Programación 4

PARCIAL FINAL EDICIÓN 2018 - SOLUCIÓN

Problema 1

Parte I)

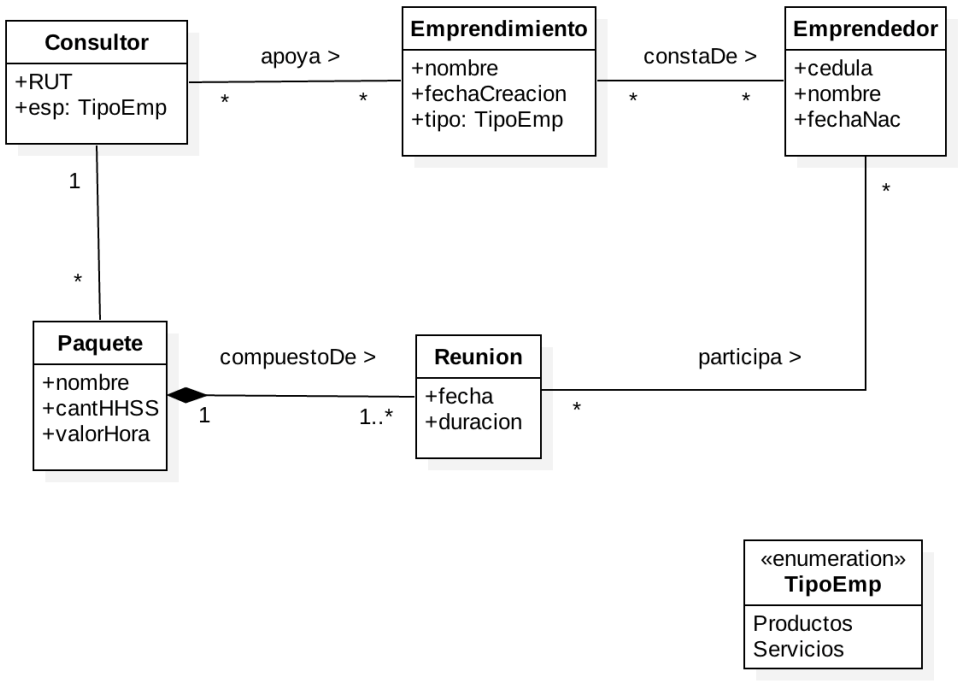
Tanto las PRE y POST condiciones se especifican en términos de:

- Existencia de objetos
- No existencia de objetos
- Existencia de links
- No existencia de links
- Modificación de valor de atributos

Luego, las PRE pueden predicar sobre los parámetros, y las POST sobre el valor de retorno.

Parte II)

a)

