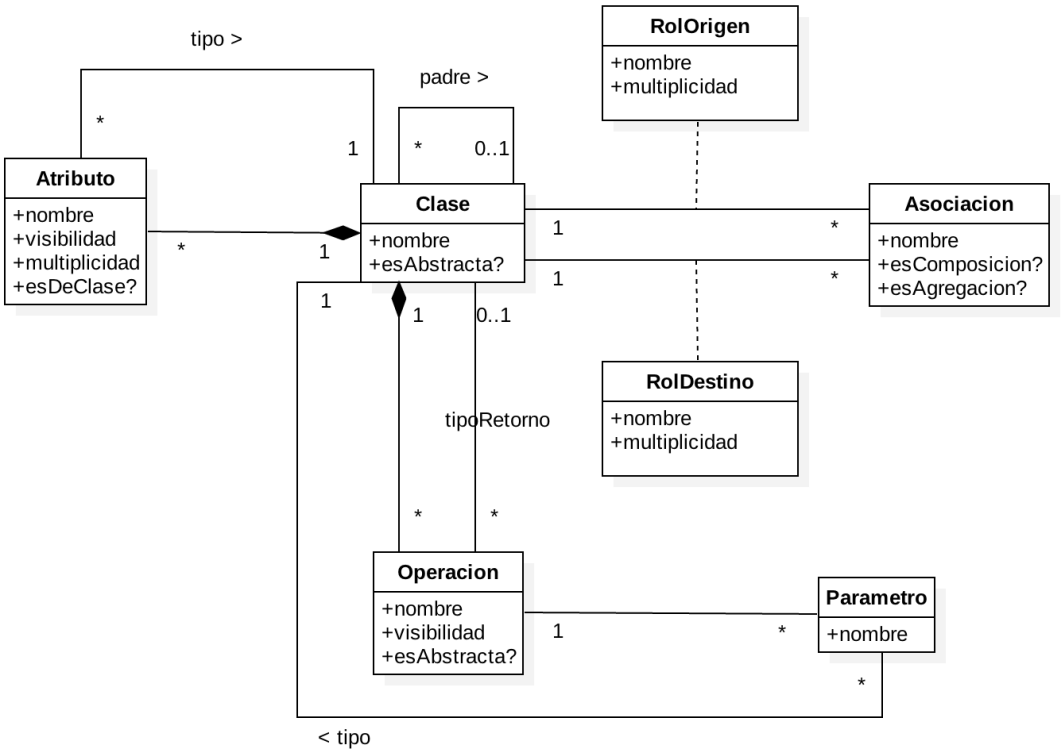
Programación 4

SOLUCIÓN EXAMEN FEBRERO 2019

Problema 1

Parte a:
Una interacción del CU se convierte en una Operación del Sistema solo si representa un Evento del Sistema, es decir, una interacción que comienza fuera del Sistema, iniciada por un Actor y a la cual el Sistema debe dar algún tipo de respuesta (atraviesa los limites del Sistema).

Parte b:



Restricciones:

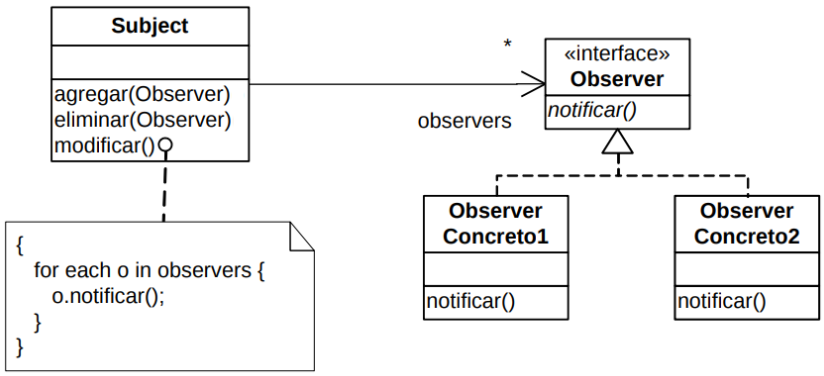
- Una clase no puede ser padre de si misma.
- El nombre de la clase la identifica.
- El nombre de la asociación la identifica.
- Si una clase contiene al menos una operación abstracta (esAbstracta=true) esta debe ser también abstracta (esAbstracta=true).

