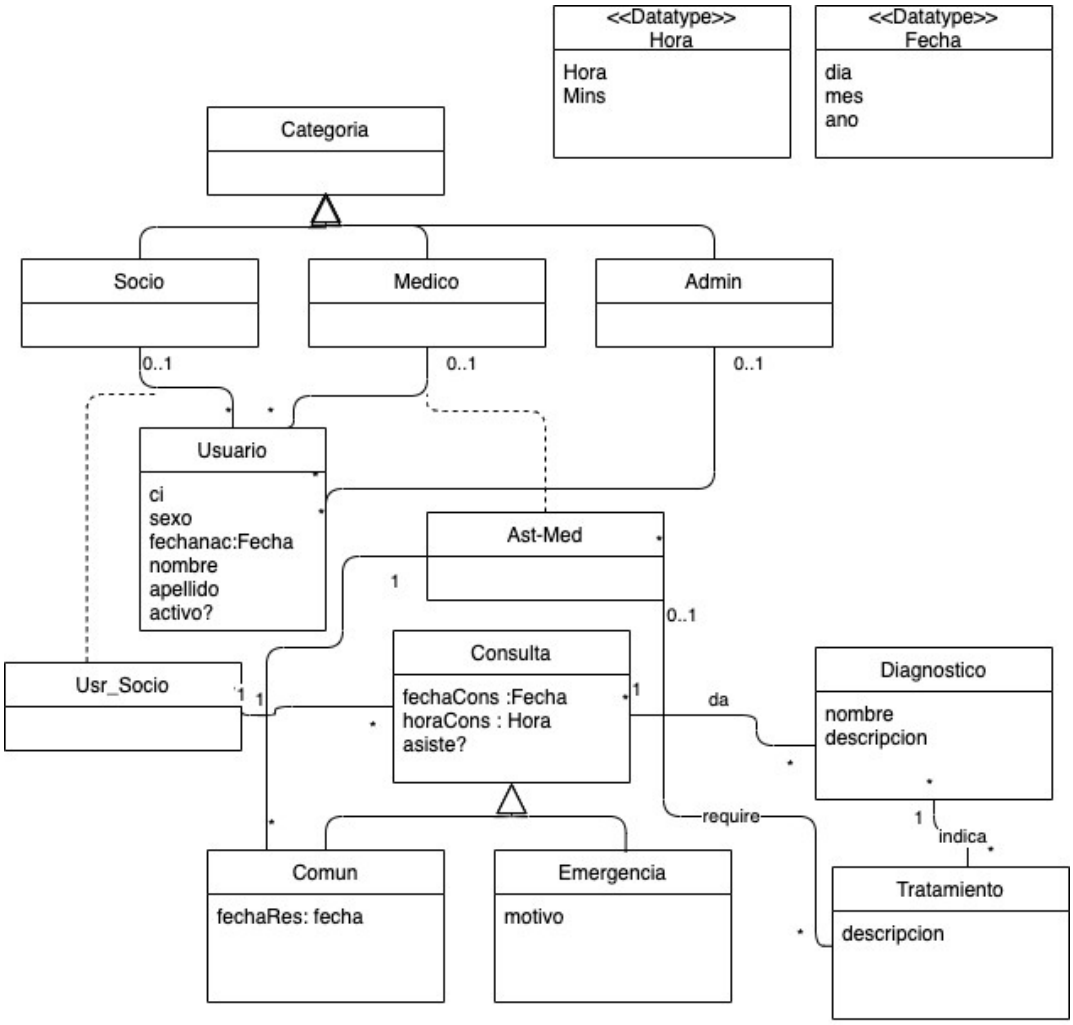
Programación 4

SOLUCIÓN EXAMEN JULIO 2020

