las compras recordadas y si corresponde asocia la direccion.
DTCompra
tracking-id: string
courier: EmpresaMensajeria
factura: string
courier: EmpresaMensajeria
regimen: RegimenCompra
peso: float
foto: string
DTDireccion
nombre: string
departamento: string
ciudad:string
calle: string
nroPuerta:int
CP:int
bloque:string
unidad:string
<<Enum>>
EmpresaMensajeria
UPS
USPS
FedEx
DHL
<<Enum>>
RegimenCompra
Franquicia
Libros_Y_Revistas
CDs_Y_DVDs

Problema 2

a)


1: idB:=getNuevoIdBusqueda()
2: fAct:=getFechaActual()
3a [dts isTypeOf(DataBIdaVuelta)]: ti:=nuevoTramo(dts.getTramoIda())
4a: tv:=nuevoTramo(dts.getTramoVuelta())
Se mantiene referencia a ti y tv
nuevaBusqueda(dtb:DataBusqueda)
:Sistema
:IdaVuelta
5a: bus:=create(idB,fAct,ti,tv)
3b[dts isTypeOf(DataBMultiTramo)]: bus:=create(idB,fAct)
:MultiTramo
4b* [dt in dts.getTramos()]: t:=nuevoTramo(dt)
5b*: agregartramo(t)
5b*.1: add(t)
:Tramo
6: add(b)
:Busqueda
nuevoTramo(dt:DataTramo):Tramo
:Sistema
1: c1:=find(dt.getCOrigen())
2: c2:=find(dt.getCDestino())
:Ciudad
3: idT:=getNuevoIdTramo():Integer
Se retorna t
4: t:=create(idT,dt.getFechaViaje(),c1,c2)
:Tramo
Se mantiene referencia a c1 y c2
vuelosAerolinea(fIni:DataFecha;fFin:DataFecha;aLinea:String):Integer
:Sistema
1*[foreach]: b:=next()
:Busqueda
Si c=TRUE se incrementa el resultado
2*: f:=getFechaBusq()
3*a[f>=fIni and f<=fFin]: c:=tieneALinea(aLinea):Boolean
b:Busqueda
3*a.1a[tiene Resultado]: tieneALinea(aLinea):Boolean
:Resultado
3*a.1a.1* [foreach]: tr:=next()
:Tr/Res
3*a.1a.2*: c:=tieneALinea(aLinea):Boolean
3*a.1a*.2*.1*[foreach]: e:=next()
tr:Tr/Res
:Etapa
Retorna aLinea=a para alguna etapa
3*a.1a*.2*.2*: a:=getAeroLinea()
e:Etapa

b)



Problema 3:

IObserver.hpp

```
class IObserver {
	public:
		virtual void notificar(float lat, float lon) = 0;
}
```

FabricaAerea.hpp

```
class FabricaAerea: {
	private:
		static FabricaAerea* instancia;
		FabricaAerea();
	public:
		~FabricaAerea();
		IControl* getIControl();
		static FabricaAerea* getInstance();
}
```

FabricaAerea.cpp

```
FabricaAerea* FabricaAerea::instancia = NULL;
FabricaAerea::FabricaAerea(){}
FabricaAerea::~FabricaAerea(){}
FabricaAerea* FabricaAerea::getInstance() {
	if(instancia == NULL)
		instancia = new FabricaAerea();
```

```
	return instancia;
}
Icontrol* FabricaAerea::getIControl() {
	return new ControladorAereo();
}
```

ControladorAereo.hpp

```
class ControladorAereo: public IControl {
	private
		map<string, Cohete*> cohetes;
	public:
		ControladorAereo();
		~ControladorAereo();
		void agregarCohete(Cohete* coh);
		void lanzarCohete(string id, float lat, float lon);
		void destruirCohete(string id);
}
```

ControladorAereo.cpp

```
ControladorAereo::ControladorAereo(){}
ControladorAereo::~ControladorAereo(){
	for(map<string, Cohete*>::iterator it = cohetes.begin(); it!
	=cohetes.end(); ++it) {
		delete it->second;
	}
}
void ControladorAereo::agregarCohete(Cohete* coh) {
	cohetes[coh->getId()] = coh;
}
void ControladorAereo::lanzarCohete(string id, float lat, float lon) {
	Cohete *c = cohetes[id];
	if (c != NULL)
		c->lanzarCohete(lat, lon);
}

void ControladorAereo::destruirCohete(string id) {
	Cohete* c = cohetes[id];
	if (c != NULL){
		cohetes.erase(id);
		delete c;
	}
}
```

Cohete.hpp

```
class Cohete {
	private:
		string id;
		string fechaCreacion;
	public:
		Cohete(string id, string fechaCreacion);
		virtual ~Cohete();
		virtual void lanzar(float lat, float lon) = 0;
		void encenderMotor();
}
```

Cohete.cpp

```
Cohete::Cohete(int id, string fechaCreacion) {
```

```
	this->id = id;
	this->fechaCreacion = fechaCreacion;
}
Cohete::~Cohete(){}
void Cohete::encenderMotor() {
	//Implementado
}
```

Atomico.hpp

```
class Atomico: public Cohete {
	private:
		int cargaAtomica;
		void alertaEmergencia(float lat, float lon);
		set<IObserver*> observadores;
	public:
		Atomico( string id, string fechaCreacion, int
cargaAtomica);
		~Atomico();
		void lanzar(float lat, float lon);
		void agregarObservador(IObserver* obs);
		void removerObservador(IObserver* obs);
}
```

Atomico.cpp

```
Atomico::Atomico(int id, string fechaCreacion, int cargaAtomica):
Cohete(id,fechaCreacion) {
	this->cargaAtomica = cargaAtomica;
}
Atomico::~Atomico(){}
void Atomico::alertaEmergencia(float lat, float lon){
	for(set<IObserver*>::iterator
iter=observadores.begin();iter!=observadores.end();++iter){
		(*iter)->notificar(lat, lon);
	}
}
void Atomico::agregarObservador(IObserver* obs){
	observadores.insert(obs);
}
void Atomico::removerObservador(IObserver* obs){
	observadores.erase(obs);
}
void Atomico::lanzar(float lat, float lon){
	encenderMotor();
	alertaEmergencia(lat,lon);
}
```

Gobierno.hpp

```
class Gobierno: public IObserver {
	private:
		void alarmaAstronomica(float lat, float lon);
	public:
		void notificar(float lat, float lon);
		Gobierno();
		~Gobierno();
}
```

Gobierno.cpp

```
Gobierno::Gobierno() {}
Gobierno::~Gobierno(){}
void Gobierno::alarmaAstronomica(float lat, float lon){
      // Implementado
}
void Gobierno::notificar(float lat, float lon){
      alarmaAstronomica(lat,lon);
}
```