Programación 4

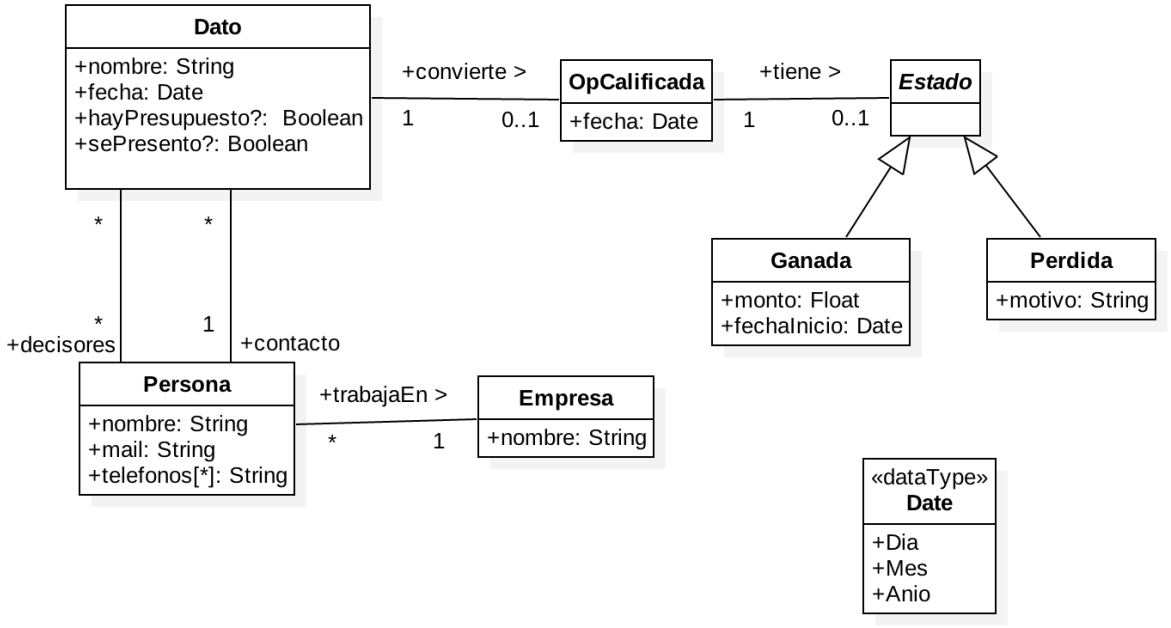
PARCIAL FINAL EDICIÓN 2017 - SOLUCIÓN

Problema 1 (35 puntos)

Parte I)

a) Un tipo asociativo es un elemento que es tanto clase como asociación.
b) Los Artefactos (definidos en la metodología) se crean utilizando Diagramas UML. Por tanto, mientras los primeros son elementos propuestos por la metodología de desarrollo (que pueden utilizar en forma restringida las construcciones UML), los segundos son herramientas gráficas que siguen el estándar UML, independientemente de la metodología.

Parte II)

