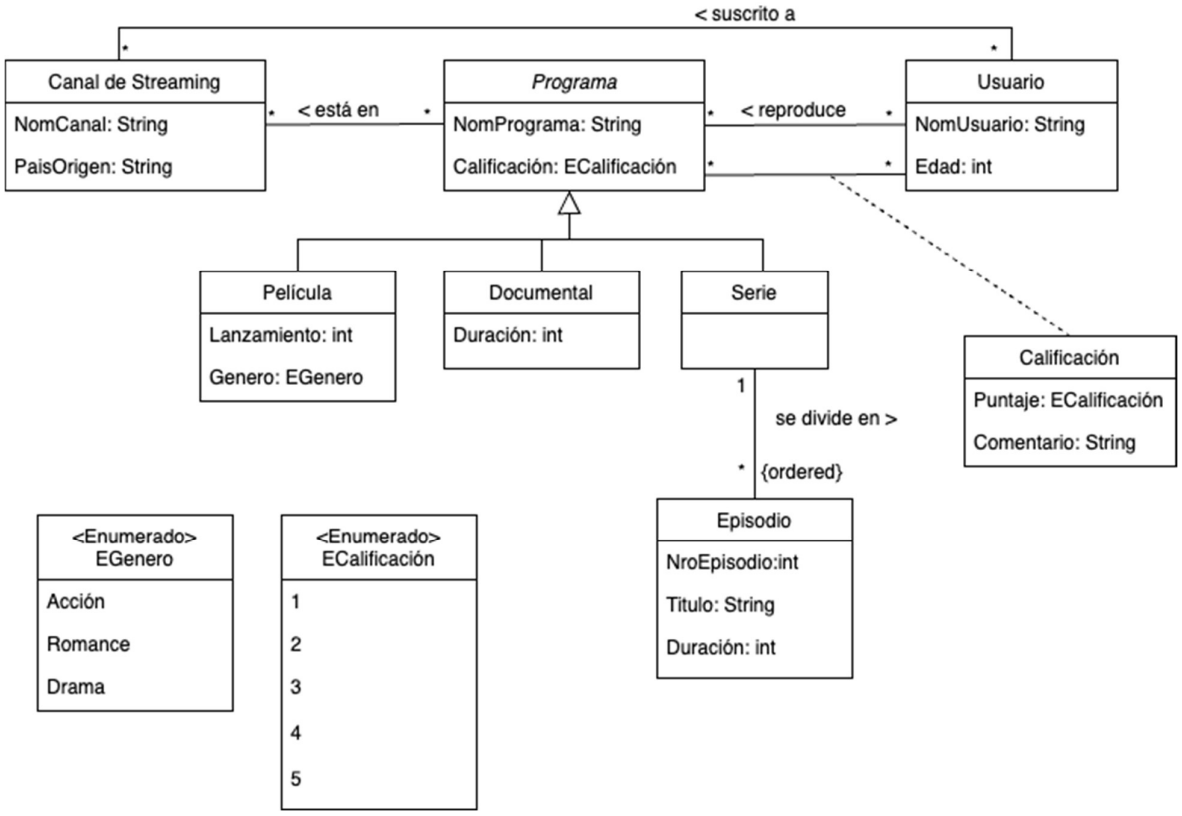
Programación 4

SOLUCIÓN EXAMEN DICIEMBRE 2024

