Trabajo Práctico — Relaciones UML

Materia: Programación Orientada a Objetos (Java)

Alumno: Pighin Bruno
Comisión: 8
Fecha: 11/10/2025

Criterios de modelado (resumen)

♦ Composición: parte con ciclo de vida dependiente; se modela con atributo final y construcción en el dueño.

◊ Agregación: referencia no exclusiva; el agregado puede existir por fuera.

↔ Asociación bidireccional 1:1: referencias cruzadas y setters sincronizados.

→ Asociación unidireccional 1→1: solo un lado conoce al otro.

Dependencia de uso: aparece como parámetro local; no se almacena.

Dependencia de creación: el método crea la instancia y no la conserva.

1) Pasaporte – Foto – Titular

Composición: Pasaporte → Foto. Asociación bidireccional: Pasaporte ↔ Titular.

```
class Foto {
    private final byte[] imagen;
    private final String formato;
    public Foto(byte[] imagen, String formato) { this.imagen = imagen;
this.formato = formato; }
    public byte[] getImagen() { return imagen; }
    public String getFormato() { return formato; }
}

class Titular {
    private String nombre; private String dni;
    private Pasaporte pasaporte; // ⇔
    public Titular(String nombre, String dni) { this.nombre = nombre; this.dni
= dni; }
    void setPasaporte(Pasaporte p) { this.pasaporte = p; if (p != null &&
p.getTitular()!=this) p.setTitular(this); }
    public Pasaporte getPasaporte() { return pasaporte; }
}

class Pasaporte {
    private final String numero; private final String fechaEmision;
    private final Foto foto;             // composición (parte-vida)
    private Titular titular;             // ⇔
    public Pasaporte(String numero, String fechaEmision, Foto foto) {
        this.numero = numero; this.fechaEmision = fechaEmision; this.foto =
foto;
    }
    public void setTitular(Titular t) { this.titular = t; if (t != null &&
t.getPasaporte()!=this) t.setPasaporte(this); }
    public Titular getTitular() { return titular; }
    public Foto getFoto() { return foto; }
}
```

2) Celular – Batería – Usuario

Agregación: Celular → Batería. Asociación bidireccional: Celular ↔ Usuario.

```
class Bateria { private String modelo; private int capacidad; /* mAh */ }
class Usuario { private String nombre, dni; private Celular celular;
    void setCelular(Celular c){ this.celular=c; if(c!=null &&
c.getUsuario()!=this) c.setUsuario(this); } }
class Celular {
    private String imei, marca, modelo;
    private Bateria bateria;           // agregación (puede existir aparte)
    private Usuario usuario;           // ⇔
    public void setBateria(Bateria b){ this.bateria=b; }
    public void setUsuario(Usuario u){ this.usuario=u; if(u!=null &&
u.celular!=this) u.setCelular(this); }
    public Usuario getUsuario(){ return usuario; }
}
```

3) Libro – Autor – Editorial

Asociación unidireccional: Libro → Autor. Agregación: Libro → Editorial.

```
class Autor { private String nombre, nacionalidad; }
class Editorial { private String nombre, direccion; }
class Libro {
    private String titulo, isbn;
    private Autor autor;           // →
    private Editorial editorial;   // agregación
}
```

4) TarjetaDeCrédito – Cliente – Banco

Asociación bidireccional: Tarjeta ↔ Cliente. Agregación: Tarjeta → Banco.

```
class Banco { private String nombre, cuit; }

class Cliente {
    private String nombre, dni; private TarjetaDeCredito tarjeta;
    void setTarjeta(TarjetaDeCredito t){ this.tarjeta=t; if(t!=null &&
t.getCliente()!=this) t.setCliente(this); }
}

class TarjetaDeCredito {
    private String numero, fechaVencimiento;
    private Cliente cliente;   // ↔
    private Banco banco;       // agregación
    void setCliente(Cliente c){ this.cliente=c; if(c!=null && c.tarjeta!=this)
c.setTarjeta(this); }
    void setBanco(Banco b){ this.banco=b; }
    Cliente getCliente(){ return cliente; }
}
```

5) Computadora – PlacaMadre – Propietario

Composición: Computadora → PlacaMadre. Asociación bidireccional: Computadora ↔ Propietario.

```
class PlacaMadre { private String modelo, chipset; }
class Propietario { private String nombre, dni; private Computadora pc;
    void setPc(Computadora c){ this.pc=c; if(c!=null &&
c.getPropietario()!=this) c.setPropietario(this); } }
class Computadora {
    private String marca, numeroSerie;
    private final PlacaMadre placa;   // composición
    private Propietario propietario;  // ⇔
    public Computadora(String marca, String numeroSerie, PlacaMadre placa){
this.marca=marca; this.numeroSerie=numeroSerie; this.placa=placa; }
    public void setPropietario(Propietario p){ this.propietario=p; if(p!=null
&& p.pc!=this) p.setPc(this); }
    public Propietario getPropietario(){ return propietario; }
}
```

6) Reserva – Cliente – Mesa

Asociación unidireccional: Reserva → Cliente. Agregación: Reserva → Mesa.

```
class ClienteR { private String nombre, telefono; }
class Mesa { private int numero, capacidad; }
class Reserva {
    private String fecha, hora;
    private ClienteR cliente;  // →
    private Mesa mesa;         // agregación
}
```

