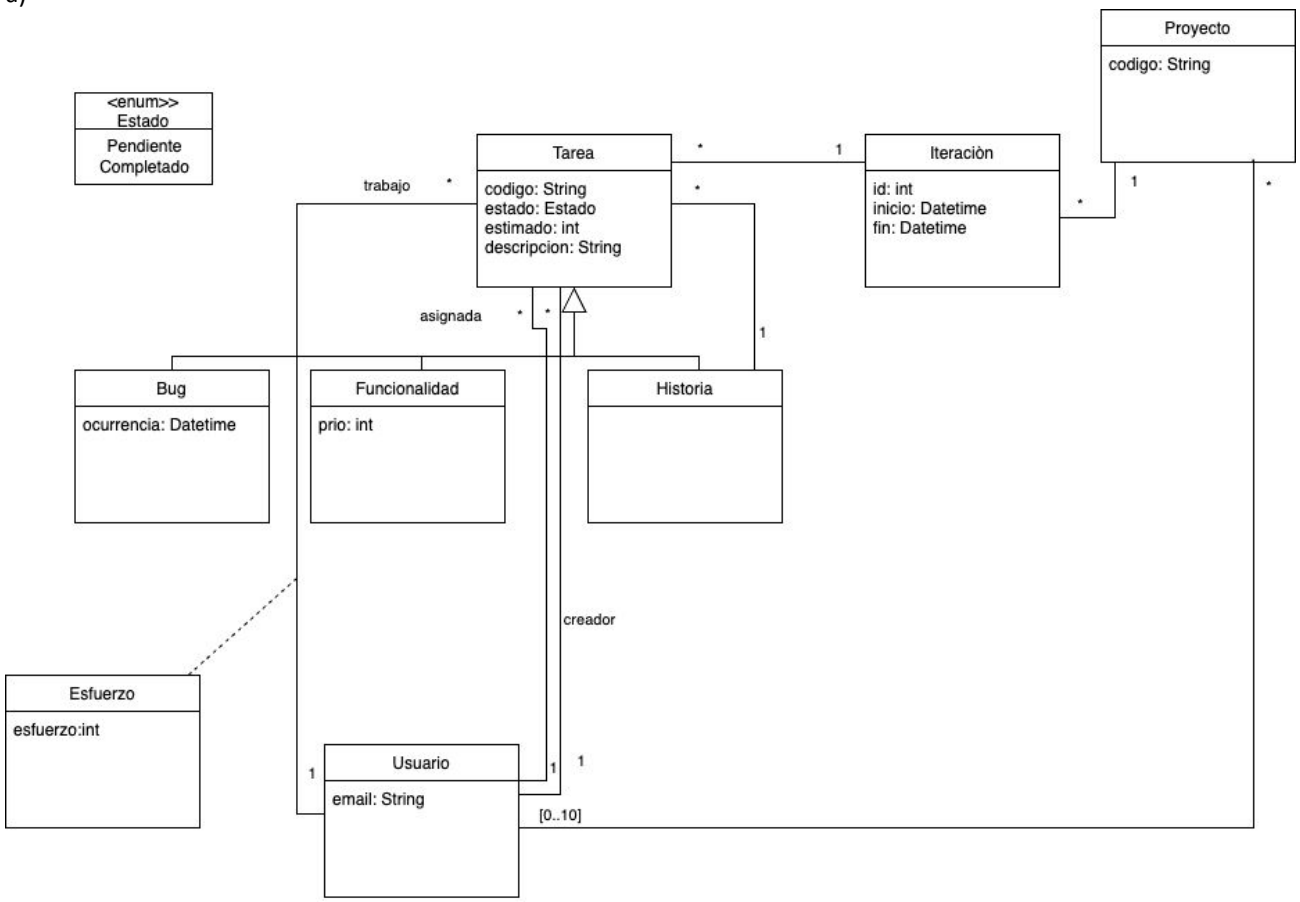
# Programación 4

SOLUCIÓN EXAMEN