



UNIVERSIDAD DEL BÍO-BÍO

Paradigmas de la Programación

Diagrama de clases

.



Diagrama de clases

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

Diagrama de clases

Se compone de:

- Clases
- Relaciones

Declaración de Clase

La declaración de atributo se compone:






- Modificador de acceso (-, #, +)
- Tipo de dato.
- Nombre de atributo.

La declaración de método se compone:

- Modificador de acceso (-, #, +)
- Dato de retorno.
- Nombre de método y parámetros.



Clase Persona - Ejemplo

 Persona
 - String nombre  - short edad  - boolean soltero  - float altura
<ul style="list-style-type: none">● +String getNombre()● +void setNombre(String nombre)● +short getEdad()● +void setEdad(short edad)● +boolean isSoltero()● +void setSoltero(boolean soltero)● +float getAltura()● +void setAltura(float altura)

Modificadores de acceso

Para indicar la accesibilidad de un atributo o método se denota a través de los símbolos asociados (-, #, +).

- : private

:protected

+ : public



UNIVERSIDAD DEL BÍO-BÍO

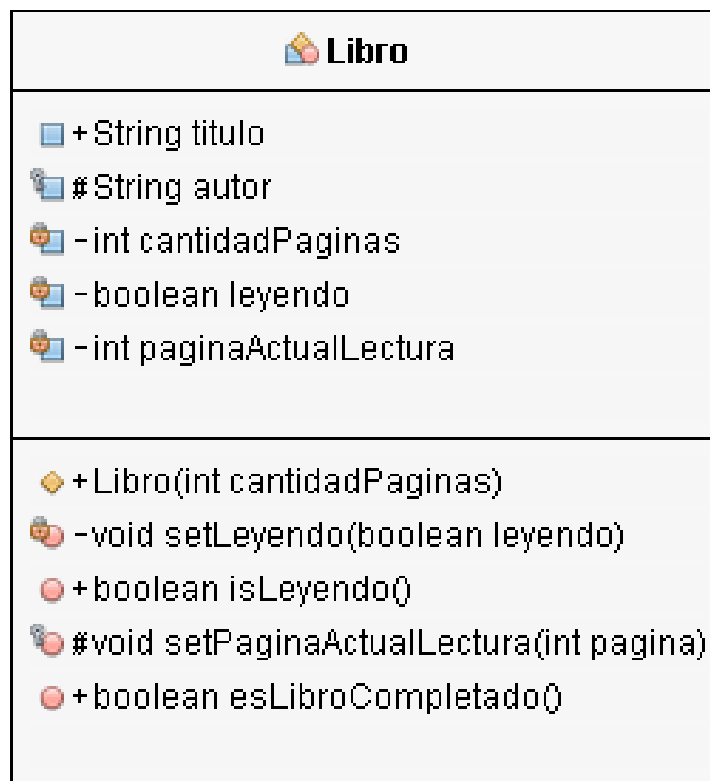
Ejercicios



Mascota




- + String nombre
- + int edad
- # String especie
- boolean jugando




- + Mascota(String nombre, int edad)
- + Mascota(String especie)
- + void jugar()
- + void dejarDeJugar()
- + boolean isJugando()





DiscoDuro

 #int capacidad
 -float velocidadLectura
 -float velocidadEscritura

 +DiscoDuro(float velocidadLectura, float velocidadEscritura)
 +DiscoDuro(float velocidadGeneral)
 +float getVelocidadGeneralPromedio()



PlanMovil

 #int costo
 - int megaBytes
 - int minutosLlamadas

◆ +PlanMovil(int costo, int megaBytes, int minutosLlamadas)
◆ +PlanMovil(int megaBytes, int minutosLlamadas)
● +int ocuparMinutos(int minutos)
● +int ocuparMegaBytes(int megabytes)



```
public class NumeroAlmacen {
```

```
    private long ultimoNumeroAlmacenado;  
    private String ultimoTipoAlmacenado;
```

```
    public void almacenar(byte numero) {  
        ultimoTipoAlmacenado = "byte";  
        ultimoNumeroAlmacenado = numero;  
    }
```

```
    public void almacenar(short numero) {  
        ultimoTipoAlmacenado = "short";  
        ultimoNumeroAlmacenado = numero;  
    }
```

```
    private void almacenar(int numero) {  
        ultimoTipoAlmacenado = "int";  
        ultimoNumeroAlmacenado = numero;  
    }
```

```
    public void almacenar(long numero) {  
        ultimoTipoAlmacenado = "long";  
        ultimoNumeroAlmacenado = numero;  
    }
```

```
    protected String getUltimoTipoAlmacenado() { return ultimoTipoAlmacenado; }
```

```
    protected long getUltimoNumeroAlmacenado() { return ultimoNumeroAlmacenado; }
```

```
}
```

Representar el siguiente código en un Diagrama de Clases

Relaciones entre clases

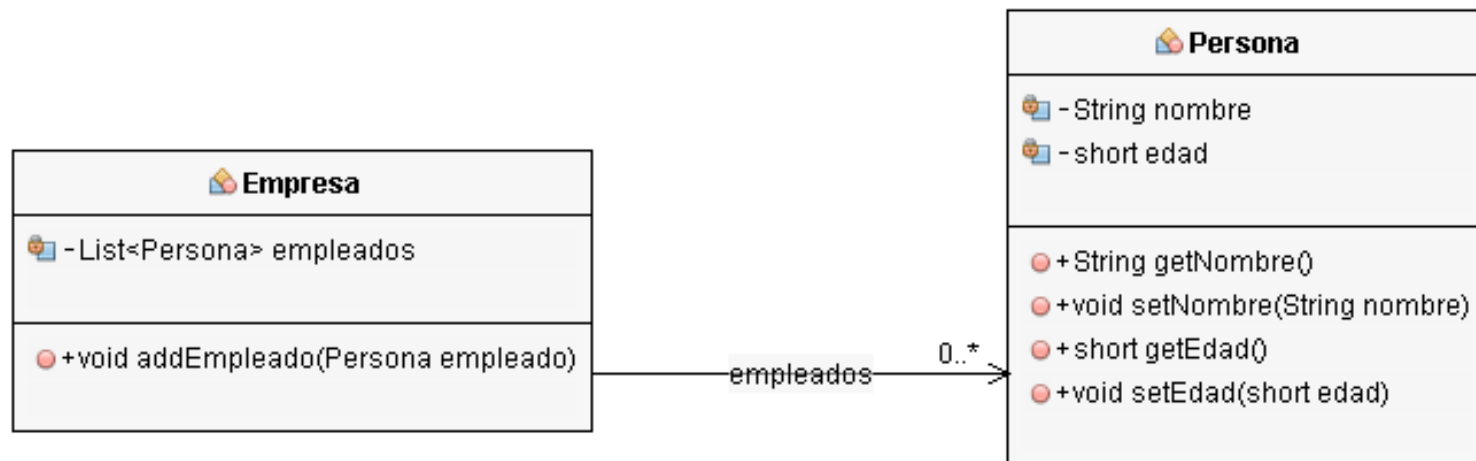
Para la indicación de relaciones se utiliza el concepto de cardinalidad. En UML, la cardinalidad de las relaciones indica el grado y nivel de dependencia entre las clases. Se anota en cada extremo de la relación y estas pueden variar en:

- **uno o muchos:** $1..^*$ ($1..n$)
- **0 o muchos:** $0..^*$ ($0..n$)
- **número fijo:** m (m denota el número).

Relaciones - Asociación

Relación estructural entre clases:

En tiempo de ejecución existe una conexión entre las instancias de las clases.



Obs. 1: La asociación en la imagen es unidireccional.

Obs. 2: La asociación puede ser bidireccional (No se indica ninguna flecha, solamente se indican las cardinalidades en cada clase).