Problema 1

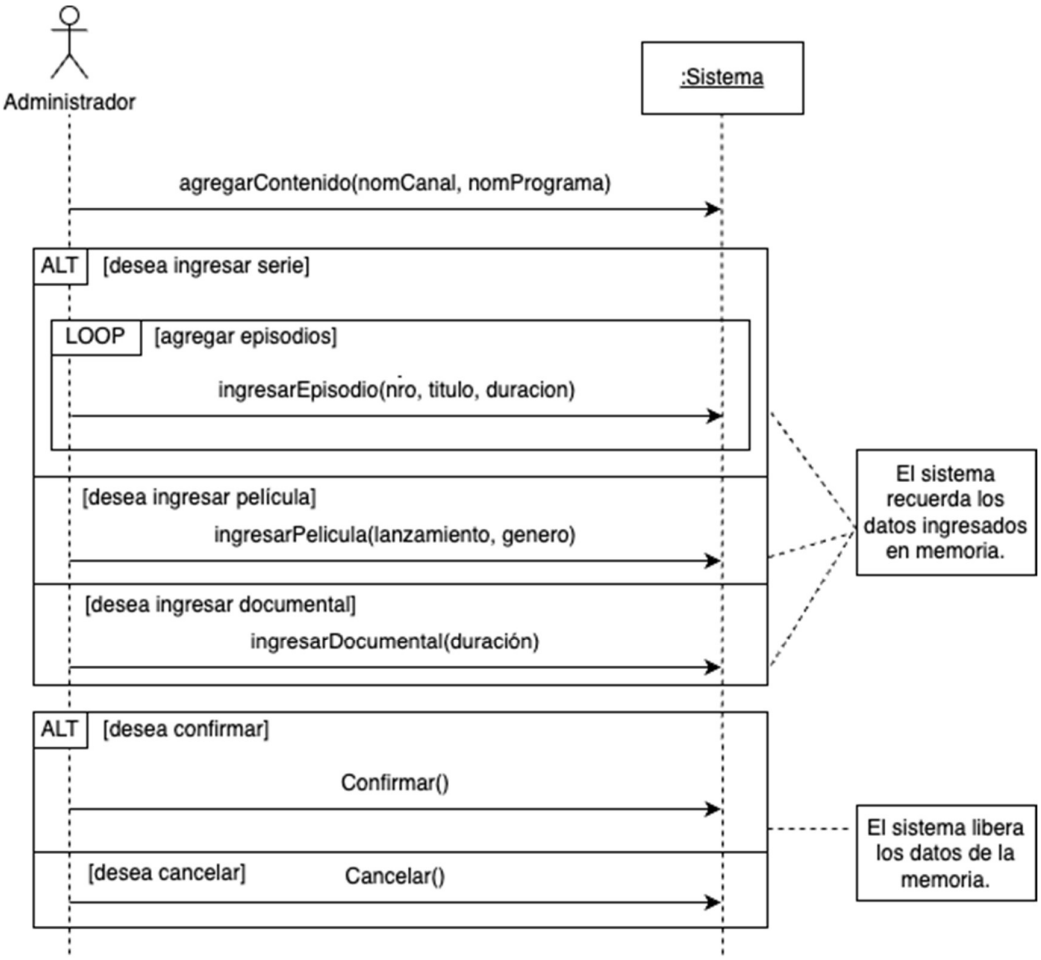
a)



Restricciones:

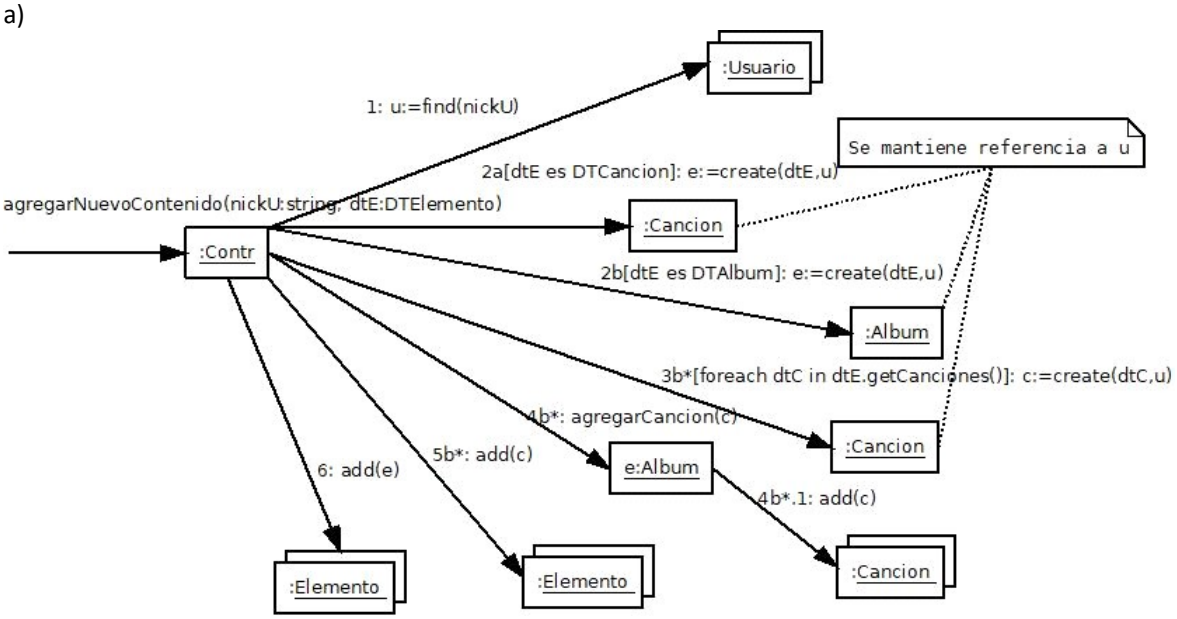
- Si existe un link 'reproduce' entre una instancia p de Programa y una instancia u de Usuario, también existe un link 'está entre p y una instancia s Canal de Streaming y otro link 'suscrito a' entre s y u.
- Si existe una instancia de Calificación entre una instancia p de Programa y una instancia u de Usuario, también existe un link 'reproduce' entre u y p.

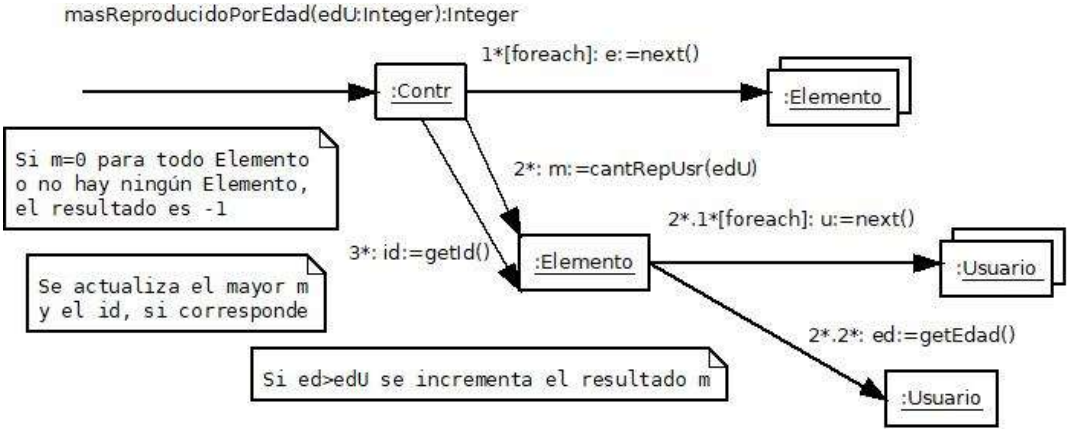
b)



