Programación 4

EXAMEN DICIEMBRE 2023

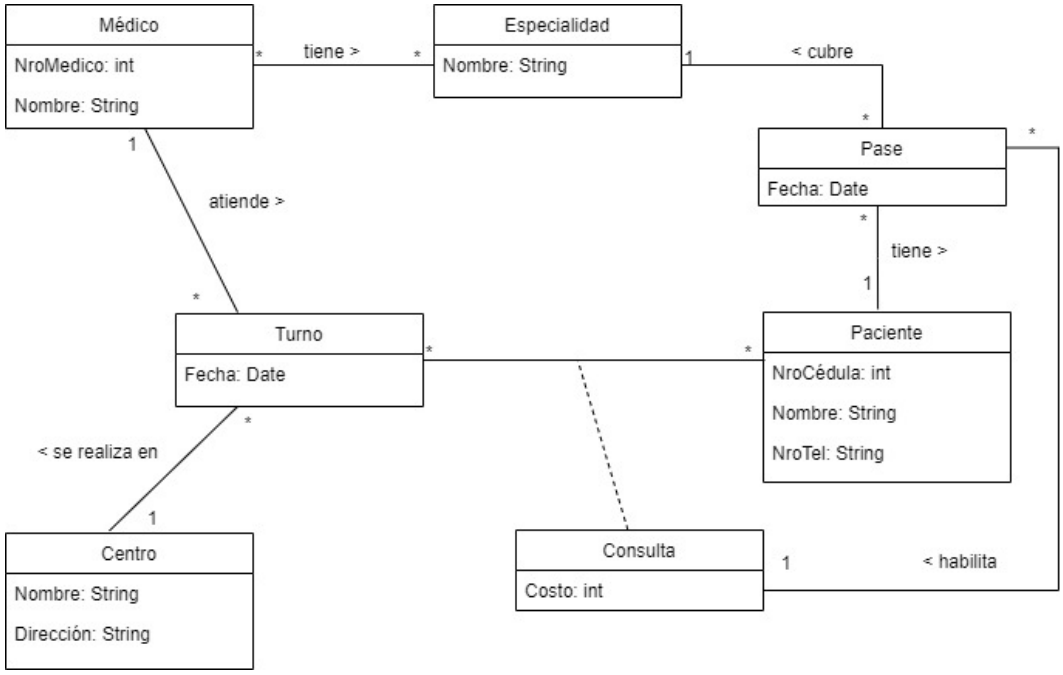
SOLUCIÓN

Problema 1 (30 puntos)

Se desea crear un software para registrar las consultas y turnos de médicos de un prestador de salud.

Se pide:

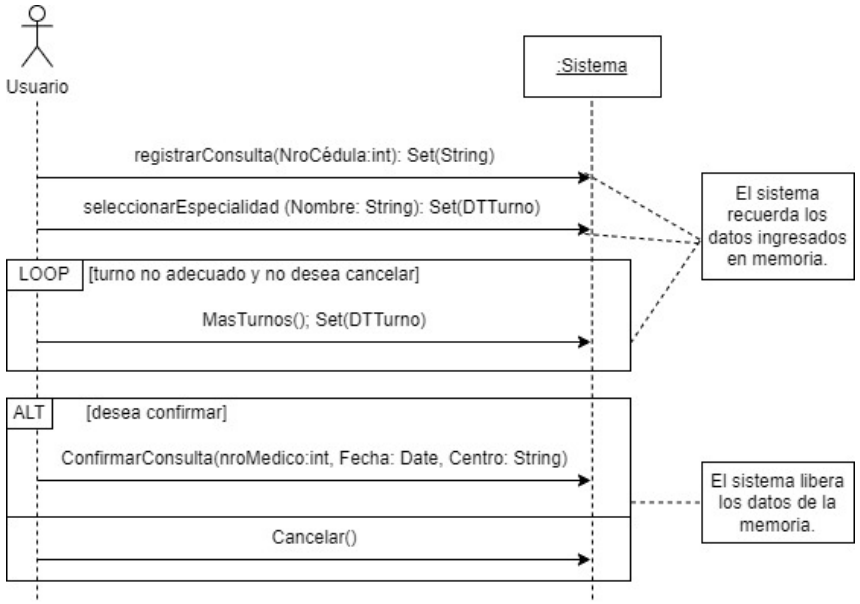
1. Realizar el Modelo de Dominio de la realidad planteada, incluyendo todas las restricciones que considere necesarias en lenguaje natural.