Código de Ejemplo - Asociación

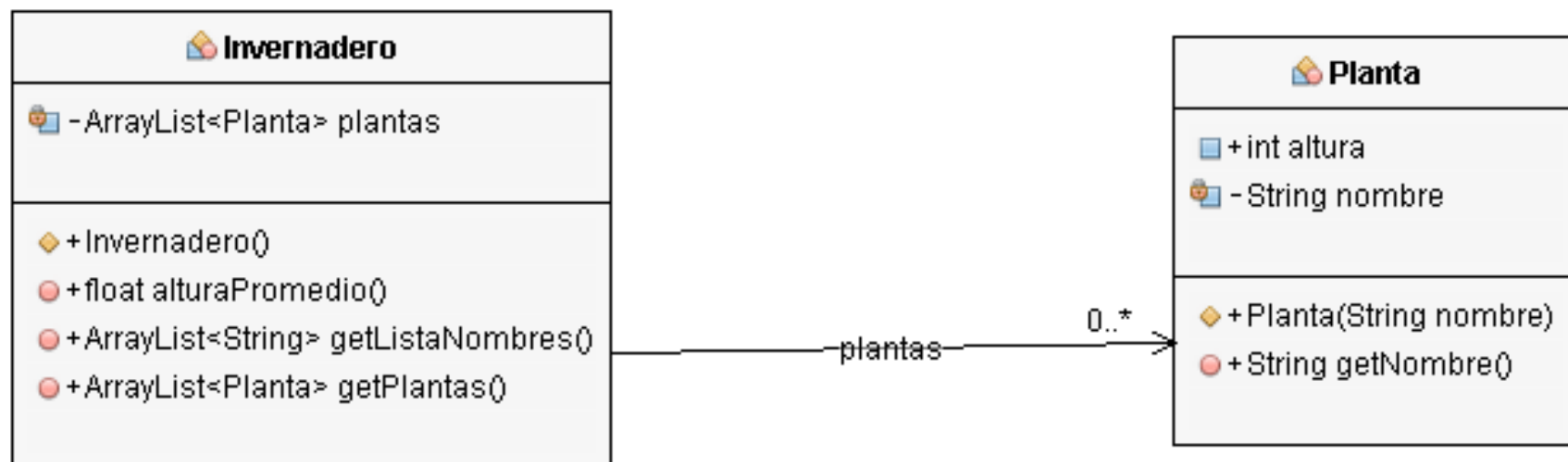
```
public class Empresa {  
  
    private List<Persona> empleados;  
  
    public Empresa() {  
        empleados = new ArrayList<>();  
    }  
  
    public void addEmpleado(Persona empleado) {  
        empleados.add(empleado);  
    }  
}
```

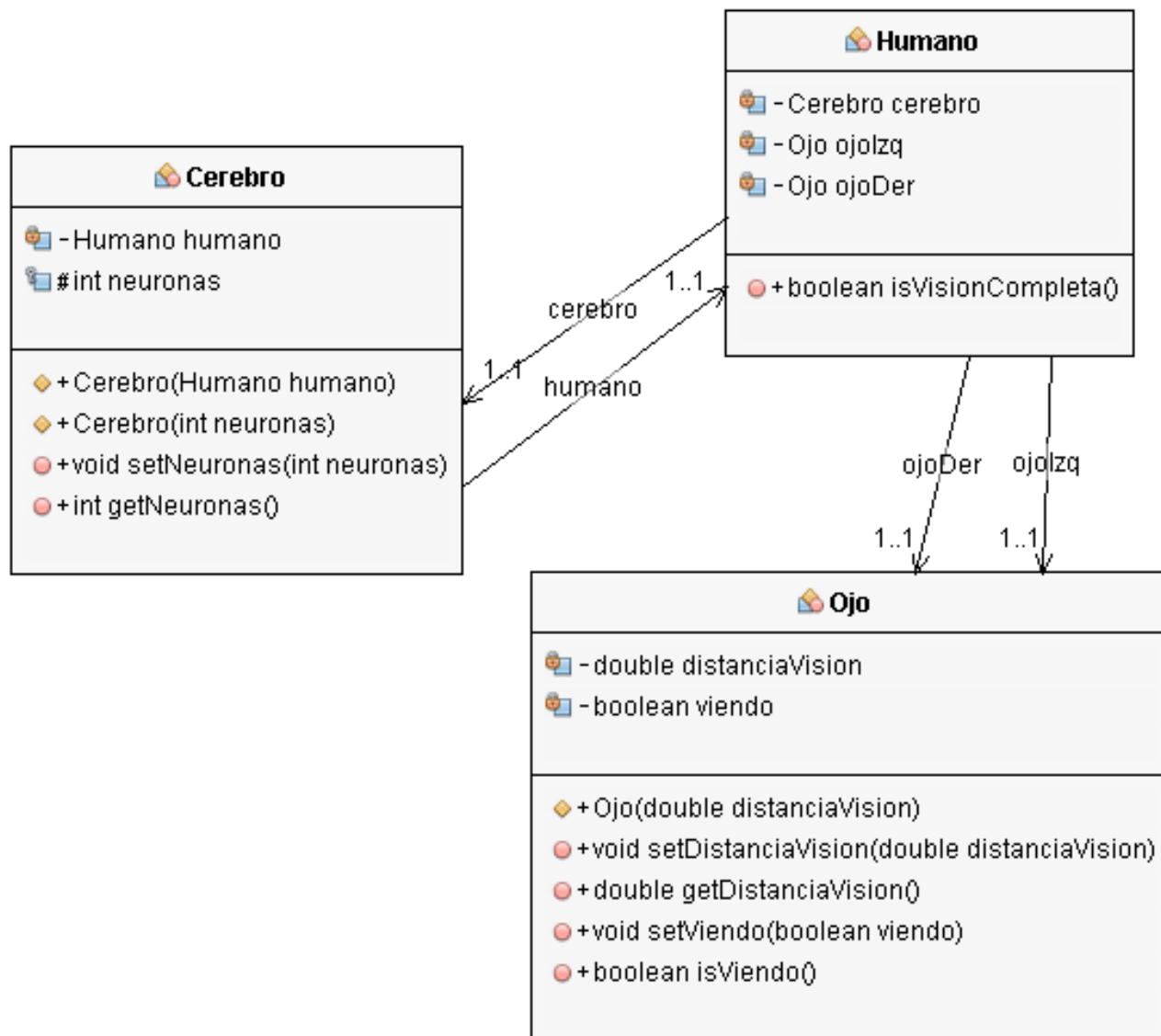
```
public class Persona {  
  
    private String nombre;  
    private short edad;  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public short getEdad() {  
        return edad;  
    }  
  
    public void setEdad(short edad) {  
        this.edad = edad;  
    }  
}
```



UNIVERSIDAD DEL BÍO-BÍO

Ejercicios







Representar el siguiente código en un Diagrama de Clases

```
public class Proceso1 {  
  
    public ArrayList<Proceso4> procesos4;  
  
    public Proceso1() {  
        procesos4 = new ArrayList<>();  
    }  
}  
  
public class Proceso2 {  
  
    public Proceso4 proceso4;  
  
}
```

```
public class Proceso3 {  
  
}  
  
public class Proceso4  
{  
    private Proceso2 proceso2;  
  
    public Proceso4()  
    {  
        proceso2 = new Proceso2();  
        proceso2.proceso4 = this;  
    }  
}
```



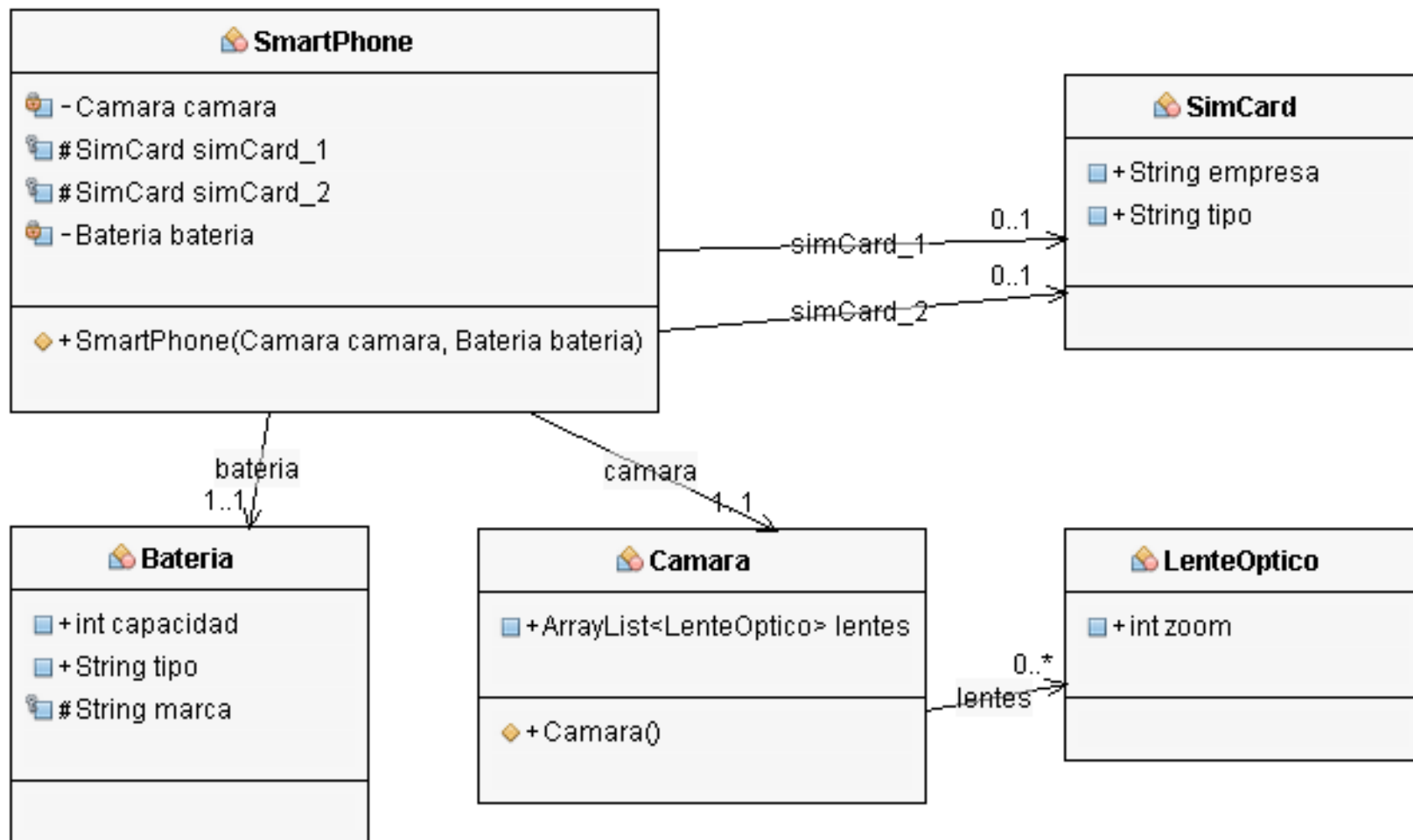
Representar el siguiente código en un Diagrama de Clases

