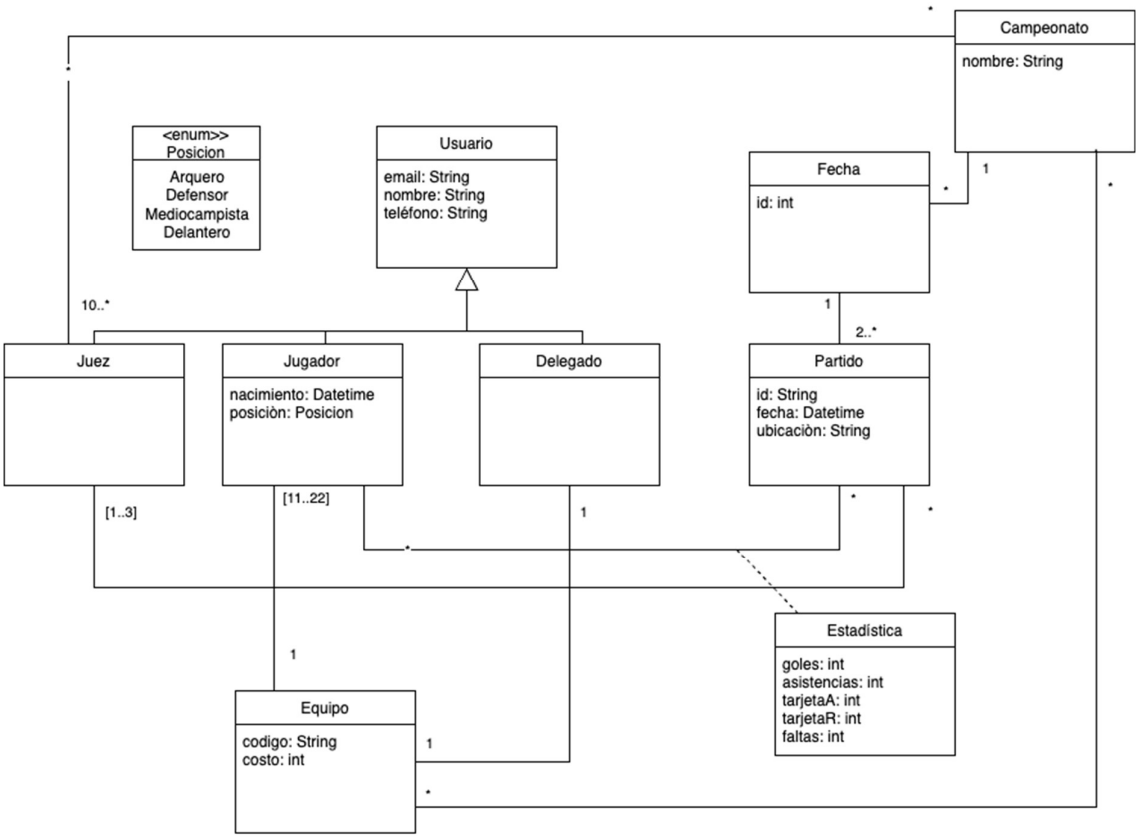
Programación 4

SOLUCIÓN PARCIAL FINAL EDICIÓN 2024

