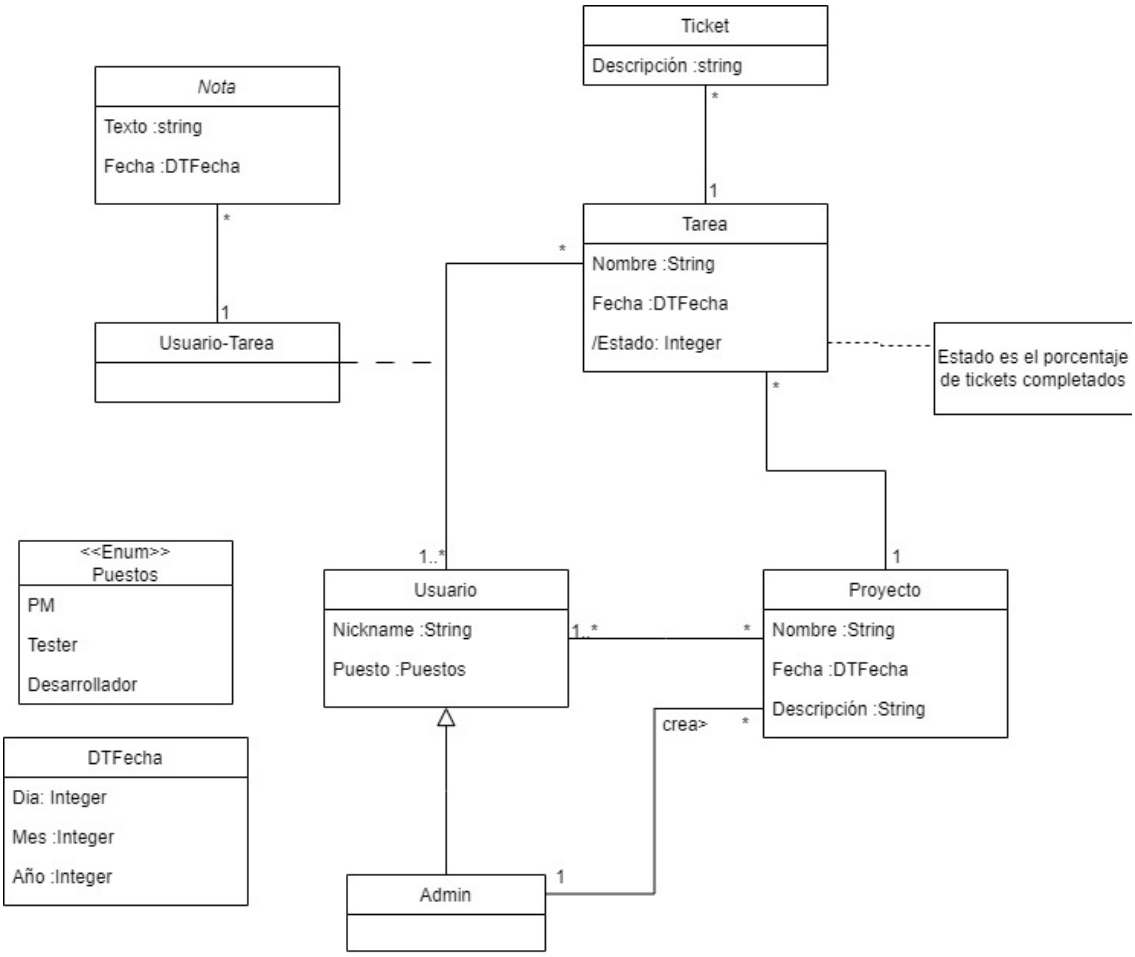
# Programación 4

SOLUCIÓN EXAMEN JULIO 2024

**Problema 1**

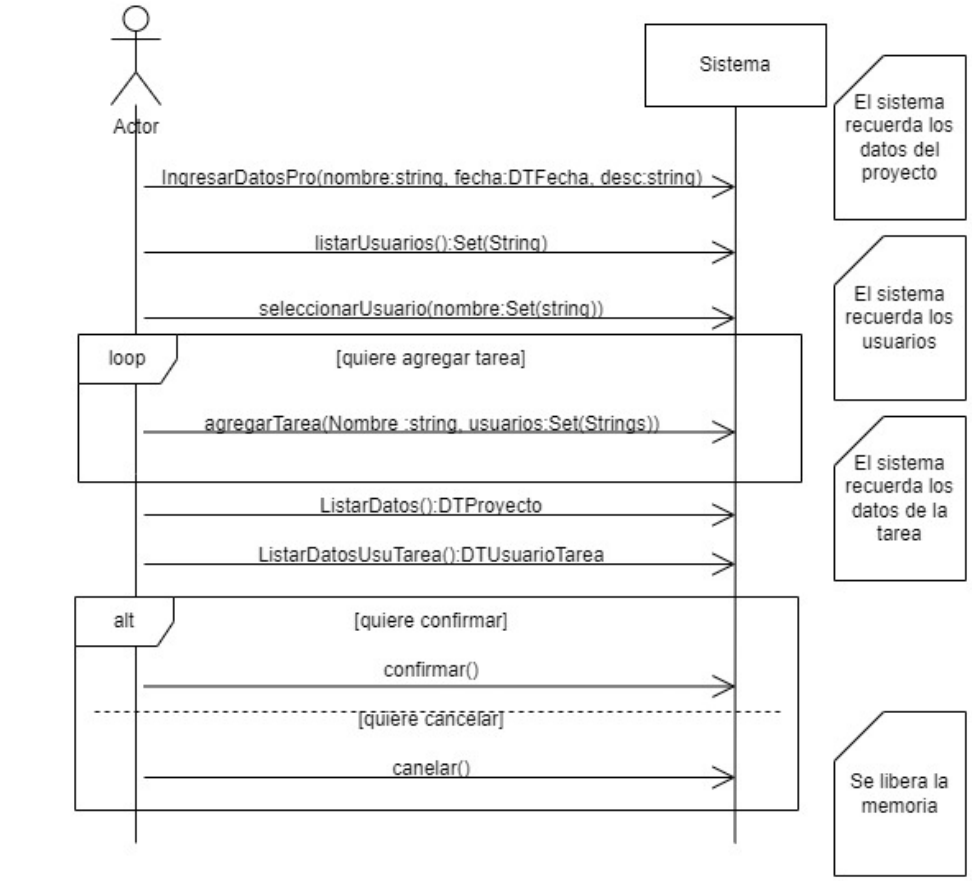
a)




Restricciones:

- El nickname del usuario es único.
- El nombre de la tarea es único.
- El nombre del proyecto es único.
- Un usuario solo puede ser asignado a una tarea de un proyecto al cual pertenece.
- El administrador que crea el proyecto también pertenece al proyecto.
- El atributo Estado se calcula como un porcentaje del total de tickets completados que pertenecen a la tarea

b)