Problema 2 (35 puntos)

Parte a:

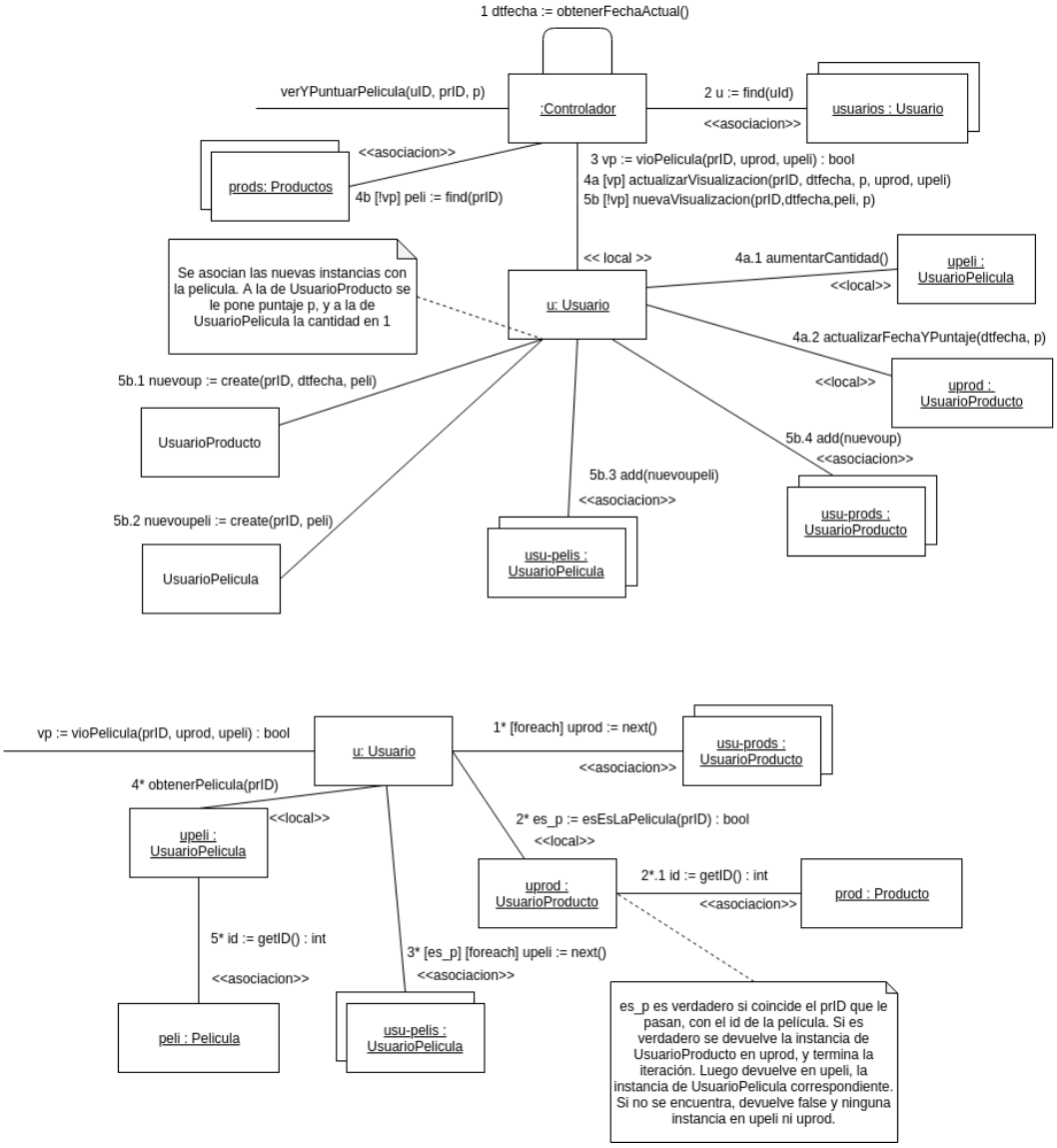
i)
Creator: A quién responsabilizar de la creación de un determinado objeto.
Alta Cohesión: Evitar que un objeto haga demasiado trabajo.

ii)
Problema tipo del observer: Definir una dependencia 1 a n entre objetos, de forma de que cuando uno cambie de estado, todos los dependientes sean notificados.