Problema 1

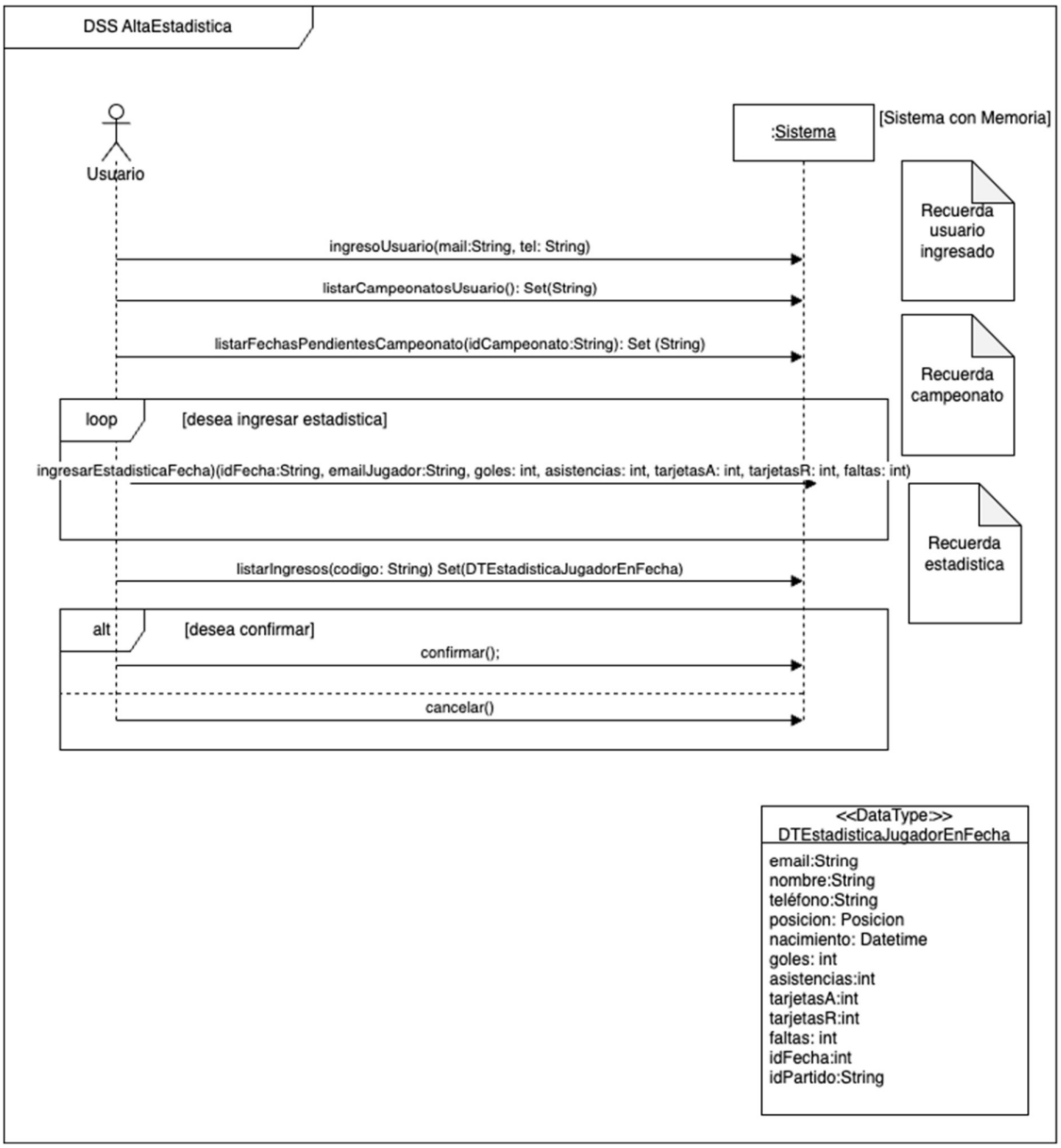
a)



Restricciones:

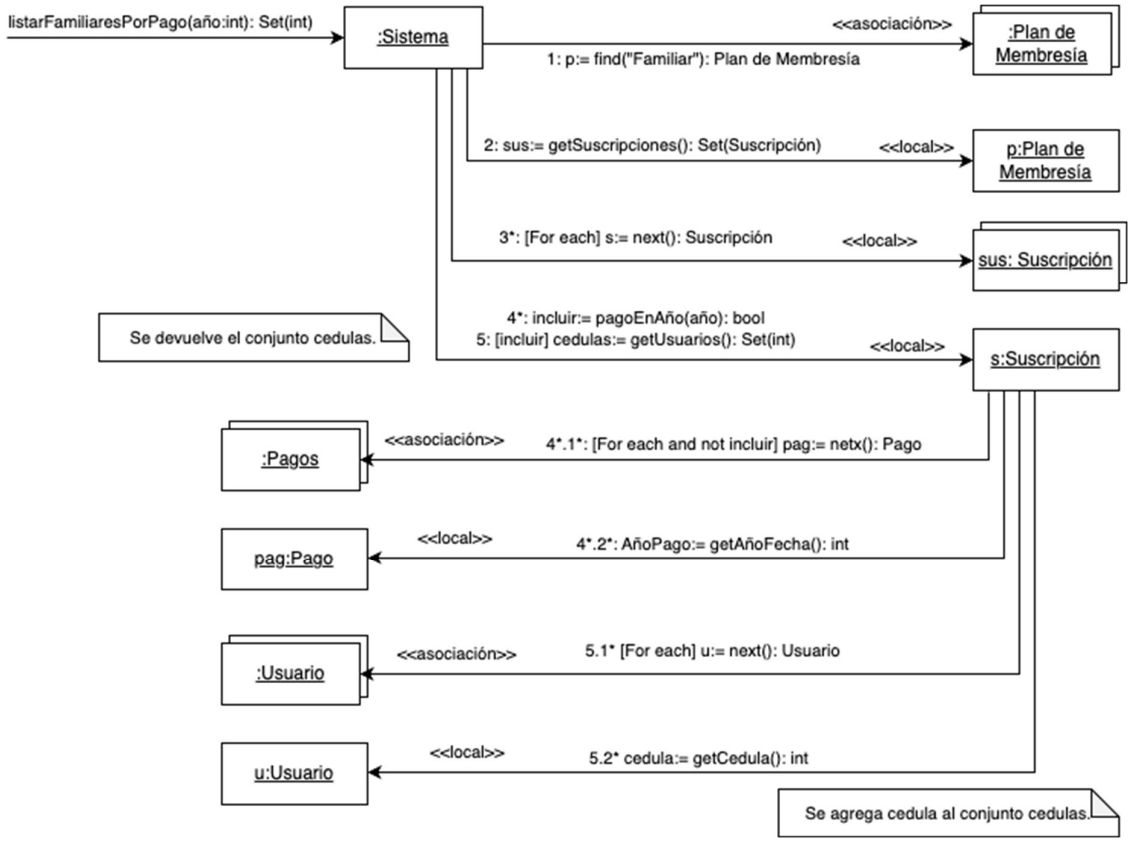
- Unicidad:
  - Email único por usuario
  - Fecha única por campeonato
  - Nombre único por campeonato
  - Id de partido único por fecha.
  - Equipo con nombre único.
- Dominio
  - Los valores de fecha son enteros secuenciales para cada campeonato comenzando en 1.
- Negocio
  - Dos jueces no pueden estar asignados a partidos concurrentes en día/hora.
  - Todo equipo debe tener al menos un arquero.
- Circular
  - Los jueces asignados a un partido deben estar asignados al campeonato al que pertenece ese partido.

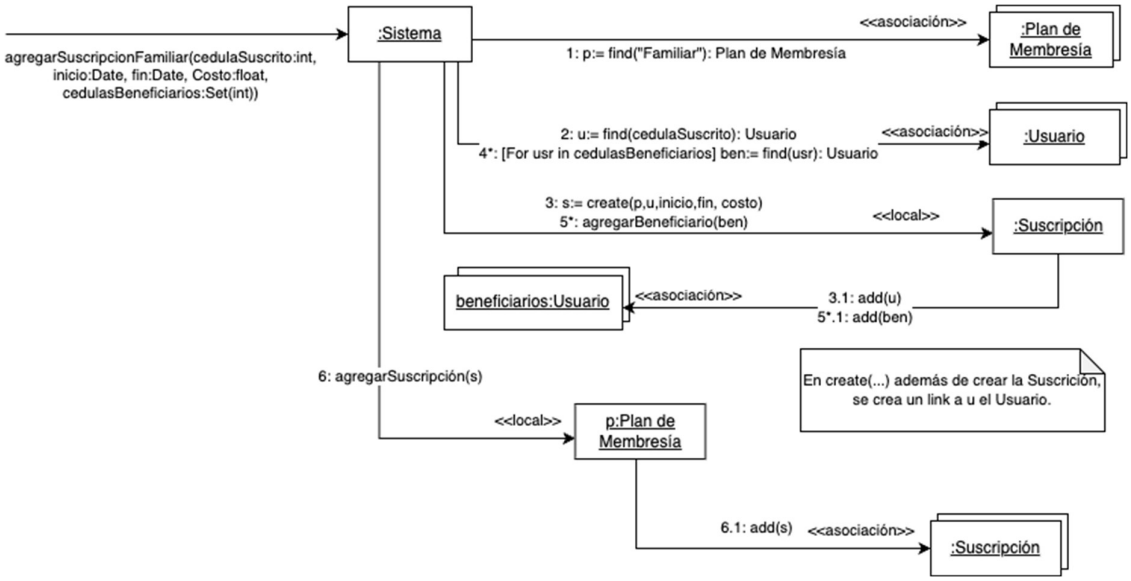
b)