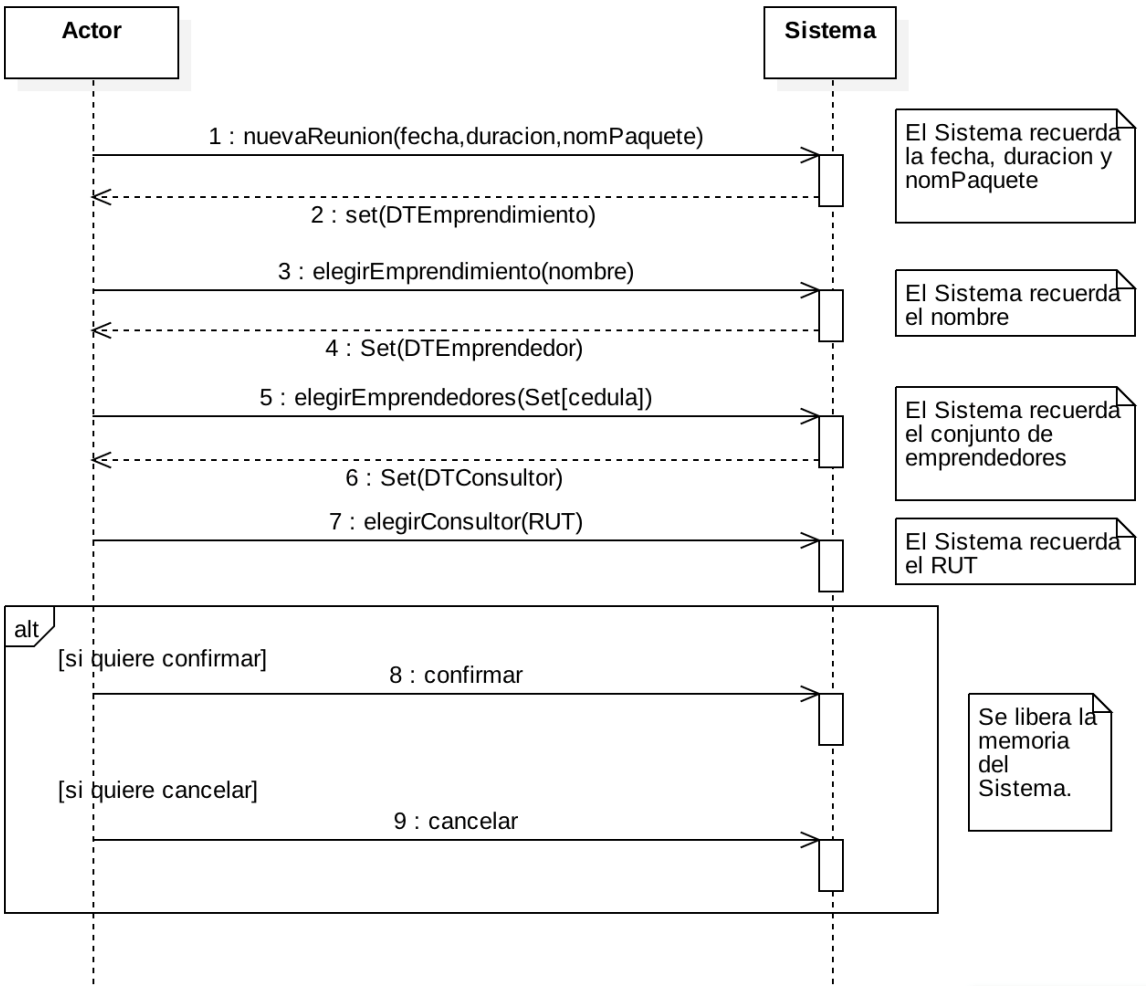


Restricciones:

- El RUT identifica a los consultores.
- La cédula identifica a los emprendedores.
- El nombre identifica al paquete.
- El nombre identifica al emprendimiento.
- El consultor solo apoya emprendimentos del tipo sobre el cual se especializa.
- En las reuniones participan emprendedores del emprendimiento que apoya el consultor.

b)