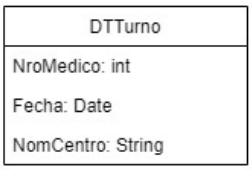


Restricciones

- No existen dos instancias de Médico con igual valor para el atributo NroMédico.
- No existen dos instancias de Especialidad con igual valor para el atributo Nombre.
- No existen dos instancias de Paciente con igual valor para el atributo NroCédula.
- No existen dos instancias de Centro con igual valor para el atributo Nombre.
- Todos los turnos asociados a un Médico cuya fecha sea la misma están asociados al mismo Centro.
- Un Paciente está asociado a una Consulta de un Turno de un Médico de una Especialidad, si la Consulta tiene asociado un Pase con fecha vigente (o sea, fechaPase <= fechaTurnoY <= fechaPase + 1 año) .

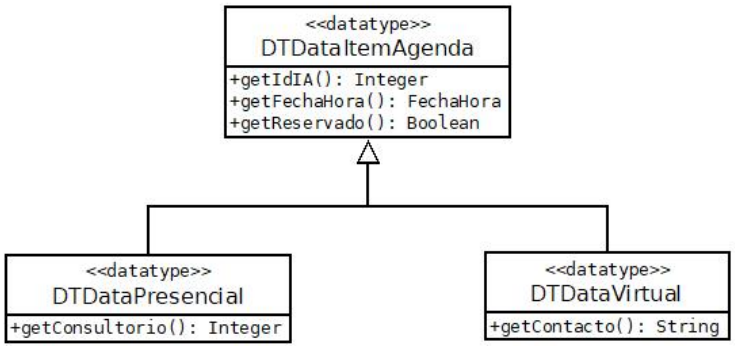
2. Realizar un Diagrama de Secuencia del Sistema (DSS) para el Caso de Uso. Indique el uso de memoria del Sistema y de datatypes, si corresponde.