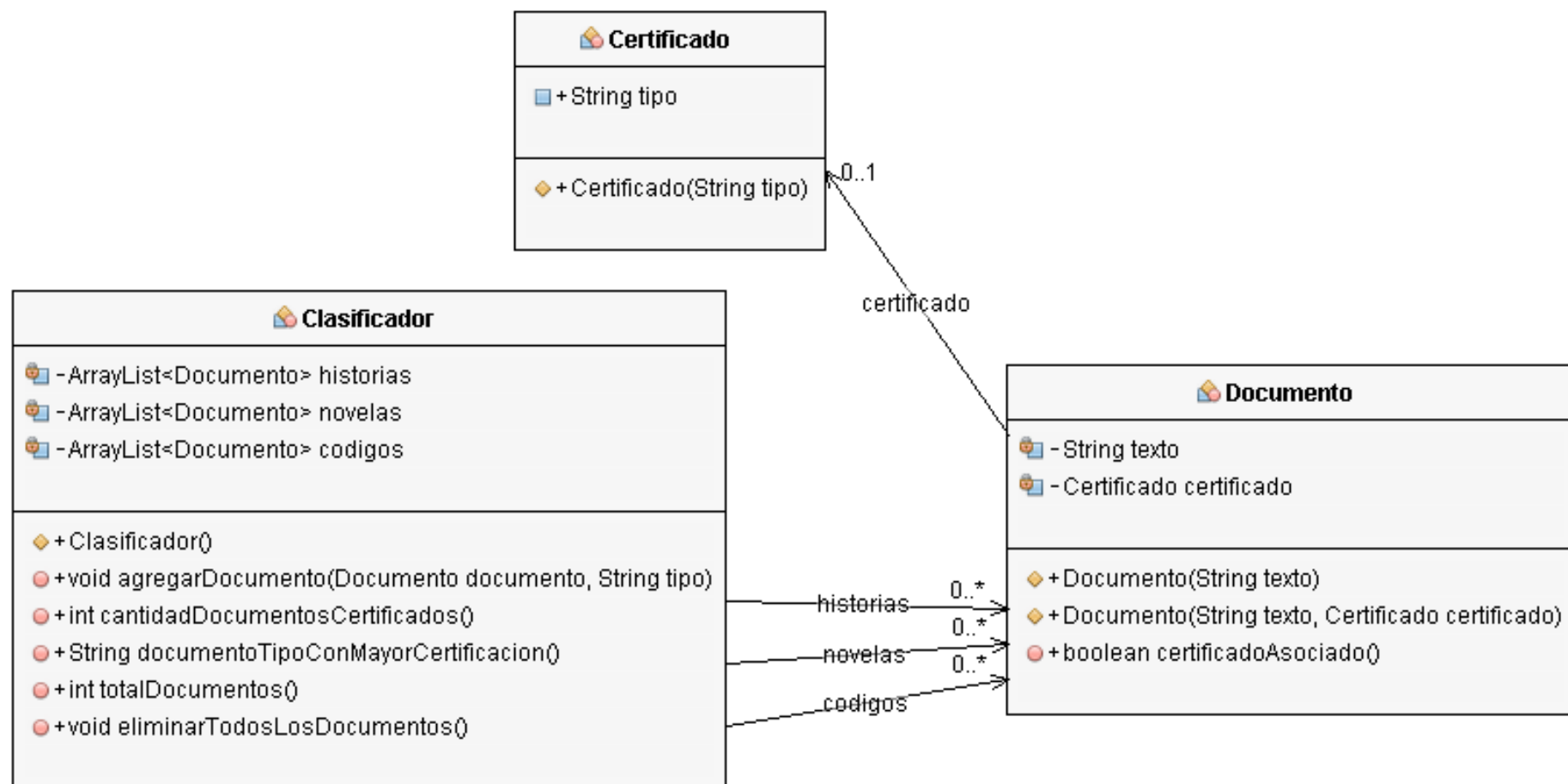
```
public class Bateria {  
  
    public int capacidad;  
    public String tipo;  
    protected String marca;  
  
}  
  
public class LenteOptico {  
  
    public int zoom;  
  
}  
  
public class SimCard {  
  
    public String empresa;  
    public String tipo;  
  
}
```

```
public class SmartPhone {  
  
    private Camara camara;  
    protected SimCard simCard_1;  
    protected SimCard simCard_2;  
    private Bateria bateria;  
  
    public SmartPhone(Camara camara, Bateria bateria) {  
        this.camara = camara;  
        this.bateria = bateria;  
    }  
  
}  
  
public class Camara {  
  
    public ArrayList<LenteOptico> lentes;  
  
    public Camara() {  
        lentes = new ArrayList<>();  
    }  
  
}
```



Solución



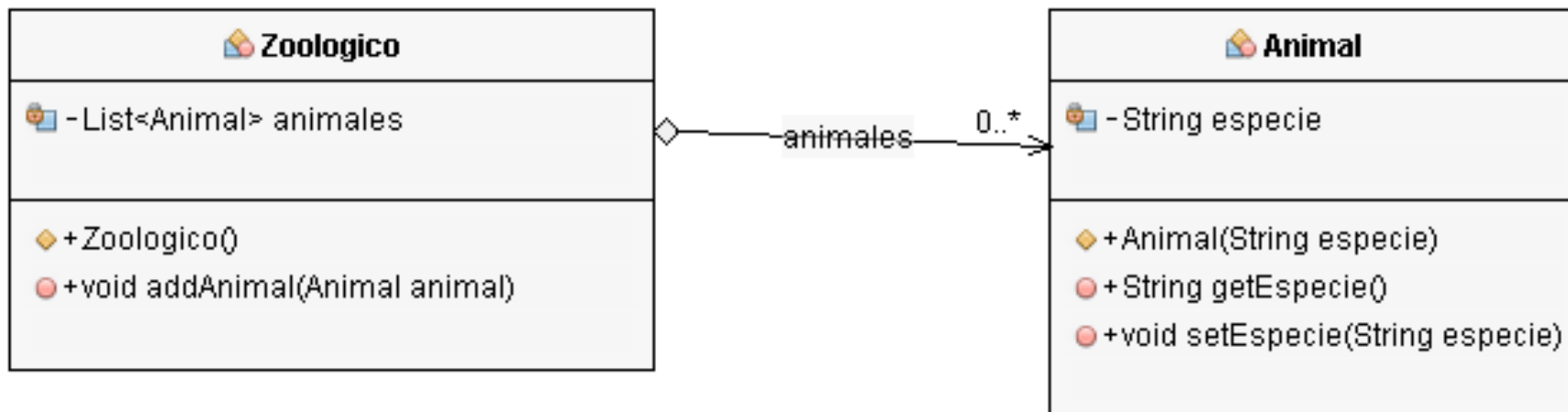


Asociación - Agregación

Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye. Este tipo de relación es comúnmente llamada Agregación (el objeto base utiliza al incluido para su funcionamiento). En el caso de Zoológico y Animal, Zoológico incluye N cantidad de animales que puede manipular. En el caso que Zoológico deje de existir, los animales continuarán “*viviendo*”.