DSS AltaEstadistica
Usuario
:Sistema
[Sistema con Memoria]
Recuerda usuario ingresado
ingresoUsuario(mail:String, tel: String)
listarCampeonatosUsuario(): Set(String)
listarFechasPendientesCampeonato(idCampeonato:String): Set (String)
Recuerda campeonato
loop
[desea ingresar estadistica]
ingresarEstadisticaFecha)(idFecha:String, emailJugador:String, goles: int, asistencias: int, tarjetasA: int, tarjetasR: int, faltas: int)
Recuerda estadistica
listarIngresos(codigo: String) Set(DTEstadisticaJugadorEnFecha)
alt
[desea confirmar]
confirmar();
cancelar()
<<DataType>>
DTEstadisticaJugadorEnFecha
email:String
nombre:String
teléfono:String
posicion: Posicion
nacimiento: Datetime
goles: int
asistencias:int
tarjetasA:int
tarjetasR:int
faltas: int
idFecha:int
idPartido:String

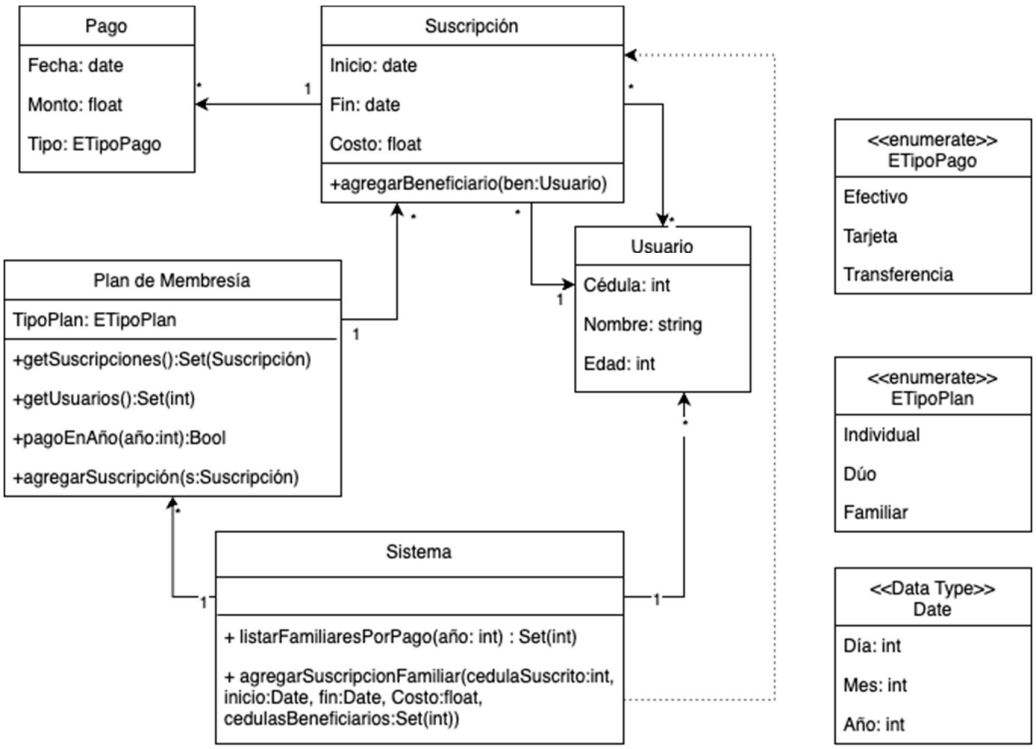
Problema 2

a)


listarFamiliaresPorPago(año:int): Set(int)
:Sistema
<<asociación>>
:Plan de Membresía
1: p:= find("Familiar"): Plan de Membresía
2: sus:= getSuscripciones(): Set(Suscripción)
<<local>>
p:Plan de Membresía
3*: [For each] s:= next(): Suscripción
<<local>>
sus: Suscripción
Se devuelve el conjunto cedulas.
4*: incluir:= pagoEnAño(año): bool
5: [incluir] cedulas:= getUsuarios(): Set(int)
<<local>>
s:Suscripción
:Pagos
<<asociación>>
4*.1*: [For each and not incluir] pag:= netx(): Pago
pag:Pago
<<local>>
4*.2*: AñoPago:= getAñoFecha(): int
:Usuario
<<asociación>>
5.1* [For each] u:= next(): Usuario
u:Usuario
<<local>>
5.2* cedula:= getCedula(): int
Se agrega cedula al conjunto cedulas.


