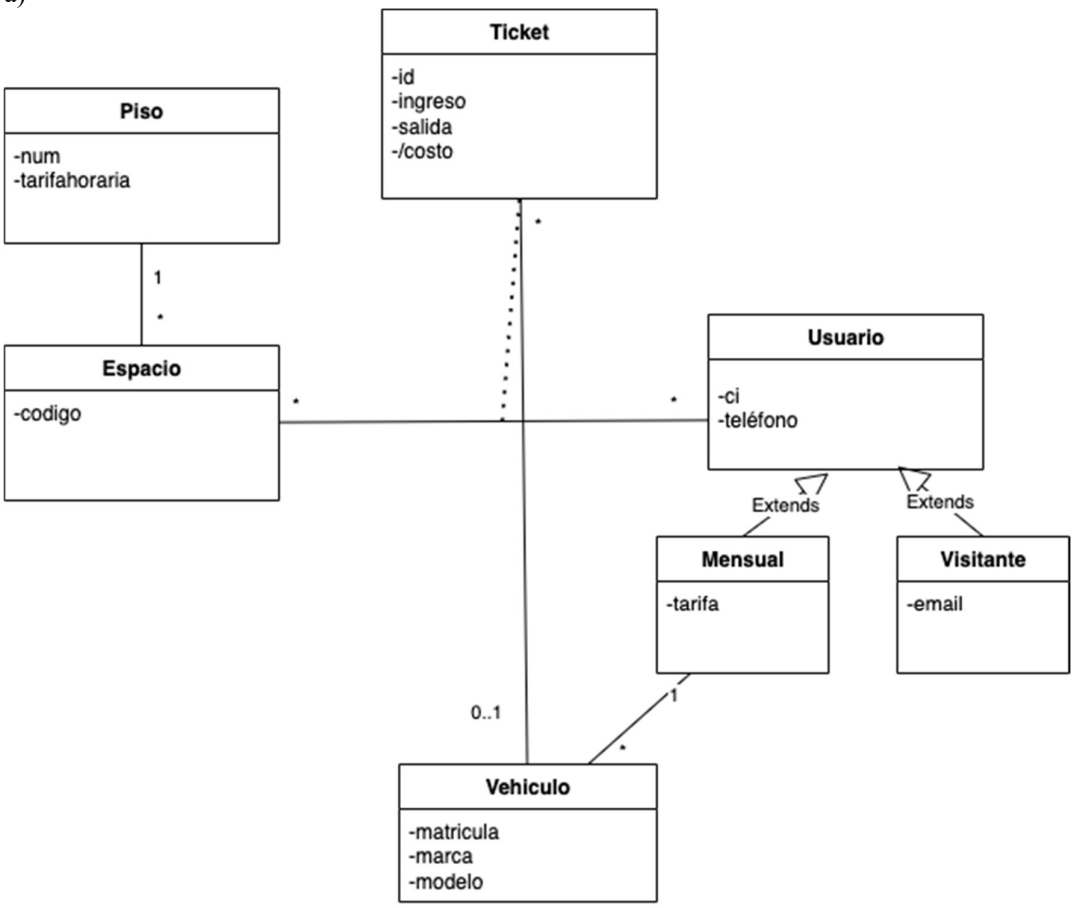
Programación 4

SOLUCIÓN EXAMEN FEBRERO 2022

Problema 1

a)



Restricciones

Identidad

- id identifica al ticket
- ci identifica al usuario
- codigo identifica a un espacio en un piso