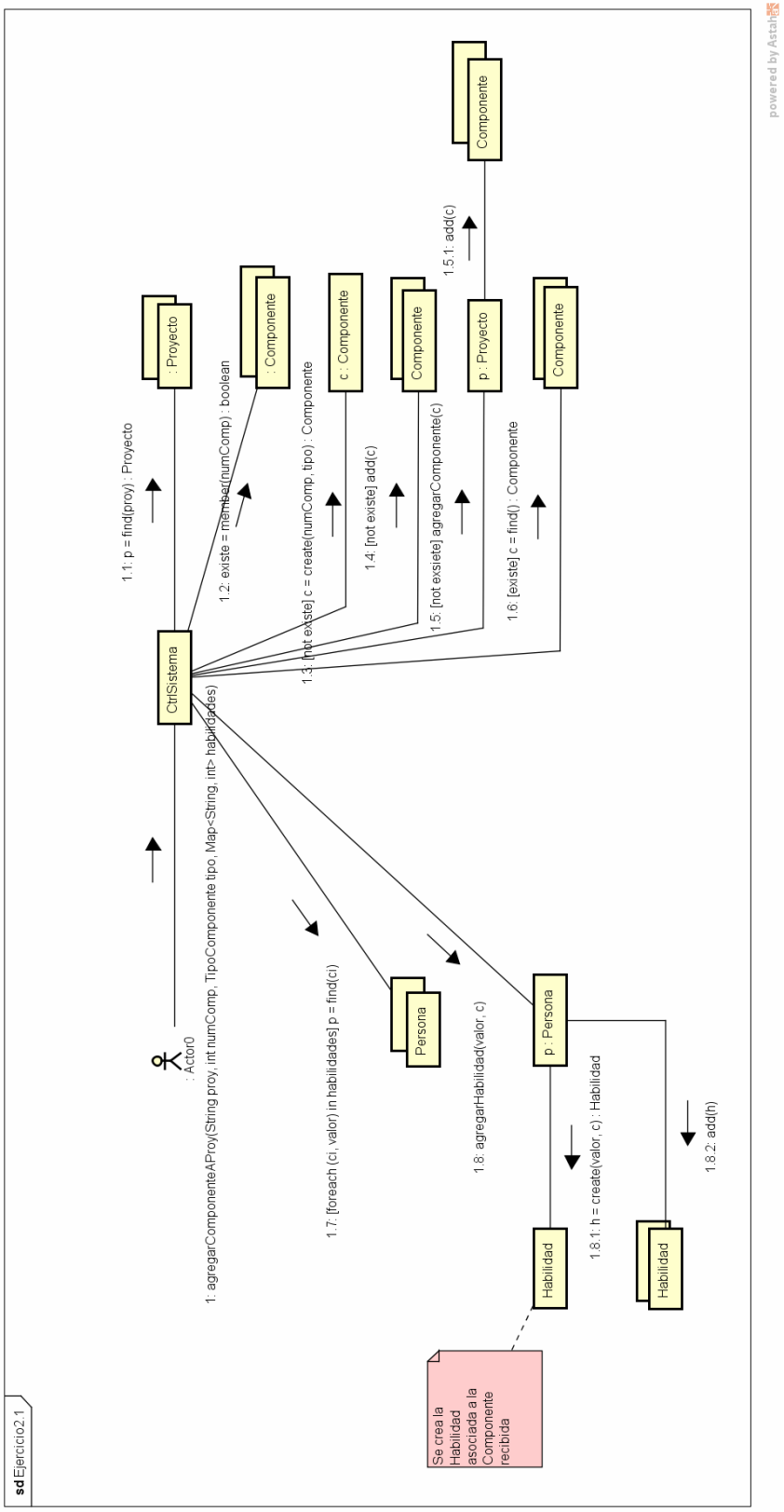


Restricciones:

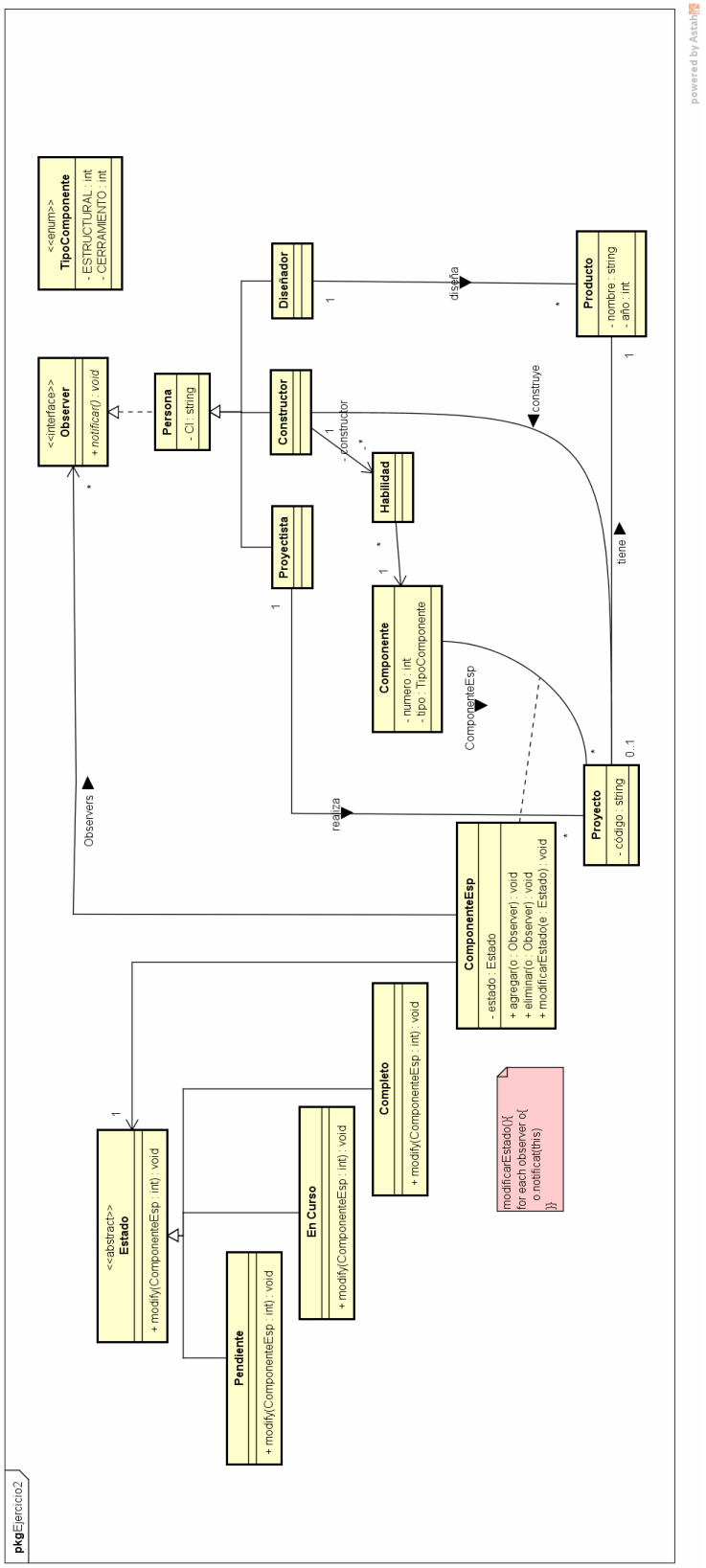
- No existen dos instancias de Dato distintas con igual valor de atributo nombre.
- Dadas una instancia de Dato d, una instancia de OpCalificada opc y una instancia de Ganada g, se cumple d.fecha <= opc.fecha <= g.fechaInicio.
- El atributo monto en Ganada debe ser positivo.
- Dada una instancia de Dato, la instancia de Persona asociada al mismo mediante "contacto" (contacto de la oportunidad) y todas las instancias de personas asociadas mediante "decisores" (tomadores de decisión) estan asociados mediante "trabajaEn" a la misma instancia de Empresa.