agregarSuscripcionFamiliar(cedulaSuscrito:int, inicio:Date, fin:Date, Costo:float, cedulasBeneficiarios:Set(int))
:Sistema
<<asociación>>
:Plan de Membresía
1: p:= find("Familiar"): Plan de Membresía
2: u:= find(cedulaSuscrito): Usuario
4*: [For usr in cedulasBeneficiarios] ben:= find(usr): Usuario
<<asociación>>
:Usuario
3: s:= create(p,u,inicio,fin, costo)
5*: agregarBeneficiario(ben)
<<local>>
:Suscripción
beneficiarios:Usuario
<<asociación>>
3.1: add(u)
5*.1: add(ben)
6: agregarSuscripción(s)
En create(...) además de crear la Suscrición, se crea un link a u el Usuario.
<<local>>
p:Plan de Membresía
6.1: add(s)
<<asociación>>
:Suscripción

b)



Problema 3

```
// .h's

class Controlador {
    private:
        std::map<int, Contenido *> contenidos;
        static int genId;
    public:
        void agregarContenido(DTContenido *);
        int contenidoMasVisitado();
};

class Contenido {
    private:
        int idCont;
        std::vector<Visualizacion *> visualizaciones;
    public:
        Contenido(int);
        Virtual int calcTotVis() = 0;
};

class Serie : public Contenido {
    private:
        std::vector<Capitulo *> capitulos;
    public:
        Serie(int);
        int calcTotVis();
};
```

```
class Capitulo : public Contenido {
    private:
        int numCap;
    public:
        Capitulo(int, int);
        int calcTotVis();
};

// .cpp's

int Controlador::genId = 1;

Controlador::agregarContenido(DTContenido *dtc) {
    DTSerie *dts = dynamic_cast<DTSerie *>(dtc);
    int nuevoId = this->genId;
    this->genId++;
    Contenido *nuevoC;
    if (dts == NULL) {
        nuevoC = new Pelicula(nuevoId);
    } else {
        Serie *nuevaS = new Serie(nuevoId);
        int cantCaps = dts->getCantCapitulos();
        Capitulo *cap;
        for (int i=1; i<=cantCaps; i++) {
            cap = new Capitulo(this->genId, i);
            nuevaS->getCapitulos()->insert(cap);
            this->contenidos[this->genId] = cap;
            this->genId++;
        }
        nuevoC = nuevaS;
    }
    this->contenidos[nuevoId] = nuevoC;
}

int Controlador::contenidoMasVisitado() {
    int vis, idMaxVis, maxVis = 0;
    std::map<int, Contenido *>::iterator it;
    for (it = this->contenidos.begin();it != this->contenidos.end(); ++it){
        vis = it->second->calcTotVis();
        if (vis > maxVis) {
            maxVis = vis;
            idMaxVis = *it->first;
        }
    }
    return idMaxVis;
}

Contenido::Contenido(int idCont) {
    this->idCont = idCont;
}

Serie::Serie(int idCont) : Contenido(idCont) {
}

int Serie::calcTotVis() {
    int result = 0;
    std::vector<Capitulo *>::iterator it;
    for (it = this->capitulos.begin(); it != this->capitulos.end(); ++it) {
            result = result + (*it)->calcTotVis();
    return result;
 }
```

```
Capitulo::Capitulo(int idCont, int numCap) : Contenido(idCont) {
    this->numCap = numCap;
}

int Capitulo::calcTotVis() {
    int result = 0;
    std::vector<Visualizacion *>::iterator it;
    for(it = this->getVisualizaciones().begin();
        it != this->getVisualizaciones().end(); ++it) {
            result = result + (*it)->getSegs();
    return result;
}
```