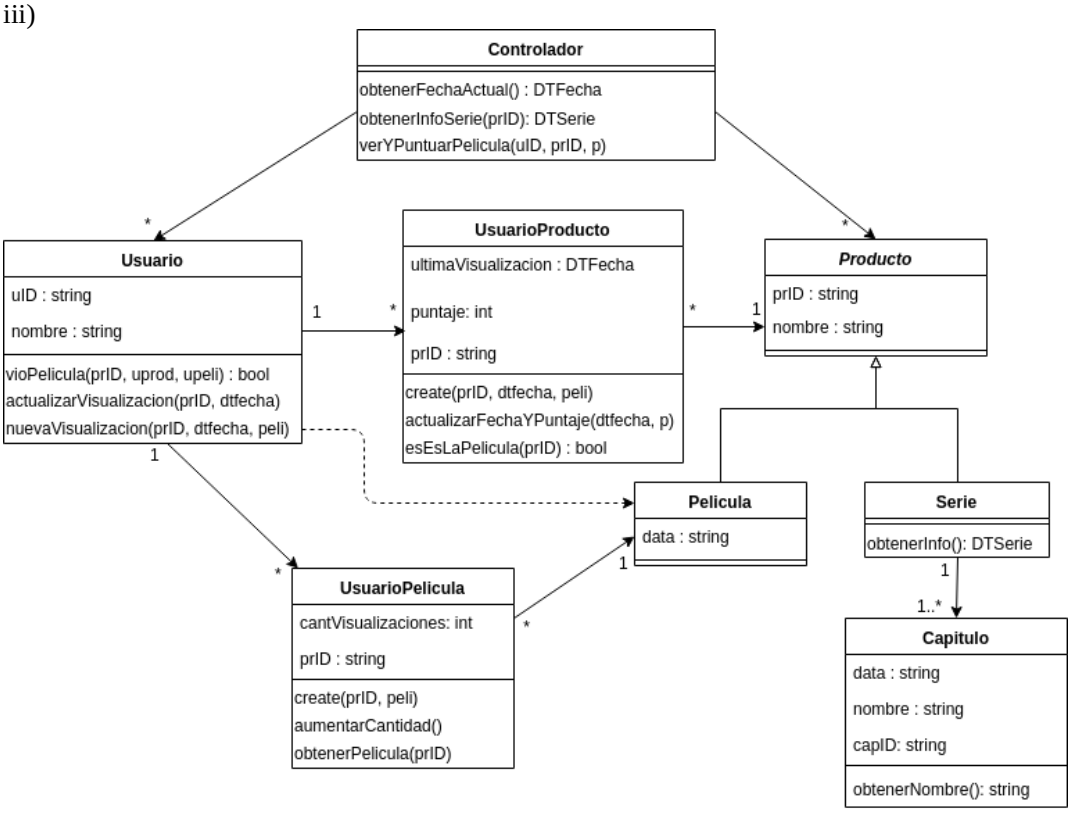
Parte b:

i) y ii)


1 dtfecha := obtenerFechaActual()
verYPuntuarPelicula(uID, prID, p)
:Controlador
2 u := find(uId)
<<asociacion>>
usuarios : Usuario
<<asociacion>>
prods: Productos
4b [!vp] peli := find(prID)
3 vp := vioPelicula(prID, uprod, upeli) : bool
4a [vp] actualizarVisualizacion(prID, dtfecha, p, uprod, upeli)
5b [!vp] nuevaVisualizacion(prID,dtfecha,peli, p)
Se asocian las nuevas instancias con la pelicula. A la de UsuarioProducto se le pone puntaje p, y a la de UsuarioPelicula la cantidad en 1
<< local >>
u: Usuario
4a.1 aumentarCantidad()
<<local>>
upeli : UsuarioPelicula
4a.2 actualizarFechaYPuntaje(dtfecha, p)
<<local>>
uprod : UsuarioProducto
5b.1 nuevoup := create(prID, dtfecha, peli)
UsuarioProducto
5b.4 add(nuevoup)
<<asociacion>>
5b.3 add(nuevoupeli)
<<asociacion>>
usu-prods : UsuarioProducto
5b.2 nuevoupeli := create(prID, peli)
UsuarioPelicula
usu-pelis : UsuarioPelicula

vp := vioPelicula(prID, uprod, upeli) : bool
u: Usuario
1* [foreach] uprod := next()
<<asociacion>>
usu-prods : UsuarioProducto
4* obtenerPelicula(prID)
upeli : UsuarioPelicula
<<local>>
2* es_p := esEsLaPelicula(prID) : bool
<<local>>
uprod : UsuarioProducto
2*.1 id := getID() : int
<<asociacion>>
prod : Producto
5* id := getID() : int
<<asociacion>>
3* [es_p] [foreach] upeli := next()
<<asociacion>>
peli : Pelicula
usu-pelis : UsuarioPelicula
es_p es verdadero si coincide el prID que le pasan, con el id de la película. Si es verdadero se devuelve la instancia de UsuarioProducto en uprod, y termina la iteración. Luego devuelve en upeli, la instancia de UsuarioPelicula correspondiente. Si no se encuentra, devuelve false y ninguna instancia en upeli ni uprod.

iii)



