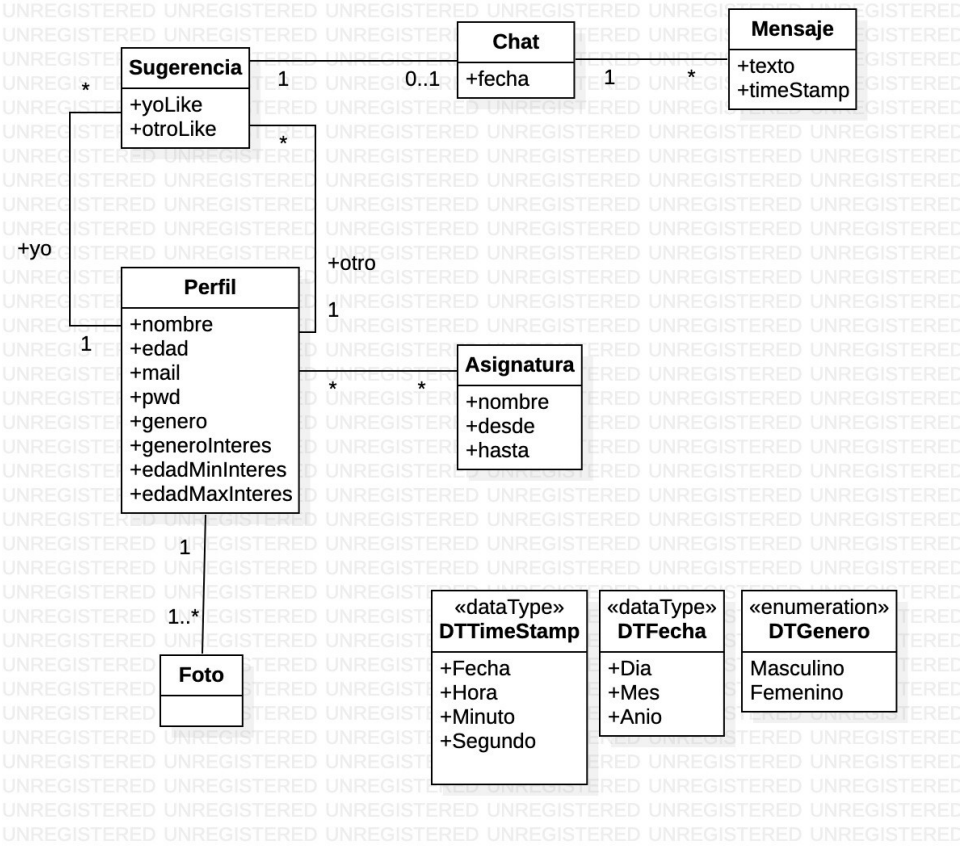
Programación 4

SOLUCIÓN EXAMEN FEBRERO 2020

Problema 1:

a)

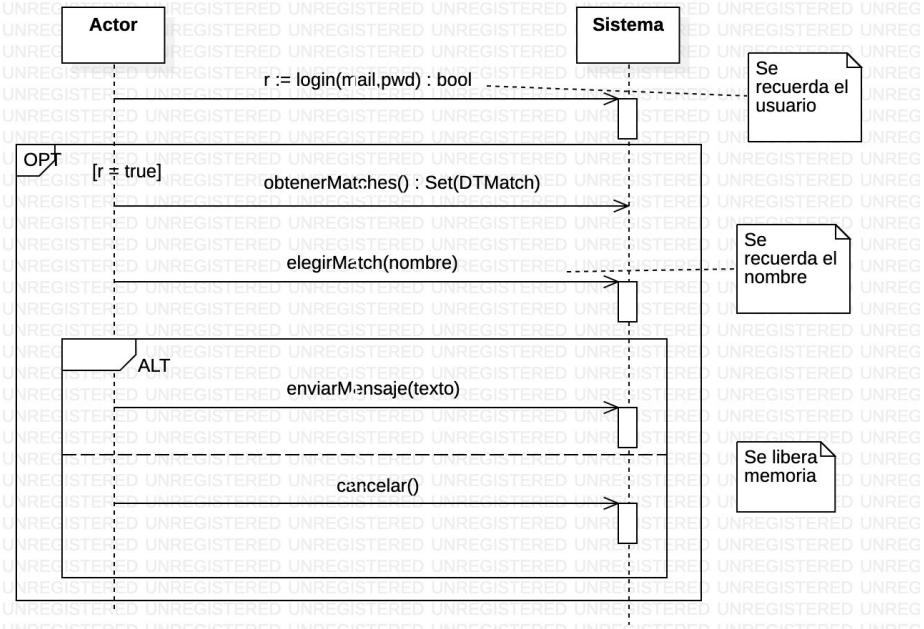


Restricciones:

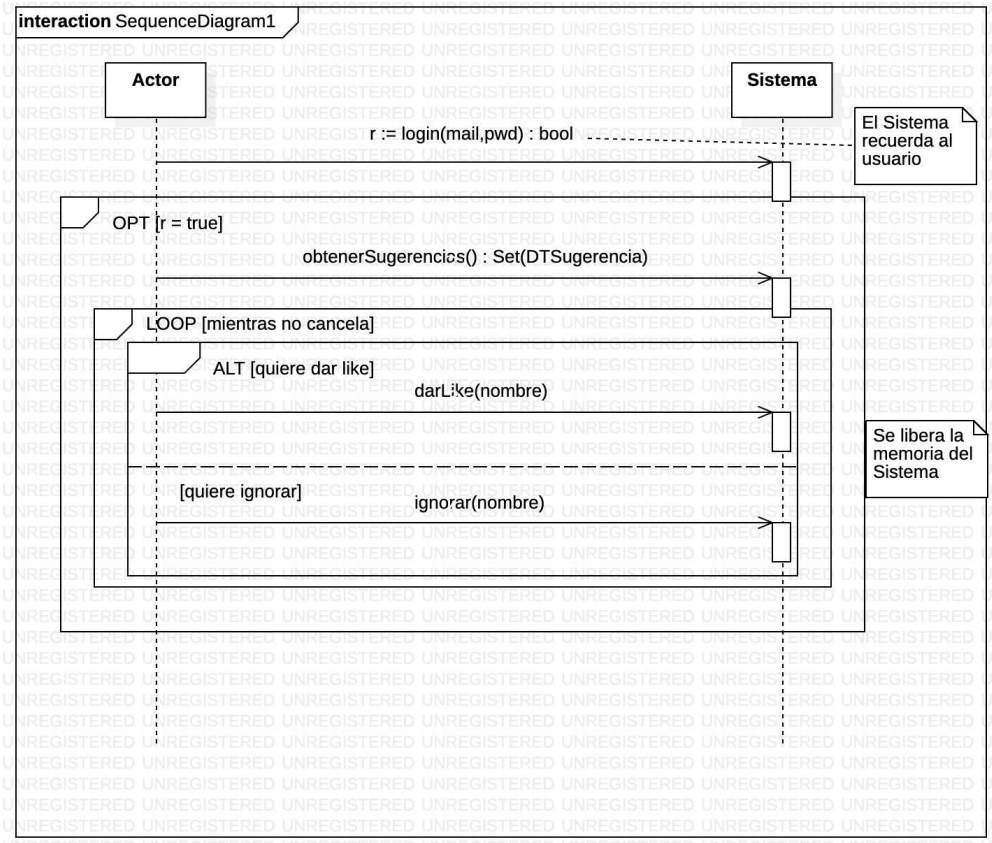
- El mail identifica al perfil.
- Un perfil no puede recibir una sugerencia de sí mismo.
- Toda sugerencia a un perfil debe respetar que su edad este dentro del rango de interés así como su género de interés.
- Toda sugerencia (asociada con 2 perfiles P1 y P2) debe respetar que el conjunto de las asignaturas asociadas a P1 intersección el conjunto de las asignaturas asociadas a P2 es distinto de vacío.
- Existe un chat asociado a una sugerencia solo si ambos atributos de sugerencia son TRUE (ambos dieron like).
- El time-stamp de cada mensaje > a la fecha en que se dio el match (fecha del Chat).

b)

DSS con memoria para Enviar Mensaje:

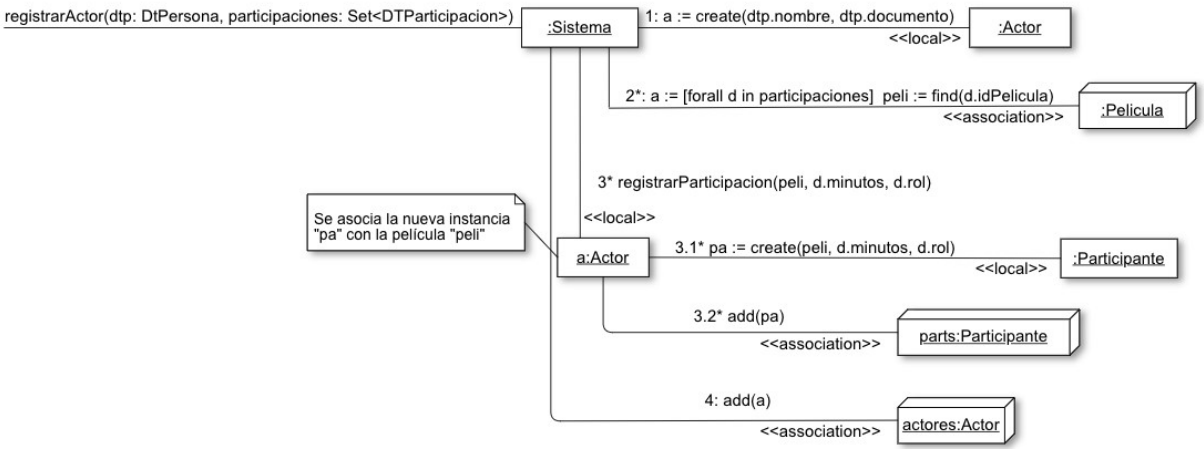


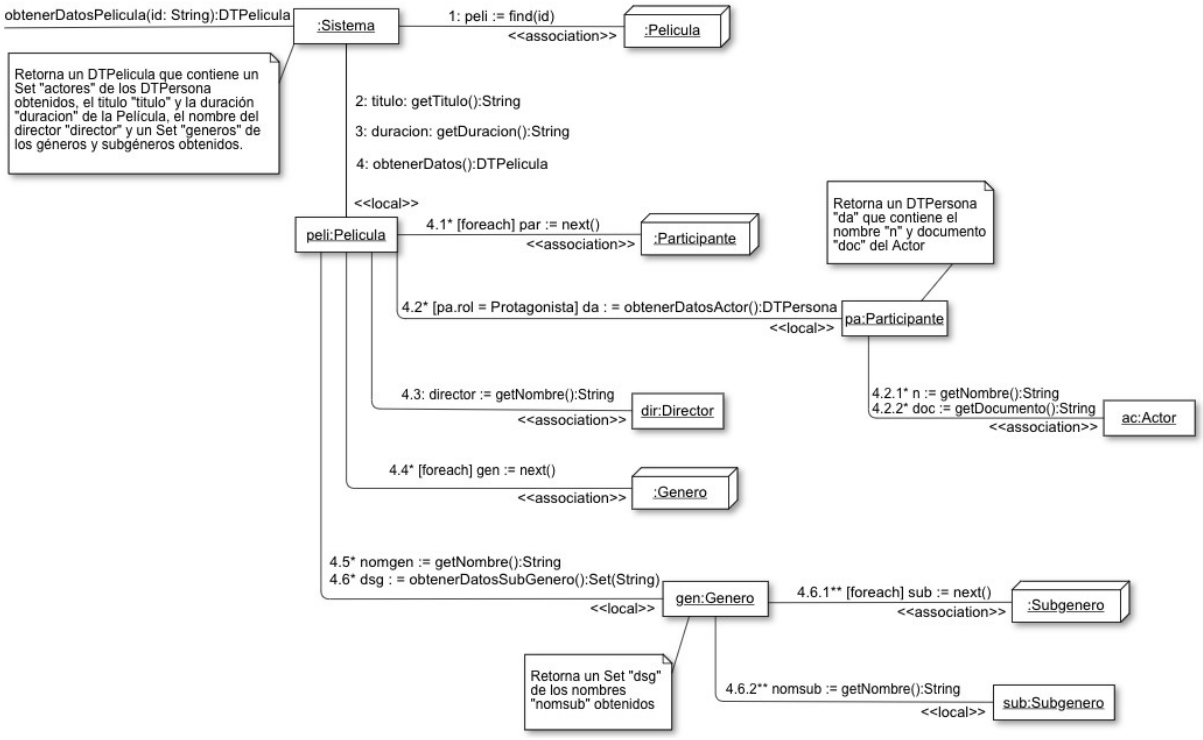
DSS con memoria para Revisar Sugerencias:



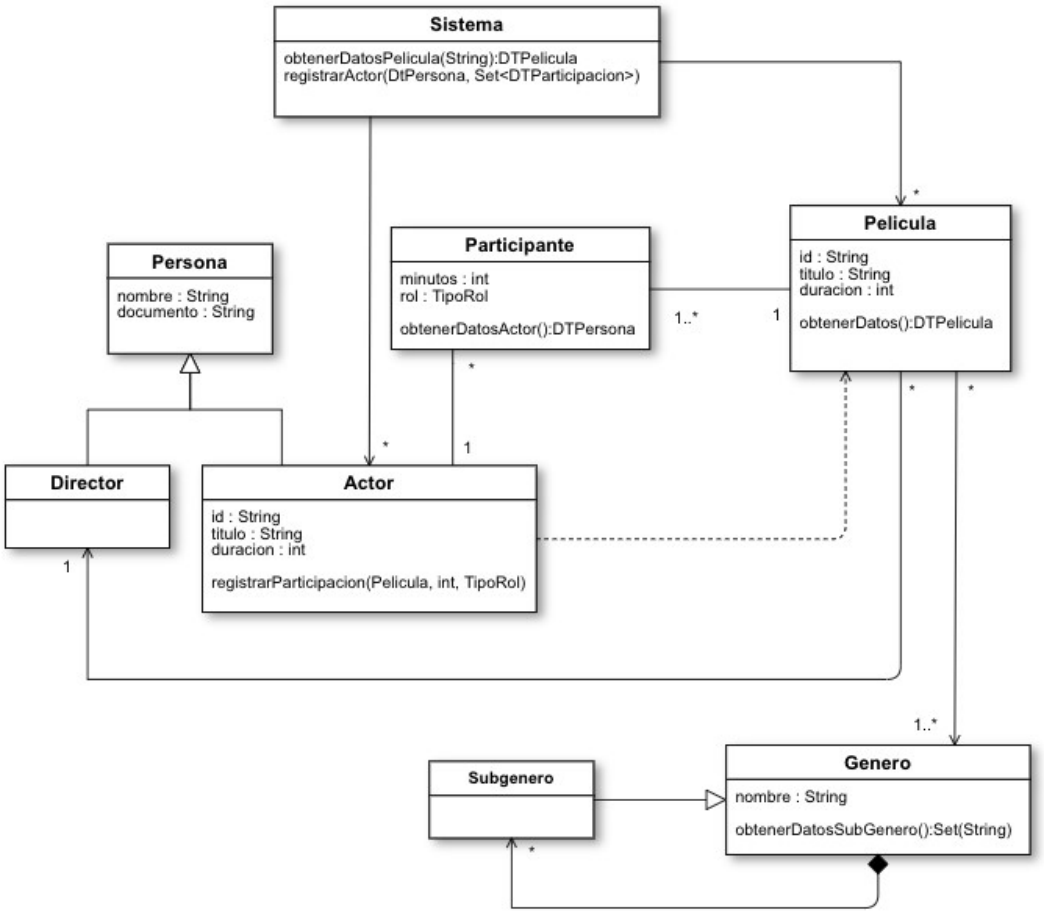
Problema 2

a)


registrarActor(dtp: DtPersona, participaciones: Set<DTParticipacion>)
:Sistema
1: a := create(dtp.nombre, dtp.documento)
<<local>>
:Actor
2*: a := [forall d in participaciones] peli := find(d.idPelicula)
<<association>>
:Pelicula
3* registrarParticipacion(peli, d.minutos, d.rol)
<<local>>
Se asocia la nueva instancia "pa" con la película "peli"
a:Actor
3.1* pa := create(peli, d.minutos, d.rol)
<<local>>
:Participante
3.2* add(pa)
<<association>>
parts:Participante
4: add(a)
<<association>>
actores:Actor