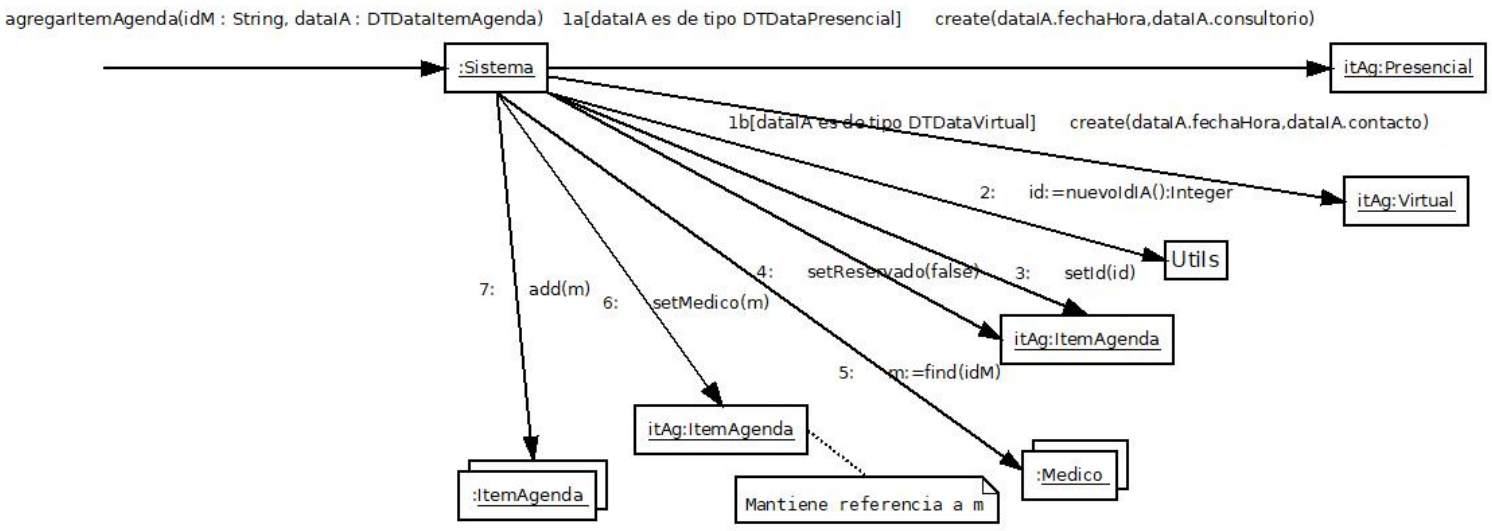


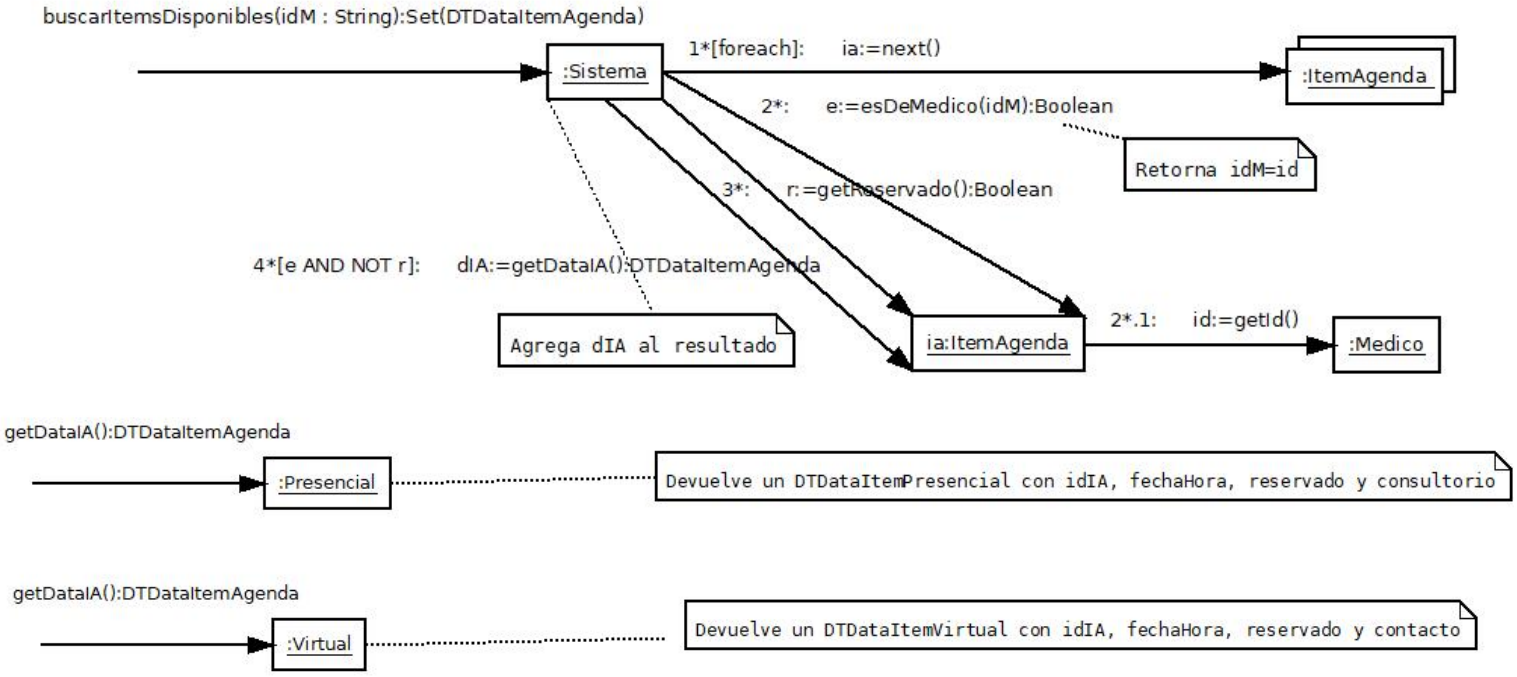
<u>Problema 2</u> (35 puntos)

a)

