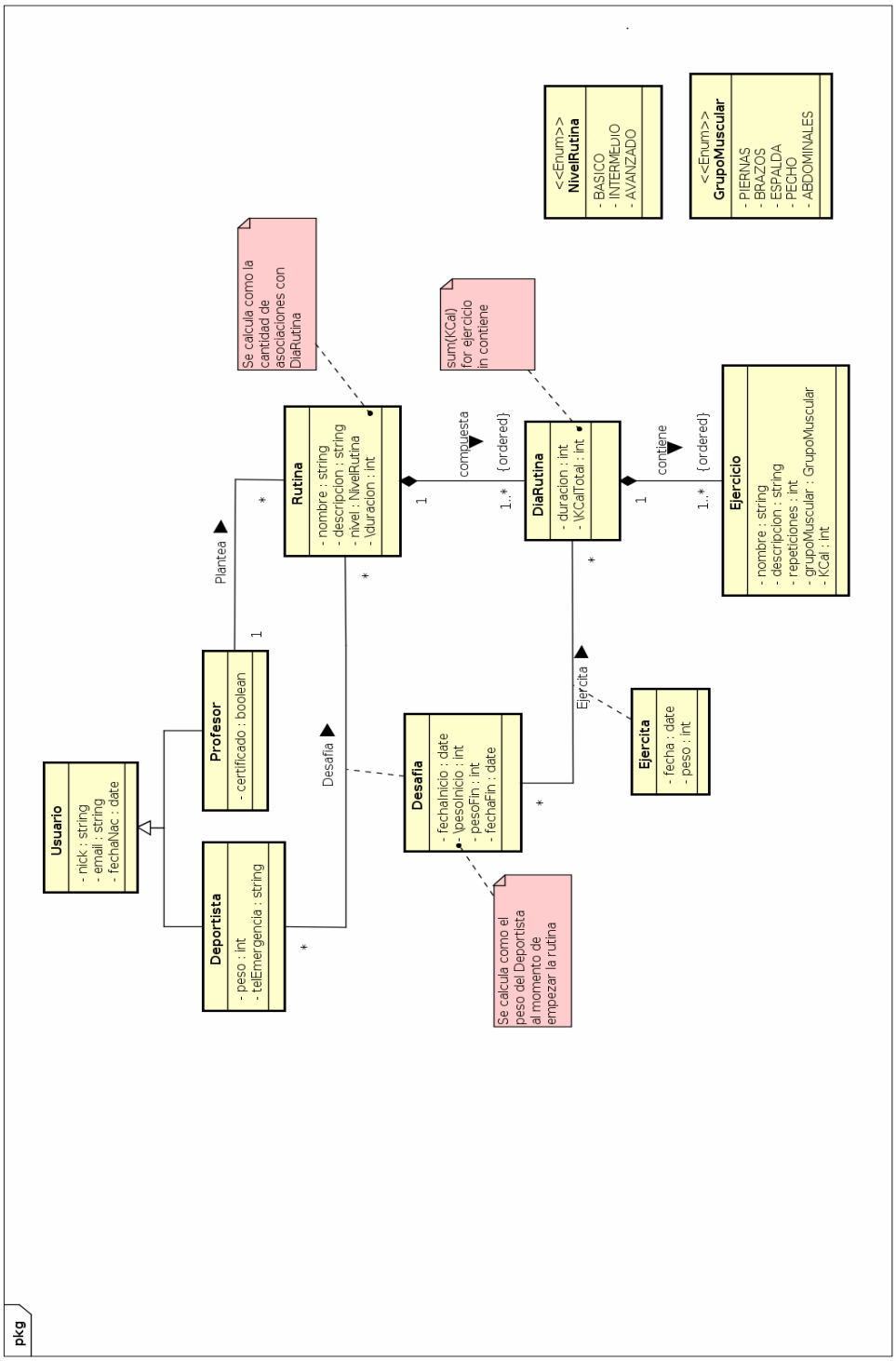
Programación 4

SOLUCION DICIEMBRE 2018

Problema 1

i)



Restricciones:

1) Unicidad

- El nick de un usuario es único.

2) Dominio de atributos:

- El peso de un Usuario es un entero positivo.
- La duración de una Rutina es un entero positivo.
- La duración de un DiaRutina es un entero positivo.
- Las repeticiones de un Ejercicio es un entero positivo.
- Las KCal de un Ejercicio es un entero positivo.
- El peso en Ejercita es un entero positivo.