FEBRERO 2025

**Problema 1**

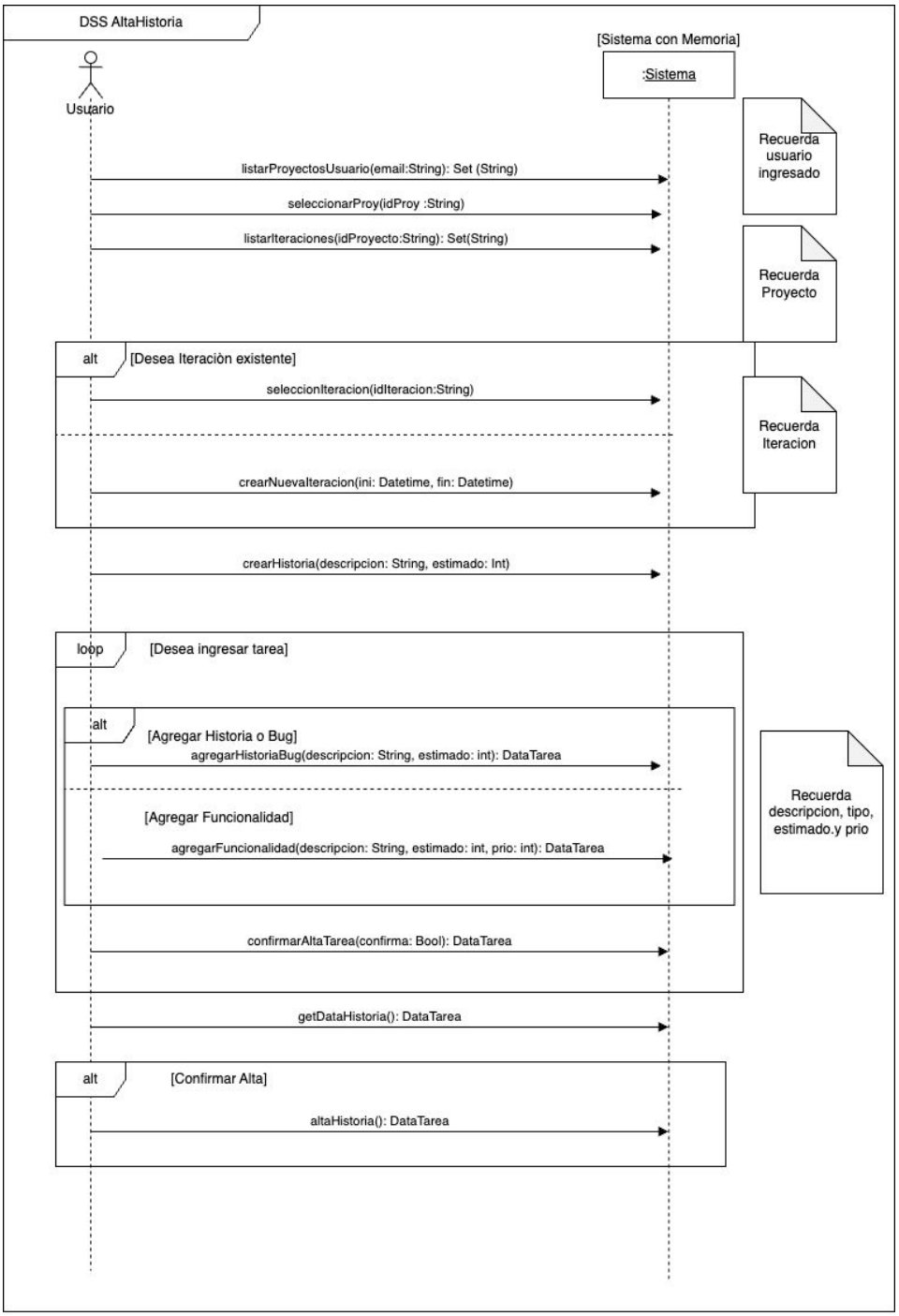
a)