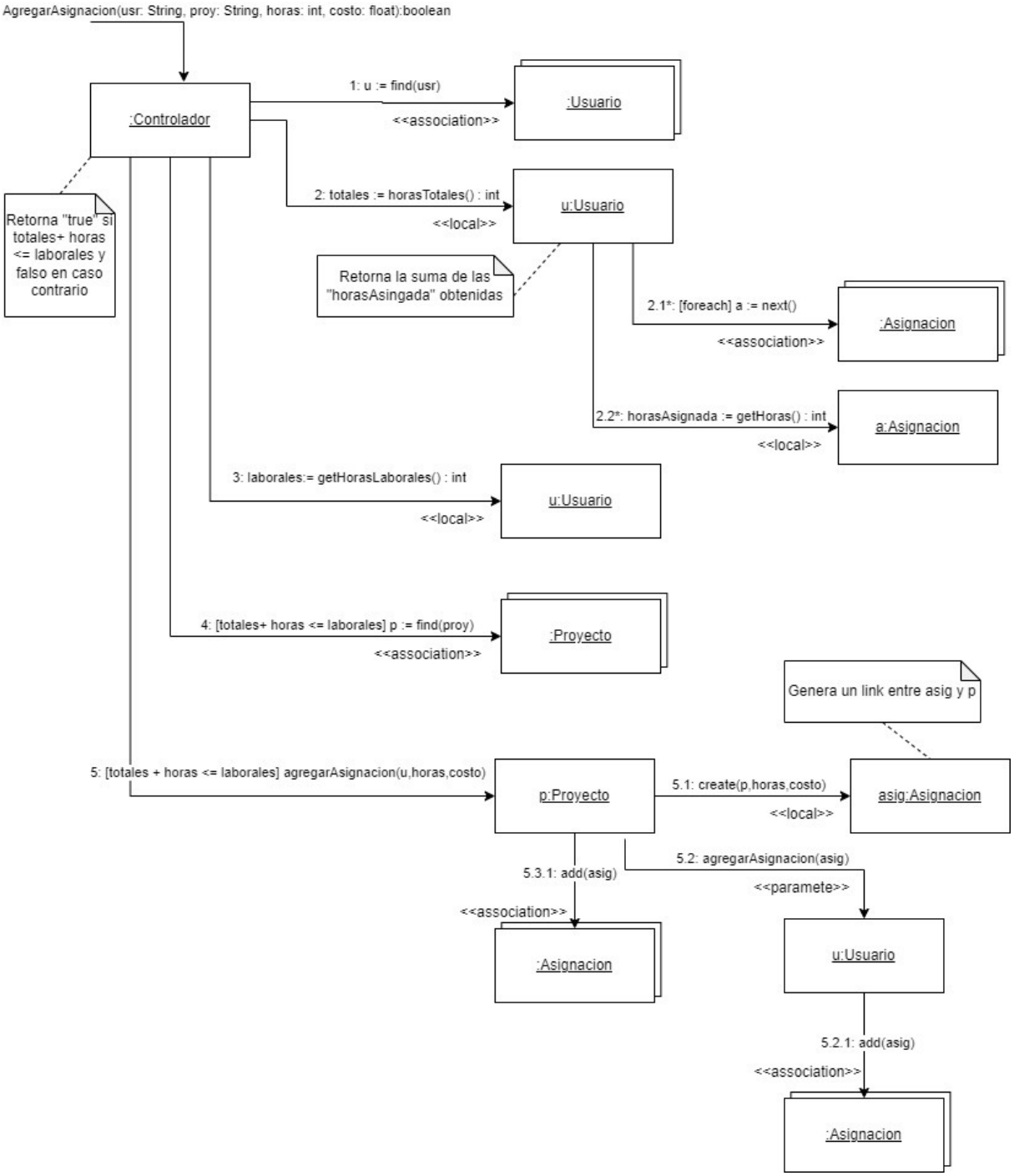

Actor
Sistema
IngresarDatosPro(nombre:string, fecha:DTFecha, desc:string)
El sistema recuerda los datos del proyecto
listarUsuarios():Set(String)
seleccionarUsuario(nombre:Set(string))
El sistema recuerda los usuarios
loop
[quiere agregar tarea]
agregarTarea(Nombre :string, usuarios:Set(Strings))
El sistema recuerda los datos de la tarea
ListarDatos():DTProyecto
ListarDatosUsuTarea():DTUsuarioTarea
alt
[quiere confirmar]
confirmar()
[quiere cancelar]
canelar()
Se libera la memoria