DSS con memoria

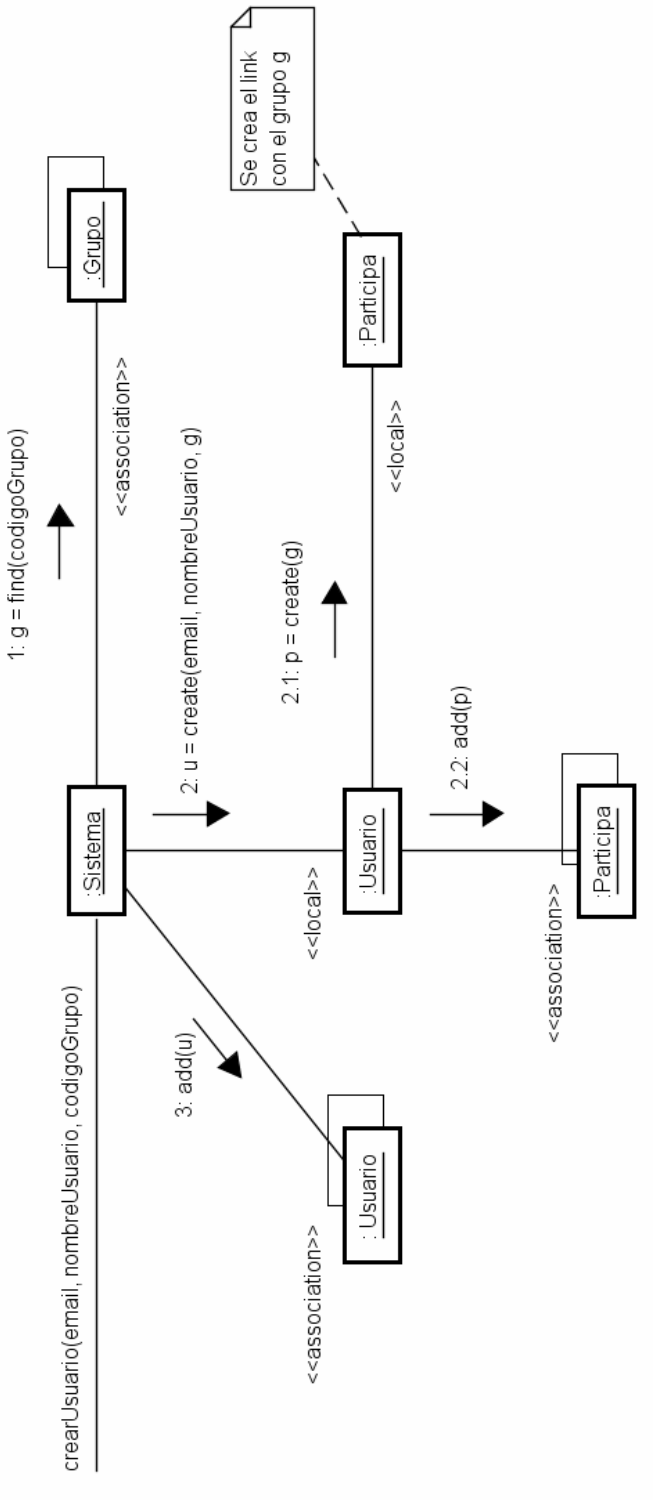


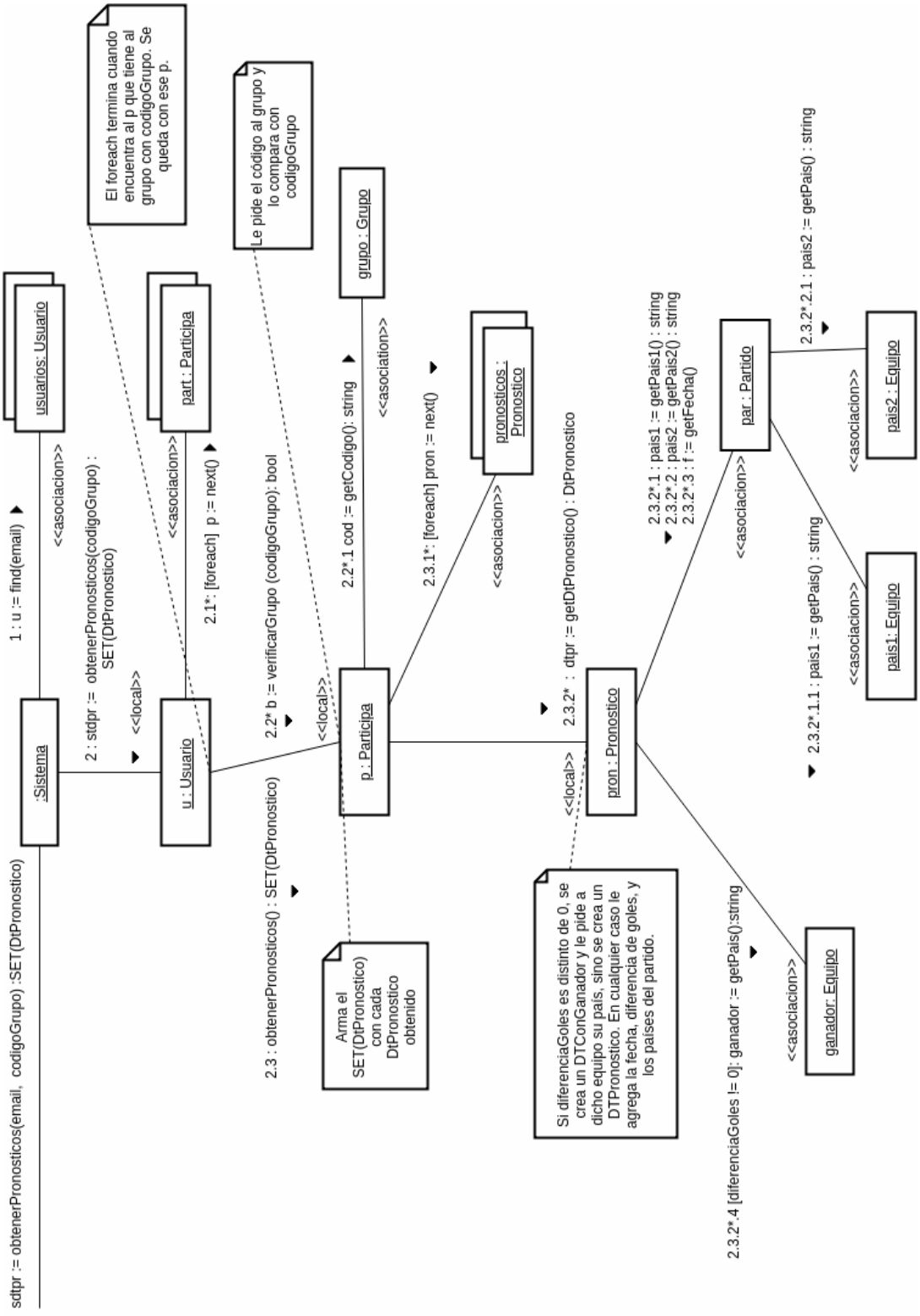
Notas:

- Se deben agregar los datatypes utilizados debajo del diagrama.
- La herramienta utilizada no permite ocultar los numeros de mensajes, los cuales NO son necesarios.