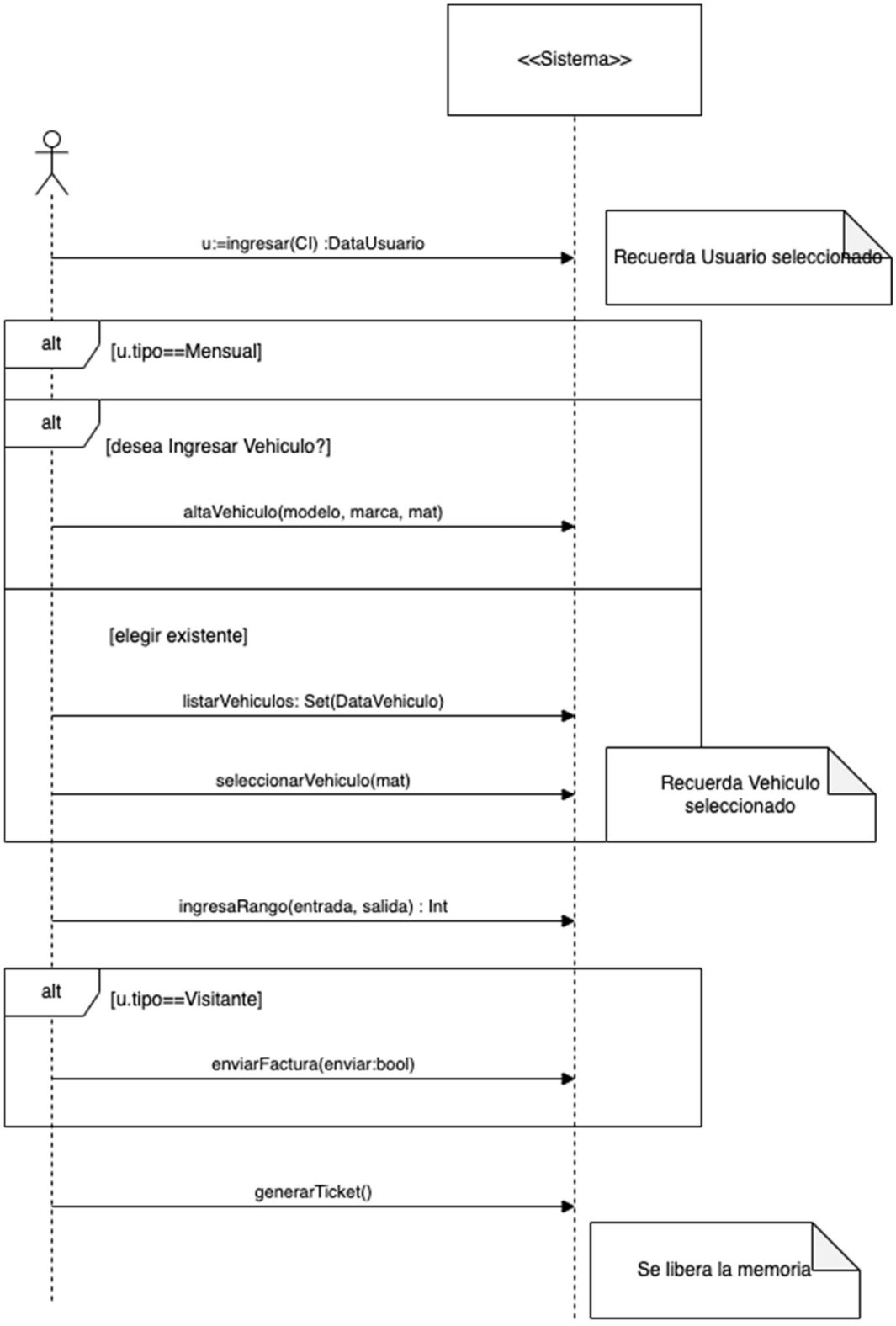
Integridad circular

- Un vehículo asociado a un ticket debe pertenecer al mismo usuario mensual asociado al ticket.

