Programación 4

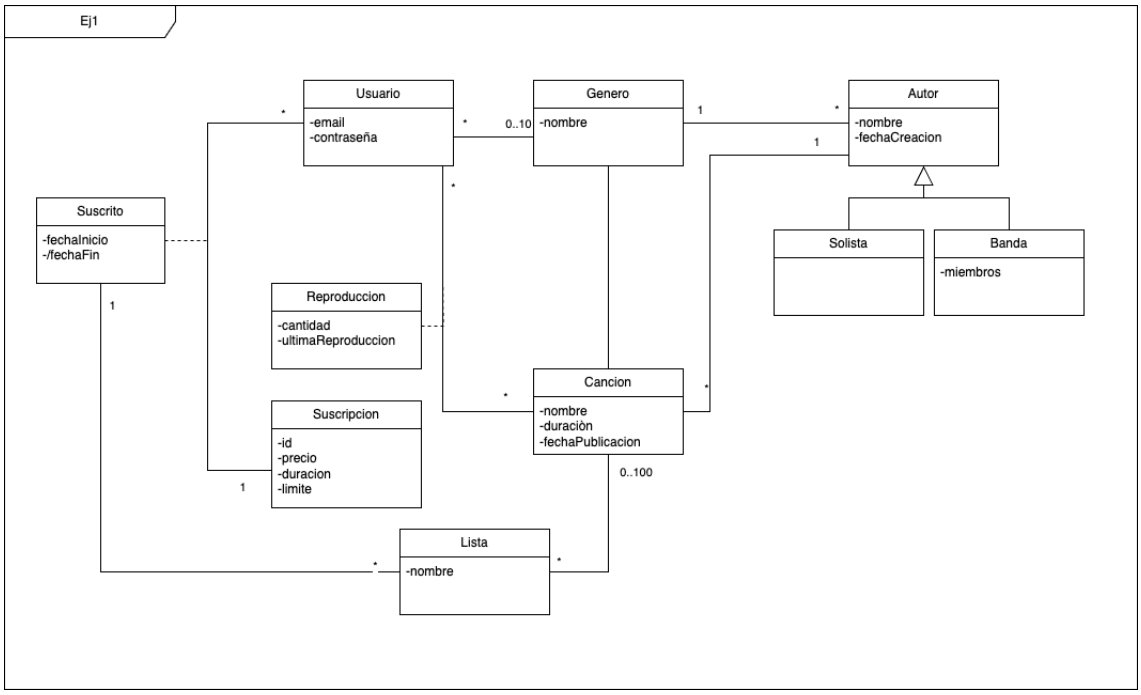
PARCIAL FINAL EDICIÓN 2023 - SOLUCIÓN

Por favor siga las siguientes indicaciones:

- Escriba con lápiz y de un solo lado de las hojas
- Escriba su nombre y número de documento en todas las hojas que entregue
- Numere las hojas e indique el total de hojas en la primera de ellas
- Recuerde entregar su número de parcial junto al parcial
- Está prohibido el uso de computadoras, tabletas o teléfonos durante el parcial

Problema 1 (25 puntos)