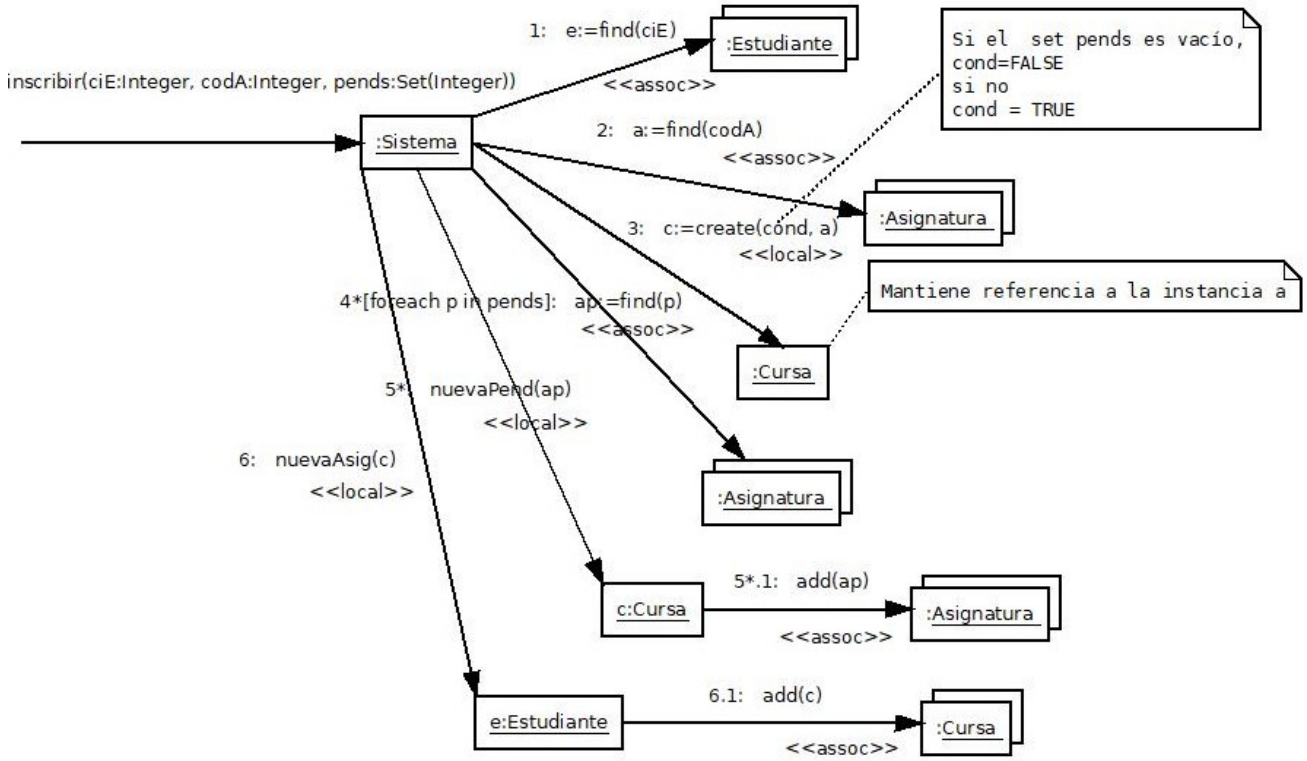
Problema 1 (30 puntos)