Obs. 1: La agregación se indica a través de un rombo transparente.

Obs. 2: El rombo va en el objeto que posee la o las referencias.



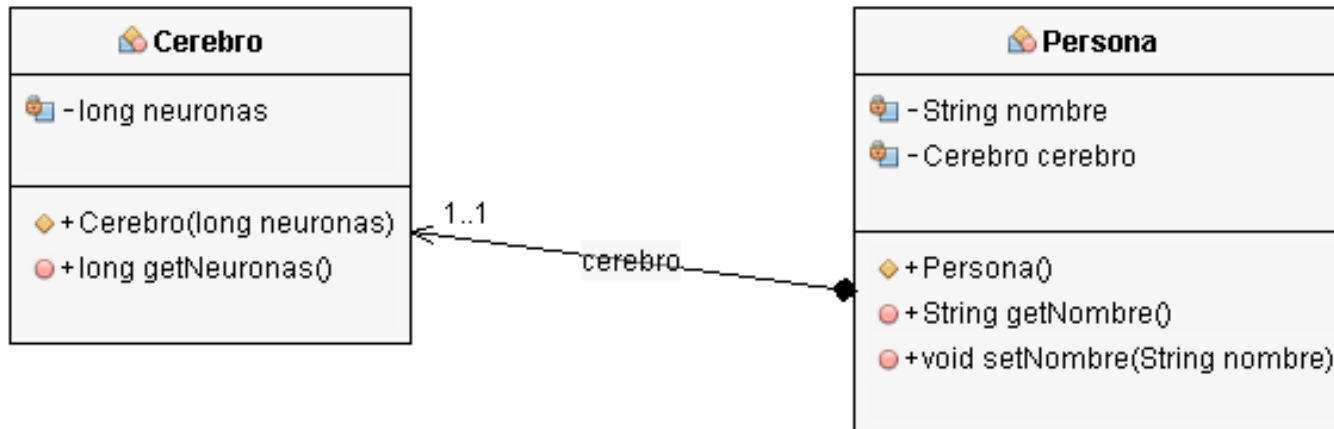
Asociación - Composición

Es un tipo de relación estática, en donde el tiempo de vida del objeto incluido esta condicionado por el tiempo de vida del que lo incluye. Este tipo de relación es comúnmente llamada Composición (el Objeto base se construye a partir del objeto incluido, es decir, es "parte/todo").

En el caso de Persona y Cerebro, Persona tiene uno y solo un cerebro, si Persona deja de existir también lo hará Cerebro.

Obs. 1: La composición se indica a través de un rombo negro.

Obs. 2: El rombo va en el objeto que posee la referencia.



Código de Ejemplo - Composición

```
public class Cerebro {  
  
    private long neuronas;  
  
    public Cerebro(long neuronas) {  
        this.neuronas = neuronas;  
    }  
  
    public long getNeuronas() {  
        return neuronas;  
    }  
}
```

```
public class Persona {  
  
    private String nombre;  
    private final Cerebro cerebro;  
  
    public Persona() {  
        cerebro = new Cerebro(Long.MAX_VALUE);  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
}
```

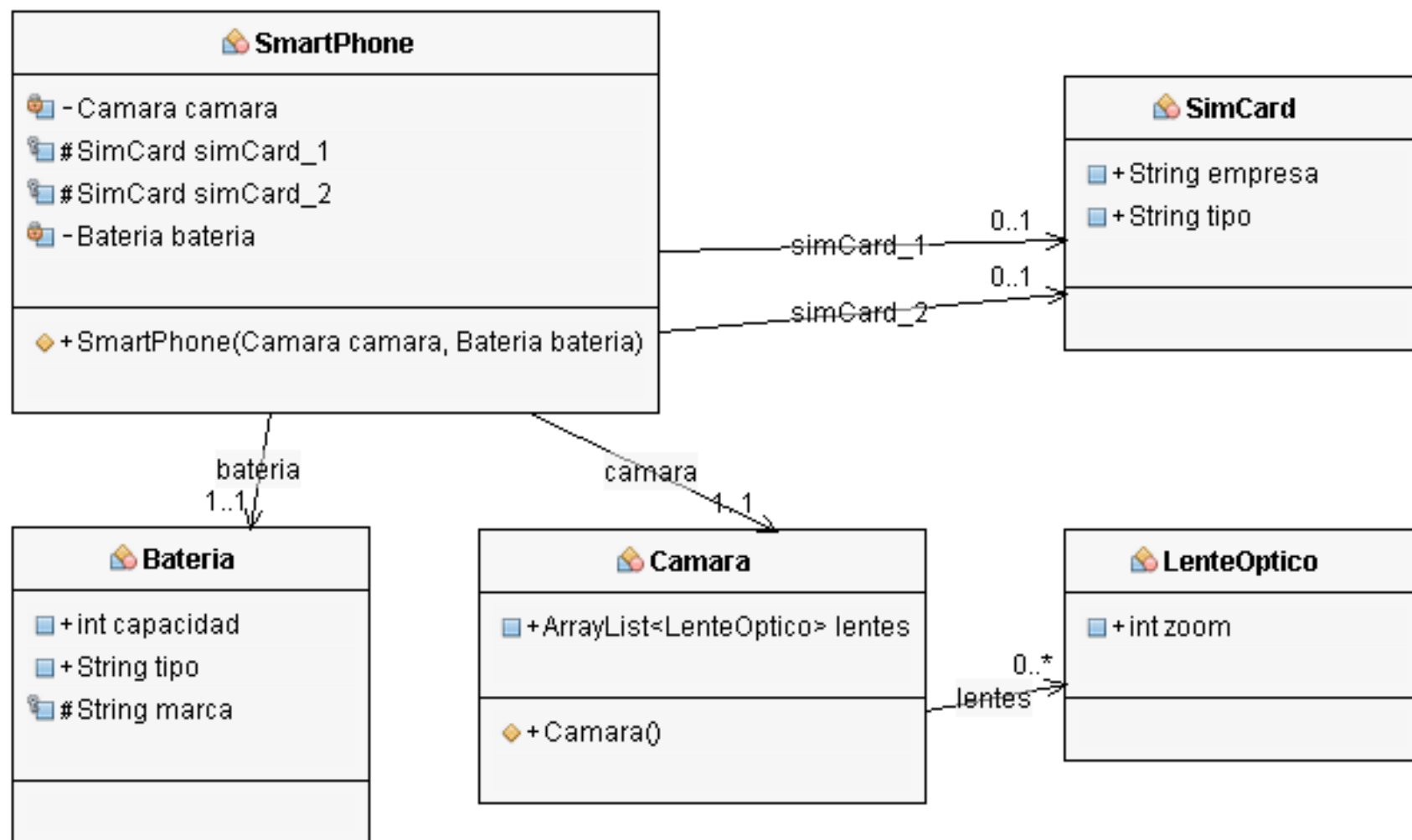
Obs.: Para que Cerebro y Persona formen una composición, la instancia de Cerebro debe venir dentro de la clase Persona. Si se recibe una instancia de cerebro por los parámetros(Argumentos) del constructor o se crea un método como por ejemplo setCerebro(Cerebro cerebro)/getCerebro() se deberá considerar esta asociación como **Agregación**.