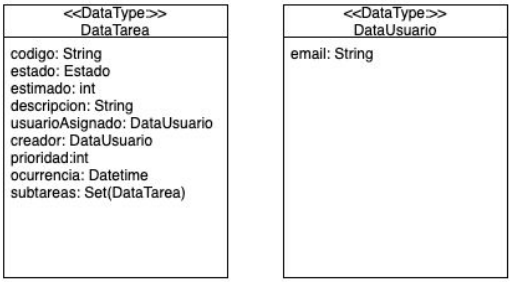



Restricciones:

- Unicidad:
  - Código de proyecto
  - Id Iteración única relativa a proyecto
  - Email de usuario
  - Código de tarea
- Dominio
  - Dos iteraciones no se superponen en un mismo proyecto
  - Formato de id de tarea
  - Fecha inicio <= fin en iteración
- Negocio
  - Un usuario no puede ser asignado a una tarea que no pertenece a un proyecto al cual está asignado
- Circular
  - Una historia no puede estar compuesta por sí misma.

b)

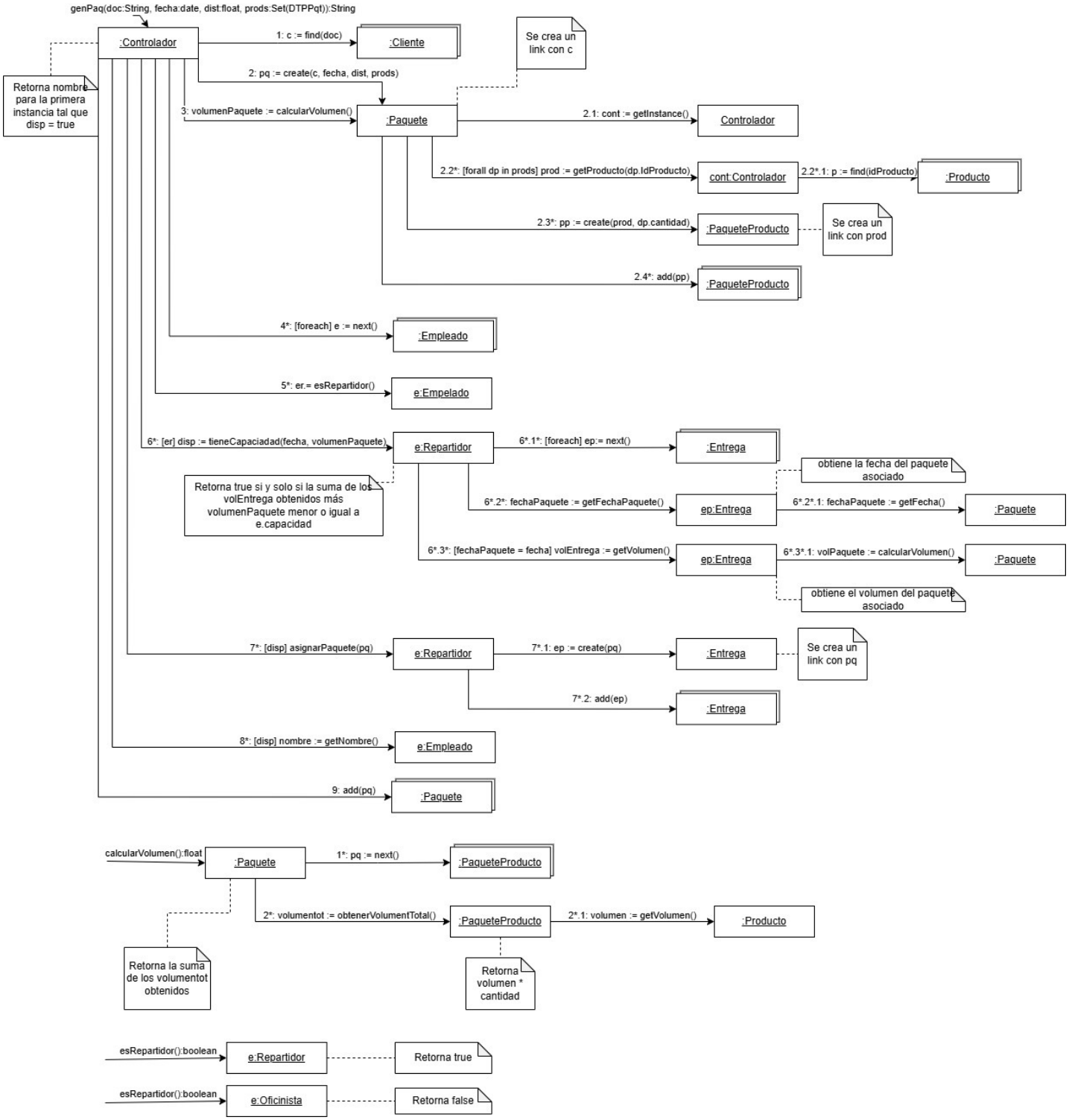

DSS AltaHistoria
[Sistema con Memoria]
:Sistema
Usuario
Recuerda usuario ingresado
listarProyectosUsuario(email:String): Set (String)
seleccionarProy(idProy :String)
listarIteraciones(idProyecto:String): Set(String)
Recuerda Proyecto
alt
[Desea Iteraciòn existente]
seleccionIteracion(idIteracion:String)
Recuerda Iteracion
crearNuevaIteracion(ini: Datetime, fin: Datetime)
crearHistoria(descripcion: String, estimado: Int)
loop
[Desea ingresar tarea]
alt
[Agregar Historia o Bug]
agregarHistoriaBug(descripcion: String, estimado: int): DataTarea
[Agregar Funcionalidad]
agregarFuncionalidad(descripcion: String, estimado: int, prio: int): DataTarea
Recuerda descripcion, tipo, estimado.y prio
confirmarAltaTarea(confirma: Bool): DataTarea
getDataHistoria(): DataTarea
alt
[Confirmar Alta]
altaHistoria(): DataTarea