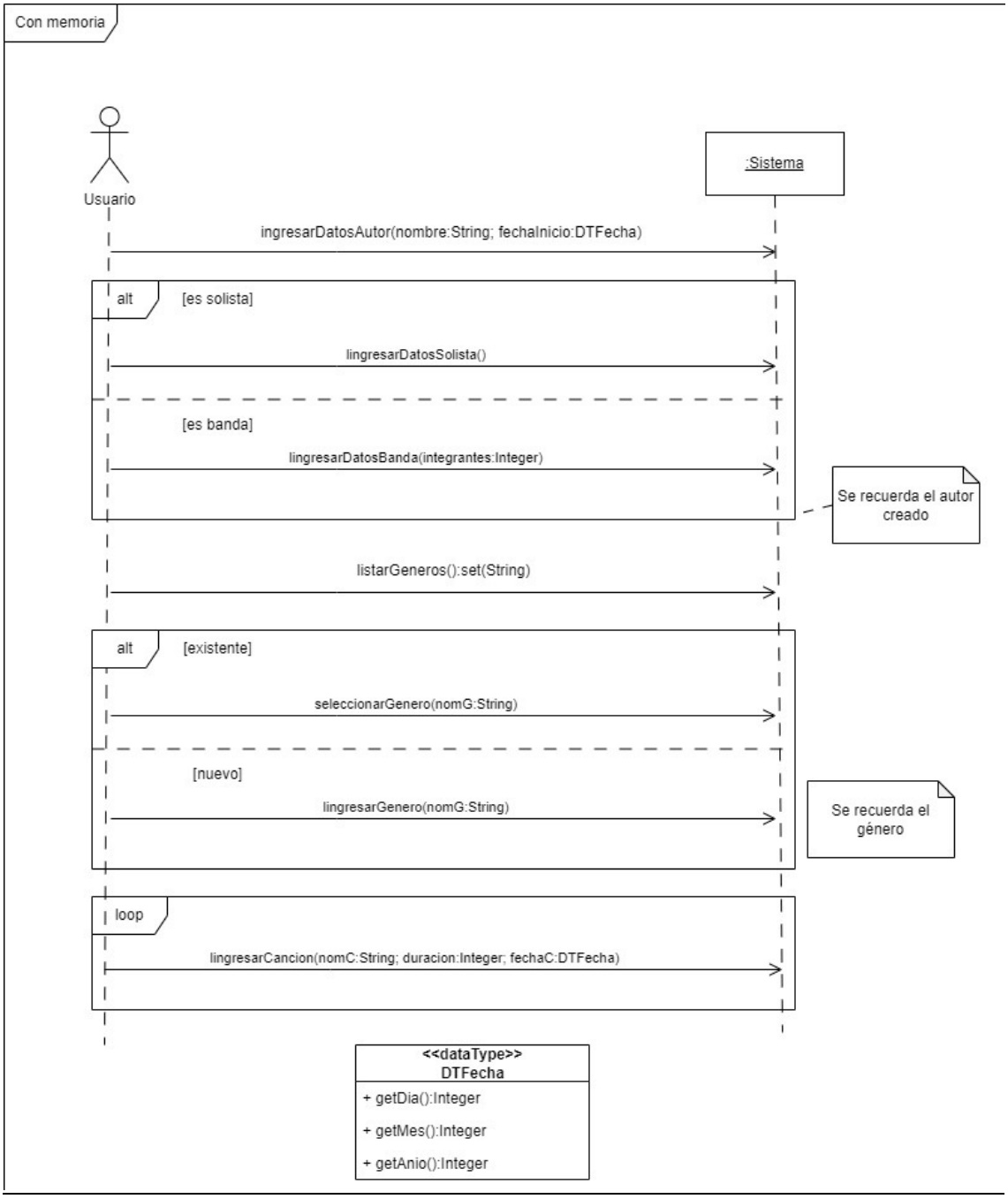
a. Realizar el Modelo de Dominio de la realidad planteada, incluyendo todas las restricciones que considere necesarias en lenguaje natural.



Restricciones:

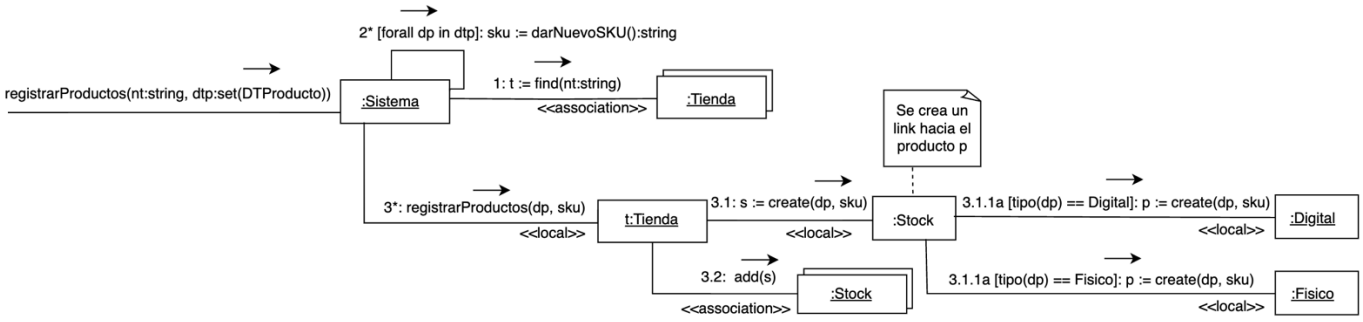
- El email de un Usuario es único.
- El nombre de un Género es único.
- El nombre de una Canción es único.
- El nombre de un Autor es único.
- El id de una Suscripción es único.
- La cantidad de listas asociadas a una suscripción no puede ser superior a limiteListas.
- La fecha de creación de una canción es superior a la fecha de inicio de su autor.

b. 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.

