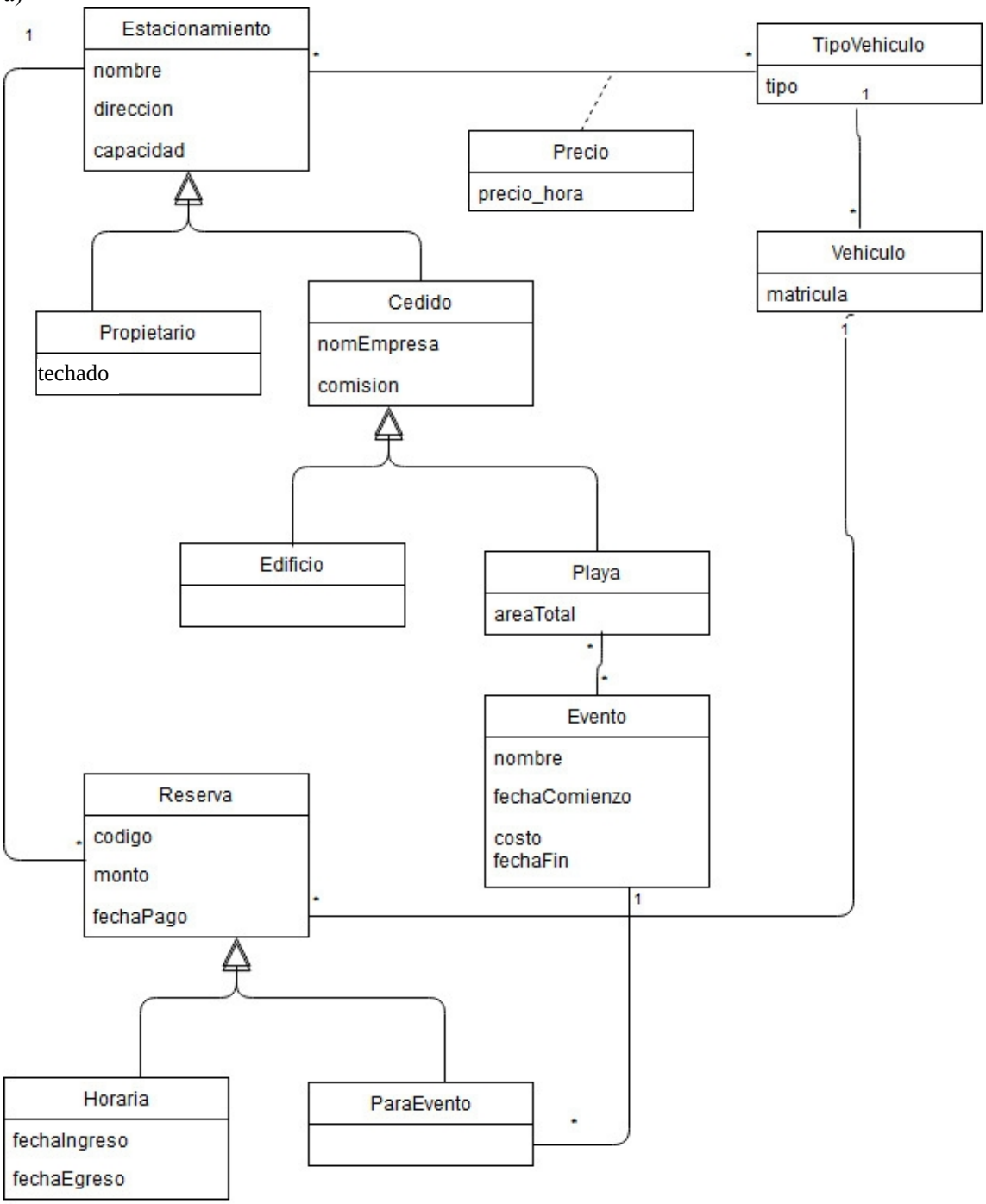
Programación 4

EXAMEN JULIO 2018 - SOLUCION

Problema 1

a)



Restricciones

Unicidad:

- Vehículo se identifica por la matrícula.
- Tipo de vehículo se identifica por tipo.
- Estacionamiento se identifica por nombre.
- Evento se identifica por el par: nombre, fechaComienzo.
- Reserva se identifica por código.