obtenerDatosPelicula(id: String):DTPelicula
:Sistema
1: peli := find(id)
<<association>>
:Pelicula
Retorna un DTPelicula que contiene un Set "actores" de los DTPersona obtenidos, el titulo "titulo" y la duración "duracion" de la Película, el nombre del director "director" y un Set "generos" de los géneros y subgéneros obtenidos.
2: titulo: getTitulo():String
3: duracion: getDuracion():String
4: obtenerDatos():DTPelicula
<<local>>
peli:Pelicula
4.1* [foreach] par := next()
<<association>>
:Participante
Retorna un DTPersona "da" que contiene el nombre "n" y documento "doc" del Actor
4.2* [pa.rol = Protagonista] da : = obtenerDatosActor():DTPersona
<<local>>
pa:Participante
4.3: director := getNombre():String
<<association>>
dir:Director
4.2.1* n := getNombre():String
4.2.2* doc := getDocumento():String
<<association>>
ac:Actor
4.4* [foreach] gen := next()
<<association>>
:Genero
4.5* nomgen := getNombre():String
4.6* dsg : = obtenerDatosSubGenero():Set(String)
<<local>>
gen:Genero
4.6.1** [foreach] sub := next()
<<association>>
:Subgenero
Retorna un Set "dsg" de los nombres "nomsub" obtenidos
4.6.2** nomsub := getNombre():String
<<local>>
sub:Subgenero

b)



Problema 3:

a) Se utiliza Composite, cuyo problema tipo es: "Componer objetos en estructuras arborescentes para representar jerarquías de objetos compuestos y tratar uniformemente los mismos."

Clase	Rol
ControladorOperacionesGeometricas	Cliente
Figura	Componente
Circulo	Hoja
Cuadrado	Hoja
Rara	Compuesto

b)

ControladorOperacionesGeometricas.h

```
class ControladorOperacionesGeometricas {
	private:
		map<int, Figura*> figuras;
	public:
		float calcularAreaFigura(int i);
}
```

ControladorOperacionesGeometricas.cpp

```
float ControladorOperacionesGeometricas::calcularAreaFigura(int i){
	return figuras[i]->calcularArea();
}
```

Figura.h

```
class Figura {
	private:
		int id;
	public:
		Figura(int id);
		virtual ~Figura();
		virtual float calcularArea() = 0;
}
```

Figura.cpp

```
Figura::Figura(int id){
	this->id = id;
}

Figura::~Figura(){}
```

Cuadrado.h

```
class Cuadrado: public Figura {
	private:
		float lado;
	public:
		Cuadrado(int id, float lado);
		~Cuadrado();
		float calcularArea();
}
```

Cuadrado.cpp

```
Cuadrado::Cuadrado(int id, float lado):Figura(id){
	this->lado = lado;
}

Cuadrado::~Cuadrado(){}

float Cuadrado::calcularArea(){
	return lado * lado;
}
```

Rara.h

```
class Rara: public Figura {
	private:
		map<int, Figura*> componentes;
	public:
		Rara(int id);
		~Rara();
		float calcularArea();
		void agregarFigura(Figura* f);
		void removerFigura(int i);
}
```

Rara.cpp

```
Rara::Rara(int id):Figura(id){}

Rara::~Rara(){
	for(map<int, Figura*>::iterator it = componentes.begin(); it !=
componentes.end(); it++) {
		delete it->second;
	}
}

float Rara::calcularArea(){
	float suma = 0;
	for(map<int, Figura*>::iterator it = componentes.begin(); it !=
componentes.end(); it++) {
		suma = suma + (it->second)->calcularArea();
	}
	return suma;
}

void Rara::agregarFigura(Figura* f){
	componentes[f->getId()] = f;
}

void Rara::removerFigura(int i){
	componentes.erase(i);
}
```

c) Se utiliza Fábrica, cuyo problema tipo es: “Permitir visibilidad desde un consumidor hacia proveedores concretos sin que el consumidor quede acoplado directamente a éstos.”

Clase	Rol
Diagramacion	Consumidor
Fabrica	Fábrica
IControladorOperacionesGeometricas	IProveedor
ControladorOperacionesGeometricas	Proveedor

Fabrica.h

```
class FabricaAerea {
	public:
		IControladorOperacionesGeometricas*
			getIControladorOperacionesGeometricas();
}
```

Fabrica.cpp

```
IControladorOperacionesGeometricas*
Fabrica::getIControladorOperacionesGeometricas(){
      return ControladorOperacionesGeometricas();
}
```

IControladorOperacionesGeometricas.h

```
class IControladorOperacionesGeometricas {
      virtual float calcularAreaFigura() = 0;
}
```

ControladorOperacionesGeometricas.h

```
class ControladorOperacionesGeometricas : public
IControladorOperacionesGeometricas {
      private:
            map<int, Figura*> figuras;
      public:
            ControladorOperacionesGeometricas();
            ~ControladorOperacionesGeometricas();
            float calcularAreaFigura(int i);
}
```

ControladorOperacionesGeometricas.cpp

No tiene cambios respecto a la parte anterior.