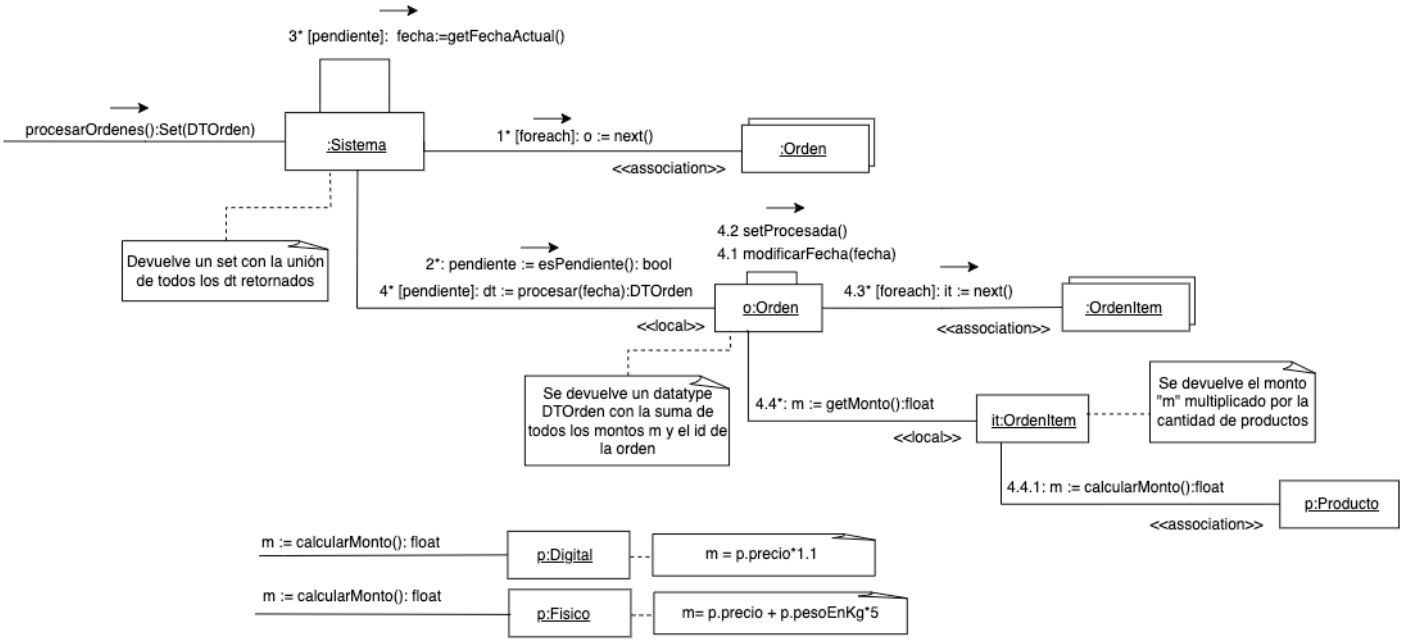


<u>Problema 2</u> (30 puntos)