Problema 3

i) CProyecto: .h y .cpp completos

```
// CProyecto.h

class CProyecto {
private:
static CProyecto* instance;
CProyecto() {}
map<string, Proyecto*> proyectos;
public:
static CProyecto* getInstance();
void createProyecto(string nomProyecto);
set<IncidenteDT*> getAllIncidentes(string nomProyecto);
void deleteProyecto(string nomProyecto);
void addIToP(string nomProyecto, Incidente* incidente);
void removeIFromP(string nomProyecto, Incidente* incidente);
};

// CProyecto.cpp

CProyecto* CProyecto::instance = NULL;

CProyecto* CProyecto::getInstance() {
if (instance == NULL) {
instance = new CProyecto();
}
return instance;
}
```

```
void CProyecto::createProyecto(string nomProyecto) {
Proyecto* pry = new Proyecto(nomProyecto);
proyectos[nomProyecto] = pry;
}

set<IncidenteDT*> CProyecto::getAllIncidentes(string nomProyecto) {
return proyectos[nomProyecto]->getAllIncidentes();
}

void CProyecto::deleteProyecto(string nomProyecto) {
Proyecto* pry = proyectos[nomProyecto];
proyectos.erase(nomProyecto);
delete pry;
}

void CProyecto::addIToP(string nomProyecto, Incidente* incidente) {
proyectos[nomProyecto]->addIncidente(incidente);
}

void CProyecto::removeIFromP(string nomProyecto, Incidente* incidente) {
proyectos[nomProyecto]->removeIncidente(incidente);
}
```

ii) CIncidente: .h y las operaciones createNuevoReq y deleteNuevoReq (NOTAR que no se pide findIncidente)

```
// CIncidente.h

class CIncidente {
private:
static CIncidente* instance;
CIncidente();
map<int, Incidente*> incidentes;
public:
static CIncidente* getInstance();
void createNuevoReqInP(string nomProyecto, int nroIncidente, string texto);
IncidenteDT* findIncidente(int nroIncidente);
void deleteIncidenteFromP(string nomProyecto, int nroIncidente);
void deleteIncidente(int nroIncidente);
};

// Operaciones de CIncidente pedidas

void CIncidente::createNuevoReqInP(string nomProyecto, int nroIncidente,
string texto) {
Incidente* inc = new NuevoRequerimiento(nroIncidente, texto);
incidentes[nroIncidente] = inc;
CProyecto::getInstance()->addIToP(nomProyecto, inc);
}

void CIncidente::deleteIncidenteFromP(string nomProyecto, int nroIncidente) {
Incidente* inc = incidentes[nroIncidente];
CProyecto::getInstance()->removeIFromP(nomProyecto, inc);
deleteIncidente(nroIncidente);
}

void CIncidente::deleteIncidente(int nroIncidente) {
Incidente* inc = incidentes[nroIncidente];
incidentes.erase(nroIncidente);
delete inc;
}
```

iii) Proyecto: .h y .cpp completos

```
// Proyecto.h

class Proyecto {
private:
string nombre;
set<Incidente*> incidentes;
public:
Proyecto(string);
set<IncidenteDT*> getAllIncidentes();
void addIncidente(Incidente*);
void removeIncidente(Incidente*);
~Proyecto();
};

// Proyecto.cpp

Proyecto::Proyecto(string n) : nombre(n) {}

set<IncidenteDT*> Proyecto::getAllIncidentes() {
set<IncidenteDT*> res;

set<Incidente*>::iterator it;
for(it = incidentes.begin(); it != incidentes.end(); ++it)
res.insert((*it)->getIncidenteDT());
return res;
}

void Proyecto::addIncidente(Incidente* inc) {
incidentes.insert(inc);
}

void Proyecto::removeIncidente(Incidente* inc) {
incidentes.erase(inc);
}

Proyecto::~Proyecto() {
set<Incidente*>::iterator it;
for(it = incidentes.begin(); it != incidentes.end(); ++it) {
incidentes.erase(*it);
CIncidente::getInstance()->deleteIncidente((*it)->getNroIncidente());
}
}
```

iv) Incidente: .h y .cpp completos

```
// Incidente.h

class Incidente {
private:
int nro;
string texto;
public:
Incidente(int, string);
virtual IncidenteDT* getIncidenteDT() = 0;
int getNroIncidente() {return nro;}
virtual ~Incidente() = 0;
};

// Incidente.cpp

Incidente::Incidente(int n, string t) : nro(n), texto(t) {}
```

v) NuevoRequerimiento: .h y constructor no vacío y destructor. El constructor no vacío recibe como parámetros el nro de incidente y el texto.

```
// NuevoRequerimiento.h

class NuevoRequerimiento : public Incidente {
private:
Condicion* pre;
Condicion* post;
Condicion* identifyPre(string);
Condicion* identifyPost(string);
public:
NuevoRequerimiento(int n, string t);
~NuevoRequerimiento();
};

// Operaciones de NuevoRequerimiento pedidas

NuevoRequerimiento::NuevoRequerimiento(int n, string t) :Incidente(n, t) {
pre = identifyPre(t);
post = identifyPost(t);
}

NuevoRequerimiento::~NuevoRequerimiento() {
delete pre;
delete post;
}
```