Reglas de negocio

- No existen dos tickets para el mismo espacio que superponen su rango de ingreso y salida.

b)

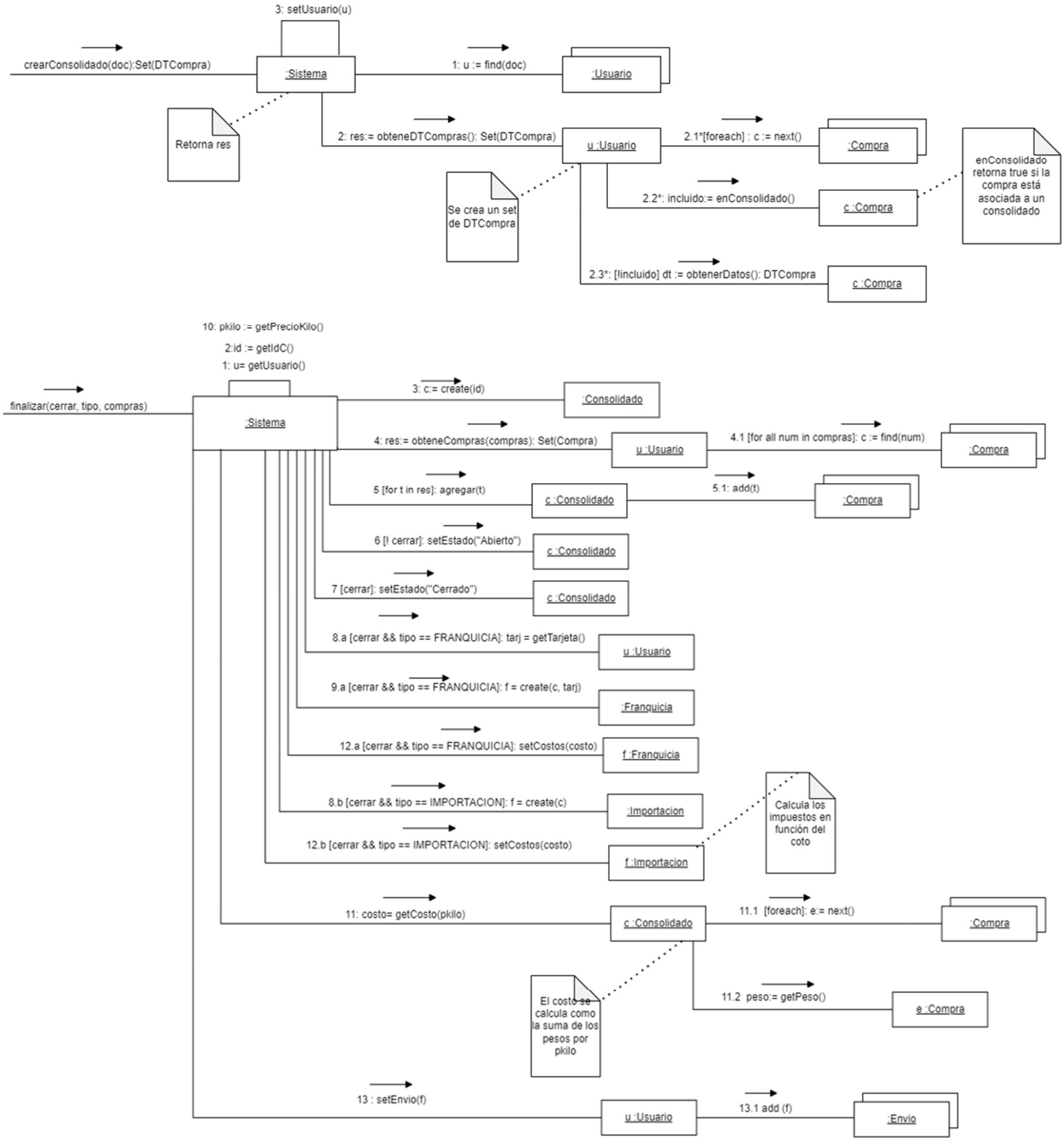

<<Sistema>>
u:=ingresar(CI) :DataUsuario
Recuerda Usuario seleccionado
alt
[u.tipo==Mensual]
alt
[desea Ingresar Vehiculo?]
altaVehiculo(modelo, marca, mat)
[elegir existente]
listarVehiculos: Set(DataVehiculo)
seleccionarVehiculo(mat)
Recuerda Vehiculo seleccionado
ingresaRango(entrada, salida) : Int
alt
[u.tipo==Visitante]
enviarFactura(enviar:bool)
generarTicket()
Se libera la memoria