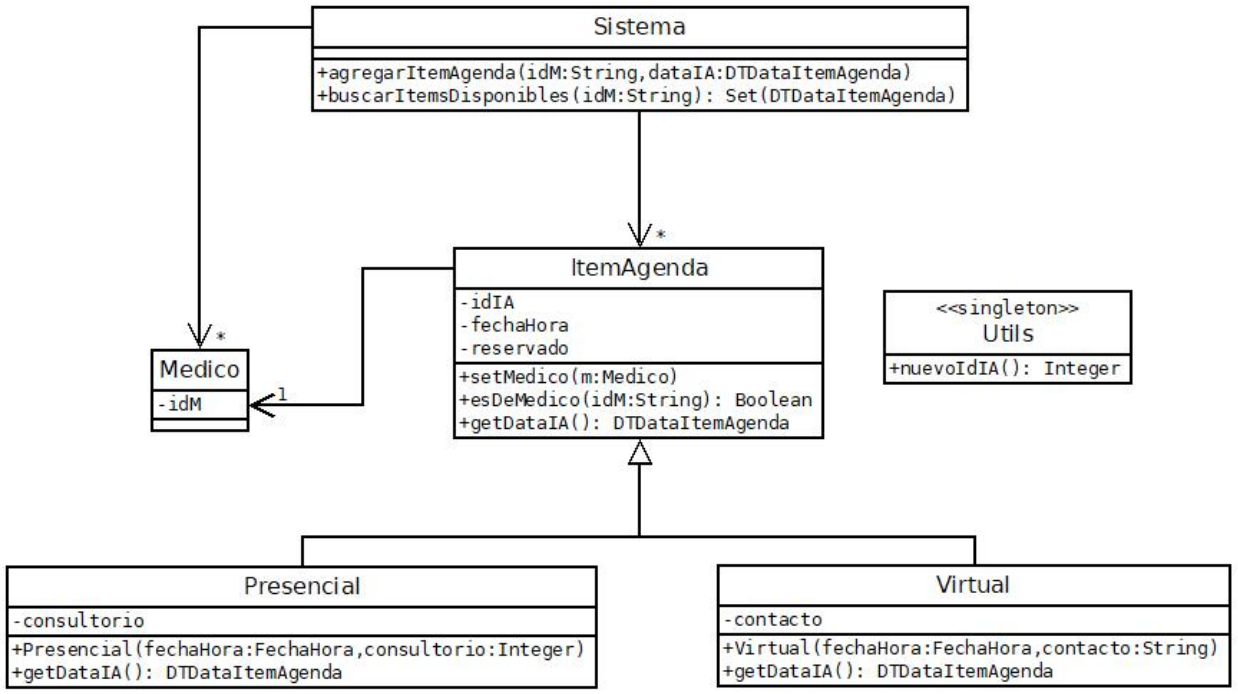


b)




buscarItemsDisponibles(idM : String):Set(DTDataItemAgenda)
:Sistema
1*[foreach]: ia:=next()
:ItemAgenda
2*: e:=esDeMedico(idM):Boolean
Retorna idM=id
3*: r:=getReservado():Boolean
4*[e AND NOT r]: dIA:=getDataIA():DTDataItemAgenda
Agrega dIA al resultado
ia:ItemAgenda
2*.1: id:=getId()
:Medico
getDataIA():DTDataItemAgenda
:Presencial
Devuelve un DTDataItemPresencial con idIA, fechaHora, reservado y consultorio
getDataIA():DTDataItemAgenda
:Virtual
Devuelve un DTDataItemVirtual con idIA, fechaHora, reservado y contacto

c)


Sistema
+agregarItemAgenda(idM:String,dataIA:DTDataItemAgenda)
+buscarItemsDisponibles(idM:String): Set(DTDataItemAgenda)
ItemAgenda
-idIA
-fechaHora
-reservado
+setMedico(m:Medico)
+esDeMedico(idM:String): Boolean
+getDataIA(): DTDataItemAgenda
<<singleton>>
Utils
+nuevoIdIA(): Integer
Medico
-idM
Presencial
-consultorio
+Presencial(fechaHora:FechaHora,consultorio:Integer)
+getDataIA(): DTDataItemAgenda
Virtual
-contacto
+Virtual(fechaHora:FechaHora,contacto:String)
+getDataIA(): DTDataItemAgenda

Problema 3 (35 puntos)

Se está desarrollando un sistema de reservas turísticas que permite reservar diferentes elementos "reservables" como hoteles y vuelos.