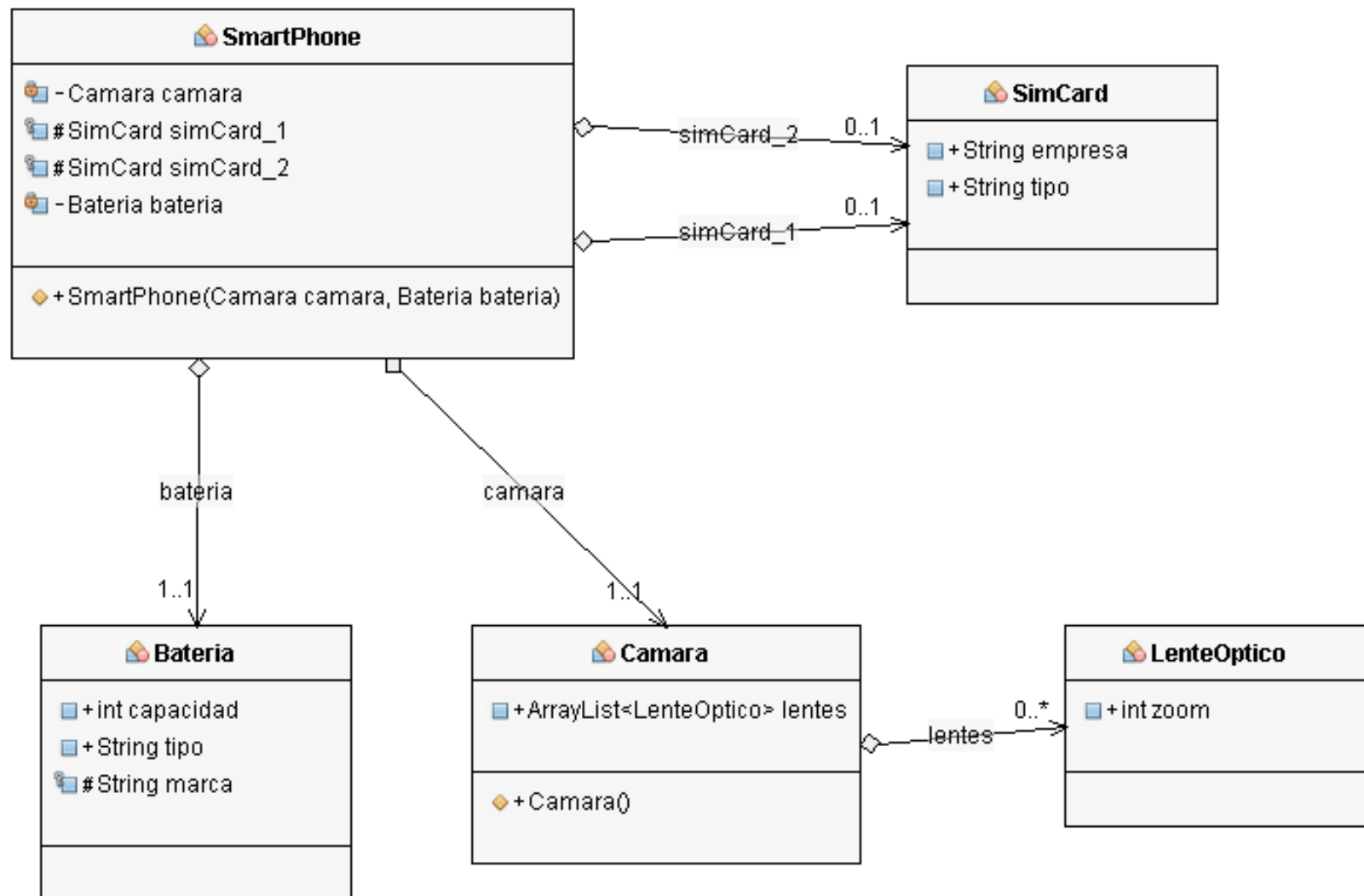


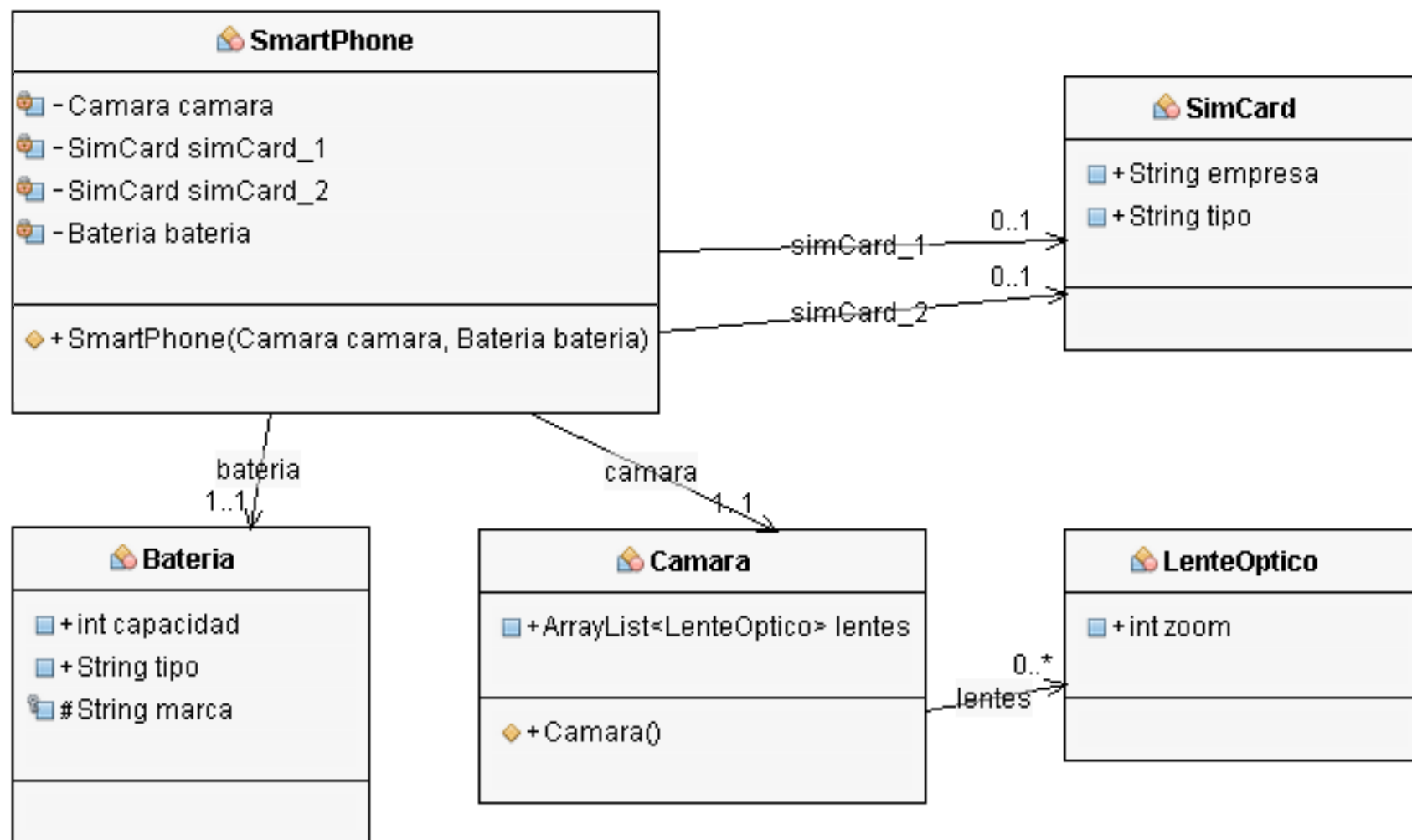
UNIVERSIDAD DEL BÍO-BÍO

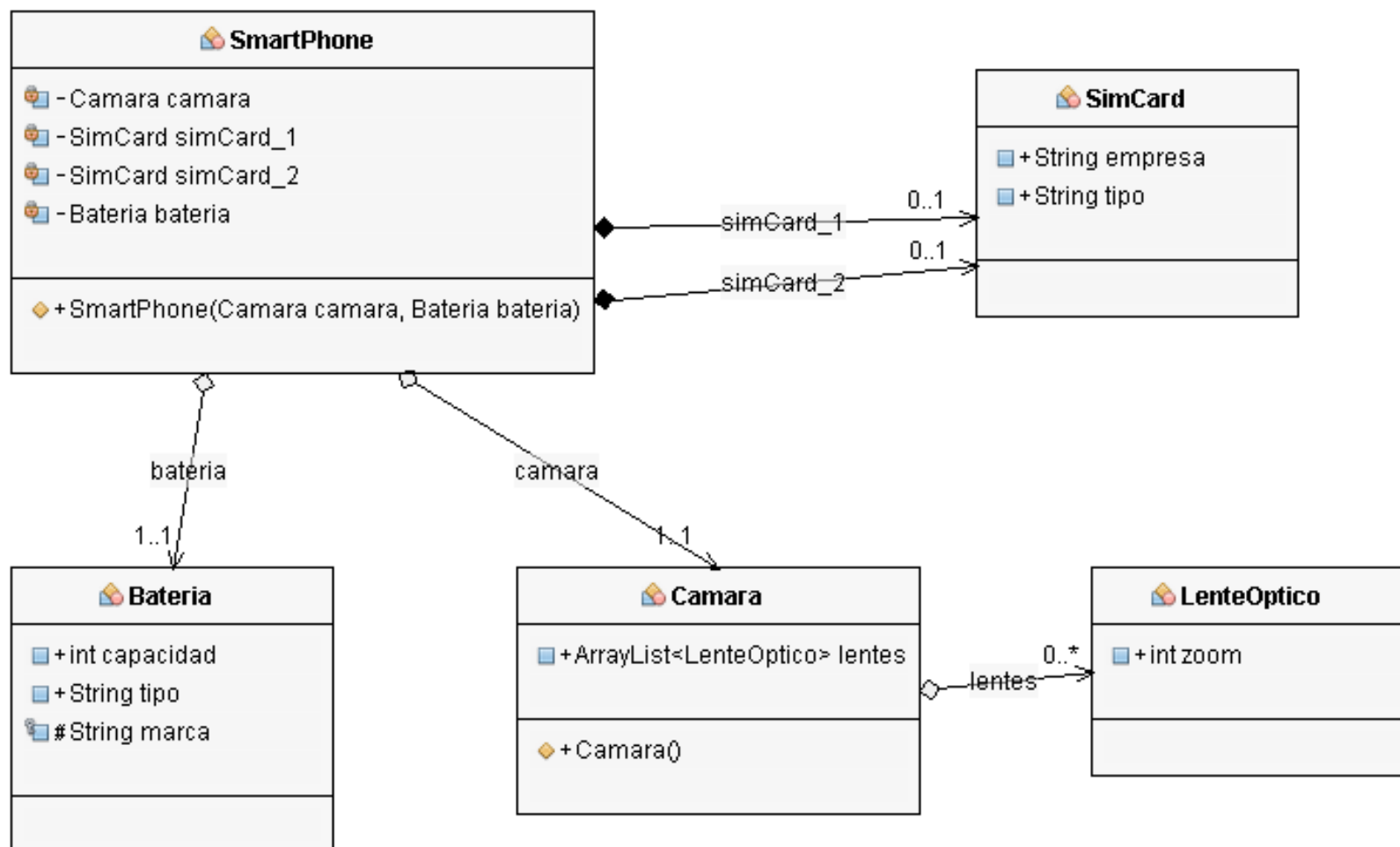
Ejercicios