Valores de atributos:

- Capacidad de Estacionamiento mayor que cero.
- Precio_hora de Precio mayor que cero.
- AreaTotal de Playa mayor que cero.
- Costo de Evento mayor que cero.
- Monto de Reserva mayor que cero.
- Fecha de Comienzo menor que Fecha Fin en Evento.
- Fecha de Ingreso menor que Fecha de Egreso en Horaria.

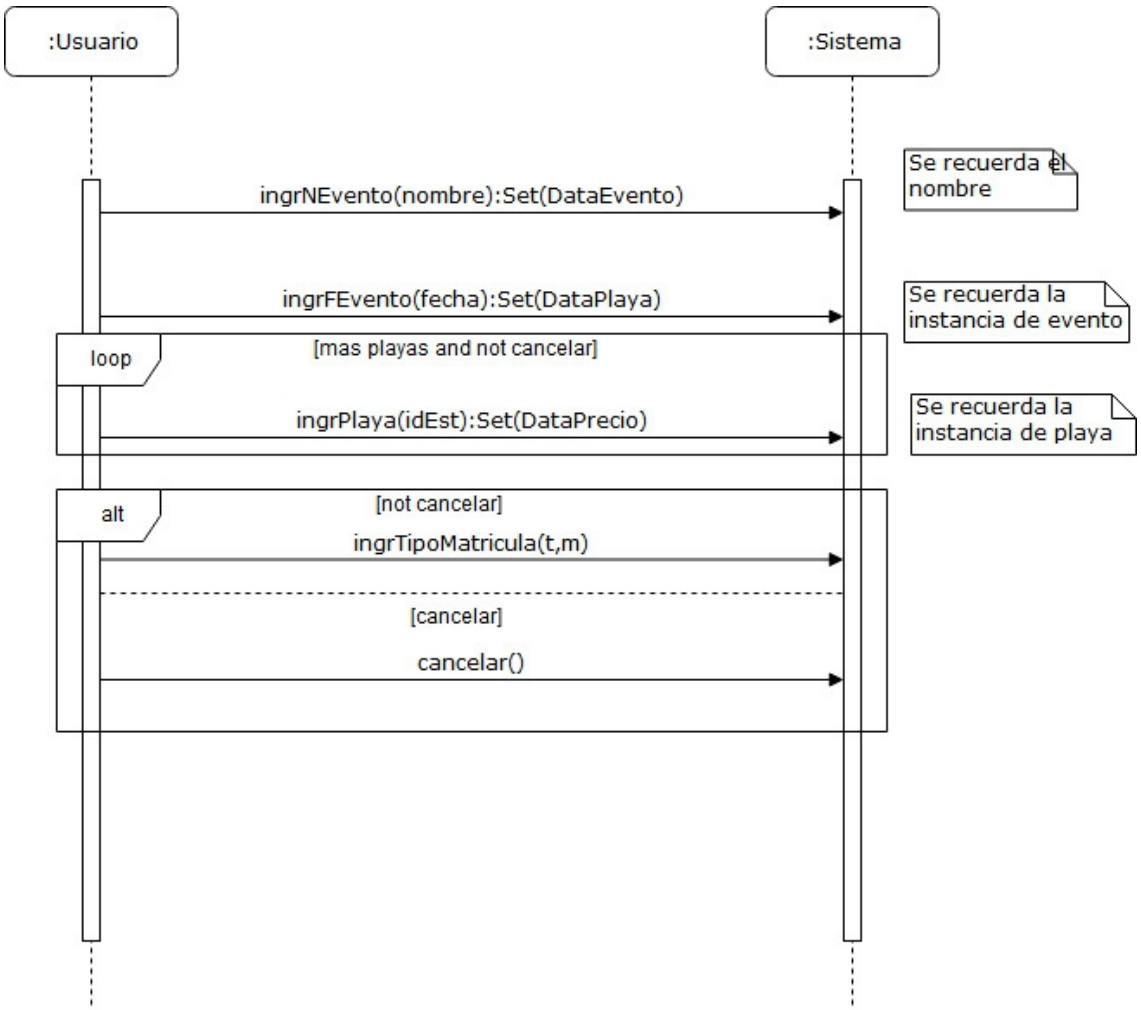
Circulares:

- Dada una reserva, el tipo del vehículo debe estar dentro de los tipos de vehículo con precio por hora del estacionamiento.
- Dada una Reserva para Evento, el Estacionamiento asociado (playa) debe estar dentro de las Playas de estacionamiento del Evento de la reserva.

Reglas de negocio:

- Dada una fecha y un estacionamiento, la cantidad de reservas del estacionamiento en esa fecha no puede exceder la capacidad del estacionamiento.
- Dada un reserva para evento, el estacionamiento asociado deber ser una playa de estacionamiento.

b)



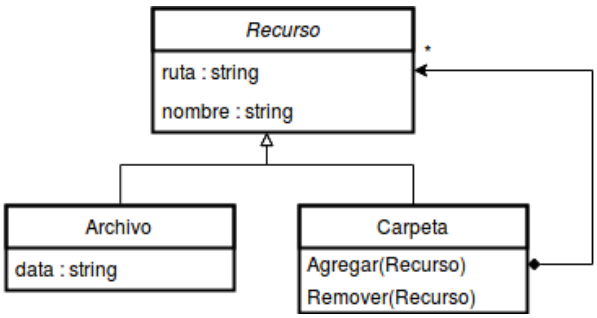
Se deben definir los data types DataEvento, DataPlaya y DataPrecio.

Problema 2

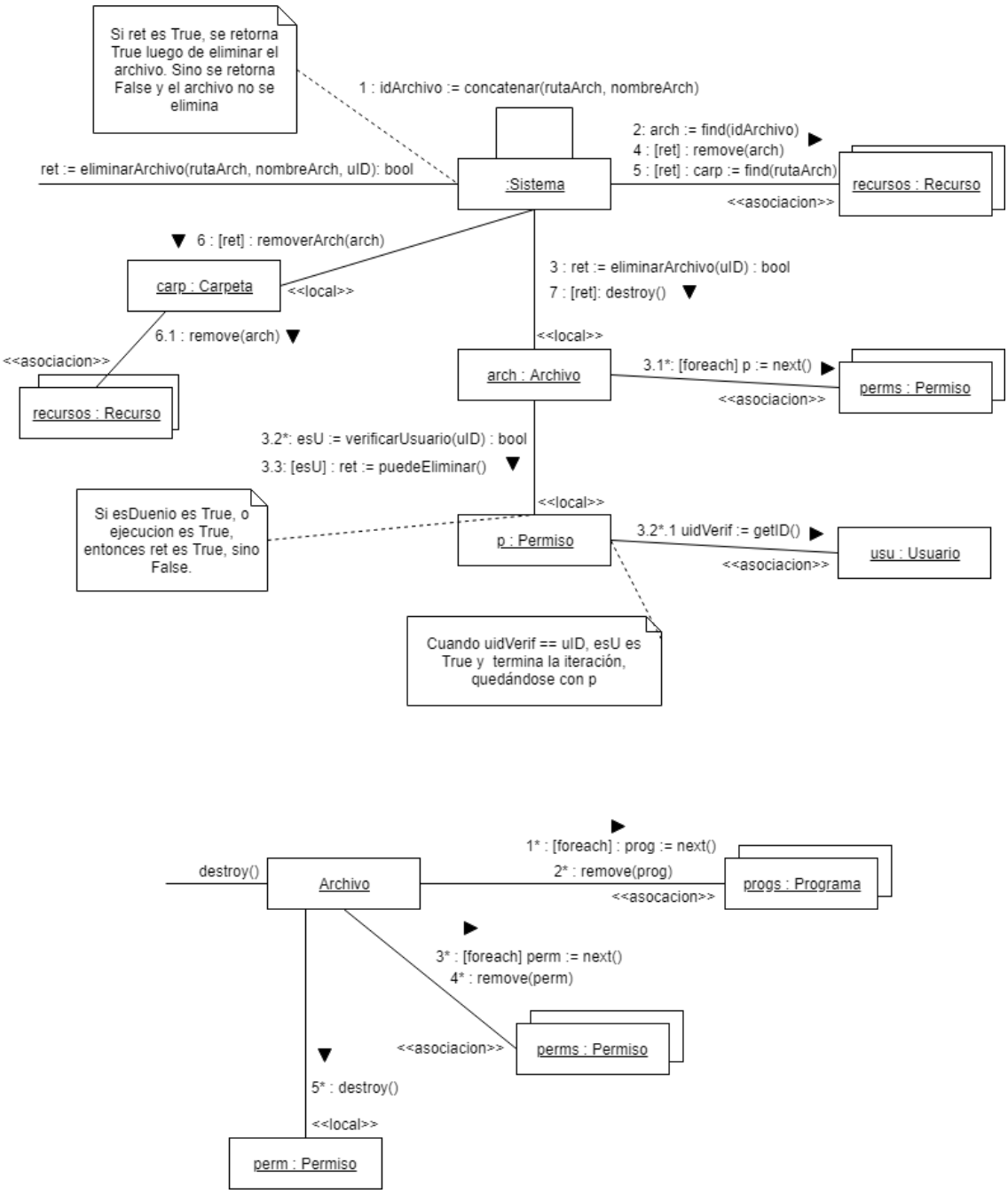
a)