Problema 2 (30 puntos)

Parte I)



Parte II)


<<enum>>
TipoComponente
- ESTRUCTURAL : int
- CERRAMIENTO : int
<<interface>>
Observer
+ notificar() : void
Persona
- CI : string
Diseñador
Constructor
Proyectista
Habilidad
Componente
- numero : int
- tipo : TipoComponente
Producto
- nombre : string
- año : int
Proyecto
- código : string
ComponenteEsp
- estado : Estado
+ agregar(o : Observer) : void
+ eliminar(o : Observer) : void
+ modificarEstado(e : Estado) : void
<<abstract>>
Estado
+ modify(ComponenteEsp : int) : void
Completo
+ modify(ComponenteEsp : int) : void
En Curso
+ modify(ComponenteEsp : int) : void
Pendiente
+ modify(ComponenteEsp : int) : void
modificarEstado(){
for each observer o{
o.notificar(this)
}}
diseña
construye
tiene
realiza
Observers
ComponenteEsp
- constructor
1
*
0..1
pkgEjercicio2

Problema 3 (35 puntos)

a)

```
class Producto {
      private:
            int idP;
      public:
            Producto(int);
            virtual ~Producto();
            virtual void setPrecioItem(DataItemProd *)=0;
};

class Fijo : public Producto {
      private:
            double precioF;
      public:
            Fijo(integer, double);
            void setPrecioItem(DataItemProd *);
};

class Personalizable : public Producto {
      private:
            map<string, Componente *> componentes;
      public:
            Personalizable(int);
            ~Personalizable()
            void setPrecioItem(DataItemProd *);
            void nuevoComp(string);
            void elimComp(string);
            void nuevaOpComp(string, string, double);
};

class Componente {
      private:
            string nomComp;
            map<string, Opcion *> opciones;
      public:
            Componente(string);
            ~Componente();
            void nuevaOp(string, double);
};

class Opcion {
      private:
            string idOp;
            double precioOp;
      public:
            Opcion(string, double);
};

class DataItemProd {
      private:
            int id;
            double precio;
      public:
            DataItemProd(int);
            ~DataItemProd();
            int getID();
            double getPrecio();
            void setPrecio(double);
```

```
};

class DataItemProdFijo : public DataItemProd {
};

class DataItemProdPers : public DataItemProd {
      private:
            set<DataCompOp> personalizacion;
      public:
            set<DataCompOp> getPersonalizacion();
};

class DataCompOp {
      private:
            string comp, op;
            int precio;
      public:
            DataCompOp(string, string, int);
            string getComp();
            string getOp();
            double getPrecioOp();
};
```

b)

```
Fijo::Fijo(integer i, double d) : Producto(i) {
      precioF=d;
}

void Fijo::setPrecioItem(DataItemProd *dip) {
      dipf->setPrecio(precioF);
}

Personalizable::Personalizable(integer i) : Producto(i) {
}

Personalizable::~Personalizable() {
      map<string, Componente *>::iterator it;
      for (it=componentes.begin(); it!=componentes.end(); ++it)
            delete (*it)->second;
}

void Personalizable::setPrecioItem(DataItemProd *dip) {
      DataItemProdPers *dipp = <dynamic_cast>(DataItemProdPers *)dip;
      set<DataCompOp>::iterator it;
      double precioAcum = 0;
      for (it=dipp.getPersonalizacion().begin();
           it!=dipp.getPersonalizacion().end(); ++it)
            precioAcum = precioAcum + (*it).getPrecioOp();
      dipp->setPrecio(precioAcum);
}

void Personalizable::nuevoComp(string s) {
      componentes[s] = new Componente(s);
}

void Personalizable::elimComp(string s) {
      Componente *c = componentes[s];
      componentes.erase(s);
      delete c;
```

```
}

void Personalizable::nuevaOpComp(string c, string o, double p) {
      componentes[c]->nuevaOp(o,p);
}

Componente::Componente(string s) {
      nomComp=s;
}

Componente::~Componente() {
      map<string, Opcion *>::iterator it;
      for (it=opciones.begin(); it!=opciones.end(); ++it)
            delete (*it)->second;
}

void Componente::nuevaOp(string o, double p) {
      opciones[o] = new Opcion(o, p);
}
```