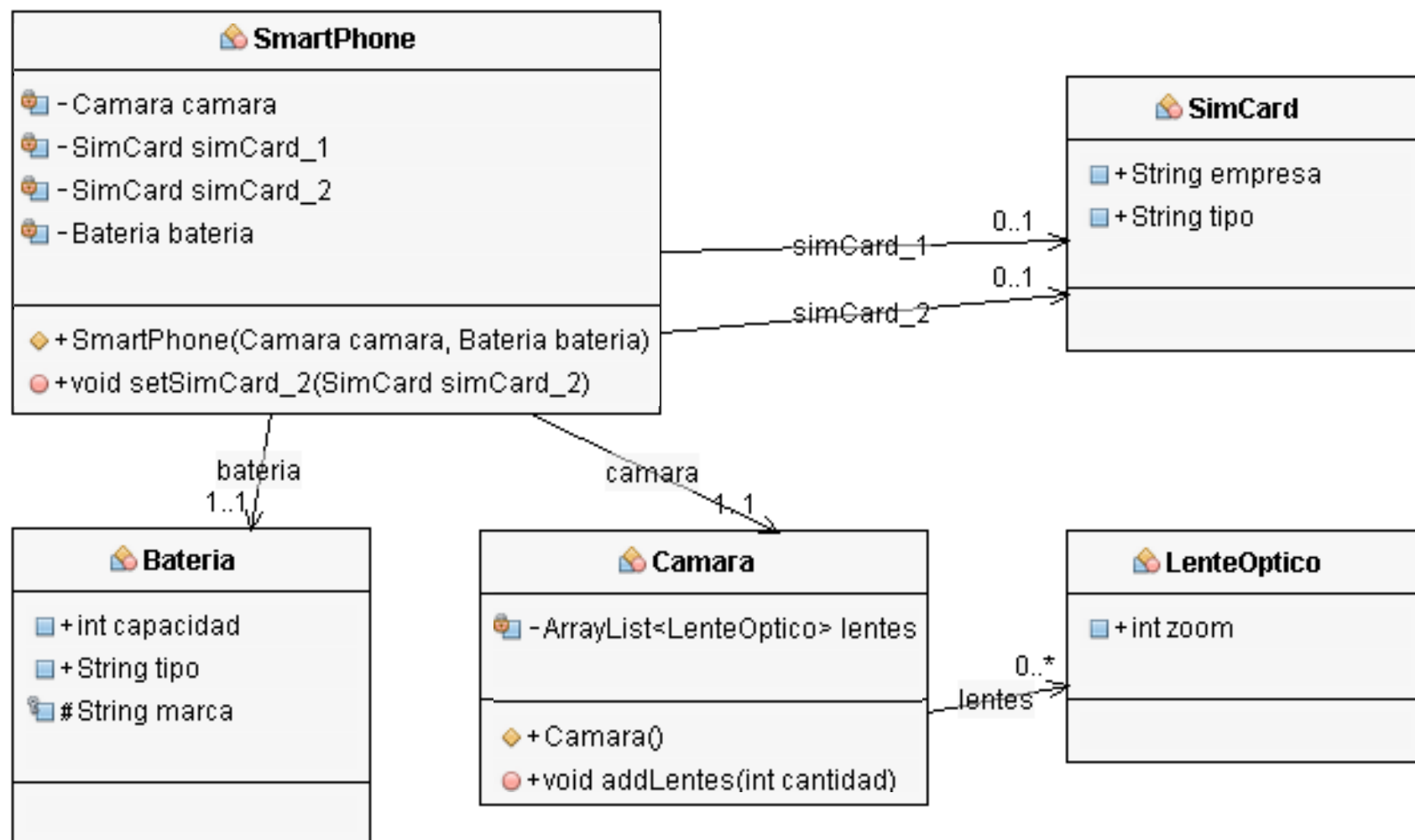
Para cada ejercicio indique en cada clase qué tipo de relación tiene con las demás clases
(Composición/Agregación).