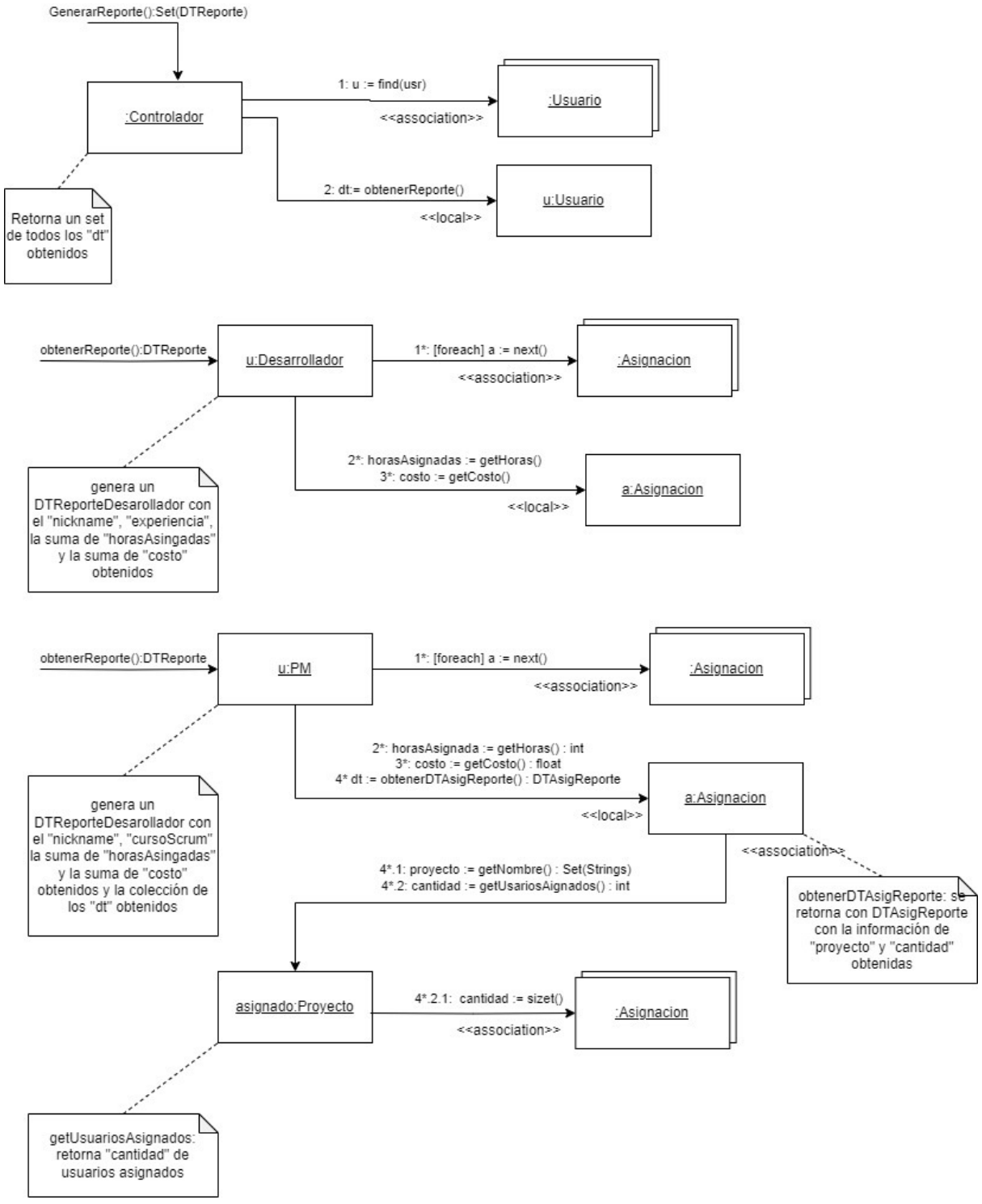

DTUsuarioTarea
NombreTarea :String
NombreUsu[*] :String


DTProyecto
Nombre:String
Fecha :DTFecha
Desc :String

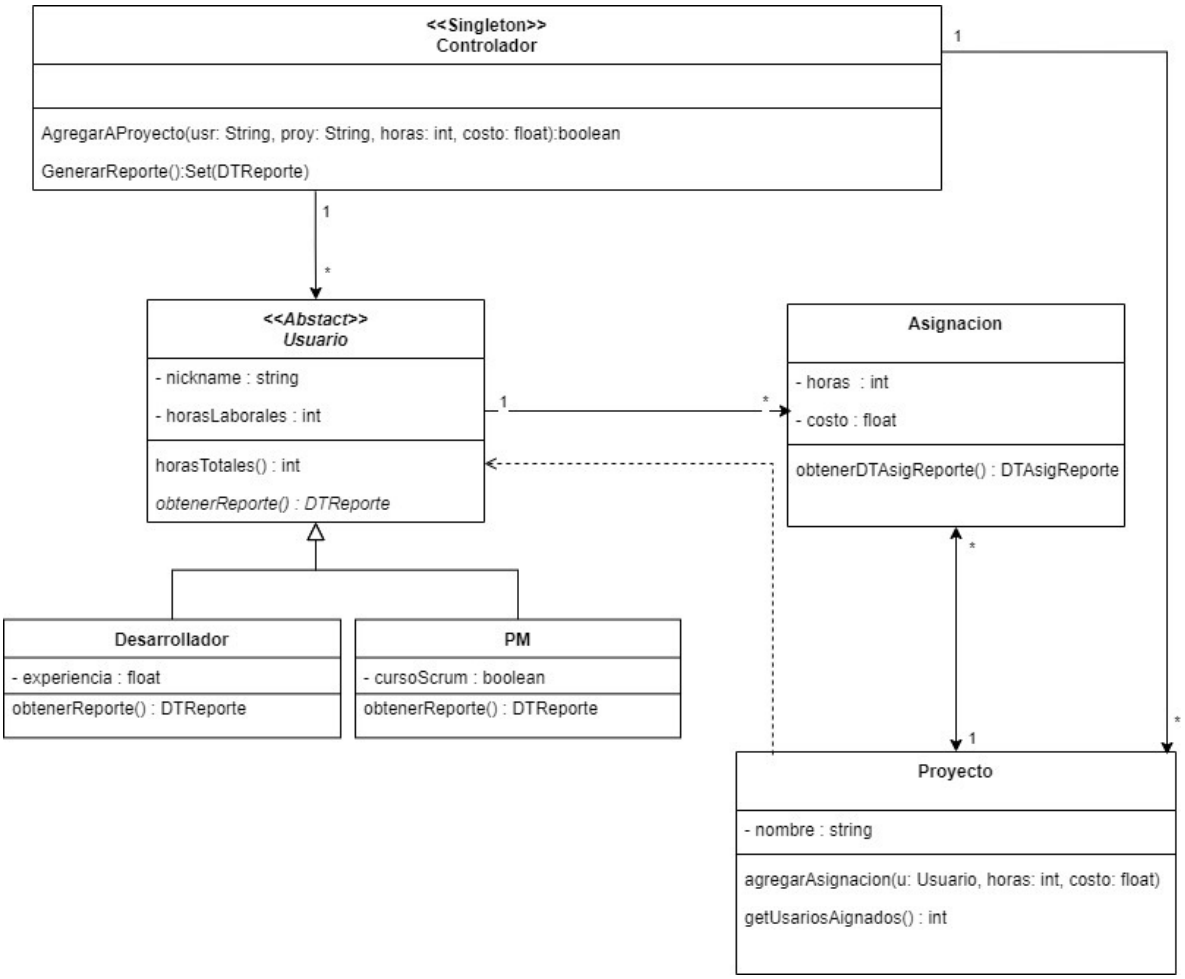
## Problema 2

a)


AgregarAsignacion(usr: String, proy: String, horas: int, costo: float):boolean
:Controlador
1: u := find(usr)
<<association>>
:Usuario
2: totales := horasTotales() : int
<<local>>
u:Usuario
Retorna "true" si totales+ horas <= laborales y falso en caso contrario
Retorna la suma de las "horasAsingada" obtenidas
2.1*: [foreach] a := next()
<<association>>
:Asignacion
2.2*: horasAsignada := getHoras() : int
<<local>>
a:Asignacion
3: laborales:= getHorasLaborales() : int
<<local>>
u:Usuario
4: [totales+ horas <= laborales] p := find(proy)
<<association>>
:Proyecto
Genera un link entre asig y p
5: [totales + horas <= laborales] agregarAsignacion(u,horas,costo)
p:Proyecto
5.1: create(p,horas,costo)
<<local>>
asig:Asignacion
5.3.1: add(asig)
<<association>>
:Asignacion
5.2: agregarAsignacion(asig)
<<paramete>>
u:Usuario
5.2.1: add(asig)
<<association>>
:Asignacion