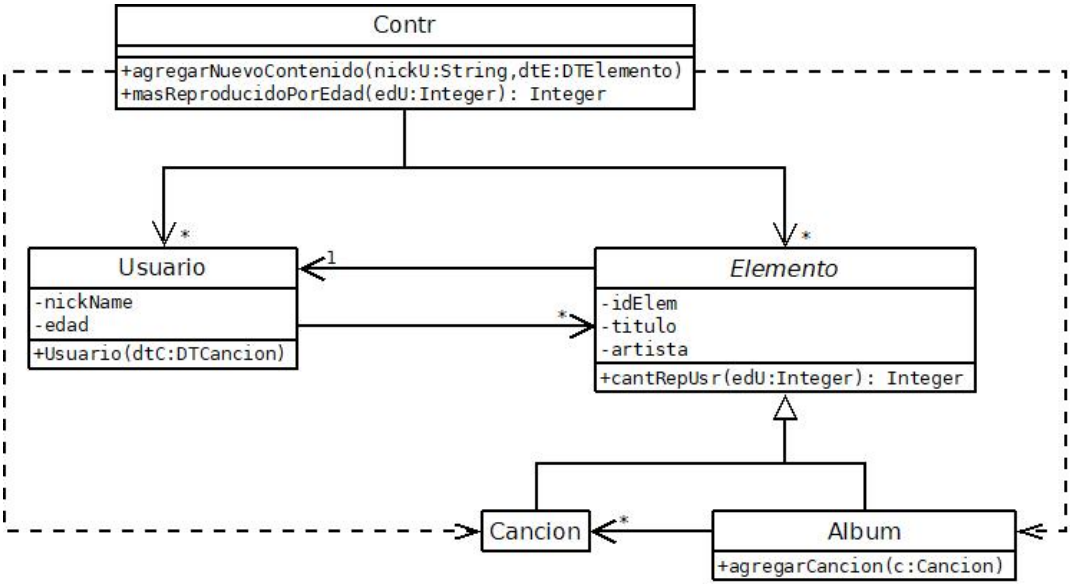
Problema 2

a)





b)



Problema 3

a) Se utiliza el patrón Composite, donde GestorTareas es el Cliente, Tarea es el Componente, TareaSimple es la Hoja y TareaCompuesta es el Compuesto. También se usa el patrón Singleton en GestorTareas.

b)

```
class GestorTareas {
private:
  int ultimoId;
  map<int, Tarea*> pendientes;
  map<int, Tarea*> realizadas;
  map<nombre, Usuario*> usuarios;
  static GestorTareas *instance;
  GestorTareas();
public:
  GestorTareas();
  static GestorTareas *getInstance();
  void crearTareaSimple(String);
  void crearTareaCompuesta(String, set<int>);
  void asignarResponsable(String, int);
  bool realizarTarea(int);
```

```
};

static GestorTareas::instance = NULL;

GestorTareas::GestorTareas() {
  ultimoId = 1;
}

static GestorTareas *GestorTareas::getInstance() {
    if (instance == null) {
      instance = new GestorTareas();
    }
    return instance;
  }

void GestorTareas::crearTareaSimple(String descripcion) {
  pendientes[ultimoId++] = new TareaSimple(descripcion);
}

void  GestorTareas::crearTareaCompuesta(String  descripcion,  set<int>
ids) {
  map<int, Tarea*> subtareas;
  for(set<int>::iterator it = ids.begin(); it != ids.end(); it++) {
    subtareas[*it] = pendientes[*it];
  }
  pendientes[ultimoId++] =
                         new  TareaCompuesta(descripcion, subtareas);
}

void GestorTareas::asignarResponsable(String nombre, int idTarea) {
  usuarios[nombre]->getTareas()[idTarea] = pendientes[idTarea];
  pendientes[idTarea]->setResponsable(usuarios[nombre]);
}

bool GestorTareas::realizarTarea(int idTarea) {
  Tarea* tarea = pendientes[idTarea];
  if (tarea->realizar()) {
    pendientes.erase(idTarea);
    realizadas[idTarea] = tarea;
    return true;
  } else {
    return false;
  }
}

class Tarea {
private:
  String descripcion;
  Usuario* responsable;
  EstadoTarea estado;
public:
  Tarea(String descripcion);
  virtual bool realizar() = 0;
}

Tarea::Tarea(String descripcion) { this->descripcion = descripcion; }

class TareaCompuesta : public Tarea {
private:
  map<int, Tarea*> subtareas;
public:
```

```
  TareaCompuesta(String, map<int, Tarea*>) ;
  bool realizar();

TareaCompuesta::TareaCompuesta(String descripcion, map<int, Tarea*>
subtareas) : Tarea(descripcion) {
  this->subtareas = subtareas;
}

bool TareaCompuesta::realizar() {
  map<int, Tarea*>::iterator it = subtareas.begin();
  while(it != subtareas.end() && it->second->getEstado() == Realizado)
    it++;
  if (it == subtareas.end()) {
    setEstado(Realizado);
    return true;
  } else {
    return false;
  }
}
```