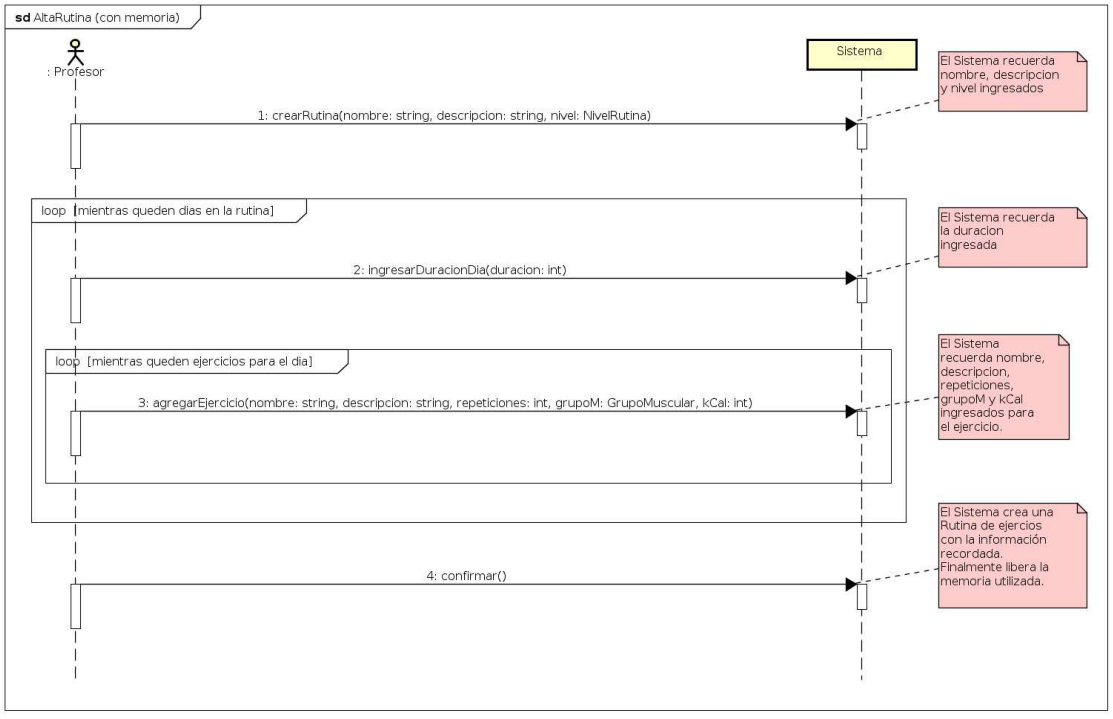
3) Atributos calculados:

- KCal en DiaRutina se calcula como la suma de KCal de todos los Ejercicios que DiaRutina contiene.
- Duración en Rutina se calcula como la cantidad de DiaRutina por la cual está compuesta.
- PesoInicio en Desafia se calcula como el valor de peso del Deportista en el momento en que desafía la Rutina.

4) Integridad circular:

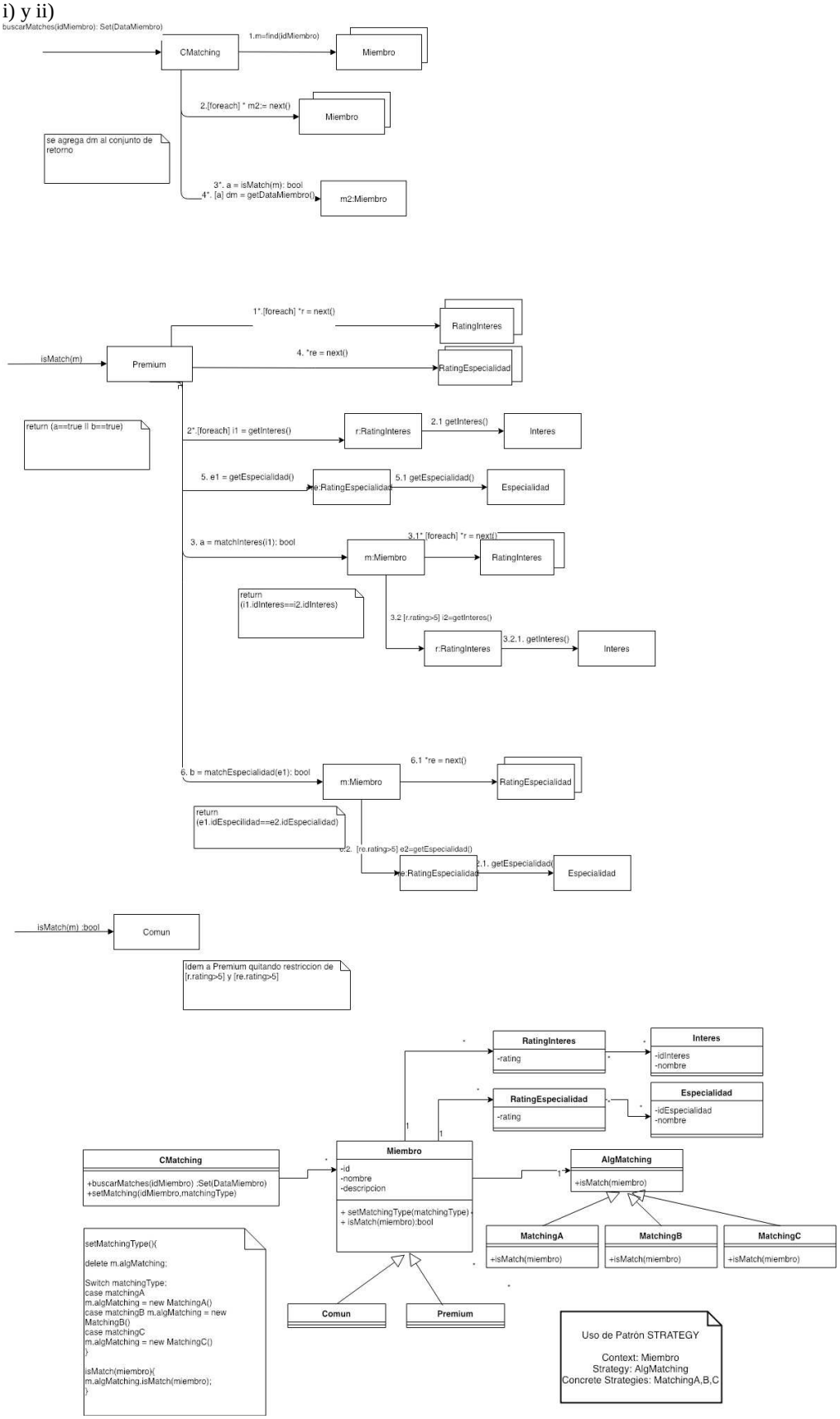
- Dado un Deportista, un Rutina y el Desafio entre estos dos, se Ejercita solamente DiaRutina de la cual la Rutina está compuesta.