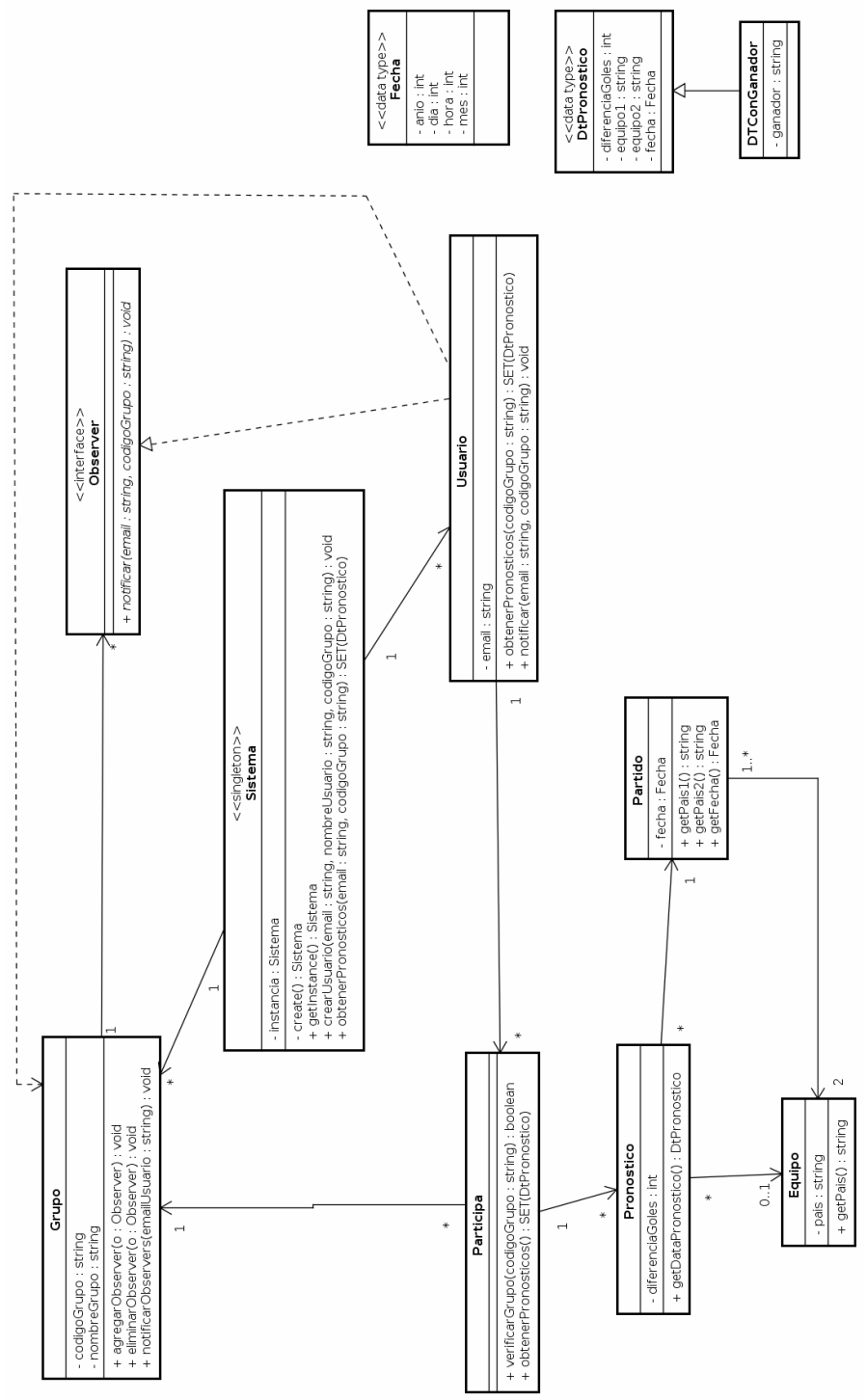
Problema 2

Parte I)