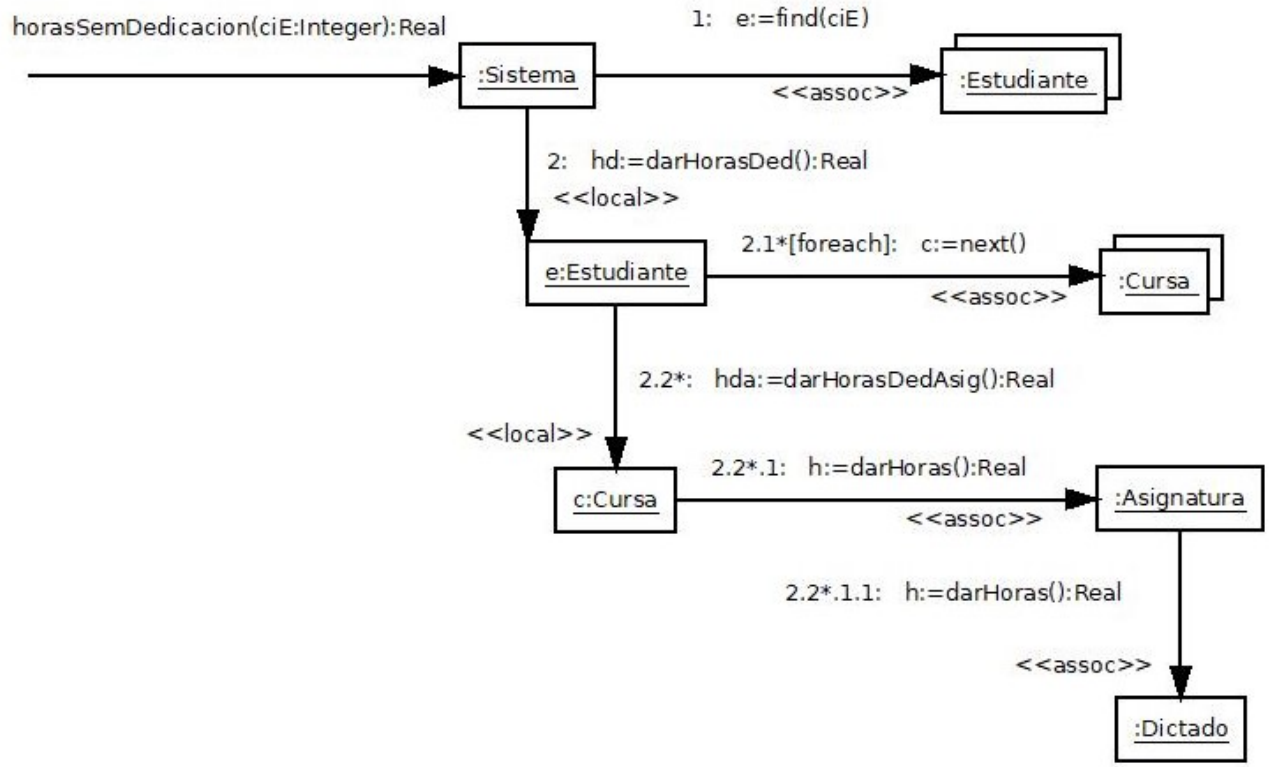


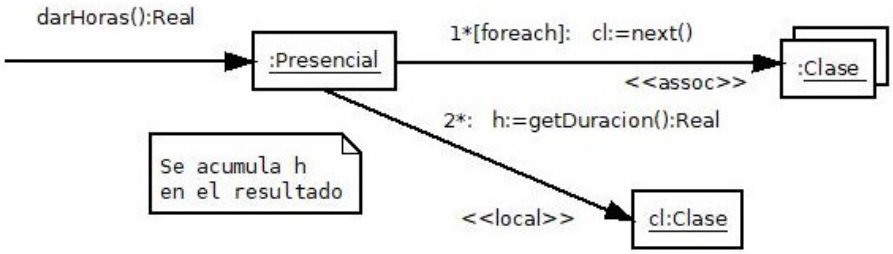
Restricciones:

- ci identifica al usuario
- ci solo contiene números
- un usuario pertenece al menos a una categoría
- un usuario perteneciente a las categorías socio y médico no puede hacer una consulta donde el médico de la consulta sea él mismo

Problema 2 (35 puntos)