ii)



Problema 2

i) y ii)



Problema 3

ControladorEventos.h

```
class ControladorEventos {
private:
	static ControladorEventos* instancia;
	ControladorEventos();
	map<string, Evento*> eventos;
	map<string, Usuario*> usuarios;
public:
	void suscribirUsuarioEvento(string idUsuario, string nombreEvento);
	void notificarOcurrenciaEvento(string nombreEvento, string datos);
	static ControladorEventos* darInstancia();
}
```

Evento.h

```
class Evento {
private:
	string nombre;
	string descripcion;
	set<Suscriptor*> suscriptores;
public:
	Evento(string nombre, string descripcion);
	void agregarSuscriptor(Suscriptor* s);
	void borrarSuscriptor(Suscriptor* s);
	void notificarSuscriptores(string datos);
}
```

Suscriptor.h

```
class Suscriptor {
private:
	string id;
public:
	Suscriptor(string id);
	virtual void notificar(string datos) = 0;
	virtual ~Suscriptor();
}
```

Usuario.h

```
class Usuario : public Suscriptor {
private:
	string nombre;
public:
	Usuario(string id, string nombre);
	void notificar(string datos);
}
```

Logger.h

```
class Logger : public Suscriptor {
private:
	set<InfoEvento*> historico;
public:
	Logger(string id);
	void notificar(string datos);
}
```

InfoEvento.h

```
class InfoEvento {
private:
	DataFecha fecha;
	string datos;
public:
	InfoEvento(DataFecha fecha, string datos);
```

```
}
```

ControladorEventos.cpp

```
ControladorEventos* ControladorEventos::instancia = NULL;

void ControladorEventos::suscribirUsuarioEvento(string idUsuario, string
nombreEvento){
	Usuario* u = usuarios[idUsuario];
	eventos[nombreEvento]->agregarSucriptor(u);
}

void ControladorEventos::notificarOcurrenciaEvento(string nombreEvento,
string datos){
	eventos[nombreEvento]->notificarSuscriptores(datos);
}

ControladorEventos* ControladorEventos::getInstancia(){
	if (instancia == NULL){
		instancia = new ControladorEventos();
	}
	return instancia;
}
```

Evento.cpp

```
Evento::Evento(string nombre, string descripcion)
	: nombre(nombre), descripcion(descripcion) {
}

void Evento::agregarSuscriptor(Suscriptor* s){
	suscriptores.insert(s);
}

void Evento::borrarSuscriptor(Suscriptor* s){
	suscriptores.erase(s);
}

void Evento::notificarSuscriptores(string datos){
	set<Suscriptor*>::iterator it;
	for (it = suscriptores.begin(); it!= suscriptores.end(); it++){
		(*it)->notificar(datos);
	}
}
```

Suscriptor.cpp

```
Suscriptor::Suscriptor(string id)
	:id(id) {
}

Suscriptor:: ~Suscriptor() {}
```

Usuario.cpp

```
Usuario::Usuario(string id, string nombre)
	: Suscriptor(id), nombre(nombre){

}

void Usuario::notificar(string datos){}
```

Logger.cpp

```
Logger::Logger(strings id)
	:Suscriptor(id) {
}
```

```
void Logger::notificar(string datos){
      DataFecha fecha = Sistema::obtenerFecha();
      InfoEvento* info = new InfoEvento(fecha, datos);
      historico.insert(info);
}
```

InfoEvento.cpp

```
InfoEvento::InfoEvento(Datafecha fecha, string datos)
      : fecha(fecha), datos(datos) {

}
```