usuarios: Usuario
part : Participa
grupo : Grupo
pronosticos : Pronostico
par : Partido
pais2 : Equipo
pais1: Equipo
ganador: Equipo
:Sistema
u : Usuario
p : Participa
pron : Pronostico
El foreach termina cuando encuentra al p que tiene al grupo con codigoGrupo. Se queda con ese p.
Le pide el código al grupo y lo compara con codigoGrupo
Arma el SET(DtPronostico) con cada DtPronostico obtenido
Si diferenciaGoles es distinto de 0, se crea un DTConGanador y le pide a dicho equipo su pais, sino se crea un DTPronostico. En cualquier caso le agrega la fecha, diferencia de goles, y los países del partido.
1 : u := find(email)
2 : stdpr := obtenerPronosticos(codigoGrupo) : SET(DtPronostico)
2.1*: [foreach] p := next()
2.2* b := verificarGrupo (codigoGrupo): bool
2.2*.1 cod := getCodigo(): string
2.3 : obtenerPronosticos() : SET(DtPronostico)
2.3.1*: [foreach] pron := next()
2.3.2* : dtpr := getDtPronostico() : DtPronostico
2.3.2*.1 : pais1 := getPais1() : string
2.3.2*.2 : pais2 := getPais2() : string
2.3.2*.3 : f := getFecha()
2.3.2*.1.1 : pais1 := getPais() : string
2.3.2*.2.1 : pais2 := getPais() : string
2.3.2*.4 [diferenciaGoles != 0]: ganador := getPais():string
sdtpr := obtenerPronosticos(email, codigoGrupo) :SET(DtPronostico)
<<asociacion>>
<<local>>

Parte II)


<<data type>>
Fecha
- anio : int
- dia : int
- hora : int
- mes : int
<<data type>>
DtPronostico
- diferenciaGoles : int
- equipo1 : string
- equipo2 : string
- fecha : Fecha
DTConGanador
- ganador : string
<<interface>>
Observer
+ notificar(email : string, codigoGrupo : string) : void
Usuario
- email : string
+ obtenerPronosticos(codigoGrupo : string) : SET(DtPronostico)
+ notificar(email : string, codigoGrupo : string) : void
<<singleton>>
Sistema
- instancia : Sistema
- create() : Sistema
+ getInstance() : Sistema
+ crearUsuario(email : string, nombreUsuario : string, codigoGrupo : string) : void
+ obtenerPronosticos(email : string, codigoGrupo : string) : SET(DtPronostico)
Partido
- fecha : Fecha
+ getPais1() : string
+ getPais2() : string
+ getFecha() : Fecha
Grupo
- codigoGrupo : string
- nombreGrupo : string
+ agregarObserver(o : Observer) : void
+ eliminarObserver(o : Observer) : void
+ notificarObservers(emailUsuario : string) : void
Participa
+ verificarGrupo(codigoGrupo : string) : boolean
+ obtenerPronosticos() : SET(DtPronostico)
Pronostico
- diferenciaGoles : int
+ getDataPronostico() : DtPronostico
Equipo
- pais : string
+ getPais() : string
1
*
1..*
0..1
2

Patrón Observer.

Roles:
Subject: Grupo.
Observer: IObserver.
Observer concreto: Usuario.

Problema 3

i)

```
class Dispositivo {
  private:
    int idDisp;
  public:
    Dispositivo(int);
    virtual ~Dispositivo();
    virtual void actualizarDocDisp(DataDocumento) = 0;
};

class Grande : public Dispositivo {
  public:
    Grande(int);
    void actualizarDocDisp(DataDocumento);
};

class Chico : public Dispositivo {
  public:
    Chico(int);
    void actualizarDocDisp(DataDocumento);
};

class Documento {
  private:
    int idDoc;
  public:
    Documento(int);
    DataDocumento getDataDoc();
};

class ControlDocs {
  private:
    map<int, Documento *> docs;
    map<int, Dispositivo *> disps;
    ControlDocs() {};
    static ControlDocs *instancia;
  public:
    static ControlDocs *getInstancia();
    void actualizarDoc(int);
    void nuevoDisp(DataDispositivo);
    void eliminarDisp(int);
};

ControlDocs *ControlDocs::instancia = NULL;
```

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

void ControlDocs::eliminarDisp(int idD) {
  Dispositivo *auxDisp = disps[idD];
  disps.erase(idD);
  delete auxDisp;
}

void ControlDocs::nuevoDisp(DataDispositivo di) {
  Dispositivo *nuevo;
  int id = di.getId();
  if (di.getTipo() == Grande)
    nuevo = new Grande(id);
  else
    nuevo = new Chico(id);
  disps[id] = nuevo;
}

void ControlDocs::actualizarDoc(int idDoc) {
  DataDocumento dDoc = (docs[idDoc])->getDataDoc():
  map<int, Dispositivo *>::iterator it;
  for (it=disps.begin(); it!=disps.end(); ++it)
    it->second->actualizarDocDisp(idDoc);
}
```