Patrón de Diseño: Composite
Roles:
Componente: Recurso
Compuesto: Carpeta
Hoja: Archivo

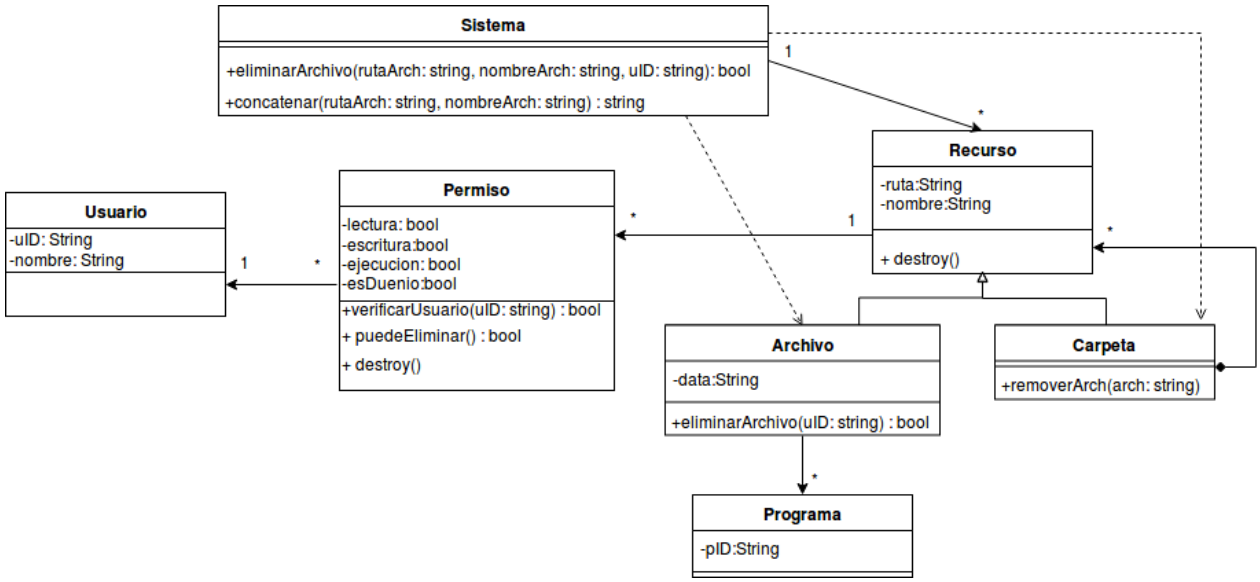
Problema general que resuelve:
Componer objetos en estructuras arborescentes para representar jerarquías de objetos compuestos y tratar uniformemente los mismos.