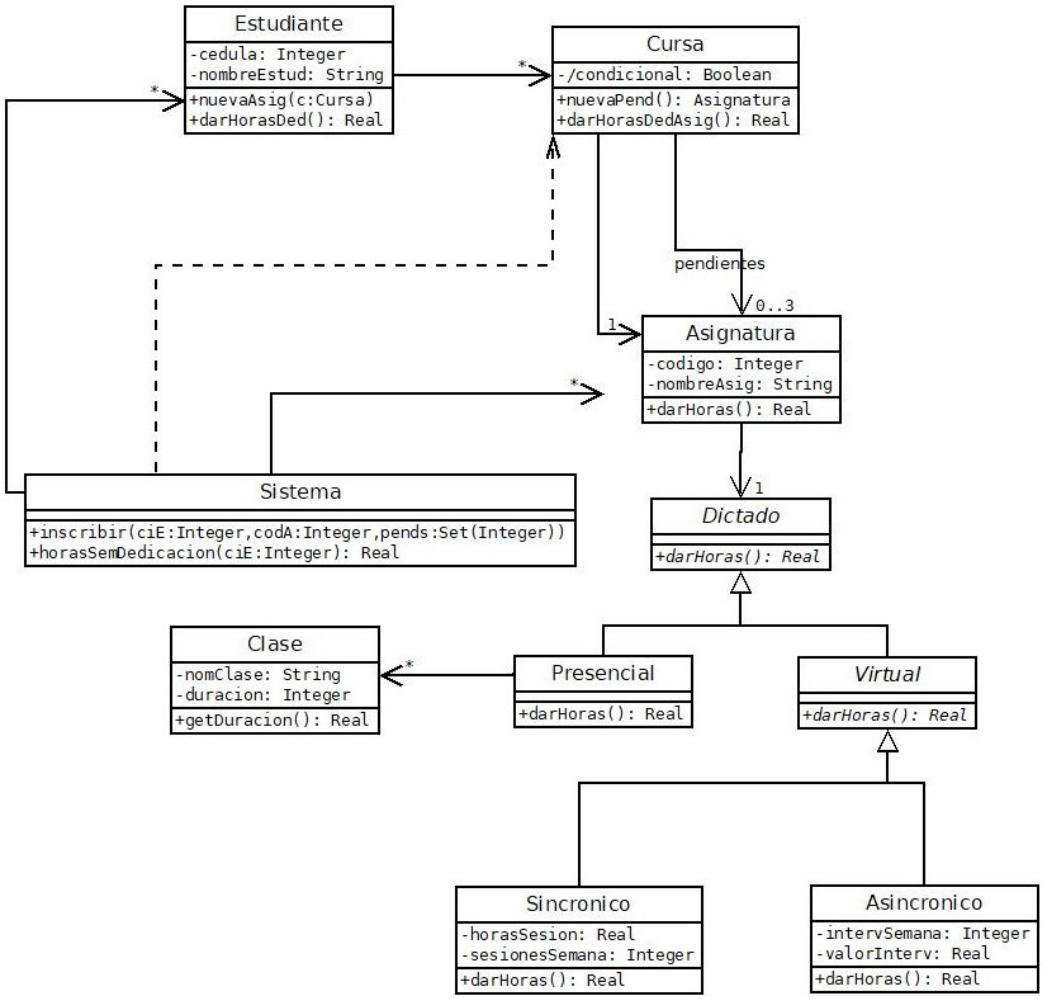





darHoras():Real
:Presencial
1*[foreach]: cl:=next()
<<assoc>>
:Clase
2*: h:=getDuracion():Real
Se acumula h
en el resultado
<<local>>
cl:Clase


darHoras():Real
:Sincronico
Se retorna
horasSesion*sesionesSemana


darHoras():Real
:Asincronico
Se retorna
intervSemana*valorInterv



Problema 3 (35 puntos)

a) Se utiliza Strategy, cuyo problema tipo es: "Permitir variar la ejecución de parte de un algoritmo sin afectar la clase que lo ejecuta".

Clase	Rol
Usuario	Contexto
Filtro	Estrategia
FiltroEdad	EstrategiaConcreta
FiltroRegionPais	EstrategiaConcreta

b)

ControladorContenidosUsuarios.h

```
class ControladorContenidosUsuarios {
private:
      map<int, Contenido*> contenidos;
      map<int, Usuario*> usuarios;
      map<int, Filtro*> filtros;
public:
      void altaContenido(string url, EnumTipo tipo);
      void agregarFiltroUsuario(int idFiltro, int idUsuario);
      void mostrarContenido(int idContenido);
}
```

ControladorContenidosUsuarios.cpp

```
int ControladorContenidosUsuarios::altaContenido(string url, EnumTipo
tipo){
      int idContenido = contenidos.size();
      Contenido c = new Contenido(idContenido, url, tipo);
      contenidos[c->getIdContenido()] = c;
      return idContenido;
}

ControladorContenidosUsuarios::agregarFiltroUsuario(int idFiltro, int
idUsuario){
      Usuarios[idUsuario]->agregarFiltro(filtros[idFiltro]);
}

ControladorContenidosUsuarios::mostrarContenido(int idContenido){
      Contenido* c = contenidos[idContenido];
      for(map<int, Usuario*>::iterator it = usuarios.begin(); it !=
      usuarios.end(); it++) {
            if(!((*it).second)->aplicarFiltros(c)) {
                  c->mostrar();
            }
      }
}
```

Contenido.h

```
class Contenido {
public:
      Contenido(int idContenido, string url, EnumTipo tipo);
      void mostrar();
}
```

Contenido.cpp

```
Contenido::Contenido(int idContenido, string url, EnumTipo tipo) {
      this->idContenido = idContenido;
      this->url = url;
      this->tipo = tipo;
}

Contenido::mostrar();
```

Ya implementada

Usuario.h

```
class Usuario {
private:
	int idUsuario;
	map<int, Filtro*> filtros;
public:
	agregarFiltro(Filtro f);
	bool aplicarFiltros(Contenido c);
}
```

Usuario.cpp

```
Usuario::agregarFiltro(Filtro f){
	filtros[f->getIdFiltro()] = f;
}

bool Usuario::aplicarFiltros(Contenido c){
	bool res = false;
	map<int, Filtro*>::iterator it = filtros.begin();
	while(!res && it != filtros.end()){
		res = ((*it).second)->verificarContenido(c);
		it++;
	}
	return res;
}
```

Filtro.h

```
class Filtro {
private:
	int idFiltro;
public:
	virtual bool verificarContenido() = 0;
}
```

FiltroRegionPais.h

```
class FiltroRegionPais: public Filtro {
public:
	bool verificarContenido();
}
```