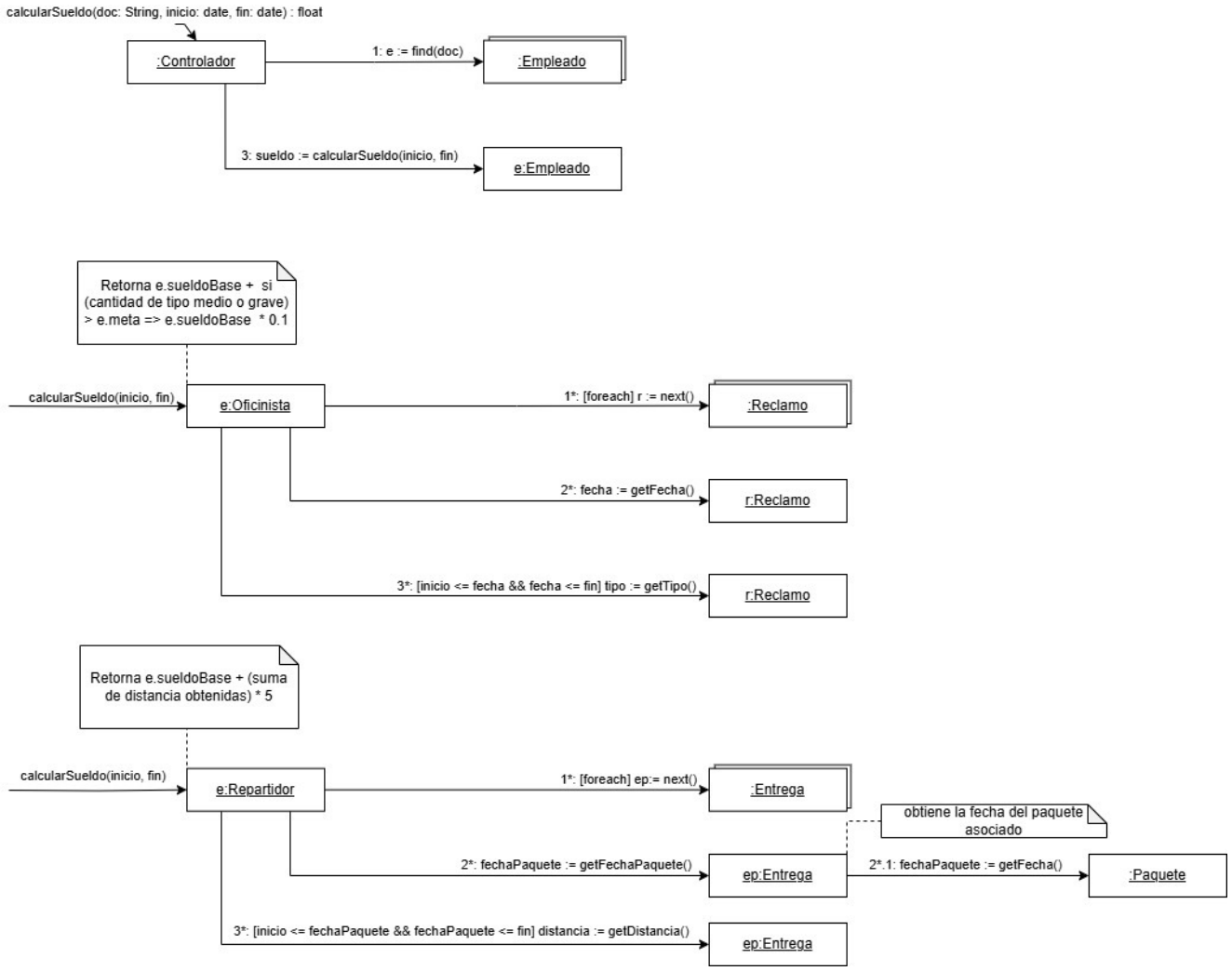

<<DataType>>
DataTarea
codigo: String
estado: Estado
estimado: int
descripcion: String
usuarioAsignado: DataUsuario
creador: DataUsuario
prioridad:int
ocurrencia: Datetime
subtareas: Set(DataTarea)