Programación 4

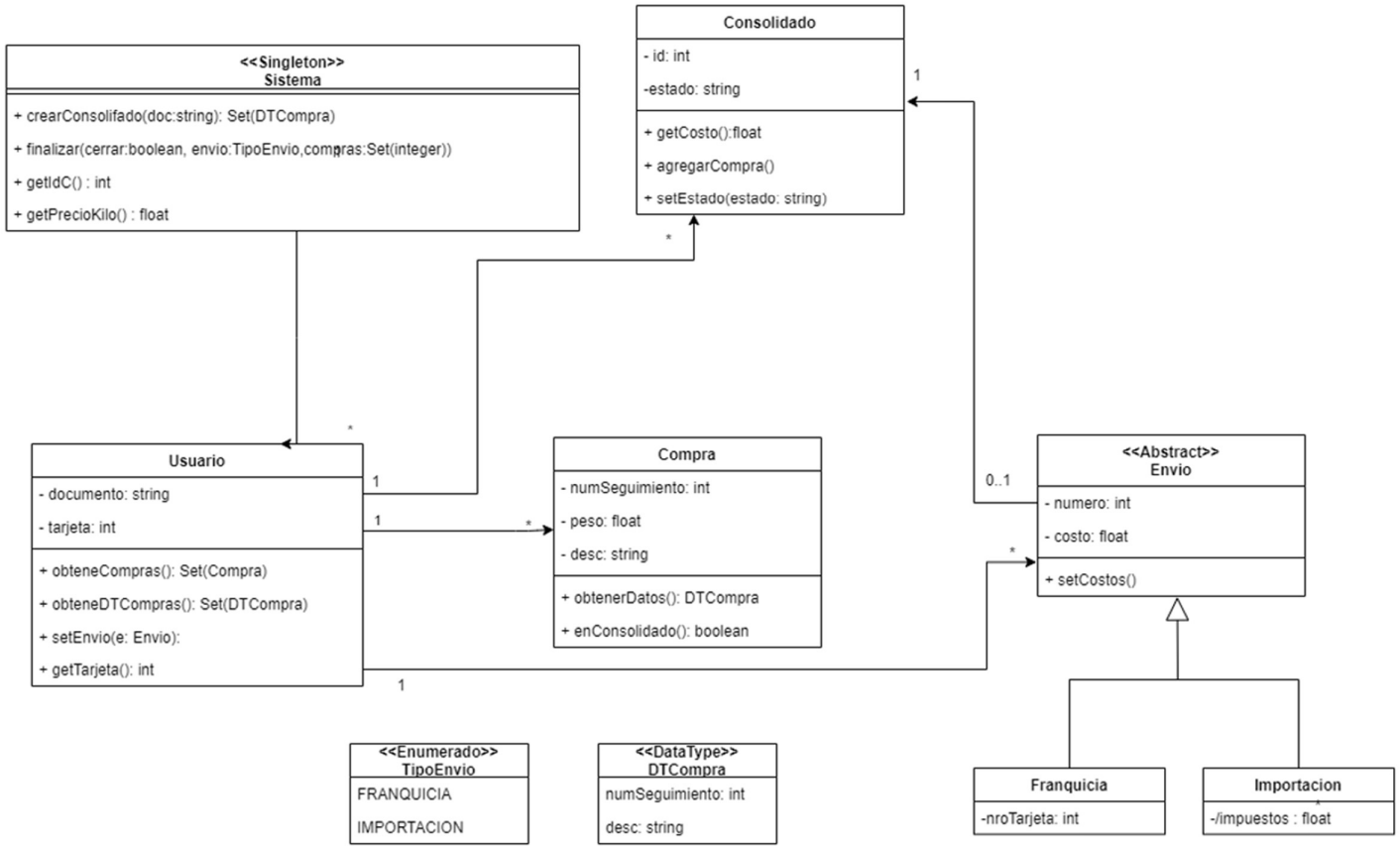
SOLUCIÓN EXAMEN FEBRERO 2022

Problema 2

a)



b)