b)



c)


Sistema
+eliminarArchivo(rutaArch: string, nombreArch: string, uID: string): bool
+concatenar(rutaArch: string, nombreArch: string) : string
1
*
Recurso
-ruta:String
-nombre:String
+ destroy()
Usuario
-uID: String
-nombre: String
1
*
Permiso
-lectura: bool
-escritura:bool
-ejecucion: bool
-esDuenio:bool
+verificarUsuario(uID: string) : bool
+ puedeEliminar() : bool
+ destroy()
*
1
*
Archivo
-data:String
+eliminarArchivo(uID: string) : bool
Carpeta
+removerArch(arch: string)
*
Programa
-pID:String

Problema 3

ControladorSatelite.h

```
class ControladorSatelite {

private:
    Transmisor* transmisor;
    ModoSatelite* modoActual;
    ControladorSatelite();
    static ControladorSatelite* instancia;

public:
     void cambiarModo();
     void enviarDato(string dato);
     Transmisor* darTransmisor();
     virtual ~ControladorSatelite();
     static ControladorSatelite* darInstancia();

};
```

ModoSatelite.h

```
class ModoSatelite {

public:
    virtual void enviarDato(ControladorSatelite* c, string dato) = 0;
    virtual ModoSatelite* darProximoModo(ControladorSatelite* c) = 0;
    virtual ~ModoSatelite();

};
```

PanelSolar.h

```
class PanelSolar : public ModoSatelite {

public:
    void enviarDato(ControladorSatelite* c, string dato);
    ModoSatelite* darProximoModo(ControladorSatelite* c);

};
```

Bateria.h

```
class Bateria : public ModoSatelite {

private:
    set<string> memoria;

public:
    void enviarDato(ControladorSatelite* c, string dato);
    ModoSatelite* darProximoModo(ControladorSatelite* c);

};
```

ControladorSatelite.cpp

```
ControladorSatelite* ControladorSatelite::instancia = NULL;

ControladorSatelite::ControladorSatelite() {
    transmisor = new Transmisor();
    modoActual = new Bateria();
}

void ControladorSatelite::cambiarModo() {
    ModoSatelite* prox = modoActual->darProximoModo(this);
    delete modoActual;
    modoActual = prox;
}

void ControladorSatelite::enviarDato(string dato) {
    modoActual->enviarDato(this, dato);
}

ControladorSatelite* ControladorSatelite::darInstancia() {
    if (instancia == NULL)
        instancia = new ControladorSatelite();
    return instancia;
}

Transmisor* ControladorSatelite::darTransmisor() {
    return transmisor;
}

ControladorSatelite::~ControladorSatelite() {
    delete modoActual;
    delete transmisor;
}
```

ModoSatelite.cpp

```
ModoSatelite::~ModoSatelite() { };
```

PanelSolar.cpp

```
void PanelSolar::enviarDato(ControladorSatelite* c, string dato) {
    Transmisor* transmisor = c->darTransmisor();
    transmisor->transmitir(dato);
}

ModoSatelite* PanelSolar::darProximoModo(ControladorSatelite* c) {
    return  new Bateria();

}
```

Bateria.cpp

```
void Bateria::enviarDato(ControladorSatelite* c, string dato) {
    memoria.insert(dato);
}

ModoSatelite* Bateria::darProximoModo(ControladorSatelite* c) {
    Transmisor* transmisor = c->darTransmisor();
    set<string>::iterator it;
    for (it = memoria.begin(); it != memoria.end(); it++) {
        string dato = *it;
        transmisor->transmitir(dato);
    }
    return new PanelSolar();
}
```