GenerarReporte():Set(DTReporte)
:Controlador
1: u := find(usr)
<<association>>
:Usuario
2: dt:= obtenerReporte()
<<local>>
u:Usuario
Retorna un set de todos los "dt" obtenidos
obtenerReporte():DTReporte
u:Desarrollador
1*: [foreach] a := next()
<<association>>
:Asignacion
2*: horasAsignadas := getHoras()
3*: costo := getCosto()
<<local>>
a:Asignacion
genera un DTReporteDesarollador con el "nickname", "experiencia", la suma de "horasAsingadas" y la suma de "costo" obtenidos
obtenerReporte():DTReporte
u:PM
1*: [foreach] a := next()
<<association>>
:Asignacion
2*: horasAsignada := getHoras() : int
3*: costo := getCosto() : float
4* dt := obtenerDTAsigReporte() : DTAsigReporte
<<local>>
a:Asignacion
<<association>>
obtenerDTAsigReporte: se retorna con DTAsigReporte con la información de "proyecto" y "cantidad" obtenidas
genera un DTReporteDesarollador con el "nickname", "cursoScrum" la suma de "horasAsingadas" y la suma de "costo" obtenidos y la colección de los "dt" obtenidos
4*.1: proyecto := getNombre() : Set(Strings)
4*.2: cantidad := getUsariosAignados() : int
asignado:Proyecto
4*.2.1: cantidad := sizet()
<<association>>
:Asignacion
getUsuariosAsignados: retorna "cantidad" de usuarios asignados

b)


<<Singleton>>
Controlador
AgregarAProyecto(usr: String, proy: String, horas: int, costo: float):boolean
GenerarReporte():Set(DTReporte)
1
*
<<Abstact>>
Usuario
- nickname : string
- horasLaborales : int
horasTotales() : int
obtenerReporte() : DTReporte
Asignacion
- horas : int
- costo : float
obtenerDTAsigReporte() : DTAsigReporte
Desarrollador
- experiencia : float
obtenerReporte() : DTReporte
PM
- cursoScrum : boolean
obtenerReporte() : DTReporte
Proyecto
- nombre : string
agregarAsignacion(u: Usuario, horas: int, costo: float)
getUsariosAignados() : int

**Problema 3**

a) Se utiliza el patrón Proxy, con los siguientes roles:
- Subject: IPersistir
- Proxy: PersRobusta
- Real subject: Persistencia

b)

```
class IPersistir {
   public:
      virtual bool conectado() = 0;
      virtual void persistir(DTPers) = 0;
      virtual ~IPersistir() {};
}

class Persistencia : public IPersistir {
   private:
      std::vector<Canal *> cans;
   public:
      Persistencia(int);
      virtual bool conectado();
      void persistir(DTPers);
      ~Persistencia();
}

class PersRobusta : public IPersistir {
   private:
      Persistencia *pers;
      std::vector<DTPers> pendientes;
   public:
      PersRobusta();
      virtual bool conectado();
      void persistir(DTPers);
      ~PersRobusta();
}

Persistencia::Persistencia(int nCans) {
   for (int i=0; i<nCans; i++)
      cans.push_back(new Canal);
}

bool Persistencia::conectado() {
   for (int i=0; i<cans.size(); i++)
      if (cans[i]->habilitado())
         return true;
   return false;
}

void Persistencia::persistir(DTPers d) {
   for (int i=0; i<cans.size(); i++)
```

```
      if (cans[i]->habilitado()) {
         cans[i]->enviar(d);
         break;
      }
}

Persistencia::~Persistencia() {
   for (int i=0; i<cans.size(); i++)
      delete cans[i];
}

PersRobusta::PersRobusta() {
   pers = new Persistencia(NCAN);
}

bool PersRobusta::conectado() {
   return true;
}

void PersRobusta::persistir(DTPers d) {
   if (pers->conectado()) {
      pers->persistir(d);
      for (int i=0; i<pendientes.size(); i++)
          pers->persistir(pendientes[i]);
      pendientes.clear();
   } else
      pendientes.push_back(d);
}

PersRobusta::~PersRobusta() {
   delete pers;
}
```