<<Singleton>>
Sistema
+ crearConsolifado(doc:string): Set(DTCompra)
+ finalizar(cerrar:boolean, envio:TipoEnvio,compras:Set(integer))
+ getIdC() : int
+ getPrecioKilo() : float
Consolidado
- id: int
-estado: string
+ getCosto():float
+ agregarCompra()
+ setEstado(estado: string)
1
*
Usuario
- documento: string
- tarjeta: int
+ obteneCompras(): Set(Compra)
+ obteneDTCompras(): Set(DTCompra)
+ setEnvio(e: Envio):
+ getTarjeta(): int
1
1
*
Compra
- numSeguimiento: int
- peso: float
- desc: string
+ obtenerDatos(): DTCompra
+ enConsolidado(): boolean
0..1
*
<<Abstract>>
Envio
- numero: int
- costo: float
+ setCostos()
1
<<Enumerado>>
TipoEnvio
FRANQUICIA
IMPORTACION
<<DataType>>
DTCompra
numSeguimiento: int
desc: string
Franquicia
-nroTarjeta: int
Importacion
-/impuestos : float

Programación 4

SOLUCIÓN EXAMEN FEBRERO 2022

Problema 3

Parte A)

```
// ICine.h
class ICine {
    virtual set<DtAsiento> asientosDisponibles(string idEspec,
DtFechaHora fechaFunc) = 0;
    virtual void realizarReserva(string idEspec, DtFechaHora
fechaFunc, set<DtAsiento> asientos) = 0;
     virtual ~ICine() {};
};

// CtrlCine.h
class CtrlCine: public ICine {
private:
    static CtrlCine* instancia;
    map<string, Espectaculo*> espectaculos;
    map<string, Sala*> salas;
    CtrlCine();
public:
    set<DtAsiento>      asientosDisponibles(string      idEspec,
DtFechaHora fechaFunc);
    void realizarReserva(string idEspec, DtFechaHora fechaFunc,
set<DtAsiento> asientos);
    static CtrlCine* darInstancia();
     ~CtrlCine();
};

// Teatro.h
class Teatro: public Espectaculo {
private:
    string elenco;
public:
    Teatro(string id, string nombre, string descripcion, string
elenco);
    virtual ~Teatro();
};
```

Parte B)

```
// CtrlCine.cpp
set<DtAsiento>   CtrlCine::asientosDisponibles(string   idEspec,
DtFechaHora fechaFunc) {
    Espectaculo* e =  espectaculos[idEspec];
    set<DtAsiento> dts = e->asientosDisponibles(fechaFunc);
    return dts;
}
```

```
// Espectaculo.cpp
set<DtAsiento>      Espectaculo::asientosDisponibles(DtFechaHora
fechaFunc) {
    Funcion* f =  funciones[fechaFunc];
    set<DtAsiento> dts = f->asientosDisponibles();
    return dts;
}

// Funcion.cpp
set<DtAsiento> Funcion::asientosDisponibles() {
    set<DtAsiento> dts;
    set<Asiento*> habilitados = sala->asientosHabilitados();
    for (set<Asiento*>::iterator it = habilitados.begin(); it !=
habilitados.end(); ++it) {
        Asiento* a = *it;
        bool reservado = reservados.find(a) != reservados.end();
        if (!reservado)
            dts.insert(a.datos());
    }
    return dts;
}

// Sala.cpp
set<Asiento> Sala::asientosHabilitados() {
    set<Asiento*> habilitados;
    for (set<Asiento*>::iterator it = asientos.begin(); it !=
asientos.end(); ++it) {
         Asiento* a = *it;
         if (a->habilitado())
            habilitados.insert(a);
    }
    return habilitados;
}

// Asiento.cpp
DtAsiento Asiento::datos() {
    return DtAsiento(fila, columna);
}

bool Asiento::habilitado() {
    return habilitado;
}
```