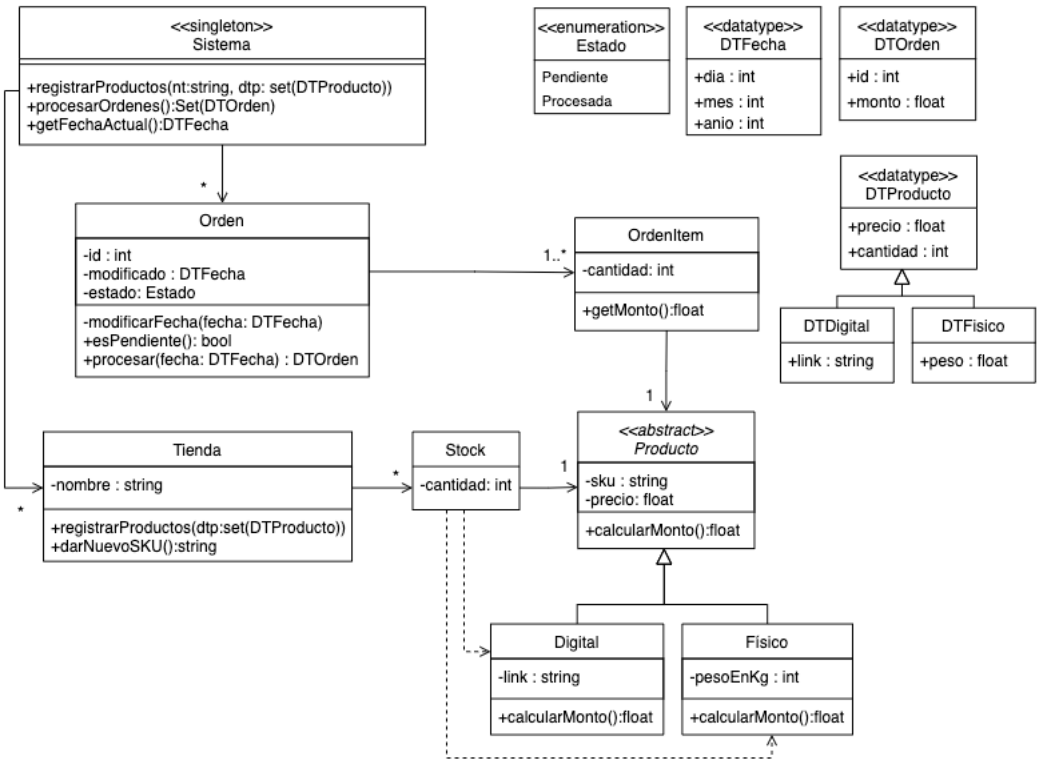
a. Realizar el Diagrama de Comunicación correspondiente a la siguiente operación del sistema, incluyendo visibilidades: `registrarProductos`.



b. Realizar el Diagrama de Comunicación correspondiente a la siguiente operación del sistema, incluyendo visibilidades: `procesarOrdenes`.



c. Realizar un único Diagrama de Clases de Diseño (DCD) resultante de las partes a) y b).



Problema 3 (30 puntos)

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

```
class Observer
{
public:
    virtual ~Observer();

    virtual void notificar(int curso, string desc) = 0;
};
```

b. Implementar en C++ el .h de la clase `Estudiante`.

```
class Estudiante : public Observer
{

private:
    string nickname;
    list<DTN *> notificaciones;
    list<Requisito *> pendientes;

public:
    Estudiante(string nickname);
    ~Estudiante();

    void notificar(int curso, string desc);
    void nuevoRequisito(Requisito * req);
};
```

c. Implementar en C++ el método de la operación `Requisito::cambioRequisito`.

```
void Requisito::cambioRequisito(string desc)
{
   this->desc = desc;
   list<Observer *>::iterator it;
   for (auto it = observadores.begin(); it != observadores.end(); ++it){
      (* it)->notificar(this->curso->getCodigo(),desc);
   }
}
```

d. Implementar en C++ el método de la operación `Estudiante::notificar`.

```
void Estudiante::notificar(int curso, string desc)
{
    //agrega nuevo requisito
    DTN * dtn = new DTN(curso,desc);
    notificaciones.push_back(dtn);
}
```

e. Implementar en C++ el método de la operación `Estudiante::nuevoRequisito`.

```
void Estudiante::nuevoRequisito(Requisito * req){
    //1. Si hay un requisitos de cursos previos a req, lanza excepción

    //Cursos previos al curso al que corresponde el requisito
    list<Curso *> previos = req->getCurso()->getPrevios();
    list<Requisito *>::iterator itreq;
    //Para cada requisito pendiente
    for (auto itreq = pendientes.begin(); itreq != pendientes.end(); ++itreq){
        //Obtengo el curso del requisito pendiente
        Curso * curso = (* itreq)->getCurso();

        //Busco el curso en los cursos previos del nuevo requisito
        //Se puede asumir que existe operación find en list/set
        list<Curso  *>::iterator  it  =  find(previos.begin(),  previos.end(),
curso);

        //Si encuentro el curso en la lista de cursos previos, error
        if (previos.end() != it)
            throw std::runtime_error("Aun existe previa");
    }

    //2. Si no hay requisitos de cursos previos
    //2.1 Se vincula al estudiante con el nuevo requisito
    pendientes.push_back(req);
    //2.2 Agrega al estudiante a la lista de observadores del requisito
    req->agregarObservador(this);
};
```