<<DataType>>
DataUsuario
email: String

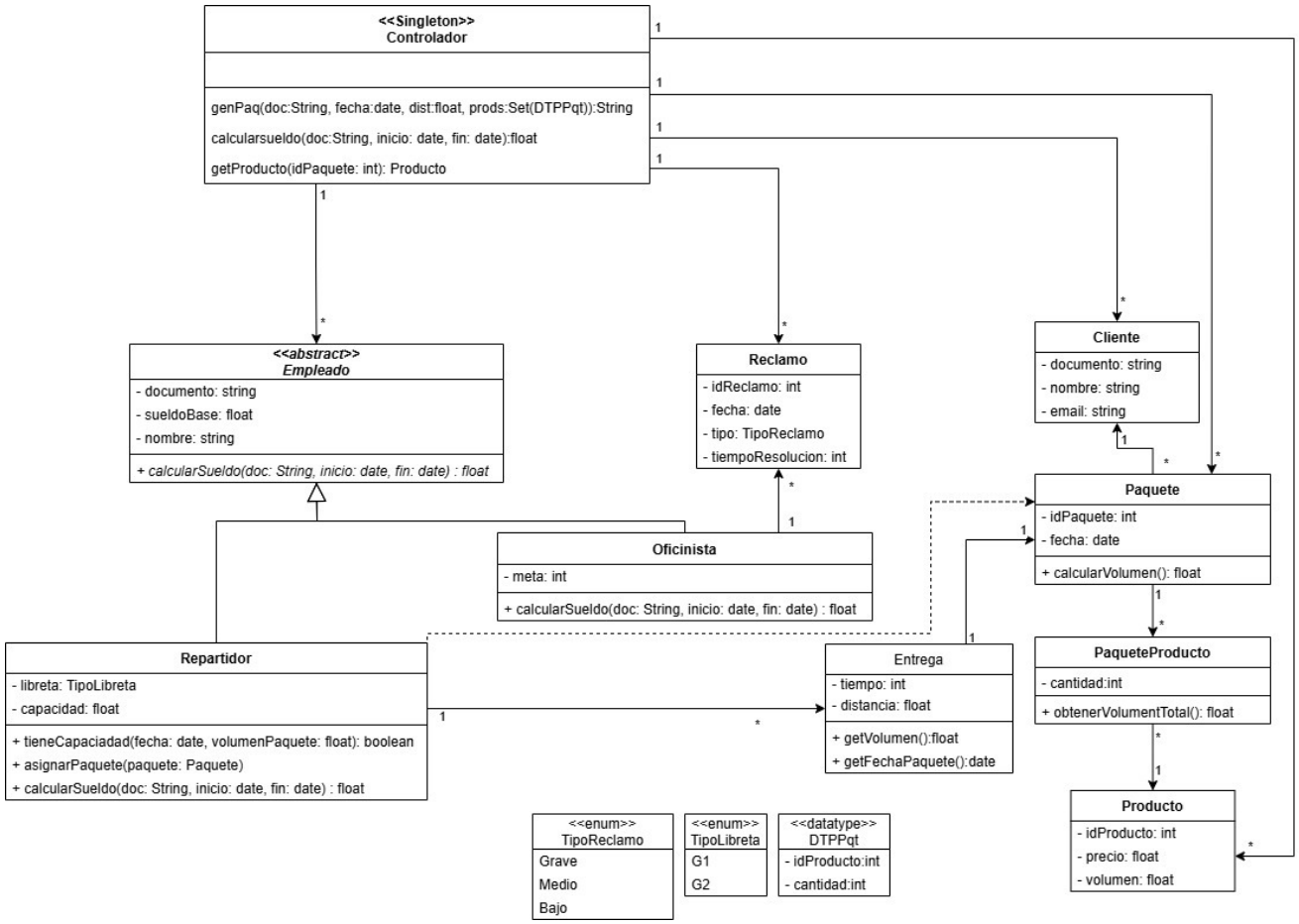
## Problema 2

a)


genPaq(doc:String, fecha:date, dist:float, prods:Set(DTPPqt)):String
:Controlador
1: c := find(doc)
:Cliente
Se crea un link con c
2: pq := create(c, fecha, dist, prods)
Retorna nombre para la primera instancia tal que disp = true
3: volumenPaquete := calcularVolumen()
:Paquete
2.1: cont := getInstance()
Controlador
2.2*: [forall dp in prods] prod := getProducto(dp.IdProducto)
cont:Controlador
2.2*.1: p := find(idProducto)
:Producto
2.3*: pp := create(prod, dp.cantidad)
:PaqueteProducto
Se crea un link con prod
2.4*: add(pp)
:PaqueteProducto
4*: [foreach] e := next()
:Empleado
5*: er.= esRepartidor()
e:Empelado
6*: [er] disp := tieneCapaciadad(fecha, volumenPaquete)
e:Repartidor
6*.1*: [foreach] ep:= next()
:Entrega
obtiene la fecha del paquete asociado
Retorna true si y solo si la suma de los volEntrega obtenidos más volumenPaquete menor o igual a e.capacidad
6*.2*: fechaPaquete := getFechaPaquete()
ep:Entrega
6*.2*.1: fechaPaquete := getFecha()
:Paquete
6*.3*: [fechaPaquete = fecha] volEntrega := getVolumen()
ep:Entrega
6*.3*.1: volPaquete := calcularVolumen()
:Paquete
obtiene el volumen del paquete asociado
7*: [disp] asignarPaquete(pq)
e:Repartidor
7*.1: ep := create(pq)
:Entrega
Se crea un link con pq
7*.2: add(ep)
:Entrega
8*: [disp] nombre := getNombre()
e:Empleado
9: add(pq)
:Paquete
calcularVolumen():float
:Paquete
1*: pq := next()
:PaqueteProducto
2*: volumentot := obtenerVolumentTotal()
:PaqueteProducto
2*.1: volumen := getVolumen()
:Producto
Retorna la suma de los volumentot obtenidos
Retorna volumen * cantidad
esRepartidor():boolean
e:Repartidor
Retorna true
esRepartidor():boolean
e:Oficinista
Retorna false