Se pide:

a) Implementar en C++ el .h de la interfaz `Reservable`.

```
class Reservable {
public:
    // Operación virtual y pura, es una interfaz
    virtual bool reservar(Cliente *) = 0;


    // Operación virtual y pura, es una interfaz
    virtual void liberar(Cliente *) = 0;

    // Destructor virtual para permitir la destrucción polimórfica
    virtual ~Reservable() = 0;
};
```

b) Implementar en C++ el .h de la clase `Vuelo`. Incluir el(los) constructor(es) que sea(n) necesario(s). No incluir destructor ni operaciones get/set.

```
class Vuelo : public Reservable {
private:
    int numero;
    string origen;
    string destino;
    map<int, Asiento*> asientos;

public:
    Vuelo(int numero, string origen, string destino);
    void addAsiento(Asiento *);
    bool reservar(Cliente *);
    void liberar(Cliente *);
};
```

c) Implementar en C++ el .cc de la clase `Vuelo`. No incluir destructor ni operaciones get/set.

```
Vuelo::Vuelo(int numero, std::string origen, std::string destino)
        : numero(numero), origen(origen), destino(destino) {
}

void Vuelo::addAsiento(Asiento * a){
    asientos.insert(pair<int, Asiento*>(a->getNumero(), a));
}
```

```
bool Vuelo::reservar(Cliente * p) {
    map<int, Asiento*>::iterator it;

    for (it = asientos.begin(); it != asientos.end(); ++it) {
        Asiento * actual = (Asiento *) it->second;
        if (actual->reservar(p))
            return true;
    }

    return false;
}

void Vuelo::liberar(Cliente * p) {
    map<int, Asiento*>::iterator it;

    for (it = asientos.begin(); it != asientos.end(); ++it) {
        Asiento * actual = (Asiento *) it->second;
        actual->liberar(p);
    }
}
```

d) Implementar en C++ el .h de la clase `Asiento`. Incluir el(los) constructor(es) que sea(n) necesario(s). No incluir destructor ni operaciones get/set.

```
class Asiento {
private:
    int numero;
    Cliente * ocupado;

public:
    Asiento(int numero);
    bool reservar(Cliente *);
    void liberar(Cliente *);
};
```

e) Implementar en C++ el .cc de la clase `Asiento`. No incluir destructor ni operaciones get/set.

```
Asiento::Asiento(int numero)
        : numero(numero) {}

bool Asiento::reservar(Cliente * p){
        if (ocupado == NULL){
                //si no está ocupado, se reserva el asiento
                ocupado = p;
                return true;
        }
        else
                //si está ocupado, no se puede reservar
                return false;
}
```

```
void Asiento::liberar(Cliente * p){
        if (ocupado == p)
                ocupado = NULL;
}
```

f) Implementar en C++ un main que permita
   a. Crear un cliente "Cliente"
   b. Crear el vuelo 101 de "Ciudad A" a "Ciudad B" con dos asientos (1 y 2)
   c. Reservar un asiento para el cliente
   d. Liberar la reserva del asiento para ese cliente

```
int main() {
    Cliente * cli = new Cliente("Cliente");

    Vuelo * vuelo = new Vuelo(101, "Ciudad A", "Ciudad B");

    Asiento * asiento1 = new Asiento(1);
    Asiento * asiento2 = new Asiento(2);

    vuelo->addAsiento(asiento1);
    vuelo->addAsiento(asiento2);

    vuelo->reservar(cli);
    vuelo->liberar(cli);
}
```