7) Vehículo – Motor – Conductor

Agregación: Vehículo → Motor. Asociación bidireccional: Vehículo ↔ Conductor.

```
class Motor { private String tipo, numeroSerie; }
class Conductor { private String nombre, licencia; private Vehiculo vehiculo;
    void setVehiculo(Vehiculo v){ this.vehiculo=v; if(v!=null &&
v.getConductor()!=this) v.setConductor(this); } }
class Vehiculo {
    private String patente, modelo;
    private Motor motor;            // agregación
    private Conductor conductor;    // ↔
    void setMotor(Motor m){ this.motor=m; }
    void setConductor(Conductor c){ this.conductor=c; if(c!=null &&
c.vehiculo!=this) c.setVehiculo(this); }
    Conductor getConductor(){ return conductor; }
}
```

8) Documento – FirmaDigital – Usuario

Composición: Documento → FirmaDigital. Agregación: FirmaDigital → Usuario.

```
class UsuarioFD { private String nombre, email; }
class FirmaDigital {
    private String codigoHash, fecha;
    private UsuarioFD usuario;  // agregación
    public FirmaDigital(String codigoHash, String fecha){
this.codigoHash=codigoHash; this.fecha=fecha; }
    void setUsuario(UsuarioFD u){ this.usuario=u; }
}
class Documento {
    private String titulo, contenido;
    private final FirmaDigital firma; // composición
    public Documento(String titulo, String contenido, FirmaDigital firma){
this.titulo=titulo; this.contenido=contenido; this.firma=firma; }
}
```

9) CitaMédica – Paciente – Profesional

Asociaciones unidireccionales: Cita → Paciente y Cita → Profesional.

```
class Paciente { private String nombre, obraSocial; }
class Profesional { private String nombre, especialidad; }
class CitaMedica {
    private String fecha, hora;
    private Paciente paciente;          // →
    private Profesional profesional;    // →
}
```

10) CuentaBancaria – ClaveSeguridad – Titular

Composición: Cuenta → ClaveSeguridad. Asociación bidireccional: Cuenta ↔ Titular.

```
class ClaveSeguridad { private String codigo; private String
ultimaModificacion; }
class TitularCB { private String nombre, dni; private CuentaBancaria cuenta;
    void setCuenta(CuentaBancaria c){ this.cuenta=c; if(c!=null &&
c.getTitular()!=this) c.setTitular(this); } }
class CuentaBancaria {
    private String cbu; private double saldo;
    private final ClaveSeguridad clave; // composición
    private TitularCB titular;          // ↔
    public CuentaBancaria(String cbu, double saldo, ClaveSeguridad clave){
this.cbu=cbu; this.saldo=saldo; this.clave=clave; }
    public void setTitular(TitularCB t){ this.titular=t; if(t!=null &&
t.cuenta!=this) t.setCuenta(this); }
    public TitularCB getTitular(){ return titular; }
}
```

11) Reproductor – Canción – Artista (Dependencia de uso)

Asociación unidireccional: Canción → Artista. Uso: Reproductor.reproducir(Cancion). No se almacena.

```
class Artista { private String nombre, genero; }
class Cancion { private String titulo; private Artista artista; /* → */ }
class Reproductor {
    public void reproducir(Cancion cancion) {
        // usa Cancion, no la guarda
        System.out.println("Reproduciendo: " + /*cancion.getTitulo()*/ "...");
    }
}
```

12) Impuesto – Contribuyente – Calculadora (Dependencia de uso)

Asociación unidireccional: Impuesto → Contribuyente. Uso: Calculadora.calcular(Impuesto). No se almacena.

```
class Contribuyente { private String nombre, cuil; }
class Impuesto { private double monto; private Contribuyente contribuyente; /*
→ */ }
class Calculadora {
    public void calcular(Impuesto impuesto) {
        // usa Impuesto, no lo guarda
        double total = impuesto /*.getMonto()*/ * 1.0;
        System.out.println("Total: " + total);
    }
}
```

13) GeneradorQR – Usuario – CodigoQR (Dependencia de creación)

Asociación unidireccional: CodigoQR → Usuario. Creación: GeneradorQR.generar(String, Usuario). No se conserva.

```
class CodigoQR {
    private String valor; private Usuario usuario; // →
    public CodigoQR(String valor, Usuario usuario){ this.valor=valor;
this.usuario=usuario; }
}
class GeneradorQR {
    public void generar(String valor, Usuario usuario) {
        CodigoQR qr = new CodigoQR(valor, usuario); // se crea y se usa
localmente
        System.out.println("QR generado");
    }
}
```

14) EditorVideo – Proyecto – Render (Dependencia de creación)

Asociación unidireccional: Render → Proyecto. Creación: EditorVideo.exportar(String, Proyecto). No se conserva.

```
class Proyecto { private String nombre; private int duracionMin; }
class Render {
    private String formato; private Proyecto proyecto; // →
    public Render(String formato, Proyecto proyecto){ this.formato=formato;
this.proyecto=proyecto; }
}
class EditorVideo {
    public void exportar(String formato, Proyecto proyecto) {
        Render r = new Render(formato, proyecto); // creación dentro del
método
        System.out.println("Export listo");
    }
}
```