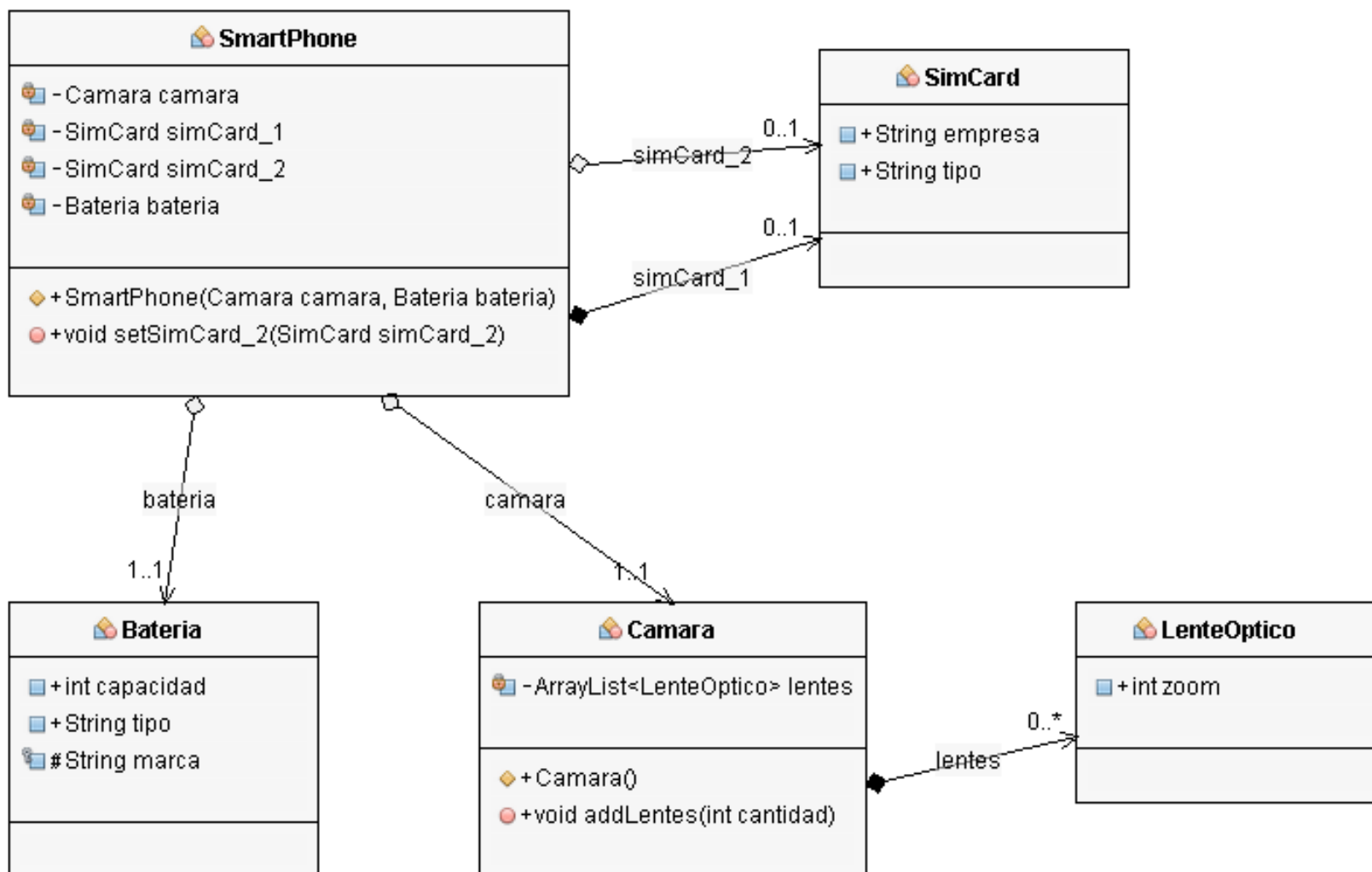


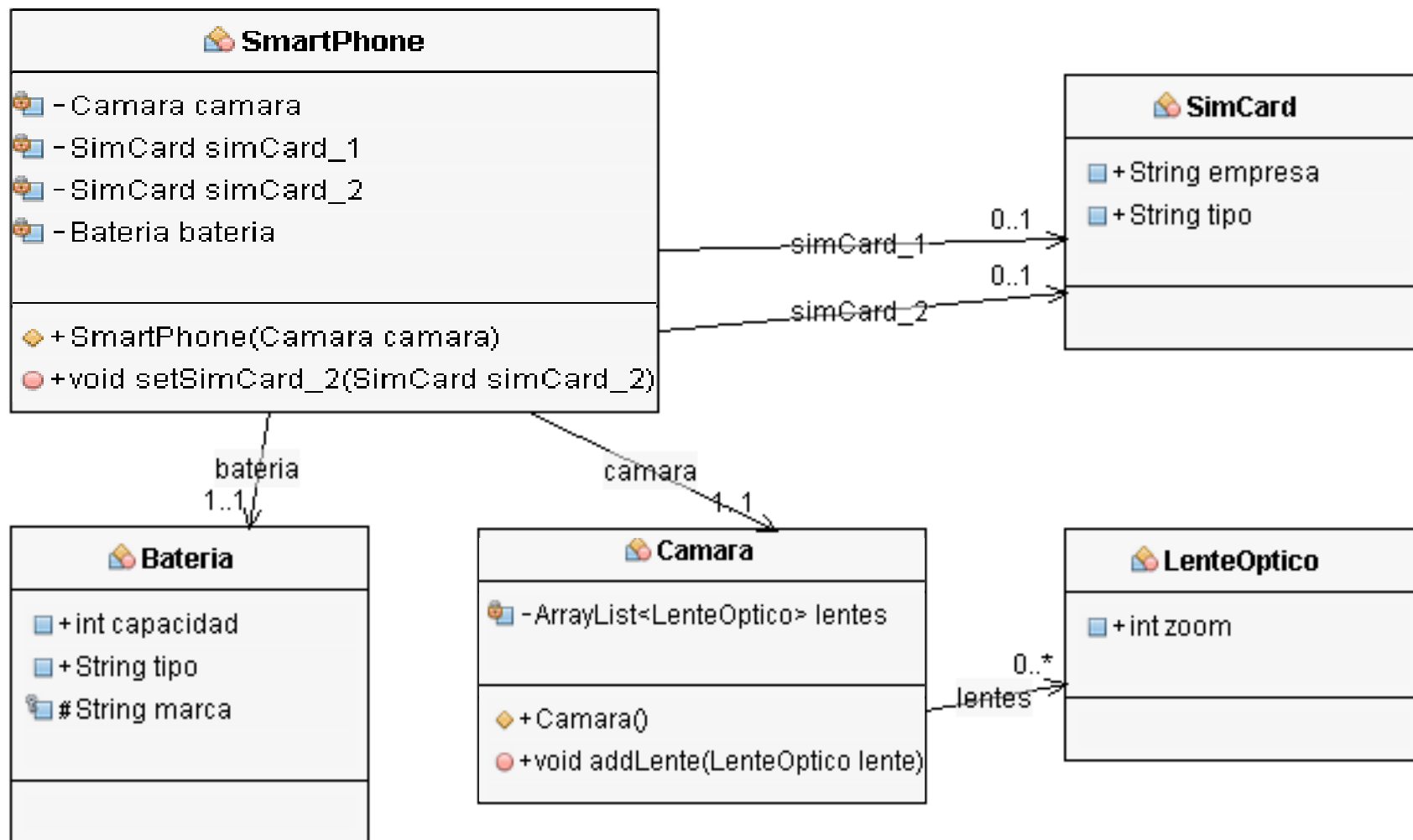


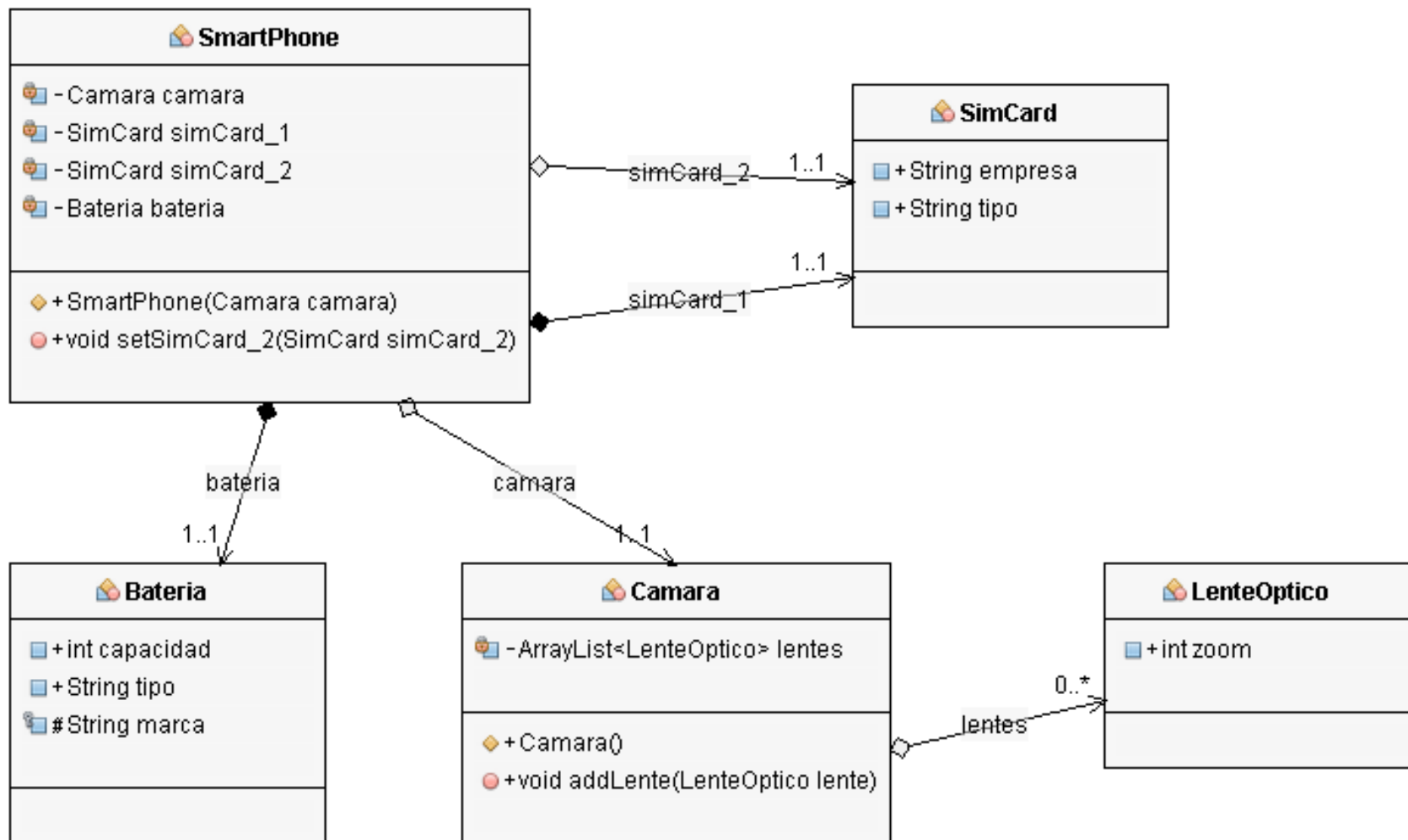


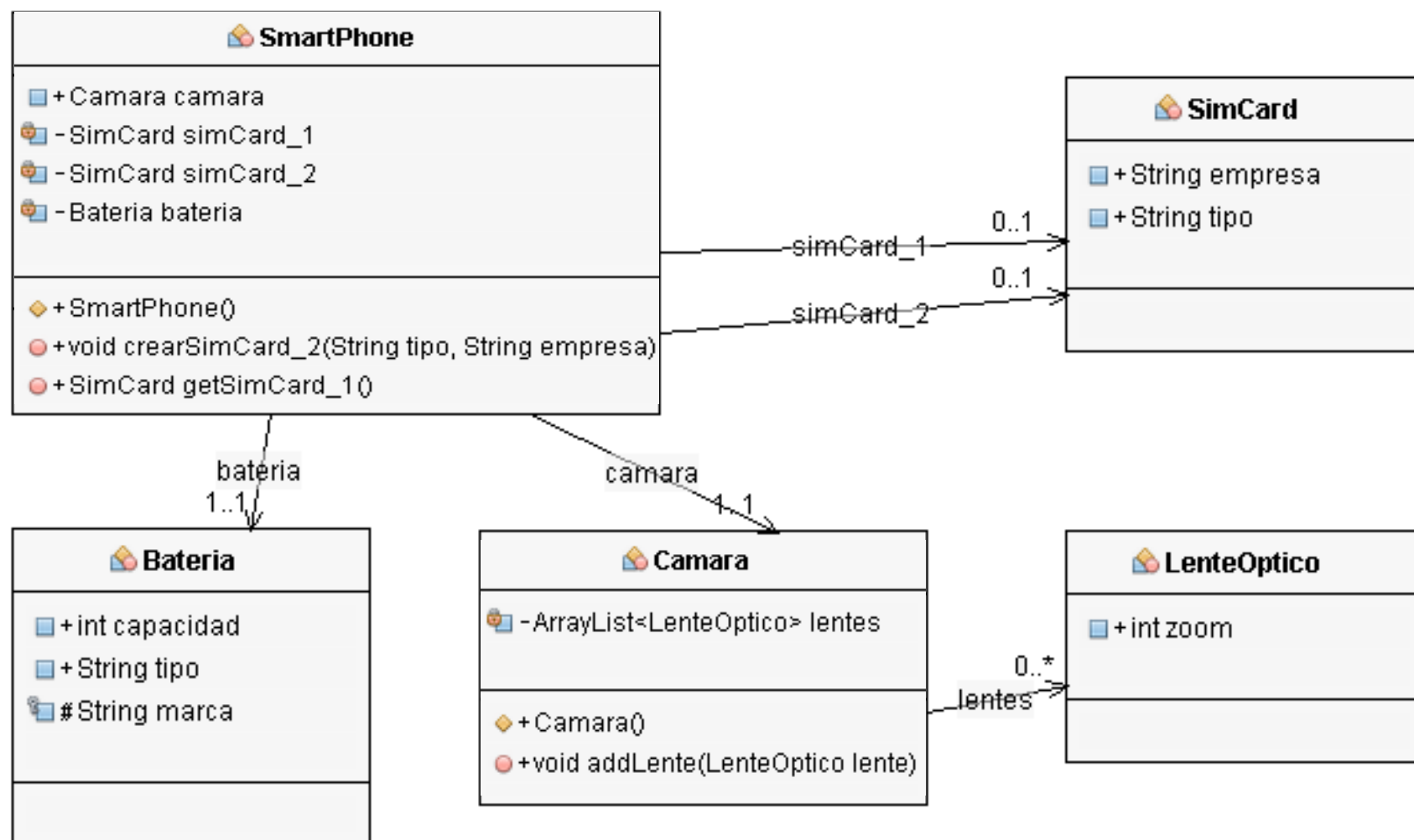


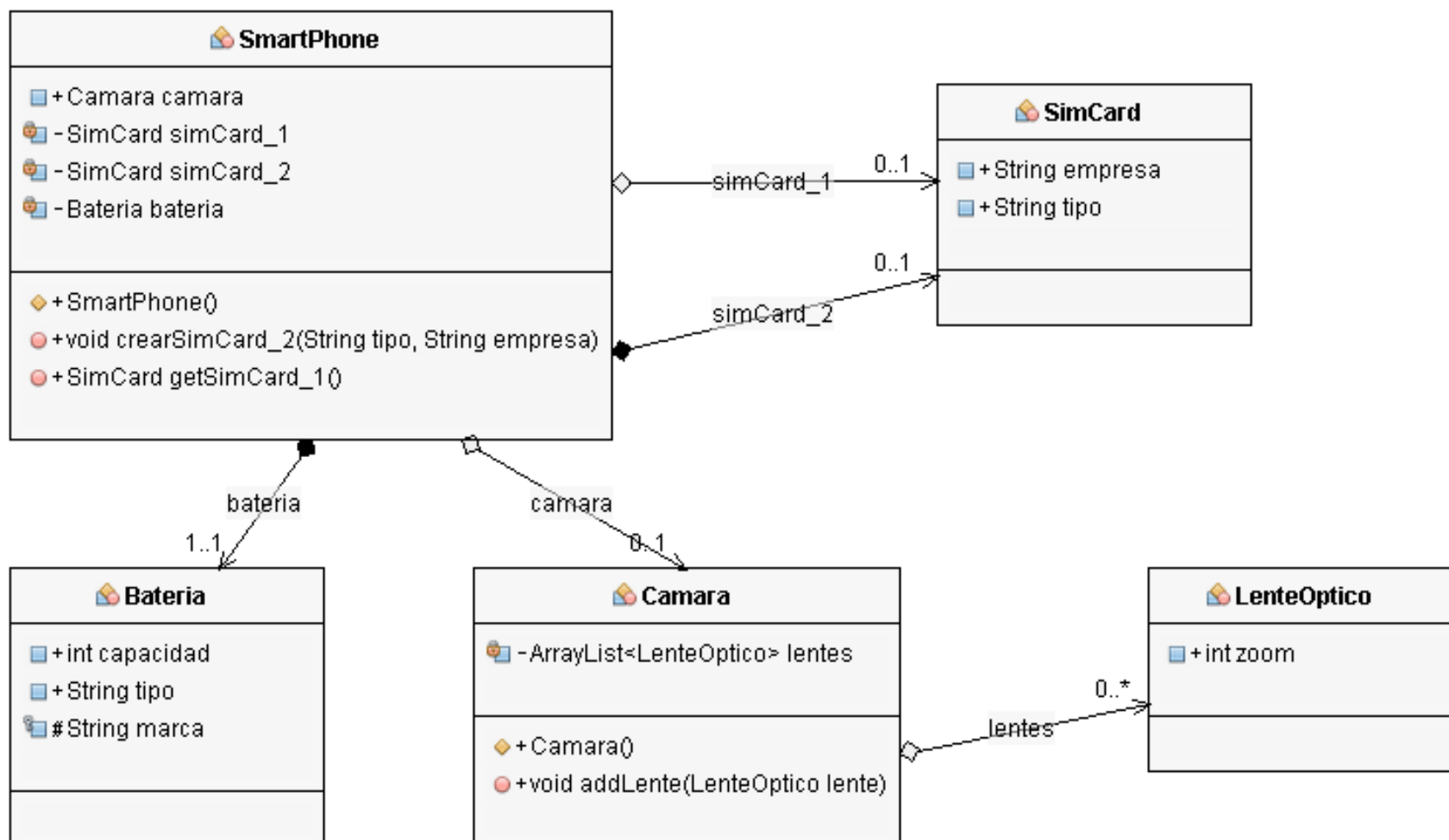












Dependencia

Una notación de dependencia es una notación gráfica que se utiliza en un diagrama de clase UML para representar una relación entre dos clases, en la que una clase (el cliente) depende de la otra clase (el proveedor). Cualquier modificación de la clase del proveedor puede afectar la clase del cliente.

Una notación de dependencia se dibuja como una línea de guión con una flecha de línea apuntando a la clase de proveedor.

Por ejemplo, la clase "Fábrica de certificados" depende de la clase "Certificado", porque se puede crear un certificado a partir de una fábrica de certificados. Cualquier modificación de la clase "Certificado" afectará a la clase "Fábrica de certificados".