calcularSueldo(doc: String, inicio: date, fin: date) : float
:Controlador
1: e := find(doc)
:Empleado
3: sueldo := calcularSueldo(inicio, fin)
e:Empleado
Retorna e.sueldoBase + si (cantidad de tipo medio o grave) > e.meta => e.sueldoBase * 0.1
calcularSueldo(inicio, fin)
e:Oficinista
1*: [foreach] r := next()
:Reclamo
2*: fecha := getFecha()
r:Reclamo
3*: [inicio <= fecha && fecha <= fin] tipo := getTipo()
r:Reclamo
Retorna e.sueldoBase + (suma de distancia obtenidas) * 5
calcularSueldo(inicio, fin)
e:Repartidor
1*: [foreach] ep:= next()
:Entrega
obtiene la fecha del paquete asociado
2*: fechaPaquete := getFechaPaquete()
ep:Entrega
2*.1: fechaPaquete := getFecha()
:Paquete
3*: [inicio <= fechaPaquete && fechaPaquete <= fin] distancia := getDistancia()
ep:Entrega

b)




## Problema 3

a)

El `GestorDeOperaciones` no necesita ser un singleton porque su función es coordinar la ejecución de estrategias en los agentes, algo que no requiere una única instancia.

El `GestorDeColecciones` sí debe ser un singleton porque almacena y administra las colecciones de proyectos, agentes y sensores. Si hubiera varias instancias, los datos podrían estar duplicados o podría haber inconsistencias.

b) Se aplica el patrón de diseño Strategy. Este patrón permite que AgenteIA utilice diferentes algoritmos sin necesidad de modificar su código. En lugar de definir un único método fijo, AgenteIA mantiene una referencia a un objeto de tipo Algoritmo, lo que le permite intercambiar estrategias según sea necesario.

c)

```
class Algoritmo {
public:
    virtual void ejecutar(std::set<int> datos) = 0;
    virtual ~Algoritmo() {}
};
```

```
class Actuador : public Algoritmo {
public:
    void ejecutar(std::set<int> datos);
    virtual ~Actuador();
};

class Proyecto {
private:
    int id;
    int presupuesto;
    std::set<AgenteIA*> agentes;
public:
    Proyecto(int id, int presupuesto);
    void ejecutarEstrategiaAgentes();
};

class AgenteIA {
private:
    int id;
    Algoritmo* estrategia;
public:
    void setAlgoritmo(Algoritmo* a);
    void ejecutarEstrategia(std::set<int>& datos);
    std::set<int> obtenerMediciones();
};

class GestorDeColecciones {
private:
    static GestorDeColecciones* instancia;
    std::map<int, Proyecto*> proyectos;
    std::map<int, AgenteIA*> agentesIA;
    std::map<int, Sensor*> sensores;
    GestorDeColecciones();
public:
    static GestorDeColecciones* getInstancia();
    Proyecto* getProyecto(int id);
    AgenteIA* getAgenteIA(int id);
    Sensor* getSensor(string serie);
};

void GestorDeOperaciones::ejecutarEstrategia(int idProyecto) {
    GestorDeColecciones* gestorCol = GestorDeColecciones::getInstancia();
    Proyecto* proyecto = gestorCol->getProyecto(idProyecto);
    proyecto->ejecutarEstrategiaAgentes();
}

void Proyecto::ejecutarEstrategiaAgentes() {
    std::set<AgenteIA*> agentes = getAgentes();
    for (std::set<AgenteIA*>::iterator it = agentes.begin();
         it != agentes.end(); ++it) {
        std::set<int> mediciones = (*it)->obtenerMediciones();
        (*it)->ejecutarEstrategia(mediciones);
    }
}
```